The 30 Year Horizon

Manuel Bronstein  William Burge  Timothy Daly
James Davenport  Michael Dewar  Martin Dunstan
Albrecht Fortenbacher  Patrizia Gianni  Johannes Grabmeier
Jocelyn Guidry  Richard Jenks  Larry Lambe
Michael Monagan  Scott Morrison  William Sit
Jonathan Steinbach  Robert Sutor  Barry Trager
Stephen Watt  Jim Wen  Clifton Williamson

Volume 11: Axiom Browser

July 4, 2018

654cda014164acd5f03fa6d197a17857c5565607
Portions Copyright (c) 2005 Timothy Daly

The Blue Bayou image Copyright (c) 2004 Jocelyn Guidry

Portions Copyright (c) 2004 Martin Dunstan
Portions Copyright (c) 2007 Alfredo Portes
Portions Copyright (c) 2007 Arthur Ralfs
Portions Copyright (c) 2005 Timothy Daly

Portions Copyright (c) 1991-2002,
The Numerical Algorithms Group Ltd.
All rights reserved.

This book and the Axiom software is licensed as follows:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of The Numerical Algorithms Group Ltd. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Inclusion of names in the list of credits is based on historical information and is as accurate as possible. Inclusion of names does not in any way imply an endorsement but represents historical influence on Axiom development.
# Contents

1 Overview ........................................ 1
   Build Instructions ................................ 1
   The Makefile .................................... 1
   Building new pages ................................ 2
      Communicating with Axiom ....................... 3
      Handling statements with no free variables ... 3
      Handling statements with free variables ........ 3
      Handling domain database lookups .............. 4
      Handling )show domain .......................... 4
      Handling lisp expressions ...................... 4
      Handling expressions that have no output ...... 4
   Defined Pages .................................... 4
   The Standard Layout .............................. 15
   Cascading Style Sheet ............................ 16
      Standard Style Sheet ............................ 16
      Menu style sheet ................................ 18
   standard head ................................... 21
   Javascript functions .............................. 22
      Show only mathml ............................... 22
      Show Full Answer ................................ 23
      Handle Free Variables ......................... 24
      axiom talker ................................... 25
   Pages ........................................... 26
      axiomfonts.xhtml ................................ 38
      algebrapage.xhtml ................................ 82
<table>
<thead>
<tr>
<th>File Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>alggrouptheory.xhtml</td>
<td>82</td>
</tr>
<tr>
<td>alggrouptheorygroup.xhtml</td>
<td>83</td>
</tr>
<tr>
<td>alggrouptheoryrepa6.xhtml</td>
<td>84</td>
</tr>
<tr>
<td>alggrouptheoryreptheory.xhtml</td>
<td>87</td>
</tr>
<tr>
<td>algnumbertheory.xhtml</td>
<td>88</td>
</tr>
<tr>
<td>algnumbertheorygalois.xhtml</td>
<td>88</td>
</tr>
<tr>
<td>basiccommand.xhtml</td>
<td>95</td>
</tr>
<tr>
<td>basiclimit.xhtml</td>
<td>96</td>
</tr>
<tr>
<td>bcexpand.xhtml</td>
<td>96</td>
</tr>
<tr>
<td>bcmatrix.xhtml</td>
<td>98</td>
</tr>
<tr>
<td>calculus.xhtml</td>
<td>102</td>
</tr>
<tr>
<td>calculuspage.xhtml</td>
<td>103</td>
</tr>
<tr>
<td>calderivatives.xhtml</td>
<td>104</td>
</tr>
<tr>
<td>calintegrals.xhtml</td>
<td>107</td>
</tr>
<tr>
<td>callaplace.xhtml</td>
<td>110</td>
</tr>
<tr>
<td>callimits.xhtml</td>
<td>111</td>
</tr>
<tr>
<td>calmoreintegrals.xhtml</td>
<td>114</td>
</tr>
<tr>
<td>calseries.xhtml</td>
<td>118</td>
</tr>
<tr>
<td>calseries1.xhtml</td>
<td>119</td>
</tr>
<tr>
<td>calseries2.xhtml</td>
<td>121</td>
</tr>
<tr>
<td>calseries3.xhtml</td>
<td>123</td>
</tr>
<tr>
<td>calseries4.xhtml</td>
<td>124</td>
</tr>
<tr>
<td>calseries5.xhtml</td>
<td>127</td>
</tr>
<tr>
<td>calseries6.xhtml</td>
<td>129</td>
</tr>
<tr>
<td>calseries7.xhtml</td>
<td>132</td>
</tr>
<tr>
<td>calseries8.xhtml</td>
<td>133</td>
</tr>
<tr>
<td>cats.xhtml</td>
<td>136</td>
</tr>
<tr>
<td>commandline.xhtml</td>
<td>137</td>
</tr>
<tr>
<td>complexlimit.xhtml</td>
<td>151</td>
</tr>
<tr>
<td>conversionfunctions.xhtml</td>
<td>152</td>
</tr>
<tr>
<td>crytopage.xhtml</td>
<td>155</td>
</tr>
<tr>
<td>crytoclass1.xhtml</td>
<td>156</td>
</tr>
<tr>
<td>crytoclass2.xhtml</td>
<td>160</td>
</tr>
<tr>
<td>CONTENTS</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>---</td>
</tr>
<tr>
<td>crytoclass3.xhtml</td>
<td>163</td>
</tr>
<tr>
<td>crytoclass4.xhtml</td>
<td>167</td>
</tr>
<tr>
<td>crytoclass5.xhtml</td>
<td>170</td>
</tr>
<tr>
<td>crytoclass6.xhtml</td>
<td>173</td>
</tr>
<tr>
<td>crytoclass7.xhtml</td>
<td>175</td>
</tr>
<tr>
<td>crytoclass8.xhtml</td>
<td>178</td>
</tr>
<tr>
<td>crytoclass9.xhtml</td>
<td>182</td>
</tr>
<tr>
<td>crytoclass10.xhtml</td>
<td>185</td>
</tr>
<tr>
<td>crytoclass11.xhtml</td>
<td>187</td>
</tr>
<tr>
<td>dbopbinary.xhtml</td>
<td>190</td>
</tr>
<tr>
<td>dbcharacteristic.xhtml</td>
<td>190</td>
</tr>
<tr>
<td>dbcomplexcomplex.xhtml</td>
<td>190</td>
</tr>
<tr>
<td>dbcomplexconjugate.xhtml</td>
<td>191</td>
</tr>
<tr>
<td>dbcomplexfactor.xhtml</td>
<td>191</td>
</tr>
<tr>
<td>dbcomplexdoublefloat.xhtml</td>
<td>191</td>
</tr>
<tr>
<td>dbcomplexfloat.xhtml</td>
<td>192</td>
</tr>
<tr>
<td>dbcompleximag.xhtml</td>
<td>192</td>
</tr>
<tr>
<td>dbcomplexnorm.xhtml</td>
<td>192</td>
</tr>
<tr>
<td>dbcomplexreal.xhtml</td>
<td>192</td>
</tr>
<tr>
<td>dbcomplexinteger.xhtml</td>
<td>193</td>
</tr>
<tr>
<td>dbexpressioninteger.xhtml</td>
<td>193</td>
</tr>
<tr>
<td>dbfractioninteger.xhtml</td>
<td>193</td>
</tr>
<tr>
<td>dbfractionpolynomialinteger.xhtml</td>
<td>193</td>
</tr>
<tr>
<td>dblookup.xhtml</td>
<td>194</td>
</tr>
<tr>
<td>dbopacos.xhtml</td>
<td>194</td>
</tr>
<tr>
<td>dbopacosh.xhtml</td>
<td>194</td>
</tr>
<tr>
<td>dbopacot.xhtml</td>
<td>194</td>
</tr>
<tr>
<td>dbopacoth.xhtml</td>
<td>195</td>
</tr>
<tr>
<td>dbopasc.c.xhtml</td>
<td>195</td>
</tr>
<tr>
<td>dbopasc.h.xhtml</td>
<td>195</td>
</tr>
<tr>
<td>dbopaddmod.xhtml</td>
<td>195</td>
</tr>
<tr>
<td>dbopairyai.xhtml</td>
<td>196</td>
</tr>
<tr>
<td>dbopairybi.xhtml</td>
<td>196</td>
</tr>
<tr>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>dbopapproximants.xhtml</td>
<td>196</td>
</tr>
<tr>
<td>dbopasin.xhtml</td>
<td>196</td>
</tr>
<tr>
<td>dbopasinh.xhtml</td>
<td>197</td>
</tr>
<tr>
<td>dbopasec.xhtml</td>
<td>197</td>
</tr>
<tr>
<td>dbopasech.xhtml</td>
<td>197</td>
</tr>
<tr>
<td>dbopatan.xhtml</td>
<td>197</td>
</tr>
<tr>
<td>dbopatanh.xhtml</td>
<td>198</td>
</tr>
<tr>
<td>dbopbernoullib.xhtml</td>
<td>198</td>
</tr>
<tr>
<td>dbopbesseli.xhtml</td>
<td>198</td>
</tr>
<tr>
<td>dbopbesselj.xhtml</td>
<td>198</td>
</tr>
<tr>
<td>dbopbesselk.xhtml</td>
<td>199</td>
</tr>
<tr>
<td>dbopbessely.xhtml</td>
<td>199</td>
</tr>
<tr>
<td>dbopbeta.xhtml</td>
<td>199</td>
</tr>
<tr>
<td>dbopcardinalnumber.xhtml</td>
<td>199</td>
</tr>
<tr>
<td>dbopchebyshevt.xhtml</td>
<td>200</td>
</tr>
<tr>
<td>dbopchebyshevu.xhtml</td>
<td>200</td>
</tr>
<tr>
<td>dbopcoefficient.xhtml</td>
<td>200</td>
</tr>
<tr>
<td>dbopcoefficients.xhtml</td>
<td>200</td>
</tr>
<tr>
<td>dbopcoerce.xhtml</td>
<td>201</td>
</tr>
<tr>
<td>dbopcolumn.xhtml</td>
<td>201</td>
</tr>
<tr>
<td>dbopcompactfraction.xhtml</td>
<td>201</td>
</tr>
<tr>
<td>dbopcomplexeigenvectors.xhtml</td>
<td>201</td>
</tr>
<tr>
<td>dbopcomplexelementary.xhtml</td>
<td>202</td>
</tr>
<tr>
<td>dbopcomplexintegrate.xhtml</td>
<td>202</td>
</tr>
<tr>
<td>dbopcomplexlimit.xhtml</td>
<td>202</td>
</tr>
<tr>
<td>dbopcomplexsolve.xhtml</td>
<td>202</td>
</tr>
<tr>
<td>dbopcontent.xhtml</td>
<td>203</td>
</tr>
<tr>
<td>dbopcontinuedfraction.xhtml</td>
<td>203</td>
</tr>
<tr>
<td>dbopconvergents.xhtml</td>
<td>203</td>
</tr>
<tr>
<td>dbopconvert.xhtml</td>
<td>203</td>
</tr>
<tr>
<td>dbopcopy.xhtml</td>
<td>204</td>
</tr>
<tr>
<td>dbopcos.xhtml</td>
<td>204</td>
</tr>
<tr>
<td>dbopcosh.xhtml</td>
<td>204</td>
</tr>
<tr>
<td>CONTENTS</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>---</td>
</tr>
<tr>
<td>dlmfinequalities.xhtml</td>
<td>388</td>
</tr>
<tr>
<td>dlmfinfiniteproducts.xhtml</td>
<td>401</td>
</tr>
<tr>
<td>dlmfinintegrals.xhtml</td>
<td>411</td>
</tr>
<tr>
<td>dlmfintegralrepresentations.xhtml</td>
<td>427</td>
</tr>
<tr>
<td>dlmfmathematicalapplications.xhtml</td>
<td>463</td>
</tr>
<tr>
<td>dlmfmethodsofcomputation.xhtml</td>
<td>472</td>
</tr>
<tr>
<td>dlmfmultidimensionalintegral.xhtml</td>
<td>474</td>
</tr>
<tr>
<td>dlmfnotation.xhtml</td>
<td>500</td>
</tr>
<tr>
<td>dlmfphysicalapplications.xhtml</td>
<td>507</td>
</tr>
<tr>
<td>dlmfpolygammafunctions.xhtml</td>
<td>518</td>
</tr>
<tr>
<td>dlmfgammaandbetafunctions.xhtml</td>
<td>528</td>
</tr>
<tr>
<td>dlmfseriesexpansions.xhtml</td>
<td>544</td>
</tr>
<tr>
<td>dlmfsums.xhtml</td>
<td>560</td>
</tr>
<tr>
<td>dlmfsoftware.xhtml</td>
<td>563</td>
</tr>
<tr>
<td>dlmfspecialvaluesandextrema.xhtml</td>
<td>563</td>
</tr>
<tr>
<td>dlmftables.xhtml</td>
<td>587</td>
</tr>
<tr>
<td>draw.xhtml</td>
<td>633</td>
</tr>
<tr>
<td>draw2donevariable.xhtml</td>
<td>635</td>
</tr>
<tr>
<td>draw2definedcurve.xhtml</td>
<td>636</td>
</tr>
<tr>
<td>draw2polynomialequation.xhtml</td>
<td>638</td>
</tr>
<tr>
<td>draw3dtwovariable.xhtml</td>
<td>639</td>
</tr>
<tr>
<td>draw3ddefinedtube.xhtml</td>
<td>641</td>
</tr>
<tr>
<td>draw3ddefinesurface.xhtml</td>
<td>642</td>
</tr>
<tr>
<td>equidifferential.xhtml</td>
<td>644</td>
</tr>
<tr>
<td>equidifferentiallinear.xhtml</td>
<td>645</td>
</tr>
<tr>
<td>equidifferentialnonlinear.xhtml</td>
<td>648</td>
</tr>
<tr>
<td>equidifferentialpowerseries.xhtml</td>
<td>652</td>
</tr>
<tr>
<td>equationpage.xhtml</td>
<td>654</td>
</tr>
<tr>
<td>equisystemlinear.xhtml</td>
<td>655</td>
</tr>
<tr>
<td>examplesexposedpage.xhtml</td>
<td>657</td>
</tr>
<tr>
<td>factored.xhtml</td>
<td>657</td>
</tr>
<tr>
<td>foundationlibrarydocpage.xhtml</td>
<td>658</td>
</tr>
<tr>
<td>funalgebraicfunctions.xhtml</td>
<td>658</td>
</tr>
<tr>
<td>Content</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>funelementaryfunctions.xhtml</td>
<td>660</td>
</tr>
<tr>
<td>funoperatoralgebra.xhtml</td>
<td>660</td>
</tr>
<tr>
<td>functionpage.xhtml</td>
<td>665</td>
</tr>
<tr>
<td>funpatternmatching.xhtml</td>
<td>666</td>
</tr>
<tr>
<td>funrationalfunctions.xhtml</td>
<td>673</td>
</tr>
<tr>
<td>funsimplification.xhtml</td>
<td>674</td>
</tr>
<tr>
<td>glossarypage.xhtml</td>
<td>677</td>
</tr>
<tr>
<td>graphexamples.xhtml</td>
<td>708</td>
</tr>
<tr>
<td>graphexamplesassorted.xhtml</td>
<td>709</td>
</tr>
<tr>
<td>graphexamplesimplicit.xhtml</td>
<td>710</td>
</tr>
<tr>
<td>graphexampleslistofpoints.xhtml</td>
<td>711</td>
</tr>
<tr>
<td>graphexamplesonevariable.xhtml</td>
<td>713</td>
</tr>
<tr>
<td>graphexamplesparametric.xhtml</td>
<td>714</td>
</tr>
<tr>
<td>graphexamplespolar.xhtml</td>
<td>715</td>
</tr>
<tr>
<td>graphexamplesthreed.xhtml</td>
<td>716</td>
</tr>
<tr>
<td>graphicspage.xhtml</td>
<td>718</td>
</tr>
<tr>
<td>graphviewports.xhtml</td>
<td>718</td>
</tr>
<tr>
<td>graph2d.xhtml</td>
<td>720</td>
</tr>
<tr>
<td>graph2dimPLICIT.xhtml</td>
<td>720</td>
</tr>
<tr>
<td>graph2dlistsofpoints.xhtml</td>
<td>721</td>
</tr>
<tr>
<td>graph2donevariable.xhtml</td>
<td>723</td>
</tr>
<tr>
<td>graph2dparametric.xhtml</td>
<td>725</td>
</tr>
<tr>
<td>graph2dpolar.xhtml</td>
<td>726</td>
</tr>
<tr>
<td>graph3d.xhtml</td>
<td>727</td>
</tr>
<tr>
<td>graph3dobjects.xhtml</td>
<td>728</td>
</tr>
<tr>
<td>graph3dparametric.xhtml</td>
<td>731</td>
</tr>
<tr>
<td>graph3dsurfaces.xhtml</td>
<td>733</td>
</tr>
<tr>
<td>graph3dtubeplots.xhtml</td>
<td>734</td>
</tr>
<tr>
<td>graph3dtwovariables.xhtml</td>
<td>736</td>
</tr>
<tr>
<td>htxtoppage.xhtml</td>
<td>737</td>
</tr>
<tr>
<td>indefiniteintegral.xhtml</td>
<td>737</td>
</tr>
<tr>
<td>introtofloat.xhtml</td>
<td>738</td>
</tr>
<tr>
<td>jenks.xhtml</td>
<td>739</td>
</tr>
<tr>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>menusimplifyexpandexpression.xhtml</td>
<td>797</td>
</tr>
<tr>
<td>menusimplifyexpandlogarithms.xhtml</td>
<td>797</td>
</tr>
<tr>
<td>menusimplifyfactorialsandgamma.xhtml</td>
<td>798</td>
</tr>
<tr>
<td>menusimplifyfactorcomplex.xhtml</td>
<td>798</td>
</tr>
<tr>
<td>menusimplifyfactorexpression.xhtml</td>
<td>798</td>
</tr>
<tr>
<td>menusimplifymoduluscomputation.xhtml</td>
<td>798</td>
</tr>
<tr>
<td>menusimplifysimplifyexpression.xhtml</td>
<td>799</td>
</tr>
<tr>
<td>menusimplifysubstitute.xhtml</td>
<td>799</td>
</tr>
<tr>
<td>menusimplifysimplifyradicals.xhtml</td>
<td>799</td>
</tr>
<tr>
<td>menusimplifytogglealgebraicflag.xhtml</td>
<td>799</td>
</tr>
<tr>
<td>menusimplifytrigsimplification.xhtml</td>
<td>800</td>
</tr>
<tr>
<td>numbasicfunctions.xhtml</td>
<td>800</td>
</tr>
<tr>
<td>numberspage.xhtml</td>
<td>805</td>
</tr>
<tr>
<td>numcardinalnumbers.xhtml</td>
<td>807</td>
</tr>
<tr>
<td>numcomplexnumbers.xhtml</td>
<td>811</td>
</tr>
<tr>
<td>numcontinuedfractions.xhtml</td>
<td>814</td>
</tr>
<tr>
<td>numexamples.xhtml</td>
<td>820</td>
</tr>
<tr>
<td>numfactorization.xhtml</td>
<td>821</td>
</tr>
<tr>
<td>numfinitefields.xhtml</td>
<td>823</td>
</tr>
<tr>
<td>numfloat.xhtml</td>
<td>824</td>
</tr>
<tr>
<td>numfractions.xhtml</td>
<td>826</td>
</tr>
<tr>
<td>numfunctions.xhtml</td>
<td>827</td>
</tr>
<tr>
<td>numgeneralinfo.xhtml</td>
<td>832</td>
</tr>
<tr>
<td>numintegerfractions.xhtml</td>
<td>833</td>
</tr>
<tr>
<td>numintegers.xhtml</td>
<td>833</td>
</tr>
<tr>
<td>nummachinefloats.xhtml</td>
<td>835</td>
</tr>
<tr>
<td>nummachinesizedintegers.xhtml</td>
<td>838</td>
</tr>
<tr>
<td>numnumbertheoreticfunctions.xhtml</td>
<td>840</td>
</tr>
<tr>
<td>numnumericfunctions.xhtml</td>
<td>842</td>
</tr>
<tr>
<td>numoctonions.xhtml</td>
<td>852</td>
</tr>
<tr>
<td>umotherbases.xhtml</td>
<td>855</td>
</tr>
<tr>
<td>numpartialfractions.xhtml</td>
<td>858</td>
</tr>
<tr>
<td>numproblems.xhtml</td>
<td>861</td>
</tr>
<tr>
<td>CONTENTS</td>
<td>xxiii</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------</td>
</tr>
<tr>
<td>bitmaps/axiom1.bitmap</td>
<td>966</td>
</tr>
<tr>
<td>License</td>
<td>971</td>
</tr>
<tr>
<td>Bibliography</td>
<td>973</td>
</tr>
</tbody>
</table>
New Foreword

On October 1, 2001 Axiom was withdrawn from the market and ended life as a commercial product. On September 3, 2002 Axiom was released under the Modified BSD license, including this document. On August 27, 2003 Axiom was released as free and open source software available for download from the Free Software Foundation’s website, Savannah.

Work on Axiom has had the generous support of the Center for Algorithms and Interactive Scientific Computation (CAISS) at City College of New York. Special thanks go to Dr. Gilbert Baumslag for his support of the long term goal.

The online version of this documentation is roughly 1000 pages. In order to make printed versions we’ve broken it up into three volumes. The first volume is tutorial in nature. The second volume is for programmers. The third volume is reference material. We’ve also added a fourth volume for developers. All of these changes represent an experiment in print-on-demand delivery of documentation. Time will tell whether the experiment succeeded.

Axiom has been in existence for over thirty years. It is estimated to contain about three hundred man-years of research and has, as of September 3, 2003, 143 people listed in the credits. All of these people have contributed directly or indirectly to making Axiom available. Axiom is being passed to the next generation. I’m looking forward to future milestones.

With that in mind I’ve introduced the theme of the “30 year horizon”. We must invent the tools that support the Computational Mathematician working 30 years from now. How will research be done when every bit of mathematical knowledge is online and instantly available? What happens when we scale Axiom by a factor of 100, giving us 1.1 million domains? How can we integrate theory with code? How will we integrate theorems and proofs of the mathematics with space-time complexity proofs and running code? What visualization tools are needed? How do we support the conceptual structures and semantics of mathematics in effective ways? How do we support results from the sciences? How do we teach the next generation to be effective Computational Mathematicians?

The “30 year horizon” is much nearer than it appears.

Tim Daly
CAISS, City College of New York
November 10, 2003 ((iHy))
Chapter 1

Overview

This book contains the Firefox browser AJAX routines.

Build Instructions

```bash
mkdir -p /home/silver/bitmaps
cp bookvol11.pamphlet /home/silver
cd /home/silver
export AXIOM=(where)
export PATH=$AXIOM/bin/lib:$AXIOM/bin:$PATH
( cd books ; gcc -o tanglec tanglec.c )
books/tanglec bookvol11.pamphlet > Makefile
make -j 10
axiom -nox
  -> )set mes auto off
  -> )set out mathml on
  -> axServer(8085,multiServ)$AXSERV
```

Now start your browser and go to:
file:///home/silver/rootpage.xhtml
and then do:
Basic Commands -> Calculus -> Differentiate -> Continue
Basic Commands -> Matrix -> Continue

You should see the result of the differentiate appear inline in the page. You can change the values in the text areas, click continue, and see the new result.

The Makefile

This Makefile assumes that it is being run in the final target directory of the build system. We walk the list of PAGES and untangle them from this file, along with any support files referenced by the pages.
Building new pages

To add a new page you need to create a page with the default layout below and add the name of the page to the PAGES variable below.

Most of the pages have a default layout of the form:

\begin{chunk}{pagename.xhtml}
\getchunk{standard head}
</head>
<body>
\getchunk{page head}

   \begin{div} align="center">Page subtitle goes here</div>
   \begin{hr} 

your basic page text goes here.

\getchunk{page foot}
\end{chunk}

There are several things to observe here: Each page lives in its own subsection and its own chunk. The pagename and the chunkname are the same. The chunk includes the standard head. The chunk includes the page head. The chunk includes the page foot.
includes the \textit{page foot} The default page layout cannot communicate with Axiom.

**Communicating with Axiom**

If your page needs to communicate with Axiom you need to add some information in the header of the page. The default page that talks to Axiom has the form:

```latex
\subsection{pagename.xhtml}
<<pagename.xhtml>>=
<<standard head>>
  <script type="text/javascript">
  <<handlefreevars>>
  <<axiom talker>>
  </script>
</head>
<body onload="resetvars();">
<<page head>>
  <div align="center">Page subtitle goes here</div>
  <hr/>
 your text goes here
 your communication blocks go here
<<page foot>>
```

**Handling statements with no free variables**

Use a \texttt{makeRequest} call with a parameter of the id. Note that the div with id of “ansXX” will get replaced automatically and the “ans” prefix is required.

```latex
<li>
  <input type="submit" id="p3" class="subbut"
          onclick="makeRequest('p3');"
          value="sin(x)" />
  <div id="ansp3"><div></div></div>
</li>
```

**Handling statements with free variables**

Free variables exist are used in statements but they are defined in other statements. To make sure the free variables have the correct values you need to include an explicit list of the other ids that need to be executed before this statement. You do this with a call to “handleFree”. It expects a list, enclosed in brackets, of the ids to execute in order. Be certain that the current id is at the end of the list.

```latex
<li>
  <input type="submit" id="p10" class="subbut"
          onclick="handleFree(['p9','p10']);"
          value="roman y" />
  <div id="ansp10"><div></div></div>
</li>
```
Handling domain database lookups

Use an anchor tag of the form:

\[ <a href="db.xhtml?Vector">Vector</a> \]

This will be interpreted by Axiom to mean that you want to do a lookup on a domain, category, or package whose name follows the question mark. Note that the domain name should NOT be an abbreviation.

Handling \)show domain

Use a block containing a showcall of the form:

\[ <li>
    <input type="submit" id="p17" class="subbut"
        onclick="showcall('p17');"
        value="")show DoubleFloat"/>
    <div id="ansp17"></div></li>
\]

Note that the \")show" must be at the beginning of the line and that there can only be one space between the word show and the following argument.

Handling lisp expressions

Use a block containing a lispcall of the form:

\[ <li>
    <input type="submit" id="p2" class="subbut"
        onclick="lispcall('p2');"
        value="(GETDATABASE '|Matrix| 'CONSTRUCTORMODEMAP)"
    />
    <div id="ansp2"></div></li>
\]

Note that this works but you can easily blow away your Axiom session with random statements. Let the coder beware.

Handling expressions that have no output

Use the CSS class=“noresult” tag on the input form. This causes the item to show up in black text. It is still executable and is generally executed by handleFree calls because it contains definitions. However, things like function definitions in Axiom return no interesting output so there is no point in clicking on them.

\[ <li>
    <input type="submit" id="p5" class="noresult"
        onclick="makeRequest('p5');"
        value=")set streams calculate 5" />
    <div id="ansp5"></div></li>
\]

Defined Pages

Every page in this file is extracted by the Makefile. This is the list of pages that will be extracted. It is organized roughly in the hierarchy that you see in the browser pages. This is convention and is not required.
The page hierarchy (used by the Makefile) is:

```
PAGES=rootpage.xhtml \
  testpage.xhtml \
  commandline.xhtml \
  menufileopen.xhtml \
  menufileread.xhtml \
  menufilesave.xhtml \
  menufilesaveas.xhtml \
  menufileuploadlibrary.xhtml \
  menufileinputfile.xhtml \
  menufiletogglepool.xhtml \
  menufileprint.xhtml \
  menufileexit.xhtml \
  menueditcopy.xhtml \
  menueditcopytext.xhtml \
  menueditcopytext.xhtml \
  menueditdeleteselection.xhtml \
  menueditcopyasimage.xhtml \
  menueditselectiontoimage.xhtml \
  menueditselectiontominput.xhtml \
  menueditcut.xhtml \
  menueditpaste.xhtml \
  menuaxiominterrupt.xhtml \
  menuaxiomrestart.xhtml \
  menuaxiomclearmemory.xhtml \
  menuaxiomaddttopath.xhtml \
  menuaxiomshowfunctions.xhtml \
  menuaxiomshowdefinition.xhtml \
  menuaxiomshowvariables.xhtml \
  menuaxiomdeletefunction.xhtml \
  menuaxiomdeletevariable.xhtml \
  menuaxiomtoggletimedisplay.xhtml \
  menuaxiomset.xhtml \
  menuaxiomdisplay.xhtml \
  menuequationssolve.xhtml \
  menuequationsolvesolveequation.xhtml \
  menuequationsrootsolveode.xhtml \
  menuequationsrealrootsseolveode.xhtml \
  menuequationsolvealgebraicsystem.xhtml \
  menuequationsolvealgebraicsystem.xhtml \
  menuequationsolvsolvealgebraicsystem.xhtml \
  menuequationsolveode.xhtml \
  menuequationsolveode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml \
  menuequationsolveodeode.xhtml 
  menuequationsolveodeode.xhtml 
```

```
CHAPTER 1. OVERVIEW

dbopsingleintegerxor.xhtml \
dbopsinh.xhtml \
dbopsetvariableorder.xhtml \
dbopsolve.xhtml \
dbopsqrt.xhtml \
dbopstar.xhtml \
dbopstarstar.xhtml \
dbopsubmatrix.xhtml \
dbopsubmatrix.xhtml \
dbopsubmod.xhtml \
dbopsurface.xhtml \
dbopsumofkthpowerdivisors.xhtml \
dboptan.xhtml \
dboptanh.xhtml \
dboptaylor.xhtml \
dboptimes.xhtml \
dboptotaldegree.xhtml \
dboptrace.xhtml \
dboptranspose.xhtml \
dboptrigs.xhtml \
dboptruncate.xhtml \
dbopvariables.xhtml \
dbopvectorise.xhtml \
dbopvectorspace.xhtml \
dbopvertconcat.xhtml \
dbopwholepart.xhtml \
dbopwholelragits.xhtml \
dbopwrite.xhtml \
dbopzeroof.xhtml \
dbopzerosof.xhtml \
dbopzeroq.xhtml \
dbpolynomialinteger.xhtml \
dbpolynomialfractioninteger.xhtml \
systemvariables.xhtml \
glossarypage.xhtml \
htxtoppage.xhtml \
refsearchpage.xhtml \
topicspage.xhtml \
numberspage.xhtml \
numintegers.xhtml \
umgeneralfraction.xhtml \
umbasicfunctions.xhtml \
umintegerfractions.xhtml \
umnumbertheoreticfunctions.xhtml \
umfactorization.xhtml \
umfunctions.xhtml \
umexamples.xhtml \
umproblems.xhtml \
umfractions.xhtml \
umrationals.xhtml \
umquotientfields.xhtml \
ummachinefloats.xhtml \
umfloat.xhtml \
introtofloat.xhtml \

The Standard Layout

Generally a page has a standard layout using a couple of chunks to minimize the typing. The defined chunks are:

- “standard head” which includes the head element, xmlns, meta, and title element. It also contains the “style” element for CSS information.
- “page head” contains the banner information
- “page foot” contains the trailing page information and the body-end and html-end tags

So the basic layout looks like

```xml
<<standard head>>
  (local and general javascript goes here)
</head>
<body>
<<page head>>
  (local page definition goes here)
<<page foot>>
```

So all you need to worry about are the actual page forms and the javascript to fetch those forms.
For “active pages”, that is those that communicate with Axiom they generally define a javascript function called “commandline” which formats the request to be sent to the host. You also need to include the “axiom talker” chunk. Note that “axiom talker” expects the “commandline” function to exist and calls it. Thus, for the page that handles differentiation calls to Axiom we add the local javascript:

```javascript
<script type="text/javascript">
function commandline(arg) {
    return(document.getElementById('comm').value);
}
</script>
```

This defined the “commandline” function and embeds the “axiom talker”. The “commandline” function knows how to fetch fields from the rest of the page and format them into a single Axiom string. This is page specific code. For example, this shows a single input line which will be sent to the host when the “Continue” is pressed:

```html
<form id="commreq">
  <p>
    Type an input command line to Axiom:<br/>
    <input type="text" id="comm" name="command" size="80"/>
  </p>
</form>
```

Note that the commandline function takes an argument which it gets from the caller, makeRequest. This argument can be used to distinguish which button was pressed.

The div section with id="mathAns" is replaced by the result sent from the server.

**Cascading Style Sheet**

**Standard Style Sheet**

This is the standard CSS style section that gets included with every page. We do this here but it could be a separate style sheet. It hardly matters either way as the style sheet is trivial.

```css
body {
    margin: 0px;
    padding: 0px;
    height: 512px;
    background: url(lightbayou.png) no-repeat;
    -webkit-background-size: cover;
    -moz-background-size: cover;
    -o-background-size: cover;
    background-size: cover;
}
```
— style —

```html
<style>
html {
  background-color: #ECEA81;
}

div.command {
  color:red;
}

div.center {
  color:blue;
}

div.reset {
  visibility:hidden;
}

div.mathml {
  color:blue;
}

input.subbut {
  background-color:#ECEA81;
  border: 0;
  color:green;
  font-family: "Courier New", Courier, monospace;
}

input.noresult {
  background-color:#ECEA81;
  border: 0;
  color:black;
  font-family: "Courier New", Courier, monospace;
}

span.cmd {
  color:green;
  font-family: "Courier New", Courier, monospace;
}

pre {
  font-family: "Courier New", Courier, monospace;
}

\getchunk{standard background}
</style>
```
Menu style sheet

— menu style —

```html
<style>
    form {
        margin-top: 0;
        margin-bottom: 0;
        padding-left: 10px;
    }

    table.main {
        background-color: #ECEA81;
        font-size: 10pt;
        font-family: arial;
    }

    .main A:link {
        font-family: arial;
        color: #016bbd;
    }

    .main A:hover {
        font-family: arial;
        color: #64747A;
    }

    .main A:visited {
        font-family: arial;
        color: #336699;
    }

    /* style the outer div to give it width */
    .menu {
        font-size: 0.85em;
    }

    /* remove all the bullets, borders and padding from the default list styling */
    .menu ul {
        padding: 0;
        width: 1000px;
        margin: 0;
        list-style-type: none;
        white-space: normal;
    }

    .menu ul ul {
        width: 90px;
    }

    /* float the list to make it horizontal and a relative position */
</style>
```
so that you can control the dropdown menu position */
.menu li {
  float: left;
  width: 90px;
  position: relative;
}

/* style the links for the top level */
.menu a, .menu a:visited {
  display: block;
  font-size: 12px;
  text-decoration: none;
  font-weight: bold;
  color: #2952a7;
  width: 99px;
  height: 32px;
  line-height: 29px;
  border: 0px solid #fff;
  border-width: 0px 0px 0 0px;
  text-align: center;
}

/* style the second level links
   if this breaks all the level 2 links appear at once */
.menu ul ul a, .menu ul ul a:visited {
  font-size: 10px;
  font-weight: normal;
  background: #d4d8bd;
  color: #000;
  height: auto;
  line-height: 1em;
  padding: 5px 10px;
  width: 78px
}

/* style the top level hover */
.menu a:hover, .menu ul ul a:hover{
  border: 1px solid #000;
  border-width: 1px 1px 0 1px;
}

.menu :hover > a, .menu ul ul :hover > a {
  border: 1px solid #000;
  border-width: 1px 1px 0 1px;
}

/* style the second level background */
.menu ul a.drop, .menu ul ul a.drop:visited {
  background: #e0d8d0;
}

/* style the second level hover */
.menu ul ul a.drop:hover{
  background: #c9ba65;
standard head

This is the standard head section. It is used on pages that do not include javascript. Note that it does NOT include the </head> so the javascript can be added easily.

— standard head —

<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns:m="http://www.w3.org/1998/Math/MathML">
<head>
<meta http-equiv="Content-Type" content="text/html" charset="us-ascii"/>
<title>Axiom Documentation</title>
\getchunk{style}
</head>

This is the standard page header.

— page head —

<div><center><img src="bitmaps/axiom2.png"/></center></div>
<hr/>
CHAPTER 1. OVERVIEW

Javascript functions

Show only mathml

This function will show only the mathml result in the response. It is useful for particular pages that have lists of equations where all you care about are the answers.

```javascript
function showanswer(mathString, indiv) {
    var mystr = mathString.split("<div\>\"\");
    var mymathstr = mystr[3].concat("</div\>\"\");
    // this turns the string into a dom fragment
    var mathRange = document.createRange();
    var mathBox=document.createElementNS('http://www.w3.org/1999/xhtml','div');
    mathRange.selectNodeContents(mathBox);
    var mymath = mathRange.createContextualFragment(mymathstr);
    mathBox.appendChild(mymath);
    // now we need to format it properly
```
Show Full Answer

This function will show the full answer in the response including the step number, the command, the mathml and the type. The algebra portion is currently ignored.

— showfullanswer —

// The structure returned from Axiom now is
// <div class="stepnum"></div>
// <div class="command"></div>
// <div class="algebra"></div>
// <div class="mathml"></div>
// <div class="type"></div>
// This function will format the output as a console session
<![CDATA[
function showanswer(mathString, indiv) {
    var mystr = mathString.split("</div>");
    // first we prepare the step number
    var mystept1 = mystr[0].lastIndexOf("=");
    var mysteptstr = mystr[0].substr(mystept1+1);
    // now we get the command
    var mycmdt1 = mystr[1].lastIndexOf("=");
    var mycmdstr = mystr[1].substr(mycmdt1+1);
    var myprompt = '('+mysteptstr+') -> '+mycmdstr;
    // now we handle the mathml
    var mymathstr = mystr[3].concat("</div>");
    // and the type, we need to insert the string "Type: ",
    var mytypet1 = mystr[4].lastIndexOf("=");
    var mytypet2 = mystr[4].substr(mythetp1+1).concat("</div>");
    var mytypestr = '<div> Type: '.concat(mytypet2);
    // bang the whole thing together
    var finaldiv='<div class="command">'+myprompt+'</div>'
        '+mymathstr+mytypestr;
    // this turns the string into a dom fragment
    var mathRange = document.createRange();
    var mathBox=document.createElementNS('http://www.w3.org/1999/xhtml','div');
    mathRange.selectNodeContents(mathBox);
    var answer = mathRange.createContextualFragment(finaldiv);
    mathBox.appendChild(answer);
    // and we stick the result into the requested div block as a child.
    var mathAns = document.getElementById(indiv);
    mathAns.removeChild(mathAns.firstChild);
    mathAns.appendChild(mathBox);
}
]]>
Handle Free Variables

<!-- handlefreevars -->

```javascript
// This is a hash table of the values we’ve evaluated.
// This is indexed by a string argument.
// A value of 0 means we need to evaluate the expression
// A value of 1 means we have evaluated the expression
Evaled = new Array();
// this says we should modify the page
hiding = 'show';
// and this is the id of the div tag to modify (defaulted)
thediv = 'mathAns';
// commandline will mark that its arg has been evaled so we don’t repeat
function commandline(arg) {
  Evaled[arg] = 0; // remember that we have set this value
  thediv = 'ans' + arg; // mark where we should put the output
  var ans = document.getElementById(arg).value;
  return(ans);
}
// the function only modifies the page if when we’re showing the
// final result, otherwise it does nothing.
function showanswer(mathString, indiv) {
  if (hiding == 'show') { // only do something useful if we’re showing
    indiv = thediv; // override the argument so we can change it
    var mystr = mathString.split('</div>');
    for (var i = 0; i < mystr.length; i++) {
      if (mystr[i].indexOf('mathml') > 0) {
        var mymathstr = mystr[i].concat('</div>');
      }
    }
  }
}
// this turns the string into a dom fragment
var mathRange = document.createRange();
var mathBox =
  document.createElementNS('http://www.w3.org/1999/xhtml', 'div');
mathRange.selectNodeContents(mathBox);
var mymath = mathRange.createContextualFragment(mymathstr);
mathBox.appendChild(mymath);
// now we need to format it properly
// and we stick the result into the requested div block as a child.
var mathAns = document.getElementById(indiv);
mathAns.removeChild(mathAns.firstChild);
mathAns.appendChild(mathBox);
}
// this function takes a list of expressions ids to evaluate
// the list contains a list of "free" expression ids that need to
// be evaluated before the last expression.
// For each expression id, if it has not yet been evaluated we
// evaluate it "hidden" otherwise we can skip the expression.
// Once we have evaluated all of the free expressions we can
// evaluate the final expression and modify the page.
```
function handleFree(arg) {
    var placename = arg.pop(); // last array val is real
    var mycnt = arg.length; // remaining free vars
    // we handle all of the prerequired expressions quietly
    hiding = 'hide';
    for (var i=0; i<mycnt; i++) { // for each of the free variables
        if (Evaled[arg[i]] == null) { // if we haven't evalued it
            Evaled[arg[i]] = 0; // remember we evalued it
            makeRequest(arg[i]); // initialize the free values
        }
    }
    // and now we start talking to the page again
    hiding = 'show'; // we want to show this
    thediv = 'ans'+placename; // at this div id
    makeRequest(placename); // and we eval and show it
}

axiom talker

<![CDATA[
function ignoreResponse() {}
function resetvars() {
    http_request = new XMLHttpRequest();
    http_request.open('POST', '127.0.0.1:8085', true);
    http_request.onreadystatechange = ignoreResponse;
    http_request.setRequestHeader('Content-Type', 'text/plain');
    http_request.send("command=)clear all");
    return(false);
}
function init() {
}
function makeRequest(arg) {
    http_request = new XMLHttpRequest();
    var command = commandline(arg);
    //alert(command);
    http_request.open('POST', '127.0.0.1:8085', true);
    http_request.onreadystatechange = handleResponse;
    http_request.setRequestHeader('Content-Type', 'text/plain');
    http_request.send("command="+command);
    return(false);
}
function lispcall(arg) {
    http_request = new XMLHttpRequest();
    var command = commandline(arg);
    //alert(command);
    http_request.open('POST', '127.0.0.1:8085', true);
    http_request.onreadystatechange = handleResponse;
]]>
http_request.setRequestHeader('Content-Type', 'text/plain');
http_request.send("lispcall">=command);
return(false);
}
function showcall(arg) {
    http_request = new XMLHttpRequest();
    var command = commandline(arg);
    //alert(command);
    http_request.open('POST', '127.0.0.1:8085', true);
    http_request.onreadystatechange = handleResponse;
    http_request.setRequestHeader('Content-Type', 'text/plain');
    http_request.send("showcall">=command);
    return(false);
}
function interpcall(arg) {
    http_request = new XMLHttpRequest();
    var command = commandline(arg);
    //alert(command);
    http_request.open('POST', '127.0.0.1:8085', true);
    http_request.onreadystatechange = handleResponse;
    http_request.setRequestHeader('Content-Type', 'text/plain');
    http_request.send("interpcall">=command);
    return(false);
}
function handleResponse() {
    if (http_request.readyState == 4) {
        if (http_request.status == 200) {
            showanswer(http_request.responseText,'mathAns');
        } else
        {
            alert('There was a problem with the request.'+ http_request.statusText);
        }
    }
}

Pages

--- testpage.xhtml ---
\getchunk{standard head}
\getchunk{menu style}
    <script type="text/javascript">
\getchunk{handlefreevars}
\getchunk{axiom talker}
    </script>
</head>
<body onload="resetvars();">"\getchunk{page head}
    <div align="center">Test Page</div>
<li>
  <a class="drop" href="edit/selectiontoinput.xhtml">
    Selection to input
  </a>
</li>
<li>
  <a class="drop" href="edit/cut.xhtml">
    Cut
  </a>
</li>
<li>
  <a class="drop" href="edit/paste.xhtml">
    Paste
  </a>
</li>
<li>
  <a class="drop" href="edit/print.xhtml">
    Print
  </a>
</li>
</ul>

<!-- End Edit Menu -->

<!-- Start Axiom Menu -->
<li>
  <a class="toplevel" href="/">
    Axiom
  </a>
</li>
<ul>
  <li>
    <a class="drop" href="axiom/interrupt.xhtml">
      Interrupt
    </a>
  </li>
  <li>
    <a class="drop" href="axiom/restart.xhtml">
      Restart
    </a>
  </li>
  <li>
    <a class="drop" href="axiom/clearmemory.xhtml">
      Clear Memory
    </a>
  </li>
  <li>
    <a class="drop" href="axiom/addtopath.xhtml">
      Add to path
    </a>
  </li>
  <li>
    <a class="drop" href="axiom/showfunctions.xhtml">
      Show functions
    </a>
  </li>
</ul>
<li><a class="drop" href="equations/rootsofpolynomial.xhtml">Roots of polynomial</a></li>
<li><a class="drop" href="equations/Real roots of polynomial.xhtml">Real roots of polynomial</a></li>
<li><a class="drop" href="equations/solvelinearsystem.xhtml">Solve linear system</a></li>
<li><a class="drop" href="equations/solvealgebraicsystem.xhtml">Solve algebraic system</a></li>
<li><a class="drop" href="equations/eliminatevariable.xhtml">Eliminate variable</a></li>
<li><a class="drop" href="equations/solveode.xhtml">Solve ODE</a></li>
<li><a class="drop" href="equations/initialvalueproblem1.xhtml">Initial value problem (1)</a></li>
<li><a class="drop" href="equations/initialvalueproblem2.xhtml">Initial value problem (2)</a></li>
<li><a class="drop" href="equations/boundaryvalueproblem.xhtml">Boundary value problem</a></li>
<li><a class="drop" href="equations/solveodewithlaplace.xhtml">Solve ODE with Laplace</a></li>
<li><a class="drop" href="equations/atvalue.xhtml">At value</a></li>
CHAPTER 1. OVERVIEW

</li>
</ul>
</li>
</ul>

<!-- End Equations Menu -->

<!-- Start Algebra Menu -->

<li>
<a class="toplevel" href="/">
Algebra<!--span class="nabla">&nabla;</span-->
</a>
</li>

<li>
<a class="drop" href="algebra/generatematrix.xhtml">
Generate matrix
</a>
</li>

<li>
<a class="drop" href="algebra/entermatrix.xhtml">
Enter matrix
</a>
</li>

<li>
<a class="drop" href="algebra/invertmatrix.xhtml">
Invert matrix
</a>
</li>

<li>
<a class="drop" href="algebra/characteristicpolynomial.xhtml">
Characteristic polynomial
</a>
</li>

<li>
<a class="drop" href="algebra/determinant.xhtml">
Determinant
</a>
</li>

<li>
<a class="drop" href="algebra/eigenvalues.xhtml">
Eigenvalues
</a>
</li>

<li>
<a class="drop" href="algebra/eigenvectors.xhtml">
Eigenvectors
</a>
</li>

<li>
<a class="drop" href="algebra/adjointmatrix.xhtml">
Adjoint matrix
</a>
</li>

<li>
<a class="drop" href="algebra/transposematrix.xhtml">
Transpose matrix
</a>
</li>
CHAPTER 1. OVERVIEW

<ul>
  <li><a class="drop" href="calculus/integrate.xhtml">Integrate</a></li>
  <li><a class="drop" href="calculus/rischintegrate.xhtml">Risch integrate</a></li>
  <li><a class="drop" href="calculus/changevariable.xhtml">Change variable</a></li>
  <li><a class="drop" href="calculus/differentiate.xhtml">Differentiate</a></li>
  <li><a class="drop" href="calculus/findlimit.xhtml">Find limit</a></li>
  <li><a class="drop" href="calculus/getseries.xhtml">Get series</a></li>
  <li><a class="drop" href="calculus/padeapproximation.xhtml">Padé approximation</a></li>
  <li><a class="drop" href="calculus/calculussum.xhtml">Calculus sum</a></li>
  <li><a class="drop" href="calculus/calculusproduct.xhtml">Calculus product</a></li>
  <li><a class="drop" href="calculus/laplacetransform.xhtml">Laplace transform</a></li>
</ul>
Inverse Laplace transform

Greatest common divisor

Least common multiple

Divide polynomials

Partial fractions

Continued fractions

Simplify expression

Simplify radicals

Factor expression
<li><a class="drop" href="simplify/factorcomplex.xhtml">Factor complex</a></li>
<li><a class="drop" href="simplify/expandexpression.xhtml">Expand expression</a></li>
<li><a class="drop" href="simplify/expandlogarithms.xhtml">Expand logarithms</a></li>
<li><a class="drop" href="simplify/contractlogarithms.xhtml">Contract logarithms</a></li>
<li><a class="drop" href="simplify/factorialsandgamma.xhtml">Factorials and Gamma</a></li>
<li><a class="drop" href="simplify/trigonometricsimplification.xhtml">Trigonometric simplification</a></li>
<li><a class="drop" href="simplify/complexsimplification.xhtml">Complex simplification</a></li>
<li><a class="drop" href="simplify/subtitute.xhtml">Substitute</a></li>
<li><a class="drop" href="simplify/evaluatenounform.xhtml">Evaluate noun form</a></li>
<li><a class="drop" href="simplify/togglealgebraicflag.xhtml">Toggle algebraic flag</a></li>
<li><a class="drop" href="simplify/addalgebraicequality.xhtml">Add algebraic equality</a></li>
<table>
<thead>
<tr>
<th>Character</th>
<th>Decimal</th>
<th>Hex</th>
<th>Entity</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Α</td>
<td>913</td>
<td>00391</td>
<td>Α</td>
<td>greek capital letter alpha</td>
</tr>
<tr>
<td>Β</td>
<td>914</td>
<td>00392</td>
<td>Β</td>
<td>greek capital letter beta</td>
</tr>
<tr>
<td>Γ</td>
<td>915</td>
<td>00393</td>
<td>Γ</td>
<td>greek capital letter gamma</td>
</tr>
<tr>
<td>Δ</td>
<td>916</td>
<td>00394</td>
<td>Δ</td>
<td>greek capital letter delta</td>
</tr>
<tr>
<td>Ε</td>
<td>917</td>
<td>00395</td>
<td>Ε</td>
<td>greek capital letter epsilon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ζ</td>
<td>greek capital letter zeta</td>
</tr>
</tbody>
</table>
<tr valign="top">
<td>&#x00397;</td>
<td>919</td>
<td>&amp;Eta;</td>
<td>greek capital letter eta</td>
</tr>
<tr valign="top">
<td>&#x00398;</td>
<td>920</td>
<td>&amp;Theta;</td>
<td>greek capital letter theta</td>
</tr>
<tr valign="top">
<td>&#x00399;</td>
<td>921</td>
<td>&amp;Iota;</td>
<td>greek capital letter iota</td>
</tr>
<tr valign="top">
<td>&#x0039A;</td>
<td>922</td>
<td>&amp;Kappa;</td>
<td>greek capital letter kappa</td>
</tr>
<tr valign="top">
<td>&#x0039B;</td>
<td>923</td>
<td>&amp;Lambda;</td>
<td>greek capital letter lambda</td>
</tr>
<tr valign="top">
<td>&#x0039C;</td>
<td>924</td>
<td>&amp;Mu;</td>
<td>greek capital letter mu</td>
</tr>
<tr valign="top">
<td>&#x0039D;</td>
<td>925</td>
<td>&amp;Nu;</td>
<td>greek capital letter nu</td>
</tr>
<tr valign="top">
<td>&#x0039E;</td>
<td>926</td>
<td>&amp;Xi;</td>
<td>greek capital letter xi</td>
</tr>
<table>
<thead>
<tr>
<th>ASCII Code</th>
<th>Digit</th>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>003A7</td>
<td>003A7</td>
<td>(\chi)</td>
<td>Greek capital letter chi</td>
</tr>
<tr>
<td>003A8</td>
<td>003A8</td>
<td>(\Psi)</td>
<td>Greek capital letter psi</td>
</tr>
<tr>
<td>003A9</td>
<td>003A9</td>
<td>(\Omega)</td>
<td>Greek capital letter omega</td>
</tr>
<tr>
<td>003B1</td>
<td>003B1</td>
<td>(\alpha)</td>
<td>Greek small letter alpha</td>
</tr>
<tr>
<td>003B2</td>
<td>003B2</td>
<td>(\beta)</td>
<td>Greek small letter beta</td>
</tr>
<tr>
<td>003B3</td>
<td>003B3</td>
<td>(\gamma)</td>
<td>Greek small letter gamma</td>
</tr>
<tr>
<td>003B4</td>
<td>003B4</td>
<td>(\delta)</td>
<td>Greek small letter delta</td>
</tr>
<tr>
<td>003B5</td>
<td>003B5</td>
<td>(\epsilon)</td>
<td>Greek small letter epsilon</td>
</tr>
<tr>
<td>Unicode</td>
<td>Decimal</td>
<td>HTML Entity</td>
<td>Greek Small Letter</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>-------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>03B6</td>
<td>950</td>
<td>&amp;#x003B6;</td>
<td>ζ (zeta)</td>
</tr>
<tr>
<td>03B7</td>
<td>951</td>
<td>&amp;#x003B7;</td>
<td>η (eta)</td>
</tr>
<tr>
<td>03B8</td>
<td>952</td>
<td>&amp;#x003B8;</td>
<td>θ (theta)</td>
</tr>
<tr>
<td>03B9</td>
<td>953</td>
<td>&amp;#x003B9;</td>
<td>ι (iota)</td>
</tr>
<tr>
<td>03BA</td>
<td>954</td>
<td>&amp;#x003BA;</td>
<td>κ (kappa)</td>
</tr>
<tr>
<td>03BB</td>
<td>955</td>
<td>&amp;#x003BB;</td>
<td>λ (lambda)</td>
</tr>
<tr>
<td>03BC</td>
<td>956</td>
<td>&amp;#x003BC;</td>
<td>μ (mu)</td>
</tr>
<tr>
<td>03BD</td>
<td>957</td>
<td>&amp;#x003BD;</td>
<td>ν (nu)</td>
</tr>
<tr>
<td>Unicode</td>
<td>Decimal</td>
<td>HTML Entity</td>
<td>Greek Letter</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>upsilon</td>
<td>966</td>
<td>υ</td>
<td>greek small letter upsilon</td>
</tr>
<tr>
<td>phi</td>
<td>967</td>
<td>φ</td>
<td>greek small letter phi</td>
</tr>
<tr>
<td>chi</td>
<td>968</td>
<td>χ</td>
<td>greek small letter chi</td>
</tr>
<tr>
<td>psi</td>
<td>969</td>
<td>ψ</td>
<td>greek small letter psi</td>
</tr>
<tr>
<td>omega</td>
<td>970</td>
<td>ω</td>
<td>greek small letter omega</td>
</tr>
<tr>
<td>macron</td>
<td>175</td>
<td>¯</td>
<td>macron</td>
</tr>
<tr>
<td>plusmn</td>
<td>177</td>
<td>±</td>
<td>plus-or-minus sign</td>
</tr>
<tr>
<td>multiplication sign</td>
<td></td>
<td>&amp;multiplication sign</td>
<td>multiplication sign</td>
</tr>
<tr>
<td>HTML Entity</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>è</td>
<td>Latin small letter e with grave</td>
<td></td>
<td></td>
</tr>
<tr>
<td>π</td>
<td>Greek small letter pi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ϕ</td>
<td>Greek phi symbol</td>
<td></td>
<td></td>
</tr>
<tr>
<td>…</td>
<td>Horizontal ellipsis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>⋯</td>
<td>Midline horizontal ellipsis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>′</td>
<td>Prime</td>
<td></td>
<td></td>
</tr>
<tr>
<td>⁡</td>
<td>Function application</td>
<td></td>
<td></td>
</tr>
<tr>
<td>⁢</td>
<td>Invisible times</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unicode Value</td>
<td>Decimal Value</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
<td>--------------------</td>
<td></td>
</tr>
<tr>
<td>0x02102</td>
<td>8450</td>
<td>Double-struck capital c</td>
<td></td>
</tr>
<tr>
<td>0x0210D</td>
<td>8565</td>
<td>Double-struck capital h</td>
<td></td>
</tr>
<tr>
<td>0x02111</td>
<td>8565</td>
<td>Black-letter capital i</td>
<td></td>
</tr>
<tr>
<td>0x02113</td>
<td>8567</td>
<td>Script small l</td>
<td></td>
</tr>
<tr>
<td>0x02115</td>
<td>8569</td>
<td>Double-struck capital n</td>
<td></td>
</tr>
<tr>
<td>0x02119</td>
<td>8573</td>
<td>Double-struck capital p</td>
<td></td>
</tr>
<tr>
<td>0x0211A</td>
<td>8574</td>
<td>Double-struck capital q</td>
<td></td>
</tr>
<tr>
<td>0x0211C</td>
<td>8578</td>
<td>Double-struck capital q</td>
<td></td>
</tr>
</tbody>
</table>
\[\text{SECTION 1. OVERVIEW}\]

<p>| (\forall) | 8712 | 2208 | (\in) | element of |
| (\not\exists) | 8713 | 2209 | (\notin) | not an element of |
| (\forall_{\text{small}}) | 8714 | 220A | (\ni) | contains as member |
| (\not\exists_{\text{small}}) | 8715 | 220B | (\ni) | does not contain as member |
| (\forall_{\text{small}}) | 8716 | 220C | end of proof |
| (\not\exists_{\text{small}}) | 8717 | 220D |
| (\forall_{\text{end of proof}}) | 8718 | 220E |
| (\not\exists_{\text{end of proof}}) | 8719 | 220F |</p>
<table>
<thead>
<tr>
<th>HTML Entities</th>
<th>Unicode</th>
<th>LaTeX</th>
</tr>
</thead>
<tbody>
<tr>
<td>∏</td>
<td>(\otimes)</td>
<td>n-ary product</td>
</tr>
<tr>
<td>∇</td>
<td>(\nabla)</td>
<td>n-ary coproduct</td>
</tr>
<tr>
<td>∑</td>
<td>(\sum)</td>
<td>n-ary summation</td>
</tr>
<tr>
<td>−</td>
<td>(-)</td>
<td>minus sign</td>
</tr>
<tr>
<td>∓</td>
<td>(\mp)</td>
<td>minus-or-plus sign</td>
</tr>
<tr>
<td>∔</td>
<td>(\cdot)</td>
<td>dot plus</td>
</tr>
<tr>
<td>÷</td>
<td>(\div)</td>
<td>division slash</td>
</tr>
<tr>
<td>∖</td>
<td>(\setminus)</td>
<td>set minus</td>
</tr>
<tr>
<td>Escape Code</td>
<td>Codepoint</td>
<td>Character</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td><code>\&amp;lowast;</code></td>
<td>8727</td>
<td>asterisk operator</td>
</tr>
<tr>
<td><code>\&amp;ring;</code></td>
<td>8728</td>
<td>ring operator</td>
</tr>
<tr>
<td><code>\&amp;bullet;</code></td>
<td>8729</td>
<td>bullet operator</td>
</tr>
<tr>
<td><code>\&amp;radic;</code></td>
<td>8730</td>
<td>square root</td>
</tr>
<tr>
<td><code>\&amp;cubrt;</code></td>
<td>8731</td>
<td>cube root</td>
</tr>
<tr>
<td><code>\&amp;qudrt;</code></td>
<td>8732</td>
<td>fourth root</td>
</tr>
<tr>
<td><code>\&amp;prop;</code></td>
<td>8733</td>
<td>proportional to</td>
</tr>
<tr>
<td><code>\&amp;infin;</code></td>
<td>8734</td>
<td>infinity</td>
</tr>
<tr>
<td>Unicode</td>
<td>HTML Entity</td>
<td>ASCII</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>8735</td>
<td>∟</td>
<td>221F</td>
</tr>
<tr>
<td>8736</td>
<td>∠</td>
<td>2220</td>
</tr>
<tr>
<td>8737</td>
<td>∡</td>
<td>2221</td>
</tr>
<tr>
<td>8738</td>
<td>∢</td>
<td>2222</td>
</tr>
<tr>
<td>8739</td>
<td>∣</td>
<td>2223</td>
</tr>
<tr>
<td>8740</td>
<td>∤</td>
<td>2224</td>
</tr>
<tr>
<td>8741</td>
<td>∥</td>
<td>2225</td>
</tr>
<tr>
<td>8742</td>
<td>∦</td>
<td>2226</td>
</tr>
<tr>
<td>&lt;td&gt;not parallel to&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;tr valign=&quot;top&quot;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;&amp;#8743;&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;8743&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;2227&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;&amp;and;&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;logical and&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;/tr&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;tr valign=&quot;top&quot;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;&amp;#8744;&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;8744&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;2228&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;&amp;or;&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;logical or&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;/tr&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;tr valign=&quot;top&quot;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;&amp;#8745;&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;8745&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;2229&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;&amp;cap;&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;intersection&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;/tr&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;tr valign=&quot;top&quot;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;&amp;#8746;&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;8746&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;222A&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;&amp;cup;&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;union&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;/tr&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;tr valign=&quot;top&quot;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;&amp;#8747;&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;8747&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;222B&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;&amp;int;&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;integral&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;/tr&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;tr valign=&quot;top&quot;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;&amp;#8748;&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;8748&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;222C&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;double integral&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;/tr&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;tr valign=&quot;top&quot;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;&amp;#8749;&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;8749&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;222D&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;triple integral&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;/tr&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;tr valign=&quot;top&quot;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;&amp;#8750;&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;td&gt;8750&lt;/td&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{\textbackslash E} )</td>
<td>\text{contour integral}</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>( \text{\textbackslash F} )</td>
<td>\text{surface integral}</td>
<td></td>
</tr>
<tr>
<td>( \text{\textbackslash O} )</td>
<td>\text{volume integral}</td>
<td></td>
</tr>
<tr>
<td>( \text{\textbackslash P} )</td>
<td>\text{clockwise integral}</td>
<td></td>
</tr>
<tr>
<td>( \text{\textbackslash Q} )</td>
<td>\text{clockwise contour integral}</td>
<td></td>
</tr>
<tr>
<td>( \text{\textbackslash R} )</td>
<td>\text{anticlockwise contour integral}</td>
<td></td>
</tr>
<tr>
<td>&amp;there4;</td>
<td>\text{therefore}</td>
<td></td>
</tr>
<tr>
<td>\because</td>
<td>\text{because}</td>
<td></td>
</tr>
<tr>
<td>( \frac{8 \times 7}{58} )</td>
<td>( 8758 )</td>
<td>( 2236 )</td>
</tr>
<tr>
<td>----------------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>( \frac{8 \times 7}{59} )</td>
<td>( 8759 )</td>
<td>( 2237 )</td>
</tr>
<tr>
<td>( \frac{8 \times 7}{60} )</td>
<td>( 8760 )</td>
<td>( 2238 )</td>
</tr>
<tr>
<td>( \frac{8 \times 7}{61} )</td>
<td>( 8761 )</td>
<td>( 2239 )</td>
</tr>
<tr>
<td>( \frac{8 \times 7}{62} )</td>
<td>( 8762 )</td>
<td>( 223A )</td>
</tr>
<tr>
<td>( \frac{8 \times 7}{63} )</td>
<td>( 8763 )</td>
<td>( 223B )</td>
</tr>
<tr>
<td>( \frac{8 \times 7}{64} )</td>
<td>( 8764 )</td>
<td>( 223C )</td>
</tr>
<tr>
<td>( \frac{8 \times 7}{65} )</td>
<td>( 8765 )</td>
<td>( 223D )</td>
</tr>
<tr>
<td>Character</td>
<td>Decimal</td>
<td>Hexadecimal</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>₡</td>
<td>8766</td>
<td>223E</td>
</tr>
<tr>
<td>₢</td>
<td>8767</td>
<td>223F</td>
</tr>
<tr>
<td>₣</td>
<td>8768</td>
<td>2240</td>
</tr>
<tr>
<td>₤</td>
<td>8769</td>
<td>2241</td>
</tr>
<tr>
<td>₥</td>
<td>8770</td>
<td>2242</td>
</tr>
<tr>
<td>₦</td>
<td>8771</td>
<td>2243</td>
</tr>
<tr>
<td>₧</td>
<td>8772</td>
<td>2244</td>
</tr>
<tr>
<td>₨</td>
<td>8773</td>
<td>2245</td>
</tr>
</tbody>
</table>
approximately equal to

approximately but not actually equal to

neither approximately nor actually equal to

almost equal to

almost equal or equal to

triple tilde

triple equal to
<table>
<thead>
<tr>
<th>Unicode</th>
<th>HTML Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8781</td>
<td>&amp;8781;</td>
<td>equivalent to</td>
</tr>
<tr>
<td>224D</td>
<td>&amp;224D;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>equivalent to</td>
</tr>
<tr>
<td>8782</td>
<td>&amp;8782;</td>
<td>geometrically equivalent to</td>
</tr>
<tr>
<td>224E</td>
<td>&amp;224E;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>geometrically equivalent to</td>
</tr>
<tr>
<td>8783</td>
<td>&amp;8783;</td>
<td>difference between</td>
</tr>
<tr>
<td>224F</td>
<td>&amp;224F;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>difference between</td>
</tr>
<tr>
<td>8784</td>
<td>&amp;8784;</td>
<td>approaches the limit</td>
</tr>
<tr>
<td>2250</td>
<td>&amp;2250;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>approaches the limit</td>
</tr>
<tr>
<td>8785</td>
<td>&amp;8785;</td>
<td>geometrically equal to</td>
</tr>
<tr>
<td>2251</td>
<td>&amp;2251;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>geometrically equal to</td>
</tr>
<tr>
<td>8786</td>
<td>&amp;8786;</td>
<td>approximately equal to or the image of</td>
</tr>
<tr>
<td>2252</td>
<td>&amp;2252;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>approximately equal to or the image of</td>
</tr>
<tr>
<td>8787</td>
<td>&amp;8787;</td>
<td>image of or approximately equal to</td>
</tr>
<tr>
<td>2253</td>
<td>&amp;2253;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>image of or approximately equal to</td>
</tr>
<tr>
<td>8788</td>
<td>&amp;8788;</td>
<td>colon equals</td>
</tr>
<tr>
<td>2254</td>
<td>&amp;2254;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>colon equals</td>
</tr>
<tr>
<td>(\Rightarrow)</td>
<td>(\Rightarrow)</td>
<td>(\Rightarrow)</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>equals colon</td>
<td>ring in equal to</td>
<td>ring equal to</td>
</tr>
<tr>
<td>estimates</td>
<td>star equals</td>
<td></td>
</tr>
<tr>
<td>Unicode</td>
<td>HTML Entity</td>
<td>ASCII</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td><code>&amp;8797;</code></td>
<td><code>&amp;#8797;</code></td>
<td><code>8797</code></td>
</tr>
<tr>
<td><code>&amp;8798;</code></td>
<td><code>&amp;#8798;</code></td>
<td><code>8798</code></td>
</tr>
<tr>
<td><code>&amp;8799;</code></td>
<td><code>&amp;#8799;</code></td>
<td><code>8799</code></td>
</tr>
<tr>
<td><code>&amp;8800;</code></td>
<td><code>&amp;#8800;</code></td>
<td><code>8800</code></td>
</tr>
<tr>
<td><code>&amp;8801;</code></td>
<td><code>&amp;#8801;</code></td>
<td><code>8801</code></td>
</tr>
<tr>
<td><code>&amp;8802;</code></td>
<td><code>&amp;#8802;</code></td>
<td><code>8802</code></td>
</tr>
<tr>
<td><code>&amp;8803;</code></td>
<td><code>&amp;#8803;</code></td>
<td><code>8803</code></td>
</tr>
</tbody>
</table>
少于或等于

大于或等于

小于等于

大于等于

小于不等于

大于不等于

大大于

大等于

大大于不等于

大大于不等于
<table>
<thead>
<tr>
<th>Unicode</th>
<th>HTML Entity</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+00AD</td>
<td>≬</td>
<td>between</td>
</tr>
<tr>
<td>U+00B1</td>
<td>≭</td>
<td>not equivalent to</td>
</tr>
<tr>
<td>U+00B2</td>
<td>≮</td>
<td>not less-than</td>
</tr>
<tr>
<td>U+00B3</td>
<td>≯</td>
<td>not greater-than</td>
</tr>
<tr>
<td>U+00B4</td>
<td>≰</td>
<td>neither less-than nor equal to</td>
</tr>
<tr>
<td>U+00B5</td>
<td>≱</td>
<td>neither greater-than nor equal to</td>
</tr>
<tr>
<td>U+00B6</td>
<td>≲</td>
<td>less-than or equivalent to</td>
</tr>
<tr>
<td>U+00B7</td>
<td>≳</td>
<td>greater-than or equivalent to</td>
</tr>
<tr>
<td>Unicode</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>U+2274</td>
<td>≠</td>
<td></td>
</tr>
<tr>
<td>U+2275</td>
<td>≮</td>
<td></td>
</tr>
<tr>
<td>U+2276</td>
<td>≯</td>
<td></td>
</tr>
<tr>
<td>U+2277</td>
<td>≮≠</td>
<td></td>
</tr>
<tr>
<td>U+2278</td>
<td>≮≯</td>
<td></td>
</tr>
<tr>
<td>U+227A</td>
<td>≺</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unicode</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+227B</td>
<td>≺≺</td>
</tr>
<tr>
<td>Symbol</td>
<td>Code</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>≦</td>
<td>8828</td>
</tr>
<tr>
<td>≧</td>
<td>8829</td>
</tr>
<tr>
<td>≼</td>
<td>8830</td>
</tr>
<tr>
<td>≽</td>
<td>8831</td>
</tr>
<tr>
<td>⊈</td>
<td>8832</td>
</tr>
<tr>
<td>⊉</td>
<td>8833</td>
</tr>
<tr>
<td>⊆</td>
<td>8834</td>
</tr>
<tr>
<td>HTML Entity</td>
<td>Unicode</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td><code>&amp;sup;</code></td>
<td>8835</td>
</tr>
<tr>
<td><code>&amp;nsub;</code></td>
<td>8836</td>
</tr>
<tr>
<td><code>&amp;nsubseteq;</code></td>
<td>8837</td>
</tr>
<tr>
<td><code>&amp;subseteq;</code></td>
<td>8838</td>
</tr>
<tr>
<td><code>&amp;supe;</code></td>
<td>8839</td>
</tr>
<tr>
<td><code>&amp;nsubseteqq;</code></td>
<td>8840</td>
</tr>
<tr>
<td><code>&amp;nsubseteqq;</code></td>
<td>8841</td>
</tr>
<tr>
<td><code>&amp;notsubsetwithnequal;</code></td>
<td>8842</td>
</tr>
<tr>
<td>Unicode Code</td>
<td>HTML Entity Code</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>U+228B</td>
<td>(⊋)</td>
</tr>
<tr>
<td>U+228C</td>
<td>(⊌)</td>
</tr>
<tr>
<td>U+228D</td>
<td>(⊍)</td>
</tr>
<tr>
<td>U+228E</td>
<td>(⊎)</td>
</tr>
<tr>
<td>U+228F</td>
<td>(⊏)</td>
</tr>
<tr>
<td>U+2290</td>
<td>(⊐)</td>
</tr>
<tr>
<td>U+2291</td>
<td>(⊑)</td>
</tr>
<tr>
<td>( \texttt{&amp;amp;#8851;} )</td>
<td>8851</td>
</tr>
<tr>
<td>( \texttt{&amp;amp;#8852;} )</td>
<td>8852</td>
</tr>
<tr>
<td>( \texttt{&amp;amp;#8853;} )</td>
<td>8853</td>
</tr>
<tr>
<td>( \texttt{&amp;amp;#8854;} )</td>
<td>8854</td>
</tr>
<tr>
<td>( \texttt{&amp;amp;#8855;} )</td>
<td>8855</td>
</tr>
<tr>
<td>( \texttt{&amp;amp;#8856;} )</td>
<td>8856</td>
</tr>
<tr>
<td>( \texttt{&amp;amp;#8857;} )</td>
<td>8857</td>
</tr>
<tr>
<td>( \texttt{&amp;amp;#8858;} )</td>
<td>8858</td>
</tr>
<tr>
<td>HTML Entity</td>
<td>Unicode</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td><code>&amp;229A;</code></td>
<td>8859</td>
</tr>
<tr>
<td><code>&amp;#8859;</code></td>
<td>229B</td>
</tr>
<tr>
<td><code>&amp;229C;</code></td>
<td>8860</td>
</tr>
<tr>
<td><code>&amp;#8860;</code></td>
<td>229D</td>
</tr>
<tr>
<td><code>&amp;229E;</code></td>
<td>8861</td>
</tr>
<tr>
<td><code>&amp;#8861;</code></td>
<td>229F</td>
</tr>
<tr>
<td><code>&amp;22A0;</code></td>
<td>8862</td>
</tr>
<tr>
<td><code>&amp;#8862;</code></td>
<td>22A1</td>
</tr>
<tr>
<td>Character</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
</tr>
<tr>
<td>&amp;8866;</td>
<td>right tack</td>
</tr>
<tr>
<td>&amp;8867;</td>
<td>left tack</td>
</tr>
<tr>
<td>&amp;8868;</td>
<td>down tack</td>
</tr>
<tr>
<td>&amp;8869;</td>
<td>up tack</td>
</tr>
<tr>
<td>&amp;8870;</td>
<td>assertion</td>
</tr>
<tr>
<td>&amp;8871;</td>
<td>models</td>
</tr>
<tr>
<td>&amp;8872;</td>
<td>true</td>
</tr>
<tr>
<td>&amp;8873;</td>
<td>forces</td>
</tr>
<tr>
<td>Character</td>
<td>Code</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>≜</td>
<td>U+2274</td>
</tr>
<tr>
<td>≝</td>
<td>U+2275</td>
</tr>
<tr>
<td>≞</td>
<td>U+2276</td>
</tr>
<tr>
<td>≟</td>
<td>U+2277</td>
</tr>
<tr>
<td>≠</td>
<td>U+2278</td>
</tr>
<tr>
<td>≡</td>
<td>U+2279</td>
</tr>
<tr>
<td>≢</td>
<td>U+227A</td>
</tr>
<tr>
<td>≣</td>
<td>U+227B</td>
</tr>
<tr>
<td>Relation</td>
<td>Symbol</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
</tr>
<tr>
<td>succeeds under relation</td>
<td>$\prec$</td>
</tr>
<tr>
<td>contains as normal subgroup</td>
<td>$\leq$</td>
</tr>
<tr>
<td>contains as normal subgroup or equal to</td>
<td>$\subseteq$</td>
</tr>
<tr>
<td>original of</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>image of</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>multimap</td>
<td>$\mapsto$</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>---------------------</td>
</tr>
<tr>
<td>8889</td>
<td>hermitian conjugate matrix</td>
</tr>
<tr>
<td>22B9</td>
<td>intercalate</td>
</tr>
<tr>
<td>8890</td>
<td>xor</td>
</tr>
<tr>
<td>22BA</td>
<td>nand</td>
</tr>
<tr>
<td>8891</td>
<td>nor</td>
</tr>
<tr>
<td>22BB</td>
<td>right angle with arc</td>
</tr>
<tr>
<td>8892</td>
<td>right triangle</td>
</tr>
<tr>
<td>22BC</td>
<td>n-ary logical and</td>
</tr>
<tr>
<td>8893</td>
<td></td>
</tr>
<tr>
<td>22BD</td>
<td></td>
</tr>
<tr>
<td>8894</td>
<td></td>
</tr>
<tr>
<td>22BE</td>
<td></td>
</tr>
<tr>
<td>8895</td>
<td></td>
</tr>
<tr>
<td>22BF</td>
<td></td>
</tr>
<tr>
<td>8896</td>
<td></td>
</tr>
<tr>
<td>22C0</td>
<td></td>
</tr>
<tr>
<td>8897</td>
<td></td>
</tr>
<tr>
<td>22C1</td>
<td></td>
</tr>
<tr>
<td>8898</td>
<td></td>
</tr>
<tr>
<td>22C2</td>
<td></td>
</tr>
<tr>
<td>8899</td>
<td></td>
</tr>
<tr>
<td>22C3</td>
<td></td>
</tr>
<tr>
<td>8890</td>
<td></td>
</tr>
<tr>
<td>22C4</td>
<td></td>
</tr>
<tr>
<td>8891</td>
<td></td>
</tr>
<tr>
<td>22C5</td>
<td></td>
</tr>
<tr>
<td>8892</td>
<td></td>
</tr>
<tr>
<td>22C6</td>
<td></td>
</tr>
<tr>
<td>8893</td>
<td></td>
</tr>
<tr>
<td>22C7</td>
<td></td>
</tr>
<tr>
<td>8894</td>
<td></td>
</tr>
<tr>
<td>22C8</td>
<td></td>
</tr>
<tr>
<td>8895</td>
<td></td>
</tr>
<tr>
<td>22C9</td>
<td></td>
</tr>
<tr>
<td>8896</td>
<td></td>
</tr>
<tr>
<td>22CA</td>
<td></td>
</tr>
<tr>
<td>8897</td>
<td></td>
</tr>
<tr>
<td>22CB</td>
<td></td>
</tr>
<tr>
<td>8898</td>
<td></td>
</tr>
<tr>
<td>22CC</td>
<td></td>
</tr>
<tr>
<td>8899</td>
<td></td>
</tr>
<tr>
<td>22CD</td>
<td></td>
</tr>
<tr>
<td>8890</td>
<td></td>
</tr>
<tr>
<td>22CE</td>
<td></td>
</tr>
<tr>
<td>8891</td>
<td></td>
</tr>
<tr>
<td>22CF</td>
<td></td>
</tr>
<tr>
<td>8892</td>
<td></td>
</tr>
<tr>
<td>22D0</td>
<td></td>
</tr>
<tr>
<td>8893</td>
<td></td>
</tr>
<tr>
<td>22D1</td>
<td></td>
</tr>
<tr>
<td>8894</td>
<td></td>
</tr>
<tr>
<td>22D2</td>
<td></td>
</tr>
<tr>
<td>8895</td>
<td></td>
</tr>
<tr>
<td>22D3</td>
<td></td>
</tr>
<tr>
<td>8896</td>
<td></td>
</tr>
<tr>
<td>22D4</td>
<td></td>
</tr>
<tr>
<td>8897</td>
<td></td>
</tr>
<tr>
<td>22D5</td>
<td></td>
</tr>
<tr>
<td>8898</td>
<td></td>
</tr>
<tr>
<td>22D6</td>
<td></td>
</tr>
<tr>
<td>8899</td>
<td></td>
</tr>
<tr>
<td>22D7</td>
<td></td>
</tr>
<tr>
<td>8890</td>
<td></td>
</tr>
<tr>
<td>22D8</td>
<td></td>
</tr>
<tr>
<td>8891</td>
<td></td>
</tr>
<tr>
<td>22D9</td>
<td></td>
</tr>
<tr>
<td>8892</td>
<td></td>
</tr>
<tr>
<td>22DA</td>
<td></td>
</tr>
<tr>
<td>8893</td>
<td></td>
</tr>
<tr>
<td>22DB</td>
<td></td>
</tr>
<tr>
<td>8894</td>
<td></td>
</tr>
<tr>
<td>22DC</td>
<td></td>
</tr>
<tr>
<td>8895</td>
<td></td>
</tr>
<tr>
<td>22DD</td>
<td></td>
</tr>
<tr>
<td>8896</td>
<td></td>
</tr>
<tr>
<td>22DE</td>
<td></td>
</tr>
<tr>
<td>8897</td>
<td></td>
</tr>
<tr>
<td>22DF</td>
<td></td>
</tr>
<tr>
<td>8898</td>
<td></td>
</tr>
<tr>
<td>22E0</td>
<td></td>
</tr>
<tr>
<td>8899</td>
<td></td>
</tr>
<tr>
<td>22E1</td>
<td></td>
</tr>
<tr>
<td>8890</td>
<td></td>
</tr>
<tr>
<td>22E2</td>
<td></td>
</tr>
<tr>
<td>8891</td>
<td></td>
</tr>
<tr>
<td>22E3</td>
<td></td>
</tr>
<tr>
<td>8892</td>
<td></td>
</tr>
<tr>
<td>22E4</td>
<td></td>
</tr>
<tr>
<td>8893</td>
<td></td>
</tr>
<tr>
<td>22E5</td>
<td></td>
</tr>
<tr>
<td>8894</td>
<td></td>
</tr>
<tr>
<td>22E6</td>
<td></td>
</tr>
<tr>
<td>8895</td>
<td></td>
</tr>
<tr>
<td>22E7</td>
<td></td>
</tr>
<tr>
<td>8896</td>
<td></td>
</tr>
<tr>
<td>22E8</td>
<td></td>
</tr>
<tr>
<td>8897</td>
<td></td>
</tr>
<tr>
<td>22E9</td>
<td></td>
</tr>
<tr>
<td>8898</td>
<td></td>
</tr>
<tr>
<td>22EA</td>
<td></td>
</tr>
<tr>
<td>8899</td>
<td></td>
</tr>
<tr>
<td>22EB</td>
<td></td>
</tr>
<tr>
<td>8890</td>
<td></td>
</tr>
<tr>
<td>22EC</td>
<td></td>
</tr>
<tr>
<td>8891</td>
<td></td>
</tr>
<tr>
<td>22ED</td>
<td></td>
</tr>
<tr>
<td>8892</td>
<td></td>
</tr>
<tr>
<td>22EE</td>
<td></td>
</tr>
<tr>
<td>8893</td>
<td></td>
</tr>
<tr>
<td>22EF</td>
<td></td>
</tr>
<tr>
<td>8894</td>
<td></td>
</tr>
<tr>
<td>22F0</td>
<td></td>
</tr>
<tr>
<td>8895</td>
<td></td>
</tr>
<tr>
<td>22F1</td>
<td></td>
</tr>
<tr>
<td>8896</td>
<td></td>
</tr>
<tr>
<td>22F2</td>
<td></td>
</tr>
<tr>
<td>8897</td>
<td></td>
</tr>
<tr>
<td>22F3</td>
<td></td>
</tr>
<tr>
<td>8898</td>
<td></td>
</tr>
<tr>
<td>22F4</td>
<td></td>
</tr>
<tr>
<td>8899</td>
<td></td>
</tr>
<tr>
<td>22F5</td>
<td></td>
</tr>
<tr>
<td>8890</td>
<td></td>
</tr>
<tr>
<td>22F6</td>
<td></td>
</tr>
<tr>
<td>8891</td>
<td></td>
</tr>
<tr>
<td>22F7</td>
<td></td>
</tr>
<tr>
<td>8892</td>
<td></td>
</tr>
<tr>
<td>22F8</td>
<td></td>
</tr>
<tr>
<td>8893</td>
<td></td>
</tr>
<tr>
<td>22F9</td>
<td></td>
</tr>
<tr>
<td>8894</td>
<td></td>
</tr>
<tr>
<td>22FA</td>
<td></td>
</tr>
<tr>
<td>8895</td>
<td></td>
</tr>
<tr>
<td>22FB</td>
<td></td>
</tr>
<tr>
<td>8896</td>
<td></td>
</tr>
<tr>
<td>22FC</td>
<td></td>
</tr>
<tr>
<td>8897</td>
<td></td>
</tr>
<tr>
<td>22FD</td>
<td></td>
</tr>
<tr>
<td>8898</td>
<td></td>
</tr>
<tr>
<td>22FE</td>
<td></td>
</tr>
<tr>
<td>8899</td>
<td></td>
</tr>
<tr>
<td>22FF</td>
<td></td>
</tr>
<tr>
<td>Unicode</td>
<td>ASCII</td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
</tr>
<tr>
<td>8897</td>
<td>8897</td>
</tr>
<tr>
<td>8898</td>
<td>8898</td>
</tr>
<tr>
<td>8899</td>
<td>8899</td>
</tr>
<tr>
<td>8900</td>
<td>8900</td>
</tr>
<tr>
<td>8901</td>
<td>8901</td>
</tr>
<tr>
<td>8902</td>
<td>8902</td>
</tr>
<tr>
<td>8903</td>
<td>8903</td>
</tr>
<tr>
<td>8904</td>
<td>8904</td>
</tr>
</tbody>
</table>

*Note: The above table provides examples of how to represent symbolic operators using ASCII characters for use in documentation or text.*
<table>
<thead>
<tr>
<th>Code</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8905</td>
<td>left normal factor dim direct product</td>
<td></td>
</tr>
<tr>
<td>8906</td>
<td>right normal factor dim direct product</td>
<td></td>
</tr>
<tr>
<td>8907</td>
<td>left semi-direct product</td>
<td></td>
</tr>
<tr>
<td>8908</td>
<td>right semi-direct product</td>
<td></td>
</tr>
<tr>
<td>8909</td>
<td>reversed tilde equals</td>
<td></td>
</tr>
<tr>
<td>8910</td>
<td>curly logical or</td>
<td></td>
</tr>
<tr>
<td>8911</td>
<td>curly logical and</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 1. OVERVIEW

<table>
<thead>
<tr>
<th>Code</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>22D0</td>
<td></td>
<td>double subset</td>
</tr>
<tr>
<td>22D1</td>
<td></td>
<td>double superset</td>
</tr>
<tr>
<td>22D2</td>
<td></td>
<td>double intersection</td>
</tr>
<tr>
<td>22D3</td>
<td></td>
<td>double union</td>
</tr>
<tr>
<td>22D4</td>
<td></td>
<td>pitchfork</td>
</tr>
<tr>
<td>22D5</td>
<td></td>
<td>equal and parallel to</td>
</tr>
<tr>
<td>22D6</td>
<td></td>
<td>less-than with dot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>greater-than with dot</td>
</tr>
<tr>
<td>Character</td>
<td>Entity</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>⋘</td>
<td>8920</td>
<td>very much less-than</td>
</tr>
<tr>
<td>⋙</td>
<td>8921</td>
<td>very much greater-than</td>
</tr>
<tr>
<td>⋚</td>
<td>8922</td>
<td>less-than equal to or greater-than</td>
</tr>
<tr>
<td>⋛</td>
<td>8923</td>
<td>greater-than equal to or less-than</td>
</tr>
<tr>
<td>⋜</td>
<td>8924</td>
<td>equal to or less-than</td>
</tr>
<tr>
<td>⋝</td>
<td>8925</td>
<td>equal to or greater-than</td>
</tr>
<tr>
<td>⋞</td>
<td>8926</td>
<td>equal to or precedes</td>
</tr>
<tr>
<td>⋟</td>
<td>8927</td>
<td>equal to or succeeds</td>
</tr>
<tr>
<td>Unicode</td>
<td>HTML Entity</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>U+22E0</td>
<td>⋠</td>
<td>does not precede or equal</td>
</tr>
<tr>
<td>U+22E1</td>
<td>⋡</td>
<td>does not succeed or equal</td>
</tr>
<tr>
<td>U+22E2</td>
<td>⋢</td>
<td>not square image of or equal to</td>
</tr>
<tr>
<td>U+22E3</td>
<td>⋣</td>
<td>not square original of or equal to</td>
</tr>
<tr>
<td>U+22E4</td>
<td>⋤</td>
<td>square image of or not equal to</td>
</tr>
<tr>
<td>U+22E5</td>
<td>⋥</td>
<td>square original of or not equal to</td>
</tr>
<tr>
<td>U+22E6</td>
<td>⋦</td>
<td>less-than but not equivalent to</td>
</tr>
<tr>
<td>Unicode</td>
<td>decimal</td>
<td>HTML entity</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>8936</td>
<td>2286</td>
<td>&lt;td&gt;greater-than but not equivalent to&lt;/td&gt;</td>
</tr>
<tr>
<td>8937</td>
<td>2287</td>
<td>&lt;td&gt;precedes but not equivalent to&lt;/td&gt;</td>
</tr>
<tr>
<td>8938</td>
<td>2288</td>
<td>&lt;td&gt;succeeds but not equivalent to&lt;/td&gt;</td>
</tr>
<tr>
<td>8939</td>
<td>2289</td>
<td>&lt;td&gt;not normal subgroup of&lt;/td&gt;</td>
</tr>
<tr>
<td>8940</td>
<td>2290</td>
<td>&lt;td&gt;does not contain as normal subgroup&lt;/td&gt;</td>
</tr>
<tr>
<td>8941</td>
<td>2291</td>
<td>&lt;td&gt;not normal subgroup of or equal to&lt;/td&gt;</td>
</tr>
<tr>
<td>8942</td>
<td>2292</td>
<td>&lt;td&gt;does not contain as normal subgroup or equal&lt;/td&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;td&gt;vertical ellipsis&lt;/td&gt;</td>
</tr>
<tr>
<td>( \text{\textbackslash \texttt{&amp;#8943;}} )</td>
<td>8043</td>
<td>( \text{\textbackslash \texttt{&amp;#8944;}} )</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>midline horizontal ellipsis</td>
<td>up right diagonal ellipsis</td>
<td>down right diagonal ellipsis</td>
</tr>
<tr>
<td>element of with dot above</td>
<td>element of with overbar</td>
<td></td>
</tr>
<tr>
<td>Unicode</td>
<td>HTML Entity</td>
<td>Hexadecimal</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>U+01F17</td>
<td>⋷</td>
<td>22F7</td>
</tr>
<tr>
<td>U+01F18</td>
<td>⋸</td>
<td>22F8</td>
</tr>
<tr>
<td>U+01F19</td>
<td>⋹</td>
<td>22F9</td>
</tr>
<tr>
<td>U+01F1A</td>
<td>⋺</td>
<td>22FA</td>
</tr>
<tr>
<td>U+01F1B</td>
<td>⋻</td>
<td>22FB</td>
</tr>
<tr>
<td>U+01F1C</td>
<td>⋼</td>
<td>22FC</td>
</tr>
<tr>
<td>U+01F1D</td>
<td>⋽</td>
<td>22FD</td>
</tr>
<tr>
<td>U+01F1E</td>
<td>⋾</td>
<td>22FE</td>
</tr>
</tbody>
</table>
Axiom provides various facilities for treating topics in abstract algebra

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>A Number Theory</td>
<td>Topics in algebraic number theory</td>
</tr>
<tr>
<td>Group Theory</td>
<td>Permutation groups; representation theory</td>
</tr>
</tbody>
</table>

algebrapage.xhtml

— algebrapage.xhtml —

alggrouptehory.xhtml

— alggrouptehory.xhtml —
Axiom can work with individual permutations, permutation groups and do representation theory.

<ul>
    <li><a href="alggrouptheorygroup.xhtml">Info on Group Theory</a></li>
    <li><a href="alggrouptheoryreptheory.xhtml">Info on Representation Theory</a></li>
    <li><a href="alggrouptheoryrepa6.xhtml">Representations of A6</a></li>
</ul>

A group is a set \( G \) together with an associative operation \(*\) satisfying the axioms of existence of a unit element and an inverse of every element of the group. The Axiom category \(<a href="db.xhtml?Group">Group</a>\) represents this setting. Many data structures in Axiom are groups and therefore there is a large variety of examples as fields and polynomials, although the main interest there is not a group structure.

To work with and in groups in a concrete manner some way of representing groups has to be chosen. A group can be given as a list of generators and as a set of relations. If there are no relations, then we have a free group, realized in the domain \(<a href="db.xhtml?FreeMonoid">FreeMonoid</a>\) which won’t be discussed here. We consider permutation groups, where a group is realized as a subgroup of the symmetric group of a set, i.e., the group of all bijections of a set, the operation being the composition of maps. Indeed, every group can be realized this way, although this may not be practical.

Furthermore group elements can be given as invertible matrices. The group operation is reflected by matrix multiplication. More precise in representation theory group homomorphisms from a group to general linear groups are constructed. Some algorithms are implemented in Axiom.
In what follows you’ll see how to use Axiom to get all the irreducible representations of the alternating group $A_6$ over the field with two elements (GF 2). First, we generate $A_6$ by a three-cycle: $x=(1,2,3)$ and a 5-cycle: $y=(2,3,4,5,6)$. Next we have Axiom calculate the permutation representation over the integers and over GF 2:

<ul>
  <li><input type="submit" id="p1" class="subbut" onclick="makeRequest('p1');" value="genA6:LIST PERM INT:=[cycle [1,2,3],cycle [2,3,4,5,6]]" />
    <div id="ansp1"><div></div></div></li>
  <li><input type="submit" id="p2" class="subbut" onclick="handleFree(['p1','p2']);" value="pRA6:=permutationRepresentation(genA6,6)" />
    <div id="ansp2"><div></div></div></li>
</ul>

Now we apply Parker’s ‘Meat-Axe’ and split it:

<ul>
  <li><input type="submit" id="p3" class="subbut" onclick="handleFree(['p1','p2','p3']);" value="sp0:=meatAxe(pRA6::(LIST MATRIX PF 2))" />
    <div id="ansp3"><div></div></div></li>
</ul>

We have found the trivial module as a quotient module and a 5-dimensional sub-module. Try to split again:

<ul>
  <li><input type="submit" id="p4" class="subbut" onclick="handleFree(['p1','p2','p3','p4']);" value="sp1:=meatAxe sp0.1" />
    <div id="ansp4"><div></div></div></li>
</ul>

and we find a 4-dimensional sub-module and the trivial one again. Now we
can test if this representation is absolutely irreducible.

And we see that this 4-dimensional representation is absolutely irreducible. So, we have found a second irreducible representation. Now, we construct a representation by reducing an irreducible one of the symmetric group $S_6$ over the integers mod 2. We take the one labelled by the partition [2,2,1,1] and restrict it to $A_6$:

This gave both a five and a four dimensional representation. Now we take the 4-dimensional one and we shall see that it is absolutely irreducible:

The two 4-dimensional representations are not equivalent:

So we have found a third irreducible representation. Now we construct a new representation using the tensor product and try to split it:
The representation is irreducible, but it may be not absolutely irreducible.

So let's try the same procedure over the field with 4 elements:

Now we find two 8-dimensional representations, $d_{A6d8a}$ and $d_{A6d8b}$. Both are absolutely irreducible.

and they are not equivalent.

So we have found five absolutely irreducible representations of $A_6$ in characteristic 2. General theory now tells us that there are no more irreducible ones. Here, for future reference are all the absolutely irreducible 2-module representations of $A_6$. 
And here again is the irreducible, but not absolutely irreducible representations of $A_6$ over $GF(2)$.
Two matrix representations of a given group are equivalent, if, by changing the basis of the underlying space, you can go from one to the other. When you change bases, you transform the matrices that are the images of elements by conjugating them by an invertible matrix.

If we can find a subspace which is fixed under the image of the group, then there exists a 'base change' after which all the representing matrices are in upper triangular block form. The block matrices on the main diagonal give a new representation of the group of lower degree. Such a representation is said to be 'reducible'.
As a sample use of Axiom's algebraic number facilities, we compute the Galois group of the polynomial \( p(x) = x^5 - 5x + 12 \)

We would like to construct a polynomial \( f(x) \) such that the splitting field of \( p(x) \) is generated by one root of \( f(x) \). First we construct a polynomial \( r(x) \) such that one root of \( r(x) \) generates the field generated by two roots of the polynomial \( p(x) \). As it will turn out, the field generated by two roots of \( p(x) \) is, in fact, the splitting field of \( p(x) \).

From the proof of the primitive element theorem we know that if \( a \) and \( b \) are algebraic numbers, then the field \( \mathbb{Q}(a,b) \) is equal to \( \mathbb{Q}(a+k*b) \) for an appropriately chosen integer \( k \). In our case, we construct the minimal polynomial of \( a[i] - a[j] \), where \( a[i] \) and \( a[j] \) are two roots of \( p(x) \). We construct this polynomial using \(<a href="dbopresultant.xhtml">resultant</a>\). The main result we need is that if \( f(x) \) is a polynomial with roots \( a[1]...a[m] \) and \( g(x) \) is a polynomial with roots \( b[1]...b[n] \), then the polynomial \( h(x) = \text{resultant}(f(y), g(x-y), y) \) is a polynomial of degree \( mn \) with roots \( a[i]+b[j], 1 \leq i \leq m, 1 \leq j \leq n \).

For \( f(x) \) we use the polynomial \( p(x) \). For \( g(x) \) we use the polynomial \(-p(-x)\). Thus, the polynomial we first construct is \( \text{resultant}(p(y), -p(y-x), y) \).

The roots of \( q(x) \) are \( a[i] - a[j], 1 \leq i,j \leq 5 \). Of course, there are five pairs \((i,j)\) with \( i=j \), so 0 is a 5-fold root of \( q(x) \). Let's get rid of this factor.

Factor the polynomial \( q_1 \).
We see that $q_1$ has two irreducible factors, each of degree 10. (The fact that the polynomial $q_1$ has two factors of degree 10 is enough to show that the Galois group of $p(x)$ is the dihedral group of order 10. (Ref: McKay, Soicher, Computing Galois Groups over the Rationals, Journal of Number Theory 20, 273-281 (1983). We do not assume the results of this paper and we continue with the computation.) Note that the type of factoredQ is

Factored Polynomial Integer</a>. This is a special data type for recording factorizations of polynomials with integer coefficients (see Factored). We can access the individual factors using the operation nthFactor.</div>
</li>
</ul>

Consider the polynomial $r=r(x)$. This is the minimal polynomial of the difference of two roots of $p(x)$. Thus, the splitting field of $p(x)$ contains a subfield of degree 10. We show that this subfield is the splitting field of $p(x)$ by showing that $p(x)$ factors completely over this field. First we create a symbolic root of the polynomial $r(x)$. (We replaced $x$ by $b$ in the polynomial $r$ so that our symbolic root would be printed as $b$.)

We next tell Axiom to view $p(x)$ as a univariate polynomial in $x$ with algebraic number coefficients. This is accomplished with this type declaration:

Factor $p(x)$ over the field $\mathbb{Q}(\beta)$. (This computation will take some time).
When factoring over number fields, it is important to specify the field over which the polynomial is to be factored, as polynomials have different factorizations over different fields. When you use the operation \(<a href="dbopfactor.xhtml">factor</a>\), the field over which the polynomial is factored is the field generated by

- the algebraic numbers that appear in the coefficients of the polynomial
- the algebraic numbers that appear in a list passed as an optional second argument of the operation

In our case, the coefficients of \(p\) are all rational numbers and only \(\beta\) appears in the list, so the field is simply \(\mathbb{Q}(\beta)\). It was necessary to give the list \([\beta]\) as a second argument of the operations because otherwise the polynomial would have to be factored over the field generated by its coefficients, namely the rational numbers.

We have shown that the splitting field of \(p(x)\) has degree 10. Since the symmetric group of degree 5 has only one transitive subgroup of order 10, we know that the Galois group of \(p(x)\) must be this group, the dihedral group of order 10. Rather than stop here, we explicitly compute the action of the Galois group on the roots of \(p(x)\).

First we assign the roots of \(p(x)\) as the values of five variables. We can obtain an individual root by negating the constant coefficient of one of the factors of \(p(x)\).
We can obtain a list of all of the roots in this way.

\[
\text{roots} := [-\text{coefficient}(\text{nthFactor(algFactors,i)},0) \text{ for } i \text{ in } 1..5]
\]

The expression \(-\text{coefficient}(\text{nthFactor(algFactors,i)},0)\) is the \(i\)th root of \(p(x)\) and the elements of \(\text{roots}\) are the \(i\)th roots of \(p(x)\) as \(i\) ranges from 1 to 5. Assign the roots as the values of the variables \(a_1..a_5\).

Next we express the roots of \(r(x)\) as polynomials in \(\beta\). We could obtain these roots by calling the operation \(<a href="dbopfactor.xhtml">factor</a>\). \(\text{factor}(r,[\beta])\) factors \(r(x)\) over \(\mathbb{Q}(\beta)\). However, this is a length computation and we can obtain the roots of \(r(x)\) as differences of the roots \(a_1,...,a_5\) of \(p(x)\). Only ten of these differences are roots of \(r(x)\) and the other ten are roots of the other irreducible factor of \(q_1\). We can determine if a given value is a root of \(r(x)\) by evaluating \(r(x)\) at that particular value. (Of course, the order in which factors are returned by the operation \(<a href="dbopfactor.xhtml">factor</a>\) is unimportant and may change with different implementations of the operation. Therefore, we cannot predict in advance which differences are roots of \(r(x)\) and which are not.) Let's look at four examples (two are roots of \(r(x)\) and two are not).
Take one of the differences that was a root of $r(x)$ and assign it to the variable $bb$. For example, if $\text{eval}(r,x,a1-a4)$ returned 0, you would enter this.

Of course, if the difference is equal to the root $\beta$, you should choose another root of $r(x)$.

Automorphisms of the splitting field are given by mapping a generator of the field, namely $\beta$, to other roots of its minimal polynomial. Let’s see what happens when $\beta$ is mapped to $bb$. We compute the images of the roots $a1,\ldots,a5$ under this automorphism.
Of course, the values $aa_1, \ldots, aa_5$ are simply a permutation of the values $a_1, \ldots, a_5$. Let’s find the value of $aa_1$ (execute as many of the following five commands as necessary).

Proceeding in this fashion, you can find the values of $aa_2 \ldots aa_5$.

You have represented the automorphism $\beta \rightarrow bb$ as a permutation of the roots $a_1, \ldots, a_5$. If you wish, you can repeat this computation for all the roots of $r(x)$ and represent the Galois group of $p(x)$ as a subgroup of the
symmetric group on five letters.

Here are two other problems that you may attack in a similar fashion:

<ol>
<li>Show that the Galois group of \( p(x)=x^4+2x^3-2x^2-2x+1 \) is the dihedral group of order eight. (The splitting field of this polynomial is the Hilbert class field of the quadratic field \( \mathbb{Q}(\sqrt{145}) \).)
</li>
<li>Show that the Galois group of \( p(x)=x^6+108 \) has order 6 and is isomorphic to the symmetric group on three letters. (The splitting field of this polynomial is the splitting field of \( x^3-2 \).)
</li>
</ol>
What kind of limit do you want to compute?:

- A real limit
  - The limit as the variable approaches a real value along the real axis.

- A complex limit
  - The limit as the variable approaches a complex value along any path in the complex plane.
Simplification
<ul>
<li>Simplify Expressions</li>
<li>Simplify Radicals</li>
<li>Factor Expressions</li>
<li>Factor Complex</li>
<li>Expand Expressions</li>
<li>Expand Logarithms</li>
<li>Contract Logarithms</li>
<li>Simplify Trigonometrics</li>
<li>Reduce Trigonometrics</li>
<li>Expand Trigonometrics</li>
<li>Canonical Trigonometrics</li>
<li>Complex to rectangular</li>
<li>Complex to polar</li>
<li>Complex to exponentials</li>
<li>Exponentials to complex</li>
</ul>

Calculus
<ul>
<li>Integrate</li>
<li>Risch Integrate</li>
<li>Change Variable</li>
<li>Differentiate</li>
<li>Find Limit</li>
<li>Get Series</li>
<li>Pade Approximation</li>
<li>Calculate Sum</li>
<li>Calculate Product</li>
<li>Laplace Transform</li>
<li>Inverse Laplace Transform</li>
<li>Greatest Common Divisor</li>
<li>Least Common Multiple</li>
<li>Divide Polynomials</li>
<li>Partial Fractions</li>
<li>Continued Fractions</li>
</ul>

Algebra
<ul>
<li>Generate Matrix</li>
<li>Enter Matrix</li>
<li>Invert Matrix</li>
<li>Characteristic Polynomial</li>
<li>Determinant</li>
<li>Eigenvalues</li>
<li>Eigenvectors</li>
<li>Adjoint Matrix</li>
<li>Transpose Matrix</li>
</ul>

Equations
<ul>
<li>Solve</li>
<li>Solve Numerically</li>
</ul>
CHAPTER 1. OVERVIEW

<ul>
<li>Roots of Polynomials</li>
<li>Real Roots of Polynomials</li>
<li>Solve Linear Systems</li>
<li>Solve Algebraic System</li>
<li>Eliminate Variable</li>
</ul>

Ordinary Differential Equations
<ul>
<li>Solve ODE</li>
<li>Solve Initial Value Problem</li>
<li>Solve Boundary Value Problem</li>
<li>Solve ODE with Laplace</li>
</ul>

Data Structures
<ul>
<li>Record</li>
<li>List</li>
<li>Set</li>
</ul>

bcmatrix.xhtml

— bcmatrix.xhtml —

\getchunk{standard head}

```javascript
function byformula() {
    // find out how many rows and columns, must be positive and nonzero
    var rcnt = parseInt(document.getElementById('rowcnt').value);
    if (rcnt <= 0) {
        alert("Rows must be positive and non-zero -- defaulting to 1");
        rcnt = 1;
        document.getElementById('rowcnt').value=1;
        return(false);
    }
    var ccnt = parseInt(document.getElementById('colcnt').value);
    if (ccnt <= 0) {
        alert("Columns must be positive and non-zero -- defaulting to 1");
        ccnt = 1;
        document.getElementById('colcnt').value=1;
        return(false);
    }
    // remove the question and the buttons
    var quest = document.getElementById('question');
    var clicks = document.getElementById('clicks');
    quest.removeChild(clicks);
    var tbl = document.getElementById('form2');
    var tblsize = tbl.rows.length;
    // make the row variable question
```
// row variable left cell
var row = tbl.insertRow(tblsize);
var cell = row.insertCell(0);
var tnode = document.createTextNode("Enter the row variable");
cell.appendChild(tnode);
// row variable right cell
cell = row.insertCell(1);
tnode = document.createElement('input');
tnode.type = 'text';
tnode.name = 'rowvar';
tnode.id = 'rowvar';
tnode.size=10;
tnode.value='i';
tnode.tabindex=21;
cell.appendChild(tnode);

// make the column variable question
// column variable left cell
tblsize = tblsize + 1;
row = tbl.insertRow(tblsize);
cell = row.insertCell(0);
tnode = document.createTextNode("Enter the column variable");
cell.appendChild(tnode);
// column variable right cell
cell = row.insertCell(1);
tnode = document.createElement('input');
tnode.type = 'text';
tnode.name = 'colvar';
tnode.id = 'colvar';
tnode.size=10;
tnode.value='j';
tnode.tabindex=22;
cell.appendChild(tnode);

// make the formula question
// column variable left cell
tblsize = tblsize + 1;
row = tbl.insertRow(tblsize);
cell = row.insertCell(0);
tnode = document.createTextNode("Enter the formulas for the elements");
cell.appendChild(tnode);
// formula input field
tblsize = tblsize + 1;
row = tbl.insertRow(tblsize);
cell = row.insertCell(0);
tnode = document.createElement('input');
tnode.type = 'text';
tnode.name = 'formula1';
tnode.id = 'formula1';
tnode.size=50;
tnode.value = '1/(x-i-j-1)';
tnode.tabindex=23;
cell.appendChild(tnode);
// insert the continue button
tblsize = tblsize + 1;
row = tbl.insertRow(tblsize);
cell = row.insertCell(0);
tnode = document.createElement('input');
tnode.type = 'button';
tnode.id = 'contbutton';
tnode.value = 'Continue';
tnode.setAttribute("onclick","makeRequest('formula');");
tnode.tabindex=24;
cell.appendChild(tnode);
return(false);
}
function byelement() {
  // find out how many rows and columns, must be positive and nonzero
  var rcnt = parseInt(document.getElementById('rowcnt').value);
  if (rcnt <= 0) {
    alert("Rows must be positive and non-zero -- defaulting to 1");
    rcnt = 1;
    document.getElementById('rowcnt').value=1;
    return(false);
  }
  var ccnt = parseInt(document.getElementById('colcnt').value);
  if (ccnt <= 0) {
    alert("Columns must be positive and non-zero -- defaulting to 1");
    ccnt = 1;
    document.getElementById('colcnt').value=1;
    return(false);
  }
  // remove the question and the buttons
  var quest = document.getElementById('question');
  var clicks = document.getElementById('clicks');
  quest.removeChild(clicks);
  // write "Elements"
  var tbl = document.getElementById('form2');
  var tblsize = tbl.rows.length;
  var row = tbl.insertRow(tblsize);
  var thecell = row.insertCell(0);
  var tnode = document.createTextNode("Elements");
  thecell.appendChild(tnode);
  // create input boxes for the matrix values
  tblsize = tblsize + 1;
  for (var i = 0 ; i < rcnt ; i++) {
    row = tbl.insertRow(tblsize);
    for (var j = 0 ; j < ccnt ; j++) {
      thecell = row.insertCell(j);
      tnode = document.createElement('input');
      tnode.type = 'text';
      tnode.name = 'a'+i+'c'+j;
      tnode.id = 'a'+i+'c'+j;
      tnode.size=10;
      tnode.tabindex=20+(i*10)+j;
      thecell.appendChild(tnode);
    }
    tblsize = tblsize + 1;
  }
  // insert the continue button
function commandline(arg) {
  if (arg == 'element') {
    var rcnt = parseInt(document.getElementById('rowcnt').value);
    var ccnt = parseInt(document.getElementById('colcnt').value);
    var cmdhead = 'matrix([[';
    var cmdtail = ']);
    for (var i = 0 ; i < rcnt ; i++) {
      var listbody = '[[
      for (var j = 0 ; j < ccnt ; j++) {
        var aij = document.getElementById('a'+i+'c'+j).value;
        listbody = listbody+aij;
        if (j != (ccnt - 1)) {
          listbody = listbody+',';
        }
      }
      listbody = listbody+']';
      if (i != (rcnt - 1)) {
        listbody = listbody+',';
      }
    cmdhead = cmdhead+listbody;
  }
  cmd = cmdhead+cmdtail;
  return(cmd);
} else {
  var rcnt = parseInt(document.getElementById('rowcnt').value);
  var ccnt = parseInt(document.getElementById('colcnt').value);
  var cmdhead = 'matrix([[';
  var cmdtail = ']);
  var formula = document.getElementById('formula1').value;
  var rowv = document.getElementById('rowvar').value;
  var colv = document.getElementById('colvar').value;
  var cmd = cmdhead+formula+' for '+colv+' in 1..'+ccnt+cmdtail;
  for '+rowv+' in 1..'+rcnt+cmdtail;
  return(cmd);
}
}
CHAPTER 1. OVERVIEW

<table id="form2">
  <tr>
    <td size="10">Rows</td>
    <td><input type="text" id="rowcnt" tabindex="10" size="10" value="2"/></td>
  </tr>
  <tr>
    <td>Columns</td>
    <td><input type="text" id="colcnt" tabindex="20" size="10" value="3"/></td>
  </tr>
</table>

<div id="question">
  <div id="clicks">
    How would you like to enter the matrix elements?
    <center>
      <input type="button" value="By Formula" onclick="byformula();"/>
      <input type="button" value="By Element" onclick="byelement();"/>
    </center>
  </div>
</div>

— calculus.xhtml —

— calculus.xhtml —
<table>
<thead>
<tr>
<th>Calculator Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do a summation</td>
<td></td>
</tr>
<tr>
<td>Compute a product</td>
<td></td>
</tr>
<tr>
<td>Limits</td>
<td>Compute limits of functional expressions</td>
</tr>
<tr>
<td>Derivatives</td>
<td>Compute derivatives and partial derivatives</td>
</tr>
<tr>
<td>Integrals</td>
<td>Introduction to Axiom’s symbolic integration</td>
</tr>
<tr>
<td>More Integrals</td>
<td>More information about symbolic integration</td>
</tr>
<tr>
<td>Table Entry</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Laplace</td>
<td>Computing Laplace transforms</td>
</tr>
<tr>
<td>Series</td>
<td>Compute series expansions of expressions</td>
</tr>
<tr>
<td>Differential Equations</td>
<td>Solve differential equations</td>
</tr>
</tbody>
</table>

**calderivatives.xhtml**

Use the Axiom function $\text{D}$ to differentiate an expression.

To find the derivative of an expression $f$ with respect to a variable $x$, enter $\text{D}(f,x)$.

```xml
<input type="submit" id="p1" class="subbut"
onclick="makeRequest('p1');"
value="f:=\exp \exp x" />
```
An optional third argument n in \( D(f,x) \) asks Axiom for the nth derivative of \( f \). This finds the fourth derivative of \( f \) with respect to \( x \).

You can also compute partial derivatives by specifying the order of differentiation.

Axiom can manipulate the derivatives (partial or iterated) of expressions involving formal operators. All the dependencies must be explicit. This returns 0 since \( F \) (so far) does not explicitly depend on \( x \).

Suppose that we have \( F \) a function of \( x, y, \) and \( z \), where \( x \) and \( y \) are
themselves functions of $z$. Start by declaring that $F$, $x$, and $y$ are operators.

You can use $F$, $x$, and $y$ in expressions.

Differentiate formally with respect to $z$. The formal derivatives appearing in $\frac{\text{d}a}{\text{d}z}$ are not just formal symbols, but do represent derivatives of $x$, $y$, and $F$.

You can evaluate the above for particular functional values of $F$, $x$, and $y$. If $x(z)$ is $\exp(z)$ and $y(z)$ is $\log(z+1)$, then this evaluates $\frac{\text{d}a}{\text{d}z}$.

You obtain the same result by first evaluating $a$ and then differentiating.
Axiom has extensive library facilities for integration. The first example is the integration of a fraction with a denominator that factors into a quadratic and a quartic irreducible polynomial. The usual partial fraction approach used by most other computer algebra systems either fails or introduces expensive unneeded algebraic numbers.

We use a factorization-free algorithm.

When real parameters are present, the form of the integral can depend on the signs of some expressions.

Rather than query the user or make sign assumptions, Axiom returns all possible answers.

The \texttt{integrate} operation generally assumes that all parameters are real. The only exception is when the integrand has complex valued quantities.
If the parameter is complex instead of real, then the notion of sign is undefined and there is a unique answer. You can request this answer by "prepending" the word "complex" to the command name.

The following two examples illustrate the limitations of table-based approaches. The two integrands are very similar, but the answer to one of them requires the addition of two new algebraic numbers.

This is the easy one. The next one looks very similar but the answer is much more complicated.

Only an algorithmic approach is guaranteed to find what new constants must be added in order to find a solution.

Some computer algebra systems use heuristics or table-driven approaches to integration. When these systems cannot determine the answer to an integration problem, they reply "I don't know". Axiom uses an algorithm for integration that conclusively proves that an integral cannot be expressed in terms of elementary functions.

When Axiom returns an integral sign, it has proved that no answer exists as an elementary function.

Axiom can handle complicated mixed functions much beyond what you can find in tables. Whenever possible, Axiom tries to express the answer using the
functions present in the integrand.

A strong structure-checking algorithm in Axiom finds hidden algebraic relationships between functions.

The discovery of this algebraic relationship is necessary for correct integration of this function. Here are the details:

If $x=\tan(t)$ and $g=\tan(t/3)$ then the following algebraic relationship is true:

\[
g^3 - 3xg^2 - 3g + x = 0
\]

Integrate $g$ using this algebraic relation; this produces:

\[
((24g^2-8)\log(3g^2-1) + (81x^2+24)g^2 + 72xg - 27x^2 - 16) / (54g^2 - 18)
\]

Rationalize the denominator, producing:

\[
(8\log(3g^2-1) - 3g^2 + 18xg + 16)/18
\]

Replace $g$ by the initial definition $g=\tan(\arctan(x)/3)$ to produce the final result.

This is an example of a mixed function where the algebraic layer is over the transcendental one.

While incomplete for non-elementary functions, Axiom can handle some of them.
CHAPTER 1. OVERVIEW

More examples of Axiom's integration capabilities are discussed in <a href="axbook/section-8.8.xhtml">Integration</a>.

callaplace.xhtml

--- callaplace.xhtml ---

Axiom can compute some forward Laplace transforms, mostly of elementary functions not involving logarithms, although some cases of special functions are handled. To compute the forward Laplace transform of F(t) with respect to t and express the result as f(s), issue the command laplace(F(t),t,s).

Here are some other non-trivial examples.

---
Axiom also knows about a few special functions.

When Axiom does not know about a particular transform, it keeps it as a formal transform in the answer.
To compute a limit, you must specify a functional expression, a variable, and a limiting value for that variable. If you do not specify a direction, Axiom attempts to compute a two-sided limit.

Issue this to compute the limit of \((x^2-2x+2)/(x^2-1)\) as \(x\) approaches 1.

Sometimes the limit when approached from the left is different from the limit from the right, and, in this case, you may wish to ask for a one-sided limit. Also, if you have a function that is only defined on one side of a particular value, you can compute a one-sided limit.

The function \(\log(x)\) is only defined to the right of zero, that is, for \(x>0\). Thus, when computing limits of functions involving \(\log(x)\), you probably want a "right-hand" limit.

A function can be defined on both sides of a particular value, but tend to different limits as its variable approaches that value from the left and from the right. We can construct an example of this as follows: Since \(\sqrt{y^2}\) is simply the absolute value of \(y\), the function \(\sqrt{y^2}/y\) is simply the sign (+1 or -1) of the nonzero real number \(y\). Therefore, \(\sqrt{y^2}/y=-1\) for \(y<0\) and \(\sqrt{y^2}/y=+1\) for \(y>0\). This is what happens when we take the limit at \(y=0\). The answer returned by Axiom gives both a "left-handed" and a "right-handed" limit.
Here is another example, this time using a more complicated function.

You can compute limits at infinity by passing either "plus infinity" or "minus infinity" as the third argument of \(<a href="dboplimit.xhtml">\text{limit}</a>\). To do this, use the constants %plusInfinity and %minusInfinity.

You can take limits of functions with parameters. As you can see, the limit is expressed in terms of the parameters.

When you use \(<a href="dboplimit.xhtml">\text{limit}</a>\), you are taking the limit of a real function of a real variable. When you compute this, Axiom returns 0 because, as a function of a real variable, \(\sin(1/z)\) is always between -1 and 1, so \(z\cdot\sin(1/z)\) tends to 0 as \(z\) tends to 0.
However, as a function of a complex variable, $\sin(1/z)$ is badly behaved near 0 (one says that $\sin(1/z)$ has an essential singularity at $z=0$). When viewed as a function of a complex variable, $z\sin(1/z)$ does not approach any limit as $z$ tends to 0 in the complex plane. Axiom indicates this when we call `<a href="dbopcomplexlimit.xhtml">complexLimit</a>`.

You can also take complex limits at infinity, that is, limits of a function of $z$ as $z$ approaches infinity on the Riemann sphere. Use the symbol `%infinity` to denote "complex infinity". As above, to compute complex limits rather than real limits, use `<a href="dbopcomplexlimit.xhtml">complexLimit</a>`.

In many cases, a limit of a real function of a real variable exists when the corresponding complex limit does not. This limit exists.

But this limit does not.
Integration is the reverse process of differentiation, that is, an integral of a function $f$ with respect to a variable $x$ is any function $g$ such that $D(g,x)$ is equal to $f$. The package <a href="db.xhtml?FunctionSpaceIntegration">FunctionSpaceIntegration</a> provides the top-level integration operation <a href="dbopintegrate.xhtml">integrate</a>, for integrating real-valued elementary functions.

Unfortunately, antiderivatives of most functions cannot be expressed in terms of elementary functions.

Given an elementary function to integrate, Axiom returns a formal integral as above only when it can prove that the integral is not elementary and not when it cannot determine the integral. In this rare case it prints a message that it cannot determine if an elementary integral exists. Similar functions may have antiderivatives that look quite different because the form of the antiderivative depends on the sign of a constant that appears in the function.
If the integrand contains parameters, then there may be several possible antiderivatives, depending on the signs of expressions of the parameters.

In this case Axiom returns a list of answers that cover all possible cases. Here you use the answer involving the square root of $a$ when $a>0$ and the answer involving the square root of $-a$ when $a<0$.

If the parameters and the variables of integration can be complex numbers rather than real, then the notion of sign is not defined. In this case all the possible answers can be expressed as one complex function. To get that function, rather than a list of real functions, use \texttt{complexIntegrate}, which is provided by the package \texttt{FunctionSpaceComplexIntegration}.

This operation is used for integrating complex-valued elementary functions.

As with the real case, antiderivatives for most complex-valued functions cannot be expressed in terms of elementary functions.

Sometimes \texttt{integrate} can involve symbolic algebraic numbers such as those returned by \texttt{rootOf}. To see how to work with these strange generated symbols (such as $\%a0$), see Using All Roots of a Polynomial.

Definite integration is the process of computing the area between the $x$-axis and the curve of a function $f(x)$. The fundamental theorem of calculus states that if $f$ is continuous on an interval $a..b$ and such that $D(g,x)$ is equal to $f$, then the definite integral of $f$ for $x$ in the interval $a..b$ is
equal to \( g(b) - g(a) \).

The package
\(<a href="db.xhtml?RationalFunctionDefiniteIntegration">RationalFunctionDefiniteIntegration</a>\)
provides the top-level definite integration operation,
\(<a href="dbopintegrate.xhtml">integrate</a>\),
for integrating real-valued rational functions.
\(<ul\>
\(<li\>
<input type="submit" id="p8" class="subbut"
  onclick="makeRequest('p8');"
  value="integrate((x^4-3*x^2+6)/(x^6-5*x^4+5*x^2+4),x=1..2)" />
<div id="ansp8"></div></div>
</li>
</ul>
Axiom checks beforehand that the function you are integrating is defined on
the interval \( a..b \), and prints an error message if it finds that this is not
the case, as in the following example:
\(<pre\>
integrate(1/(x^2-2),x=1..2)
\(<pre\>
Error detected within library code:
Pole in path of integration
</pre>
When parameters are present in the function, the function may or may not be
defined on the interval of integration.

If this is the case, Axiom issues a warning that a pole might lie in the
path of integration, and does not compute the integral.
\(<ul\>
\(<li\>
<input type="submit" id="p9" class="subbut"
  onclick="makeRequest('p9');"
  value="integrate(1/(x^2-a),x=1..2)" />
<div id="ansp9"></div></div>
</li>
</ul>
If you know that you are using values of the parameter for which the
function has no pole in the interval of integration, use the string
"noPole" as a third argument to <a href="dbopintegrate.xhtml">integrate</a>.

The value here is, of course, incorrect if \( \sqrt{a} \) is between 1 and 2.
\(<ul\>
\(<li\>
<input type="submit" id="p10" class="subbut"
  onclick="makeRequest('p10');"
  value='integrate(1/(x^2-a),x=1..2,"noPole")' />
<div id="ansp10"></div></div>
</li>
</ul>
Axiom has very sophisticated facilities for working with power series. Infinite series are represented by a list of the coefficients that have already been determined, together with a function for computing the additional coefficients if needed. The system command that determines how many terms of a series is displayed is

```
)set streams calculate
```

By default Axiom will display ten terms. Series can be created from expressions, from functions for the series coefficients, and from applications of operations on existing series. The most general function for creating a series is called `<a href="dbopseries.xhtml">series</a>`, although you can also use `<a href="dboptaylor.xhtml">taylor</a>`, `<a href="dboplaurent.xhtml">laurent</a>`, and `<a href="dboppuiseux.xhtml">puiseux</a>` in situations where you know what kind of exponents are involved.

For information about solving differential equations in terms of power series see `<a href="axbook/section-8.10.xhtml#subsec-8.10.3">Power Series Solutions of Differential Equations</a>`

<ul>
  <li><a href="calseries1.xhtml">Creation of Power Series</a></li>
  <li><a href="calseries2.xhtml">Coefficients of Power Series</a></li>
  <li><a href="calseries3.xhtml">Power Series Arithmetic</a></li>
  <li><a href="calseries4.xhtml">Functions on Power Series</a></li>
</ul>
This is the easiest way to create a power series. This tells Axiom that \( x \) is to be treated as a power series, so functions of \( x \) are again power series.

We didn't say anything about the coefficients of the power series, so the coefficients are general expressions over the integers. This allows us to introduce denominators, symbolic constants, and other variables as needed. Here the coefficients are integers (note that the coefficients are the
Fibonacci numbers).

This series has coefficients that are rational numbers.

When you enter this expression you introduce the symbolic constants \( \sin(1) \) and \( \cos(1) \).

When you enter the expression the variable \( a \) appears in the resulting series expansion.

You can also convert an expression into a series expansion. This expression creates the series expansion of \( 1/\log(v) \) about \( v=1 \). For details and more examples see Converting to Power Series.

You can create power series with more general coefficients. You normally accomplish this via a type declaration, see Declarations.
Functions on Power Series for some warnings about working with declared series.

We declare that \( y \) is a one-variable Taylor series (\( \text{UTS} \) is the abbreviation for \( \text{UnivariateTaylorSeries} \) in the variable \( z \) with \( \text{FLOAT} \) (that is, floating-point) coefficients, centered about 0. Then, by assignment, we obtain the Taylor expansion of \( \exp(z) \) with floating-point coefficients.

\[
\begin{array}{l}
\text{\#7} \quad y: \text{UTS}('z,0):= \exp(z) \\
\end{array}
\]

You can also create a power series by giving an explicit formula for the \( n \)th coefficient. For details and more examples see Power Series from Formulas.

To create a series about \( w=0 \) whose \( n \)th Taylor coefficient is \( 1/n! \), you can evaluate this expression. This is the Taylor expansion of \( \exp(w) \) at \( w=0 \).

\[
\begin{array}{l}
\text{\#8} \quad \text{series}(1/\text{factorial}(n),n,w=0) \\
\end{array}
\]
series are represented by a list of the coefficients that have already
been determined, together with a function for computing additional
coefficients. (This is known as lazy evaluation.) When you ask for a
coefficient that hasn’t yet been computed, Axiom computes whatever
additional coefficients it needs and then stores them in the representation
of the power series.

Here’s an example of how to extract the coefficients of a power series.

<ul>
<li>
  <input type="submit" id="p1" class="subbut"
       onclick="makeRequest('p1');"
       value="x:=series('x)" />
  <div id="ansp1"><div></div></div>
</li>
<li>
  <input type="submit" id="p2" class="subbut"
       onclick="handleFree(['p1','p2']);"
       value="y:=exp(x)*sin(x)" />
  <div id="ansp2"><div></div></div>
</li>
</ul>

This coefficient is readily available

<ul>
<li>
  <input type="submit" id="p3" class="subbut"
       onclick="handleFree(['p1','p2','p3']);"
       value="coefficient(y,6)" />
  <div id="ansp3"><div></div></div>
</li>
</ul>

But let’s get the fifteenth coefficient of y

<ul>
<li>
  <input type="submit" id="p4" class="subbut"
       onclick="handleFree(['p1','p2','p4']);"
       value="coefficient(y,15)" />
  <div id="ansp4"><div></div></div>
</li>
</ul>

If you look at y then you see that the coefficients up to order 15 have
all been computed.

<ul>
<li>
  <input type="submit" id="p5" class="subbut"
       onclick="handleFree(['p1','p2','p4','p5']);"
       value="/" />
  <div id="ansp5"><div></div></div>
</li>
</ul>

\getchunk{page foot}
You can manipulate power series using the usual arithmetic operations $\pm, \times, \div$. The results of these operations are also power series.

You can also compute $f(x)^g(x)$, where $f(x)$ and $g(x)$ are two power series.
Once you have created a power series, you can apply transcendental functions (for example, \(\exp\), \(\log\), \(\sin\), \(\tan\), \(\cosh\), etc.) to it.

To demonstrate this, we first create the power series expansion of the rational function \(x^2/(1-6x+x^2)\) about \(x=0\).

If you want to compute the series expansion of \(\sin(x^2/(1-6x+x^2))\) you simply compute the sine of \(rat\).
Warning: the type of the coefficients of a power series may affect the kind of computations that you can do with that series. This can only happen when you have made a declaration to specify a series domain with a certain type of coefficient.

If you evaluate \( y \) then you have declared that \( y \) is a one variable Taylor series (\(<a href="db.xhtml?UnivariateTaylorSeries">UTS</a>\) is the abbreviation for \(<a href="db.xhtml?UnivariateTaylorSeries">UnivariateTaylorSeries</a>\) in the variable \( y \) with \(<a href="dbfractioninteger.xhtml">FRAC INT</a>\) (that is, fractions of integers) coefficients, centered about 0.

You can now compute certain power series in \( y \), provided that these series have rational coefficients.

You can get examples of such series by applying transcendental functions to series in \( y \) that have no constant terms.

Similarly, you can compute the logarithm of a power series with rational coefficients if the constant coefficient is 1.
If you wanted to apply, say, the operation \( \exp \) to a power series with a nonzero constant coefficient \( a_0 \), then the constant coefficient of the result would be \( \exp(a_0) \), which is not a rational number. Therefore, evaluating \( \exp(2+\tan(y)) \) would generate an error message.

If you want to compute the Taylor expansion of \( \exp(2+\tan(y)) \), you must ensure that the coefficient domain has an operation \( \exp \) defined for it. An example of such a domain is the type of formal functional expressions over the integers. When working with coefficients of this type

\[
\begin{align*}
\text{another way to create Taylor series whose coefficients are expressions over the integers is to use } & \text{ which works similarly to } \text{ series. This is equivalent to the previous computation, except that now we are using the variable } w \text{ instead of } z. \\
\end{align*}
\]

Another way to create Taylor series whose coefficients are expressions over the integers is to use \( \text{ series. This is equivalent to the previous computation, except that now we are using the variable } w \text{ instead of } z. \\
\end{align*}
\]

\[
\begin{align*}
\text{another way to create Taylor series whose coefficients are expressions over the integers is to use } & \text{ which works similarly to } \text{ series. This is equivalent to the previous computation, except that now we are using the variable } w \text{ instead of } z. \\
\end{align*}
\]
Converting to Power Series

The `ExpressionToUnivariatePowerSeries` package provides operations for computing series expansions of functions.

Evaluate this to compute the Taylor expansion of \( \sin x \) about \( x=0 \). The first argument, \( \sin(x) \), specifies the function whose series expansion is to be computed and the second argument, \( x=0 \), specifies that the series is to be expanded in powers of \( (x-0) \), that is, in powers of \( x \).

Here is the Taylor expansion of \( \sin x \) about \( x=\frac{\pi}{6} \):

The function to be expanded into a series may have variables other than the series variable. For example, we may expand \( \tan(x+y) \) as a Taylor series in \( x \).
A more interesting function is \((t^x e^{-x \cdot t})/(e^t - 1)\).
When we expand this function as a Taylor series in \(t\) the \(n\)th order
coefficient is the \(n\)th Bernoulli polynomial divided by \(n!\).

Therefore, this and the next expression produce the same result.

Technically, a series with terms of negative degree is not considered to
be a Taylor series, but rather a Laurent series. If you try to compute a
Taylor series expansion of \(x/\log(x)\) at \(x=1\) via \(\text{taylor}(x/\log(x), x=1)\) you
get an error message. The reason is that the function has a pole at \(x=1\),
meaning that its series expansion about this point has terms of negative
degree. A series with finitely many terms of negative degree is called a
Laurent series. You get the desired series expansion by issuing this.

Similarly, a series with terms of fractional degree is neither a Taylor
series nor a Laurent series. Such a series is called a Puiseux series. The
expression \(\text{laurent}(\sqrt{\sec(x)}, x=3\pi/2)\) results in an error message
because the series expansion about this point has terms of fractional degree.
However, this command produces what you want.
Finally, consider the case of functions that do not have Puiseux expansions about certain points. An example of this is $x^x$ about $x=0$. Puiseux($x^x$, $x=0$) produces an error message because of the type of singularity of the function at $x=0$. The general function <a href="dbopseries.xhtml">series</a> can be used in this case. Notice that the series returned is not, strictly speaking, a power series because of the $\log(x)$ in the expansion.

The operation <a href="dbopseries.xhtml">series</a> returns the most general type of infinite series. The user who is not interested in distinguishing between various types of infinite series may wish to use this operation exclusively.

The <a href="db.xhtml?GenerateUnivariatePowerSeries">GenerateUnivariatePowerSeries</a> package enables you to create power series from explicit formulas for their nth coefficients. In what follows, we construct series expansions for certain transcendental functions by giving formulas for their coefficients. You can also compute such series expansions directly by simply specifying the function and the point about which the series is to be expanded. See <a href="axbook/section-8.9.xhtml#subsec-8.9.5">Converting to Power Series</a> for more information.
Consider the Taylor expansion of $\%e^x$ about $x=0$:

\[\%e^x = 1 + x + x^2/2 + x^3/6 + \ldots\]
\[= \text{sum from } n=0 \text{ to } n=\infty \text{ of } x^n/n!\]

The $n$th Taylor coefficient is $1/n!$. This is how to create this series in Axiom.

\begin{verbatim}
<input type="submit" id="p1" class="subbut"
    onclick="makeRequest('p1');"
    value="series(n+->1/factorial(n),x=0)" />
</li>
</ul>

The first argument specifies the formula for the $n$th coefficient by giving a function that maps $n$ to $1/n!$. The second argument specifies that the series is to be expanded in powers of $(x-0)$, that is, in powers of $x$. Since we did not specify an initial degree, the first term in the series was the term of degree 0 (the constant term). Note that the formula was given as an anonymous function. These are discussed in <a href="axbook/section-6.17.xhtml">Anonymous Functions</a>.

Consider the Taylor expansion of $\log x$ about $x=1$:

\[\log x = (x-1) - (x-1)^2/2 + (x-1)^3/3 - \ldots\]
\[= \text{sum from } n=1 \text{ to } n=\infty \text{ of } (-1)^{n-1} (x-1)^n/n\]

If you were to evaluate the expression $\text{series}(n+->(-1)^{(n-1)}/n,x=1)$ you would get an error message because Axiom would try to calculate a term of degree $n=1,\ldots$ are to be computed.

\begin{verbatim}
<input type="submit" id="p2" class="subbut"
    onclick="makeRequest('p2');"
    value="series(n+->(-1)^{(n-1)}/n,x=1,1..)" />
</li>
</ul>

Next consider the Taylor expansion of an odd function, say, $\sin(x)$:

\[\sin x = x - x^3/3! + x^5/5! - \ldots\]

Here every other coefficient is zero and we would like to give an explicit formula only for the odd Taylor coefficients. This is one way to do it. The third argument, $1..$, specifies that the first term to be computed is the term of degree 1. The fourth argument, 2, specifies that we increment by 2 to find the degrees of subsequent terms, that is, the next term is of degree $1+2$, the next of degree $1+2+2$, etc.

\begin{verbatim}
<input type="submit" id="p3" class="subbut"
    onclick="makeRequest('p3');"
    value="series(n+->(-1)^{(n-1)/2}/factorial(n),x=0,1..,2)" />
</li>
</ul>
The initial degree and the increment do not have to be integers. For example, this expression produces a series expansion of $\sin(x^{\frac{1}{3}})$.

While the increment must be positive, the initial degree may be negative. This yields the Laurent expansion of $\csc(x)$ at $x=0$.

Of course, the reciprocal of this power series is the Taylor expansion of $\sin(x)$.

As a final example, here is the Taylor expansion of $\arcsin(x)$ about $x=0$.

When we compute the sine of this series, we get $x$ (in the sense that all higher terms computed so far are zero).

As we discussed in
<a href="calseries5.xhtml">Converting to Power Series</a>, you can also use
the operations
<a href="dboptaylor.xhtml">taylor</a>,
<a href="dboplaurent.xhtml">laurent</a>, and
<a href="dboppuiseux.xhtml">puiseux</a>, instead of
<a href="dbopseries.xhtml">series</a> if you know ahead of time what kind of exponents a series has. You can’t go wrong with
<a href="dbopseries.xhtml">series</a> though.

---

calseries7.xhtml

— calseries7.xhtml —

Use <a href="dbopeval.xhtml">eval</a> to substitute a numerical value for a variable in a power series. For example, here’s a way to obtain numerical approximations of %e from the Taylor series expansion of exp(x).

First you create the desired Taylor expansion.

```html
<ul>
  <li>
    <input type="submit" id="p1" class="subbut"
      onclick="makeRequest('p1');"
      value="f:=taylor(exp(x))" />
    <div id="ansp1"><div></div></div>
  </li>
</ul>
```

Then you evaluate the series at the value 1.0. The result is a sequence of the partial sums.

```html
<ul>
  <li>
    <input type="submit" id="p2" class="subbut"
      onclick="handleFree(['p1','p2']);"
      value="eval(f,1.0)" />
    <div id="ansp2"><div></div></div>
  </li>
</ul>
```
Axiom provides operations for computing definite and indefinite sums.

You can compute the sum of the first ten fourth powers by evaluating this.
This creates a list whose entries are $m^4$ as $m$ ranges from 1 to 10, and then computes the sum of the entries of that list.

```plaintext
<ul>
  <li>
    <input type="submit" id="p1" class="subbut"
      onclick="makeRequest('p1');" value="reduce(+,[m^4 for m in 1..10])" />
    <div id="ansp1"><div></div></div>
  </li>
</ul>
```

You can also compute a formula for the sum of the first $k$ fourth powers, where $k$ is an unspecified positive integer.

```plaintext
<ul>
  <li>
    <input type="submit" id="p2" class="subbut"
      onclick="makeRequest('p2');" value="sum4:=sum(m^4,m=1..k)" />
    <div id="ansp2"><div></div></div>
  </li>
</ul>
```

This formula is valid for any positive integer $k$. For instance, if we replace $k$ by 10, we obtain the number we computed earlier.

```plaintext
<ul>
  <li>
    <input type="submit" id="p3" class="subbut"
      onclick="handleFree(['p2','p3']);" value="eval(sum4,k=10)" />
    <div id="ansp3"><div></div></div>
  </li>
</ul>
```

You can compute a formula for the sum of the first $k$ $n$th powers in a similar fashion. Just replace the 4 in the definition of `sum4` by any expression not involving $k$. Axiom computes these formulas using Bernoulli polynomials; we use the rest of this section to describe this method.

First consider this function of $t$ and $x$. 
Since the expressions involved get quite large, we tell Axiom to show us only terms of degree up to 5.

If we look at the Taylor expansion of $f(x,t)$ about $t=0$, we see that the coefficients of the powers of $t$ are polynomials in $x$.

In fact, the $n$th coefficient in this series is essentially the $n$th Bernoulli polynomial: the $n$th coefficient of the series is $1/n!B_n(x)$, where $B_n(x)$ is the $n$th Bernoulli polynomial. Thus, to obtain the $n$th Bernoulli polynomial, we multiply the $n$th coefficient of the series $ff$ by $n!$. For example, the sixth Bernoulli polynomial is this.

We derive some properties of the function $f(x,t)$. First we compute $f(x+1,t)-f(x-t)$.

If we normalize $g$, we see that it has a particularly simple form.
From this it follows that the nth coefficient in the Taylor expansion of \( g(x,t) \) at \( t=0 \) is \( \frac{1}{(n-1)!} x^{n-1} \). If you want to check this, evaluate the next expression.

However, since

\[
g(x,t) = f(x+1,t) - f(x,t)
\]

it follows that the nth coefficient is

\[
\frac{1}{n!} * (B_n(x+1) - B_n(x))
\]

Equating coefficients, we see that

\[
\frac{1}{(n-1)!} * x^{n-1} = \frac{1}{n!} * (B_n(x+1) - B_n(x))
\]

and, therefore

\[
x^{n-1} = \frac{1}{n} * (B_n(x+1) - B_n(x))
\]

Let's apply this formula repeatedly, letting \( x \) vary between two integers \( a \) and \( b \), with \( a \leq b \):

\[
a^{n-1} = \frac{1}{n} * (B_n(a+1) - B_n(a))
\]

\[
(a+1)^{n-1} = \frac{1}{n} * (B_n(a+2) - B_n(a+1))
\]

\[
(a+2)^{n-1} = \frac{1}{n} * (B_n(a+3) - B_n(a+2))
\]

...

\[
(b-1)^{n-1} = \frac{1}{n} * (B_n(b) - B_n(b-1))
\]

\[
b^{n-1} = \frac{1}{n} * (B_n(b+1) - B_n(b))
\]

When we add these equations we find that the sum of the left-hand sides is

\[
\sum_{m=a}^{b} m^{n-1}
\]

the sum of the \( (n-1) \)-st powers from \( a \) to \( b \). The sum of the right-hand sides is a "telescoping series". After cancellation, the sum is simply

\[
\frac{1}{n}*(B_n(b+1)-B_n(a))
\]
Replacing \( n \) by \( n+1 \), we have shown that

\[
\sum_{m=a}^{b} m^n = \frac{1}{n+1} \left( B_{n+1}(b+1) - B_{n+1}(a) \right)
\]

Let’s use this to obtain the formula for the sum of fourth powers. First we obtain the Bernoulli polynomial \( B_5 \).

\[
B_5 := \frac{\text{factorial}(5) \cdot \text{coefficient}(ff,5)}
\]

To find the sum of the first \( k \) 4th powers, we multiply \( \frac{1}{5} \) by

\[
\frac{1}{5} \left( \text{eval}(B_5, x=k+1) - \text{eval}(B_5, x=1) \right)
\]

This is the same formula that we obtained via \( \sum_{m=4}^{m=1..k} \)

At this point you may want to do the same computation, but with an exponent other than 4. For example, you might try to find a formula for the sum of the first \( k \) 20th powers.

---

cats.xhtml

---
cats.xhtml---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
<div align="center">
CATS -- Computer Algebra Test Suite
The Computer Algebra Test Suite is intended to show that Axiom conforms to various published standards. Axiom implementations of these functions are tested against reference publications.

In order to show standards compliance we need to examine Axiom’s behavior against known good results. Where possible, these results are also tested against other available computer algebra systems.

The available test suites are:
<ol>
    <li><a href="dlmf.xhtml">Gamma Function</a></li>
</ol>
c.title=title;
c.content=content;
c.type=type;
c.visible=visible;
return(c);
}

var pamphlet = '{is:"pamphlet","+
    'head:"",'+
    'body:[],'+
    'tail:"",'+
    'visible:"false"}';

function newpamphlet(title,content,type,visible) {
    var p = eval("("+pamphlet+"))");
    p.title=title;
    p.content=content;
    p.type=type;
    p.visible=visible;
    return(p);
}

function dump(d) {
    if (d.is == "card") {
        return("CARD"\n"title: "+d.title+"\n"content:"+d.content+"\n"visible:"+d.visible+"\n"type: "+d.type);
    }
    if (d.is == "pamphlet") {
        return("PAMPHLET"\n"title: "+d.title+"\n"content:"+d.content+"\n"visible:"+d.visible+"\n"type: "+d.type);
    }
}

function makeone() {
    var p = newpamphlet("makeone",card,"text","true");
    alert(dumppamphlet(p));
    alert(dump(p));
}
<!-- Begin File Menu -->
<li>
<a class="toplevel" href="/">
File</a>
</li>
<ul>
<li>
<a class="drop" href="javascript:popUp('menufileopen.xhtml')">
Open
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menufileread.xhtml')">
Read file
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menufilesave.xhtml')">
Save
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menufilesaveas.xhtml')">
Save as
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menufileloadlibrary.xhtml')">
Load library
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menufileinputfile.xhtml')">
Input file
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menufiletogglespool.xhtml')">
Toggle spool
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menufileprint.xhtml')">
Print
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menufileexit.xhtml')">
Exit
</a>
</li>
</ul>
<li>
<a class="drop" href="javascript:popUp('menuaxiomdeletefunction.xhtml')">
Delete function
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menuaxiomdeletevariable.xhtml')">
Delete variable
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menuaxiomtoggletimedisplay.xhtml')">
Toggle time display
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menuaxiomset.xhtml')">
Set ... 
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menuaxiomdisplay.xhtml')">
Display ...
</a>
</li>
</ul>

<!-- End Axiom Menu -->
<!-- Start Equations Menu -->

<li>
<a class="toplevel" href="/">
Equations</a>
</li>
<ul>
<li>
<a class="drop" href="javascript:popUp('menuequationssolve.xhtml')">
Solve
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menuequationssolvenumerically.xhtml')">
Solve numerically
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menuequationsrootsofpolynomial.xhtml')">
</a>
</li>
</ul>

<!-- End Equations Menu -->
Roots of polynomial
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menuequationsrealrootsofpolynomial.xhtml')">
Real roots of polynomial</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menuequationssolvealgebraicsystem.xhtml')">
Solve algebraic system</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menuequationseliminatevariable.xhtml')">
Eliminate variable</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menuequationssolveode.xhtml')">
Solve ODE</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menuequationsinitialvalueproblem1.xhtml')">
Initial value problem (1)</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menuequationsinitialvalueproblem2.xhtml')">
Initial value problem (2)</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menequationsboundaryvalueproblem.xhtml')">
Boundary value problem</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menuequationsolveodewithlaplace.xhtml')"></a>

</li>
CHAPTER 1. OVERVIEW

Solve ODE with Laplace
</a>
</li>
<li>
<a class="drop"
   href="javascript:popUp('menuequationsatvalue.xhtml')">
   At value
</a>
</li>
</ul>
</li>
</ul>
<!-- End Equations Menu -->
<!-- Start Algebra Menu -->
<li>
<a class="toplevel" href="/">
Algebra</a>
</li>
<li>
<a class="drop"
   href="javascript:popUp('menualgebrageneratematrix.xhtml')">
Generate matrix
</a>
</li>
<li>
<a class="drop"
   href="javascript:popUp('menualgebraentermatrix.xhtml')">
Enter matrix
</a>
</li>
<li>
<a class="drop"
   href="javascript:popUp('menualgebrainvertmatrix.xhtml')">
Invert matrix
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menualgebracharacteristicpolynomial.xhtml')">
Characteristic polynomial
</a>
</li>
<li>
<a class="drop"
   href="javascript:popUp('menualgebraeigenvalues.xhtml')">
Eigenvalues
</a>
</li>
CHAPTER 1. OVERVIEW

Calculus

<ul>
  <li>
    <a class="drop" href="javascript:popUp('menucalculuslevel3.xhtml')">Level 3</a>
  </li>
  <li>
    <a class="drop" href="javascript:popUp('menucalculuslevel3a.xhtml')">Level 3 A</a>
  </li>
  <li>
    <a class="drop" href="javascript:popUp('menucalculuslevel3b.xhtml')">Level 3 B</a>
  </li>
  <li>
    <a class="drop" href="javascript:popUp('menucalculuslevel3c.xhtml')">Level 3 C</a>
  </li>
  <li>
    <a class="drop" href="javascript:popUp('menucalculusintegrate.xhtml')">Integrate</a>
  </li>
  <li>
    <a class="drop" href="javascript:popUp('menucalculusrischintegrate.xhtml')">Risch integrate</a>
  </li>
  <li>
    <a class="drop" href="javascript:popUp('menucalculusschangevariable.xhtml')">Change variable</a>
  </li>
  <li>
    <a class="drop" href="javascript:popUp('menucalculussdifferentiate.xhtml')">Differentiate</a>
  </li>
</ul>
<a class="drop" href="javascript:popUp('menucalculusfindlimit.xhtml')">
  Find limit
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menucalculusgetseries.xhtml')">
  Get series
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menucalculuspadeapproximation.xhtml')">
  Pade approximation
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menucalculuscalculussum.xhtml')">
  Calculus sum
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menucalculuscalculusproduct.xhtml')">
  Calculus product
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menucalculuslaplacetransform.xhtml')">
  Laplace transform
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menucalculusinverselaplacetransform.xhtml')">
  Inverse Laplace transform
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menucalculusgreatestcommondivisor.xhtml')">
  Greatest common divisor
</a>
</li>
<li>
<a class="drop" href="javascript:popUp('menucalculusleastcommonmultiple.xhtml')">
  Least common multiple
</a>
</li>
<a class="drop" href="javascript:popUp('menusimplifyexpandlogarithms.xhtml')">Expand logarithms</a>
<a class="drop" href="javascript:popUp('menusimplifycontractlogarithms.xhtml')">Contract logarithms</a>
<a class="drop" href="javascript:popUp('menusimplifyfactorialsandgamma.xhtml')">Factorials and Gamma</a>
<a class="drop" href="javascript:popUp('menusimplifytrigsimplification.xhtml')">Trigonometric simplification</a>
<a class="drop" href="javascript:popUp('menusimplifycomplexsimplification.xhtml')">Complex simplification</a>
<a class="drop" href="javascript:popUp('menusimplifysubtitute.xhtml')">Substitute</a>
<a class="drop" href="javascript:popUp('menusimplifyevaluatenounform.xhtml')">Evaluate noun form</a>
<a class="drop" href="javascript:popUp('menusimplifytogglealgebraicflag.xhtml')">Toggle algebraic flag</a>
<a class="drop" href="javascript:popUp('menusimplifyaddalgebraicequality.xhtml')">Add algebraic equality</a>
CHAPTER 1. OVERVIEW

<a href="javascript:popUp('menusimplifymoduluscomputation.xhtml')">Modulus computation</a>

<a href="javascript:popUp('menunumerictogglenumericoutput.xhtml')">Toggle numeric output</a>

<a href="javascript:popUp('menunumerictofloat.xhtml')">To float</a>

<a href="javascript:popUp('menunumerictobigfloat.xhtml')">To bigfloat</a>

<a href="javascript:popUp('menunumericsetprecision.xhtml')">Set precision</a>

<a href="javascript:makeone();">makeone</a>
Type an input command line to Axiom:<br/>
<input type="text" id="p1" onclick="interpcall('p1');"
value="integrate(sin(x),x)"
size="80" />
</p>
</form>
<center>
<input type="button" value="Continue" name="continue"
onclick="intercall('p1');"/>
</center>
<div id="mathAns"></div>

complexlimit.xhtml

— complexlimit.xhtml —

```javascript
function commandline(arg) {
  var myform = document.getElementById("form2");
  var myfunct = myform.expr.value;
  var myvar = myform.vars.value;
  var ans = "";
  // decide what the limit point should be
  var finite = document.getElementById('finite').checked;
  if (finite == true) {
    var myreal = document.getElementById('fpreal').value;
    var mycomplex = document.getElementById('fpcomplex').value;
    if (myreal == 0) {
      if (mycomplex == 0) {
        ans = 'complexLimit('+myfunct+','+myvar+'=0)';
      } else {
        ans = 'complexLimit('+myfunct+','+myvar+'='+mycomplex+'*%i)';
      }
    } else {
      if (mycomplex == 0) {
        ans = 'complexLimit('+myfunct+','+myvar+'='+myreal+')';
      } else {
        ans = 'complexLimit('+myfunct+','+myvar+'='+myreal+'+'+mycomplex+'*%i)';
      }
    }
  } else {
    ans = 'complexLimit('+myfunct+','+myvar+'=%infinity)';
  }
  return(ans);
}
```

\getchunk{showfullanswer}
CHAPTER 1. OVERVIEW

conversionfunctions.xhtml

— conversionfunctions.xhtml —

You can use conversion (see Jenks section 9.27.2) to go back and forth between Integer, Fraction(Integer) and Float, as appropriate.
Since you are explicitly asking for a conversion, you must take responsibility for any loss of exactness.

This conversion cannot be performed: use<br />
<a href="dboptruncate.xhtml">truncate</a> or<br />
<a href="dbopround.xhtml">round</a> if that is what you intend.

The operations<br />
<a href="dboptruncate.xhtml">truncate</a> and<br />
<a href="dbopround.xhtml">round</a> truncate to the nearest integral <a href="db.xhtml?Float">Float</a> respectively.
The operation \(<a href="dbopfractionpart.xhtml">fractionPart</a>\) computes the fractional part of \(x\), that is, \(x - \text{truncate} \, x\).

The operation \(<a href="dbopdigits.xhtml">digits</a>\) allows the user to set the precision. It returns the previous value it was using.

The precision is only limited by the computer memory available. Calculations at 500 or more digits of precision are not difficult.
Numbers of type Float are represented as a record of two integers, namely, the mantissa and the exponent where the base of the exponent is binary. That is, the floating point number (m,e) represents the number m*2**e. A consequence of using a binary base is that decimal numbers can not, in general, be represented exactly.

— crytopage.xhtml —
<li> <a href="cryptoclass3.xhtml">
Laboratory Class 3: Number Theory
</a>
</li>
<li> <a href="cryptoclass4.xhtml">
Laboratory Class 4: Simple Cryptosystems
</a>
</li>
<li> <a href="cryptoclass5.xhtml">
Laboratory Class 5: RSA and public-key cryptosystems
</a>
</li>
<li> <a href="cryptoclass6.xhtml">
Laboratory Class 6: Digital Signatures
</a>
</li>
<li> <a href="cryptoclass7.xhtml">
Laboratory Class 7: Knapsack cryptosystems
</a>
</li>
<li> <a href="cryptoclass8.xhtml">
Laboratory Class 8: Modes of Encryption
</a>
</li>
<li> <a href="cryptoclass9.xhtml">
Laboratory Class 9: Hash Functions
</a>
</li>
</ol>

**Numbers and arithmetic**

- You can treat Axiom like a glorified calculator. Enter the following:

  - `<span class="cmd">3+5</span>`
<li> \(5 \times 7\)</li>
<li> \(2.3 / 3.5\)</li>
<li> \((3^4)^5\)</li>
<li> \(3 \times (4^5)\)</li>
</ul>

What happens if you enter the last command without the brackets?</li>

To obtain the factorial \(n!\), use the Axiom command
\[
\text{factorial(n)}
\]

By trial and error, find the smallest number whose factorial ends in six zeros.

</li>

<b>Lists</b></li>

Assignment is done using "\texttt{:=}" where the \texttt{colon-equals} symbols are used for assigning a particular object to a variable.

Lists are created using square brackets;

We can operate on all elements of a list using the \texttt{reduce} command:

Of course, these could be done as single commands:

Notice how the last result is given as a single large fraction. To obtain a decimal result we can do either of two things:

Convert the output to be of type \texttt{'Float'}:

Two colons can be used to change the type of an object.
CHAPTER 1. OVERVIEW

Use floats in the initial command:

\[ \texttt{reduce(*,[1.0/j for j in 5..15])} \]

Using lists, add up the first 1000 integers.

By trial and error, find the smallest number \( n \) for which the sum of the first \( n \) reciprocals is bigger than 8.

We can also add numbers by using the \texttt{sum} function; here's how to add the first 100 reciprocals:

\[ \texttt{sum(1.0/k,k = 1..100)} \]

Functions and maps

We shall create a simple function, and apply it to \texttt{mylist1} from above:

\[ \texttt{f(x) == x-2} \]

Supposing we want to subtract 2 from every element of a list without having to create a function first. In this case we can use the "mapping" symbols:

\[ \texttt{map(x +-> x-2,mylist1)} \]

Create a list called \texttt{nums} containing all the integers from 1 to 100. Now we shall create a simple function \texttt{f(x)} which returns \texttt{x} if it is prime, and 0 otherwise. The Axiom function \texttt{prime?} tests for primality:

\[ \texttt{f(x)==if prime?(x) then x else 0} \]

Now apply this function \texttt{f} to \texttt{nums}. Remove all the zeros:

\[ \texttt{remove(0,%)} \]
and determine how many primes there are, using the hash symbol # which can be used to count the number of elements in a list:

These last commands can be done as a single command:

These last commands can be done as a single command:

These last commands can be done as a single command:

How many primes are there less than 1000? Less than 10000?

Alternatively, we can list all the primes below 100 by creating our list using the "such that" operator---a vertical stroke:

or we could just return the length of the list:

or we could just return the length of the list:

How many primes are there less than 2000? Less than 15000?

Housekeeping

Axiom contains many commands for managing your workspace and your environment; such commands are all prefixed with a right parenthesis.

Sometimes you need to clear a variable, say a variable x:

Most commands of this sort can be abbreviated using their first two letters:

To clean out everything:
To see what variables you've accumulated over your work:

\( )\text{display names} \) or abbreviated as \( )\text{d n} \)

You may have noticed earlier that Axiom poured out lots of messages when it first "got going". These can be turned off:

\( )\text{set messages autoload off} \)

Note here that if you just type in \( )\text{set} \) or its abbreviation \( )\text{se} \), you'll be presented with the list of all the possible options. Likewise \( )\text{se me} \) lists all possible options for messages, and so on.

Can you find the command which turns on a time function, so gives the time to compute each command?

The command \( )\text{summary} \) gives a quick summary of these commands.

To quit Axiom, type

\( )\text{quit} \) or its one letter abbreviation \( )\text{q} \), followed by \( )\text{y} \) to confirm.

Characters and Strings
All printable characters have a fixed ASCII value; some of which are:

<table>
<thead>
<tr>
<th>Character</th>
<th>A B Y Z a b y z</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII Value</td>
<td>65 66 89 90 97 98 121 122</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Character</th>
<th>0 1 8 9 , - . /</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII Value</td>
<td>48 49 56 57 44 45 46 47</td>
</tr>
</tbody>
</table>

To obtain values 0 to 25 for A to Z, we need to subtract 65 from the ASCII values.

In Axiom, the `ord` command gives the ASCII value of a character. Create a string such as:

```plaintext
str:="THISISASTRING"
```

A string can be turned into a list of characters using `members(str)`.

This means a string can be turned into a list of ASCII values by mapping the `ord` function onto the list of members:

```plaintext
map(ord,members(str))
```

To obtain values in the 0--25 range, try using an unnamed function:

```plaintext
strn:=map(x +-> ord(x)-65,members(str))
```

Use this last command to create a function `str2lst` which will take a string (assumed to be of capital letters, with no spaces or punctuation), and return a list of values between 0 and 25.

To go the other way, we first need to add 65 to all elements of `strn`:

```plaintext
map(x +-> x+65,strn)
```

Turn this into characters with `char`:

```plaintext
map(char,%)```

These can be done as a single command:
<ul>
  <li>
    <span class="cmd">map(x +-> char(x+65),strn)</span></li>
</ul>

To put them all together as a single string we can concatenate them with the <tt>concat</tt> function from the <tt>String</tt> domain:

<ul>
  <li>
    <span class="cmd">concat(%)$String</span></li>
</ul>

In one line:

<ul>
  <li>
    <span class="cmd">concat(map(x +-> char(x+65),strn)$String</span></li>
</ul>

Alternatively, we could convert the characters to type <tt>String</tt> before concatenation:

<ol>
  <li>
    <span class="cmd">concat(map(x +-> char(x+65)::String,strn))</span></li>
</ol>

Use either version of this last command to create a function <tt>lst2str</tt> which will take a list of values between 0 and 25 and return a string.

Create a text file in one of your private directories called <tt>my3720.input</tt> and copy your <tt>str2lst</tt> and <tt>lst2str</tt> functions to it.

You can read command line input from a file with the extension <tt>.input</tt> using the <tt>)read</tt> command:

<ul>
  <li>
    <span class="cmd">)read my3720</span></li>
</ul>

The Caesar cipher can be implemented by the following three steps:

<ol>
  <li>
    Turn the string into a list,
  </li>
  <li>
    Add 3 to every number in the list,
  </li>
  <li>
    Turn this new list back into a string.
</ol>

To ensure that step (2) remains in the 0--25 range, we need to use the <tt>rem</tt> function. These can all be put together as:

<ol>
  <li>
    <span class="cmd">caesar(str) == lst2str(map(x +-> (x+3) rem 26, str2lst(str)))</span>
  </li>
</ol>
Try this out on a few strings of your choice.

By replacing the "<tt>+3</tt>" in the <tt>caesar</tt> function with "<tt>+n</tt>" create a new function called <tt>trans(str,n)</tt> which implements a general translation cipher.

Test it out; these two commands should produce the same results.

<ul>
  <li><span class="cmd">caesar("MYSTRING")</span></li>
  <li><span class="cmd">trans("MYSTRING",3)</span></li>
</ul>

If you like, add the <tt>caesar</tt> and <tt>trans</tt> functions to your <tt>my3720.input</tt> file.

Test your <tt>trans</tt> function out on a few other strings and translation values.

The <tt>ROT13</tt> cipher is used in Usenet postings to hide information which might be considered offensive. It is a translation cipher with a shift of 13. Since 13 is half of 26, this means that encryption and decryption are exactly the same. Apply <tt>ROT13</tt> to:

<ul>
  <li>GUVFVNIRELFREVBHFOHFVARFF</li>
</ul>

Consider this string which has been produced with a translation cipher. To decrypt it, simply apply all possible shifts until you obtain understandable text.

<ul>
  <li>IUDTCUQBBOEKHCEDUO</li>
</ul>

To apply all the possible shifts do:

<ol>
  <li><span class="cmd">ct:="IUDTCUQBBOEKHCEDUO"</span></li>
  <li>for i in 1..26 repeat output trans(ct,i)</li>
</ol>

What is the plaintext?

---

cryptoclass3.xhtml

--- cryptoclass3.xhtml ---

\getchunk{standard head}
Check out the commands `gcd` and `factor`, and test them on different numbers, small and large.

Axiom provides a few useful commands for taking apart the factors of an object:

- `n := 5040
- `f := factor(n)
- `numf := numberOfFactors(f)
- `fs := [nthFactor(f, i) for i in 1..numf]
- `es := [nthExponent(f, i) for i in 1..numf]
- `reduce(*, [fs.i ^ es.i for i in 1..numf])

The last command simply multiplies all the factors to their powers.

Check out the commands `prime?`, `nextPrime` and `prevPrime`.

To compute the \(i\)-th prime, we can construct a stream in Axiom:

```axiom
primes: Stream Integer := [i for i in 2.. | prime? i]
```

Now we can find, for example, the 100-th prime, and the 2500-th prime:

```axiom
primes.100
primes.2500
```

Create random 10 digit primes:

```axiom
p := nextPrime(random(10^10))
q := nextPrime(random(10^10))
```

Now multiply them and factor the product. How long did it take?

Try the same thing with 12 digit primes and 15 digit primes.
The extended Euclidean algorithm is implemented by the command 
<tt>extendedEuclidean</tt>. Here's how to use it:

```plaintext
<ol>
  <li> <span class="cmd">a:=1149</span></li>
  <li> <span class="cmd">b:=3137</span></li>
  <li> <span class="cmd">g:=extendedEuclidean(a,b)</span></li>
  <li> <span class="cmd">s:=g.coef1</span></li>
  <li> <span class="cmd">t:=g.coef2</span></li>
</ol>
```

and now test them:

```plaintext
<ol>
  <li> <span class="cmd">s*a+t*b</span></li>
</ol>
```

Try this on a few other numbers.

Axiom uses the command <tt>positiveRemainder</tt> instead of <tt>mod</tt> command, so let's define <tt>mod</tt> to be a renaming of the <tt>positiveRemainder</tt> function:

```plaintext
<ol>
  <li> <span class="cmd">mod ==> positiveRemainder</span></li>
</ol>
```

Now the commands <tt>addmod</tt>, <tt>submod</tt>, <tt>mulmod</tt>, and <tt>invmod</tt> can be used to perform modular arithmetic. Here's a few examples; first a simple modulus calculation:

```plaintext
<ol>
  <li> <span class="cmd">-10 mod 3</span></li>
</ol>
```

Addition, subtraction and multiplication mod 14:

```plaintext
<ol>
  <li> <span class="cmd">addmod(10,13,14)</span></li>
  <li> <span class="cmd">submod(17,23,14)</span></li>
  <li> <span class="cmd">mulmod(13,27,14)</span></li>
</ol>
```

Powers and inverses:

```plaintext
<ol>
  <li> <span class="cmd">powmod(19,237,14)</span></li>
  <li> <span class="cmd">invmod(11,14)</span></li>
</ol>
```

Find out what happens if you try to take an inverse of a number not relatively prime to the modulus:

```plaintext
<ol>
  <li> <span class="cmd">invmod(12,14)</span></li>
</ol>
```

Try these command with a few other numbers, and test out the examples in the notes.
The second method, which can be more powerful, is to treat all numbers as elements of the residue values 0 to $n-1$. This can be done with the $\text{IntegerMod}$ construction, or its abbreviation $\text{ZMOD}$. Here's a few examples:

```plaintext
a:=11::ZMOD 14
```

This declares the variable $a$ to be a member of the residue class modulo 14. Now all arithmetic including $a$ will be reduced to this same class of values:

```plaintext
a+25
a*39
a^537
```

Inversion can be done with the $\text{recip}$ command:

```plaintext
recip(a)
```

We don't have to define a variable first. All the above commands could be equivalently written as:

```plaintext
(11::ZMOD 14)+25
11::ZMOD 14*39
11::ZMOD 14^537
recip(11::ZMOD 14)
```

If the modulus is a prime, then division (by non-zero values) is also possible. Axiom provides the alternative construction $\text{PrimeField}$ or more simply $\text{PF}$. For example:

```plaintext
a:=7::PF 11
```

All the above arithmetic operations of addition, subtraction, multiplication and powers work, but now we also have inversion:

```plaintext
1/a
```

Using any of the methods you like, test out Fermat's theorem for a large prime $p$ and an integer $a$.

Euler's totient function is implemented with $\text{eulerPh}$. Choose a large integer $n$, a random $a$ with $\gcd(a,n)=1$, and test Euler's theorem
We have experimented with the Caesar cipher and the more general translation cipher. We shall start looking at the Vigenère cipher. The trick is to add the correct letter of the code to the letter of the key:

\[
\text{Index of plain text } i: 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8 \\
\text{Plaintext: W I T H D R A W} \\
\text{Key: C O D E C O D E} \\
\text{Index of key } j: 1\ 2\ 3\ 4\ 1\ 2\ 3\ 4
\]

The indices of the key repeat 1, 2, 3, 4. We can get a repetition of length four by using a modulus of 4:

\[
i: 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8 \\
i-1: 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7 \\
i-1 \pmod{4}: 0\ 1\ 2\ 3\ 0\ 1\ 2\ 3 \\
i-1 \pmod{4} + 1: 1\ 2\ 3\ 4\ 1\ 2\ 3\ 4
\]

What we need to do is to subtract one before the modulus, and add one after:

\[
i: 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8 \\
i-1: 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7 \\
i-1 \pmod{4}: 0\ 1\ 2\ 3\ 0\ 1\ 2\ 3 \\
i-1 \pmod{4} + 1: 1\ 2\ 3\ 4\ 1\ 2\ 3\ 4
\]

This means that in the Vigenère cipher, we add the \(i\)-th character of the plaintext, and the \(j\)-th character of the key, where

\[
j = i-1 \pmod{n} + 1
\]

with \(n\) being the length of the key.

\[
\begin{itemize}
\item First read in the \texttt{rcm3720.input} file you have created:
\end{itemize}
\]
<li> You may have to include the full path here.</li>
<li> Enter a plaintext</li>
<ul>
<li><span class="cmd">plaintext:="WITHDRAWONEHUNDREDOLLARS"</span></li>
</ul>
<li> and a keyword:</li>
<ul>
<li><span class="cmd">key := "CODE"</span></li>
</ul>
<li> Now we can obtain the lengths of the plaintext and key with the hash symbol:</li>
<ul>
<li><span class="cmd">pn:=#plaintext</span></li>
<li><span class="cmd">kn:=#key</span></li>
</ul>
<li> Turn both plaintext and key into lists of numbers:</li>
<ul>
<li><span class="cmd">pl:=str2lst(plaintext)</span></li>
<li><span class="cmd">kl:=str2lst(key)</span></li>
</ul>
<li> Now we can add them using the formula for <tt>j</tt> above to obtain the list corresponding to the ciphertext:

```plaintext
cl:=[(pl.i+kl.((i-1) rem kn+1))::ZMOD 26 for i in 1..pn]
```

</li>
<li> And obtain the ciphertext (we need to convert our list to a list of integers first):
<ul>
<li><span class="cmd">ciphertext:=lst2str(cl::List INT)</span></li>
</ul>
</li>
<li> Try a few other Vigenère encryptions.</li>
<li> To decrypt, we just subtract the key value from the ciphertext value:
<ul>
<li><span class="cmd">pl:=[(cl.i+kl.((i-1) rem kn+1))::ZMOD 26 for i in 1..pn]</span></li>
</ul>
</li>
Now for the Hill (matrix) cipher. We shall use 3 x 3 matrices, so first create a plaintext whose length is a multiple of 3:

```
plaintext:="WITHDRAWONEHUNDREDOLLARSXX"
```

```
pl:=str2lst(plaintext)
r:=3
c:=#pl/r
```

The values $r$ and $c$ are the row and column numbers of the plaintext matrix.

Now put all the plaintext values into a $r \times c$ matrix:

```
S:=matrix([[pl.(r*(i-1)+j) for i in 1..c] for j in 1..r])
```

Create the key matrix:

```
M:Matrix ZMOD 26:=matrix([[22,11,19],[15,20,24],[25,21,16]])
```

Multiply the two matrices:

```
C:=M*S
```

Notice how the results are automatically reduced modulo 26, because that is how the matrix $M$ was defined.

Now we have to read off the elements of $C$ into a single list; this can be done by transposing the matrix, and reading off the rows as lists:

```
CL:=concat(transpose(C)::List List ZMOD 26)
```

And finally turn into ciphertext:
Finally, here's how we can invert our matrix $M$ modulo 26:

1. $\text{adj} := \text{adjoint}(M).\text{adjMat}$
2. $\text{invdet} := \text{recip}(...)$
3. $MI := \text{invdet} * \text{adj}$

Or alternatively, as one command:

\[
MI := \text{recip}(\text{det}(M)) * \text{adjoint}(M).\text{adjMat}
\]

Check the result:

\[
M * MI
\]
is also available <a href="rcm3720.input">here</a>. If you obtain it from the website, save it to a place of your choice, and <tt>read</tt> it into your Axiom session using the full path, as shown above.

Now create some large primes and their product:

- <span class="cmd">r() == rand(2^100)</span>
- <span class="cmd">p:=nextPrime(r())</span>
- <span class="cmd">q:=nextPrime(r())</span>
- <span class="cmd">n:=p*q</span>

Choose a value <tt>e</tt> and ensure that it is relatively prime to your <tt>(p-1)(q-1)</tt>, and determine <tt>d=e^-1 mod (p-1)(q-1)</tt>. (Use the <tt>invmod</tt> function here).

Create a plaintext:

- <span class="cmd">pl:="This is my plaintext.""></span>

Encrypt this number using the RSA method:

- <span class="cmd">ct:=powmod(pln,e,n)</span>

and decrypt the result:

- <span class="cmd">decrypt:=powmod(ct,d,n)</span>
- <span class="cmd">num2str(decrypt)</span>

With a friend, swap your public keys and use them to send each other a ciphertext encrypted with your friend’s public key.

Now decrypt the ciphertext you have received using your private key.

Now try Rabin: create two large primes <tt>p</tt> and <tt>q</tt> and ensure that each is equal to 3 mod 4. (You might have to run the <tt>nextPrime</tt> command a few times until you get primes which work.)

Create <tt>N=pq</tt> and create a plaintext <tt>pl</tt>, and its numerical equivalent.

Determine the ciphertext <tt>c</tt> by squaring your number mod <tt>N</tt>. 

Determine the $s$ and $t$ for which $sp+ tq=1$ by using the extendedEuclidean function.

Now follow through the Rabin decryption:

- $cp := \text{powmod}(c,(p+1)/4,N)$
- $cq := \text{powmod}(c,(q+1)/4,N)$
- $c1 := (s*p*cq+t*q*cp) \mod N, \text{num2str}(c1:\text{INT})$
- $c2 := (s*p*cq-t*q*cp) \mod N, \text{num2str}(c2:\text{INT})$
- $c3 := (-s*p*cq-t*q*cp) \mod N, \text{num2str}(c3:\text{INT})$
- $c4 := (-s*p*cq+t*q*cp) \mod N, \text{num2str}(c4:\text{INT})$

One of the outputs $c1$, $c2$, $c3$ and $c4$ should produce the correct plaintext; the others should be gibberish.

As above, swap public keys with a friend, and use those public keys to encrypt a message to him or her. Now decrypt the ciphertext you have been given.

For the El Gamal system, you need a large prime and a primitive root. Create a large prime $p$ and find a primitive root $a$ using.

- $a := \text{primitiveElement()} \mod p$

The $\text{primitiveElement()}$ command is not very efficient, so if it seems to be taking a long time, abort the computation and try with another prime.

Do this in pairs with a friend, so that you each agree on a large prime and a primitive root.

Now choose a random value $A$:

- $A := \text{random}(p-1)$
and create your public key $A_1=a^A \pmod{p}$:
\[
A_1 := a^A
\]
Swap public keys with your friend.

Create a plaintext $pl$ and its number $pln$, and create the ciphertext as follows (where $A_1$ is your friend's public key):
\[
\begin{align*}
&k := \text{random}(p-1) \\
&K := A_1^k \\
&C := [a^k, K*pln]
\end{align*}
\]
This pair $C$ is the ciphertext you send to your friend.

Now decrypt the ciphertext you have been sent:
\[
\begin{align*}
&K := C.1^A \\
&m := C.2/K \\
&\text{num2str}(m::\text{INT})
\end{align*}
\]

---

cryptoclass6.xhtml

— cryptoclass6.xhtml —

You will need to read in the <a href="rcm3720.input">rcm3720.input</a> file for the <tt>str2num</tt> and <tt>num2str</tt> procedures.

<b>NOTE:</b> To save typing in all the messages and long signature numbers, just copy them from <a href="signatures.txt">signatures.txt</a>.
For an RSA signature scheme, I provide the public key \((n,e)\), where

\[
\begin{align*}
n &= 2^{137} - 1, \quad e = 17 \\
\end{align*}
\]

This value \(n\) has two large prime factors.

Use my public key to verify my signature of the following message:

This is my text.
68767027465671577191073128495082796700768

Now try with the public key

\[
\begin{align*}
n &= (6^{67} - 1)/5, \quad e = 17 \\
\end{align*}
\]

to verify my signature:

Please feed my dog!
1703215098456351993605104919259566435843590978852633

For a Rabin signature scheme, I provide the public key

\[
\begin{align*}
N &= (7^{74} - 1)/6, \\
\end{align*}
\]

which I know can be factorized into two large primes.

Check the following message and signature:

Arrive Thursday.
189479723122534414019783447271411895509

For an El Gamal signature scheme, I choose the next prime after

\[
\begin{align*}
2^{150} &= 369851585774063312693119161120024351761244461 \\
\end{align*}
\]

which has a primitive root \(a = 2\). My public key is

Leave AT ONCE!,
1389080525305754392111976715361069425353578198
For a DSS signature, choose \(<tt>p</tt>\) to be the next prime after 170
and \(q=143441505468590696209\).

Verify that \(<tt>q</tt>\) is a divisor of \(<tt>p-1</tt>\).

A primitive root of \(<tt>p</tt>\) is \(<tt>a=3</tt>\).

Use this primitive root to determine
\[
(p-1)/q
\]
\[
g = a \mod p
\]

The public key value is
\[
B=139425688065959564848116770226045673904445792389839.
\]

Now using these values, verify this signature:
\[
Now's your chance!
64609209464638355801
13824808741200493330
\]

Now exchange some public keys with a friend, and sign messages to each other. Then verify the signatures you have been sent. Make sure you try each of

- RSA signatures,
- Rabin signatures,
- El Gamal signatures,
- DSS.
You will need to read in the `rcm3720.input` file for various necessary procedures.

The subset sum problem

We will first experiment with this problem; creating random lists and adding up elements from them.

- Start with a list of eight elements:
  - `$\text{ln}:=8$`
  - `$\text{lst}:=\left[\text{random}(10^6)\text{ for }i\text{ in }1..\text{ln}\right]$`
  - `$\text{m}:=\left[\text{random}(2)\text{ for }i\text{ in }1..\text{ln}\right]$`
  - `$\text{c}:=\text{reduce(}+\text{,}[\text{m}.i*\text{lst}.i\text{ for }i\text{ in }1..\text{ln}]\text{)}$`
  - `$\text{subsetsum(lst,c)}$`

The `subsetsum` command implements a fairly non-efficient command for attempting to solve the subset sum problem for an arbitrary list.

Try the above commands, but starting with a length `ln` of 12. You should find the command is a bit slower this time. Use this command to time it:

```
)set messages time on
```

Experiment with lengths of 16 and 20. How long does the `subsetsum` command take for each of these values?

Superincreasing sequences

- Create a superincreasing sequence with
  - `$\text{ln}:=8$`
  - `$\text{lst}:=\left[\text{random}(10^6)\text{ for }i\text{ in }1..\text{ln}\right]$`
  - For `i` in `2..ln` repeat
    - `$\text{lst}.i:=\text{reduce}(+\text{,}[\text{lst}.j\text{ for }j\text{ in }1..i-1])\text{+random(10)}+1$`
Now create \( m \) and \( c \) as above. This time, solve the problem with
\[
\text{siSolve(lst,c)}
\]
Now try with larger lengths: 12, 16 and 20, and time the commands each time.
What can you say about solving the subset sum problem for general and superincreasing lists?

The Merkle-Hellman additive knapsack system

Create a superincreasing list of length \( \ln \) 10, and call it \( a \). Create a new number \( N \) greater than the sum of all values of \( a \). Check with
\[
N > \text{reduce(+,[a.i for i in 1..ln])}
\]
Now choose (randomly) a value \( wN \) and which is relatively prime to \( N \). Then construct your public key:
\[
b := \text{map}(x +\to x \cdot w \text{ rem } N,a)
\]
Now for an encryption and decryption. Create a random message \( m \) as above, and encrypt it to a ciphertext \( c \) using the public key \( b \).
Decrypt it as follows:
\[
c1 := \text{inv_mod(w,N)} \cdot c \text{ rem } N
\]
\[
\text{siSolve(a,c1)}
\]
Experiment with longer lists and messages: 12, 16, 20 or even larger.

The Merkle-Hellman multiplicative knapsack system

Choose \( a \) to be the first ten primes, and a large prime \( p \):
\[
a := [2,3,5,7,11,13,17,19,23,29]
\]
\[
p := 6469785001
\]
CHAPTER 1. OVERVIEW

Check that \( p \) is greater than the product of all elements of \( a \):

\[
p > \text{reduce}(*,[a.i \text{ for } i \in 1..10])
\]

And that \( p-1 \) has only small factors:

\[
\text{factor}(p-1)
\]

Choose as a primitive root the value 34:

\[
r := 34
\]

\[
\text{primitive?(r)}\text{PF}(p)
\]

And compute the public key:

\[
b := \text{map}(x \rightarrow \text{discreteLog}(r,x)\text{PF}(p),a)
\]

Create a message of length 10, and encrypt it using the public key \( b \):

\[
c := \text{reduce}(+,[m.i*b.i::\text{INT} \text{ for } i \in 1..\ln])
\]

Decryption is now done with:

\[
c1 := \text{powmod}(r,c,p)
\]

\[
\text{factor}(c1)
\]
We will investigate the different modes of encryption using the Hill (matrix) cryptosystem. Start off by entering some matrices:

\[ M := \begin{bmatrix} 15 & 9 & 21 \\ 2 & 10 & 7 \\ 16 & 11 & 12 \end{bmatrix} \text{::Matrix ZMOD 26} \]

\[ M_I := \begin{bmatrix} 7 & 17 & 19 \\ 24 & 0 & 23 \\ 12 & 25 & 10 \end{bmatrix} \text{::Matrix ZMOD 26} \]

Check that you've entered everything correctly with

\[ M \cdot M_I \]

Note that because the matrices were defined in terms of numbers mod 26, their product is automatically reduced mod 26.

Now enter the following column vector:

\[ \text{zero31:=matrix([[[0],[0],[0]]]} \text{::Matrix ZMOD 26} \]

For this lab, rather than fiddling about with translations between letters and numbers, all our work will be done with numbers alone (in the range 0..25).

For electronic codebook mode, encryption is performed by multiplying each plaintext block by the matrix, and decryption by multiplying each ciphertext block by the inverse matrix:

\[ C = M \cdot P, \quad P = M^{-1} \cdot C \]

where all arithmetic is performed mod 26.
Start by entering a plaintext, which will be a list of column vectors:

\[ P := \text{matrix}([[3*i],[3*i+1],[3*i+2]]) \text{ for } i \text{ in } 0..7 \]

and a list which will receive the ciphertext:

\[ C := [\text{zero31} \text{ for } i \text{ in } 1..8] \]

Encrypt it:

\[ \text{for } i \text{ in } 1..8 \text{ repeat } C.i := M \cdot P.i \]

Now decrypt (first make an empty list \( D \)):

\[ D := [\text{zero31} \text{ for } i \text{ in } 1..8] \]

\[ \text{for } i \text{ in } 1..8 \text{ repeat } D.i := M^{-1} \cdot C.i \]

If all has worked out, the list \( D \) should be the same plaintext you obtained earlier.

Now change one value in the plaintext:

\[ Q := P \]
\[ Q.3 := \text{matrix}([[6],[19],[8]]) \]

Now encrypt the new plaintext \( Q \) to a ciphertext \( E \). How does this ciphertext differ from the ciphertext \( C \) obtained from \( P \)?

Check that you can decrypt \( E \) to obtain \( Q \).

For cipherblock chaining mode, the encryption formula for the Hill cryptosystem is

\[
C = M(P + C_{i-1})
\]

and decryption is

\[
-1
\]
\[ P = M \cdot C - C \]
\[ i \quad i \quad i-1 \]

To enable us to use these formulas, we shall first add an extra column to the front of \( P \) and \( C \):

- \( P := \text{append}([\text{zero31}], P) \)
- \( C := \text{append}([\text{zero31}], C) \)

Next, we need to create an initialization vector:

- \( IV := \text{matrix}([[\text{random}(26)] \text{ for } i \text{ in } 1..3]) \)

Now for encryption:

- \( C.1 := IV \)
- \( \text{for } i \text{ in } 2..9 \text{ repeat } C.i := M \cdot (P.i + C.(i-1)) \)

Let's try to decrypt the ciphertext, using the CBC formula:

- \( D := \text{[zero31 for } i \text{ in } 1..9] \)
- \( \text{for } i \text{ in } 2..9 \text{ repeat } D.i := M \cdot (C.i - C.(i-1)) \)

Did it work out?

As before, change one value in the plaintext:

- \( Q := P \)
- \( Q.4 := \text{matrix}([[6], [19], [8]]) \)

Now encrypt \( Q \) to \( E \) following the procedure outlined above. Compare \( E \) with \( C \)---

- How much difference is there?
- How does this difference compare with the differences of ciphertexts obtained with ECB?

Just to make sure you can do it, decrypt \( E \) and make sure you end up with a list equal to \( Q \).
Output feedback mode works by creating a key stream, and then adding it to the plaintext to obtain the ciphertext. With the Hill system, and an initialization vector IV:

$$k = \text{IV}, \ k = M^i_k$$

and then

$$c = p + k$$

First, the key stream:

1. \(K := \text{[zero31 for } i \text{ in 1..9]}\)
2. \(K.1 := \text{IV}\)
3. \(\text{for } i \text{ in 2..9 repeat } K.i := M*K.(i-1)\)

Next the encryption:

1. \(\text{for } i \text{ in 2..9 repeat } C.i := K.i + P.i\)

What is the formula for decryption? Apply it to your ciphertext \(C\).
Given two prime numbers $p$ and $q$, and their product $N$, we can define a hash of a number $n$ to be

$$n \text{ hash} = g \pmod{N}$$

This is provably collision resistant, because if we want to find two hashes which are equal, then we need to find $m$ and $n$ for which

$$m \text{ hash} = n \text{ hash} \pmod{N}$$

or that

$$m - n = 1 \pmod{\phi(N)}$$

By Euler's theorem, we know that

$$\phi(N) = g = 1 \pmod{N}$$

This means that finding a collision requires finding two numbers $m$ and $n$ for which

$$m = n \pmod{\phi(N)}$$

Since computing $\phi(N)$ requires a knowledge of the factorization of $N$, this will be hard if $p$ and $q$ are large.

Enter the following commands:

- $p := \text{nextPrime}(87654321)$
- $q := \text{nextPrime}(98765432)$
- $N := p \times q$
- $g := 17$

Read in the utility file <a href="rcm3720.input">rcm3720.input</a>

Now experiment with the following hashes:

- $n := \text{str2num}(\text{"A cat"})$
- $h := \text{powmod}(g, n, N)$
- $n := \text{str2num}(\text{"A bat"})$
- $h := \text{powmod}(g, n, N)$
Even though the strings are very similar, how similar are the hash values?

Experiment with hashing some other strings—some short, some long.

Read in a text file (any text file, of any length) as follows:

```
  f:TextFile:=open("\full\path\to\file","input")
```

```
  str:=""
  while not endOfFile?(f) repeat str:=concat(str,readLine(f));
```

Now the variable `str` will contain the file as one long string. Hash this string, by converting it to a number first.

Try this with a few different text files, of different lengths—some short, some long.

A simplified version of MASH

We shall experiment with a simplified version of the MASH hash function:

```
  Start with two prime numbers `p` and `q`, and their product `N`.
  Turn the data to be hashed into a single integer `n`.
  Express `n` as ‘‘digits’’ in base `N`:

  \[
  n = a_0 + a_1 N + a_2 N + a_3 N + \ldots + a_q N
  \]

  Start with `H` being the largest prime less than `N`.
  For `i` from 0 to `q`:

  \[
  H \texttt{#60;-- (}H + a_i\texttt{) } +H \pmod{N}
  \]
The final value of $H$ is the hash.

With $p$, $q$, and $N$ as before, pick a long string (or the string from a text file) to be hashed, and turn it into a number $n$.

Determine the ‘‘digits’’ in base $N$:

Now create the hash:

Note that since the elements of the list $a$ are already defined as being modulo $N$, we don’t have to use a mod function in this last step.

Create the hashes of a few other strings and files. What happens if you try to hash a really long text file?

Experiment with hashing using some other (large) primes.
binary lists. Since the definitions of the sDES functions require lists to be
indexed starting at 0, but in Axiom lists are indexed starting at 1, many of
the operations will have extra ones added at some stage.

<ul>
<li>Save the file `<tt>des.input</tt>` to a directory in which you
have write access. Read the file into Axiom, and open up the file
with a text editor.
</li>
<li>Compare the first command `<tt>perm(b)</tt>` with the initial
permutation for sDES defined in page 94 of the notes. How do the
indices in the Axiom command relate to the indices of the
permutation in the notes?
</li>
<li>Now using the above procedure as a guide, write a procedure called
`<tt>invperm</tt>` to perform the inverse permutation.
</li>
<li>Test this procedure: it should invert the permutation you
obtained from the `<tt>perm</tt>` procedure.
</li>
<li>The `<tt>subkey</tt>` procedure creates two lists: one for the
first subkey, and one for the second. Edit the procedure to include
the second subkey as given on the bottom of page 95.
</li>
<li>Write a procedure called `<tt>expperm</tt>` which implements the
expansion permutation on page 96; use the `<tt>perm</tt>` and
`<tt>invperm</tt>` procedures as guides.
</li>
<li>Using the `<tt>sbox0</tt>` procedure as a guide, write a procedure
to implement S-box 1.
</li>
<li>The mixing function shown in figure 8.5 in the notes is
implemented as `<tt>mix</tt>`. This procedure has been
commented.
</li>
<li>Comment each line of the `<tt>feistel</tt>` and `<tt>sdes</tt>`
procedures in a similar fashion.
</li>
<li>Test the `<tt>sdes</tt>` procedure on the example given in the notes.
</li>
<li>Modify your procedure to implement sDES decryption, using the
scheme given on page 99.
</li>
</ul>
<li> Test that your decryption procedure works; that it decrypts the ciphertext produced by your encryption procedure to the original plaintext. </li>
</ul>

— cryptoclass11.xhtml —

Enter the following definition of the finite field
<pre>
3
\[ \mathbb{Z}[x]/(x^2+x+1) \]
</pre>
<ul>
<li> \texttt{F:=FFP(PF 2,x\^{}3+x+1)} </li>
</ul>

To perform field operations, we need to create a generator of the field: a symbol which can be used to generate all elements as polynomials:
<ul>
<li> \texttt{x:=generator()}$F$ </li>
<li> \texttt{(x^2+1)(x+1)} in the field: </li>
<li> \texttt{(x^2+1)*(x+1)} </li>
<li> \texttt{1/(x^2+x)}: </li>
<li> \texttt{1/(x^2+x)}
<br/>Note that Axiom returns its answer in terms of a dummy variable. </li>
</ul>

We can also list tables of powers:
Before we enter a new field, we need to clear \( x \) and its properties:

\[
\text{cl pr x}
\]

Now for a slightly bigger field:

\[
\begin{align*}
4 & \quad 3 \\
Z & \langle x^4 + x^3 + 1 \rangle \\
2 & \\
\end{align*}
\]

\[
\text{F2:=FFP(PF 2,x^4+x^3+1)}
\]

Create a list of powers of \( x \).

Evaluate \( (x^3 + x + 1) / (x^3 + x^2) \) in this field.

Enter the Rijndael field,

\[
\begin{align*}
8 & \quad 4 & \quad 3 \\
Z & \langle x^8 + x^4 + x^3 + 1 \rangle \\
2 & \\
\end{align*}
\]

and call it \( GR \).

Determine whether \( x \) is a primitive element in this field:

\[
\text{x:=generator()$GR}
\]

Is \( x+1 \) a primitive element?

Investigate the workings of MixColumn. First create the matrix:

\[
M: \text{Matrix GR:=matrix([[x,x+1,1,1],[1,x,x+1,1],[1,1,x,x+1],[x+1,1,1,x]])}
\]
Instead of multiplying a matrix \( C \) by \( M \), we shall just look at a single column, created randomly:

\[
\text{C:Matrix GR:=matrix([[random() for j in 1..4]])}
\]

These can be multiplied directly in Axiom:

\[
\text{D:=M*C}
\]

Remarkably enough, Axiom can operate on matrices over a finite field as easily as it can operate on numerical matrices. For example, given that

\[
\text{D=MC}
\]

it follows that

\[
\text{C=M \quad D}
\]

or that

\[
\text{-1 \quad M \quad D-C=0}
\]

To test this, first create the matrix inverse:

\[
\text{MI:=inverse(M)}
\]

Now multiply by \( D \) and subtract \( C \). What does the result tell you about the truth of the final equation?

To explore \text{MixColumn} a bit more, we shall look at the inverse of \( M \). First, here's a small function which converts from a polynomial to an integer (treating the coefficients of the polynomial as digits of a binary number):

\[
\text{poly2int(p)\Rightarrow(tmp:=\text{reverse(coordinates(p)}),return integer \text{wholeRadix(tmp::LIST INT)$RadixExpansion(2)})}
\]
First check the matrix $M$:

```
m = map((x -> poly2int(x)::INT), M)
```

Is this what you should have?

Now apply the same command but to $MI$ instead of to $M$. What is the result?
CHAPTER 1. OVERVIEW

— dbcomplexfloat.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbcomplexfloat not implemented
\getchunk{page foot}

— dbcompleximag.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbcompleximag not implemented
\getchunk{page foot}

— dbcomplexnorm.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbcomplexnorm not implemented
\getchunk{page foot}

— dbcomplexreal.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbcomplexreal not implemented
\getchunk{page foot}
dbcomplexinteger.xhtml

--- dbcomplexinteger.xhtml ---
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbcomplexinteger not implemented
\getchunk{page foot}

---

dbexpressioninteger.xhtml

--- dbexpressioninteger.xhtml ---
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
dbexpressioninteger not implemented
\getchunk{page foot}

---

dbfractioninteger.xhtml

--- dbfractioninteger.xhtml ---
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
dbfractioninteger not implemented
\getchunk{page foot}

---

dbfractionpolynomialinteger.xhtml

--- dbfractionpolynomialinteger.xhtml ---
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
dbfractionpolynomialinteger not implemented
\getchunk{page foot}

---
dblookup.xhtml

— dblookup.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dblookup not implemented
\getchunk{page foot}

-----

dbopacos.xhtml

— dbopacos.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopacos not implemented
\getchunk{page foot}

-----

dbopacosh.xhtml

— dbopacosh.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopacosh not implemented
\getchunk{page foot}

-----

dbopacot.xhtml

— dbopacot.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopacot not implemented
\getchunk{page foot}

-----
dbopacoth.xhtml

— dbopacoth.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopacoth not implemented
\getchunk{page foot}

——

dbopacsc.xhtml

— dbopacsc.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopacsc not implemented
\getchunk{page foot}

——

dbopacsch.xhtml

— dbopacsch.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopacsch not implemented
\getchunk{page foot}

——

dbobaddmod.xhtml

— dbobaddmod.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbobaddmod not implemented
\getchunk{page foot}

——
dbopatanh.xhtml

--- dbopatanh.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopatanh not implemented
\getchunk{page foot}

---

dbopbernoullib.xhtml

--- dbopbernoullib.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
dbopbernoullib not implemented
\getchunk{page foot}

---

dbopbesseli.xhtml

--- dbopbesseli.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopbesseli not implemented
\getchunk{page foot}

---

dbopbesselj.xhtml

--- dbopbesselj.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopbesselj not implemented
\getchunk{page foot}

---
dbopbesselk.xhtml

<table>
<thead>
<tr>
<th>dbopbesselk.xhtml</th>
</tr>
</thead>
<tbody>
<tr>
<td>standard head</td>
</tr>
<tr>
<td>body</td>
</tr>
<tr>
<td>page head</td>
</tr>
<tr>
<td>dbopbesselk not implemented</td>
</tr>
<tr>
<td>page foot</td>
</tr>
</tbody>
</table>

---

dbopbessely.xhtml

<table>
<thead>
<tr>
<th>dbopbessely.xhtml</th>
</tr>
</thead>
<tbody>
<tr>
<td>standard head</td>
</tr>
<tr>
<td>body</td>
</tr>
<tr>
<td>page head</td>
</tr>
<tr>
<td>dbopbessely not implemented</td>
</tr>
<tr>
<td>page foot</td>
</tr>
</tbody>
</table>

---

dbopbeta.xhtml

<table>
<thead>
<tr>
<th>dbopbeta.xhtml</th>
</tr>
</thead>
<tbody>
<tr>
<td>standard head</td>
</tr>
<tr>
<td>body</td>
</tr>
<tr>
<td>page head</td>
</tr>
<tr>
<td>dbopbeta not implemented</td>
</tr>
<tr>
<td>page foot</td>
</tr>
</tbody>
</table>

---

dbopcardinalnumber.xhtml

<table>
<thead>
<tr>
<th>dbopcardinalnumber.xhtml</th>
</tr>
</thead>
<tbody>
<tr>
<td>standard head</td>
</tr>
<tr>
<td>body</td>
</tr>
<tr>
<td>page head</td>
</tr>
<tr>
<td>dbopcardinalnumber not implemented</td>
</tr>
<tr>
<td>page foot</td>
</tr>
</tbody>
</table>

---
dbopchebyshevt.xhtml

— dbopchebyshevt.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
dbopchebyshevt not implemented
\getchunk{page foot}

———

dbopchebyshevu.xhtml

— dbopchebyshevu.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
dbopchebyshevu not implemented
\getchunk{page foot}

———

dbopcoefficient.xhtml

— dbopcoefficient.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopcoefficient not implemented
\getchunk{page foot}

———

dbopcoefficients.xhtml

— dbopcoefficients.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopcoefficients not implemented
\getchunk{page foot}

———
dbopcoerce.xhtml

---

\getchunk{standard head}
\getchunk{page head}
dbopcoerce not implemented
\getchunk{page foot}

---

dbopcolumn.xhtml

---

\getchunk{standard head}
\getchunk{page head}
dbopcolumn not implemented
\getchunk{page foot}

---

dbopcompactfraction.xhtml

---

\getchunk{standard head}
\getchunk{page head}
dbopcompactfraction not implemented
\getchunk{page foot}

---

dbopcomplexeigenvectors.xhtml

---

\getchunk{standard head}
\getchunk{page head}
dbopcomplexeigenvectors not implemented
\getchunk{page foot}

---
dbopcomplexelementary.xhtml

---

dbopcomplexelementary.xhtml

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopcomplexelementary not implemented
\getchunk{page foot}

---

dbopcomplexintegrate.xhtml

---

dbopcomplexintegrate.xhtml

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopcomplexintegrate not implemented
\getchunk{page foot}

---

dbopcomplexlimit.xhtml

---

dbopcomplexlimit.xhtml

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopcomplexlimit not implemented
\getchunk{page foot}

---

dbopcomplexsolve.xhtml

---

dbopcomplexsolve.xhtml

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopcomplexsolve not implemented
\getchunk{page foot}

---
dbopcontent.xhtml

---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopcontent not implemented
\getchunk{page foot}

---

dbopcontinuedfraction.xhtml

---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopcontinuedfraction not implemented
\getchunk{page foot}

---

dbopconvergents.xhtml

---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopconvergents not implemented
\getchunk{page foot}

---

dbopconvert.xhtml

---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopconvert not implemented
\getchunk{page foot}

---
dbopcopy.xhtml

— dbopcopy.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopcopy not implemented
\getchunk{page foot}

dbopcos.xhtml

— dbopcos.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopcos not implemented
\getchunk{page foot}

dbopcosh.xhtml

— dbopcosh.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopcosh not implemented
\getchunk{page foot}

dbopcot.xhtml

— dbopcot.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopcot not implemented
\getchunk{page foot}
dbopcoth.xhtml

--- dbopcoth.xhtml ---
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopcoth not implemented
\getchunk{page foot}

-----

dbopcount.xhtml

--- dbopcount.xhtml ---
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopcount not implemented
\getchunk{page foot}

-----

dbopcountableq.xhtml

--- dbopcountableq.xhtml ---
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopcountableq not implemented
\getchunk{page foot}

-----

dbopcreate3space.xhtml

--- dbopcreate3space.xhtml ---
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
 dbopcreate3space not implemented
\getchunk{page foot}

-----
dbopcyclotomic.xhtml

--- dbopcyclotomic.xhtml ---

```html
getchunk{standard head}
</head>
<body>
getchunk{page head}
dbopcyclotomic not implemented
getchunk{page foot}
```

---

dbopd.xhtml

--- dbopd.xhtml ---

```html
getchunk{standard head}
</head>
<body>
getchunk{page head}
dbopd not implemented
getchunk{page foot}
```

---

dbopdecimal.xhtml

--- dbopdecimal.xhtml ---

```html
getchunk{standard head}
</head>
<body>
getchunk{page head}
dbopdecimal not implemented
getchunk{page foot}
```

---

dbopdefiningpolynomial.xhtml

--- dbopdefiningpolynomial.xhtml ---

```html
getchunk{standard head}
</head>
<body>
getchunk{page head}
dbopdefiningpolynomial not implemented
getchunk{page foot}
```
dbopdegree.xhtml

— dbopdegree.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopdegree not implemented
\getchunk{page foot}

---------

dbopdenom.xhtml

— dbopdenom.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopdenom not implemented
\getchunk{page foot}

---------

dbopdraw.xhtml

— dbopdraw.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopdraw not implemented
\getchunk{page foot}

---------

dbopdeterminant.xhtml

— dbopdeterminant.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopdeterminant not implemented
\getchunk{page foot}

---------
dbopdiagonalmatrix.xhtml

— dbopdiagonalmatrix.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopdiagonalmatrix not implemented
\getchunk{page foot}

——

dbopdigamma.xhtml

— dbopdigamma.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopdigamma not implemented
\getchunk{page foot}

——

dbopdigits.xhtml

— dbopdigits.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopdigits not implemented
\getchunk{page foot}

——

dbopdimension.xhtml

— dbopdimension.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopdimension not implemented
\getchunk{page foot}

——
dboeigenvalues.xhtml

— dboeigenvalues.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
   dboeigenvalues not implemented
\getchunk{page foot}

———

dboeigenvector.xhtml

— dboeigenvector.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
   dboeigenvector not implemented
\getchunk{page foot}

———

dboeigenvectors.xhtml

— dboeigenvectors.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
   dboeigenvectors not implemented
\getchunk{page foot}

———

dbopelt.xhtml

— dbopelt.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
   dbopelt not implemented
\getchunk{page foot}

———
dbopequal.xhtml

<table>
<thead>
<tr>
<th>dbopequal.xhtml</th>
</tr>
</thead>
</table>
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopequal not implemented
\getchunk{page foot}

---

dbopeulere.xhtml

<table>
<thead>
<tr>
<th>dbopeulere.xhtml</th>
</tr>
</thead>
</table>
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopeulere not implemented
\getchunk{page foot}

---

dbopeulerphi.xhtml

<table>
<thead>
<tr>
<th>dbopeulerphi.xhtml</th>
</tr>
</thead>
</table>
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopeulerphi not implemented
\getchunk{page foot}

---

dbopeval.xhtml

<table>
<thead>
<tr>
<th>dbopeval.xhtml</th>
</tr>
</thead>
</table>
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopeval not implemented
\getchunk{page foot}

---
dbopevenq.xhtml

— dbopevenq.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopevenq not implemented
\getchunk{page foot}


dbopexp.xhtml

— dbopexp.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopexp not implemented
\getchunk{page foot}


dbopexquo.xhtml

— dbopexquo.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopexquo not implemented
\getchunk{page foot}


dbopfactor.xhtml

— dbopfactor.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopfactor not implemented
\getchunk{page foot}


dbopfactorfraction.xhtml

— dbopfactorfraction.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopfactorfraction not implemented
\getchunk{page foot}
——

dbopfibonacci.xhtml

— dbopfibonacci.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopfibonacci not implemented
\getchunk{page foot}
——

dbopfiniteq.xhtml

— dbopfiniteq.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopfiniteq not implemented
\getchunk{page foot}
——

dbopfirstdenom.xhtml

— dbopfirstdenom.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopfirstdenom not implemented
\getchunk{page foot}
——
dbopgcd.xhtml

— dbopgcd.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopgcd not implemented
\getchunk{page foot}

-----

dbophermiteh.xhtml

— dbophermiteh.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbophermiteh not implemented
\getchunk{page foot}

-----

dbophex.xhtml

— dbophex.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbophex not implemented
\getchunk{page foot}

-----

dbophorizconcat.xhtml

— dbophorizconcat.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbophorizconcat not implemented
\getchunk{page foot}

-----
dboplegendre.xhtml

— dboplegendre.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dboplegendre not implemented
\getchunk{page foot}

---

dboplength.xhtml

— dboplength.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dboplength not implemented
\getchunk{page foot}

---

dboblimit.xhtml

— dboblimit.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dboblimit not implemented
\getchunk{page foot}

---

dboplog.xhtml

— dboplog.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dboplog not implemented
\getchunk{page foot}

---
dboploggamma.xhtml

--- dboploggamma.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
dboploggamma not implemented
\getchunk{page foot}

---

dbopmainvariable.xhtml

--- dbopmainvariable.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
dbopmainvariable not implemented
\getchunk{page foot}

---

dbopmakegraphimage.xhtml

--- dbopmakegraphimage.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
dbopmakegraphimage not implemented
\getchunk{page foot}

---

dbopmakeobject.xhtml

--- dbopmakeobject.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
dbopmakeobject not implemented
\getchunk{page foot}

---
dbopmakeviewport3d.xhtml

--- dbopmakeviewport3d.xhtml ---
\getchunk{standard head}
\getchunk{page head}
dbopmakeviewport3d not implemented
\getchunk{page foot}

---

dbomap.xhtml

--- dbomap.xhtml ---
\getchunk{standard head}
\getchunk{page head}
dbomap not implemented
\getchunk{page foot}

---

dbomapbang.xhtml

--- dbomapbang.xhtml ---
\getchunk{standard head}
\getchunk{page head}
dbomapbang not implemented
\getchunk{page foot}

---

dbomatrix.xhtml

--- dbomatrix.xhtml ---
\getchunk{standard head}
\getchunk{page head}
dbomatrix not implemented
\getchunk{page foot}

---
dbopmax.xhtml

---
dbopmax.xhtml---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopmax not implemented
\getchunk{page foot}

---
dbopmemberq.xhtml

---
dbopmemberq.xhtml---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopmemberq not implemented
\getchunk{page foot}

---
dbopmin.xhtml

---
dbopmin.xhtml---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopmin not implemented
\getchunk{page foot}

---
dbopminimumdegree.xhtml

---
dbopminimumdegree.xhtml---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopminimumdegree not implemented
\getchunk{page foot}

---
CHAPTER 1. OVERVIEW

dbopminus.xhtml

— dbopminus.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopminus not implemented
\getchunk{page foot}

— dbopmoebiusmu.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopmoebiusmu.xhtml not implemented
\getchunk{page foot}

— dbopmonicdivide.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopmonicdivide not implemented
\getchunk{page foot}

— dbopmulmod.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopmulmod not implemented
\getchunk{page foot}
dbopncols.xhtml

— dbopncols.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopncols not implemented
\getchunk{page foot}

——

dbopnegativeq.xhtml

— dbopnegativeq.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopnegativeq not implemented
\getchunk{page foot}

——

dbopnew.xhtml

— dbopnew.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopnew not implemented
\getchunk{page foot}

——

dbopnextprime.xhtml

— dbopnextprime.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopnextprime not implemented
\getchunk{page foot}

——
dbopnorm.xhtml

— dbopnorm.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopnorm not implemented
\getchunk{page foot}

dbopnrows.xhtml

— dbopnrows.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopnrows not implemented
\getchunk{page foot}

dbopnthfractionalterm.xhtml

— dbopnthfractionalterm.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopnthfractionalterm not implemented
\getchunk{page foot}

dbopnthroot.xhtml

— dbopnthroot.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopnthroot not implemented
\getchunk{page foot}
dbopoperator.xhtml

--- dbopoperator.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopoperator not implemented
\getchunk{page foot}

---

dboporthonormalbasis.xhtml

--- dboporthonormalbasis.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dboporthonormalbasis not implemented
\getchunk{page foot}

---

dbopoutputfixed.xhtml

--- dbopoutputfixed.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopoutputfixed not implemented
\getchunk{page foot}

---

dbopoutputfloating.xhtml

--- dbopoutputfloating.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopoutputfloating not implemented
\getchunk{page foot}

---
CHAPTER 1. OVERVIEW

dbopnullspace.xhtml

    — dbopnullspace.xhtml —
    \getchunk{standard head}
    </head>
    <body>
    \getchunk{page head}
      dbopnullspace not implemented
    \getchunk{page foot}

    — dbopnumberoffractionalterms.xhtml —
    \getchunk{standard head}
    </head>
    <body>
    \getchunk{page head}
      dbopnumberoffractionalterms not implemented
    \getchunk{page foot}

    — dboppartialfraction.xhtml —
    \getchunk{standard head}
    </head>
    <body>
    \getchunk{page head}
      dboppartialfraction not implemented
    \getchunk{page foot}

    — dboppartialquotients.xhtml —
    \getchunk{standard head}
    </head>
    <body>
    \getchunk{page head}
      dboppartialquotients not implemented
    \getchunk{page foot}
CHAPTER 1. OVERVIEW

dboppolygamma.xhtml

— dboppolygamma.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dboppolygamma not implemented
\getchunk{page foot}

——

dboppositiveq.xhtml

— dboppositiveq.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dboppositiveq not implemented
\getchunk{page foot}

——

dboppositiveremainder.xhtml

— dboppositiveremainder.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dboppositiveremainder not implemented
\getchunk{page foot}

——

dbopprefixragits.xhtml

— dbopprefixragits.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopprefixragits not implemented
\getchunk{page foot}

——
dbopprevprime.xhtml

— dbopprevprime.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
   dbopprevprime not implemented
\getchunk{page foot}

-----

dbopprimefactor.xhtml

— dbopprimefactor.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
   dbopprimefactor not implemented
\getchunk{page foot}

-----

dbopprimeq.xhtml

— dbopprimeq.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
   dbopprimeq not implemented
\getchunk{page foot}

-----

dbopprimes.xhtml

— dbopprimes.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
   dbopprimes not implemented
\getchunk{page foot}

-----
dboprealeigenvectors.xhtml

— dboprealeigenvectors.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dboprealeigenvectors not implemented
\getchunk{page foot}

— dboprealelementary.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dboprealelementary not implemented
\getchunk{page foot}  

— dbopreduce.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopreduce not implemented
\getchunk{page foot}

— dbopreductum.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopreductum not implemented
\getchunk{page foot}
dboprem.xhtml

— dboprem.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dboprem not implemented
\getchunk{page foot}

—

dbopquo.xhtml

— dbopquo.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopquo not implemented
\getchunk{page foot}

—

dbopresetvariableorder.xhtml

— dbopresetvariableorder.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopresetvariableorder not implemented
\getchunk{page foot}

—

dbopresultant.xhtml

— dbopresultant.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopresultant not implemented
\getchunk{page foot}

—
dboprootof.xhtml

---

dboprootof.xhtml

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dboprootof not implemented
\getchunk{page foot}

---

dboprootsimp.xhtml

---

dboprootsimp.xhtml

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dboprootsimp not implemented
\getchunk{page foot}

---

dboprootsof.xhtml

---

dboprootsof.xhtml

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dboprootsof not implemented
\getchunk{page foot}

---

dbopseries.xhtml

---

dbopseries.xhtml

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopseries not implemented
\getchunk{page foot}

---
CHAPTER 1. OVERVIEW

dbopseteltbang.xhtml

— dbopseteltbang.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopseteltbang not implemented
\getchunk{page foot}

———

dbopsetrowbang.xhtml

— dbopsetrowbang.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopsetrowbang not implemented
\getchunk{page foot}

———

dbopsetelt.xhtml

— dbopsetelt.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopsetelt not implemented
\getchunk{page foot}

———

dbopsetsubmatrixbang.xhtml

— dbopsetsubmatrixbang.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopsetsubmatrixbang not implemented
\getchunk{page foot}

———
dbopsign.xhtml

— dbopsign.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopsign not implemented
\getchunk{page foot}

———

dbopsimplify.xhtml

— dbopsimplify.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopsimplify not implemented
\getchunk{page foot}

———

dbopseriessolve.xhtml

— dbopseriessolve.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopseriessolve not implemented
\getchunk{page foot}

———

dbopsin.xhtml

— dbopsin.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopsin not implemented
\getchunk{page foot}

———
dbopsingleintegerand.xhtml

— dbopsingleintegerand.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
   dbopsingleintegerand not implemented
\getchunk{page foot}

-----

dbopsingleintegernot.xhtml

— dbopsingleintegernot.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
   dbopsingleintegernot not implemented
\getchunk{page foot}

-----

dbopsingleintegeror.xhtml

— dbopsingleintegeror.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
   dbopsingleintegeror not implemented
\getchunk{page foot}

-----

dbopsingleintegerxor.xhtml

— dbopsingleintegerxor.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
   dbopsingleintegerxor not implemented
\getchunk{page foot}

-----
dbopsec.xhtml

---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopsec not implemented
\getchunk{page foot}

---

dbopsech.xhtml

---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopsech not implemented
\getchunk{page foot}

---

dbopsetvariableorder.xhtml

---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopsetvariableorder not implemented
\getchunk{page foot}

---

dbopsinh.xhtml

---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dbopsinh not implemented
\getchunk{page foot}

---
dbopsolve.xhtml

— dbopsolve.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopsolve not implemented
\getchunk{page foot}

_______

dbopsqrt.xhtml

— dbopsqrt.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopsqrt not implemented
\getchunk{page foot}

_______

dbopstar.xhtml

— dbopstar.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopstar not implemented
\getchunk{page foot}

_______

dbopstarstar.xhtml

— dbopstarstar.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopstarstar not implemented
\getchunk{page foot}

_______
dbopsubmatrix.xhtml
—— dbopsubmatrix.xhtml ——

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopsubmatrix not implemented
\getchunk{page foot}

———

dbopsubmod.xhtml
—— dbopsubmod.xhtml ——

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopsubmod not implemented
\getchunk{page foot}

———

dbopsurface.xhtml
—— dbopsurface.xhtml ——

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopsurface not implemented
\getchunk{page foot}

———

dbopsumofkthpowerdivisors.xhtml
—— dbopsumofkthpowerdivisors.xhtml ——

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopsumofkthpowerdivisors.xhtml not implemented
\getchunk{page foot}

———
dboptotaldegree.xhtml

— dboptotaldegree.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dboptotaldegree not implemented
\getchunk{page foot}

———

dboptrace.xhtml

— dboptrace.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dboptrace not implemented
\getchunk{page foot}

———

dboptranspose.xhtml

— dboptranspose.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dboptranspose not implemented
\getchunk{page foot}

———

dboprigs.xhtml

— dboprigs.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
    dboprigs not implemented
\getchunk{page foot}

———
dboptruncate.xhtml

--- dboptruncate.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dboptruncate not implemented
\getchunk{page foot}

---

dbopvariables.xhtml

--- dbopvariables.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopvariables not implemented
\getchunk{page foot}

---

dbopvectorise.xhtml

--- dbopvectorise.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopvectorise not implemented
\getchunk{page foot}

---

dbopvectorspace.xhtml

--- dbopvectorspace.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopvectorspace not implemented
\getchunk{page foot}

---
CHAPTER 1. OVERVIEW

dbopvertconcat.xhtml

— dbopvertconcat.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopvertconcat not implemented
\getchunk{page foot}


dbopwholepart.xhtml

— dbopwholepart.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopwholepart not implemented
\getchunk{page foot}


dbpolynomialinteger.xhtml

— dbpolynomialinteger.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbpolynomialinteger not implemented
\getchunk{page foot}


dbpolynomialfractioninteger.xhtml

— dbpolynomialfractioninteger.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbpolynomialfractioninteger not implemented
\getchunk{page foot}
dbopwholeragits.xhtml

— dbopwholeragits.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
  dbopwholeragits not implemented
\getchunk{page foot}

-----

definiteintegral.xhtml

— definiteintegral.xhtml —

\getchunk{standard head}
<script type="text/javascript">
function commandline(arg) {
  var myform = document.getElementById("form2");
  var ans = 'integrate('+myform.expr.value+','+myform.vars.value+'='+
    myform.lower.value+'..'+myform.upper.value+');'
  return(ans);
}
\getchunk{showfullanswer}
\getchunk{axiom talker}
</script>
\getchunk{page head}
<form id="form2">
  Enter the function you want to integrate:<br/>
  <input type="text" id="expr" tabindex="10" size="50"
     value="1/(x^2+6)"/><br/>
  Enter the variable of integration:<br/>
  <input type="text" id="vars" tabindex="20" size="5" value="x"/><br/>
  Enter a lower limit:<br/>
  <input type="text" id="lower" tabindex="30" value="%minusInfinity"/><br/>
  Enter an upper limit:<br/>
  <input type="text" id="upper" tabindex="40" value="%plusInfinity"/><br/>
</form>
\getchunk{continue button}
\getchunk{answer field}
\getchunk{page foot}

-----
CHAPTER 1. OVERVIEW

determinantofhilbert.xhtml

— determinantofhilbert.xhtml —

\getchunk{standard head}
\getchunk{handlefreevars}
\getchunk{axiom talker}
</script>
</head>
<body>
\getchunk{page head}
Consider the problem of computing the determinant of a 10 by 10 Hilbert matrix. The (i,j)-th entry of a Hilbert matrix is given by $1/(i+j+1)$.

First do the computation using rational numbers to obtain the exact result.
<ul>
<li>
<input type="submit" id="p1" class="subbut"
onclick="makeRequest('p1');"
value="a:Matrix FRAC INT:=matrix [ [1/(i+j+1) for j in 0..9] for i in 0..9]"
/>
<div id="ansp1"><div></div></div>
</li>
</ul>
This version of <a href="dbopdeterminant.xhtml">determinant</a> uses Gaussian elimination.
<ul>
<li>
<input type="submit" id="p2" class="subbut"
onclick="makeRequest('p2');"
value="d:=determinant a"
/>
<div id="ansp2"><div></div></div>
</li>
<li>
<input type="submit" id="p3" class="subbut"
onclick="handleFree(['p2','p3']);"
value="d::Float"
/>
<div id="ansp3"><div></div></div>
</li>
</ul>
Now use hardware floats.
<ul>
<li>
<input type="submit" id="p4" class="subbut"
onclick="makeRequest('p4');"
value="b:Matrix DFLOAT:=matrix [ [1/(i+j+1$DFLOAT) for j in 0..9] for i in 0..9]"
/>
<div id="ansp4"><div></div></div>
</li>
</ul>
The result given by hardware floats is correct to only four significant digits of precision. In the jargon of numerical analysis, the Hilbert matrix is said to be “ill-conditioned”.

Now repeat the computation at a higher precision using<br>
<a href="db.xhtml?Float">Float</a>

Reset <a href="dbopdigits.xhtml">digits</a> to its default value.
differentiate.xhtml

--- differentiate.xhtml ---

```javascript
function commandline(arg) {
  var myform = document.getElementById("form2");
  return ('differentiate('+myform.expr.value+','+[myform.vars.value],[+myform.powers.value+']));
}
```

--- dlmf.xhtml ---

```html
The Gamma function is an extension of the factorial function to real and complex numbers. For positive integers,
```
These pages explore Axiom's facilities for handling the Gamma function. In particular we try to show that Axiom conforms to published standards.
CHAPTER 1. OVERVIEW

 CHAPTER 1. OVERVIEW

 Mathematical Applications

 Physical Applications

 Computation

 Methods of Computation

 Tables

 Approximations

 Axiom Software

 ---

 Rational Approximations

 Expansions in Chebyshev Series

 Approximations in the Complex Plane

 Rational Approximations

 Cody and Hillstrom (1967)

 gives minimax rational approximations for

 \[ \ln \eta \]

 &copy; 2010 NIST/SAIC
for the ranges
\begin{align*}
\sin^{-1}(x) & \leq 1.5 & 0.5 \leq x \leq 1.5, \\
\sin^{-1}(x) & \leq 4 & 1.5 \leq x \leq 4, \\
\sin^{-1}(x) & \leq 12 & 4 \leq x \leq 12;
\end{align*}
precision is variable.
Hart \textit{et.al.} (1968)
gives minimax polynomial and rational approximations to
\begin{align*}
\sin^{-1}(x) & = \frac{\pi}{2} - \ln(\sqrt{1-x^2} + x), \\
\ln(\sin^{-1}(x)) & = \frac{\pi}{2} - \ln(x + \sqrt{1-x^2}),
\end{align*}
in the intervals

\[
\begin{align*}
0 \leq x & \leq 1, \\
8 \leq x & \leq 1000, \\
12 \leq x & \leq 1000.
\end{align*}
\]

precision is variable.

Cody et al. (1973) gives minimax rational approximations for

\[0.5 \leq x \leq 3\] and
\[3 \leq x < \infty;\] precision is variable.

For additional approximations see
Hart et al. (1968) (Appendix B),
Luke (1975) (pp. 2223), and

Expansions in Chebyshev Series

gives the coefficients to 20D for the Chebyshev-series expansions of
\[
\frac{1}{\alpha (1 + x)}.
\]
\[
\alpha (x + 3).
\]
\[
\ln \alpha (x + 3).
\]
\[
\theta.
\]
\( (x + 3) \), and the first six derivatives of

\[
\psi(x + 3)
\]

for

\[
0 \leq x \leq 1
\]

These coefficients are reproduced in

Luke(1975)

Clenshaw(1962)

also gives 20D Chebyshev-series coefficients for

\[
\psi(1 + x)
\]
and its reciprocal for

\[
\frac{\frac{\gamma}{(z+1)}}{A(z)}
\]

where

\[
A \frac{\gamma}{(z+1)}
\]


Approximations in the Complex Plane

Rational approximations for

\[
\frac{\frac{\gamma}{(z+1)}}{A(z)}
\]

where
\[ x = \left( \frac{1}{2} \right)^{z+c+\frac{1}{2}} \exp \left( -\left( z + c + \frac{1}{2} \right) \right) \]
and approximations for
\[ \gamma(z+1) \]

, and approximations for
\[ \gamma(z) \]

based on the Pad approximants for two forms of the incomplete gamma function are in


Luke(1975)

(pp. 1316) provides explicit rational approximations for
\[ \gamma(z+1) \]

\[ \gamma(z) \]
As
\[ z \to \infty \text{ in the sector } \left| \phi z \right| \leq \pi - \delta, \]
\[ z \to \infty \text{ in the sector } \left| \phi z \right| \leq \pi - \delta, \]
\[
\ln (\lambda (z)) = \ln |z| - \frac{1}{2} |z|
\]
\[ z + \frac{1}{2} \ln(2\pi) + \sum_{k=1}^{\infty} \frac{B_{2k}}{2k(2k-1)} \]
\[
\mathcal{D} (z) \leq \ln z - \frac{1}{2z} - \sum_{k=1}^{2} \frac{1}{z^k} 
\]
\[
\sum_{k=1}^{\infty} \frac{B_{2k}}{2k} z^{2k} = \frac{z}{e^z - 1}.
\]

For the Bernoulli numbers
\[
B_{2k} = \sum_{k=1}^{\infty} \frac{B_{2k}}{2k} z^{2k},
\]
Also,
\[(m - z) \leq \left(\frac{2\pi}{m}\right)^\frac{1}{2} \left(\sum_{k=0}^{\infty} \frac{g_k}{k}\right)\]
\[
\frac{z}{k}\frac{1}{12}\frac{1}{288}
\]
\[ g_3 = -\frac{139}{51840}, \quad g_4 = -\frac{571}{2488320}, \quad g_5 = \frac{163879}{209018880}. \]
\[
\begin{align*}
\text{If } & \quad g_6 = \frac{5246819}{75246796800}, \\
\text{and } & \quad g_k = \sqrt{2} \left( \frac{1}{2} \right)^k a_{2k},
\end{align*}
\]
where
\[
\begin{align*}
a_0 & = \frac{1}{2} \sqrt{2} \\
a_0 + \frac{1}{2} \sqrt{2} & = a_k
\end{align*}
\]
and
\[ a_k - 1 + \frac{1}{3} a_{k-2} + \cdots + \frac{1}{k+1} a_k + 0 \]
\[
\frac{1}{k}a_k = a_{k-1} \quad k \geq 1.
\]

Wrench (1968) gives exact values of \( g_k \) up to \( g_{20} \). Spira (1971) corrects errors in Wrench's results and also supplies exact and 45D values of
For an asymptotic expansion of $g_k$ as $k \to \infty$ see Boyd (1994). With the same conditions
\[
\sqrt{2\pi} \lesssim \gamma = \frac{a + b - (1/2)}{a + b + (1/2)}
\]
where 
\[ a > 0 \]
and 
\[ b \notin \mathbb{P} \]
are both fixed, and

\[ \ln(\theta(z + h)) \]
\[
\log z + h - \frac{1}{2} \log (2\pi) + \frac{1}{2} \log (z + h - \frac{1}{2}) - z - \frac{1}{2} \log (2\pi) + \frac{1}{2} \log (z - \frac{1}{2})
\]
\[ \sum_{k=2}^{\infty} \frac{(-1)^k B_k(h)}{k(k-1)} \]
where \( h \in [0, 1) \) is fixed.

Also as

\[
\begin{align*}
\psi & \rightarrow \mathbb{R}^\infty \\
|\theta|
\end{align*}
\]

\[(x + \varepsilon) \mid \sqrt{\frac{1}{2\pi}} \leq |y| \leq \varepsilon (\frac{1}{2}) \leq \frac{\pi}{|y|} \]
CHAPTER 1. OVERVIEW

uniformly for bounded real values of $x$.

Error Bounds and Exponential Improvement

If the sums in the expansions (Equation 1) and (Equation 2) are terminated at

$$k = n - 1$$

and

$$z$$

is real and positive, then the remainder terms are bounded in magnitude by the first neglected terms and have the same sign. If

$$z$$

is complex, then the remainder terms are bounded in magnitude by

$$\sec^2$$
\[ n^{\left(\frac{1}{2}\phi_z\right)} \]

for (Equation 1), and

\[ \sec^2(n^2 + 1)\left(\frac{1}{2}\phi_z\right) \]

for (Equation 2), times the first neglected terms.

For the remainder term in
(\text{Equation 3}) write

\begin{align*}
\gamma(z) &= \gamma - z^2 \left( \frac{2}{\pi z} \right) \left( \sum_{k=1}^{\infty} \frac{1}{2^{k+1}} \right)
\end{align*}
\[
\sum_{K \in \{1, 2, 3, \ldots\}} \left( g_2 \left( Z \right) \right)
\]
Then

\[
|\mathbb{R}_K(z)| \leq \frac{(1 + \beta(K))}{\beta(K)}
\]
\[
\frac{2}{K + 1} \left(1 + \min \left(\sec \left(\phi(z)\right), 2^{\left|z\right| K + 1}\right)\right)
\]
<m:mi>\text{K}\</m:mi>
<m:mstyle scriptlevel="+1">
<m:mfrac>
<m:mn>1</m:mn>/\m:mn>2
</m:mfrac>
</m:mstyle>
</div>

</div align="right">
<m:math display="inline">
<m:mrow>
<m:mo>|</m:mo><m:mrow>
<m:mi>\text{ph}</m:mi><m:mi>z</m:mi><m:mo>|</m:mo><m:mrow>
<m:mi>&lt;</m:mi><m:mrow>
<m:mfrac>
<m:mn>1</m:mn>/\m:mn>2
</m:mfrac>
</m:mrow>
</m:mrow>
</m:mrow>
</m:math>
</div>

</h4>Ratios</h4>
<p>If
</p>
<m:math display="inline">
<m:mrow>
<m:mi>a</m:mi><m:mrow>
<m:mo>(</m:mo>
</m:mrow>
</m:mrow>
</m:math>
and

\( b \in \mathbb{R} \) are fixed as

\( z \rightarrow \infty \) in

\[ |\text{ph} z| \leq \pi - \delta (\phi < \pi) \]
Then

\[
\frac{\theta(z + a)}{\theta(z + b)} \leq z^{a - b}
\]
\[
\frac{\gamma(z + a)}{\gamma(z + b)} \leq z^{a-b} \sum_{k=0}^{\infty} G_k
\]
Also, with the added condition
}\[ |b-a| > 0, \]

Also, with the added condition
\[
|\frac{\theta}{z^k}| > 0,
\]
<m:mrow><m:mo>+</m:mo><m:mi>a</m:mi></m:mrow><m:mrow><m:mo>)</m:mo></m:mrow></m:mrow><m:mrow><m:mi mathvariant="normal">\(\alpha\)</m:mi><m:mrow><m:mo>(</m:mo><m:mrow><m:mi>z</m:mi><m:mo>+</m:mo><m:mi>b</m:mi></m:mrow><m:mo>)</m:mo></m:mrow></m:mrow></m:mrow><m:mfrac><m:mrow><m:msup><m:mrow><m:mo>(</m:mo><m:mrow><m:mi>z</m:mi><m:mo>+</m:mfrac><m:mrow><m:mrow><m:mi>a</m:mi><m:mo>+</m:mo><m:mi>b</m:mi></m:mrow><m:mo>-</m:mo><m:mn>1</m:mn></m:mrow><m:mo>/</m:mo><m:mn>2</m:mn></m:mfrac></m:mrow><m:mo>)</m:mo></m:mrow><m:mo>)</m:mo></m:msup><m:mrow><m:msup><m:mrow><m:mo>(</m:mo><m:mrow><m:mi>z</m:mi><m:mo>+</m:mfrac><m:mrow><m:mi>a</m:mi><m:mo>-</m:mo><m:mi>b</m:mi></m:mrow></m:mfrac></m:mrow><m:mo>)</m:mo></m:msup></m:mrow></m:mfrac><m:mrow><m:munderover><m:mo movablelimits="false">\(\sum\)</m:mo><m:mrow><m:mi>k</m:mi></m:mrow></m:munderover></m:mrow><m:mrow><m:mo movablelimits="false">\(\sum\)</m:mo><m:mrow><m:mi>k</m:mi></m:mrow></m:mrow>
\[ \sum_{k} \left( \frac{H_k(x, y)}{(z + \frac{1}{2}(a + b - 1))} \right)^2 \]
Here

\[
G_0(a,b) = 1,
\]

\[
G_1(a,b) = \frac{a}{b}.
\]
\[
\frac{1}{2}(a - b)(a + b - 1)
\]

\[
G_2(a, b) = \frac{1}{2}(a - b)(a + b - 1)
\]
\[
\frac{1}{12} (a - b) (3(a^2 - b^2) - (a - b))
\]
\begin{center}
\begin{align*}
H_0(a,b) &= 1, \\
H_1(a,b) &= \frac{1}{b} + 1.
\end{align*}
\end{center}
a - b = \frac{1}{12} (a - b + 1) - (a - b - 1)
$$H(a, b) = \frac{1}{240}(a - b)^4$$
In terms of generalized Bernoulli polynomials we have for

\[ G_k(a, b) = (a - b, k) \]
\[(a-b+1)(a)\]
\[ \left( B^2 \right)^{k} \left( (a - b + 1) \right) \left( \frac{a - b + 1}{2} \right) \]
\[
\frac{\zeta(z+a)}{\zeta(z+b)} \frac{\zeta(z+c)}{\text{\textcircled{1}}}
\]
\[ k = 0 \sum_{k} \left( \left( -1 \right)^k \right) \frac{(c - a)^k}{k!} \frac{(c - b)^k}{k!} \]
\[
(a + b - c + z - k) \quad dlmfbarnesgfunction.xhtml
\]
dlmfbarnesgfunction.xhtml
— dlmfbarnesgfunction.xhtml —

\[
(a + b - c + z - k) \quad dlmfbarnesgfunction.xhtml
\]

<math display="inline">
\( (a + b - c + z - k) \) 
</math>

The Gamma Function -- Barnes G-Function (Double Gamma Function)

<math display="inline">
\( \Gamma(a, b, c, z, k) \) 
</math>

Barnes

<math display="inline">
\( \Gamma(a, b, c, z, k) \) 
</math>

Double Gamma Function
\[ G(z+1) = \bar{\gamma}(z) G(z) \]
\[ G(n) = (n-2)! \left( n - \frac{3}{n} \right)! \]
\[ n = 2, 3, \ldots \]

\[ G(z + 1) = \left( 2\pi \right)^{\frac{z}{2}} \exp \left( -z \right) \]
\[ \frac{1}{2} \cdot z (z + 1) - \frac{1}{2} \cdot \beta z^2 \times \sum_{k=1}^{\infty} \]
$\left(1 + \frac{z}{k}\right)^k \exp\left(-z + \frac{z^2}{2k}\right)$
\[
\frac{1}{2}z \ln (2\pi) - \frac{1}{2}z (z + 1)
\]
\[
\begin{align*}
\ln z & = \ln \left( z + 1 \right) \quad + \quad \int_0^z \ln \left( t + 1 \right) \, dt \\
\end{align*}
\]
The \( \text{Ln} \)'s have their principal values on the positive real axis and are continued via continuity.

When \( z \rightarrow \infty \) in
\[
|\phi z| \leq \pi - \delta (\lt \pi)
\]
\[
\ln G(z) = \ln \left( \frac{1}{4} z^2 + z + \pi (z + 1) \right)
\]
\[
\frac{1}{2} z \left( z + 1 \right) + \frac{1}{12} \right)

\ln z - \ln A + \sum_{i=1}^{\infty} \frac{1}{i^2}
\]
\[
\sum_{k=1}^{\infty} \frac{B_{2k}}{2^{k+1} (2k)!} \left( \frac{2}{k+1} \right) \left( \frac{2}{k+2} \right)^2 \left( \frac{2}{k} + 1 \right) \left( \frac{2}{k+2} \right)^2 \left( \frac{2}{k} \right)^2 z^{2k}
\]
see Ferreira and Lpez(2001). This reference also provides bounds for the error term. Here

$$\frac{1}{2}B_{2k} + \frac{1}{2}A$$ is the Bernoulli number, and

$$A = \frac{1}{\zeta(2)} = 1.28242 71291 00622 63687 \ldots$$

where

$$\zeta(2) = 1.28242 71291 00622 63687 \ldots$$

is Glaisher's constant, given by
$C = \lim_{n \to \infty} \left( \sum_{k=1}^{n} \ln k - \left( \frac{1}{2} n^2 + \frac{1}{2} \right) \right)
\[
\frac{1}{2n} + \frac{1}{12n} > \ln n + \frac{1}{4n^2} = \gamma + \ln (n^{1/2})
\]
\[
\frac{2\pi - \frac{1}{12}}{\frac{2\pi}{4} - (2^{-1})} = \frac{1}{12} - \frac{1}{8}
\]
and
\( \alpha' \) is the derivative of the zeta function

For Glaisher's constant see also

In this section all fractional powers have their principal values, except where noted otherwise. In the next 4 equations it is assumed

\[ a > 0 \] and
\[ a \neq 0 \]
\[ b > 0 \]

Euler's Beta Integral

\[
\begin{align*}
B(a, b) &= \int_0^1 t^{a-1} (1-t)^{b-1} \, dt
\end{align*}
\]
\[ b - 1 \frac{\gamma(a)}{\gamma(b)} = \frac{\gamma(a + b)}{\gamma(a)\gamma(b)} \]
\[ \int_{0}^{\frac{\pi}{2}} \left( \sin^{2}a \cos^{2}b - 1 \right) \, \mathrm{d}\theta = \frac{1}{2} \mathcal{B}(1) \]
\[ \int_{0}^{\infty} \frac{t}{a+bt} \, dt \]
\[
\frac{\int_0^1 t^{a-1} (1-t)^{b-1}}{B(a,b)}
\]
\[
\frac{\int t\,dt + z}{a + b} = B(a, b) - a(1 + z)
\]
\[ z - b \] with
\[ |\text{ph}z| < \pi \]
and the integration path along the real axis.
\[ a - 1 \cos(bt) \, \text{d}t = \frac{\pi}{2a} \frac{1}{aB(\frac{1}{2}(a + b + 1))} \]
\[ \frac{1}{2} \left( (a - b + 1) + 1 \right) \]

\[
\begin{align*}
\text{if } a &> 0, \\
\int_a^{b+1} &
\end{align*}
\]
\[ \frac{\pi}{2^{a-1}} \cdot \frac{\sin(t) - 1}{b} \times \frac{\pi}{b} = \frac{\pi}{2^{a-1} \cdot \frac{\sin(t) - 1}{b}} \]
\[
\frac{2}{a B (a + b + 1)} = \frac{1}{a (a - b + 1)}
\]
\[
\int_0^\infty \frac{\cosh(2bt)}{(\cosh t)^2} \, dt > 0
\]
\[ \frac{d}{dt} = 4a - 1B(a + b) - (a - b) \]
\[
\left| \int_{\infty}^{\infty} \frac{1}{2\pi} \left( w + \frac{\partial}{\partial t} \right) a \right| \]

\[ b = \left( \frac{(w + z)}{1 - a - b} \right) B \]
\[(a, b)\]

\[\sum_{i=1}^{n} a_i \geq 1, \quad \sum_{i=1}^{n} b_i > 0, \quad \sum_{i=1}^{n} c_i > 0\]
The fractional powers have their principal values when \( w > 0 \) and \( z > 0 \), and are continued via continuity.
\[ P(t-a) (< (1-t) - 1 - b) \, \mathbb{1} = \frac{1}{b \, B(a, b)} \]
0 < c < 1,
\( \| (a + b) > 0 \)
\( \frac{1}{2\pi} \int_0^\infty \)
<m:mn>1</m:mn><m:mo>+</m:mo><m:msubsup><m:mrow><m:mi>a</m:mi><m:mo>-</m:mo><m:mn>1</m:mn></m:mrow><m:mn>1</m:mn><m:mn>1</m:mn><m:mo>)</m:mo></m:msubsup><m:msup><m:mrow><m:mi>b</m:mi><m:mo>-</m:mn>1</m:mn></m:mrow><m:mn>1</m:mn></m:msup><m:mrow><m:mi mathvariant="normal">\&</m:mi><m:mi>\text{\textcopyright}</m:mi><m:mi>t</m:mi></m:mrow><m:mo>=</m:mo><m:mfrac><m:mrow><m:mi>\sin</m:mi><m:mrow><m:mo>(</m:mo><m:mrow><m:mi>\pi</m:mi><m:mi>b</m:mi></m:mrow><m:mo>)</m:mo></m:mrow></m:mrow><m:mi>\pi</m:mi></m:mfrac><m:mi>B</m:mi>
In the next two equations the fractional powers are continuous on the integration paths and take their principal values at the beginning.
\[
\frac{2\pi a}{-1} \int_0^{1} (0 + t) \left( 1 + t \right) - a - b \, dt
\]
\[
B(a,b) = B(\beta, \alpha) > 0, \quad \text{when } a \text{ is not an integer and the contour cuts the real axis between } -1 \text{ and the origin.}
\]

\[
\text{Contour for second loop integral for the beta function.}
\]
Pochhammer’s Integral

When

\[a, b \in \mathbb{R}\]

\[
\int_{P}^{t} (1 + \frac{0}{1 - 0}) \, dt
\]

\[
= \int_{a}^{b} \, dt
\]
\[
\frac{1}{b-1} \frac{\mathrm{d}t}{= \frac{-4}{\pi} (a+b)} \sin (\pi a)
\]
where the contour starts from an arbitrary point $P$ in the interval $(0, 1)$, circles $0$ and then $1$ in the positive sense, circles
and then in the negative sense, and returns to \( P \). It can always be deformed into the contour shown here.

\[ t \]-plane. Contour for Pochhammer’s integral.

---

dlmfcontinuedfractions.xhtml

--- dlmfcontinuedfractions.xhtml ---

For

\[
\frac{1}{z + a}
\]
\[
\ln \gamma(z) + z - (z - \frac{1}{2}) \ln z - \frac{1}{2} \ln z
\]
\[(2\pi) = a_0 + z + a_1 + z + a_2 + z\]
\[
\frac{a}{3} + z + \frac{a}{4} + \frac{a}{5} + \text{E}
\]
\[
\begin{align*}
  a_0 &= \frac{1}{12}, \\
  a_1 &= \frac{1}{30}, \\
  a_2 &= \frac{53}{210}, \\
  a_3 &= \frac{210}{53}.
\end{align*}
\]
\[ \frac{195}{371} \]

\[ \frac{22999}{22737} \]

\[ \frac{299\,44523}{197\,33142} \]

\[ \frac{10\,953\,524\,109}{4\,826\,427\,562} \]
For rational values of $a_{7}$ to $a_{11}$ and 40S values of $a_{0}$ to $a_{40}$, see Char(1980). Also see Jones and Thron(1980) (pp. 348350) and Lorentzen and Waadeland(1992) (pp. 221224) for further information.
The Gamma Function -- Definitions

Gamma and Psi Functions
Euler's Constant
Pochhammer's Symbol

Gamma and Psi Functions
Euler's Integral

\[ \Gamma(z) = \int_0^\infty t^{z-1} e^{-t} \, dt \]
When \[ |z| > 0 \], \( \frac{\pi}{\sin(z)} \) is defined by analytic continuation. It is a meromorphic function with no zeros, and with simple poles of residue
\[ z = -n \]

\[ \frac{1}{\xi(z)} \]

is entire, with simple zeros at

\[ z = -n \]

\[ \psi(z) = \]

\[ \xi(z) = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]

\[ = \]
\[
\frac{\frac{\alpha'(z)}{z}}{\frac{\alpha(z)}{z}} = z \\
\text{for } y \neq 0, -1, -2, \ldots
\]
\[(z)\] is meromorphic with simple poles of residue 
\[-1\] at \[z = -n\].

\section*{Euler's Constant}
\begin{equation}
\alpha = \lim_{n \to \infty} \left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right)
\end{equation}
\[
\left( n^\frac{1}{n} - \ln n \right) = 0.57721 56649 01532 86060 \ldots
\]

Pochhammer's Symbol
\[
(\sqrt{a+n})^2 = (\sqrt{a+1})(\sqrt{a+2}) \cdot (\sqrt{a+n-1})
\]
\[
\begin{align*}
\log\left(\frac{\log(a+n)}{\log(a)}\right) &= \log\left(\frac{\log(a+n)}{\log(a)}\right) \\
\log(a) &= \log(a) \\
-\frac{n}{n-1} &< a < -\frac{n}{n+1}
\end{align*}
\]
In the Gamma Function, the recurrence relation
\[
\Gamma(z) \Gamma(z+1) = z\Gamma(z)
\]
is given, where \( \Gamma(z) \) is the Gamma function.
$z(\theta(z + 1)) = \theta(z) + \frac{1}{z}$
Reflection

\[
\begin{align*}
\frac{\pi}{\sin(\pi z)} &= \frac{\pi}{\sin(\pi (1-z))}
\end{align*}
\]

\[z \neq 2\pi m + \pi
\]
$0, 1, \ldots$
\[
\begin{align*}
\text{Multiplication} & : \\
\frac{2}{z} & \neq 0, -1, -2, \ldots
\end{align*}
\]
\[
\frac{\alpha (2z)}{= \frac{\pi}{2} - \frac{1}{2} \frac{2}{z} - 1 \alpha (z)}
\]
\[ \text{PAGES} \]

\[ + \frac{1}{2} ) \]

\[ 3 \neq 0, -1, -2, \ldots \]

\[ \text{\( \rho \)}(3 \]
\[
\z \cdot (\frac{1}{2})^{2 - \gamma} \cdot \frac{3^z - (\frac{1}{2})^2}{\gamma(z + 1)}
\]
\[ \frac{1}{3} \sqrt{\gamma(z + \frac{2}{3})} \]

\[ n \neq 0, -1, -2 \]
\[
\vartheta(n^z(nz)) = \left(2\pi\right)^n \frac{1}{2^n (1 - n)} \frac{n - nz}{n^z}.
\]
\[(1)\] 
\[\sum_{k=0}^{n-1} \gamma(z + \frac{k}{n})\]
378

CHAPTER 1. OVERVIEW

$$\sum_{i=0}^{n-1} \alpha_k(\frac{k}{n}) = (2\pi)^\frac{(n-1)}{2} (\frac{n}{2})$$
\[ \dot{\phi}(2z) = \frac{1}{2} (\phi(z) + \phi(z + \frac{1}{2})). \]
\[
\begin{align*}
\Phi(nz) &= \frac{1}{n} \sum_{k=0}^{n-1} \Phi(k) \\
\end{align*}
\]
(z + \frac{k}{n}) + \ln(n)

Bohr-Mollerup Theorem

If a positive function $f(x)$ on $(0, \infty)$ satisfies
\[
(f(x + 1)) = f(x)\]

and

\[
(f(1)) = 1,
\]

and

\[
\ln(f(x))
\]

is convex, then
\( f(x) = \gamma(x) \).
CHAPTER 1. OVERVIEW

\[ \log(x) \quad \text{and} \quad \frac{1}{\log(x)} \]

To create these two graphs in Axiom:

```axiom
-- Draw the first graph in a viewport
viewport1:=draw(Gamma(i), i=-4.2..4, adaptive=true, unit=[1.0,1.0])
-- Draw the second graph in a viewport
viewport2:=draw(1/Gamma(i), i=-4.2..4, adaptive=true, unit=[1.0,1.0])
-- Get the Gamma graph from the first viewport and layer it on top
putGraph(viewport2,getGraph(viewport1,1),2)
-- Remove the points and leave the lines
points(viewport2,1,"off")
points(viewport2,2,"off")
-- Show the combined graph
makeViewport2D(viewport2)
```

\[ \ln(\log(x)) \]

This function is convex on
\( (0, \infty) \) \; \text{compare <a href="dlmffunctionalrelations.xhtml#bohrmolleruptheorem">Functional Relations</a>}

You can construct this graph with the Axiom commands:

```
-- draw the graph of log(Gamma) in a viewport
viewport1:=draw(log Gamma(i), i=0..8, adaptive==true, unit==[1.0,1.0])
-- turn off the points and leave the lines
points(viewport1,1,"off")
```

<h4>The Psi Function</h4>

This function is a special case of the polygamma function. In particular,

\[ \psi(x) \] is equal to polygamma(0,x).

![psi.png](bitmaps/psi.png)
You can reconstruct this graph in Axiom by:

```axiom
-- first construct the psi function
psi(x)==polygamma(0,x)
-- draw the graph in a viewport
viewport:=draw(psi(y),y=-3.5..4,adaptive==true)
-- make the gradient obvious
scale(viewport,1,0.9,22.5)
-- and recenter the graph
translate(viewport,1,0,-0.02)
-- turn off the points and keep the line
points(viewport,1,"off")
```

<h4>Complex Argument</h4>

You can reconstruct this image in Axiom with:

```axiom
-- Set up the default viewpoint
viewPhiDefault(-%pi/4)
-- define the point set function
gam(x,y)==
g:=Gamma complex(x,y)
 point [x,y,max(min(real g,4),-4), argument g]
-- draw the image and remember the viewport
viewport:=draw(gam, -4..4,-3..3,var1Steps==100,var2Steps==100)
-- set the color mapping for the image
colorDef(viewport,blue(),blue())
-- and smoothly shade it
drawStyle(viewport,"smooth")
```
You can reproduce this image from Axiom with:

```axiom
-- Set up the default viewpoint
viewPhiDefault(-%pi/4)
-- Define the complex Gamma inverse function
gaminv(x,y)==
g:=1/(Gamma complex(x,y))
point [x,y,max(min(real g,4),-4), argument g]
-- draw the 3D image and remember the viewport
viewport:=draw(gaminv, -4..4,-3..3,var1Steps==100,var2Steps==100)
-- make the image a uniform color
colorDef(viewport,blue(),blue())
-- and make it pretty
drawStyle(viewport, "smooth")
```

To get these exact images with the colored background you need to use GIMP to set the background. The steps I used are:

1. Save the image as a pixmap
2. Open the saved file in gimp
3. Dialogs->Colors->ColorPicker button
4. Eyedrop the color of the web page
Set the color as the foreground on the FG/BG page

Dialogs→Layers

Duplicate Layer

Layer→Stack→Select bottom layer

Edit→Fill with Foreground color

(on Layers panel)Select image

(on Layers panel) Mode→Darken Only

Note that you may have to use "lighten only" first before it will allow you to choose "darken only".

Inequalities

Contents

Real Variables

Complex Variables

Throughout this subsection

\[ x > 0 \]

Digital Library of Mathematical Functions

The Gamma Function -- Inequalities
\[ \frac{1}{2} \left( \frac{1}{2} \right)^2 - x \left( \frac{1}{2} \right) - x \frac{\mathrm{d}}{\mathrm{d}x} \left( \frac{1}{2} \right) \leq \frac{1}{2} \]
\[ (12x) \leq 2 \]

\[ \frac{1}{\theta(x)} + \frac{1}{\theta\left(\frac{1}{x}\right)} \leq 2 \]
\[
\frac{1}{(\beta(x))^2} + \frac{1}{(\beta(\frac{1}{x}))^2} \leq 2
\]
\[
\frac{x}{1-x} < \frac{\theta(x+1)}{\theta(x+s)} < (x+1)^{1-s}
\]
\[
0 < s < 1
\]

\[
\exp\left( \frac{1}{2} \left( -s + \varphi(x + \frac{s}{2}) \right) \right)
\]
\[
\frac{\lambda(x+1)}{\lambda(x+s)} \leq 
\exp\left(\frac{(1-s)\xi(x+m)}{1-s}\right)
\]
\[
\left(\frac{1}{2}\right)^{n+1} \leq s < 1
\]
\[
|\frac{\partial y}{\partial y}(x) - \|\alpha(x + \frac{\partial y}{\partial y})\| \geq 0
\]
(\text{sech}(\pi y))^2 \frac{1}{2} \sigma(x)

For $b - a$
\[ a \geq 0, \quad z = x + \sqrt{y} \quad \text{with} \quad x > 0, \]

\[
\lambda \left( z + a \right)
\]
\[ \frac{\sqrt{z + b}}{1} \leq \frac{|z|}{b - a} \]

For \( x \geq 0 \),
\[
\frac{1}{2}(2\pi)\leq|z|\leq|x-(\frac{1}{2})|\frac{\pi}{|y|}
\]
\begin{equation*}
\exp\left(\frac{1}{6} |z| - 1 \right)
\end{equation*}
The Gamma Function -- Infinite Products

Infinite Products

\[
\Gamma(z) = \lim_{k \to \infty} \frac{k!}{k^z (k+z-1)!}
\]
\[\sum_{k=0}^{\infty} \frac{1}{\theta(z)} = 1, \quad z \neq 0, -1, -2, \ldots\]
\[ z^{1/\lambda} \left( \frac{1}{1 + \frac{z}{k}} \right)^{1} |_k \]
\[
\frac{\varphi(x)\varphi(x + \gamma y)}{2} = \sum_{k=0}^{\infty} (1 + \frac{y^2}{(x+k)^2})
\]
\[
\sum_{k=1}^{m} a_k \neq 0, -1, \ldots
\]
\[ \sum_{k=1}^{m} b_k = \sum_{k=0}^{\infty} \left( a_1 + k \right) \]

\[ \frac{a_k}{m_{\text{sub}}} \]

\[ a_{n_{\text{sub}}} \]

\[ a_{m_{\text{sub}}} \]

\[ a_{k_{\text{sub}}} \]
$\left( \frac{a^2 + k}{m} \right) \cdot \left( \frac{a^m + k}{m} \right) \cdot \left( \frac{b^1 + k}{m} \right) \cdot \left( \frac{b^2 + k}{m} \right)$
\[
\frac{b^m + k}{\frac{\gamma (b_1)}{\gamma (b_2) \cdot \gamma (b_m)}}
\]
<m:mrow><m:mfrac><m:mn>1</m:mn><m:mrow><m:mrow><m:mi>a</m:mi><m:msub><m:mi>a</m:mi><m:mn>1</m:mn></m:msub></m:mrow><m:mrow><m:mi>m</m:mi><m:msub><m:mi>a</m:mi><m:mi>m</m:mi></m:msub></m:mrow></m:mrow></m:mfrac></m:mrow>

provided that none of the
<mm:math display="inline">b_k</mm:math>
is zero or a negative integer.
<div align="center">
<a href="http://dlmf.nist.gov">
Digital Library of Mathematical Functions</a>
</div>
<h3>Integrals</h3>

\[ \int_{c}^{c+\infty} \frac{1}{2\pi i} \, \mathrm{d}z \]

\[ = \frac{1}{\pi} \int_{c}^{c+\infty} \frac{1}{2\pi i} \, \mathrm{d}z \]

\[ = \frac{1}{\pi} \int_{c}^{c+\infty} \frac{1}{2\pi i} \, \mathrm{d}z \]
$$\gamma(s+a)(b-s)^{-z} = \gamma(a+b)^{-z}$$
$$a^\left(1+z\right)\frac{a+b}{\sqrt[n]{a+b}}>0,$$
\[ c < \frac{1}{2\pi}, \quad |ph| < \pi. \]

\[
\int_{-\infty}^{\infty} |\theta (r)| dr.
\]
\[ a + \frac{b}{2} \left( \sqrt{2 \pi} - 2a \right) t \]
\[
\sin b (2a/a^2) \frac{1}{2\pi} \quad (a > 0, \quad 0 < b < \pi).
\]

Barnes's Beta Integral
\[ \int_{-\infty}^{\infty} \theta(a + \theta t) \, dt \int_{-\infty}^{\infty} \theta(b + \theta t) \, dt \int_{-\infty}^{\infty} \theta(c - \theta t) \, dt \]
\[ \frac{\frac{\gamma'(d - \partial t)}{\gamma t}}{\gamma'(a + c)} = \frac{\gamma'(a + d)}{\gamma'(b + \cdots)} \]
CHAPTER 1. OVERVIEW

\[
\int_{-\infty}^{\infty} \frac{1}{\Gamma(a+t)\Gamma(b+t)} \, dt > 0
\]

Ramanujan’s Beta Integral

\[
\frac{1}{\Gamma(a+b)} = \int_{0}^{1} \frac{1}{\sqrt{x(1-x)}} \, dx
\]
\[
\frac{b + t}{\alpha(c - t)} = \frac{\alpha(a + b + c + d - 3)}{}
\]
\[ 
\frac{\lambda(a - 1)}{\lambda(a + c)} \cdot \frac{\lambda(a - 1)}{\lambda(a + d)} \cdot \frac{\lambda(b - 1)}{\lambda(b + c)} 
\]
\[(b + d - 1)\]

\[\left(\alpha + \beta + \gamma + \delta\right) > 3.\]

**de Branges-Wilson Beta Integral**

\[
\begin{align*}
\frac{1}{\pi^2} \int_0^1 \frac{x^a(1-x)^b}{(1+x)^{a+b+1}} \, dx
\end{align*}
\]
\[
\frac{1}{4\pi} \int_{a_k}^{a_k + \mathfrak{t}} \mathfrak{r}(a_k) \, da_k
\]
\[ \frac{\frac{\theta}{t}}{\frac{\theta}{t-2t}} = \begin{pmatrix} 1 & \leq \end{pmatrix} \]
\[ j < k \leq 4 \]

\[ \sum \left( a_j + a_k \right) \cdot \left( a_1 + a_2 + a_3 + a_4 \right) \]
\[ \Gamma(a_k) > 0, \quad k = 1, \ldots, 4. \]
CHAPTER 1. OVERVIEW

Integral Representations

Contents

- Gamma Function
- Psi Function and Euler's Constant

Gamma Function

\[
\frac{1}{\Gamma(z)} = \int_0^\infty e^{-t} \frac{1}{t^z} \, dt
\]
\[-z^{\mu} (t^{\nu} - 1) \text{d}t\]
430

<math display="block">
  <mrow>
    <mfrac>
      <mn>1</mn>
      <mi>\theta</mi>
    </mfrac>
    =
    <mfrac>
      <mn>1</mn>
      <mrow>
        <mn>2</mn>
        <mi>\pi</mi>
      </mrow>
      <msubsup>
        <mo>-</mo>
        <mrow>
          <mn>0</mn>
        </mrow>
        <mo>+</mo>
        <mi>t</mi>
      </msubsup>
    </mfrac>
  </mrow>
</math> (The fractional powers have their principal values.)

Hankel's Loop Integral

<math display="block">
  <mrow>
    <mfrac>
      <mn>1</mn>
      <mi>\theta</mi>
    </mfrac>
    =
    <mfrac>
      <mn>1</mn>
      <mrow>
        <mn>2</mn>
        <mi>\pi</mi>
      </mrow>
      <msubsup>
        <mo>-</mo>
        <mrow>
          <mn>0</mn>
        </mrow>
        <mo>+</mo>
        <mi>t</mi>
      </msubsup>
    </mfrac>
  </mrow>
</math>
\[ t - z \frac{\mathbf{i}}{t} \]

where the contour begins at \(-\infty\), circles the origin once in the positive direction, and returns to \(-\infty\).

\[ t^z \] has its principal value where \( t \) crosses the positive real axis, and is continuous.

Contour for Hankel's loop integral.
CHAPTER 1. OVERVIEW

\[
\begin{align*}
    c(z) &= \int_{-\infty}^{\infty} |t|^{2z-1} d\gamma
\end{align*}
\]
\[ t^2 \int c \, \mathrm{d}t > 0 \]

, \[ |z| > 0 \]

where the path is the real axis.

<p>where the path is the real axis.</p>
\[ \int_{1}^{\infty} t^{z-1} \frac{\mathcal{L}}{t} + \sum_{k=0}^{\infty} (\frac{-1}{k}) \]
\[ \binom{+k}{k} \times k! \]

\[
\begin{align*}
\mathcal{R}(z) &\neq 0, -1, -2, \ldots
\end{align*}
\]
\[ \int_0^{\infty} t^{z-1} \left( \left( \frac{-1}{t} \right)^{-k} \sum_{k=0}^{n} \frac{(-1)^k}{t^k} \right) \]
\[ \frac{1}{k!} \left( \int t^{\varpi} \right) \]

\[
\begin{align*}
- \, n - 1 < z < - \, n
\end{align*}
\]

\[
\begin{align*}
\frac{e^{-\lambda}}{\sqrt{2\pi\lambda}} & < \frac{e^{-\lambda}}{\sqrt{2\pi\lambda}} \\
\frac{e^{-\lambda}}{\sqrt{2\pi\lambda}} & < \frac{e^{-\lambda}}{\sqrt{2\pi\lambda}}
\end{align*}
\]
\[ \cos(\frac{1}{2} \pi z) = \frac{1}{\sqrt{t^2 - 1}} \cos t \, dt \]
\[0 < z < 1,\]

\[
\frac{\sin\left(z\right)}{2\pi z} = \int_{0}^{\infty} e^{-sz} \, ds.
\]
\[ t^{\text{sin}} t \sin t \]
\[
\int_{0}^{\infty} \cos(t^n) \, dt = \frac{1}{n} \cdot 2^n \cot \left( \frac{n\pi}{2} \right)
\]
\[ n = 2, 3, 4, \ldots \]
\[
\int_{0}^{\infty} \sin(t^n) \, dt
\]

\(n = 2, 3, 4, \ldots\).

**Binet’s Formula**

\[
\ln\Phi = \frac{\sqrt{5} - 1}{2} \quad \text{and} \quad \ln\phi = \frac{1 - \sqrt{5}}{2}
\]
\[ \mathcal{C}(z) = (z - \frac{1}{2}) \ln(z) - z + \frac{1}{2} \ln(2\pi); \]
\[
\int_{0}^{\infty} \frac{\arctan \left( \frac{t}{z} \right)}{\sqrt{2\pi t}} dt - 1
\]

where
and the inverse tangent has its principal value.
\[ \frac{1}{\pi z - s \sin(\pi s)} \]
where

\[ |\text{ph}z| \leq \pi - \delta \]

(\(<\pi\))
\(1 < c < 2\), and
\(\alpha(s)\),


### Psi Function and Euler's Constant

For
\[
\Psi(z) > 0
\]

\[
\Psi(z) = \ldots
\]
\[
\int_0^\infty \left( \frac{1}{t} - \frac{1}{t+z} \right) \, dt
\]
φ(z) = ln z + \int_{0}^{\infty} \left( \frac{1}{t} - \frac{1}{1 - e^{-t}} \right) dt
\[ \frac{1}{\mu} - t \z \mu = \int_{0}^{\infty} (\frac{1}{\mu} - t + \frac{1}{\mu} - \frac{t}{2}) \, dt. \]
\[
\frac{1}{z} \ln(z) = 1 + \frac{1}{t} - \frac{1}{t^2 + \frac{1}{t^2} + \cdots}
\]
\[
\frac{m:mi}{2}z - \frac{2 \int_{0}^{\infty} \frac{t}{t^{2} + z^{2}} \left( t^{2} + \pi \frac{2}{\pi} - 1 \right) dt}{m:mm}
\]
\[ \mathcal{R}(z) + \beta = \int_0^\infty \frac{1}{\sqrt{t} - z} \, dt \]
\begin{align*}
\frac{1}{t} &= \int_0^1 \frac{1 - (t - 1)z - 1}{1 - t} \, dt
\end{align*}
\[ \theta(z+1) = -\beta + \frac{1}{2\pi \sqrt{c^2+m^2}} \]
\[ \frac{\pi z}{\sin(\pi s) - (s - 1)} \]
where
\[ | \phi \pm \varphi | \leq \pi - \delta (\phi < \pi) \]
and
\[ 1 < c < 2 \].
\[ \int_0^\infty \frac{1}{1 + t} \, dt = \int_0^\infty \left( 1 - \frac{\ln t}{\sqrt{t}} \right) \, dt \]
\[ \frac{\mathrm{d}t}{t} = \int_0^1 (1 - \frac{1}{t} - t) \, \mathrm{d}t - \int_1^\infty \frac{1}{t} \, \mathrm{d}t \]
\[
\frac{1}{t} = \int_0^\infty \left( \frac{1}{t} - \frac{1}{t} \right) dt
\]
\begin{equation}
\int t \text{d}t
\end{equation}

---

**Mathematical Applications**

**Contents**

- Summation of Rational Functions
- Mellin-Barnes Integrals
- \(n\)-Dimensional Sphere

**Summation of Rational Functions**

As shown in [Temme(1996)](http://dlmf.nist.gov/Contents/bib/T#temme:1996:sfi) (3.4), the results given in [Series Expansions](http://dlmfseriesexpansions.xhtml) can be used to sum infinite series of rational functions.

**Example**

---

dlmfmathematicalapplications.xhtml

---

dlmfmathematicalapplications.xhtml ---
$S = \sum_{k=0}^{\infty} a_k$, \\
$a_k = \frac{k}{(3k+2)(2k+1)}$.
By decomposition into partial fractions

\[
\frac{2}{k+1} = \frac{2}{k} + \frac{2}{3} - \frac{1}{k}
\]
\[
\frac{1}{2} \leq \frac{1}{k+1} - \frac{1}{k+\frac{1}{2}} = \left(\frac{1}{k+1} - \frac{1}{k+\frac{1}{2}}\right) - 2\left(\frac{1}{k+\frac{1}{2}}\right).
\]
\[ k + \frac{1}{k} - \frac{1}{k + \frac{2}{3}} \]

Hence from (Series Expansions 6), (Special Values and Extrema Equation 13) and (Equation 19)
\[ \frac{\frac{2}{3} \cdot 3 \ln 3 - 2 \ln 2 - \frac{1}{3} \pi \sqrt{3}}{\pi} = \frac{3}{2} \]
Mellin-Barnes Integrals

Many special functions can be represented as a Mellin-Barnes integral, that is, an integral of a product of gamma functions, reciprocals of gamma functions, and a power of $z$, the integration contour being doubly-infinite and eventually parallel to the imaginary axis. The left-hand side of (Integral Equation 1) is a typical example. By translating the contour parallel to itself and summing the residues of the integrand, asymptotic expansions of $f(z)$ for large $|z|$, or small $|z|$, can be obtained complete with an integral representation of the error term.
The volume \( V \) and surface area \( A \) of the \( n \)-dimensional sphere of radius \( r \) are given by

\[
V = \frac{\pi^{\frac{n}{2}}}{\Gamma\left(\frac{n}{2}\right)} r^n
\]

\[
A = \frac{n \pi^{\frac{n}{2}}}{\Gamma\left(\frac{n}{2}\right)} r^n
\]
\[
S = \frac{2\pi}{2n} \left(\frac{1}{n} - 1 \right) \sin \left(\frac{1}{2n}\right)
\]
An effective way of computing \( \Gamma(z) \) in the right half-plane is backward recurrence, beginning with a value generated from the
Or we can use forward recurrence, with an initial value.

For the left half-plane we can continue the backward recurrence or make use of the reflection formula.

Similarly for

\[
\ln(\gamma(z)), \quad \psi(z), \quad \text{and the polygamma functions.}
\]

Multidimensional Integrals

Let
\[ V_n = \{ t_1 + t_2 + \cdots + t_n \leq 1 \} \]
be the simplex.
Then for $k \geq 0$,

$$k = 1, 2, \ldots, n+1,$$

$$\mathbb{E} \left( \sum_{n=1}^{N} V_{n} \right) \leq t \leq 1.$$
\[ z_1 \leq t \leq z_2 \leq \cdots \leq t_n \leq z_n \leq \cdots \leq t_1 \leq z_1 \]
\[ \frac{\theta(z_1)}{\theta(z_2)} \leq \theta(z_n) = 1 + z_1 + \text{...} \]
\[
\begin{align*}
\sum_{i=2}^{n} (x^2 + \ldots + x^n) \\
\frac{1}{V_n} \sum_{k=1}^{n} t_k
\end{align*}
\]
\[(z_{n+1} - 1) \leq \int_{t_k}^{t_k + \lambda_k} \sum_{k=1}^{n} (z_k - 1) dt_k\]
\[
\mu(z_1) = \mu(z_2) \mu(n + 1) + \mu(z_1 + z_2 + \cdots + z_{n+1})
\]
\[ \prod_{i=1}^{n} (t_i) = \prod_{i=1}^{n} \left( \frac{1}{j} \right) \]

**Selberg-type Integrals**

\[
\Delta (t_1, t_2, \ldots, t_n) = \prod_{i=1}^{n} \left( \frac{1}{j} \right)
\]
Then

\[
\begin{align*}
\int_{0}^{1} \left[ t_{j} - t_{k} \right] \, dt
\end{align*}
\]
\[ \frac{1}{2^m} \cdot \mu \left( \sum_{k=1}^{n} t_{k} - \left( t_{1} - \cdots - t_{n} \right) \right) \]

\[ \leq \mu \left( t_{n} \right) \]

\[ \frac{1}{2^{c}} \cdot \mu \left( \sum_{k=1}^{n} t_{k} \right) - \frac{1}{2^{c}} \cdot \mu \left( \sum_{k=1}^{n} t_{k} \right) \]
\[ b - 1 \int_{t_k}^1 \, \frac{1}{(\beta(1 + c))^n} \, dt_k \]
\[
\begin{align*}
\sum_{k=1}^{m} & \left( a + \sum_{n=1}^{k} (n-k) c \right) + \left( a + b + \sum_{n=2}^{m} (2n-k-1) c \right) \\
& \quad \left( a + \sum_{n=1}^{k} (n-k) c \right) + \left( a + b + \sum_{n=2}^{m} (2n-k-1) c \right)
\end{align*}
\]
\[
\underover{\frac{\sum_{i=k+n}^{a+(n-k)c}}{\sum_{i=k+n}^{b+(n-k)c}}}{\sum_{i=k+n}^{1+n}}{\sum_{i=k+n}^{1+n}}
\]
\[
\frac{m}{n} = \frac{1}{k+1} \cdot \frac{a+b+(2n-k-1)c}{k+1}
\]
provided that
\[
\sum_{a}^{b} > 0, \sum_{c} > -\min\left(\frac{1}{n}, \sum_{a}\frac{(n-1)}{a(n-1)}\right),
\]
\[
\sum_{a}^{b} > 0, \sum_{c} > -\min\left(\frac{1}{n}, \sum_{a}\frac{(n-1)}{a(n-1)}\right),
\]
\[ \frac{b}{(n-1)} \]

Secondly,

\[
\int_{0}^{\infty} n(0.1)^{t_1} (0.2)^{t_2} \, dt_1 \, dt_2
\]
\[\left| t_{m} \right| \leq 2c \sum_{k=1}^{n} (t_1, \ldots, t_n) \]
\[
\frac{d}{dt} t^k = \sum_{i=1}^{m} \left( a + (n-k) c \right)
\]
\[ k = 1^n \]
\[ \alpha(a + (n - k)c) \]
\[ \alpha(1 + kc) \]
\[ n \]
\[(1 + c^n)^n \leq \frac{1}{n!}
\]

when
\[
\left| a \right| > 0,
\]
\[
\left| c \right| > -\min \left( \frac{1}{n}, \frac{a}{\left| c \right|} \right).
\]
CHAPTER 1. OVERVIEW

Thirdly,

\[
\frac{1}{n^2} - \frac{1}{(n-1)^2} = \frac{1}{n^2} - \frac{1}{n^2} \pm \frac{1}{n^2} = \frac{1}{n^2} - \frac{1}{n^2} = \frac{1}{n^2} - \frac{1}{n^2}
\]
\[
\left| \frac{\Delta (t_1, \ldots, t_n)}{2c} \right| \sum_{k=1}^{n} \exp \left( \sum_{i=1}^{n} \frac{t_i}{k} \right)
\]
\[ \frac{1}{2} \int t_k^2 (1 + k c)^n \]
Dyson's Integral

\[
\frac{1}{(\pi n)^2} \int [x^{1/(1+c)}] \, dx
\]
\[\mu, \mu, n \leq j < k \leq n \]

\[|\mathcal{L}_{\mathcal{G}}(\theta_j) - \mathcal{L}_{\mathcal{G}}(\theta_k)|^2\]
\( \frac{\int_{\theta_1}^{\theta_n} \# \text{b} \, dn}{\# \text{b}^{\text{normal}} \cdot (1 + \# \text{b}^{\text{normal}}) \# \text{b}^{\text{normal}} (1 + \# \text{b})} = \frac{\# \text{r}}{\# \text{r}(1 + \# \text{b}^{\text{normal}}) \# \text{r}(1 + \# \text{b})} \)
\[ \frac{1}{n!} \cdot b > \frac{1}{n} \]

---

dlmfnotation.xhtml

---

- The Gamma Function -- Notation

\[ j, k \]
\[ m, n \] nonnegative integers.

except in <a href="dlmfphysicalapplications.xhtml">Physical Applications</a>

real variables.

complex variable.
\[
\begin{align*}
(b, q, s, w) & \text{ real or complex variables with} \\
& \quad |q| < 1. \\
\alpha & \text{ arbitrary small positive constant.} \\
\mathbb{C} & \text{ complex plane (excluding infinity).} \\
\mathbb{R} & \text{ real line (excluding infinity).}
\end{align*}
\]
\[ \binom{n}{m} = \frac{n!}{m!(n-m)!} \]

- **Empty Sums**: zero.
- **Empty Products**: unity.
The main functions treated in this chapter are the gamma function $\Gamma(z)$, the psi function $\psi(z)$, the beta function $B(a,b)$, and the $q$-gamma function $\Gamma_q(z)$.
The notation \( \zeta(z) \) is due to Legendre. Alternative notations for this function are: \( \pi(z-1) \) (Gauss) and \( (z-1)! \). Alternative notations for the psi function are:
<table>
<thead>
<tr>
<th>$\Phi(z)$</th>
<th>Gauss; Jahnke and Emde(1945)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Phi(z)$</td>
<td>Whittaker and Watson(1927)</td>
</tr>
</tbody>
</table>

\[ \Phi(z) (z - 1) \]

\[ \Phi(z) \]

\[ \Phi(z) \]
<table>
<thead>
<tr>
<th>PAGES</th>
<th>507</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><em>F</em>(z - 1)</th>
<th>Davis (1933)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pairman (1919)</td>
</tr>
</tbody>
</table>

---

**dlmfphysicalapplications.xhtml**

---

**dlmfphysicalapplications.xhtml** —

---

---
CHAPTER 1. OVERVIEW

<h3>Physical Applications</h3>

Suppose the potential energy of a gas of point charges with positions and free to move on the infinite line, is given by

\[
\mathcal{W} = \frac{1}{2} \sum_{i=1}^{n} \frac{1}{x_i - x_{i+1}}.
\]
\[
\sum_{\lambda=1}^{n} x_{\lambda}^{2} - \sum_{1 \leq j \leq n} \ln |x_{\lambda} - x_{j}| \]

The probability density of the positions when the gas is in thermodynamic...
equilibrium is:

\[
P(x_1, \ldots, x_n) = C \exp(-\frac{W(kT)})
\]
\[
\begin{align*}
\text{where } & \quad k \text{ is the Boltzmann constant, } \\
&T \text{ the temperature and } \\
C \text{ a constant.}
\end{align*}
\]
Then the partition function (with \( \beta = \frac{1}{kT} \)) is given by

\[
\begin{align*}
\chi(n)(\beta) &= \frac{1}{n!} \left( \sum_{m=0}^{\infty} m^m \right)
\end{align*}
\]
\[
\frac{n^2}{\beta W} \frac{\mathcal{E}}{x} = \left(\frac{2\pi}{n^2\beta^2} - \frac{2\pi}{n^2\beta^2} - \frac{2\pi}{n^2\beta^2}\right)
\]
\[
\begin{align*}
&\beta_n(n - 1) \cdot 4 \\
&\times (\beta(1 + \frac{1}{2}\beta)) - n
\end{align*}
\]
\[
\sum_{j=1}^{n} \gamma (1 + \frac{1}{2} j \beta)
\]

For \( n \) charges free to move on a circular wire of radius 1,
and the partition function is given by
\[ m \subseteq (\beta) = \frac{1}{(2\pi)^n} \int_{[\, -\pi, \pi \, ]^n} \left( W_{\beta, W} - W \right) \]
\[ \int_{\phi_1} \int_{\phi_n} = \frac{\gamma (1 + \frac{1}{2}n\beta)}{\gamma (1 + \frac{1}{2}n)} \]
The functions
\[
\psi(n) - \psi(n + 1) = \frac{1}{n}.
\]

\[
\psi(n) = \frac{(-1)^{n+1} B_n}{n}\psi(n)
\]

---

dlmfpolygammafunctions.xhtml

---

dlmfpolygammafunctions.xhtml ---

The Gamma Function -- Polygamma Functions

The functions
\[
\psi(n) - \psi(n + 1) = \frac{1}{n}.
\]
\[
\psi(n) = \frac{(-1)^{n+1} B_n}{n}\psi(n)
\]
are called the \textit{polygamma functions}. In particular, \(\psi(\z)\) is the \textit{trigamma function}; \(\psi''(\z)\), \(\psi'''(3)\), etc.,
are the tetra-, penta-, and hexagamma functions respectively. Most properties of these functions follow straightforwardly by differentiation of properties of the psi function. This includes asymptotic expansions.

In the second and third equations,

\[
\alpha(n + 1) = \frac{\psi'(z)}{z^{n+1}}
\]
\[
\frac{1}{\sum_{k=0}^{\infty} (k + z)^2} \quad \text{if} \quad z \neq 0, -1, -2, \ldots
\]
$$\phi(n) = (1 - \frac{1}{m})n^m$$

\[
\phi(n) = \prod_{p|n} (1 - \frac{1}{p}) \prod_{d|n, d \neq n} \phi(d)
\]
\[
\frac{1}{2} \cdot \left( n + 1 \right) = (2n + 1) n!
\]
\[
\left(2^n + 1\right) - 1 \right)
\phi(n+1)
\]

\[
\phi'(n+\frac{1}{2})
\]

\[
\frac{1}{m:n}
\]

\[
\frac{1}{m:mm}
\]

\[
\frac{1}{m:mm}
\]

\[
\frac{1}{m:mm}
\]

\[
\frac{1}{m:mm}
\]

\[
\frac{1}{m:mm}
\]

\[
\frac{1}{m:mm}
\]

\[
\frac{1}{m:mm}
\]

\[
\frac{1}{m:mm}
\]
\[
\frac{1}{2} \pi^2 - 4 \sum_{k=1}^{n} \frac{1}{(2k-1)^2}
\]
As \( z \to \infty \) in 
\[
|\text{ph}z| \leq \pi - \delta (\pi < \delta)
\]
\[ \underbrace{\frac{1}{z} + \frac{1}{2z^2}}_{\text{mfracs}} + \sum_{k=1}^{\infty} \frac{B_{2k}}{z^{2k+1}} \]
CHAPTER 1. OVERVIEW

dlmfqgammaandbetafunctions.xhtml
\[
\left(\begin{array}{c} a \\ q \end{array}\right)^n = \underbrace{\left(1 - a^q^k\right)}_{k=0}^{n-1} \]
\[
\frac{n!}{q(q + 1)(q + 2)\cdots(q + n - 1)}
\]
When $|q| < 1$,
\[
\frac{1}{(a;q)_\infty} = \sum_{k=0}^{\infty} (1 - a^k q^k)
\]

When \(0 < q < 1\), \(\Gamma_q\) is the \(q\)-Gamma Function.
\[ q < 1, \]

\[
\left( \frac{(q ; q)}{\infty} \right)^{1 - q} / (1 - q) = 1 - z
\]
\[ \int_{q_z}^{\infty} \mathcal{R}_q(1) = \mathcal{R}_q(2) = 1 \]
\[ n! = \lambda q (n + 1) (z + 1) \]
\[
\frac{1 - q^z}{1 - q} \nu(q(z))
\]

Also, \( \ln(\nu(q(x))) \) is convex for
and the analog of the Bohr-Mollerup theorem holds.

If
\[
0 < q < r < 1
\]

then
\[
\frac{\Gamma(q)}{\Gamma(r)} < \frac{\Gamma(q)(x)}{\Gamma(r)(x)}
\]
when \(0 < x < 1\) or when
\[
\eta(x) > \eta(r)(x),
\]
and

when
For generalized asymptotic expansions of
\[ \frac{\ln q}{|z|} \rightarrow \infty \] 

see Olde Daalhuis (1994) and Moak (1984).
\[
\frac{\theta(q(a))}{\theta(q(b))} = \frac{\theta(q(a+b))}{\theta(q(b))}
\]
\[
B_{q}(a, b) = \int_{0}^{1} \frac{t^{a-1}}{(t^{q}; q)} dt
\]
\[(t^q)^b; q < 1\]
Throughout this subsection $\alpha(k)$ is

\[
\frac{1}{k!}\frac{\Gamma(k+\frac{b}{a})}{\Gamma(k+1)}\frac{1}{\Gamma(k+\frac{b}{a})}.
\]
\[ \sum_{k=1}^{\infty} c_k z^k = \prod_{k=1}^{\infty} (1 - \alpha_k z^k) \]

where

\[ c_1 = 1, \quad c_2 = \alpha_3, \quad \ldots \]
\(c_k = \beta_{k-1} - \gamma_{(2)}c_{k-2}\)
\[ c = \alpha (3 - 3k) - \cdots + \alpha (k - 1) - 1^k \]
\[
\mathcal{C}_k \geq 3
\]

For 15D numerical values of \( c_k \) see Abramowitz and Stegun (1964) (p. 256), and for 31D values see Wrench (1968).

\[
\ln(1 + z) = \frac{z}{\pi} \sum_{k=1}^\infty \frac{(-1)^{k-1} z^k}{k2^{2k-1}k!}
\]
\[
\ln(1 + z) + z(1 - \beta) + \sum_{k=2}^{\infty}(-1)^k k
\]
\[ \frac{z^k}{k} \quad \text{if} \quad |z| < 2. \]
\[ m_{\text{mo}} ) ( m_{\text{mo}} \) \\
\[ m_{\text{mo}} = m_{\text{mi}} \] \\
\[ m_{\text{mo}} = - m_{\text{mi}} \] \\
\[ m_{\text{mo}} = m_{\text{mn}} = 2 \] \\
\[ m_{\text{row}} \] \\
\[ m_{\text{mi}} k \] \\
\[ m_{\text{mn}} = 2 \] \\
\[ m_{\text{row}} \] \\
\[ m_{\text{mi}} ( k ) \] \\
\[ m_{\text{mi}} = k \] \\
\[ m_{\text{row}} \] \\
\[ m_{\text{mi}} = k \] \\
\[ m_{\text{mi}} z \] \\
\[ m_{\text{row}} \] \\
\[ m_{\text{mi}} k \] \\
\[ m_{\text{mn}} = 1 \] \\
\[ m_{\text{row}} \] \\
\[ m_{\text{row}} \] \\
\[ m_{\text{row}} \] \\
\[ m_{\text{row}} \] \\
\[ m_{\text{math}} \]
\(|z| < 1\),

\(\phi = (1 + z) = \frac{1}{2z} - \frac{\pi}{2 \cot(\pi z)}\)
\[ z^2 - 1 + 1 - \beta - \sum_{k=1}^{\infty} (\beta(2k+1)) - 1 \]
For 20D numerical values of the coefficients of the Maclaurin series for

\[
\mathcal{C}(z) = -\chi \quad \text{when} \quad z \neq 0, -1, -2, \ldots
\]
\[ \frac{1}{z} + \sum_{k=1}^{\infty} \frac{z}{k(z+k)} = -\gamma + \sum_{k=0}^{\infty} \left( \frac{1}{k} - \frac{1}{k+z} \right) \]
\[
\frac{1}{k+1} - \frac{1}{k+z}
\]

and

\[
\phi(z+1) - \phi(z+2)
\]
\[
\sum_{k=0}^{\infty} \frac{(-1)^k}{k+z} = 2^{z-2}
\]

Also,
\[ \phi(y + 1) = \sum_{k=1}^{\infty} \frac{y}{k^2 + y^2} \]
CHAPTER 1. OVERVIEW

dlmfsums.xhtml

— dlmfsums.xhtml —
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
<div align="center">
<a href="http://dlmf.nist.gov">
Digital Library of Mathematical Functions
</a><br/>
The Gamma Function -- Sums
</div>
<hr/>
<h3>Sums</h3>
<div align="center">
<m:math display="block">
<m:mrow>
<m:mrow>
<m:mrow>
<m:munderover>
<m:mo movablelimits="false">\sum</m:mo>
<m:mrow>
<m:mi>k</m:mi>
</m:mrow>
</m:munderover>
<m:msup>
<m:mrow>
<m:mo>(</m:mo>
<m:mrow>
<m:mo>-1</m:mo>
</m:mrow>
</m:msup>
<m:mi mathvariant="normal">\zeta</m:mi>
</m:mrow>
<m:mrow>
<m:mrow>
<m:mo>(k)</m:mo>
</m:mrow>
</m:mrow>
</m:mrow>
</m:math>
</div>
</div>
\[
\pi = -\frac{8}{\pi^2}
\]

\[
\sum_{k=1}^{\infty} \frac{1}{k} \left( k + 1 \right) = \delta
\]
For further sums involving the psi function see
Hansen(1975)(pp.360367). For sums of gamma functions see
Andrews <em>et.al.</em>(1999)(Chapters 2 and 3).

For related sums involving finite field analogs of the gamma and
beta functions (Gauss and Jacobi sums) see
Andrews <em>et.al.</em>(1999)(Chapter 1) and
Terras(1999)
dlmfsoftware.xhtml

— dlmfsoftware.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
<div align="center">
<a href="http://dlmf.nist.gov">
    Digital Library of Mathematical Functions
</a><br/>
    The Gamma Function -- Software
</div>
<hr/>
\getchunk{page foot}

———

dlmfspecialvaluesandextrema.xhtml

— dlmfspecialvaluesandextrema.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
<div align="center">
<a href="http://dlmf.nist.gov">
    Digital Library of Mathematical Functions
</a><br/>
    The Gamma Function -- Special Values and Extrema
</div>
<hr/>
<h3>Special Values and Extrema</h3>
<h6>Contents</h6>
<ul>
    <li>Gamma Function</li>
    <li>Psi Function</li>
    <li>Extrema</li>
</ul>
<h4>Gamma Function</h4>
<div align="center">
    \m:math display="block">
        \m:mrow
            \m:mrow
                \m:mrow
                    \m:mi mathvariant="normal">\#x0393;</m:mi>
            \m:mrow>
        \m:mrow>
    \m:math display="block">
        \m:mrow
            \m:mrow
                \m:mrow
                    \m:mi mathvariant="normal">\#x0393;</m:mi>
            \m:mrow>
\[(\frac{1}{n+1}) = \frac{n}{n!}\]

\[\#\ \mathcal{R}(\partial y)\]
\[
\rho \left( \frac{1}{2} + \frac{1}{2} \right) = \frac{1}{2} \sinh \left( \pi y \right)
\]
\[
\frac{1}{2} - \frac{\|y\|}{\|\beta\|} = \|\beta\| \frac{1}{2} + \frac{\|y\|}{\|\beta\|}
\]
\[ \mbox{m} \frac{\mbox{m} \tan^2 \left( \frac{\pi}{4} \right)}{\cosh \left( \frac{\pi}{4} y \right)} \]
\[ 3 < \sinh(\pi y) \leq \cosh(\pi y) \]
\[ \alpha(1/2) = \frac{\pi}{2} = 1.77245385090551602729\]

\[ \alpha(1/3) = \frac{\pi}{3} = 1.04719755119659774616\]
</div align="center">
</m:math display="block">
<m:mrow>
</m:mrow>
</div>
</m:row>
</m:mrow>
</div align="center">
</m:math display="block">
</m:mrow>
</div>
</m:row>
</m:mrow>
</div align="center">
</m:math display="block">
</m:mrow>
</div>
\[ \gamma = 3.62560990822190831193 \ldots \]

\[ \gamma (\frac{3}{4}) = 1.22541670246517764512 \ldots \]

\[ \gamma'(1) = \frac{1}{2} 1.2453643626994 \ldots \]
CHAPTER 1. OVERVIEW

Psi Function

\[ \psi(1) = -\alpha \]

\[ \psi\left(\frac{1}{2}\right) = -\alpha \]
$$\frac{1}{n} - \ln 2 = \sum_{k=1}^{n} \left( \frac{1}{k} - \varsigma \right)$$
\[
\phi(n+\frac{1}{2}) = -\gamma - 2\ln 2 + 2(1+\frac{1}{3}+\ldots)
\]
\[
\frac{1}{2n - 1} = \phi(y)
\]
\[
\frac{2\pi}{2} + \coth(\pi y)
\]
\[
\frac{\pi}{2} \tanh(\pi y) = -\frac{1}{\pi y + 1}\]

\[
\frac{\pi}{2} \tanh(\pi y) = -\frac{1}{\pi y + 1}\]
\[
\sqrt{2}\sinh y + \frac{\pi}{2}\coth(\pi y) = 0
\]

where \(0 < p < q\) are integers, then
\[
\begin{align*}
\left(-\alpha - \ln q - \frac{\pi}{2} \cot \left( \frac{\pi}{p} \frac{k}{q} \right) \right) + \frac{1}{2} \sum_{k=1}^{q-1} \left( k - \frac{1}{q} ight)
\end{align*}
\]
\[ \cos \left( \frac{2\pi k p}{q} \right) \ln \left( 2 - 2 \cos \left( \frac{2\pi k}{q} \right) \right) \]
Extrema

\[ \alpha' = \varphi_n = 0. \]
<table>
<thead>
<tr>
<th>( \mathbb{E} )</th>
<th>( \sigma(x^n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.46163 21449</td>
</tr>
<tr>
<td>1</td>
<td>-0.50408 30083</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \mathbb{E} )</th>
<th>( \mu(x^n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.88560 31944</td>
</tr>
<tr>
<td>1</td>
<td>3.54464 36112</td>
</tr>
<tr>
<td>2</td>
<td>-1.57349 84732</td>
</tr>
<tr>
<td>----</td>
<td>---------------</td>
</tr>
<tr>
<td>3</td>
<td>-2.61072 08875</td>
</tr>
<tr>
<td>4</td>
<td>-3.63529 33665</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>5</td>
<td>-4.65323 77626</td>
</tr>
<tr>
<td>6</td>
<td>-5.66716 24513</td>
</tr>
<tr>
<td>7</td>
<td>-6.67841 82649</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>8</td>
<td>$-7.68778,83250$</td>
</tr>
<tr>
<td>9</td>
<td>$-8.69576,41633$</td>
</tr>
<tr>
<td>10</td>
<td>$-9.70267,25406$</td>
</tr>
</tbody>
</table>
\[
\lim_{n \to \infty} x_n = -n + \frac{1}{\pi} \arctan \left( \frac{\pi}{\ln n} \right) + O(1)
\]
These tables show Axiom's compliance with published standard values. In all cases shown here Axiom conforms to the accuracy of the published tables.

<ul>
  <li>The Gamma Function</li>
</ul>
<li>The Psi Function</li>
</ul>

<h4>The Gamma Function</h4>

This table was constructed from the published values in the Handbook of Mathematical Functions, by Milton Abramowitz and Irene A. Stegun, by Dover (1965), pp 267-270.

The first column is the point where the Gamma function is evaluated.
The second column is the value reported in the Handbook.
The third column is the actual value computed by Axiom at the given point.
The fourth column is the difference of Axiom’s value and the Handbook value.

<table border="1">
  <tr>
    <th>point</th>
    <th>Handbook Value</th>
    <th>Axiom Computed Value</th>
    <th>Difference</th>
  </tr>
  <tr align="right">td>1.000</td>
    <td>1.0000000000</td>
    <td>1.0000000000000000</td>
    <td align="right">0.000000000</td>
  </tr>
  <tr align="right">td>1.005</td>
    <td>0.9971385354</td>
    <td>0.9971385352483757</td>
    <td align="right">-1.51E-10</td>
  </tr>
  <tr align="right">td>1.010</td>
    <td>0.9943258512</td>
    <td>0.99432585118631189</td>
    <td align="right">-2.03E-11</td>
  </tr>
  <tr align="right">td>1.015</td>
    <td>0.9915612888</td>
    <td>0.9915612888413123</td>
    <td align="right">4.14E-11</td>
  </tr>
  <tr align="right">td>1.020</td>
    <td>0.9888420303</td>
    <td>0.9888420325387897</td>
    <td align="right">4.31E-11</td>
  </tr>
  <tr align="right">td>1.025</td>
    <td>0.9861739633</td>
<table>
<thead>
<tr>
<th></th>
<th>0.98617396313592742</th>
<th>-1.54E-10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.98354995053928918</td>
<td>7.59E-11</td>
</tr>
<tr>
<td></td>
<td>0.98097156056367696</td>
<td>-4.60E-11</td>
</tr>
<tr>
<td></td>
<td>0.9784382009247683</td>
<td>3.00E-11</td>
</tr>
<tr>
<td></td>
<td>0.97594929183099266</td>
<td>-6.55E-11</td>
</tr>
<tr>
<td></td>
<td>0.97350426556841785</td>
<td>-2.72E-11</td>
</tr>
<tr>
<td></td>
<td>0.97110256624499502</td>
<td>-6.77E-11</td>
</tr>
<tr>
<td></td>
<td>0.96874364951272707</td>
<td>-2.36E-12</td>
</tr>
<tr>
<td></td>
<td>0.9664269822977113</td>
<td>-1.37E-11</td>
</tr>
<tr>
<td></td>
<td>0.964152042536858</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>0.94547431492209555</td>
<td>1.12E-11</td>
<td></td>
</tr>
<tr>
<td>0.9435901856</td>
<td>0.94359018561564112</td>
<td></td>
</tr>
<tr>
<td>1.06E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.120</td>
<td>0.9417426998470138</td>
<td></td>
</tr>
<tr>
<td>0.9417426998470138</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.39E-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.125</td>
<td>0.9399314497298807</td>
<td></td>
</tr>
<tr>
<td>0.9399314497298807</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.67E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.130</td>
<td>0.93815603556085947</td>
<td></td>
</tr>
<tr>
<td>0.93815603556085947</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.14E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.135</td>
<td>0.9364160657</td>
<td></td>
</tr>
<tr>
<td>0.9364160656689694</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.97E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.140</td>
<td>0.9347111562</td>
<td></td>
</tr>
<tr>
<td>0.93471115622975964</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.05E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.145</td>
<td>0.9330409311</td>
<td></td>
</tr>
<tr>
<td>0.93304093109978414</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.51E-12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.150</td>
<td>0.9314050217</td>
<td></td>
</tr>
<tr>
<td>0.93140502165323868</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.93E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.155</td>
<td>0.9298030666</td>
<td></td>
</tr>
<tr>
<td>0.9298030666</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.165</td>
<td>0.9282347120</td>
<td>0.92823471196190366</td>
</tr>
<tr>
<td>1.170</td>
<td>0.9266996106</td>
<td>0.92669961062266581</td>
</tr>
<tr>
<td>1.175</td>
<td>0.9251974225</td>
<td>0.92519742251686099</td>
</tr>
<tr>
<td>1.180</td>
<td>0.9237278143</td>
<td>0.92372781430006712</td>
</tr>
<tr>
<td>1.185</td>
<td>0.9222904591</td>
<td>0.92229045925047382</td>
</tr>
<tr>
<td>1.190</td>
<td>0.9208503713</td>
<td>0.9208503713299241</td>
</tr>
<tr>
<td>1.195</td>
<td>0.9195112341</td>
<td>0.91951123406686597</td>
</tr>
<tr>
<td>1.200</td>
<td>0.9181687424</td>
<td>0.91816874239667101</td>
</tr>
<tr>
<td>1.205</td>
<td>0.9168572606</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>0.91685726056661909</td>
<td>0.9155764930</td>
<td>0.91557649299806532</td>
</tr>
<tr>
<td>1.210</td>
<td>0.9143261400</td>
<td>0.91432614997006778</td>
</tr>
<tr>
<td>1.215</td>
<td>0.9131059475</td>
<td>0.91310594750311536</td>
</tr>
<tr>
<td>1.220</td>
<td>0.9119156071</td>
<td>0.91191560725927312</td>
</tr>
<tr>
<td>1.225</td>
<td>0.9107548564</td>
<td>0.910754856376556895</td>
</tr>
<tr>
<td>1.230</td>
<td>0.9096234274</td>
<td>0.90962342744426173</td>
</tr>
<tr>
<td>1.235</td>
<td>0.9085210583</td>
<td>0.90852105834198582</td>
</tr>
<tr>
<td>1.240</td>
<td>0.9074474922</td>
<td>0.90744749215126341</td>
</tr>
<tr>
<td>1.245</td>
<td>0.9064024771</td>
<td>0.9064024771</td>
</tr>
<tr>
<td>Value</td>
<td>Count 1</td>
<td>Count 2</td>
</tr>
<tr>
<td>-------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>1.255</td>
<td>0.9053857663</td>
<td>0.9053857663</td>
</tr>
<tr>
<td>1.260</td>
<td>0.9043971178</td>
<td>0.90439711780910215</td>
</tr>
<tr>
<td>1.265</td>
<td>0.9034362946</td>
<td>0.90343629466913566</td>
</tr>
<tr>
<td>1.270</td>
<td>0.9025030645</td>
<td>0.9025030646208062</td>
</tr>
<tr>
<td>1.275</td>
<td>0.9015971994</td>
<td>0.90159719946523187</td>
</tr>
<tr>
<td>1.280</td>
<td>0.9007184765</td>
<td>0.90071847650745973</td>
</tr>
<tr>
<td>1.285</td>
<td>0.8998666769</td>
<td>0.89986667689491762</td>
</tr>
<tr>
<td>1.290</td>
<td>0.8990415863</td>
<td>0.89904158628967101</td>
</tr>
<tr>
<td>1.295</td>
<td>0.8982429947</td>
<td>0.8982429947</td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>1.30</td>
<td>0.894299468914737</td>
<td>-1.72E-11</td>
</tr>
<tr>
<td>1.30</td>
<td>0.897069630804477</td>
<td>2.65E-12</td>
</tr>
<tr>
<td>1.30</td>
<td>0.8967448951215833</td>
<td>2.37E-11</td>
</tr>
<tr>
<td>1.30</td>
<td>0.89672448951215833</td>
<td>4.53E-11</td>
</tr>
<tr>
<td>1.30</td>
<td>0.89600417674396082</td>
<td>4.53E-11</td>
</tr>
<tr>
<td>1.30</td>
<td>0.8953095644995535</td>
<td>5.43E-11</td>
</tr>
<tr>
<td>1.30</td>
<td>0.89600417674396082</td>
<td>6.12E-11</td>
</tr>
<tr>
<td>1.30</td>
<td>0.8953095644995535</td>
<td>1.28E-11</td>
</tr>
<tr>
<td>1.30</td>
<td>0.893996686686686083</td>
<td>7.95E-11</td>
</tr>
<tr>
<td>1.30</td>
<td>0.893996686686686083</td>
<td>7.95E-11</td>
</tr>
<tr>
<td>1.30</td>
<td>0.8937805346103716</td>
<td>-3.97E-11</td>
</tr>
<tr>
<td>1.30</td>
<td>0.8937805346103716</td>
<td>1.51E-10</td>
</tr>
<tr>
<td>1.30</td>
<td>0.8927438516233538</td>
<td>1.51E-10</td>
</tr>
<tr>
<td>1.30</td>
<td>0.8927438516233538</td>
<td>1.51E-10</td>
</tr>
<tr>
<td>1.30</td>
<td>0.8922155072</td>
<td>1.51E-10</td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>1.345</td>
<td>0.8916712485</td>
<td>0.89167124863270442</td>
</tr>
<tr>
<td>1.350</td>
<td>0.8911514420</td>
<td>0.89115144202666452</td>
</tr>
<tr>
<td>1.355</td>
<td>0.8906559235</td>
<td>0.8906559234803057</td>
</tr>
<tr>
<td>1.360</td>
<td>0.8901845324</td>
<td>0.8901845323574008</td>
</tr>
<tr>
<td>1.365</td>
<td>0.8897371116</td>
<td>0.88973711163470881</td>
</tr>
<tr>
<td>1.370</td>
<td>0.8893135074</td>
<td>0.88931350742948501</td>
</tr>
<tr>
<td>1.375</td>
<td>0.8889135692</td>
<td>0.88891356915622532</td>
</tr>
<tr>
<td>1.380</td>
<td>0.8885371494</td>
<td>0.88853714943101736</td>
</tr>
<tr>
<td>1.385</td>
<td>0.8881841041</td>
<td>0.88818410415622532</td>
</tr>
<tr>
<td>Value 1</td>
<td>Value 2</td>
<td>Value 3</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>0.88818410401940351</td>
<td>-9.53E-11</td>
<td></td>
</tr>
<tr>
<td>1.390</td>
<td>0.8878542918</td>
<td>0.88785429178544073</td>
</tr>
<tr>
<td>1.395</td>
<td>0.8875476748</td>
<td>0.88754764193323</td>
</tr>
<tr>
<td>1.400</td>
<td>0.8872638175</td>
<td>0.88726381750180738</td>
</tr>
<tr>
<td>1.405</td>
<td>0.8870028884</td>
<td>0.88700288823059736</td>
</tr>
<tr>
<td>1.410</td>
<td>0.8867646576</td>
<td>0.88676465760002188</td>
</tr>
<tr>
<td>1.415</td>
<td>0.8865489993</td>
<td>0.88654899924499497</td>
</tr>
<tr>
<td>1.420</td>
<td>0.8863557896</td>
<td>0.88635578960951567</td>
</tr>
<tr>
<td>1.425</td>
<td>0.8861849081</td>
<td>0.886184907918420432</td>
</tr>
<tr>
<td>1.430</td>
<td>0.8860362361</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>1.425</td>
<td>0.88603623612466142</td>
<td>2.35E-11</td>
</tr>
<tr>
<td>1.430</td>
<td>0.885906587</td>
<td>1.59E-10</td>
</tr>
<tr>
<td>1.435</td>
<td>0.885906587072826</td>
<td>-5.45E-11</td>
</tr>
<tr>
<td>1.440</td>
<td>0.88580506344804788</td>
<td>5.12E-11</td>
</tr>
<tr>
<td>1.445</td>
<td>0.88580506344804788</td>
<td>5.12E-11</td>
</tr>
<tr>
<td>1.450</td>
<td>0.88566138027095553</td>
<td>-3.63E-11</td>
</tr>
<tr>
<td>1.455</td>
<td>0.88566138027095553</td>
<td>9.99E-9</td>
</tr>
<tr>
<td>1.460</td>
<td>0.88560433642511449</td>
<td>3.29E-11</td>
</tr>
<tr>
<td>1.465</td>
<td>0.88560433642511449</td>
<td>3.29E-11</td>
</tr>
<tr>
<td>1.470</td>
<td>0.88560804952797856</td>
<td>4.00E-11</td>
</tr>
<tr>
<td>1.475</td>
<td>0.88560804952797856</td>
<td>4.00E-11</td>
</tr>
<tr>
<td>1.480</td>
<td>0.8856312168767672</td>
<td>-2.25E-11</td>
</tr>
<tr>
<td>1.485</td>
<td>0.8856312168767672</td>
<td>-2.25E-11</td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>1.480</td>
<td>0.88567945767984679</td>
<td>1.68E-10</td>
</tr>
<tr>
<td>1.485</td>
<td>0.8857469646</td>
<td>3.58E-11</td>
</tr>
<tr>
<td>1.490</td>
<td>0.8858355520</td>
<td>1.39E-11</td>
</tr>
<tr>
<td>1.495</td>
<td>0.8859451316</td>
<td>2.22E-11</td>
</tr>
<tr>
<td>1.500</td>
<td>0.8860756174</td>
<td>9.20E-11</td>
</tr>
<tr>
<td>1.505</td>
<td>0.8862269255</td>
<td>5.14E-11</td>
</tr>
<tr>
<td>1.510</td>
<td>0.8863989744</td>
<td>5.62E-11</td>
</tr>
<tr>
<td>1.515</td>
<td>0.8865916850</td>
<td>8.75E-11</td>
</tr>
<tr>
<td>1.520</td>
<td>0.8868049797</td>
<td>1.15E-10</td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>1.525</td>
<td>0.8870387832457031</td>
<td>3.78E-11</td>
</tr>
<tr>
<td>1.530</td>
<td>0.8872930231</td>
<td>3.89E-11</td>
</tr>
<tr>
<td>1.535</td>
<td>0.8875676278</td>
<td>5.05E-11</td>
</tr>
<tr>
<td>1.540</td>
<td>0.8878625287</td>
<td>-2.97E-11</td>
</tr>
<tr>
<td>1.545</td>
<td>0.8881776586</td>
<td>-1.03E-11</td>
</tr>
<tr>
<td>1.550</td>
<td>0.8885129527</td>
<td>-4.41E-11</td>
</tr>
<tr>
<td>1.555</td>
<td>0.8888683478</td>
<td>2.74E-12</td>
</tr>
<tr>
<td>1.560</td>
<td>0.8892437830</td>
<td>-1.06E-11</td>
</tr>
<tr>
<td>1.565</td>
<td>0.8896391990</td>
<td>3.65E-12</td>
</tr>
<tr>
<td></td>
<td>0.89005465387</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>1.570</td>
<td>0.89005453859597561</td>
<td>-1.04E-10</td>
</tr>
<tr>
<td>1.575</td>
<td>0.8904476886</td>
<td>0.89094476859629979</td>
</tr>
<tr>
<td>1.580</td>
<td>0.8914196537</td>
<td>0.8919140515765388</td>
</tr>
<tr>
<td>1.585</td>
<td>0.892482141</td>
<td>0.8929619949</td>
</tr>
<tr>
<td>1.590</td>
<td>0.8935153493</td>
<td>0.8940823423580575</td>
</tr>
<tr>
<td>1.595</td>
<td>0.8946806085</td>
<td></td>
</tr>
<tr>
<td>1.615</td>
<td>0.895292432590298</td>
<td>0.8952924327</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>1.620</td>
<td>0.8959236865</td>
<td>0.895923686518247</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>1.625</td>
<td>0.8965742800</td>
<td>0.8965742800565979</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>1.630</td>
<td>0.8972442326</td>
<td>0.89724423258250552</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>1.635</td>
<td>0.8979334930</td>
<td>0.89793349308892934</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>1.640</td>
<td>0.8986420302</td>
<td>0.89864203016845012</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>1.645</td>
<td>0.8993598138</td>
<td>0.89935981399746452</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>1.650</td>
<td>0.9001158163</td>
<td>0.9001168163207648</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>1.655</td>
<td>0.9008830104</td>
<td></td>
</tr>
<tr>
<td>1.660</td>
<td>0.9016683712</td>
<td>0.90166837115595</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>1.665</td>
<td>0.9024728748</td>
<td>0.90247287490643413</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>1.670</td>
<td>0.9032964995</td>
<td>0.9032964995021503</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>1.675</td>
<td>0.9041392243</td>
<td>0.90413922432675797</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>1.680</td>
<td>0.9050010302</td>
<td>0.90500103023115419</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>1.685</td>
<td>0.9058818996</td>
<td>0.90588189953639731</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>1.690</td>
<td>0.9067818160</td>
<td>0.90678181602099839</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>1.695</td>
<td>0.9077007650</td>
<td>0.90770076490852225</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>1.700</td>
<td>0.9086387329</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>1.705</td>
<td>0.90863873285549646</td>
<td>-5.97E-11</td>
</tr>
<tr>
<td>1.710</td>
<td>0.9105716796</td>
<td>5.89E-11</td>
</tr>
<tr>
<td>1.715</td>
<td>0.9115666390</td>
<td>-1.31E-10</td>
</tr>
<tr>
<td>1.720</td>
<td>0.9125805779</td>
<td>-1.93E-11</td>
</tr>
<tr>
<td>1.725</td>
<td>0.9136134904</td>
<td>-1.16E-10</td>
</tr>
<tr>
<td>1.730</td>
<td>0.9146653712</td>
<td>-2.63E-11</td>
</tr>
<tr>
<td>1.735</td>
<td>0.9157362171</td>
<td>-2.15E-10</td>
</tr>
<tr>
<td>1.740</td>
<td>0.9168260252</td>
<td>-5.47E-11</td>
</tr>
<tr>
<td>1.745</td>
<td>0.9179347950</td>
<td>0.9179347950</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>y'</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>1.75</td>
<td>0.91793</td>
<td>8.97E-12</td>
</tr>
<tr>
<td>1.755</td>
<td>0.92020</td>
<td>3.95E-11</td>
</tr>
<tr>
<td>1.76</td>
<td>0.92137</td>
<td>4.68E-12</td>
</tr>
<tr>
<td>1.765</td>
<td>0.92256</td>
<td>4.88E-11</td>
</tr>
<tr>
<td>1.77</td>
<td>0.92376</td>
<td>2.96E-11</td>
</tr>
<tr>
<td>1.775</td>
<td>0.92498</td>
<td>2.89E-11</td>
</tr>
<tr>
<td>1.78</td>
<td>0.92623</td>
<td>8.37E-12</td>
</tr>
<tr>
<td>1.785</td>
<td>0.92748</td>
<td>-1.97E-10</td>
</tr>
<tr>
<td>1.79</td>
<td>0.92876</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>y1</td>
<td>y2</td>
</tr>
<tr>
<td>----</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>1.840</td>
<td>0.94114178588178177</td>
<td>0.9426123634</td>
</tr>
<tr>
<td>1.845</td>
<td>1.845</td>
<td>0.94410215177851575</td>
</tr>
<tr>
<td>1.850</td>
<td>1.850</td>
<td>0.945611764</td>
</tr>
<tr>
<td>1.855</td>
<td>1.855</td>
<td>0.9471394637</td>
</tr>
<tr>
<td>1.860</td>
<td>1.860</td>
<td>0.9486870417</td>
</tr>
<tr>
<td>1.865</td>
<td>1.865</td>
<td>0.9502539389</td>
</tr>
<tr>
<td>1.870</td>
<td>1.870</td>
<td>0.9518401855</td>
</tr>
<tr>
<td>1.875</td>
<td>1.875</td>
<td>0.9534458127</td>
</tr>
<tr>
<td>1.880</td>
<td>1.880</td>
<td>0.9550708530</td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
</tr>
<tr>
<td>-------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>1.885</td>
<td>0.95507085297311556</td>
<td>-2.73E-11</td>
</tr>
<tr>
<td>1.890</td>
<td>0.9567153398</td>
<td>-6.02E-11</td>
</tr>
<tr>
<td>1.895</td>
<td>0.9583793077</td>
<td>1.97E-11</td>
</tr>
<tr>
<td>1.900</td>
<td>0.9600627927</td>
<td>8.60E-11</td>
</tr>
<tr>
<td>1.905</td>
<td>0.9617658319</td>
<td>2.60E-11</td>
</tr>
<tr>
<td>1.910</td>
<td>0.9634884632</td>
<td>5.75E-11</td>
</tr>
<tr>
<td>1.915</td>
<td>0.9652307261</td>
<td>3.05E-11</td>
</tr>
<tr>
<td>1.920</td>
<td>0.9669926608</td>
<td>5.78E-13</td>
</tr>
<tr>
<td>1.925</td>
<td>0.9687743090</td>
<td>1.66E-11</td>
</tr>
<tr>
<td></td>
<td>0.9705757134</td>
<td></td>
</tr>
<tr>
<td>Value of Time (s)</td>
<td>Value of the Normalized Function (y)</td>
<td>Value of the Normalized Function (y) with Noise Reduction</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>1.930</td>
<td>0.9723969178</td>
<td>0.9723969177808085</td>
</tr>
<tr>
<td>1.935</td>
<td>0.9742379672</td>
<td>0.97423796710926569</td>
</tr>
<tr>
<td>1.940</td>
<td>0.9760989075</td>
<td>0.9760989074347727</td>
</tr>
<tr>
<td>1.945</td>
<td>0.9779797861</td>
<td>0.97797978608432246</td>
</tr>
<tr>
<td>1.950</td>
<td>0.9798806513</td>
<td>0.9798806512770295</td>
</tr>
<tr>
<td>1.955</td>
<td>0.9818015524</td>
<td>0.98180155250954815</td>
</tr>
<tr>
<td>1.960</td>
<td>0.9837425404</td>
<td>0.98374254036106346</td>
</tr>
<tr>
<td>1.965</td>
<td>0.9857036664</td>
<td>0.985703666530647</td>
</tr>
<tr>
<td>1.970</td>
<td>0.9876849838</td>
<td></td>
</tr>
</tbody>
</table>
Axiom implements the polygamma function which allows for multiple derivatives. The Psi function is a special case of the polygamma function for zero derivatives. For the purpose of this table it is defined as:
<pre>
Psi(x) == polygamma(0,x)
</pre>

The first column is the point where the Gamma function is evaluated. The second column is the value reported in the Handbook. The third column is the actual value computed by Axiom at the given point. The fourth column is the difference of Axiom’s value and the Handbook value.
<table>
<thead>
<tr>
<th>point</th>
<th>Handbook Value</th>
<th>Axiom Computed Value</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000</td>
<td>-0.5772156649</td>
<td>-0.57721566490153275</td>
<td>-1.53E-12</td>
</tr>
<tr>
<td>1.005</td>
<td>-0.5690209113</td>
<td>-0.56902091134438304</td>
<td>-4.43E-11</td>
</tr>
<tr>
<td>1.010</td>
<td>-0.5608854579</td>
<td>-0.56088545786867472</td>
<td>3.13E-11</td>
</tr>
<tr>
<td>1.015</td>
<td>-0.5528085105</td>
<td>-0.55280851559434629</td>
<td>-5.65E-12</td>
</tr>
<tr>
<td>1.020</td>
<td>-0.5447893105</td>
<td>-0.54478931045617984</td>
<td>4.38E-11</td>
</tr>
<tr>
<td>1.025</td>
<td>-0.5368270828</td>
<td>-0.53682708284938863</td>
<td>-4.93E-11</td>
</tr>
<tr>
<td>1.030</td>
<td>-0.5289210873</td>
<td>-0.5289210872854303</td>
<td>1.45E-11</td>
</tr>
<tr>
<td>1.035</td>
<td>-0.5210705921</td>
<td>-0.52107059205771</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
<td>Column 3</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>1.040</td>
<td>4.22E-11</td>
<td>-1.68E-11</td>
<td></td>
</tr>
<tr>
<td>1.045</td>
<td>4.49E-11</td>
<td>-1.57E-11</td>
<td></td>
</tr>
<tr>
<td>1.050</td>
<td>1.29E-13</td>
<td>-1.48E-11</td>
<td></td>
</tr>
<tr>
<td>1.055</td>
<td>2.47E-11</td>
<td>-1.48E-11</td>
<td></td>
</tr>
<tr>
<td>1.060</td>
<td>2.47E-11</td>
<td>-1.48E-11</td>
<td></td>
</tr>
<tr>
<td>1.065</td>
<td>2.47E-11</td>
<td>-1.48E-11</td>
<td></td>
</tr>
<tr>
<td>1.070</td>
<td>3.24E-11</td>
<td>-1.16E-11</td>
<td></td>
</tr>
<tr>
<td>1.075</td>
<td>1.16E-11</td>
<td>-1.16E-11</td>
<td></td>
</tr>
<tr>
<td>1.080</td>
<td>1.16E-11</td>
<td>-1.16E-11</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
<td>Column 3</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>1.085</td>
<td>-1.71E-12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.090</td>
<td>1.27E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.095</td>
<td>4.66E-12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.100</td>
<td>-8.78E-12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.105</td>
<td>-1.10E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.110</td>
<td>3.92E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.115</td>
<td>2.83E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.120</td>
<td>-3.42E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.125</td>
<td>-3.88E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
<td>Column 3</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>1.125</td>
<td>4.14E-12</td>
<td>0.3815738268</td>
<td>0.38157382683879215</td>
</tr>
<tr>
<td>1.130</td>
<td></td>
<td>0.3746983110</td>
<td>0.37469831095919082</td>
</tr>
<tr>
<td>1.135</td>
<td></td>
<td>0.3678666106</td>
<td>0.36786661060774969</td>
</tr>
<tr>
<td>1.140</td>
<td></td>
<td>0.3610752291</td>
<td>0.3610752291075089</td>
</tr>
<tr>
<td>1.145</td>
<td></td>
<td>0.3543266780</td>
<td>0.35432667797627904</td>
</tr>
<tr>
<td>1.150</td>
<td></td>
<td>0.3476194768</td>
<td>0.34761947675362337</td>
</tr>
<tr>
<td>1.155</td>
<td></td>
<td>0.3409531528</td>
<td>0.34095315283226135</td>
</tr>
<tr>
<td>1.160</td>
<td></td>
<td>0.3343272413</td>
<td>0.3343272412937619</td>
</tr>
<tr>
<td>1.165</td>
<td></td>
<td>0.3277412847</td>
<td>0.3277412847483927</td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
<td>Column 3</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>1.175</td>
<td>-4.83E-11</td>
<td>-0.3211948331790081</td>
<td>2.09E-11</td>
</tr>
<tr>
<td>1.180</td>
<td>-0.31468744378886082</td>
<td>1.11E-11</td>
<td></td>
</tr>
<tr>
<td>1.185</td>
<td>-0.30821868085320625</td>
<td>4.67E-11</td>
<td></td>
</tr>
<tr>
<td>1.190</td>
<td>-0.30178811569218843</td>
<td>2.53E-11</td>
<td></td>
</tr>
<tr>
<td>1.195</td>
<td>-0.2953953259418296</td>
<td>-4.18E-11</td>
<td></td>
</tr>
<tr>
<td>1.200</td>
<td>-0.28903989659218843</td>
<td>7.81E-12</td>
<td></td>
</tr>
<tr>
<td>1.205</td>
<td>-0.28272141867731704</td>
<td>2.26E-11</td>
<td></td>
</tr>
<tr>
<td>1.210</td>
<td>-0.2764394897321919</td>
<td>-3.21E-11</td>
<td></td>
</tr>
<tr>
<td>1.215</td>
<td>-0.27019371354735244</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Calculation</td>
<td>Error</td>
<td>Error Percentage</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>-------</td>
<td>------------------</td>
</tr>
<tr>
<td>1.220</td>
<td>0.2639837000</td>
<td>-4.2E-11</td>
<td>1.6E-11</td>
</tr>
<tr>
<td>1.225</td>
<td>0.257690652</td>
<td>-0.26398370004422023</td>
<td>4.65E-11</td>
</tr>
<tr>
<td>1.230</td>
<td>0.25166943069609982</td>
<td>-0.25166943069609982</td>
<td>3.90E-12</td>
</tr>
<tr>
<td>1.235</td>
<td>0.245644246789726</td>
<td>-0.245644246789726</td>
<td>3.21E-11</td>
</tr>
<tr>
<td>1.240</td>
<td>0.23949367912593666</td>
<td>-0.23949367912593666</td>
<td>2.59E-11</td>
</tr>
<tr>
<td>1.245</td>
<td>0.2334568341253</td>
<td>-0.2334568341253</td>
<td>2.16E-11</td>
</tr>
<tr>
<td>1.250</td>
<td>0.227453533762652</td>
<td>-0.227453533762652</td>
<td>2.37E-11</td>
</tr>
<tr>
<td>1.255</td>
<td>0.2214834266</td>
<td>-0.2214834266088165</td>
<td>8.88E-12</td>
</tr>
<tr>
<td>1.260</td>
<td>0.2155461686</td>
<td>-0.2155461686002652</td>
<td>1.15E-12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>1.265</td>
<td>-2.65E-13</td>
<td>1.265</td>
<td>-0.20964193</td>
</tr>
<tr>
<td>1.270</td>
<td>-0.20964193</td>
<td>1.270</td>
<td>-0.2096414193</td>
</tr>
<tr>
<td>1.275</td>
<td>-0.2096414193</td>
<td>1.275</td>
<td>-0.20964141930911384</td>
</tr>
<tr>
<td>1.280</td>
<td>-0.20964141930911384</td>
<td>1.280</td>
<td>-0.209641419309113840911382</td>
</tr>
<tr>
<td>1.285</td>
<td>-0.209641419309113840911382</td>
<td>1.285</td>
<td>-0.209641419309113840911382090911381</td>
</tr>
<tr>
<td>1.290</td>
<td>-0.209641419309113840911382090911381</td>
<td>1.290</td>
<td>-0.2096414193091138409113820909113810911380</td>
</tr>
<tr>
<td>1.295</td>
<td>-0.2096414193091138409113820909113810911380</td>
<td>1.295</td>
<td>-0.2096414193091138409113820909113810911380091137999</td>
</tr>
<tr>
<td>1.300</td>
<td>-0.2096414193091138409113820909113810911380091137999</td>
<td>1.300</td>
<td>-0.20964141930911384091138209091138109113800911379990911379984</td>
</tr>
<tr>
<td>1.305</td>
<td>-0.20964141930911384091138209091138109113800911379990911379984</td>
<td>1.305</td>
<td>-0.209641419309113840911382090911381091138009113799909113799840911379974</td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
<td>Column 3</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>1.310</td>
<td>-0.1579078803</td>
<td>-0.15790788033614178</td>
<td>-3.15E-11</td>
</tr>
<tr>
<td>1.315</td>
<td>-0.1523105782</td>
<td>-0.15231057824556994</td>
<td>-4.55E-11</td>
</tr>
<tr>
<td>1.320</td>
<td>-0.1467423568</td>
<td>-0.14674235679599959</td>
<td>4.00E-12</td>
</tr>
<tr>
<td>1.325</td>
<td>-0.1412029305</td>
<td>-0.14120293051842803</td>
<td>-1.84E-11</td>
</tr>
<tr>
<td>1.330</td>
<td>-0.1356920180</td>
<td>-0.13569201796416941</td>
<td>3.58E-11</td>
</tr>
<tr>
<td>1.335</td>
<td>-0.1302093416</td>
<td>-0.13020934163201769</td>
<td>-3.20E-11</td>
</tr>
<tr>
<td>1.340</td>
<td>-0.1247546279</td>
<td>-0.12475462789700376</td>
<td>2.99E-12</td>
</tr>
<tr>
<td>1.345</td>
<td>-0.1193276069</td>
<td>-0.11932760694070754</td>
<td>-4.07E-11</td>
</tr>
<tr>
<td>1.350</td>
<td>-0.1139280127</td>
<td>-0.11392801268308839</td>
<td>...</td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
<td>Column 3</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>1.69E-11</td>
<td>1.355</td>
<td>-0.1085555827</td>
<td>-0.1085558271580501</td>
</tr>
<tr>
<td>1.360</td>
<td>-0.1032100582</td>
<td>-0.10321005823697738</td>
<td>-3.69E-11</td>
</tr>
<tr>
<td>1.365</td>
<td>-0.0978911840</td>
<td>-0.097891183987354968</td>
<td>1.26E-11</td>
</tr>
<tr>
<td>1.370</td>
<td>-0.0925987082</td>
<td>-0.092598708187860979</td>
<td>1.21E-11</td>
</tr>
<tr>
<td>1.375</td>
<td>-0.0873323825</td>
<td>-0.087332382478473081</td>
<td>2.15E-11</td>
</tr>
<tr>
<td>1.380</td>
<td>-0.0820919619</td>
<td>-0.082091961858406615</td>
<td>4.15E-11</td>
</tr>
<tr>
<td>1.385</td>
<td>-0.0768772046</td>
<td>-0.076877204627574525</td>
<td>4.15E-11</td>
</tr>
<tr>
<td>1.390</td>
<td>-0.0716878723</td>
<td>-0.071687872329281643</td>
<td>-2.75E-11</td>
</tr>
<tr>
<td>1.395</td>
<td>-0.0665237297</td>
<td>-0.066523729694132228</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>First Column</td>
<td>Second Column</td>
<td>Third Column</td>
</tr>
<tr>
<td>--------</td>
<td>--------------</td>
<td>---------------</td>
<td>--------------</td>
</tr>
<tr>
<td>1.400</td>
<td>-0.061384546</td>
<td>-0.061384544585116108</td>
<td>1.48E-11</td>
</tr>
<tr>
<td>1.405</td>
<td>-0.0562700879</td>
<td>-0.056270087943841696</td>
<td>-4.38E-11</td>
</tr>
<tr>
<td>1.410</td>
<td>-0.0511801337</td>
<td>-0.051180133737897426</td>
<td>-3.78E-11</td>
</tr>
<tr>
<td>1.415</td>
<td>-0.0461144589</td>
<td>-0.046114458909301992</td>
<td>-9.30E-12</td>
</tr>
<tr>
<td>1.420</td>
<td>-0.0410728433</td>
<td>-0.041072843324024277</td>
<td>-2.40E-11</td>
</tr>
<tr>
<td>1.425</td>
<td>-0.0360550697</td>
<td>-0.036055069722547906</td>
<td>-2.25E-11</td>
</tr>
<tr>
<td>1.430</td>
<td>-0.0310609237</td>
<td>-0.031060923671447194</td>
<td>2.85E-11</td>
</tr>
<tr>
<td>1.435</td>
<td>-0.0260901935</td>
<td>-0.02609019351596098</td>
<td>-1.59E-11</td>
</tr>
<tr>
<td>1.440</td>
<td>-0.0211426703</td>
<td>-0.021142670333530678</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Difference</td>
<td>Relative Difference</td>
<td>Absolute Difference</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
<td>---------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>1.445</td>
<td>-0.0162181479</td>
<td>-0.016218147888283685</td>
<td>1.17E-11</td>
</tr>
<tr>
<td>1.450</td>
<td>-0.0113164226</td>
<td>-0.0113164225886445718</td>
<td>1.35E-11</td>
</tr>
<tr>
<td>1.455</td>
<td>-0.0064372934</td>
<td>-0.0064372934326406561</td>
<td>-3.26E-11</td>
</tr>
<tr>
<td>1.460</td>
<td>-0.0015805620</td>
<td>-0.0015805619870833398</td>
<td>1.29E-11</td>
</tr>
<tr>
<td>1.465</td>
<td>0.0032539677</td>
<td>0.0032539676763745362</td>
<td>-2.36E-11</td>
</tr>
<tr>
<td>1.470</td>
<td>0.0080664890</td>
<td>0.0080664890113649745</td>
<td>-1.11E-11</td>
</tr>
<tr>
<td>1.475</td>
<td>0.0128571930</td>
<td>0.012857193039295334</td>
<td>3.92E-11</td>
</tr>
<tr>
<td>1.480</td>
<td>0.0176262684</td>
<td>0.017626268388849287</td>
<td>-1.11E-11</td>
</tr>
<tr>
<td>1.485</td>
<td>0.0223739013</td>
<td>0.022373901334705404</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
<td>Column 3</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>1.490</td>
<td>0.027102758</td>
<td>0.02710275835486465</td>
<td>3.47E-11</td>
</tr>
<tr>
<td>1.495</td>
<td>0.0318056736</td>
<td>0.031805673570971468</td>
<td>3.54E-11</td>
</tr>
<tr>
<td>1.500</td>
<td>0.03648999740</td>
<td>0.0364899973978576673</td>
<td>-2.90E-11</td>
</tr>
<tr>
<td>1.505</td>
<td>0.0411536543</td>
<td>0.041153654289123542</td>
<td>-2.14E-11</td>
</tr>
<tr>
<td>1.510</td>
<td>0.0457967896</td>
<td>0.045796789561914686</td>
<td>-1.08E-11</td>
</tr>
<tr>
<td>1.515</td>
<td>0.0504195527</td>
<td>0.050419552719128236</td>
<td>-3.80E-11</td>
</tr>
<tr>
<td>1.520</td>
<td>0.0550221146</td>
<td>0.055022114579561307</td>
<td>-2.04E-11</td>
</tr>
<tr>
<td>1.525</td>
<td>0.0596046439</td>
<td>0.05960464389166209</td>
<td>-8.33E-12</td>
</tr>
<tr>
<td>1.530</td>
<td>0.0641673074</td>
<td>0.064167307366077231</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
<td>Column 3</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>1.535</td>
<td>-3.39E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.540</td>
<td>0.0687102697</td>
<td>0.068710269707385141</td>
<td>7.38E-12</td>
</tr>
<tr>
<td>1.545</td>
<td>0.07323693645366138</td>
<td>0.07323693645366138</td>
<td>4.53E-11</td>
</tr>
<tr>
<td>1.550</td>
<td>0.07773773996562497</td>
<td>0.07773773996562497</td>
<td>9.96E-9</td>
</tr>
<tr>
<td>1.555</td>
<td>0.0822225675</td>
<td>0.082222567539644631</td>
<td>3.96E-11</td>
</tr>
<tr>
<td>1.560</td>
<td>0.0866883334</td>
<td>0.08668833354262828</td>
<td>-4.57E-11</td>
</tr>
<tr>
<td>1.565</td>
<td>0.0911351925</td>
<td>0.091135192540635401</td>
<td>4.06E-11</td>
</tr>
<tr>
<td>1.570</td>
<td>0.0955632984</td>
<td>0.095563298402570163</td>
<td>2.57E-12</td>
</tr>
<tr>
<td>1.575</td>
<td>0.0999728024</td>
<td>0.09997280244444731</td>
<td>4.44E-11</td>
</tr>
<tr>
<td>1.580</td>
<td>0.1043638544</td>
<td>0.10436385439851947</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
<td>Column 3</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>1.580</td>
<td>-1.48E-12</td>
<td>0.1087366023</td>
<td>0.10873660225178161</td>
</tr>
<tr>
<td>1.585</td>
<td>0.1130911923</td>
<td>0.11309119227228603</td>
<td>-2.77E-11</td>
</tr>
<tr>
<td>1.590</td>
<td>0.1174277690</td>
<td>0.11742776903501095</td>
<td>3.50E-11</td>
</tr>
<tr>
<td>1.595</td>
<td>0.1217464754</td>
<td>0.1217464754723916</td>
<td>4.72E-11</td>
</tr>
<tr>
<td>1.600</td>
<td>0.1260474528</td>
<td>0.12604745277347584</td>
<td>-2.65E-11</td>
</tr>
<tr>
<td>1.605</td>
<td>0.1303308407</td>
<td>0.13033084065991318</td>
<td>-4.00E-11</td>
</tr>
<tr>
<td>1.610</td>
<td>0.1345967772</td>
<td>0.13459677715844587</td>
<td>-4.15E-11</td>
</tr>
<tr>
<td>1.615</td>
<td>0.1388453988</td>
<td>0.13884539875025736</td>
<td>-4.15E-11</td>
</tr>
<tr>
<td>1.620</td>
<td>0.1430768404</td>
<td>0.14307684036898005</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
<td>Column 3</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>1.625</td>
<td>-3.10E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.630</td>
<td></td>
<td>0.1472912354</td>
<td></td>
</tr>
<tr>
<td>1.635</td>
<td></td>
<td>0.14729123542343325</td>
<td></td>
</tr>
<tr>
<td>1.640</td>
<td></td>
<td>1.99E-11</td>
<td></td>
</tr>
<tr>
<td>1.645</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.650</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.655</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.660</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.665</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>Y1</td>
<td>Y2</td>
<td>Y3</td>
</tr>
<tr>
<td>----</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>1.670</td>
<td>4.17E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.675</td>
<td>2.67E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.680</td>
<td>2.17E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.685</td>
<td>1.67E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.690</td>
<td>1.17E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.695</td>
<td>0.67E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.700</td>
<td>0.17E-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.705</td>
<td>-5.31E-12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.710</td>
<td>-6.31E-12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Parameter 1</td>
<td>Parameter 2</td>
<td>Parameter 3</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>1.715</td>
<td>0.2203789037</td>
<td>0.2203789036859658</td>
<td>-1.03E-11</td>
</tr>
<tr>
<td>1.720</td>
<td>0.2242928871</td>
<td>0.22429288714615725</td>
<td>4.61E-11</td>
</tr>
<tr>
<td>1.725</td>
<td>0.2281922037</td>
<td>0.22819220368823745</td>
<td>-1.17E-11</td>
</tr>
<tr>
<td>1.730</td>
<td>0.2320769593</td>
<td>0.23207695930067274</td>
<td>6.72E-13</td>
</tr>
<tr>
<td>1.735</td>
<td>0.2359472589</td>
<td>0.23594725885781176</td>
<td>-4.21E-11</td>
</tr>
<tr>
<td>1.740</td>
<td>0.2398032061</td>
<td>0.23980320613509676</td>
<td>3.50E-11</td>
</tr>
<tr>
<td>1.745</td>
<td>0.2436449038</td>
<td>0.24364490382402559</td>
<td>2.40E-11</td>
</tr>
<tr>
<td>1.750</td>
<td>0.2474724535</td>
<td>0.24747245354666612</td>
<td>4.68E-11</td>
</tr>
<tr>
<td>1.755</td>
<td>0.2512859559</td>
<td>0.25128595587109781</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Parameter 1</th>
<th>Parameter 2</th>
<th>Parameter 3</th>
<th>Parameter 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.715</td>
<td>0.2203789037</td>
<td>0.2203789036859658</td>
<td>-1.03E-11</td>
<td></td>
</tr>
<tr>
<td>1.720</td>
<td>0.2242928871</td>
<td>0.22429288714615725</td>
<td>4.61E-11</td>
<td></td>
</tr>
<tr>
<td>1.725</td>
<td>0.2281922037</td>
<td>0.22819220368823745</td>
<td>-1.17E-11</td>
<td></td>
</tr>
<tr>
<td>1.730</td>
<td>0.2320769593</td>
<td>0.23207695930067274</td>
<td>6.72E-13</td>
<td></td>
</tr>
<tr>
<td>1.735</td>
<td>0.2359472589</td>
<td>0.23594725885781176</td>
<td>-4.21E-11</td>
<td></td>
</tr>
<tr>
<td>1.740</td>
<td>0.2398032061</td>
<td>0.23980320613509676</td>
<td>3.50E-11</td>
<td></td>
</tr>
<tr>
<td>1.745</td>
<td>0.2436449038</td>
<td>0.24364490382402559</td>
<td>2.40E-11</td>
<td></td>
</tr>
<tr>
<td>1.750</td>
<td>0.2474724535</td>
<td>0.24747245354666612</td>
<td>4.68E-11</td>
<td></td>
</tr>
<tr>
<td>1.755</td>
<td>0.2512859559</td>
<td>0.25128595587109781</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>y_1</td>
<td>y_2</td>
<td>y_3</td>
<td>y_4</td>
</tr>
<tr>
<td>----</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>1.760</td>
<td>2.550855103</td>
<td>0.25508551032368809</td>
<td>0.25508551032368809</td>
<td>-2.89E-11</td>
</tr>
<tr>
<td>1.765</td>
<td>2.588712154</td>
<td>0.25887121540503744</td>
<td>0.25887121540503744</td>
<td>2.36E-11</td>
</tr>
<tr>
<td>1.770</td>
<td>2.6264131686</td>
<td>0.262641316860276249</td>
<td>0.262641316860276249</td>
<td>5.03E-12</td>
</tr>
<tr>
<td>1.775</td>
<td>2.664014664</td>
<td>0.2664014664052331</td>
<td>0.2664014664052331</td>
<td>2.76E-12</td>
</tr>
<tr>
<td>1.780</td>
<td>2.701462043</td>
<td>0.27014620431488368</td>
<td>0.27014620431488368</td>
<td>5.23E-12</td>
</tr>
<tr>
<td>1.785</td>
<td>2.738774769</td>
<td>0.27387747686131236</td>
<td>0.27387747686131236</td>
<td>1.48E-11</td>
</tr>
<tr>
<td>1.790</td>
<td>2.775953776</td>
<td>0.27759537761416786</td>
<td>0.27759537761416786</td>
<td>-3.86E-11</td>
</tr>
<tr>
<td>1.795</td>
<td>2.812999992</td>
<td>0.281299991958266</td>
<td>0.281299991958266</td>
<td>1.41E-11</td>
</tr>
<tr>
<td>1.800</td>
<td>2.849914333</td>
<td>0.2849914332938619</td>
<td>0.2849914332938619</td>
<td>-4.17E-12</td>
</tr>
</tbody>
</table>
| Temperature | a | b | c | d 
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.805</td>
<td>-6.13E-12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.810</td>
<td>1.805</td>
<td>0.2923351012</td>
<td>0.29233510118877948</td>
<td>-1.12E-11</td>
</tr>
<tr>
<td>1.815</td>
<td>1.810</td>
<td>0.2959875138</td>
<td>0.29598751379626109</td>
<td>-3.73E-12</td>
</tr>
<tr>
<td>1.820</td>
<td>1.815</td>
<td>0.3032539367</td>
<td>0.3032539368840539</td>
<td>-1.15E-11</td>
</tr>
<tr>
<td>1.825</td>
<td>1.820</td>
<td>0.3068681205</td>
<td>0.30686812049650136</td>
<td>-3.49E-12</td>
</tr>
<tr>
<td>1.830</td>
<td>1.825</td>
<td>0.3104697335</td>
<td>0.31046973346594764</td>
<td>-3.40E-12</td>
</tr>
<tr>
<td>1.835</td>
<td>1.830</td>
<td>0.3140586802</td>
<td>0.31405868023156859</td>
<td>3.15E-11</td>
</tr>
<tr>
<td>1.840</td>
<td>1.835</td>
<td>0.3176355846</td>
<td>0.31763558459703256</td>
<td>3.15E-11</td>
</tr>
<tr>
<td>1.850</td>
<td>0.3211999895</td>
<td>0.32119998954547946</td>
<td>4.54E-11</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>----------------------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>1.855</td>
<td>0.32475215724997797</td>
<td>0.32475215724997797</td>
<td>4.99E-11</td>
<td></td>
</tr>
<tr>
<td>1.860</td>
<td>0.32829216908382075</td>
<td>0.32829216908382075</td>
<td>-1.61E-11</td>
<td></td>
</tr>
<tr>
<td>1.865</td>
<td>0.33182010563065989</td>
<td>0.33182010563065989</td>
<td>3.06E-11</td>
<td></td>
</tr>
<tr>
<td>1.870</td>
<td>0.335360469448569</td>
<td>0.335360469448569</td>
<td>-5.51E-12</td>
<td></td>
</tr>
<tr>
<td>1.875</td>
<td>0.33884007130944738</td>
<td>0.33884007130944738</td>
<td>9.44E-12</td>
<td></td>
</tr>
<tr>
<td>1.880</td>
<td>0.3423225774952925</td>
<td>0.3423225774952925</td>
<td>4.95E-11</td>
<td></td>
</tr>
<tr>
<td>1.885</td>
<td>0.34581268353806771</td>
<td>0.34581268353806771</td>
<td>3.80E-11</td>
<td></td>
</tr>
<tr>
<td>1.890</td>
<td>0.3492814255713492</td>
<td>0.3492814255713492</td>
<td>5.82E-11</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Column 1</td>
<td>Column 2</td>
<td>Column 3</td>
<td>Column 4</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>1.895</td>
<td>-4.28E-11</td>
<td>0.3527385595567692</td>
<td>-4.32E-11</td>
<td></td>
</tr>
<tr>
<td>1.900</td>
<td>0.35618461612</td>
<td>0.3527385595567692</td>
<td>-3.59E-11</td>
<td></td>
</tr>
<tr>
<td>1.905</td>
<td>0.3596183049</td>
<td>0.356184616116406026</td>
<td>-7.88E-12</td>
<td></td>
</tr>
<tr>
<td>1.910</td>
<td>0.3630410646</td>
<td>0.3596183049</td>
<td>4.88E-11</td>
<td></td>
</tr>
<tr>
<td>1.915</td>
<td>0.3664525136</td>
<td>0.3630410646</td>
<td>4.58E-11</td>
<td></td>
</tr>
<tr>
<td>1.920</td>
<td>0.3698527244</td>
<td>0.36645251364580167</td>
<td>6.40E-12</td>
<td></td>
</tr>
<tr>
<td>1.925</td>
<td>0.3732417688</td>
<td>0.36985272440640171</td>
<td>-2.53E-11</td>
<td></td>
</tr>
<tr>
<td>1.930</td>
<td>0.3766197179</td>
<td>0.37324176877469795</td>
<td>2.34E-11</td>
<td></td>
</tr>
<tr>
<td>1.935</td>
<td>0.3799866424</td>
<td>0.37661971792349891</td>
<td>2.34E-11</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>--------</td>
<td>--------------------------</td>
<td>--------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>1.940</td>
<td>3.833426119</td>
<td>-3.74E-11</td>
<td>1.940</td>
<td>0.38334261194674013</td>
</tr>
<tr>
<td>1.945</td>
<td>0.38668769599</td>
<td>-1.62E-11</td>
<td>1.950</td>
<td>0.38668769588372298</td>
</tr>
<tr>
<td>1.955</td>
<td>0.3933454805</td>
<td>-4.13E-11</td>
<td>1.960</td>
<td>0.39334548045868012</td>
</tr>
<tr>
<td>1.965</td>
<td>0.3999605371</td>
<td>2.54E-12</td>
<td>1.970</td>
<td>0.39996053710254509</td>
</tr>
<tr>
<td>1.975</td>
<td>0.4065333970</td>
<td>-3.40E-11</td>
<td>1.980</td>
<td>0.4065333969592627</td>
</tr>
<tr>
<td>1.985</td>
<td>0.4098041664</td>
<td>4.00E-11</td>
<td>1.990</td>
<td>0.40980416644999071</td>
</tr>
<tr>
<td>Value</td>
<td>y_1</td>
<td>y_2</td>
<td>Error</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
<td>-----</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>1.985</td>
<td>0.4130645815688626</td>
<td>0.4130645815688626</td>
<td>-3.11E-11</td>
<td></td>
</tr>
<tr>
<td>1.990</td>
<td>0.41631470604541487</td>
<td>0.41631470604541487</td>
<td>4.54E-11</td>
<td></td>
</tr>
<tr>
<td>1.995</td>
<td>0.41955460302810832</td>
<td>0.41955460302810832</td>
<td>2.81E-11</td>
<td></td>
</tr>
<tr>
<td>2.000</td>
<td>0.42278433509846725</td>
<td>0.42278433509846725</td>
<td>-1.53E-12</td>
<td></td>
</tr>
</tbody>
</table>

---

draw.xhtml

--- draw.xhtml ---
## Overview

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A function of one variable</td>
<td>$y = f(x)$</td>
</tr>
<tr>
<td>A parametrically defined curve</td>
<td>$(x(t), y(t))$</td>
</tr>
<tr>
<td>A solution to a polynomial equation</td>
<td>$p(x,y) = 0$</td>
</tr>
</tbody>
</table>

### Three Dimensional Plots

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A function of two variable</td>
<td>$y = f(x,y)$</td>
</tr>
<tr>
<td>A parametrically defined tube</td>
<td></td>
</tr>
</tbody>
</table>
\[(x(t), y(t), z(t))\]

\[(x(u,v), y(u,v), z(u,v))\]

---

draw2donevariable.xhtml

--- draw2donevariable.xhtml ---

\getchunk{standard head}

\begin{verbatim}
\script{type="text/javascript"}

function CommandLine(arg) {
    myfunc = document.getElementById('function').value;
    myvar = document.getElementById('var').value;
    myfrom = document.getElementById('range1').value;
    myto = document.getElementById('range2').value;
    mytitle = document.getElementById('title1').value;
    if (mytitle == '') {
        ans = 'draw(' + myfunc + ', ' + myvar + '=' + myfrom + '..' + myto + ');
    } else {
        ans = 'draw(' + myfunc + ', ' + myvar + '=' + myfrom + '..' + myto + ', title="" + mytitle + '"');
    }
    alert(ans);
    return(ans);
}
\end{verbatim}

\getchunk{showfullanswer}

\getchunk{axiom talker}

\end{verbatim}

\getchunk{page head}

\begin{center}
Drawing $y=f(x)$

where $y$ is the dependent variable and

where $x$ is the independent variable

<table>
<thead>
<tr>
<th>What function $f$ would you like to draw?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x\cos(x)$</td>
</tr>
</tbody>
</table>

Enter independent variable and range:
Variable: $x$
ranges from: 0 to: 30

Optionally enter a title for your curve:
y = $x\cos(x)$

---

draw2definedcurve.xhtml
mytitle = document.getElementById('title1').value;
if (mytitle == "") {
    ans =
    'draw(curve('+myfunc1+','+myfunc2+'),'+myvar+'='+myfrom+'..'+myto+');
} else {
    ans =
    'draw(curve('+myfunc1+','+myfunc2+'),'+myvar+'='+myfrom+'..'+myto+
    ',title="'+mytitle+'")';
}
alert(ans);
return(ans);
}

\getchunk{showfullanswer}
\getchunk{axiom talker}

</script>
</head>
<body>
\getchunk{page head}
<center>
Drawing a parametrically defined curve<br/>
(f1(t),f2(t))<br/>
in terms of two functions f1 and f2<br/>
and an independent variable t
</center>
<table>
<tr>
<td>
Enter the two functions:<br/>
Function 1:<br/>
<input type="text" id="function1" size="80" tabindex="10"
    value="-9*sin(4*t/5)"/><br/>
Function 2:<br/>
<input type="text" id="function2" size="80" tabindex="20"
    value="8*sin(t)"/><br/>
</td>
</tr>
<tr>
<td>
Enter the independent variable and range:<br/>
Variable:<br/>
<input type="text" id="var" size="10" tabindex="30" value="t"/>
ranges from:<br/>
<input type="text" id="range1" size="10" tabindex="40" value="-5*%pi"/>
to:<br/>
<input type="text" id="range2" size="10" tabindex="45" value="5*%pi"/></td>
</tr>
<tr>
<td>
Optionally enter a title for your curve:<br/>
<input type="text" id="title1" size="20" tabindex="50"
    value="Lissajous"/>
</td>
</tr>
Plotting the solution to \( p(x,y)=0 \), where \( p \) is a polynomial in two variables \( x \) and \( y \)

Enter the polynomial \( p \):<br/>

```latex
\begin{align*}
\text{value} &= y^2+7xy-(x^3+16x)\end{align*}
```
Enter the variables:<br/>
Variable 1:<br/>
ranges from:<br/>Variable 2:<br/>
ranges from:<br/>Optionally enter a title for your curve:
CHAPTER 1. OVERVIEW

```javascript
}  
alert(ans);  
return(ans);  
}

\getchunk{showfullanswer}
\getchunk{axiom talker}

\script
  \head
  \body
\getchunk{page head}
\center
  Drawing $z=f(x,y)$
  where $z$ is the dependent variable and
  where $x, y$ are the independent variables
\center
\table
\tr
  <td>
    What function $f$ which you like to draw?  
    <input type="text" id="function1" size=80" tabindex="10"  
      value="exp(cos(x-y)-sin(x*y))-2"/>
  </td>
\tr
  <td>
    Enter the independent variables and ranges:
    Variable 1:  
    <input type="text" id="var1" size=10" tabindex="30" value="x"/>
    ranges from:  
    <input type="text" id="range11" size=10" tabindex="40" value="-5"/>
    to:  
    <input type="text" id="range21" size=10" tabindex="45" value="5"/>
    Variable 2:  
    <input type="text" id="var2" size=10" tabindex="46" value="y"/>
    ranges from:  
    <input type="text" id="range12" size=10" tabindex="47" value="-5"/>
    to:  
    <input type="text" id="range22" size=10" tabindex="48" value="5"/>
  </td>
\tr
  <td>
    Optionally enter a title for your curve:
    <input type="text" id="title1" size=20" tabindex="50"/>
  </td>
\tr
\table
\getchunk{continue button}
\getchunk{answer field}
\getchunk{page foot}
Drawing a parametrically defined curve: \((f_1(t), f_2(t), f_3(t))\) in terms of three functions \(f_1\), \(f_2\), and \(f_3\) and an independent variable \(t\).

Enter the three functions of the independent variable:

Function \(f_1\):

\[
1.3 \cos(2t) \cos(4t) + \sin(4t) \cos(t)
\]

Function \(f_2\):

\[
1.3 \sin(2t) \cos(4t) - \sin(4t) \sin(t)
\]

Function \(f_3\):

\[
2.5 \cos(4t)
\]
<tr>
<td>
Enter the independent variable and range:<br/>
Variable:<br/>
<input type="text" id="var1" size="10" tabindex="40" value="t"/>
ranges from:<br/>
<input type="text" id="range1" size="10" tabindex="50" value="0"/>
<
</td>
</tr>
<tr>
<td>
Optionally enter a title for your surface:<br/>
<input type="text" id="title1" size="20" tabindex="70" value="knot"/>
</td>
</tr>
</table>

---

draw3ddefinedsurface.xhtml

--- draw3ddefinedsurface.xhtml ---

<script type="text/javascript">
function commandline(arg) {
  myfunc1 = document.getElementById('function1').value;
  myfunc2 = document.getElementById('function2').value;
  myfunc3 = document.getElementById('function3').value;
  myvar1 = document.getElementById('var1').value;
  myfrom1 = document.getElementById('range1').value;
  myto1 = document.getElementById('range2').value;
  myvar2 = document.getElementById('var11').value;
  myfrom2 = document.getElementById('range11').value;
  myto2 = document.getElementById('range21').value;
  mytitle = document.getElementById('title1').value;
  if (mytitle == "") {
    ans='draw(surface('+myfunc1+','+myfunc2+','+myfunc3+'),'+
    myvar1+'='+myfrom1+'..'+myto1+','+
    myvar2+'='+myfrom2+'..'+myto2+');
  } else {
    ans='draw(surface('+myfunc1+','+myfunc2+','+myfunc3+'),'+
    myvar1+'='+myfrom1+'..'+myto1+','+
    myvar2+'='+myfrom2+'..'+myto2+',title=="'+mytitle+'")';
  }
  alert(ans);
</script>
return(ans);
}
</script>

</head>
<body>
<center>
Drawing a parametrically defined surface<br/>
(f1(u,v), f2(u,v), f3(u,v))<br/>
in terms of three functions f1, f2, and f3<br/>
and two independent variables u and v
</center>
<table>
<tr><td>
Enter the three functions of the independent variable:<br/>
Function f1:<br/>Function f2:<br/>Function f3:<br/></td></tr>
<tr><td>
Enter the independent variables and range:<br/>
Variable 1:<br/>ranges from:<br/>to:<br/>
Variable 2:<br/>ranges from:<br/>to:<br/></td></tr>
<tr><td>
Optionally enter a title for your surface:<br/></td></tr>
</table>
In this section we discuss Axiom’s facilities for solving differential equations in closed-form and in series.

Axiom provides facilities for closed-form solution of single differential equations of the following kinds:

- linear ordinary differential equations
- non-linear first order ordinary differential equations when integrating factors can be found just by integration

For a discussion of the solution of systems of linear and polynomial equations, see <a href="axbook/section-8.5.xhtml">Solution of Linear and Polynomial Equations</a>.

- Closed-Form Solutions of Linear Differential Equations
- Closed-Form Solutions of Non-Linear Differential Equations
- Power Series Solutions of Differential Equations
A differential equation is an equation involving an unknown function and one or more of its derivatives. The equation is called ordinary if derivatives with respect to only one dependent variable appear in the equation (it is called partial otherwise). The package <a href="db.xhtml?ElementaryFunctionODESolver">ElementaryFunctionODESolver</a> provides the top-level operation <a href="dbopsolve.xhtml">solve</a> for finding closed-form solutions of ordinary differential equations.

To solve a differential equation, you must first create an operator for the unknown function. We let $y$ be the unknown function in terms of $x$.

You then type the equation using <a href="dbopd.xhtml">D</a> to create the derivatives of the unknown function $y(x)$ where $x$ is any symbol you choose (the so-called dependent variable). This is how you enter the equation

$$y'' + y' + y = 0$$
The simplest way to invoke the `<a href="dbopsolve.xhtml">solve</a>` command is with three arguments,

- the differential equation
- the operator representing the unknown function
- the dependent variable

So, to solve the above equation, we enter this.

Since linear ordinary differential equations have infinitely many solutions, `<a href="dbopsolve.xhtml">solve</a>` returns a particular solution \( f_p \) and a basis \( f_1, \ldots, f_n \) for the solutions of the corresponding homogeneous equation. Any expression of the form \( f_p + c_1 f_1 + \ldots + c_n f_n \) where the \( c_i \) do not involve the dependent variable is also a solution. This is similar to what you get when you solve systems of linear algebraic equations.

A way to select a unique solution is to specify initial conditions: choose a value \( a \) for the dependent variable and specify the values of the unknown function and its derivatives at \( a \). If the number of initial conditions is equal to the order of the equation, then the solution is unique (if it exists in closed form) and `<a href="dbopsolve.xhtml">solve</a>` tries to find it. To specify initial conditions to `<a href="dbopsolve.xhtml">solve</a>`, use an `<a href="db.xhtml?Equation">Equation</a>` of the form \( x=a \) for the third parameter instead of the dependent variable, and add a fourth parameter consisting of the list of values \( y(a), y'(a), \ldots \)

To find the solution of \( y''+y=0 \) satisfying \( y(0)=y'(0)=1 \), do this.

You can omit the "\( =0 \)" when you enter the equation to be solved.

Axiom is not limited to linear differential equations with constant coefficients. It can also find solutions when the coefficients are rational or algebraic functions of the dependent variable. Furthermore,
Axiom is not limited by the order of the equation. Axiom can solve the following third order equations with polynomial coefficients.

\[
\text{deq} := x^3 \frac{d^3}{dx^3} y(x) + x^2 \frac{d^2}{dx^2} y(x) - 2x \frac{dy}{dx} + 2y = 2x^4
\]

On the other hand, and in contrast with the operation \(\text{integrate}\) it can happen that Axiom finds no solution and that some closed-form solution still exists. While it is mathematically complicated to describe exactly when the solutions are guaranteed to be found, the following statements are correct and form good guidelines for linear ordinary differential equations.

\[
\text{If the coefficients are constants, Axiom finds a complete basis of solutions (i.e. all solutions).}
\]

\[
\text{If the coefficients are rational functions in the dependent variable, Axiom at least finds all solutions that do not involve algebraic functions.}
\]

Note that this last statement does not mean that Axiom does not find the solutions that are algebraic functions. It means that it is not guaranteed that the algebraic function solutions will be found. This is an example where all the algebraic solutions are found.
Let's solve the differential equation
\[ y' = \frac{y}{x + y \log y} \]
Using the notation
\[ m(x,y) + n(x,y)y' = 0 \]
we have \( m = -y \) and \( n = x + y \log y \)

We first check for exactness, that is, does \( \frac{dm}{dy} = \frac{dn}{dx} \)?

We are given a non-linear first order ordinary differential equation manually when an integrating factor can be found just by integration. At the end, we show you how to solve it directly.
This is not zero, so the equation is not exact. Therefore we must look for an integrating factor, that is, a function \( \mu(x,y) \) such that \( \frac{d(\mu m)}{dy} = \frac{d(\mu n)}{dx} \). Normally, we first search for \( \mu(x,y) \) depending only on \( x \) or only on \( y \). Let's search for such a \( \mu(x) \) first.

If the above is zero for a function \( \mu \) that does not depend on \( y \), then \( \mu(x) \) is an integrating factor. Let's look for one that depends on \( x \) only. The solution depends on \( y \), so there is no integrating factor that depends on \( x \) only. Let's look for one that depends on \( y \) only.

We've found one. The above \( \mu(y) \) is an integrating factor. We must multiply our initial equation (that is, \( m \) and \( n \)) by the integrating factor.
Let's check for exactness.

We must solve the exact equation, that is, find a function $s(x,y)$ such that $ds/dx=m$ and $ds/dy=n$. We start by writing

\[ s(x,y) = h(y) + \int m(x) \, dx \]

where $h(y)$ is an unknown function of $y$. This guarantees that $ds/dx=m$.

All we want is to find $h(y)$ such that $ds/dy=n$. 
The above particular solution is the \( h(y) \) we want, so we just replace \( h(y) \) by it in the implicit solution.

A first integral of the initial equation is obtained by setting this result equal to an arbitrary constant.

Now that we've seen how to solve the equation "by hand" we show you how to do it with the `<a href="dbopsolve.xhtml">solve</a>` operation. First define \( y \) to be an operator.

Next we create the differential equation.

Finally, we solve it.
The command to solve differential equations in power series around a particular initial point with specific initial conditions is called \texttt{seriesSolve}. It can take a variety of parameters, so we illustrate its use with some examples.

Since the coefficients of some solutions are quite large, we reset the default to compute only seven terms.

You can solve a single nonlinear equation of any order. For example, we solve

\[
y''' = \sin(y'') \cdot \exp(y) + \cos(x)
\]

subject to \(y(0)=1, \ y'(0)=0, \ y''(0)=0\)

We first tell Axiom that the symbol 'y denotes a new operator.
Solve it around $x=0$ with the initial conditions $y(0)=1$, $y'(0)=y''(0)=0$.

You can also solve a system of nonlinear first order equations. For example, we solve a system that has $\tan(t)$ and $\sec(t)$ as solutions.

We tell Axiom that $x$ is also an operator.

Enter the two equations forming our system.

Solve the system around $t=0$ with the initial conditions $x(0)=0$ and $y(0)=1$. Notice that since we give the unknowns in the order $[x,y]$, the answer is a list of two series in the order $[\text{series for } x(t), \text{series for } y(t)]$. The order in which we give the equations and the initial conditions has no effect on the order of the solution.
Axiom lets you solve equations of various types:

- [Solution of Systems of Linear Equations](#) - Solve systems of linear equations
- [Solution of a Single Polynomial Equation](#) - Find roots of polynomials
- [Solution of a System of Polynomial Equations](#) - Solve systems of polynomial equations
- [Solution of Differential Equations](#) - Closed form and series solutions of differential equations
You can use the operation <a href="dbopsolve.xhtml">solve</a> to solve systems of linear equations.

The operation <a href="dbopsolve.xhtml">solve</a> takes two arguments, the list of equations and the list of the unknowns to be solved for. A system of linear equations need not have a unique solution.

To solve the linear system:

\[
\begin{align*}
    x + y + x &= 8 \\
    3x - 2y + z &= 0 \\
    x + 2y + 2z &= 17 
\end{align*}
\]

evaluate this expression.

\[
\begin{align*}
    x + 2y + 3z &= 2 \\
    2x + 3y + 4z &= 2 \\
    3x + 4y + 5z &= 2 
\end{align*}
\]

Parameters are given as new variables starting with a percent sign and "\%" and the variables are expressed in terms of the parameters. If the system has no solutions then the empty list is returned.

When you solve the linear system

\[
\begin{align*}
    x + 2y + 3z &= 2 \\
    2x + 3y + 4z &= 2 \\
    3x + 4y + 5z &= 2 
\end{align*}
\]

with this expression you get a solution involving a parameter.
The system can also be presented as a matrix and a vector. The matrix contains the coefficients of the linear equations and the vector contains the numbers appearing on the right-hand sides of the equations. You may input the matrix as a list of rows and the vector as a list of its elements.

To solve the system:
<pre>
  x + y + z = 8
  2*x - 2*y + z = 0
  x + 2*y + 2*z = 17
</pre>
in matrix form you would evaluate this expression.

The solutions are presented as a Record with two components: the component particular contains a particular solution of the given system or the item "failed" if there are no solutions, the component basis contains a list of vectors that are a basis for the space of solutions of the corresponding homogeneous system. If the system of linear equations does not have a unique solution, then the basis component contains non-trivial vectors.

This happens when you solve the linear system
<pre>
  x + 2*y + 3*z = 2
  2*x + 3*y + 4*z = 2
  3*x + 4*y + 5*z = 2
</pre>
with this command.

All solutions of this system are obtained by adding the particular solution with a linear combination of the basis vectors.

When no solution exists then "failed" is returned as the particular component, as follows:
When you want to solve a system of homogeneous equations (that is, a system where the numbers on the right-hand sides of the equations are all zero) in the matrix form you can omit the second argument and use the \texttt{nullSpace} operation.

This computes the solutions of the following system of equations:

\begin{verbatim}
  x + 2*y + 3*z = 0
  2*x + 3*y + 4*z = 0
  3*x + 4*y + 5*z = 0
\end{verbatim}

The result is given as a list of vectors and these vectors form a basis for the solution space.
Algebraic functions are functions defined by algebraic equations. There are two ways of constructing them, either by using rational powers or implicitly. For rational powers, use \[ f = \sqrt{1+x^{1/3}} \] or the system functions \[ \text{sqrt} \] and \[ \text{nthRoot} \] for square and nth roots.

To define an algebraic function implicitly use \[ \text{rootOf} \]. The following line defines a function \( y \) of \( x \) satisfying the equation

\[ y^3 = x^2 - x^3 + 1 \]
You can manipulate, differentiate or integrate an implicitly defined algebraic function like any other Axiom function.

Higher powers of algebraic functions are automatically reduced during calculations.

But denominators are not automatically rationalized.

Use <a href="dbopratenom.xhtml">ratDenom</a> to remove the algebraic quantities from the denominator.
Axiom has most of the usual functions from calculus built-in. You can substitute values or another elementary function for variables with the function eval. As you can see, the substitutions are made "in parallel" as in the case of polynomials. It’s also possible to substitute expressions for kernels instead of variables.
Given any ring R, the ring of the linear operators over R is called \(<a href="db.xhtml?Operator">Operator</a>(R)\).
To create an operator over R, first create a basic operator using the operation \(<a href="dbopoperator.xhtml">operator</a>\), and then convert it to \(<a href="db.xhtml?Operator">Operator(R)</a>\) for the R you want. We choose R to be the two by two matrices over the integers.

Create the operator tilde on R

To attach an evaluation function (from R to R) to an operator over R, use \(\text{evaluate}(op,f)\) where \(op\) is an operator over R and \(f\) is a function \(R \rightarrow R\). This needs to be done only once when the operator is defined. Note that \(f\) must be \(<a href="db.xhtml?Integer">Integer</a>\) linear (that is, \(f(ax+y) = af(x) + f(y)\) for any integer \(a\) and any \(x\) and \(y\) in R). We now attach the transpose map to the above operator \(t\).
Operators can be manipulated formally as in any ring: \( + \) is the pointwise addition and \( * \) is composition. Any element \( x \) of \( R \) can be converted to an operator \( op_x \) over \( R \), and the evaluation function of \( op_x \) is left-multiplication by \( x \). Multiplying on the left by this matrix swaps the two rows.

Can you guess what is the action of the following operator?

Hint: applying \( \rho \) four times gives the identity, so \( \rho^4 - 1 \) should return 0 when applied to any two by two matrix.

Now check with this matrix
As you have probably guessed by now, rho acts on matrices by rotating the elements clockwise.

Do the swapping of rows and transposition commute? We can check by computing their bracket.

Next we demonstrate how to define a differential operator on a polynomial ring. This is the recursive definition of the nth Legendre polynomial.
Now attach a map to it.

This is the differential equation satisfied by the nth Legendre polynomial.

Now we verify this for \( n=15 \). Here is the polynomial.

Here is the operator.

Here is the evaluation.
In Axiom, a function is an expression in one or more variables. (Think of it as a function of those variables.) You can also define a function by rules or use a built-in function. Axiom lets you convert expressions to compiled functions.

### Table of Functions

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rational Functions</td>
<td>Quotients of polynomials</td>
</tr>
<tr>
<td>Algebraic Functions</td>
<td>Those defined by polynomial</td>
</tr>
<tr>
<td>Elementary Functions</td>
<td>The elementary functions of calculus</td>
</tr>
<tr>
<td>Simplification</td>
<td>How to simplify expressions</td>
</tr>
</tbody>
</table>

</ul>
A common mathematical formula is
\[
\log(x)+\log(y)==\log(x*y)
\]
for any x and y. The presence of the word "any" indicates that x and y can stand for arbitrary mathematical expressions in the above formula. You can use such mathematical formulas in Axiom to specify "rewrite rules". Rewrite rules are objects in Axiom that can be assigned to variables for later use, often for the purpose of simplification. Rewrite rules look like ordinary function definitions except that they are preceded by the reserved word rule. For example, a rewrite rule for the above formula is:
\[
\text{rule } \log(x) + \log(y) == \log(x*y)
\]
Like function definitions, no action is taken when a rewrite rule is issued. Think of rewrite rules as functions that take one argument. When a rewrite rule A=B is applied to an argument f, its meaning is "rewrite every subexpressions of f that matches A by B". The left-and side of a rewrite rule is called a pattern.
its right-hand side is
called its substitution.

Create a rewrite rule named logrule. The generated symbol begins with a
"%" and is a place holder for any other terms that might occur in the sum.

Create an expression with logarithms.

Apply logrule to f.

The meaning of our example rewrite rule is "for all expressions x and y,
rewrite log(x) and log(y) by log(x*y)". Patterns generally have both operation
names
(here, log and +) and variables (here, x and y). By default, every operation name stands for
itself. The log matches only "log" and not
any other operation such as sin. On the other
hand, variables do not stand for themselves. Rather, a variable denotes a
pattern variable
that is free to match any expression whatsoever.

When a rewrite rule is applied, a process called
pattern matching goes to work by systematically
scanning the subexpressions of the argument. When a subexpression is found
that matches the pattern, the subexpression is replaced by the right hand
side of the rule. The details of what happens will be covered later.

The customary Axiom notation for patterns is actually a shorthand for a
longer, more general notation. Pattern variables can be made explicit
by using a percent ("%") as the first character of the variable name. To
say that a name stands for itself, you can prefix that name with a quote
operator ("'"). Although the current Axiom parser does not let you quote
an operation name, this more general notation gives you an alternative way of giving the same rewrite rule:

```pre
rule log(%x) + log(%y) == log(x*y)
</pre>
```

This longer notation gives you patterns that the standard notation won’t handle. For example, the rule

```pre
rule %f(c * 'x) == c*%f(x)
</pre>
```

means "for all f and c, replace f(y) by c*f(x) when y is the product of c and the explicit variable x".

Thus the pattern can have several adornments on the names that appear there. Normally, all of these adornments are dropped in the substitution on the right hand side. To summarize:

```hr/>
To enter a single rule in Axiom, use the following syntax:
<pre>
rule lefthandside == righthandside
</pre>
```

The lefthandside is a pattern to be matched and the righthandside is its substitution. The rule is an object of type

```a href="db.xhtml?RewriteRule">RewriteRule</a> that can be assigned to a variable and applied to expressions to transform them.
</hr/>
```

Rewrite rules can be collected into rulesets so that a set of rules can be applied at once. Here is another simplification rule for logarithms.

```pre
rule y*log(x) == log(x**y)
</pre>
```

for any x and y. If instead of giving a single rule following the reserved word rule you give a "pile" of rules, you create what is called a ruleset. Like rules, rulesets are objects in Axiom and can be assigned to variables. You will find it useful to group commonly used rules into input files, and read them in as needed. Create a ruleset named logrules.

```ul>
<li>
<input type="submit" id="p4" class="subbut"
onclick="makeRequest('p4');"
value="logrules:=rule (log(x)+log(y)==log(x*y) ; y*log(x)==log(x^y))" />
</li>
</ul>
```

Again, create an expression f containing logarithms.

```ul>
<li>
<input type="submit" id="p5" class="subbut"
onclick="makeRequest('p5');"
value="f:=a*log(sin x)-2*log x" />
</li>
</ul>
```

Apply the ruleset logrules to f.
We have allowed pattern variables to match arbitrary expressions in the above examples. Often you want a variable to match only expressions satisfying some predicate. For example, you may want to apply the transformation

\[
y \log(x) = \log(x^y)
\]

only when \( y \) is an integer. The way to restrict a pattern variable \( y \) by a predicate \( f(y) \) is by using a vertical bar "|", which means "such that", in much the same way it is used in function definitions. You do this only once but at the earliest (meaning deepest and leftmost) part of the pattern. This restricts the logarithmic rule to create integer exponents only.

Compare this with the result of applying the previous set of rules.

You should be aware that you might need to apply a function like \( \text{integer} \) within your predicate expression to actually apply the test function. Here we use \( \text{integer} \) because \( n \) has type \( \text{Expression Integer} \) but \( \text{even?} \) is an operation defined on the integers.
Here is the application of the rule.

This is an example of some of the usual identities involving products of sines and cosines.

Another qualification you will often want to use is to allow a pattern to match an identity element. Using the pattern x+y, for example, neither x nor y matches the expression 0. Similarly, if a pattern contains a product x*y or an exponentiation x^y, then neither x nor y matches 1. If identical elements were matched, pattern matching would generally loop. Here is an expansion rule for exponentials.

This rule would cause infinite rewriting on this if either a or b were allowed to match 0.
There are occasions when you do want a pattern variable in a sum or product to match 0 or 1. If so, prefix its name with a "?" whenever it appears in a left-hand side of a rule. For example, consider the following rule for the exponential integral

\[
\text{integral}\left(\frac{y+\exp x}{x},x\right) = \text{integral}\left(\frac{y}{x},x\right) + \text{Ei} x
\]

for any \(x\) and \(y\). This rule is valid if \(y=0\). One solution is to create a Ruleset with two rules, one with and one without \(y\). A better solution is to use an "optional" pattern variable. Define rule eirule with a pattern variable \(?y\) to indicate that an expression may or may not occur.

Apply rule eirule to an integral without this term.

Apply rule eirule to an integral with this term.

Here is one final adornment you will find useful. When matching a pattern of the form \(x+y\) to an expression containing a long sum of the form \(a+\ldots+b\), there is no way to predict in advance which subset of the sum matches \(x\) and which matches \(y\). Aside from efficiency, this is generally unimportant since the rule holds for any possible combination of matches for \(x\) and \(y\). In some situations, however, you many want to say which pattern variable is a sum (or product) of several terms, and which should match only a single term. To do this, put a prefix colon ("::") before the pattern variable that you want to match multiple terms. The remaining rules involve operators \(u\) and \(v\).
These definitions tell Axiom that \( u \) and \( v \) are formal operators to be used in expressions.

First define \( \text{myRule} \) with no restrictions on the pattern variables \( x \) and \( y \).

Apply \( \text{myRule} \) to an expression.

Define \( \text{myOtherRule} \) to match several terms so that the rule gets applied recursively.

Apply \( \text{myOtherRule} \) to the same expression.

Here are some final remarks on pattern matching. Pattern matching provides a very useful paradigm for solving certain classes of problems, namely,
those that involve transformations of one form to another and back. However, it is important to recognize its limitations.

First, pattern matching slows down as the number of rules you have to apply increases. Thus it is good practice to organize the sets of rules you use optimally so that irrelevant rules are never included.

Second, careless use of pattern matching can lead to wrong answers. You should avoid pattern matching to handle hidden algebraic relationships that can go undetected by other programs. As a simple example, a symbol such as "J" can easily be used to represent the square root of -1 or some other important algebraic quantity. Many algorithms branch on whether an expression is zero or not, then divide by that expression if it is not. If you fail to simplify an expression involving powers of J to -1, algorithms may incorrectly assume an expression is no-zero, take a wrong branch, and produce a meaningless result.

Pattern matching should also not be used as a substitute for a domain. In Axiom, objects of one domain are transformed to objects of other domains using well-defined operations. Pattern matching should be used on objects that are all of the same type. Thus if your application can be handled by type in Axiom and you think you need pattern matching consider this choice carefully. You may well be better served by extending an existing domain or by building a new domain of objects for your application.
Use the functions <a href="dbopnumer.xhtml">numer</a> and <a href="dbopdenom.xhtml">denom</a> to recover the numerator and denominator of a fraction:

Since these are polynomials, you can apply all of the <a href="polynomialpage.xhtml">polynomial operations</a> to them. You can substitute values or other rational functions for the variables using the function <a href="dbopeval.xhtml">eval</a>. The syntax for <a href="dbopeval.xhtml">eval</a> is similar to the one for polynomials:
Simplifying an expression often means different things at different times. Axiom offers a large number of "simplification" functions. The most common one, which performs the usual trigonometric simplifications is \(<a href="dbopsimplify.xhtml">simplify</a>\).

If the result of \(<a href="dbopsimplify.xhtml">simplify</a>\) is not satisfactory, specific transformations are available. For example, to rewrite g in terms of secants and cosecants instead of sines and cosines, issues:

To apply the logarithm simplification rules to h, issue:

Since the square root of x^2 is the absolute value of x and not x itself, algebraic radicals are not automatically simplified, but you can specifically request it by calling \(<a href="dboprootsimp.xhtml">rootSimp</a>\):
There are other transformations which are sometimes useful. Use the functions 
<a href="dbopcomplexelementary.xhtml">complexElementary</a> and 
<a href="dboptrigs.xhtml">trigs</a> to go back and forth between
the complex exponential and trigonometric forms of an elementary function.

Similarly, the functions 
<a href="dboprealelementary.xhtml">realElementary</a> and 
<a href="dbophtrigs.xhtml">htrigs</a> convert hyperbolic functions in
and out of their exponential form.
Axiom has other transformations, most of which are in the packages
ElementaryFunctionStructurePackage, TrigonometricManipulations, AlgebraicManipulations, and TranscendentalManipulations. If you need to apply a simplification rule not built into the system you can use Axiom's pattern matcher.

— glossarypage.xhtml —
Suffix character for pattern variables.

The special type \( \text{?} \) means don't care. For example, the declaration

\[ x : \text{Polynomial} \text{ ?} \]

must be polynomials over an arbitrary underlying domain.

**abstract datatype**

A programming language principle used in Axiom where a datatype is defined in two parts: (1) a public part describing a set of exports, principally operations that apply to objects of that type, and (2) a private part describing the implementation of the datatype usually in terms of a representation for objects of the type. Programs which create and otherwise manipulate objects of the type may only do so through its exports. The representation and other implementation information is specifically hidden.

**abstraction**

Described functionally or conceptually without regard to implementation.

**accuracy**

The degree of exactness of an approximation or measurement. In computer algebra systems, computations are typically carried out with complete accuracy using integers or rational numbers of indefinite size. Domain \( \text{Float} \) provides a function \( \text{precision} \) from \( \text{Float} \) to change the precision for floating point computations. Computations using \( \text{DoubleFloat} \) have a fixed precision but uncertain accuracy.

**add-chain**

A hierarchy formed by domain extensions. If domain \( A \) extends domain \( B \) and domain \( B \) extends domain \( C \), then \( A \) has \( \text{add-chain} \) \( B \) and \( C \).

**aggregate**

A data structure designed to hold multiple values. Examples of aggregates are \( \text{List} \), \( \text{Set} \), \( \text{Matrix} \) and \( \text{Bits} \).

**AKCL**

Austin Kyoto Common LISP, a version of KCL produced by William Schelter, Austin, Texas.
a step-by-step procedure for a solution of a problem; a program

\begin{itemize}
\item ancestor (of a domain) a category which is a parent of the domain, or a parent of a parent and so on.
\end{itemize}

\begin{itemize}
\item application (syntax) an expression denoting "application" of a function to a set of argument parameters. Applications are written as parameterized form. For example, the form \( f(x, y) \) indicates the "application of the function \( f \) to the tuple of arguments \( x \) and \( y \). See also evaluation and invocation.
\end{itemize}

\begin{itemize}
\item apply
\end{itemize}

\begin{itemize}
\item argument
\begin{itemize}
\item (actual argument) a value passed to a function at the time of a call application; also called an actual parameter.
\item (formal argument) a variable used in the definition of a function to denote the actual argument passed when the function is called.
\end{itemize}
\end{itemize}

\begin{itemize}
\item arity
\begin{itemize}
\item (function) the number of arguments.
\item (operator or operation) corresponds to the arity of a function implementing the operator or operation.
\end{itemize}
\end{itemize}

\begin{itemize}
\item assignment (syntax) an expression of the form \( x := e \), meaning "assign the value of \( e \) to \( x \). After evaluation, the variable \( x \) pointer to an object obtained by evaluating the expression \( e \). If \( x \) has a type as a result of a previous declaration, the object assigned to \( x \) must have that type. An interpreter must often
coercion the value of e to make that happen. For example, in the interpreter,
first declaration to be a float. This declaration causes the interpreter to coerce 11 to 11.0 in order to assign a floating point value to \( x \).

attribute a name or functional form denoting any useful computational property. For example,
asserts that "*" is commutative. Also, \( \text{finiteAggregate} \) is used to assert that an aggregate has a finite number of immediate components.

basis \( S \) is a basis of a module \( M \) over a ring if \( S \) generates \( S \) is linearly independent

benefactor (of a given domain) a domain or package that the given domain explicitly references (for example, calls functions from) in its implementation

binary operation or function with arity 2

binding the association of a variable with properties such as value and type. The top-level environment in the interpreter consists of bindings for all user variables and functions. Every function has an associated set of bindings, one for each formal argument and local variable.

block a control structure where expressions are sequentially evaluated.

body a function body or loop body.

boolean objects denoted by the literals \( \text{true} \) and \( \text{false} \); elements of domain \( \text{Boolean} \). See also \( \text{Bits} \).

built-in function in the standard Axiom library. Contrast \( \text{user function} \).
(noun) a mechanism for immediate retrieval of previously computed data. For example, a function which does a lengthy computation might store its values in a hash table using argument as a key. The hash table then serves as a cache for the function (see also set function cache). Also, when previous values are compiled, the previous values are normally cached (use set functions recurrence to change this).

(verb) to save values in a cache.

capsule
the part of the function body of a domain constructor that defines the functions implemented by the constructor.

case
an operator used to conditionally evaluate code based on the branch of a Union. For example, if value is a Union(Integer, "failed") the conditional expression evaluate A if u case Integer then A else B otherwise.

category
second-order types which serve to define useful "classification worlds" for domains, such as algebraic constructs (e.g. groups, rings, fields), and data structures (e.g. homogeneous aggregates, collections, dictionaries). Examples of categories are Ring ("the class of all rings") and Aggregate ("the class of all aggregates"). The categories of a given world are arranged in a hierarchy (formally, a directed acyclic graph). Each category inherits the properties of all its ancestors. Thus, for example, the category of ordered rings inherits the properties of the category of rings and those of the ordered sets. Categories provide a database of algebraic knowledge and ensure mathematical correctness, e.g. that "matrices of polynomials" is correct but "polynomials of hash tables"
is not, that the multiply operation for "polynomials of continued fractions" is commutative, but that for "matrices of power series" is not. Optionally provide "default definitions" for operations they export. Categories are defined in Axiom by functions called category constructors. Technically, a category designates a class of domains with common operations and attributes but usually with different functions and representations for its constituent objects. Categories are always defined using the Axiom library language (see also category extension). See also file catdef.spad for definitions of basic algebraic categories in Axiom.

A function that creates categories, described by an abstract datatype in the Axiom programming language. For example, the category constructor Module is a function which takes a domain parameter $R$ and creates the category "modules over $R$".

A category definition, an expression usually of the form $A \equiv B$ with ... . In English, this means "category A is a B with the new operations and attributes as given by ... . See, for example, file catdef.spad for a definitions of the algebra categories in Axiom, aggcat.spad for data structure categories.

Hierarchy formed by category extensions. The root category is Object. A category can be defined as a Join of two or more categories so as to have multiple parents. Categories may also have parameterized so as to allow conditional inheritance.

An element of a character set, as represented by a keyboard key.

A component of a string. For example, the 0th element of the string "hello there" is the character 'h'.

(of a given domain) any domain or package that explicitly calls functions from the given domain.
coercion

an automatic transformation of an object of one type to an object of a similar or desired target type. In the interpreter, coercions and retractions are done automatically by the interpreter when a type mismatch occurs. Compare conversion.

comment
textual remarks imbedded in code. Comments are preceded by a double dash (---). For Axiom library code, stylized comments for on-line documentation are preceded by a two plus signs (++).

Common LISP
A version of LISP adopted as an informal standard by major users and suppliers of LISP.

compile-time
the time when category or domain constructors are compiled. Contrast run-time.

compiler
a program that generates low-level code from a higher-level source language. Axiom has three compilers.

A graphics compiler converts graphical formulas to a compiled subroutine so that points can be rapidly produced for graphics commands.

An interpreter compiler optionally compiles user functions when first invoked (use set functions compile to turn this feature on).

A library compiler compiles all constructors.

computational object
In Axiom, domains are objects. This term is used to distinguish the objects which are members of domains rather than domains themselves.

conditional
a control structure of the form if A then B else C; The evaluation of A produces A true or false. If true, B evaluates to
produce a value; otherwise \( C \) evaluates to produce a value. When the value is not used, \( C \) part can be omitted.

- **constant** (syntax) a reserved word used in signatures in Axiom programming language to signify that an operation always returns the same value. For example, the signature \( 0 : \text{constant} \to \$$ \) in the source code of \(~\text{type}\) \( \text{AbelianMonoid} \) tells the Axiom compiler that \( 0 \) is a constant so that suitable optimizations might be performed.

- **constructor** a function which creates a category, domain, or package.

- **continuation** when a line of a program is so long that it must be broken into several lines, then all but the first line are called continuation lines. If such a line is given interactively, then each incomplete line must end with an underscore.

- **control structure** program structures which can specify a departure from normal sequential execution. Axiom has four kinds of control structures: blocks, case statements, conditionals, and loops.

- **conversion** the transformation of an object on one type to one of another type. Conversions performed automatically are called coercions. These happen when the interpreter has a type mismatch and a similar or declared target type is needed. In general, the user must use the infix operation \( :: \) to cause this transformation.

- **copying semantics** the programming language semantics used in Pascal but not in Axiom. See also pointer semantics for details.

- **data structure** a structure for storing data in the computer. Examples are lists and hash tables.

- **datatype** equivalent to domain in Axiom.

- **declaration** (syntax) an expression of the form \(~\text{type}\) where \(~\text{type}\) is some type. A declaration forces all values
assigned to \( T \) to be of that type. If a value is of a different type, the interpreter will try to coerce the value to type \( T \). Declarations are necessary in case of ambiguity or when a user wants to introduce an an

default definition

a function defined by a category. Such definitions appear category definitions of the form

\[ C: \text{Category} \Rightarrow T \text{ add I} \]

in an optional implementation part \( \text{add} \) to the right of the keyword \( \text{add} \). Declarations are necessary in case of ambiguity or when a user wants to introduce an an

default package

a optional package of functions associated with a category. Such functions are necessarily defined in terms over other functions exported by the category.

definition

An expression of the form

\[ f(a) \Rightarrow b \]

defining function

\[ f \]

with formal arguments \( a \) and body \( b \); equivalent to the statement

\[ \text{f := (a) \Rightarrow b} \]

definition

An expression of the form

\[ a \Rightarrow b \]

where \( a \) is a symbol, equivalent to

\[ a() \Rightarrow b \]

See also macro where a similar substitution is done at parse time.

delimiter

a character which marks the beginning or end of some syntactically correct unit in the language, e.g. " for strings, blanks for identifiers.

destructive operation

An operation which changes a component or structure of a value. In Axiom, all destructive operations have names which end with an exclamation mark (e.g., \( \text{reverse} \)). For example, domain \( \text{List} \) has two operations to reverse the elements of a list, one named \( \text{reverse} \) which returns a copy of the original list with the elements reversed, another named
which reverses the elements in place, thus destructively changing the original list.

Documentation

- On-line or hard copy descriptions of Axiom;
- Text in library code preceded by `++` comments as opposed to general comments preceded by `--`.

Domain

A domain corresponds to the usual notion of abstract datatypes: that of a set of values and a set of "exported operations" for the creation and manipulation of these values. Datatypes are parameterized, dynamically constructed, and can combine with others in any meaningful way, e.g. "lists of floats" (\texttt{List Float}), "fractions of polynomials with integer coefficients" (\texttt{Fraction Polynomial Integer}), "matrices of infinite streams of cardinal numbers" (\texttt{Matrix Stream CardinalNumber}). Technically, a domain denotes a class of objects, a class of operations for creating and other manipulating these objects, and a class of attributes describing computationally useful properties. Domains also provide functions for each operation often in terms of some representation for the objects. A domain itself is an object created by a function called a domain constructor.

Domain constructor

A function that creates domains, described by an abstract datatype in the Axiom programming language. Simple domains like \texttt{Integer} and \texttt{Boolean} are created by domain constructors with no arguments. Most domain constructors take one or more parameters, one usually denoting an underlying domain. For example, the domain \texttt{Matrix(R)} denotes "matrices over \texttt{R}". Domains \texttt{Mapping}, \texttt{Record}, and \texttt{Union} are primitive domains. All other domains are written in the Axiom programming language and can be modified by users with access to the library source code.

Domain extension
a domain constructor $A$ is said to extend $B$ if $A \equiv B \text{ add } \ldots$. This intuitively means "functions not defined by $A$ are assumed to come from $B$". Successive domain extensions form add-chains affecting the search order for functions not implemented directly by the domain during dynamic lookup.

### Dot notation

Using an infix dot (`.`) for function application. If $u = [7, 4, -11]$ then both $u(2)$ and $u.2$ return 4. Dot notation nests to left. Thus $(f . g . h)$ is equivalent to $(f . (g . h))$.

### Dynamic vs. Compile-time

That which is done at run-time as opposed to compile-time. For example, the interpreter will build the domain "matrices over integers" dynamically in response to user input. However, the compilation of all functions for matrices and integers is done during compile-time. Contrast static.

### Dynamic Lookup

In Axiom, a domain may or may not explicitly provide function definitions for all of its exported operations. These definitions may instead come from domains in the add-chain or from default packages. When a function call is made for an operation in the domain, up to five steps are carried out.

1. If the domain itself implements a function for the operation, that function is returned.
2. Each of the domains in the add-chain are searched for one which implements the function; if found, the function is returned.
3. Each of the default packages for the domain are searched in order of the lineage. If any of the default packages implements the function, the first one found is returned.
4. Each of the default packages for each of the domains in the add-chain are searched in the
order of their <a href="#p30933">lineage</a>. If any of the default packages implements the function, the first one found is returned.
</li>
<li> If all of the above steps fail, an error message is reported.
</li>
</ol>
</li>
</li>
<li><a name="p19071" class="glabel"/>empty</b>
the unique value of objects with type <div class="gtype">Void</div>.</li>
</li>
<li><a name="p19131" class="glabel"/>environment</b>
a set of <a href="#p4735">bindings</a>.</li>
</li>
<li><a name="p19167" class="glabel"/>evaluation</b>
a systematic process which transforms an <a href="#p20659">expression</a> into an object called the <a href="#p52710">value</a> of the expression. Evaluation may produce <a href="#p46699">side effects</a>.</li>
</li>
<li><a name="p19348" class="glabel"/>exit</b><div class="gsyntax">(reserved word)</div> an <a href="#p36278">operator</a> which forces an exit from the current block. For example, the <a href="#p5086">block</a><div class="gspad">(a := 1; if i > 0 then exit a; a := 2)</div> will prematurely exit at the second statement with value 1 if the value of <a href="#p52710">i</a> is greater than 0. See <a href="#p210"><div class="gspad">==></div></a> for an alternate syntax.
</li>
<li><a name="p19681" class="glabel"/>explicit export</b><ol>
<li>(of a domain <div class="gspad">D</div>) any <a href="#p4093">attribute</a>, <a href="#p36041">operation</a>, or <a href="#p6628">category</a> explicitly mentioned in the <a href="#p50664">type</a> specification part <div class="gspad">T</div> for the domain constructor definition <div class="gspad">D: T == I</div></li>
<li>(of a category <div class="gspad">C</div>) any <a href="#p4093">attribute</a>, <a href="#p36041">operation</a>, or <a href="#p6628">category</a> explicitly mentioned in the <a href="#p50664">type</a> specification part <div class="gspad">T</div> for the domain constructor definition <div class="gspad">C: <a href="#p6537">Category</a> == T</div></li>
</ol></li>
</li>
<li><a name="p20171" class="glabel"/>export</b>
<a href="#p19681">explicit export</a> or <a href="#p27325">implicit export</a> of a domain or category</li>
</li>
<li><a name="p20259" class="glabel"/>expose</b> some constructors are <div class="gsyntax">exposed</div>, others <div class="gsyntax">unexposed</div>. Exposed domains and packages
are recognized by the interpreter. Use 
<div class="gcmd">)set expose</div>
to control change what is exposed. To see both exposed 
and unexposed constructors, use the browser with give the system 
command <div class="gcmd">)set hyperdoc browse exposure 
on</div>. Unexposed constructors will now appear prefixed by star 
(\langle div class="gspad">*</div>).

\li\<a name="p20659" class="glabel"/>expression\</li> 
  \li \li\ any syntactically correct program fragment. 
  \li\li\ an element of domain \langle div class="gtype">Expression</div> 
  \li\li\ 
\li\li\ <a name="p20757" class="glabel"/>extend\</li> 
  see \langle a href="#p8634">category extension</div>\rangle or \langle a href="#p16819">domain 
  extension</div>\rangle
\li\li\ <a name="p20829" class="glabel"/>field\</li> 
  \langle div class="gsyntax">(algebra)</div>\langle a href="#p17507">domain</div>\rangle 
  which is \langle a href="#p45405">ring</div>\rangle where every non-zero element is 
  invertible and where \langle div class="gspad">xy=yx</div>; a member of 
  category \langle div class="gtype">Field</div>\rangle. For a complete list of 
  fields, click on \langle div class="gsyntax">Domains</div> under 
  \langle div class="gsyntax">Cross Reference</div> for 
  \langle div class="gtype">Field</div>\rangle.
\li\li\ <a name="p21109" class="glabel"/>file\</li> 
  a program or collection of data stored on disk, tape or other medium. 
\li\li\ <a name="p21186" class="glabel"/>float\</li> 
  a floating-point number with user-specified precision; an element of 
  domain \langle div class="gtype">Float</div>\rangle. Floats are 
  \langle a href="#p31774">literals</div> which are written two ways: without an 
  exponent (e.g. \langle div class="gspad">3.1416</div>), or with an exponent 
  (e.g. \langle div class="gspad">3.12E-12</div>). Use function 
  \langle a href="#p42318">precision</div> to change the precision of the mantissage 
  (20 digits by default). See also \langle a href="#p47066">small float</div>\rangle.
\li\li\ <a name="p21594" class="glabel"/>formal parameter\</li> 
  (of a function) an identifier \langle a href="#p4735">bound</div>\rangle to the value 
  of an actual \langle a href="#p2885">argument</div>\rangle on 
  \langle a href="#p29675">invocation</div>\rangle. In the function definition 
  \langle div class="gspad">f(x, y) \Rightarrow u</div>, for example, 
  \langle div class="gspad">x</div> and \langle div class="gspad">y</div> are the formal 
  parameter.
\li\li\ <a name="p21847" class="glabel"/>frame\</li> 
  the basic unit of an interactive session; each frame has its own 
  \langle a href="#p47691">step number</div>, \langle a href="#p21021">environment</div>, and 
  \langle a href="#p26034">history</div>. In one interactive session, users can 
  can create and drop frames, and have several active frames simultaneously.
A keyword used in user-defined functions to declare that a variable is a free variable of that function. For example, the statement \( \text{free } x \) declares the variable \( x \) within the body of a function \( f \) to be a free variable in \( f \). Without such a declaration, any variable \( x \) which appears on the left hand side of an assignment is regarded as a local variable of that function. If the intention of the assignment is to give an value to a global variable \( x \), the body of that function must contain the statement \( \text{free } x \).

A variable which appears in a body of a function but is not bound by that function. See local variable by default.

The part of a function's definition which is evaluated when the function is called at run-time; the part of the function definition to the right of the \( == \). See also generic function.

An expression denoting "application" of a function to a set of parameters. Applications are written as a
parameterized form. For example, the form \( f(x, y) \) indicates the "application of the function \( f \) to the tuple of arguments \( x \) and \( y \). See also <a href="#p19167">evaluation</a> and <a href="#p29675">invocation</a>.

- **garbage collection**
  a system function that automatically recycles memory cells from the heap. Axiom is built upon <a href="#p25771">Common LISP</a> which provides this facility.

- **garbage collector**
  a mechanism for reclaiming storage in the heap.

- **Gaussian**
  a complex-valued expression, e.g. one with both a real and imaginary part; a member of a <div class="gtype">Complex</div> domain.

- **generic function**
  the use of one function to operate on objects of different types; One might regard Axiom as supporting generic <a href="#p36041">operations</a> but not generic functions. One operation <div class="gspad">+: (D, D) -> D</div> exists for adding elements in a ring; each ring however provides its own type-specific function for implementing this operation.

- **global variable**
  A variable which can be referenced freely by functions. In Axiom, all top-level user-defined variables defined during an interactive user session are global variables. Axiom does not allow <div class="gsyntax">fluid variables</div>, that is, variables <a href="#p4735">bound</a> by functions which can be referenced by functions those functions call.

- **Groebner basis**
  a special basis for a polynomial ideal that allows a simple test for membership. It is useful in solving systems of polynomial equations.

- **group**
  a <div class="gtype">monoid</div> where every element has a multiplicative inverse.

- **hash table**
  a data structure that efficiently maps a given object to another. A hash table consists of a set of <div class="gsyntax">entries</div>, each of which associates a <div class="gsyntax">key</div> with a <div class="gsyntax">value</div>. Finding the object stored under a key can be very fast even if there are a large number of entries since keys are <div class="gsyntax">hashed</div> into numerical codes for fast lookup.

- **heap**
  an area of storage used by data in programs. For example, AXIOM will
use the heap to hold the partial results of symbolic computations. When cancellations occur, these results remain in the heap until garbage collected.

A mechanism which records the results for an interactive computation. Using the history facility, users can save computations, review previous steps of a computation, and restore a previous interactive session at some later time. For details, issue the system command history to the interpreter. See also frame.

An Axiom name; a literal of type Symbol. An identifier begins with an alphabetical character or % and may be followed by alphabetic characters, digits, ? or !. Certain distinguished reserved words are not allowed as identifiers but have special meaning in the Axiom.

An object is immutable if it cannot be changed by an operation; not a mutable object. Algebraic objects generally immutable: changing an algebraic expression involves copying parts of the original object. One exception is a matrix object of type Matrix. Examples of mutable objects are data structures such as those of type List. See also pointer semantics.

(of a domain or category) any attribute or operation which is either an explicit export or else an explicit export of some category which an explicit category export extends.

A variable that counts the number of times a loop is repeated.

The "address" of an element in a data structure (see also category LinearAggregate).
an operator placed between two operands; also called a binary operator, e.g. \( a + b \). An infix operator may also be used as a prefix, e.g. \( +(a, b) \) is also permissible in the Axiom language. Infix operators have a relative precedence.

input area

a rectangular area on a screen into which users can enter text.

instantiate

to build a domain, category, or package at run-time.

integer

a literal object of domain \( \text{Integer} \), the class of integers with an unbounded number of digits. Integer literals consist of one or more consecutive digits (0-9) with no embedded blanks. Underscores can be used to separate digits in long integers if desirable.

interactive

a system where the user interacts with the computer step-by-step

interpreter

the subsystem of Axiom responsible for handling user input during an interactive session. The following somewhat simplified description of the typical action of the interpreter. The interpreter parses the user input expression to create an expression tree then does a bottom-up traversal of the tree. Each subtree encountered which is not a value consists of a root node denoting an operation name and one or more leaf nodes denoting operands. The interpreter resolves type mismatches and uses type-inferencing and a library database to determine appropriate types of the operands and the result, and an operation to be performed. The interpreter then builds a domain to perform the indicated operation, then invokes a function from the domain to compute a value. The subtree is then replaced by that value and the process continues. Once the entire tree has been processed, the value replacing the top node of the tree is displayed back to the user as the value of the expression.

invocation

(of a function) the run-time process involved in evaluating a function. This process has two steps. First, a local environment is created where formal arguments are locally bound to their respective actual arguments. Second, the function body is evaluated in that local environment. The evaluation of a function is terminated either by
completely evaluating the function body or by the evaluation of a
expression.
</li>
<li><a name="p30286" class="glabel"/>iteration
repeated evaluation of an expression or a sequence of
expressions. Iterations use the reserved words
for, while, and repeat.
</li>
<li><a name="p30459" class="glabel"/>Join
a primitive Axiom function taking two or more categories as arguments
and producing a category containing all of the operations and
attributes from the respective categories.
</li>
<li><a name="p30645" class="glabel"/>KCL
Kyoto Common LISP, a version of Common LISP which features compilation of the compilation of LISP into the C Programming Language
</li>
<li><a name="p30801" class="glabel"/>library
In Axiom , a collection of compiled modules representing the a
category or domain constructor.
</li>
<li><a name="p30933" class="glabel"/>lineage
the sequence of default packages for a given domain to be searched during
dynamic lookup. This sequence is computed first by ordering the category
ancestors of the domain according to their level number, an integer equal to the
minimum distance of the domain from the category. Parents have level 1, parents of parents have level 2, and so on. Among categories with
equal level numbers, ones which appear in the left-most branches of the
source code come first. See also dynamic lookup.
</li>
<li><a name="p31518" class="glabel"/>LISP
acronym for List Processing Language, a language designed for the
manipulation of nonnumerical data. The Axiom library is translated into LISP then compiled into machine code by an underlying LISP.
</li>
<li><a name="p31730" class="glabel"/>list
an object of a List domain.
</li>
<li><a name="p31774" class="glabel"/>literal
an object with a special syntax in the language. In Axiom , there are five types of literals: booleans, integers, floats, strings, and symbols.
</li>
<li><a name="p31998" class="glabel"/>local
A keyword used in user-defined functions to declare that a variable is a
local variable of that function. Because of default assumptions on variables, such a declaration is not necessary but is available to the user for clarity when appropriate.

(a function) a variable bound by that function and such that its binding is invisible to any function that function calls. Also called a lexical variable. By default in the interpreter:

any variable which appears on the left hand side of an assignment is regarded a local variable of that function. If the intention of an assignment is to change the value of a global variable, the body of the function must then contain the statement.

any other variable is regarded as a free variable.

An optional declaration is available to explicitly declare a variable to be a local variable. All formal parameters to the function can be regarded as local variables to the function.

an expression containing a repeat

a collection expression having a for or a while, e.g. [f(i) for i in S].

the part of a loop following the repeat that tells what to do each iteration. For example, the body of the loop for x in S repeat B is. For a collection expression, the body of the loop precedes the initial for or while.

An expression of the form macro a == b where a is a
symbol causes \(a\) to be textually replaced by the expression \(b\) at parse time.

An expression of the form \(\text{macro } f(a) == b\) defines a parameterized macro expansion for a parameterized form \(f\). This macro causes a form \(f(x)\) to be textually replaced by the expression \(c\) at parse time, where \(c\) is the expression obtained by replacing \(a\) by \(x\) everywhere in \(b\). See also \(\text{definition}\) where a similar substitution is done during evaluation.

mode is a type expression containing a question-mark \(?\). For example, the mode \(P ?\) designates the class of all polynomials over an arbitrary ring.

monoid is a set with a single, associative operation and an identity element.

mutable objects which contain pointers to other objects and which have operations defined on them which alter these pointers. Contrast \(\text{immutable}\). Axiom uses \(\text{pointer}\) semantics as does \(\text{LISP}\) in contrast with many other languages such as Pascal which use \(\text{copying}\) semantics. See \(\text{pointer}\) semantics for details.

name is a symbol denoting a variable, i.e. the variable \(x\).

a \(\text{symbol}\) denoting an \(\text{operation}\), i.e. the operation \(\text{divide}: \text{Integer} \times \text{Integer} \to \text{Integer}\).

nullary a function with no arguments, e.g. \(\text{characteristic}\).

nullary
operation or function with \( \text{arity} = 0 \).

- **Object**: A category with no operations or attributes, from which most categories in Axiom are category extensions.

- **object**: A data entity created or manipulated by programs. Elements of domains, functions, and domains themselves are objects. Whereas categories are created by functions, they cannot be dynamically manipulated in the current system and are thus not considered as objects. The most basic objects are literals; all other objects must be created.

- **object code**: Code which can be directly executed by hardware; also known as machine language.

- **operand**: An argument of an operator (regarding an operator as a function).

- **operation**: An abstraction of a function, described by a signature. For example, 

  \[
  \text{fact: NonNegativeInteger -> NonNegativeInteger}
  \]

  Describes an operation for "the factorial of a (non-negative) integer".

- **operator**: Special reserved words in the language such as \(+\) and \(*\); operators can be either prefix or infix and have a relative precedence.

- **overloading**: The use of the same name to denote distinct functions; a function is identified by a signature identifying its name, the number and types of its arguments, and its return types. If two functions can have identical signatures, a package call must be made to distinguish the two.

- **package**: A domain whose exported operations depend solely on the parameters and other explicit domains, e.g. a package for solving systems of equations of polynomials over any field, e.g. floats, rational numbers, complex rational functions, or power series. Facilities for integration, differential equations, solution of linear or polynomial...
equations, and group theory are provided by "packages". Technically, a package is a domain which has no containing the symbol $. While domains intuitively provide computational objects you can compute with, packages intuitively provide functions (polymorphic functions) which will work over a variety of datatypes.

A package call is an expression of the form $e \text{ $D$}$ where $e$ is an application and $D$ denotes some package (or domain).

A parameterized datatype is a domain that is built on another, for example, polynomials with integer coefficients.

A parameterized form is an expression of the form $f(x, y)$, an application of a function.

A parent (of a domain) is a category which is explicitly declared in the source code definition for the domain to be an export of the domain.

A parse is the transformation of a user input string representing a valid Axiom expression into an internal representation as a tree-structure.

A partially ordered set is a set with a reflexive, transitive and antisymmetric operation.
The left hand side of a rewrite rule is called a pattern. Rewrite rules can be used to perform pattern matching, usually for simplification. The right hand side of a rule is called the substitution.

Pattern matching:

- (on expressions) Given a expression called a "subject" using a set of "rewrite rules". Each rule has the form $A = B$ where $A$ indicates a expression called a "pattern" and $B$ denotes a "replacement". The meaning of this rule is "replace $A$ by $B". If a given pattern $A$ matches a subexpression of $u$, that subexpression is replaced by $B$. Once rewritten, pattern matching continues until no further changes occur.

- (on strings) the attempt to match a string indicating a "pattern" to another string called a "subject", for example, for the purpose of identifying a list of names. In a browser, users may enter search strings for the purpose of identifying constructors, operations, and attributes.

Pattern variable:

In a rule a symbol which is not a recognized function acts as a pattern variable and is free to match any subexpression.

Pile:

Alternate syntax for a block, using indentation and column alignment (see also <a href="#p5086">block</a>).

Pointer:

A reference implemented by a link directed from one object to another in the computer memory. An object is said to refer to another if it has a pointer to that other object. Objects can also refer to themselves (cyclic references are legal). Also more than one object can refer to the same object. See also <a href="#p39949">pointer semantics</a>.

Pointer semantics:

The programming language semantics used in languages such as LISP which allow objects to be mutable. Consider the following sequence of Axiom statements:

```axiom
x : Vector Integer := [1, 4, 7]
y := x
```
The function \texttt{swap!} from \texttt{Vector} is used to interchange the 2nd and 3rd value in the list \texttt{x} producing the value \texttt{[1, 7, 4]}. What value does \texttt{y} have after evaluation of the third statement? The answer is different in Axiom than it is in a language with copying semantics. In Axiom, first the vector \texttt{[1, 2, 3]} is created and the variable \texttt{x} set to \texttt{point} to this object. Let \texttt{y} call this object \texttt{V}. Now \texttt{V} refers to its immutable components \texttt{1, 2, and 3}. Next, the variable \texttt{x} is made to point to \texttt{V} just as \texttt{y} does. Now the third statement interchanges the last 2 elements of \texttt{V} (the \texttt{swap!} operation is destructive, that is, it changes the elements \texttt{in place}). Both \texttt{x} and \texttt{y} perceive this change to \texttt{V}. Thus both \texttt{x} and \texttt{y} then have the value \texttt{[1, 7, 4]}.

In Pascal, the second statement causes a copy of \texttt{V} to be stored under \texttt{y}. Thus the change to \texttt{V} made by the third statement does not affect \texttt{y}.

\textbf{polymorphic} function parameterized by one or more domains; a \texttt{function} defined categorically. Every function defined in a domain or package constructor with a domain-valued parameter is polymorphic. For example, the same matrix \texttt{gfunction} function is used to multiply "matrices over integers" as "matrices over matrices over integers".

\textbf{postfix} operator that follows its single operand. Postfix operators are not available in Axiom.

\textbf{precedence} (syntax) refers to the so-called binding power of an operator. For example, \texttt{a + b * c} is equivalent to...
\( a + (b \times c) \). 

- **precision**
  - the number of digits in the specification of a number, e.g. as set by
  \[ \text{precision} \]
  from \[ \text{gtype} \]: \[ \text{Float} \].

- **predicate**
  - a Boolean valued function, e.g.
    \[ \text{odd: Integer \to Boolean} \].
  - an Boolean valued expression

- **prefix**
  - an operator such as \( \text{-} \) and \( \text{not} \) that is written before its single operand. Every function of one argument can be used as a prefix operator. For example, all of the following have equivalent meaning in Axiom:
    \[ \text{f(x)} \], \[ \text{f \; x} \], and \[ \text{f \; . \; x} \]. See also \[ \text{dot notation} \].

- **quote**
  - the prefix operator \( \text{'} \) meaning do not evaluate.

- **Record**
  - a domain constructor used to create an inhomogeneous aggregate composed of pairs of "selectors" and values. A Record domain is written in the form
    \[ \text{Record(a1:D1, ..., an:Dn)} \]
    (\( n > 0 \)) where \[ \text{Record(a1)} \], ..., \[ \text{Record(an)} \] are identifiers called the selectors of the record, and \[ D1 \], ..., \[ Dn \] are domains indicating the type of the component stored under selector
    \[ \text{Record} \] \[ \text{gspad} \] \[ \text{an} \].

- **recurrence relation**
  - A relation which can be expressed as a function
    \[ f(n) \] with some argument \[ f(n) \] which depends on the value of \[ f(k) \] at \[ k < n \] previous values. In many cases, Axiom will rewrite a recurrence relation on compilation so as to cache its previous \[ f(k) \] values and therefore make the computation significantly more efficient.

- **recursion**
  - use of a self-reference within the body of a function. Indirect
recursion is when a function uses a function below it in the call chain.
</li>
<li><a name="p43948" class="glabel"/>recursive
<ol>
<li>A function that calls itself, either directly or indirectly through another function.
</li>
<li>self-referential. See also <a href="#p43948">recursive</a>.
</li>
</ol>
</li>
<li><a name="p44097" class="glabel"/>reference
see <a href="#p39600">pointer</a>
</li>
<li><a name="p44126" class="glabel"/>Rep
a special identifier used as <a href="#p32278">local variable</a> of a domain constructor body to denote the representation domain for objects of a domain.
</li>
<li><a name="p44277" class="glabel"/>representation
a <a href="#p17507">domain</a> providing a data structure for elements of a domain; generally denoted by the special identifier <a href="#p44126">Rep</a> in the Axiom programming language. As domains are <a href="#p725">abstract datatypes</a>, this representation is not available to users of the domain, only to functions defined in the <a href="#p23911">function body</a> for a domain constructor. Any domain can be used as a representation.
</li>
<li><a name="p44698" class="glabel"/>reserved word
a special sequence of non-blank characters with special meaning in the Axiom language. Examples of reserved words are names such as <div class="gfunction">for</div>, <div class="gfunction">if</div>, and <div class="gfunction">free</div>, operator names such as <div class="gfunction"></div> and <div class="gfunction">mod</div>, special character strings such as <div class="gspad">==</div> and <div class="gspad">:=</div>.
</li>
<li><a name="p45044" class="glabel"/>retraction
to move an object in a parameterized domain back to the underlying domain, for example to move the object <div class="gspad">7</div> from a "fraction of integers" (domain <div class="gtype">Fraction Integer</div>) to "the integers" (domain <div class="gtype">Integer</div>).</li>
<li><a name="p45280" class="glabel"/>return
when leaving a function, the value of the expression following <div class="gfunction">return</div> becomes the value of the function.
</li>
<li><a name="p45405" class="glabel"/>ring
a set with a commutative addition, associative multiplication, a unit element, and multiplication distributes over addition and subtraction.</li>
<li><a name="p45557" class="glabel"/>rule
1. An expression of the form
\[
\text{rule } A \equiv B
\]
indicating a "rewrite rule". 2. An expression of the form
\[
\text{rule}(R_1;\ldots;R_n)
\]
indicating a set of "rewrite rules"
\[
R_1,\ldots,R_n
\]
See \href{p38661}{pattern matching} for details.

### run-time

The time of doing a computation. Contrast
\href{p10167}{compile-time}, rather than prior to it;
\href{p17507}{dynamic} as opposed to
\href{p47594}{static}. For example, the decision of the interpreter
to build a structure such as "matrices with power series entries" in
response to user input is made at run-time.

### run-time check

An error-checking which can be done only when the program receives
user input; for example, confirming that a value is in the proper
range for a computation.

### search order

The sequence of \href{p13571}{default packages} for a given
domain to be searched during \href{p17853}{dynamic lookup}.
This sequence is computed first by ordering the category
\href{p2335}{ancestors} of the domain according to their
\href{gsyntax}{level number}, an integer equal to the minimum distance of the domain from the category. Parents have level
1, parents of parents have level 2, and so on. Among categories with
equal level numbers, ones which appear in the left-most branches of
\href{gsyntax}{Join}s in the source code come first. See also \href{p17853}{dynamic lookup}.

### search string

A string entered into an \href{p28103}{input area} on a screen

### selector

An identifier used to address a component value of a
\href{p43000}{Record} datatype.

### semantics

The relationships between symbols and their meanings. The rules for
obtaining the \href{gsyntax}{meaning} of any syntactically
valid expression.

### semigroup

An \href{p34266}{monoid} which need not have an identity; it is closed and associative.

### side effect

An action which changes a component or structure of a value. See
\href{p14365}{destructive operation} for details.

### signature
an expression describing an
operation. A signature has the form as
\[ \text{name : source} \rightarrow \text{target} \], where
\[ \text{source} \] gives the type of the arguments of the
operation, and \[ \text{target} \] gives the type of the
result.

- **small float**
  the domain for hardware floating point arithmetic as provided by the
  computer hardware.

- **small integer**
  the domain for hardware integer arithmetic, as provided by the
  computer hardware.

- **source**
  the type of the argument of a
  function; the type expression before the
  \[ \rightarrow \] in a \[ \text{signature} \]. For
  example, the source of
  \[ f : \text{(Integer, Integer)} \rightarrow \text{Integer} \]
is
  \[ \text{(Integer, Integer)} \].

- **sparse**
  data structure whose elements are mostly identical (a sparse matrix
  is one filled with mostly zeroes).

- **static**
  that computation done before run-time, such as compilation. Contrast
  \[ \text{dynamic} \].

- **step number**
  the number which precedes user input lines in an interactive session;
  the output of user results is also labeled by this number.

- **stream**
  an object of \[ \text{Stream(R)} \], a generalization of
  a \[ \text{list} \] to allow an infinite number of
  elements. Elements of a stream are computed "on demand". Strings are
  used to implement various forms of power series.

- **string**
  an object of domain \[ \text{String} \]. Strings are
  \[ \text{literals} \] consisting of an arbitrary sequence of
  \[ \text{characters} \] surrounded by double-quotes
  \[ (\text{"}) \], e.g.

- **subdomain**
  a
  \[ \text{domain} \] together with a
  \[ \text{predicate} \] characterizing which members of
  the domain belong to the subdomain. The exports of a subdomain are usually
  distinct from the domain itself. A fundamental assumption however is
that values in the subdomain are automatically coerced to values in the domain. For example, if \( n \) and \( m \) are declared to be members of a subdomain of the integers, then any binary operation from \( \text{Integer} \) is available on \( n \) and \( m \). On the other hand, if the result of that operation is to be assigned to, say, \( k \), also declared to be of that subdomain, a run-time check is generally necessary to ensure that the result belongs to the subdomain.

The right hand side of a rule is called the substitution. The left hand side of a rewrite rule is called a pattern. Rewrite rules can be used to perform pattern matching, usually for simplification.

The use of an expression to filter an iteration.

A syntax operator which placed after its operand. Suffix operators are not allowed in the Axiom language.

Objects denoted by identifiers; an element of domain \( \text{Symbol} \). The interpreter defaultly converts a symbol into \( \text{Variable}(x) \).

Rules of grammar, punctuation etc. for forming correct expressions.

Top-level Axiom statements that begin with. System commands allow users to query the database, read files, trace functions, and so on.

An identifier used to discriminate a branch of a \( \text{Union} \) type.

The \( \text{Type} \) of the result of a function; the type expression following the \( \text{signature} \) in a \( \text{signature} \).

Refers to direct user interactions with the Axiom interpreter.
<li><a name="p50064" class="glabel"/>
totally ordered set</li>
(algebra) a partially ordered set where any two elements are comparable.
</li>
<li><a name="p50148" class="glabel"/>
trace
use of system function <div class="gcmd">trace</div> to track the arguments passed to a function and the values returned.
</li>
<li><a name="p50262" class="glabel"/>
tuple
an expression of two or more other expressions separated by commas, e.g. <div class="gspad">4, 7, 11</div>. Tuples are also used for multiple arguments both for applications (e.g. <div class="gspad">f(x, y)</div>) and in signatures (e.g. <div class="gspad">(Integer, Integer) -> Integer</div>). A tuple is not a data structure, rather a syntax mechanism for grouping expressions.
</li>
<li><a name="p50664" class="glabel"/>
type
The type of any subdomain is the unique symbol <div class="gsyntax">Category</div>. The type of a domain is any category that domain belongs to. The type of any other object is either the (unique) domain that object belongs to or any subdomain of that domain. The type of objects is in general not unique.
</li>
<li><a name="p51002" class="glabel"/>
type checking
a system function which determines whether the datatype of an object is appropriate for a given operation.
</li>
<li><a name="p51114" class="glabel"/>
type constructor
a domain constructor or category constructor.
</li>
<li><a name="p51189" class="glabel"/>
type inference
when the interpreter chooses the type for an object based on context. For example, if the user interactively issues the definition <div align="center" class="gspad">f(x) == (x + %i)**2</div> then issues <div class="gspad">f(2)</div>, the interpreter will infer the type of <div class="gspad">f</div> to be <div class="gspad">Integer -> Complex Integer</div>.
</li>
<li><a name="p51480" class="glabel"/>
unary
operation or function with <div class="hhref" href="#p3173">arity</div> 1
</li>
<li><a name="p51532" class="glabel"/>
underlying domain
for a domain that has a single domain-valued parameter, the underlying domain refers to that parameter. For example, the domain "matrices of integers" (div class="gtype">Matrix Integer</div>) has underlying domain <div class="gtype">Integer</div>.
</li>
<li><a name="p51780" class="glabel"/>
Union
a domain.
constructor used to combine any set of domains into a single domain. A Union domain is written in the form
\[
\text{Union}(a_1:D_1, \ldots, a_n:D_n)
\]
where
\[
(n > 0)
\]
are identifiers called the tags of the union, and
\[
(D_1, \ldots, D_n)
\]
are domains called the branches of the union. The tags are optional, but required when two of the branches are equal, e.g.
\[
\text{Union}(\text{inches:Integer}, \text{centimeters:Integer})
\]. In the interpreter, values of union domains are automatically coerced to values in the branches and vice-versa as appropriate. See also
\[
\text{case}
\].

unit
\[(\text{algebra})\]
an invertible element.

user function
\[
\text{a function defined by a user during an interactive session. Contrast}
\]
\[
\text{built-in function}
\].

user variable
\[
\text{a variable created by the user at top-level during an interactive session}
\]

value
\[
\text{the result of evaluating an expression.}
\]

variable
\[
\text{a means of referring to an object but itself is not an object. A variable has a name and an associated binding created by evaluation of Axiom expressions such as \text{declarations}, \text{assignments}, and \text{definitions}. In the top-level environment of the interpreter, variables are global variables. Such variables can be freely referenced in user-defined functions although a free declaration is needed to assign values to them. See \text{local variable} for details.}
\]

Void
\[
\text{the type given when the value and type of an expression are not needed. Also used when there is no guarantee at run-time that a value and predictable mode will result.}
\]
a symbol which matches any substring including the empty string; for
example, the search string `<div class="gsyntax">*an*</div>` matches an
word containing the consecutive letters `<div class="gsyntax">a</div>`
and `<div class="gsyntax">n</div>`.

A workspace is an interactive record of the user input and output held in an
interactive history file. Each user input and corresponding output
expression in the workspace has a corresponding `<a href="#p47691">step
number</a>`. The current output expression in the workspace is referred
to as `<div class="gspad">%</div>`. The output expression associated
with step number `<div class="gspad">n</div>` is referred to by `<div
class="gspad">n</div>`. The `<div class="gspad">k</div>`-th previous
output expression relative to the current step number `<div
class="gspad">n</div>` is referred to by `<div
class="gspad">-k</div>`. Each interactive `<a href="#p21847">frame</a>` has its own
workspace.

Here are some examples of Axiom graphics.

- Assorted Examples
  Examples of each type of Axiom Graphics
- Three Dimensional Graphics
  Plot parametrically defined surfaces of three functions.
- Functions of One Variable
  Plot curves defined by an equation \( y=f(x) \)
- Parametric Curves
  Plot curves defined by parametric equations \( x=f(t), y=f(t) \)
- Polar Coordinates
  Plot curves given in polar form by an equation \( r=f(\theta) \)
Implicit Curves

Plot non-singular curves defined by a polynomial equation

Lists of Points

Plot lists of points in the (x,y)-plane

Function of two variables: $z=f(x,y)$

$$
\begin{align*}
\text{Function of two variables: } & z=f(x,y) \\
\text{Function of one variable: } & y=f(x) \\
\text{Plane parametric curve: } & x=f(t), y=g(t) \\
\text{Space parametric curve: } & x=f(t), y=g(t), z=h(t)
\end{align*}
$$
CHAPTER 1. OVERVIEW

A Conic Section (Hyperbola)

Polar coordinates: $r = f(\theta)$

Implicit curves: $p(x, y) = 0$
An Elliptic Curve

Cartesian Ovals

Cassinian Ovals: two loops

---

graphexampleslistofpoints.xhtml

--- graphexampleslistofpoints.xhtml ---

```html

```

```javascript

```
Plotting Lists of Points

\[ p := \text{map(point,[[1.,1.],[0.1.],[0.0.],[1.0.],[1.5.],[0.5.],[0.0.5.],[.5.1.],[.25.25],[.25.75],[.75.75]]}) \]


\[ lsize := [6,6,6,6,8,8,8,8,10,10,10,10] \]

\[ pc1 := \text{pastel red()} \]

\[ pc2 := \text{dim green()} \]

\[ pc3 := \text{pastel yellow()} \]

\[ lpc := [pc1,pc1,pc1,pc1,pc2,pc2,pc2,pc2,pc3,pc3,pc3,pc3] \]

\[ lc := [\text{pastel blue()}, \text{light yellow()}, \text{dim green()}, \text{bright red()}, \text{light green()}, \text{dim yellow()}, \text{bright blue()} \]
The `makeViewport2D` command takes a list of options as a parameter in this example. The string "Lines" is designated as the viewport's title.

---

`graphexamplesonevariable.xhtml`

---

`graphexamplesonevariable.xhtml`---
CHAPTER 1. OVERVIEW

---

graphexamplesparametric.xhtml

---

graphexamplesparametric.xhtml ---

The Lemnicate of Bernoulli

Lissajous curve

A gnarly closed curve

Another closed curve
A circle
<ul>
<li>
<input type="submit" id="p1" class="subbut"
onclick="makeRequest('p1');"
value="draw(1,t=0..2*%pi,coordinates==polar)" />
<div id="ansp1"></div></div></li>
</ul>

A spiral
<ul>
<li>
<input type="submit" id="p2" class="subbut"
onclick="makeRequest('p2');"
value="draw(t,t=0..100,coordinates==polar)" />
<div id="ansp2"></div></div></li>
</ul>

A Petal Curve
<ul>
<li>
<input type="submit" id="p3" class="subbut"
onclick="makeRequest('p3');"
value="draw(sin(4*t),t=0..2*%pi,coordinates==polar)" />
<div id="ansp3"></div></div></li>
</ul>

A Limacon
<ul>
<li>
<input type="submit" id="p4" class="subbut"
    onclick="makeRequest('p4');"
    value="draw(2+3*sin(t),t=0..2*%pi,coordinates==polar)" />
<div id="ansp4"><div></div></div>
</li>
</ul>

graphexamplesthreed.xhtml

Pears Surface

<ul>
<li>
<input type="submit" id="p1" class="subbut"
    onclick="makeRequest('p1');"
    value='draw(surface((1+exp(-100*u*u))*sin(%pi*u)*sin(%pi*v),(1+exp(-100*u*u))*sin(%pi*u)*cos(%pi*v),(1+exp(-100*u*u))*cos(%pi*u)),u=0..1,v=0..2,title=="Pear")' />
<div id="ansp1"><div></div></div>
</li>
</ul>

Trigonometric Screw

<ul>
<li>
<input type="submit" id="p2" class="subbut"
    onclick="makeRequest('p2');"
    value='draw(surface(x*cos(y),x*sin(y),y*cos(x)),x=-4..4,y=0..2*%pi,var1Steps==40,var2Steps==40,title=="Trigonometric Screw")' />
<div id="ansp2"><div></div></div>
</li>
</ul>

Etruscan Venus

<ul>
<li>
<input type="submit" id="p3" class="noresult"
    onclick="makeRequest('p3');"
/>
value="a:=1.3*cos(2*x)*cos(y)+sin(y)*cos(x)" />
</li>
</ul>

Banchoff Klein Bottle

<ul>

<li>
<input type="submit" id="p7" class="noresult"
onclick="makeRequest('p7');"
value="f:=(cos(x)*(cos(x/2)*(sqrt(2)+cos(y))+(sin(x/2)*sin(y)*cos(y)))/" />
</li>

<li>
<input type="submit" id="p8" class="noresult"
onclick="makeRequest('p8');"
value="g:=(sin(x)*(cos(x/2)*(sqrt(2)+cos(y))+(sin(x/2)*sin(y)*cos(y)))/" />
</li>

<li>
<input type="submit" id="p9" class="noresult"
onclick="makeRequest('p9');"
value="h:=-sin(x/2)*(sqrt(2)+cos(y)+cos(x/2)*sin(y)*cos(y)/" />
</li>

<li>
<input type="submit" id="p10" class="subbut"
onclick="handleFree(['p7','p8','p9','p10']);"
value='draw(surface(f,g,h),x=0..4*%pi,y=0..%pi,var1Steps==50,var2Steps==50,title="Banchoff Klein Bottle")' />
</li>

\getchunk{page foot}
Axiom can plot curves and surfaces of various types, as well as lists of points in the plane.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Examples
  - See examples of Axiom graphics

- 2D Graphics
  - Graphics in the real and complex plane

- 3D Graphics
  - Plot surfaces, curves, or tubes around curves

- Viewports
  - Customize graphics using Viewports
To get a viewport on a page, you first need to create one in Axiom and write it out to a file that can be called up. For example, we draw a saddle function and assign the result to the variable v.

Now that we've created the viewport, we want to write the data out to a file. To do this, we use the `<a href="dbopwrite.xhtml">write</a>` command which takes as arguments the viewport to write out, the title of the file to be written to, and an optional argument telling the write command what type (or types) of data you want to write (in addition to the ones that Axiom writes). The optional argument could be a string, like "pixmap", or a list of strings, like ["postscript", "pixmap"]'). We need a "bitmap" data type to include a graph in a page so in this case, we write the viewport and tell it to also write a "pixmap" file:

Currently supported file formats are "pixmap", "bitmap", "postscript" and "image".

Axiom automatically adds ".view" at the end of the viewport data file to specify the file type. The ".view" is actually a directory and contains a bitmap file, usually called image.bm.Z, which is a compressed bitmap. Firefox can display bitmap files, as shown here. Clicking on the image should start a "live graphics copy" so you can manipulate the image.
CHAPTER 1. OVERVIEW

Two Dimensional Graphics

- Functions of One Variable: Plot curves defined by an equation $y=f(x)$
- Parametric Curves: Plot curves defined by parametric equations $x=f(t), y=g(t)$
- Polar Coordinates: Plot curves given in polar form by an equation $r=f(\theta)$
- Implicit Curves: Plot non-singular curves defined by a polynomial equation
- List of Points: Plot lists of points in the $(x,y)$-plane

Axiom has facilities for graphing a non-singular algebraic curve in a...
rectangular region of the plane. An algebraic curve is a curve defined by a polynomial equation \( p(x,y) = 0 \). Non-singular means that the curve is "smooth" in that it does not cross itself or come to a point (cusp). Algebraically, this means that for any point \((a,b)\) on the curve (i.e. a point such that \(p(a,b) = 0\)), the partial derivatives \( \frac{dp}{dx}(a,b) \) and \( \frac{dp}{dy}(a,b) \) are not both zero. We require that the polynomial have rational or integral coefficients. Here is a Cartesian ovals algebraic curve example:

\[
\begin{align*}
\text{ul} \\
\text{li} \\
\text{input type="submit" id="p1" class="noresult" onclick="makeRequest('p1');" value="p:=((x^2+y^2+1)-8*x)^2-(8*(x^2+y^2+1)-4*x-1)" />}
\text{div id="ansp1"} & \text{div} & \text{div} & \text{div} \\
\text{li} \\
\text{input type="submit" id="p2" class="subbut" onclick="handleFree(['p1','p2']);" value='draw(p=0,x,y,range==[-1..11,-7..7],title=="Cartesian Ovals")' />}
\text{div id="ansp2"} & \text{div} & \text{div} & \text{div}
\end{align*}
\]

A range must be declared for each variable specified in the algebraic curve equation.

---

**graph2dlistsofpoints.xhtml**

---

**graph2dlistsofpoints.xhtml** —

\(\text{getchunk}{standard\ head}\)
\(\text{getchunk}{text/javascript}\)
\(\text{getchunk}{handlefreevars}\)
\(\text{getchunk}{axiom\ talker}\)
\(\text{getchunk}{page head}\)
\(\text{div align="center"}>List\ of\ Points</div>\)
\(\text{hr}/\)
Axiom has the ability to create lists of points in a two dimensional graphics viewport. This is done by utilizing the
\(\text{a href=\"db.xhtml?GraphImage\"}>GraphImage</a> and
\(\text{a href=\"db.xhtml?TwoDimensionalViewport\"}>TwoDimensionalViewport</a>
domain facilities.

In this example, the \(\text{a href=\"dbopmakegraphimage.xhtml\"}>makeGraphImage</a> takes a list of lists of points parameter, a list of colors for each point in the graph, a list of colors for each line in the graph, and a list of numbers which indicate the size of each point in the graph. The following lines create list of lists of points which can be read be made into two
First we make a list of points:

```html
<input type="submit" id="p1" class="noresult"
onclick="makeRequest('p1');"
value="p:=map(point,[[1.,1.],[0.,1.],[0.,0.],[1.,0.],[1.,5.],[0.,5.],[.5,0.],[0.,5.],[.25,.25],[.25,.75],[.75,.75],[.75,.25]])" />

<p id="ansp1"></p>
```

Then we select pairs of these points which represent the endpoints of lines.

```html
<input type="submit" id="p2" class="noresult"
onclick="makeRequest('p2');"

<p id="ansp2"></p>
```

Next we set the point color and size, and the line color for all components of the graph.

```html
<input type="submit" id="p3" class="noresult"
onclick="makeRequest('p3');"
value="lsize:=[6,6,6,8,8,8,10,10,10,10]" />

<p id="ansp3"></p>
```

```html
<input type="submit" id="p4" class="noresult"
onclick="makeRequest('p4');"
value="pc1:=pastel red()" />

<p id="ansp4"></p>
```

```html
<input type="submit" id="p5" class="noresult"
onclick="makeRequest('p5');"
value="pc2:=dim green()" />

<p id="ansp5"></p>
```

```html
<input type="submit" id="p6" class="noresult"
onclick="makeRequest('p6');"
value="pc3:=pastel yellow()" />

<p id="ansp6"></p>
```

```html
<input type="submit" id="p7" class="noresult"
onclick="makeRequest('p7');"
value="lpc:=[pc1,pc1,pc1,pc2,pc2,pc2,pc2,pc3,pc3,pc3,pc3,pc3]" />

<p id="ansp7"></p>
```
Now the graph image is created and named according to the component specifications indicated above. The <a href="dbopmakeviewport2d.xhtml">makeViewport2D</a> command then creates a two dimensional viewport for this graph according to the list of options specified within the brackets.

The <a href="dbopmakeviewport2d.xhtml">makeViewport2D</a> command takes a list of options as a parameter. In this example the string "Lines" is designated as the viewport’s title.

---

graph2donevariable.xhtml

--- graph2donevariable.xhtml ---

Here we wish to plot a function $y=f(x)$ on the interval $[a,b]$. As an example, let's take the function $y=\sin(\tan(x))-\tan(\sin(x))$ on the interval $[0,6]$. Here is the simplest command that will do this:
CHAPTER 1. OVERVIEW

Notice that Axiom compiled a function before the graph was put on the screen. The expression \( \sin(\tan(x)) - \tan(\sin(x)) \) was converted to a compiled function so that its value for various values of \( x \) could be computed quickly and efficiently. Let's graph the same function on a different interval and this time we'll give the graph a title. The title is a string, which is an optional argument of the command 'draw'.

Once again the expression \( \sin(\tan(x)) - \tan(\sin(x)) \) was converted to a compiled function before any points were computed. If you want to graph the same function on a number of intervals, it's a good idea to write down a function definition so that the function only has to be compiled once. Here's an example:

Notice that our titles can be whatever we want, as long as they are enclosed by double quotes. However, a title which is too long to fit within the viewport title window will be clipped.

\getchunk{page foot}
One way of producing interesting curves is by using parametric equations. Let $x=f(t)$ and $y=g(t)$ for two functions $f$ and $g$ as the parameter $t$ ranges over an interval $[a,b]$. Here's an example:

```plaintext
<input type="submit" id="p1" class="subbut"
onclick="makeRequest('p1');"
value="draw(curve(sin(t)*sin(2*t)*sin(3*t),sin(4*t)*sin(5*t)*sin(6*t)),t=0..2*%pi)" />
<div id="ansp1"><div></div></div>
```

Here $0..2*%pi$ represents the interval over which the variable $t$ ranges. In the case of parametric curves, Axiom will compile two functions, one for each of the functions $f$ and $g$. You may also put a title on a graph. The title may be an arbitrary string and is an optional argument to the command 'draw'. For example:

```plaintext
<input type="submit" id="p2" class="subbut"
onclick="makeRequest('p2');"
value='draw(curve(cos(t),sin(t)),t=0..2*%pi,title=="The Unit Circle")' />
<div id="ansp2"><div></div></div>
```

If you plan on plotting $x=f(t)$, $y=g(t)$ as $t$ ranges over several intervals, you may want to define functions $f$ and $g$, so that they need not be recompiled every time you create a new graph. Here's an example:

```plaintext
<input type="submit" id="p3" class="subbut"
onclick="makeRequest('p3');"
value="f(t:SF):SF == sin(3*t/4)" />
<div id="ansp3"><div></div></div>
```

```plaintext
<input type="submit" id="p4" class="subbut"
onclick="makeRequest('p4');"
value="g(t:SF):SF == sin(t)" />
<div id="ansp4"><div></div></div>
```
These examples show how the curve changes as the range of the parameter $t$ varies.

graph2dpolar.xhtml
You may also define your own functions, when you plan on plotting the same curve as \( \theta \) varies over several intervals.

For information on plotting graphs in other coordinate systems see the pages for the \(<a href="db.xhtml?CoordinateSystems">CoordinateSystems</a>\) domain.
CHAPTER 1. OVERVIEW

Plot a tube around a parametric space curve

Plot surfaces defined by x=f(u,v), y=g(u,v), z=h(u,v)

Create objects constructed from geometric primitives

Create the empty three-space object space.

Add these three curves to the space.

Object can be sent to this space using the operations exported by the ThreeSpace domain. The following examples place curves into space.

---

graph3dobjects.xhtml

-- graph3dobjects.xhtml --

Create the empty three-space object space.

Add these three curves to the space.

Objects can be sent to this space using the operations exported by the ThreeSpace domain. The following examples place curves into space.

Add these three curves to the space.
Create and display the viewport using <a href="dbopmakeviewport3d.xhtml">makeViewport3D</a>. Options may also be given but here are displayed as a list with values enclosed in parentheses.

As a second example of the use of primitives, we generate a cube using a polygon mesh. It is important to use a consistent orientation of the polygons for correct generation of three-dimensional objects.

Again start with an empty three-space object.

For convenience, give the <a href="db.xhtml?DoubleFloat">DoubleFloat</a> values +1 and -1 names.
Define the vertices of the cube.

Define the vertices of the cube.

- \( a \) = point \([x, y, 1::DFLOAT]\)
- \( b \) = point \([y, x, y, 4::DFLOAT]\)
- \( c \) = point \([y, x, x, 8::DFLOAT]\)
- \( d \) = point \([x, x, x, 12::DFLOAT]\)
- \( e \) = point \([x, y, y, 16::DFLOAT]\)
- \( f \) = point \([y, y, y, 20::DFLOAT]\)
- \( g \) = point \([y, y, x, 24::DFLOAT]\)
Add the faces of the cube as polygons to the space using a consistent orientation.

Create and display the viewport.

---

This page describes the plotting in three dimensional space of a curve defined by the parametric equations \( x = f(t) \), \( y = g(t) \), \( z = h(t) \), where \( f, g, \) and \( h \) are functions of the parameter \( t \) which ranges over a specified interval. The basic draw command for this function utilizes either the uncompiled functions or compiled functions format and uses the \(<a href="dbopcurve.xhtml">curve</a>\) command to specify the three functions for the \( x, y, \) and \( z \) components of the curve. The general format for uncompiled functions is:

\[
draw(curve(f(t), g(t), h(t)), t=a..b)
\]
where a..b is the segment defining the interval [a,b] over which the parameter \( t \) ranges. In this case the functions are not compiled until the draw command is executed. Here is an example:

```
<input type="submit" id="p1" class="subbut"
    onclick="makeRequest('p1');"
    value="draw(curve(cos(t),sin(t),t,-12..12)" />
<div id="ansp1"></div></div>
```

In the case of compiled functions, the functions are named and compiled independently. This is useful if you intend to use the functions often, or if the functions are long and complex. The following lines show functions whose parameters are of the type SmallFloat. The functions are compiled and stored by Axiom when entered.

```
<input type="submit" id="p2" class="noresult"
    onclick="makeRequest('p2');"
    value="i1(t:SF):SF==sin(t)*cos(3*t/5)" />
<div id="ansp2"></div></div>
```

```
<input type="submit" id="p3" class="noresult"
    onclick="makeRequest('p3');"
    value="i2(t:SF):SF==cos(t)*cos(3*t/5)" />
<div id="ansp3"></div></div>
```

```
<input type="submit" id="p4" class="noresult"
    onclick="makeRequest('p4');"
    value="i3(t:SF):SF==cos(t)*sin(3*t/5)" />
<div id="ansp4"></div></div>
```

Once the functions are compiled the draw command only needs the names of the functions to execute. Here is a compiled functions example:

```
<input type="submit" id="p5" class="subbut"
    onclick="handleFree(['p2','p3','p4','p5']);"
    value="draw(curve(i1,i2,i3),0..15*%pi)" />
<div id="ansp5"></div></div>
```

Note that the parameter range does not take the variable name as in the case of uncompiled functions. It is understood that the indicated range applies to the parameter of the functions, which in this case is \( t \).
Parametric Surfaces

Graphing a surface defined by $x=f(u,v)$, $y=g(u,v)$, $z=h(u,v)$. This page describes plotting of surfaces defined by the parametric equations of two variables, $x=f(u,v)$, $y=g(u,v)$, and $z=h(u,v)$, for which the ranges of $u$ and $v$ are explicitly defined. The basic draw command for this function utilizes either the uncompiled function or compiled function format and uses the $\text{surface}$ command to specify the three functions for the $x$, $y$, and $z$ components of the surface. The general format for uncompiled functions is:

\[
\text{draw(surface}(f(u,v),g(u,v),h(u,v)), u=a..b, v=c..d)\]

where $a..b$ and $c..d$ are segments defining the intervals $[a,b]$ and $[c,d]$ over which the parameters $u$ and $v$ span. In this case the functions are not compiled until the draw command is executed. Here is an example of a surface plotted using the parabolic cylindrical coordinate system option:

\[
\text{draw(surface}(u*\cos(v),u*\sin(v),v*\cos(u)), u=-4..4, v=0..2*\pi, \text{coordinates==parabolicCylindrical})
\]

In the case of compiled functions, the functions are named and compiled independently. This is useful if you intend to use the functions often, or if the functions are long and complex. The following lines show functions whose parameters are of the type SmallFloat. The functions are compiled and stored by Axiom when entered.

\[
\text{n1}(u:SF,v:SF):SF == u*\cos(v)
\]

\[
\text{n2}(u:SF,v:SF):SF == u*\sin(v)
\]
Once the function is compiled the draw command only needs the names of the functions to execute. Here is a compiled functions example plotted using the toroidal coordinate system option:

This page describes the plotting in three dimensional space of a tube around a parametric space curve defined by the parametric equations $x=f(t)$, $y=g(t)$, $z=h(t)$, where $f$, $g$, and $h$ are functions of the parameter $t$ which ranges over a specified interval. The basic draw command for this function utilizes either the uncompiled functions or compiled functions format and uses the `<a href="dbopcurve.xhtml">curve</a>` command to specify the three functions for the $x$, $y$, and $z$ components of the curve. This uses the same format as that for space curves except that it requires a specification for the radius of the tube. If the radius of the tube is 0, then the result is the space curve itself. The general format for uncompiled functions is:
\begin{verbatim}
  draw(curve(f(t),g(t),h(t)),t=a..b,tubeRadius==r)
\end{verbatim}
where a..b is the segment defining the interval \([a,b]\) over which the parameter \(t\) ranges, and the tubeRadius is indicated by the variable \(r\). In this case the functions are not compiled until the draw command is executed. Here is an example:

\begin{verbatim}
  \begin{itemize}
    \item \(\text{\texttt{draw(curve(sin(t)*cos(3*t/5),cos(t)*cos(3*t/5),cos(t)*sin(3*t/5)),t=0..15*\pi,tubeRadius==.15)}}\)
  \end{itemize}
\end{verbatim}

In the case of compiled functions, the functions are named and compiled independently. This is useful if you intend to use the functions often, or if the functions are long and complex. The following lines show functions whose parameters are of the type SmallFloat. The functions are compiled and stored by Axiom when entered.

\begin{verbatim}
  \begin{itemize}
    \item \(\text{\texttt{t1(t:SF):SF==4/(2-sin(3*t))*cos(2*t)}}\)
    \item \(\text{\texttt{t2(t:SF):SF==4/(2-sin(3*t))*sin(2*t)}}\)
    \item \(\text{\texttt{t3(t:SF):SF==4/(2-sin(3*t))*cos(3*t)}}\)
  \end{itemize}
\end{verbatim}

Once the function is compiled the draw command only needs the names of the functions to execute. Here is a compiled functions example of a trefoil knot:

\begin{verbatim}
  \begin{itemize}
    \item \(\text{\texttt{\{\texttt{t2, t3, p5}\}}}\)
  \end{itemize}
\end{verbatim}

Note that the parameter range does not take the variable name as in the case of uncompiled functions. It is understood that the indicated range applies to the parameter of the functions, which in this case is \(t\).
Typically, the radius of the tube should be set between 0 and 1. A radius of less than 0 results in its positive counterpart and a radius of greater than one cause self-intersection.

---

**graph3dtwovariables.xhtml**

--- **graph3dtwovariables.xhtml** ---

```xml
<script type="text/javascript">
<!--
getchunk{handlefreevars}
getchunk{axiom talker}
</script>
</body>
<getchunk{page head}
<getchunk{page foot}

This page describes the plotting of surfaces defined by an equation of two variables, \( z = f(x,y) \), for which the ranges of \( x \) and \( y \) are explicitly defined. The basic draw command for this function utilizes either the uncompiled function or compiled function format. The general format for an uncompiled function is:

```pre```
draw(f(x,y), x=a..b, y=c..d)
```

where \( a..b \) and \( c..d \) are segments defining the intervals \([a,b]\) and \([c,d]\) over which the variables \( x \) and \( y \) span. In this case, the function is not compiled until the draw command is executed. Here is an example:

```ul```
1. <input type="submit" id="p1" class="subbut" onclick="makeRequest('p1');" value="draw(cos(x*y),x=-3..3,y=-3..3)" />
   <div id="ansp1"><div></div></div>
</ul>

In the case of a compiled function, the function is named and compiled independently. This is useful if you intend to use a function often, or if the function is long and complex. The following line shows a function whose parameters are of the type SmallFloat. The function is compiled and stored by Axiom when it is entered.

```ul```
1. <input type="submit" id="p2" class="noresult" onclick="makeRequest('p2');" value="f(x:SF,y:SF):SF==sin(x)*cos(y)" />
   <div id="ansp2"><div></div></div>
</ul>
```
Once the function is compiled the draw command only needs the name of the function to execute. Here is a compiled function example:

```html
<ul>
  <li>
    <input type="submit" id="p3" class="subbut"
      onclick="handleFree(['p2','p3']);"
      value="draw(f,-%pi..%pi,-%pi..%pi)" />
    <div id="ansp3"><div></div></div>
  </li>
</ul>
```

Note that the parameter ranges do not take the variable names as in the case of uncompiled functions. The variables are entered in the order in which they are defined in the function specification. In this case the first range specifies the x-variable and the second range specifies the y-variable.
Scientific notation is supported for input and output of floating point numbers. A floating point number is written as a string of digits containing a decimal point optionally followed by the letter "E", and then the exponent. We begin by doing some calculations using arbitrary precision floats. The default precision is twenty decimal digits.

\begin{itemize}
\item \texttt{\texttt{sqrt(1.2+2.3/3.4^4.5)}}
\end{itemize}

A decimal base for the exponent is assumed, so the number 1.234E2 denotes 1.234*10**2

The normal arithmetic operations are available for floating point numbers.

\begin{itemize}
\item \texttt{\texttt{sqrt(1.2+2.3/3.4^4.5)}}
\end{itemize}
jenks.xhtml

— jenks.xhtml —

<getchunk{standard head}>
</head>
<body>
<getchunk{page head}>
<center>
<a href="axbook/book-contents.xhtml">
<img src="axbook/ps/lightbayou.png"/>
</a>
</center>
<center>
<h1>
<a href="axbook/book-contents.xhtml">
AXIOM -- Richard D. Jenks and Robert S. Sutor
</a>
</h1>
</center>
<center>
<h2>
<a href="axbook/book-contents.xhtml">
The Scientific Computation System
</a>
</h2>
</center>
<center>
<h2>
<a href="axbook/book-contents.xhtml">
Volume 0 -- The Textbook
</a>
</h2>
</center>
<a href="axbook/book-contents.xhtml#chapter0">
Chapter 0: Introduction to Axiom
</a>
<a href="axbook/book-contents.xhtml#chapter1">
Chapter 1: An Overview of Axiom
</a>
<a href="axbook/book-contents.xhtml#chapter2">
Chapter 2: Using Types and Modes
</a>
<a href="axbook/book-contents.xhtml#chapter3">
Chapter 3: Using HyperDoc
</a>
Enter the formula for the general coefficient of the series:
<input type="text" id="function" size="80" tabindex="10" value="(-1)^(n-1)/(n+2)"/>

Enter the index variable for your formula:
<input type="text" id="ivar" size="10" tabindex="20" value="n"/>

Enter the power series variable:
<input type="text" id="pvar" size="10" tabindex="30" value="x"/>

Enter the point about which to expand:
<input type="text" id="evar" size="10" tabindex="40" value="0"/>
For Laurent Series, the exponent of the power series variable ranges from an initial value, an arbitrary integer value, to plus infinity; the step size is any positive integer.

Enter the initial value of the index (an integer):
<input type="text" id="ival" size="10" tabindex="50" value="-1"/>

Enter the step size (a positive integer):
<input type="text" id="sval" size="10" tabindex="60" value="1"/>

---

linalgpage.xhtml

— linalgpage.xhtml —

<a href="linintro.xhtml">Introduction</a>

Create and manipulate matrices. Work with the entries of a matrix. Perform matrix arithmetic.

<a href="lincreate.xhtml">Creating Matrices</a>
Create matrices from scratch and from other matrices

Algebraic manipulations with matrices. Compute the inverse, determinant, and trace of a matrix. Find the rank, nullspace, and row echelon form of a matrix.

How to compute eigenvalues and eigenvectors

Example: Determinant of a Hilbert Matrix
Computing the Permanent
Working with Vectors
Working with Square Matrices
Working with One-Dimensional Arrays
Conversion is the process of changing an object of one type into an object of another type. The syntax for conversion is object::newType.

By default, 3 has the type PositiveInteger.

We can change this into an object of type Fraction Integer by using "::"

Chapter 1. Overview

Working with Two-Dimensional Arrays

Conversion (Polynomials of Matrices)

Conversion is the process of changing an object of one type into an object of another type. The syntax for conversion is object::newType.

By default, 3 has the type PositiveInteger.

We can change this into an object of type Fraction Integer by using "::".
A coercion is a special kind of conversion that Axiom is allowed to do automatically when you enter an expression. Coercions are usually somewhat safer than more general conversions. The Axiom library contains operations called `<a href="dbopcoerce.xhtml">coerce</a>` and `<a href="dbopconvert.xhtml">convert</a>`. Only the `<a href="dbopcoerce.xhtml">coerce</a>` operations can be used by the interpreter to change an object into an object of another type unless you explicitly use a `"::"`.

By now you will be quite familiar with what types and modes look like. It is useful to think of a type or mode as a pattern for what you want the result to be. Let's start with a square matrix of polynomials with complex rational number coefficients.

```
\begin{itemize}
  \item \texttt{m:SquareMatrix(2,POLY COMPLEX FRAC INT)}
  \item \texttt{m:=matrix \begin{bmatrix} x-3/4*%i,zy^2+1/2 \\
                                3/7*%i*y^4-x,12-%i*9/5 \end{bmatrix}}
\end{itemize}
```

We first want to interchange the `<a href="db.xhtml?Complex">Complex</a>` and `<a href="db.xhtml?Fraction">Fraction</a>` layers. We do the conversion by doing the interchange in the type expression.

```
\begin{itemize}
  \item \texttt{m1:=m::SquareMatrix(2,POLY FRAC COMPLEX INT)}
\end{itemize}
```

Interchange the `<a href="db.xhtml?Polynomial">Polynomial</a>` and the `<a href="db.xhtml?Fraction">Fraction</a>` levels.

```
\begin{itemize}
  \item \texttt{m2:=m1::SquareMatrix(2,FRAC POLY COMPLEX INT)}
\end{itemize}
```

Interchange the `<a href="db.xhtml?Polynomial">Polynomial</a>` and the `<a href="db.xhtml?Complex">Complex</a>` levels.
All the entries have changed types, although in comparing the last two results only the entry in the lower left corner looks different. We did all the intermediate steps to show you what Axiom can do.

In fact, we could have combined all these into one conversion.

There are times when Axiom is not able to do the conversion in one step. You may need to break up the transformation into several conversions in order to get an object of the desired type.

We cannot move either the `<a href="db.xhtml?Fraction">Fraction</a>` or `<a href="db.xhtml?Complex">Complex</a>` above (or to the left of, depending on how you look at it) because each of these levels requires that its argument type have commutative multiplication, whereas `<a href="db.xhtml?SquareMatrix">SquareMatrix</a>` does not. `<a href="db.xhtml?Fraction">Fraction</a>` requires that its argument belong to the category `<a href="db.xhtml?IntegralDomain">IntegralDomain</a>`; and `<a href="db.xhtml?Complex">Complex</a>` requires that it belongs to `<a href="db.xhtml?CommutativeRing">CommutativeRing</a>`. See the `<a href="axbook/section-2.1.xhtml">Jenks section 2.1</a>` for a brief discussion of categories. The `<a href="db.xhtml?Integer">Integer</a>` level did not move anywhere because it does not allow any arguments. We also did not move the `<a href="db.xhtml?SquareMatrix">SquareMatrix</a>` part anywhere, but we could have. Recall that m looks like this:

If we want a polynomial with matrix coefficients rather than a matrix with polynomial entries, we can just do the conversion.
We have not yet used modes for any conversions. Modes are a great shorthand for indicating the type of the object you want. Instead of using the long type expression in the last example we could have simply said this:

We can also indicate more structure if we want the entries of the matrices to be fractions.
value="m:Matrix(Integer):=new(3,3,0)" />
</li>
</ul>
To change the entry in the second row, third column to 5, use
<a href="dbopsetelt.xhtml">setelt</a>.
<ul>
<li>
<input type="submit" id="p2" class="subbut"
onclick="handleFree(['p1','p2']);"
value="setelt(m,2,3,5)" />
</li>
</ul>
An alternative syntax is to use assignment.
<ul>
<li>
<input type="submit" id="p3" class="subbut"
onclick="handleFree(['p1','p3']);"
value="m(1,2):=10" />
</li>
</ul>
The matrix was destructively modified.
<ul>
<li>
<input type="submit" id="p4" class="subbut"
onclick="handleFree(['p1','p4']);"
value="m" />
</li>
</ul>
If you already have the matrix entries as a list of lists, use
<a href="dbopmatrix.xhtml">matrix</a>.
<ul>
<li>
<input type="submit" id="p5" class="subbut"
onclick="makeRequest('p5');"
value="matrix [[1,2,3,4],[0,9,8,7]]" />
</li>
</ul>
If the matrix is diagonal, use
<a href="dbopdiagonalmatrix.xhtml">diagonalMatrix</a>.
<ul>
<li>
<input type="submit" id="p6" class="subbut"
onclick="makeRequest('p6');"
value="dm:=diagonalMatrix [1,x^2,x^3,x^4,x^5]" />
</li>
</ul>
Use <a href="dbopsetrowbang.xhtml">setRow!</a> and
<a href="dbopsetcolumnbang.xhtml">setColumn!</a>.
Use `<a href="dbopcopy.xhtml">copy</a>` to make a copy of a matrix.

Use `<a href="dbopsubmatrix.xhtml">subMatrix</a>` to extract part of an existing matrix. The syntax is

```
subMatrix(m,firstrow,lastrow,firstcol,lastcol)
```

To change a submatrix, use
CHAPTER 1. OVERVIEW

If e is too big to fit where you specify, an error message is displayed. Use

Matrices can be joined either horizontally or vertically to make new matrices.

Use <a href="dbophorizconcat.xhtml">horizConcat</a> to append them side to
The two matrices must have the same number of rows.

Use `<a href="dbopvertconcat.xhtml">vertConcat</a>` to stack one upon the other. The two matrices must have the same number of columns.

The operation `<a href="dboptranspose.xhtml">transpose</a>` is used to create a new matrix by reflection across the main diagonal.

In this section we show you some of Axiom’s facilities for computing and manipulating eigenvalues and eigenvectors, also called characteristic values and characteristic vectors, respectively.

Let’s first create a matrix with integer entries.
Suppose we have a matrix $m1 := \begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & -2 \\ 1 & -2 & 4 \end{bmatrix}$.

To get a list of the rational eigenvalues, use the operation \(\text{eigenvalues}(m1)\).

Given an explicit eigenvalue, \(\text{eigenvector}\) computes the eigenvectors corresponding to it.

The operation \(\text{eigenvectors}(m1)\) returns a list of pairs of values and vectors. When an eigenvalue is rational, Axiom gives you the value explicitly; otherwise, its minimal polynomial is given, (the polynomial of lowest degree with the eigenvalues as roots), together with a parametric representation of the eigenvector using the eigenvalue. This means that if you ask Axiom to \(\text{solve}\) the minimal polynomial, then you can substitute these roots into the parametric form of the corresponding eigenvectors.

You must be aware that unless an exact eigenvalue has been computed, the eigenvector may be badly in error.

Another possibility is to use the operation \(\text{radicalEigenvectors}(m1)\) which tries to compute explicitly the eigenvectors in terms of radicals.
Alternatively, Axiom can compute real or complex approximations to the
eigenvectors and eigenvalues using the operations
\(<a href="dboprealeigenvectors.xhtml">realEigenvectors</a> or
\(<a href="dbopcomplexeigenvectors.xhtml">complexEigenvectors</a>\). They
each take an additional argument epsilon to specify the "precision"
required. In the real case, this means that each approximation will be
within plus or minus epsilon of the actual result. In the complex case, this
means that each approximation will be within plus or minus epsilon of the
actual result in each of the real and imaginary parts.

The precision can be specified as a \(<a href="db.xhtml?Float">Float</a>\) if
the results are desired in floating-point notation, or as
\(<a href="dbfractioninteger.xhtml">Fraction Integer</a>\) if the results are
to be expressed using rational (or complex rational) numbers.

If an \(n\) by \(n\) matrix has \(n\) distinct eigenvalues (and therefore \(n\) eigenvectors)
the operation \(<a href="dbopeigenmatrix.xhtml">eigenMatrix</a>\) gives you a
matrix of the eigenvectors.

If a symmetric matrix has a basis of orthonormal eigenvectors, then
\(<a href="dboporthonormalbasis.xhtml">orthonormalBasis</a>\) computes a list
of these vectors.
Consider the problem of computing the determinant of a 10 by 10 Hilbert matrix. The \((i,j)\)-th entry of a Hilbert matrix is given by \(1/(i+j+1)\).

First do the computation using rational numbers to obtain the exact result.

\[
a := \text{MATRIX FRAC INT} := \text{matrix} \left[ \frac{1}{i+j+1} \text{ for } j \text{ in } 0\ldots9 \text{ for } i \text{ in } 0\ldots9 \right]
\]

\[
d := \text{determinant} \ a
\]

\[
d := \text{Float}
\]
The result given by hardware floats is correct only to four significant digits of precision. In the jargon of numerical analysis, the Hilbert matrix is said to be "ill-conditioned".

Now repeat the computation at a higher precision using Float.
To get higher dimensional aggregates, you can create one-dimensional aggregates with elements that are themselves aggregates, for example, lists of list, one-dimensional arrays of list of multisets, and so on. For applications requiring two-dimensional homogeneous aggregates, you will likely find two-dimensional arrays and matrices useful.

The entries in `<a href="db.xhtml?TwoDimensionalArray">TwoDimensionalArray</a>` and `<a href="?Matrix">Matrix</a>` objects are all the same type, except that those for `<a href="db.xhtml?Matrix">Matrix</a>` must belong to a `<a href="db.xhtml?Ring">Ring</a>`. You create and access elements in roughly the same way. Since matrices have an understood algebraic structure, certain algebraic operations are available for matrices but not for arrays. Because of this, we limit our discussion here to `<a href="db.xhtml?Matrix">Matrix</a>`, that can be regarded as an extension of `<a href="db.xhtml?TwoDimensionalArray">TwoDimensionalArray</a>`. See `<a href="pagetwodimensionalarray.xhtml">TwoDimensionalArray</a>` for more information about Axiom's linear algebra facilities.

You can create a matrix from a list of lists, where each of the inner lists represents a row of the matrix.

```html
<ul>
<li>
<input type="submit" id="p1" class="subbut" onclick="makeRequest('p1');"
value="m:=matrix([[1,2],[3,4]])" />
</li>
</ul>
```

The "collections" construct (see `<a href="axbook/section-5.5.xhtml">Computation of Eigenvalues and Eigenvectors</a>`, and `<a href="axbook/section-8.5.xhtml">Solution of Linear and Polynomial Equations</a>"
Creating Lists and Streams with Iterators is useful for creating matrices whose entries are given by formulas.

Let $vm$ denote the three by three Vandermonde matrix.

You can also pull out a row or a column.

You can do arithmetic.

You can perform operations such as $\text{transpose}$, $\text{trace}$, and $\text{determinant}$.
linoperations.xhtml

Axiom provides both left and right scalar multiplication. You can add, subtract, and multiply matrices provided, of course, that the matrices have compatible dimensions. If not, an error message is displayed.

This following product is defined but n*m is not.
The operations `nrows` and `ncols` return the number of rows and columns of a matrix. You can extract a row or a column of a matrix using the operations `row` and `column`. The object returned is a `Vector`. Here is the third column of the matrix `n`.

You can multiply a matrix on the left by a "row vector" and on the right by a "column vector".

The operation `inverse` computes the inverse of a matrix if the matrix is invertible, and returns "failed" if not. This Hilbert matrix invertible.

This matrix is not invertible.
The operation <a href="dbopdeterminant.xhtml">determinant</a> computes the determinant of a matrix provided that the entries of the matrix belong to a <a href="db.xhtml?CommutativeRing">CommutativeRing</a>. The above matrix mm is not invertible and, hence, must have determinant 0.

The operation <a href="dboptrace.xhtml">trace</a> computes the trace of a square matrix.

The operation <a href="dboprank.xhtml">rank</a> computes the rank of a matrix: the maximal number of linearly independent rows or columns.

The operation <a href="dbopnullity.xhtml">nullity</a> computes the nullity of a matrix: the dimension of its null space.

The operation <a href="dbopnullspace.xhtml">nullSpace</a> returns a list containing a basis for the null space of a matrix. Note that the nullity is the number of elements in a basis for the null space.
The operation <a href="dbopprovechelon.xhtml">rowEchelon</a> returns the row echelon form of a matrix. It is easy to see that the rank of this matrix is two and that its nullity is also two.

For more information see <a href="axbook/section-1.6.xhtml">Expanding to Higher Dimensions</a>, <a href="axbook/section-8.4.xhtml">Computation of Eigenvalues and Eigenvectors</a>, and <a href="axbook/section-9.27.xhtml#subsec-9.27.4">An Example: Determinant of a Hilbert Matrix</a>. Also see <a href="db.xhtml?Permanent">Permanent</a>, <a href="db.xhtml?Vector">Vector</a>, <a href="db.xhtml?OneDimensionalArray">OneDimensionalArray</a>, and <a href="db.xhtml?TwoDimensionalArray">TwoDimensionalArray</a>. Issue the system command

```plaintext
show Matrix
```

to display the full list of operations defined by <a href="db.xhtml?Matrix">Matrix</a>.

linpermaent.xhtml

— linpermaent.xhtml —

```plaintext
Permanent
```
The package <a href="db.xhtml?Permanent">Permanent</a> provides the function <a href="dboppermanent.xhtml">permanent</a> for square matrices. The <a href="dboppermanent.xhtml">permanent</a> of a square matrix can be computed in the same way as the determinant by expansion of minors except that for the permanent the sign for each element is 1, rather than being 1 if the row plus column indices is positive and -1 otherwise. This function is much more difficult to compute efficiently than the <a href="dbopdeterminant.xhtml">determinant</a>. An example of the use of <a href="dboppermanent.xhtml">permanent</a> is the calculation of the nth derangement number, defined to be the number of different possibilities for n couples to dance but never with their own spouse. Consider an n by x matrix with entries 0 on the diagonal and 1 elsewhere. Think of the rows as one-half of each couple (for example, the males) and the columns the other half. The permanent of such a matrix gives the desired derangement number.

<ul>
<li>\[ \text{permanent}(\text{kn}(n)::\text{SMATRIX}(n, \text{INT})) \text{ for } n \text{ in } 1..13 \]
</li>
</ul>

Here are some derangement numbers, which you see grow quite fast.

<ul>
</ul>

linsquarematrices.xhtml
The top level matrix type in Axiom is \[ \text{Matrix}, \] see \( \text{Matrix} \), which provides basic arithmetic and linear algebra functions. However, since the matrices can be of any size it is not true that any pair can be added or multiplied. Thus \( \text{Matrix} \) has little algebraic structure.

Sometimes you want to use matrices as coefficients for polynomials or in other algebraic contexts. In this case, \( \text{SquareMatrix} \) should be used. The domain \( \text{SquareMatrix}(n,R) \) gives the ring of \( n \) by \( n \) square matrices over \( R \).

The usual arithmetic operations are available.

Square matrices can be used where ring elements are required. For example, here is a matrix with matrix entries.

Or you can construct a polynomial with square matrix coefficients.

This value can be converted to a square matrix with polynomial coefficients.
The <a href="db.xhtml?Vector">Vector</a> domain is used for storing data in a one-dimensional indexed data structure. A vector is a homogeneous data structure in that all the components of the vector must belong to the same Axiom domain. Each vector has a fixed length specified by the user; vectors are not extensible. This domain is similar to the <a href="db.xhtml?OneDimensionalArray">OneDimensionalArray</a> domain, except that when the components of a <a href="db.xhtml?Vector">Vector</a> belong to a <a href="db.xhtml?Ring">Ring</a>, arithmetic operations are provided. For more examples of operations that are defined for both <a href="db.xhtml?Vector">Vector</a> and <a href="db.xhtml?OneDimensionalArray">OneDimensionalArray</a>, see <a href="pageonedimensionalarray.xhtml">OneDimensionalArray</a>.
As with the `<a href="db.xhtml?OneDimensionalArray">OneDimensionalArray</a>` domain, a
`<a href="db.xhtml?Vector">Vector</a>` can be created by calling the operation
`<a href="dbopnew.xhtml">new</a>`<a href="db.xhtml?Vector">/</a>, its components can be accessed by calling
the operations `<a href="dbopelt.xhtml">elt</a>`<a href="db.xhtml?Vector">/</a> and
`<a href="dbopqelt.xhtml">qelt</a>`<a href="db.xhtml?Vector">/</a>, and its components can be reset by
calling the operations
`<a href="dbopsetelt.xhtml">setelt</a>`<a href="db.xhtml?Vector">/</a> and
`<a href="dbopseteltbang.xhtml">setelt!</a>`<a href="db.xhtml?Vector">/</a>. This creates a vector of
integers of length 5 all of whose components are 12.

This is how you create a vector from a list of its components.

Indexing for vectors begins at 1. The last element has index equal to
the length of the vector, which is computed by
`<a href="dboplength.xhtml">#</a>`.<a href="db.xhtml?Vector">/</a>

This is the standard way to use `<a href="dbopelt.xhtml">elt</a>` to extract
an element.

This is the standard way to use `setelt` to change an element. It is the
same as if you had typed `setelt(v,3,99)`.
Now look at $v$ to see the change. You can use $qelt$ and $qsetelt!$ (instead of $elt$ and $setelt$, respectively) but only when you know that the index is within the valid range.

When the components belong to a Ring, Axiom provides arithmetic operations for Vector. These include left and right scalar multiplication.

Addition and subtraction are also available

Of course, when adding or subtracting, the two vectors must have the same length or an error message is displayed.
The <a href="db.xhtml?OneDimensionalArray">OneDimensionalArray</a> is used for storing data in a one-dimensional indexed data structure. Such an array is a homogeneous data structure in that all the entries of the array must belong to the same Axiom domain. Each array has a fixed length specified by the user and arrays are not extensible. The indexing of one-dimensional arrays is one-based. This means that the "first" element of an array is given the index 1. See also <a href="db.xhtml?Vector">Vector</a> and <a href="db.xhtml?FlexibleArray">FlexibleArray</a>. To create a
one-dimensional array, apply the operation
<a href="dboponedimensionalarray.xhtml">oneDimensionalArray</a> to a list.

Another approach is to first create a, a one-dimensional array of 10 0’s.
<a href="db.xhtml?OneDimensionalArray">OneDimensionalArray</a> has a convenient abbreviation
<a href="db.xhtml?OneDimensionalArray">ARRAY1</a>.

Set each ith element to i, then display the result.

Square each element by mapping the function i+->i**2 onto each element.

Reverse the elements in place.

Swap the 4th and 5th element.
value="swap!(a,4,5); a" />
</div id="ansp6"></div></div>
</li>
</ul>
Sort the elements in place.
<ul>
<li>
<input type="submit" id="p7" class="subbut"
onclick="handleFree(['p1','p2','p3','p4','p5','p6','p7']);"
value="sort! a" />
</div id="ansp7"></div></div>
</li>
</ul>
Create a new one-dimensional array b containing the last 5 elements of a.
<ul>
<li>
<input type="submit" id="p8" class="subbut"
onclick="handleFree(['p1','p2','p3','p4','p5','p6','p7','p8']);"
value="b:=a(6..10)" />
</div id="ansp8"></div></div>
</li>
</ul>
Replace the first 5 elements of a with those of b.
<ul>
<li>
<input type="submit" id="p9" class="subbut"
onclick="handleFree(['p1','p2','p3','p4','p5','p6','p7','p8','p9']);"
value="copyInto!(a,b,1)" />
</div id="ansp9"></div></div>
</li>
</ul>
------

lin2darrays.xhtml

---------

lin2darrays.xhtml

---------

\getchunk{standard head}
\getchunk{handlefreevars}
\getchunk{axiom talker}
\script
\getchunk{page head}

The <a href="db.xhtml?TwoDimensionalArray">TwoDimensionalArray</a> is used for storing data in a two-dimensional data structure indexed by row and by column. Such an array is a homogeneous data structure in that all the entries of the array must belong to the same Axiom domain (although see
each array has a fixed number of rows and columns specified by the user and arrays are not extensible. In Axiom, the indexing of two-dimensional arrays is one-based. This means that both the "first" row of an array and the "first" column of an array are given the index 1. Thus, the entry in the upper left corner of an array is in position (1,1).

The operation new creates an array with a specified number of rows and columns and fills the components of that array with a specified entry. The arguments of this operation specify the number of rows, the number of columns, and the entry. This creates a five-by-four array of integers, all of which are zero.

```html
<input type="submit" id="p1" class="subbut" onclick="makeRequest('p1');" value="arr:ARRAY2 INT:=new(5,4,0)" />
</li>
</ul>

The entries of this array can be set to other integers using the operation elt.

```html
<input type="submit" id="p2" class="subbut" onclick="handleFree(['p1','p2']);" value="setelt(arr,1,1,17)" />
</div></div></li></ul>

Now the first element of the array is 17.

```html
<input type="submit" id="p3" class="subbut" onclick="handleFree(['p1','p2','p3']);" value="arr" />
</div></div></li></ul>

Likewise, elements of an array are extracted using the operation elt.

```html
<input type="submit" id="p4" class="subbut" onclick="handleFree(['p1','p2','p4']);" value="elt(arr,1,1)" />
</div></div></li></ul>

Another way to use these two operations is as follows. This sets the element in position (3,2) of the array to 15.
This extracts the element in position (3,2) of the array.

The operations \(<a href="dbopelt.xhtml">elt</a>\) and \(<a href="dbopsetelt.xhtml">setelt</a>\) come equipped with an error check which verifies that the indices are in the proper ranges. For example, the above array has five rows and four columns, so if you ask for the entry in position (6,2) with \(arr(6,2)\) Axiom displays an error message. If there is no need for an error check, you can call the operations \(<a href="dbopqelt.xhtml">qelt</a>\) and \(<a href="dbopqseteltbang.xhtml">qsetelt!</a>\) which provide the same functionality but without the error check. Typically, these operations are called in well-tested programs.

The operations \(<a href="dboprow.xhtml">row</a>\) and \(<a href="dbopcolumn.xhtml">column</a>\) extract rows and columns, respectively, and return objects of \(<a href="db.xhtml?OneDimensionalArray">OneDimensionalArray</a>\) with the same underlying element type.

You can determine the dimensions of an array by calling the operations \(<a href="dbopnrows.xhtml">nrows</a>\) and \(<a href="dbopncols.xhtml">ncols</a>\), which return the number of rows and columns, respectively.
To apply an operation to every element of an array, use <a href="dbopmap.xhtml">map</a>. This creates a new array. This expression negates every element.

```
<input type="submit" id="p11" class="subbut"
onclick="handleFree(['p1','p2','pS','p11']);"
value="map(-,arr)" />
```

This creates an array where all the elements are doubled.

```
<input type="submit" id="p12" class="subbut"
onclick="handleFree(['p1','p2','pS','p12']);"
value="map((x+->x+x),arr)" />
```

To change the array destructively, use <a href="dbopmapbang.xhtml">map!</a> instead of <a href="dbopmap.xhtml">map</a>. If you need to make a copy of any array, use <a href="dbopcopy.xhtml">copy</a>.

```
<input type="submit" id="p13" class="subbut"
onclick="handleFree(['p1','p2','pS','p13']);"
value="arrc:=copy(arr)" />
```

```
<input type="submit" id="p14" class="subbut"
onclick="handleFree(['p1','p2','pS','p13','p14']);"
value="map!(-,arrc)" />
```

```
<input type="submit" id="p15" class="subbut"
onclick="handleFree(['p1','p2','pS','p13','p14','p15']);"
value="arrc" />
```
Use `<a href="dbopmemberq.xhtml">member?</a>` to see if a given element is in an array.

To see how many times an element appears in an array, use `<a href="dbopcount.xhtml">count</a>`.

For more information about the operations available for `<a href="db.xhtml?TwoDimensionalArray">TwoDimensionalArray</a>`, issue

For more information on related topics, see `<a href="pagematrix.xhtml">Matrix</a>` and `<a href="lin1darrays.xhtml">OneDimensionalArray</a>`.

\getchunk{page foot}
Enter search string (use <b>*</b> for wild card unless counter-indicated):
<form>
<input type="text" name="searchbox" size="50"/>
</form>

<table>
<tr>
<td><a href="(|kSearch| '|\stringvalue{pattern}|).xhtml">Constructors</a></td>
<td>Search for categories, domains, or packages.</td>
</tr>
<tr>
<td><a href="(|oSearch| '|\stringvalue{pattern}|).xhtml">Operations</a></td>
<td>Search for operations.</td>
</tr>
<tr>
<td><a href="(|aSearch| '|\stringvalue{pattern}|).xhtml">Attributes</a></td>
<td>Search for attributes.</td>
</tr>
</table>
| **General** | Search for all three of the above. |
| **Documentation** | Search library documentation. |
| **Complete** | All of the above. |
| **Selective** | Detailed search with selectable options. |
| **Reference** | Search Reference documentation (\* wild card is not accepted). |
| **Commands** | View system command documentation. |
menualgebraeigenvalues.xhtml

— menualgebraeigenvalues.xhtml —

menualgebraeigenvectors.xhtml

— menualgebraeigenvectors.xhtml —

menualgebraentermatrix.xhtml

— menualgebraentermatrix.xhtml —

menualgebrainvertmatrix.xhtml

— menualgebrainvertmatrix.xhtml —
<table>
<thead>
<tr>
<th>menualgebrageneratematrix.xhtml</th>
<th>menualgebramakelist.xhtml</th>
<th>menualgebramaptolist.xhtml</th>
<th>menualgebramaptomatrix.xhtml</th>
</tr>
</thead>
</table>

---

menualgebrageneratematrix.xhtml

---

menualgebramakelist.xhtml

---

menualgebramaptolist.xhtml

---

menualgebramaptomatrix.xhtml

---

CHAPTER 1. OVERVIEW
menualgebrareducelist.xhtml

---

menualgebratreduce list.xhtml

---

menualgebratransposematrix.xhtml

---

menuaxiomaddtopath.xhtml

---
CHAPTER 1. OVERVIEW

menuaxiomclearmemory.xhtml

— menuaxiomclearmemory.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menuaxiomclearmemory not implemented
\getchunk{page foot}

——

menuaxiomdeletefunction.xhtml

— menuaxiomdeletefunction.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menuaxiomdeletefunction not implemented
\getchunk{page foot}

——

menuaxiomdeletevariable.xhtml

— menuaxiomdeletevariable.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menuaxiomdeletevariable not implemented
\getchunk{page foot}

——

menuaxiominterrupt.xhtml

— menuaxiominterrupt.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menuaxiominterrupt not implemented
\getchunk{page foot}

——
menuaxiomrestart.xhtml

--- menuaxiomrestart.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menuaxiomrestart not implemented
\getchunk{page foot}

---

menuaxiomshowdefinition.xhtml

--- menuaxiomshowdefinition.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menuaxiomshowdefinition not implemented
\getchunk{page foot}

---

menuaxiomdisplay.xhtml

--- menuaxiomdisplay.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menuaxiomdisplay not implemented
\getchunk{page foot}

---

menuaxiomset.xhtml

--- menuaxiomset.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menuaxiomset not implemented
\getchunk{page foot}

---
CHAPTER 1. OVERVIEW

menuaxiomshowfunctions.xhtml

--- menuaxiomshowfunctions.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menuaxiomshowfunctions not implemented
\getchunk{page foot}

---

menuaxiomshowvariables.xhtml

--- menuaxiomshowvariables.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menuaxiomshowvariables not implemented
\getchunk{page foot}

---

menuaxiomtoggletimedisplay.xhtml

--- menuaxiomtoggletimedisplay.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menuaxiomtoggletimedisplay not implemented
\getchunk{page foot}

---

menucalculuscalculussum.xhtml

--- menucalculuscalculussum.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menucalculuscalculussum not implemented
\getchunk{page foot}

---
menueditpaste.xhtml

    — menueditpaste.xhtml —

    \getchunk{standard head}
    </head>
    <body>
    \getchunk{page head}
    menueditpaste not implemented
    \getchunk{page foot}
    

menueditdeleteselection.xhtml

    — menueditdeleteselection.xhtml —

    \getchunk{standard head}
    </head>
    <body>
    \getchunk{page head}
    menueditdeleteselection not implemented
    \getchunk{page foot}
    

menueditselectiontoimage.xhtml

    — menueditselectiontoimage.xhtml —

    \getchunk{standard head}
    </head>
    <body>
    \getchunk{page head}
    menueditselectiontoimage not implemented
    \getchunk{page foot}
    

menueditselectiontoinput.xhtml

    — menueditselectiontoinput.xhtml —

    \getchunk{standard head}
    </head>
    <body>
    \getchunk{page head}
    menueditselectiontoinput not implemented
    \getchunk{page foot}
menuequationsrealrootsofpolynomial.xhtml

--- menuequationsrealrootsofpolynomial.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menuequationsrealrootsofpolynomial not implemented
\getchunk{page foot}

---

menuequationsatvalue.xhtml

--- menuequationsatvalue.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menuequationsatvalue not implemented
\getchunk{page foot}

---

menuequationsboundaryvalueproblem.xhtml

--- menuequationsboundaryvalueproblem.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menuequationsboundaryvalueproblem not implemented
\getchunk{page foot}

---

menuequationsinitialvalueproblem1.xhtml

--- menuequationsinitialvalueproblem1.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menuequationsinitialvalueproblem1 not implemented
\getchunk{page foot}

---
menuequationssolvenumerically.xhtml

--- menuequationssolvenumerically.xhtml ---
\getchunk{standard head}
</head>
<body>
\getchunk{page head}

menuequationssolvenumerically not implemented
\getchunk{page foot}

---

menufileexit.xhtml

--- menufileexit.xhtml ---
\getchunk{standard head}
</head>
<body>
\getchunk{page head}

menufileexit not implemented
\getchunk{page foot}

---

menufileinputfile.xhtml

--- menufileinputfile.xhtml ---
\getchunk{standard head}
</head>
<body>
\getchunk{page head}

menufileinputfile not implemented
\getchunk{page foot}

---

menufileloadlibrary.xhtml

--- menufileloadlibrary.xhtml ---
\getchunk{standard head}
</head>
<body>
\getchunk{page head}

menufileloadlibrary not implemented
\getchunk{page foot}

---
menufileopen.xhtml

--- menufileopen.xhtml ---
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menufileopen not implemented
\getchunk{page foot}

---

menufileprint.xhtml

--- menufileprint.xhtml ---
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menufileprint not implemented
\getchunk{page foot}

---

menufileread.xhtml

--- menufileread.xhtml ---
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menufileread not implemented
\getchunk{page foot}

---

menufilesave.xhtml

--- menufilesave.xhtml ---
\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menufilesave not implemented
\getchunk{page foot}

---
menufilesaveas.xhtml

--- menufilesaveas.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menufilesaveas not implemented
\getchunk{page foot}

---

menufiletogglespool.xhtml

--- menufiletogglespool.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menufiletogglespool not implemented
\getchunk{page foot}

---

menunumericsetprecision.xhtml

--- menunumericsetprecision.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menunumericsetprecision not implemented
\getchunk{page foot}

---

menunumerictobigfloat.xhtml

--- menunumerictobigfloat.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menunumerictobigfloat not implemented
\getchunk{page foot}

---
CHAPTER 1. OVERVIEW

menunumericstofloat.xhtml

--- menunumericstofloat.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menunumericstofloat not implemented
\getchunk{page foot}


---

menunumericstogglenumericoutput.xhtml

--- menunumericstogglenumericoutput.xhtml ---

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menunumericstogglenumericoutput not implemented
\getchunk{page foot}


---

menusimplifyaddalgebraicequality.xhtml

--- menusimplifyaddalgebraicequality.xhtml ---

\getchunk{standard headaddalgebraicequality}
</head>
<body>
\getchunk{page head}
menusimplifyaddalgebraicequality not implemented
\getchunk{page foot}


---

menusimplifycomplexsimplification.xhtml

--- menusimplifycomplexsimplification.xhtml ---

\getchunk{standard headcomplexsimplification}
</head>
<body>
\getchunk{page head}
menusimplifycomplexsimplification not implemented
\getchunk{page foot}


---
menusimplifysimplifyexpression.xhtml

— menusimplifysimplifyexpression.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menusimplifysimplifyexpression not implemented
\getchunk{page foot}

———

menusimplifysubtitute.xhtml

— menusimplifysubtitute.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menusimplifysubtitute not implemented
\getchunk{page foot}

———

menusimplifysimplifyradicals.xhtml

— menusimplifysimplifyradicals.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menusimplifysimplifyradicals not implemented
\getchunk{page foot}

———

menusimplifytogglealgebraicflag.xhtml

— menusimplifytogglealgebraicflag.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
menusimplifytogglealgebraicflag not implemented
\getchunk{page foot}

———
The size of an integer in Axiom is only limited by the amount of computer storage you have available. The usual arithmetic operations are available.

There are a number of ways of working with the sign of an integer. Let’s use the $x$ as an example.

First of all, there is the absolute value function.
The `<a href="dbopsign.xhtml">sign</a>` operation returns -1 if its argument is negative, 0 if zero and 1 if positive.

You can determine if an integer is negative in several other ways.

Similarly, you can find out if it is positive.
This is the recommended way of determining whether an integer is zero.

Use the &lt;a href="dbopzeroq.xhtml"&gt;zero?&lt;/a&gt; whenever you are testing any mathematical object for equality with zero. This is usually more efficient than using &lt;a href="dbopequal.xhtml">=</a> (think of matrices: it is easier to tell if a matrix is zero by just checking term by term than constructing another "zero" matrix and comparing the two matrices term by term) and also avoids the problem that &lt;a href="dbopequal.xhtml">=</a> is usually used for creating equations.

This is the recommended way of determining whether an integer is equal to one.

This syntax is used to test equality using &lt;a href="dbopequal.xhtml">=</a>. It says that you want a &lt;a href="db.xhtml?Boolean">Boolean</a>&lt;/a&gt; (true or false) answer rather than an equation.

The operations &lt;a href="dbopoddq.xhtml">odd?&lt;/a&gt; and &lt;a href="dbopevenq.xhtml">even?&lt;/a&gt; determine whether an integer is odd or even, respectively. They each return a &lt;a href="db.xhtml?Boolean">Boolean</a>&lt;/a&gt; object.
The operation `<a href="dbopgcd.xhtml">gcd</a>` computes the greatest common divisor of two integers.

The operation `<a href="dboplcm.xhtml">lcm</a>` computes their least common multiple.

To determine the maximum of two integers, use `<a href="dbopmax.xhtml">max</a>`.

To determine the minimum, use `<a href="dbopmin.xhtml">min</a>`.

The `<a href="dbopreduce.xhtml">reduce</a>` operation is used to extend binary operations to more than two arguments. For example, you can use `<a href="dbopreduce.xhtml">reduce</a>` to find the maximum integer in a list or compute the least common multiple of all integers in a list.
The infix operator "/' is not used to compute the quotient of integers. Rather, it is used to create rational numbers as described in Fractions.

The infix operator <a href="dbopquo.xhtml">quo</a> computes the integer quotient.

The infix operation <a href="dboprem.xhtml">rem</a> computes the integer remainder.

One integer is evenly divisible by another if the remainder is zero. The operation <a href="dbopexquo.xhtml">exquo</a> can also be used. See Unions for an example.
The operation `<a href="dbopdivide.xhtml">divide</a>` returns a record of the quotient and remainder and thus is more efficient when both are needed.

Records are discussed in detail in `<a href="axbook/section-2.4.xhtml">Records</a>`.

The following types of numbers are among those available in Axiom

<table>
<thead>
<tr>
<th>Integers</th>
<th>Arithmetic with arbitrarily large integers</th>
</tr>
</thead>
</table>
CHAPTER 1. OVERVIEW

- Fractions: Rational numbers and general fractions
- Machine Floats: Fixed precision machine floating point arithmetic
- Real Numbers: Arbitrary precision decimal arithmetic
- Complex Numbers: Complex numbers in general
- Finite Fields: Arithmetic in characteristic p

Additional topics:
- Numeric Functions
- Cardinal Numbers
- Machine-sized Integers
- Roman Numerals
- Continued Fractions
- Partial Fractions
- Quaternions
- Octonions
The <a href="dbopcardinalnumber.xhtml">CardinalNumber</a> can be used for values indicating the cardinality of sets, both finite and infinite. For example, the <a href="dbopdimension.xhtml">dimension</a> operation in the category <a href="dbopvectorspace.xhtml">VectorSpace</a> returns a cardinal number.

The non-negative integers have a natural construction as cardinals

\pre
0=#{ }, 1={0}, 2={0,1}, ..., n={i | 0 \leq i < n}
\pre

The fact that 0 acts as a zero for the multiplication of cardinals is equivalent to the axiom of choice.

Cardinal numbers can be created by conversion from non-negative integers.

\ul
\li
<input type="submit" id="p1" class="subbut" onclick="makeRequest('p1');" value="c0:=0::CardinalNumber" />
<div id="ansp1"></div></div></li>
\li
<input type="submit" id="p2" class="subbut" onclick="makeRequest('p2');" value="c1:=1::CardinalNumber" />
<div id="ansp2"></div></div></li>
The can also be obtained as the named cardinal $\text{Aleph}(n)$

Similarly, the <a href="dbopcountableq.xhtml">countable?</a> operation determines whether a value is a countable cardinal, that is, finite or $\text{Aleph}(0)$. 

Arithmetic operations are defined on cardinal numbers as follows:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x+y = #(X+Y)</td>
<td>cardinality of the disjoint union</td>
</tr>
<tr>
<td>x-y = #(X-Y)</td>
<td>cardinality of the relative complement</td>
</tr>
<tr>
<td>x<em>y = #(X</em>Y)</td>
<td>cardinality of the Cartesian product</td>
</tr>
<tr>
<td>x<strong>y = #(X</strong>Y)</td>
<td>cardinality of the set of maps from Y to X</td>
</tr>
</tbody>
</table>

Here are some arithmetic examples:
Subtraction is a partial operation; it is not defined when subtracting a larger cardinal from a smaller one, nor when subtracting two equal infinite cardinals.

The generalized continuum hypothesis asserts that

\[
2^{\aleph_i} = \aleph(i+1)
\]

and is independent of the axioms of set theory. (Goedel, The consistency of the continuum hypothesis, Ann. Math. Studies, Princeton Univ. Press, 1940) The \(<a href="dbopcardinalnumber.xhtml">CardinalNumber</a> domain provides an operation to assert whether the hypothesis is to be assumed.

When the generalized continuum hypothesis is assumed, exponentiation to a transfinite power is allowed.

Three commonly encountered cardinal numbers are

\[
a = \#\mathbb{Z} \quad \text{countable infinity} \\
c = \#\mathbb{R} \quad \text{the continuum} \\
f = \#\{g: [0,1] \to \mathbb{R}\}
\]
In this domain, these values are obtained under the generalized continuum hypothesis in this way:

\[
\begin{align*}
\text{a} & := \aleph_0 \\
\text{c} & := 2^\text{a} \\
\text{f} & := 2^\text{c}
\end{align*}
\]
Complex objects are created by the \(<a href="dbcomplexcomplex.xhtml">complex</a>\) operation

The standard arithmetic operations are available.

If \(R\) is a field, you can also divide the complex objects.

Use a conversion (see \(<a href="axbook/section-2.7.xhtml">Conversion</a>\) in section 2.7) to view the last object as a fraction of complex integers.
The predefined macro \(\texttt{i}\) is defined to be \(\text{complex}(0,1)\).

You can also compute the conjugate and norm of a complex number.

The real and imag operations are provided to extract the real and imaginary parts, respectively.

The domain Complex Integer is also called the Gaussian integers. If \(R\) is the integers (or, more generally, a Euclidean Domain), you can compute greatest common divisors.
You can also compute least common multiples

You can factor Gaussian integers.

Continued fractions have been a fascinating and useful tool in mathematics for well over three hundred years. Axiom implements continued fractions for fractions of any Euclidean domain. In practice, this usually means rational numbers. In this section we demonstrate some of the operations available for manipulating both finite and infinite continued fractions.
It may be helpful if you review
<a href="db.xhtml?Stream">Stream</a> to remind yourself of some of the
operations with streams.

The <a href="db.xhtml?ContinuedFraction">ContinuedFraction</a> domain is a
field and therefore you can add, subtract, multiply, and divide the
fractions. The
<a href="dbopcontinuedfraction.xhtml">continuedFraction</a> operation
converts its fractional argument to a continued fraction.

This display is the compact form of the bulkier
<pre>
3 + 1
-----------
7 + 1
-----------
15 + 1
----------
1 + 1
----------
25 + 1
--------
1 + 1
------
7 + 1
-
4
</pre>

You can write any rational number in a similar form. The fraction will
be finite and you can always take the "numerators" to be 1. That is, any
rational number can be written as a simple, finite continued fraction of
the form
<pre>
a(1) + 1
-------------
a(2) + 1
-------------
a(3) + 1
......
a(n-1) + 1
--------
a(n)
</pre>
The \( a(i) \) are called partial quotients and the operation \(<a href="dboppartialquotients.xhtml">partialQuotients</a>\) creates a stream of them.

By considering more and more of the fraction, you get the \(<a href="dbopconvergents.xhtml">convergents</a>\). For example, the first convergent is \( a(1) \), the second is \( a(1) + 1/a(2) \) and so on.

Since this is a finite continued fraction, the last convergent is the original rational number, in reduced form. The result of \(<a href="dbopapproximants.xhtml">approximants</a>\) is always an infinite stream, though it may just repeat the "last" value.

Inverting \( c \) only changes the partial quotients of its fraction by inserting a 0 at the beginning of the list.

Do this to recover the original continued fraction from this list of partial quotients. The three argument form of the \(<a href="dbopcontinuedfraction.xhtml">continuedFraction</a>\) operation takes an element which is the whole part of the fraction, a stream of elements which are the denominators of the fraction.
The streams need not be finite for continuedFraction. Can you guess which irrational number has the following continued fraction? See the end of this section for the answer.

In 1737 Euler discovered the infinite continued fraction expansion
\[
e - 1 = \frac{1}{2 + \frac{1}{6 + \frac{1}{10 + \frac{1}{14 + \ldots}}}}
\]

We use this expansion to compute rational and floating point approximations of e. (For this and other interesting expansions, see C. D. Olds, Continued Fractions, New Mathematical Library, Random House, New York, 1963 pp.134-139).

By looking at the above expansion, we see that the whole part is 0 and the numerators are all equal to 1. This constructs the stream of denominators.

Therefore this is the continued fraction expansion for (e-1)/2.

These are the rational number convergents.
You can get rational convergents for $e$ by multiplying by 2 and adding 1.

You can also compute the floating point approximations to these convergents.

Compare this to the value of $e$ computed by the exp operation in Float.

In about 1658, Lord Brouncker established the following expansion for $\frac{4}{\pi}$.

\[
1 + \frac{1}{2 + \frac{9}{2 + \frac{25}{2 + \frac{49}{2 + \cdots}}}}
\]

Let's use this expansion to compute rational and floating point approximations for $\pi$.
As you can see, the values are converging to

```
pi = 3.14159265358979323846..., but not very quickly.
```

You need not restrict yourself to continued fractions of integers. Here is
an expansion for a quotient of Gaussian integers.

```
This is an expansion for a quotient of polynomials in one variable with
rational number coefficients.
```
To conclude this section, we give you evidence that

\[
\begin{align*}
    z &= 3 + 1 \\
    & \quad \frac{\frac{6}{3}}{\frac{1}{1}} \\
    & \quad \frac{\frac{3}{1}}{\frac{1}{1}} \\
    & \quad \frac{3}{1} \\
    & \quad 6 + 1 \\
    & \quad 3 + 1 \\
    & \quad 3 + \ldots
\end{align*}
\]

is the expansion of the square root of 11.

One can show that if an integer of the form $2^{k}+1$ is prime, then $k$ must be a power of two.

Pierre Fermat conjectured that every integer of the form $2^{2^n}+1$ is prime. Let’s look for a counterexample. First define a function:

\[
\begin{align*}
f(n:\text{NNI}):\text{INT} &= 2^{2^n}+1
\end{align*}
\]
Now try commands like:

```
<ul>
  <li>
    <input type="submit" id="p2" class="subbut"
           onclick="handleFree(['p1','p2']);"
           value="factor f(1)" />
    <div id="ansp2"></div>
  </li>
  <li>
    <input type="submit" id="p3" class="subbut"
           onclick="handleFree(['p1','p3']);"
           value="factor f(2)" />
    <div id="ansp3"></div>
  </li>
</ul>
```

until you find an integer of this form which is composite. You can also try the following command:

```
<ul>
  <li>
    <input type="submit" id="p4" class="subbut"
           onclick="handleFree(['p1','p4']);"
           value="[factor f(n) for n in 1..6]" />
    <div id="ansp4"></div>
  </li>
</ul>
```

Obviously, Fermat didn't have access to Axiom.

---

**numfactorization.xhtml**

Use the operation `<a href="dbopfactor.xhtml">factor</a>` to factor integers. It returns an object of type `<a href="db.xhtml?Factored(Integer)">Factored Integer</a>`. See `<a href="factored.xhtml">Factored</a>` for a discussion of the manipulation of factored objects.
CHAPTER 1. OVERVIEW

The operation <a href="dbopprimeq.xhtml">prime?</a> returns true or false depending on whether its argument is a prime.

The operation <a href="dbopnextprime.xhtml">nextPrime</a> returns the least prime number greater than its argument.

The operation <a href="dbopprevprime.xhtml">prevPrime</a> returns the greatest prime number less than its argument.

To compute all primes between two integers (inclusively), use the operation <a href="dbopprimes.xhtml">primes</a>.

You might sometimes want to see the factorization of an integer when it is considered a Gaussian (that is, complex) integer. See
A finite field is a finite algebraic structure where on can add, multiply, and divide under the same laws (for example, commutativity, associativity, or distributivity) as apply to the rational, real, or complex numbers. Unlike those three fields, for any finite field there exists a positive prime integer $p$, called the characteristic, such that $p \times x = 0$ for any element $x$ in the finite field. In fact, the number of elements in a finite filed is a power of the characteristic and for each prime $p$ and positive integer $n$ there exists exactly one finite field with $p^n$ elements, up to an isomorphism. (For more information about the algebraic structure and properties of finite fields, see for example, S. Lang, Algebra, Second Edition, New York, Addison-Wesley Publishing Company, Inc. 1984, ISBN 0 201 05476 6; or R. Lidl, H. Niederreiter, Finite Fields, Encyclopedia of Mathematics and Its Applications, Vol. 20, Cambridge. Cambridge Univ. Press, 1983, ISBN 0 521 30240 4)

When $n=1$, the field has $p$ elements and is called a prime field, discussed in Modular Arithmetic and Prime Fields in section 8.11.1. There are several ways of implementing extensions of finite fields, and Axiom provides quite a bit of freedom to allow you to choose the one that is best for your application. Moreover, we provide operations for converting among the different representations of extensions and different extensions of a single field. Finally, note that you usually need to package call operations from finite fields if the operations do not take as an argument an object of the field. See Package Calling and Target Types in section 2.9 for more information on package calling.
Axiom provides two kinds of floating point numbers. The domain Float (abbreviation FLOAT)
implements a model of arbitrary precisions floating point numbers. The domain
\[ \text{DoubleFloat} \]
(abbreviation \[ \text{DFLOAT} \]) is intended to make available hardware floating point arithmetic in Axiom. The actual model of floating point
\[ \text{DoubleFloat} \] that Axiom provides is system dependent. For example, on the IBM System 370, Axiom uses IBM double precision which has fourteen hexadecimal digits of precision or roughly sixteen decimal digits. Arbitrary precision floats allow the user to specify the precision at which arithmetic operations are computed. Although this is an attractive facility, it comes at a cost. Arbitrary precision floating point arithmetic typically takes twenty to two hundred times more time than hardware floating point.

For more information about Axiom’s numeric and graphic facilities see
\[ \text{Graphics} \] in section 7,\[ \text{Numeric Functions} \] in section 8.1, and \[ \text{DoubleFloat} \]
<ul>
<li>
\[ \text{Introduction to Float} \] (see \[ \text{Jenks sect} \]ion 9.27.1)
</li>
<li>
\[ \text{Conversion Functions} \] (see \[ \text{Jenks sect} \]ion 9.27.2)
</li>
<li>
\[ \text{Output Functions} \] (see \[ \text{Jenks sect} \]ion 9.27.3)
</li>
<li>
\[ \text{Determinant of a Hilbert Matrix} \] (see \[ \text{Jenks sect} \]ion 9.27.4)
</li>
</ul>
Axiom handles fractions in many different contexts and will automatically simplify fractions whenever possible. Here are some examples:

1. \( \frac{1}{4} \cdot \frac{1}{5} \)
2. \( \frac{x^2 + 1}{x - 1} \)
3. \( \frac{x^2 - 3x + 2}{x + 2} \)
4. \( f \cdot g \)

Additional Topics:
- Rational Numbers
- Quotient Fields
- Quotients over an arbitrary integral domain
The `<a href="db.xhtml?IntegerNumberTheoryFunctions">IntegerNumberTheoryFunctions</a>` package contains a variety of operations of interest to number theorists. Many of these operations deal with divisibility properties of integers (Recall that an integer a divides an integer b if there is an integer c such that b=a*c.)

The operation `<a href="dbopdivisors.xhtml">divisors</a>` returns a list of the divisors of an integer.

You can now compute the number of divisors of 144 and the sum of the divisors of 144 by counting and summing the elements of the list we just created.

```
<input type="submit" id="p1" class="subbut"
onclick="makeRequest('p1');"
value="div144:=divisors(144)" />
<div id="ansp1"><div></div></div>

<input type="submit" id="p2" class="subbut"
onclick="handleFree(['p1','p2']);"
value="#(div144)" />
<div id="ansp2"><div></div></div>

<input type="submit" id="p3" class="subbut"
onclick="handleFree(['p1','p3']);"
value="reduce(+,div144)" />
<div id="ansp3"><div></div></div>
```
Of course, you can compute the number of divisors of an integer $n$, usually denoted $d(n)$, and the sum of the divisors of an integer $n$, usually denoted $\varsigma(n)$, without ever listing the divisors of $n$.

In Axiom, you can simply call the operations

- $d(n)$ can be computed using the operation $\text{numberOfDivisors}(n)$.
- $\varsigma(n)$ can be computed using the operation $\text{sumOfDivisors}(n)$.

The key is that $d(n)$ and $\varsigma(n)$ are "multiplicative functions". This means that when $n$ and $m$ are relatively prime, that is, when $n$ and $m$ have no factors in common, then $d(nm) = d(n)d(m)$ and $\varsigma(nm) = \varsigma(n)\varsigma(m)$. Note that these functions are trivial to compute when $n$ is a prime power and are computed for general $n$ from the prime factorization of $n$. Other examples of multiplicative functions are $\varsigma_k(n)$, the sum of the $k$-th powers of the divisors of $n$ and $\phi(n)$, the number of integers between 1 and $n$ which are prime to $n$. The corresponding Axiom operations are called

- $\varsigma_k(n)$ can be computed using the operation $\text{sumOfKthPowerDivisors}(n,k)$.
- $\phi(n)$ can be computed using the operation $\text{eulerPhi}(n)$.

An interesting function is called $\mu(n)$, the Moebius mu function, defined as

$$\mu(n) =
\begin{cases}
0 & \text{if } n \text{ has a repeated prime factor (i.e. is divisible by a square)} \\
1 & \text{if } n \text{ is 1} \\
(-1)^k & \text{if } n \text{ is the product of } k \text{ distinct primes}
\end{cases}$$

The corresponding Axiom operation is
This function occurs in the following theorem:

**Theorem** (Moebius Inversion Formula):

Let $f(n)$ be a function on the positive integers and let $F(n)$ be defined by $F(n) = \sum f(n)$ over $d \mid n$ where the sum is taken over the positive divisors of $n$. Then the values of $f(n)$ can be recovered from the values of $F(n): f(n) = \sum \mu_F(n/d)$ over $d \mid n$, where the sum is taken over the positive divisors of $n$.

When $f(n) = 1$, the $F(n) = d(n)$. Thus, if you sum $\mu(d) \ast d(n/d)$ over the positive divisors of $d$ of $n$, you should always get 1.

Similarly, when $f(n) = n$, then $F(n) = \sigma(n)$. Thus, if you sum $\mu(d) \ast \sigma(n/d)$ over the positive divisors $d$ of $n$, you should always get $n$. 
The Fibonacci numbers are defined by
\[
\begin{align*}
F(1) &= 1 \\
F(2) &= 1 \\
F(n) &= F(n-1) + F(n-2) \text{ for } n = 3, 4, \ldots
\end{align*}
\]

The operation \href{dbopfibonacci.xhtml}{fibonacci} computes the
nth Fibonacci number.

Fibonacci numbers can also be expressed as sums of binomial
coefficients.
Quadratic symbols can be computed with the operations \( \text{legendre} \) and \( \text{jacobi} \). The Legendre symbol \((a/p)\) is defined for integers \(a\) and \(p\) with \(p\) an odd prime number. By definition,
\[
\begin{align*}
(a/p) &= -1 \text{ when } a \text{ is not a square } \pmod{p} \\
(a/p) &= 0 \text{ when } a \text{ is divisible by } p \\
(a/p) &= +1 \text{ when } a \text{ is a square } \pmod{p}
\end{align*}
\]
You compute \((a/p)\) via the command \texttt{legendre}(a,p)

The Jacobi symbol \((a/n)\) is the usual extension of the Legendre symbol, where \(n\) is an arbitrary integer. The most important property of the Jacobi symbol is the following: if \(K\) is a quadratic field with discriminant \(d\) and quadratic character \(\chi\), the \(\chi(n)=(d/n)\). Thus, you can use the Jacobi symbol to compute, say, the class numbers of imaginary quadratic fields from a standard class number formula. This function computes the class number of the imaginary quadratic field with discriminant \(d\).
Axiom provides many operations for manipulating arbitrary precision integers. In this section we will show some of those that come from Integer itself plus some that are implemented in other packages. More examples of integers are in the following sections: Numbers, IntegerNumberTheoryFunctions, DecimalExpansion, BinaryExpansion, HexadecimalExpansion, and RadixExpansion.

Basic Functions, Primes and Factorization, Some Number Theoretic Functions
In Axiom, integers can be as large as you like. Try the following examples.

- `<li> <input type="submit" id="p1" value="x:=factorial(200)" class="subbut" onclick="makeRequest('p1');"/>
    <div id="ansp1"><div></div></div>
</li>`

- `<li> <input type="submit" id="p2" value="y:=2^90-1" class="subbut" onclick="makeRequest('p2');"/>
    <div id="ansp2"><div></div></div>
</li>`

Of course, you can now do arithmetic as usual on these (very) large integers:

- `<li> <input type="submit" id="p3" value="x+y" class="subbut" onclick="handleFree(['p1','p2','p3']);"/>
    <div id="ansp3"><div></div></div>
</li>`

- `<li> <input type="submit" id="p4" value="x-y" class="subbut" onclick="handleFree(['p1','p2','p4']);"/>
    <div id="ansp4"><div></div></div>
</li>`

- `<li> <input type="submit" id="p5" value="x*y" class="subbut" onclick="handleFree(['p1','p2','p5']);"/>
    <div id="ansp5"><div></div></div>
</li>`
Axiom can factor integers, but numbers with small prime factors will factor more rapidly than numbers with large prime factors.

Additional topics

<table>
<thead>
<tr>
<th>Link</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Info</td>
<td>General information and examples of integers</td>
</tr>
<tr>
<td>Factorization</td>
<td>Primes and factorization</td>
</tr>
<tr>
<td>Functions</td>
<td>Number theoretic functions</td>
</tr>
<tr>
<td>Examples</td>
<td>Examples from number theory</td>
</tr>
</tbody>
</table>
Axiom provides two kinds of floating point numbers. The domain \(<a href="db.xhtml?Float">Float</a>\) (abbreviation \(<a href="db.xhtml?Float">FLOAT</a>\)) implements a model of arbitrary precisions floating point numbers. The domain \(<a href="db.xhtml?DoubleFloat">DoubleFloat</a>\) (abbreviation \(<a href="db.xhtml?DoubleFloat">DFLOAT</a>\)) is intended to make available hardware floating point arithmetic in Axiom. The actual model of floating point \(<a href="db.xhtml?DoubleFloat">DoubleFloat</a>\) that Axiom provides is system dependent. For example, on the IBM System 370, Axiom uses IBM double precision which has fourteen hexadecimal digits of precision or roughly sixteen decimal digits. Arbitrary precision floats allow the user to specify the precision at which arithmetic operations are computed. Although this is an attractive facility, it comes at a cost. Arbitrary precision floating point arithmetic typically takes twenty to two hundred times more time than hardware floating point.

By default, floating point numbers that you enter into Axiom are of type \(<a href="db.xhtml?Float">Float</a>\).

\(<li>\<input type="submit" id="p1" class="sbutton" onclick="makeRequest('p1');" value="2.71828" />\<div id="ansp1">\</div></li>\)
You must therefore tell Axiom that you want to use `DoubleFloat` values and operations. The following are some conservative guidelines for getting Axiom to use `DoubleFloat`.

To get a value of type `DoubleFloat`, use a target with "@", ...

```<ul>
<li><input type="submit" id="p2" class="subbut"
onclick="makeRequest('p2');"
value="2.71828@DoubleFloat"/>
<div id="ansp2"><div></div></div></li>
</ul>
```
a conversion,...

```<ul>
<li><input type="submit" id="p3" class="subbut"
onclick="makeRequest('p3');"
value="2.71828::DoubleFloat"/>
<div id="ansp3"><div></div></div></li>
</ul>
```
or an assignment to a declared variable. It is more efficient if you use a target rather than an explicit or implicit conversion.

```<ul>
<li><input type="submit" id="p4" class="noresult"
onclick="makeRequest('p4');"
value="eApprox:DoubleFloat:=2.71828"/>
<div id="ansp4"><div></div></div></li>
</ul>
```
You also need to declare functions that work with `DoubleFloat`.

```<ul>
<li><input type="submit" id="p5" class="noresult"
onclick="makeRequest('p5');"
value="avg:List DoubleFloat -> DoubleFloat"/>
<div id="ansp5"><div></div></div></li>
</ul>
```
```<ul>
<li><input type="submit" id="p6" class="noresult"
onclick="makeRequest('p6');"
value="avg l==(empty? l => 0::DoubleFloat; reduce(_+,l)/#l)"/>
<div id="ansp6"><div></div></div></li>
</ul>
```
```<ul>
<li><input type="submit" id="p7" class="subbut"
onclick="handleFree(['p5','p6','p7'])"
value="avg []" /></li>
</ul>
```
Use package calling for operations from <a href="db.xhtml?DoubleFloat">DoubleFloat</a> unless the arguments themselves are already of type <a href="db.xhtml?DoubleFloat">DoubleFloat</a>.

By far, the most common usage of <a href="db.xhtml?DoubleFloat">DoubleFloat</a> is for functions to be graphied. For more information about Axiom's numerical and graphical facilities, see <a href="axbook/book-contents.xhtml#chapter7">Graphics</a> in section 7, <a href="axbook/book-contents.xhtml#chapter8">Numeric Functions</a> in section 8.1, and <a href="numfloat.xhtml">Float</a>.

The usual arithmetic and elementary functions are available for <a href="db.xhtml?DoubleFloat">DoubleFloat</a>. Use

\begin{verbatim}
\texttt{)show DoubleFloat}
\end{verbatim}

to get a list of operations.
nummachinesizedintegers.xhtml

--- nummachinesizedintegers.xhtml ---

\getchunk{standard head}
\getchunk{handlefreevars}
\getchunk{axiom talker}
</script>
</head>
<body onload="resetvars();">
\getchunk{page head}
<div align="center">Machine-sized Integers</div>
<hr/>
The <a href="db.xhtml?SingleInteger">SingleInteger</a> is intended to provide support in Axiom for machine integer arithmetic. It is generally much faster than (bignum) <a href="db.xhtml?Integer">Integer</a> arithmetic but suffers from a limited range of values. Since Axiom can be implemented on top of various dialects of Lisp, the actual representation of small integers may not correspond exactly to the host machines integer representation.

You can discover the minimum and maximum values in your implementation by using <a href="dbopmin.xhtml">min</a> and <a href="dbopmax.xhtml">max</a>:

<ul>
<li>
<input type="submit" id="p1" class="subbut" onclick="makeRequest('p1');" value="min()$SingleInteger" />
<div id="ansp1"><div></div></div>
</li>
<li>
<input type="submit" id="p2" class="subbut" onclick="makeRequest('p2');" value="max()$SingleInteger" />
<div id="ansp2"><div></div></div>
</li>
</ul>
To avoid confusion with <a href="db.xhtml?Integer">Integer</a>, which is the default type for integers, you usually need to work with declared variables (see <a href="axbook/section-2.3.xhtml">Declarations</a>).

<ul>
<li>
<input type="submit" id="p3" class="subbut" onclick="makeRequest('p3');" value="a:=1234::SingleInteger" />
<div id="ansp3"><div></div></div>
</li>
</ul>
or use package calling (see <a href="axbook/section-2.9.xhtml">Package Calling and Target Types</a>).

<ul>
<li>
<input type="submit" id="p4" class="subbut" onclick="makeRequest('p4');" value="b:=1234$SingleInteger" />
<div id="ansp4"><div></div></div>
</li>
</ul>
You can add, multiply, and subtract SingleInteger objects, and ask for the greatest common divisor (gcd).

The least common multiple (lcm) is also available.

Operations mulmod, addmod, submod, and invmod are similar -- they provide arithmetic modulo a given small integer. Here is 5*6 mod 13.

To reduce a small integer modulo a prime, use positiveRemainder.

Operations And, Or, xor, and Not provide bit level operations on small integers.
Use `shift(int, numToShift)` to shift bits, where `int` is shifted left if `numToShift` is positive, right if negative.

```
(use shift(int, numToShift))
```

Many other operations are available for small integers, including many of those provided for `<a href="db.xhtml?Integer">Integer</a>`.

To see other operations use the system command

```
(use show SingleInteger)
```

numnumbertheoreticfunctions.xhtml

--- numnumbertheoreticfunctions.xhtml ---

Axiom provides several number theoretic operations for integers. More examples are in `<a href="numfunctions.xhtml">IntegerNumberTheoryFunctions</a>`.

The operation `<a href="dbopfibonacci.xhtml">fibonacci</a>` computes the
Fibonacci numbers. The algorithm has a running time $O(\log(n)^{**3})$ for argument $n$.

The operation `<a href="dboplegendre.xhtml">legendre</a>` computes the Legendre symbol for its two integer arguments where the second one is prime. If you know the second argument to be prime, use `<a href="dbopjacobi.xhtml">jacobi</a>` instead where no check is made.

The operation `<a href="dbopjacobi.xhtml">jacobi</a>` computes the Jacobi symbol for its two integer arguments. By convention, 0 is returned if the greatest common divisor of the numerator and denominator is not 1.

The operation `<a href="dbopeulerphi.xhtml">eulerPhi</a>` computes the values of Euler's $\phi$-function where $\phi(n)$ equals the number of positive integers less than or equal to $n$ that are relatively prime to the positive integer $n$.

The operation `<a href="dbopmoebiusmu.xhtml">moebiusMu</a>` computes the Moebius function.

The operation `<a href="dbopmoebiusmu.xhtml">moebiusMu</a>` computes the Moebius function.

The operation `<a href="dbopmoebiusmu.xhtml">moebiusMu</a>` computes the Moebius function.

The operation `<a href="dbopmoebiusmu.xhtml">moebiusMu</a>` computes the Moebius function.

The operation `<a href="dbopmoebiusmu.xhtml">moebiusMu</a>` computes the Moebius function.

The operation `<a href="dbopmoebiusmu.xhtml">moebiusMu</a>` computes the Moebius function.
Although they have somewhat limited utility, Axiom provides Roman numerals.

numnumericfunctions.xhtml

--- numnumericfunctions.xhtml ---

Axiom provides two basic floating point types:
Float and DoubleFloat. This section describes how to use numerical operations defined on these types and the related complex types. As we mentioned in An Overview of Axiom in chapter 1., the Float type is a software implementation of floating point numbers in which the exponent and the significand may have any number of digits. See Float for detailed information about this domain. The DoubleFloat is usually a hardware implementation of floating point numbers, corresponding to machine double precision. The types Complex Float and Complex DoubleFloat are the corresponding software implementations of complex floating point numbers. In this section the term floating point type means any of these four types. The floating point types implement the basic elementary functions. These include (where $ means Float, Float, Complex Float, Complex DoubleFloat):

- \texttt{exp}: $ \rightarrow $<br />
- \texttt{log}: $ \rightarrow $<br />
- \texttt{sin}, \texttt{cos}, \texttt{tan}, \texttt{cot}, \texttt{sec}, \texttt{csc}: $ \rightarrow $<br />
- \texttt{asin}, \texttt{acos}, \texttt{atan}, \texttt{acot}, \texttt{asec}, \texttt{acsc}: $ \rightarrow $<br />
- \texttt{sinh}, \texttt{cosh}, \texttt{tanh}, \texttt{coth}, \texttt{sech}, \texttt{csch}: $ \rightarrow $<br />
- \texttt{asinh}, \texttt{acosh}, \texttt{atanh}, \texttt{acoth}, \texttt{asech}, \texttt{acsch}: $ \rightarrow $<br />
- \texttt{pi}: () \rightarrow $<br />
- \texttt{sqrt}: $ \rightarrow $<br />
- \texttt{nthRoot}: ($, \text{Integer}) \rightarrow $<br />
- \texttt{**}: ($, \text{Fraction Integer}) \rightarrow $
CHAPTER 1. OVERVIEW

The handling of roots depends on whether the floating point type is real or complex. For the real floating point types, if a real root exists the one with the same sign as the radicand is returned; for the complex floating point types, the principal value is returned. Also, for real floating point types the inverse functions produce errors if the results are not real. This includes cases such as \( \text{asin}(1.2), \log(-3.2), \sqrt{-1,1} \). The default floating point type is \(<a href="numfloat.xhtml">Float</a>\), just use normal decimal notation.

\[
\begin{align*}
\text{exp}(3.1) & \\
\text{exp}(3.1+4.5\times\%i) & \\
\text{exp}(3.1:\text{DFLOAT}+4.5:\text{DFLOAT}\times\%i) & \\
\end{align*}
\]

To evaluate functions using \(<a href="nummachinefloats.xhtml">DoubleFloat</a>\) or \(<a href="dbcomplexdoublefloat.xhtml">Complex DoubleFloat</a>\), a declaration or conversion is required.

\[
\begin{align*}
\text{exp}(3.1;4.5;\%i) & \\
\text{exp}(3.1::\text{DFLOAT}+4.5::\text{DFLOAT}\times\%i) & \\
\end{align*}
\]

A number of special functions are provided by the package \(<a href="db.xhtml">DoubleFloatSpecialFunctions</a>\) for the machine precision floating point types. The special functions provided are listed below, where \( F \) stands for the types \(<a href="numfloat.xhtml">Float</a>\) or \(<a href="dbcomplexfloat.xhtml">Complex Float</a>\). The real versions of the functions yield an error if the result is not real.

\[
\begin{align*}
\text{Gamma}(z) & = \int_0^{\infty} t^{z-1} \exp(-t) \, dt
\end{align*}
\]
\[\text{Beta}(u,v) = \frac{\Gamma(u) \Gamma(v)}{\Gamma(u+v)}\]

\[\log\Gamma(z)\] is the natural logarithm of \(\Gamma(z)\). This can often be computed even if \(\Gamma(z)\) cannot.

\[\psi(z) = \frac{\Gamma'(z)}{\Gamma(z)}\]

\[\psi^{(n)}(z)\] is the \(n\)-th derivative of \(\psi(z)\).

\[J_v(z)\] is the Bessel function of the first kind, \(J_v(z)\). This function satisfies the differential equation \[z^2w''(z) + zw'(z) - (z^2 - v^2)w(z) = 0\]

\[Y_v(z)\] is the Bessel function of the second kind, \(Y_v(z)\). This function satisfies the same differential equation as \(J_v(z)\). The implementation simply uses the relation \[Y_v(z) = (J_v(z) \cos(v\pi) - J_{-v}(z))/\sin(v\pi)\]

\[I_v(z)\] is the modified Bessel function of the first kind, \(I_v(z)\). This function satisfies the differential equation \[z^2w''(z) + zw'(z) - (z^2 + v^2)w(z) = 0\]

\[K_v(z)\] is the modified Bessel function of the second kind, \(K_v(z)\). This function satisfies the same differential equation as \(I_v(z)\). The implementation simply uses the relation \[K_v(z) = \frac{\pi}{2} \left( \frac{I_v(z) - I_{-v}(z)}{(2 \sin(v\pi))} \right)\]

\[\text{airyAi}(z)\] is the Airy function \(Ai(z)\). This function satisfies the differential equation \[z^2w''(z) - zw'(z) = 0\]
$w''(z)-zw(z)=0$

The implementation simply uses the relation

\[ A_i(-z)=\frac{1}{3}\sqrt{z}\left(J\left(-\frac{1}{3},\frac{2}{3}z^{3/2}\right)+J\left(\frac{1}{3},\frac{2}{3}z^{3/2}\right)\right) \]

\[ \text{airyBi}(z) \text{ is the Airy function } B_i(z). \text{ This function satisfies the same differential equation as airyAi.} \]

The implementation simply uses the relation

\[ B_i(-z)=\frac{1}{3}\sqrt{3z}\left(J\left(-\frac{1}{3},\frac{2}{3}z^{3/2}\right)-J\left(\frac{1}{3},\frac{2}{3}z^{3/2}\right)\right) \]

\[ \text{hypergeometric0F1}(c,z) \text{ is the hypergeometric function } \, _0F_1(c;z). \text{ The above special functions are defined only for small floating point types. If you give } \text{float} \text{ arguments, they are converted to } \text{double} \text{ by Axiom.} \]

\[ \Gamma(0.5)^2 \]

\[ \text{besselI}(2.1+1.1\text{i},2.1*1.1) \]

A number of additional operations may be used to compute numerical values. These are special polynomial functions that can be evaluated for values in any commutative ring \( R \), and in particular for values in any floating-point type. The following operations are provided by the package \text{OrthogonalPolynomialFunctions}:

\[ \text{chebyshevT}(n,z) \text{ is the } n\text{th Chebyshev polynomial of the first kind, } T_n(z). \text{ These are defined by} \]

\[ (1-tz)/(1-2tz+t^2)=\sum{T_n(z)\cdot t^n, n=0..} \]

\[ \text{chebyshevU}(n,z) \text{ is the } n\text{th Chebyshev polynomial of the second kind, } U_n(z). \text{ These are defined by} \]

\[ 1/(1-2tz+t^2)=\sum{U_n(z)\cdot t^n, n=0..} \]

\[ \text{hermiteH}(n,z) \]

\[ (\text{NonNegativeInteger},R) \rightarrow R \]
hermiteH(n,z) is the nth Hermite polynomial, \( H[n](z) \). These are defined by

\[
\exp(2tz-t^2) = \sum_{n=0}^{\infty} H[n](z)t^n/n!, \quad n=0,\ldots
\]

laguerreL(n,z) is the nth Laguerre polynomial, \( L[n](z) \). These are defined by

\[
\frac{\exp(-t/z/(1-t))}{1-t} = \sum_{n=0}^{\infty} L[n](z)t^n/n!, \quad n=0,\ldots
\]

labuerreL(m,n,2) is the associated Laguerre polynomial, \( L[m][n](z) \). This is the nth derivative of \( L[n](z) \).

legendreP(n,z) is the nth Legendre polynomial, \( P[n](z) \). These are defined by

\[
\frac{1}{\sqrt{1-2zt+t^2}} = \sum_{n=0}^{\infty} P[n](z)t^n, \quad n=0,\ldots
\]

These operations require non-negative integers for the indices, but otherwise the argument can be given as desired.

The expression \( \text{chebyshevT}(n,z) \) evaluates to the nth Chebyshev polynomial of the first kind.

The expression \( \text{chebyshevT}(3,5.0+6.0*%i) \) and \( \text{chebyshevT}(3,5.0::\text{DoubleFloat}) \)
The expression \( \text{chebyshevU}(n,z) \) evaluates to the \( n \)th Chebyshev polynomial of the second kind.

The expression \( \text{hermiteH}(n,z) \) evaluates to the \( n \)th Hermite polynomial.

The expression \( \text{laguerreL}(n,z) \) evaluates to the \( n \)th Laguerre polynomial.
The expression \( \text{legendreP}(n, z) \) evaluates to the \( n \)th Legendre polynomial.

Finally, three number-theoretic polynomial operations may be evaluated. The following operations are provided by the package \(<a href="db.xhtml?NumberTheoreticPolynomialFunctions">NumberTheoreticPolynomialFunctions</a>\).

**Bernoulli Polynomial**

- \( \text{bernoulliB}(n, z) \) is the \( n \)th Bernoulli polynomial, \( B[n](z) \). These are defined by
  \[
  t \times \exp(z \times t)/(\exp(t) - 1) = \sum B[n](z) \times t^n/n! \text{ for } n=0,\ldots.
  \]

**Euler Polynomial**

- \( \text{eulerE}(n, z) \) is the \( n \)th Euler polynomial, \( E[n](z) \). These are defined by
  \[
  2 \times \exp(z \times t)/(\exp(t) + 1) = \sum E[n](z) \times t^n/n! \text{ for } n=0,\ldots.
  \]

**Cyclotomic Polynomial**

- \( \text{cyclotomic}(n, z) \) is the \( n \)th cyclotomic polynomial \( \phi(n,z) \). This is the polynomial whose roots are precisely the primitive \( n \)th roots of unity. This polynomial has degree given by the Euler totient function \( \phi(n) \).
The expression bernoulliB(n,z) evaluates to the nth Bernoulli polynomial.

\[ \text{bernoulliB}(3,z) \]

\[ \text{bernoulliB}(3,0.7+0.4*%i) \]

The expression eulerE(n,z) evaluates to the nth Euler polynomial.

\[ \text{eulerE}(3,z) \]

\[ \text{eulerE}(3,0.7+0.4*%i) \]

The expression cyclotomic(n,z) evaluates to the nth cyclotomic polynomial.

\[ \text{cyclotomic}(3,z) \]

\[ \text{cyclotomic}(3,(-1.0+0.0*%i)^{(2/3)}) \]

Drawing complex functions in Axiom is presently somewhat awkward compared to drawing real functions. It is necessary to use the draw operations that operate on functions rather than expressions.

This is the complex exponential function. When this is displayed in color, the height is the value of the real part of the function and the color is
the imaginary part. Red indicates large negative imaginary values, green indicates imaginary values near zero and blue/violet indicates large positive imaginary values.

This is the complex arctangent function. Again, the height is the real part of the function value but here the color indicates the function value's phase. The position of the branch cuts are clearly visible and one can see that the function is real only for a real argument.

This is the complex Gamma function.

This shows the real Beta function near the origin.

This is the Bessel function J(\(\alpha\),x) for index \(\alpha\) in the range -6..4 and argument x in the range 2..14.

This is the modified Bessel function I(\(\alpha\),x) evaluated for various real values of the index \(\alpha\) and fixed argument x=5.
This is similar to the last example except the index alpha takes on complex values in a 6x6 rectangle centered on the origin.

The Octonions, also called the Cayley-Dixon algebra, defined over a commutative ring are an eight-dimensional non-associative algebra. Their construction from quaternions is similar to the construction of quaternions from complex numbers (see <a href="numquaternions.xhtml">Quaternion</a>). As <a href="db.xhtml?Octonion">Octonion</a> creates an eight-dimensional algebra, you have to give eight components to construct an octonion.
Or you can use two quaternions to create an octonion.

```html
<input type="submit" id="p3" class="subbut" onclick="makeRequest('p3');"
value="oci3:=octon(quatern(-7,-12,3,-10),quatern(5,6,9,0))" />
</li>
</ul>

You can easily demonstrate the non-associativity of multiplication.

```html
<input type="submit" id="p4" class="subbut"
onclick="handleFree(['p1','p2','p3','p4']);"
value="(oci1*oci2)*oci3-oci1*(oci2*oci3)" />
</li>
</ul>

As with the quaternions, we have a real part, the imaginary parts i, j, k, and four additional imaginary parts E, I, J, and K. These parts correspond to the canonical basis (1,i,j,k,E,I,J,K). For each basis element there is a component operation to extract the coefficient of the basis element for a given octonion.

```html
<input type="submit" id="p5" class="subbut"
onclick="handleFree(['p1','p5']);"
value="[real oci1, imagi oci1, imagj oci1, imagk oci1, imagE oci1, imagI oci1, imagJ oci1, imagK oci1]" />
</li>
</ul>

A basis with respect to the quaternions is given by (1,E). However, you might ask, what then are the commuting rules? To answer this, we create some generic elements. We do this in Axim by simply changing the ground ring from

```html
<a href="db.xhtml?Integer">Integer</a> to
<a href="dbpolynomialinteger.xhtml">Polynomial Integer</a>.
</li>
</ul>

```html
<input type="submit" id="p6" class="subbut" onclick="makeRequest('p6');"
value="q:Quaternion Polynomial Integer:=quatern(q1,qi,qj,qk)" />
</li>
</ul>

```html
<input type="submit" id="p7" class="subbut" onclick="makeRequest('p7');"
value="E:Octonion Polynomial Integer:=octon(0,0,0,0,1,0,0,0)" />
</li>
</ul>

Note that quaternions are automatically converted to octonions in the obvious way.

```html
<input type="submit" id="p8" class="subbut"
Finally, we check that the \(\text{norm}\), defined as the sum of the squares of the coefficients, is a multiplicative map.

Since the result is 0, the norm is multiplicative.

Issue the system command
to display the list of operations defined by Octonion.

numotherbases.xhtml

— numotherbases.xhtml —

It is possible to expand numbers in general bases. Here we expand 111 in base 5. This means

\[
2 1 0 2 1 - \\
10 +10 +10 = 4*5 +2*5 +5
\]

You can expand fractions to form repeating expansions.
For bases from 11 to 36 the letters A through Z are used.

For bases greater than 36, the ragits are separated by blanks.

The \texttt{RadixExpansion} type provides operations to obtain the individual ragits. Here is a rational number in base 8.

The operation \texttt{wholeRagits} returns a list of the ragits for the integral part of the number.
The operations `<a href="dboprefixragits.xhtml">prefixRagits</a>` and `<a href="dbopycleragits.xhtml">cycleRagits</a>` returns lists of the initial and repeating ragist in the fractional part of the number.

You can construct any radix expansion by giving the whole, prefix, and cycle parts. The declaration is necessary to let Axiom know the base of the ragits.

If there is no repeating part, then the list `[0]` should be used.

Of course, it's possible to recover the fraction representation:

```plaintext
<input type="submit" id="p12" class="subbut" onclick="handleFree(['p10','p12']);"
    value="f0:=prefixRagits a" />
</li>
</ul>
<input type="submit" id="p13" class="subbut" onclick="handleFree(['p10','p13']);"
    value="f1:=cycleRagits a" />
</li>
</ul>

You can construct any radix expansion by giving the whole, prefix, and cycle parts. The declaration is necessary to let Axiom know the base of the ragits.

If there is no repeating part, then the list `[0]` should be used.

Of course, it's possible to recover the fraction representation:

```plaintext
<input type="submit" id="p14" class="subbut" onclick="handleFree(['p11','p12','p13','p14']);"
    value="u:RadixExpansion(8):=wholeRadix(w)+fractRadix(f0,f1)" />
</li>
</ul>

If you are not interested in the repeating nature of the expansion, an infinite stream of ragits can be obtained using

```plaintext
<input type="submit" id="p15" class="subbut" onclick="makeRequest('p15');"
    value="v:RadixExpansion(12):=fractRadix([1,2,3,11],[0])" />
</li>
</ul>
```

Of course, it's possible to recover the fraction representation:

```plaintext
<input type="submit" id="p16" class="subbut" onclick="handleFree(['p14','p16']);"
    value="fractRagits(u)" />
</li>
</ul>
```

Of course, it's possible to recover the fraction representation:

```plaintext
<input type="submit" id="p17" class="subbut" onclick="handleFree(['p10','p17']);"
CHAPTER 1. OVERVIEW

Issue the system command

to display the full list of operations defined by RadixExpansion. More examples of expansions are available in DecimalExpansion, BinaryExpansion, and HexadecimalExpansion.

numpartialfractions.xhtml

A partial fraction is a decomposition of a quotient into a sum of quotients where the denominators of the summand are powers of primes. (Most people first encounter partial fractions when they are learning integral calculus. For a technical discussion of partial fractions see, for example, Lang’s Algebra.) For example, the rational number 1/6 is decomposed into 1/2-1/3.

You can compute partial fractions of quotients of objects from domains belonging to the category EuclideanDomain. For example, Integer, Complex Integer, and UnivariatePolynomial all belong to EuclideanDomain. In the examples following, we demonstrate how to decompose quotients of each of these kinds of objects into partial fractions.
It is necessary that we know how to factor the denominator when we want to compute a partial fraction. Although the interpreter can often do this automatically, it may be necessary for you to include a call to \(<a href="dbopfactor.xhtml">factor</a>\). In these examples, it is not necessary to factor the denominators explicitly. The main operation for computing partial fractions is called \(<a href="dboppartialfraction.xhtml">partialFraction</a>\) and we use this to compute a decomposition of \(1/10\). The first argument top \(<a href="dboppartialfraction.xhtml">partialFraction</a>\) is the numerator of the quotient and the second argument is the factored denominator.

\(<ul><li><input type="submit" id="p1" class="subbut" onclick="makeRequest('p1');" value="partialFraction(1,factorial 10)" /><div id="ansp1"><div></div></div></li></ul>\)

Since the denominators are powers of primes, it may be possible to expand the numerators further with respect to those primes. Use the operation \(<a href="dboppadicfraction.xhtml">padicFraction</a>\) to do this.

\(<ul><li><input type="submit" id="p2" class="subbut" onclick="makeRequest('p2');" value="padicFraction(f)" /> <div id="ansp2"><div></div></div></li></ul>\)

The operation \(<a href="dbopcompactfraction.xhtml">compactFraction</a>\) returns an expanded fraction into the usual form. The compacted version is used internally for computational efficiency.

\(<ul><li><input type="submit" id="p3" class="subbut" onclick="handleFree(['p2','p3']);" value="compactFraction(f)" /> <div id="ansp3"><div></div></div></li></ul>\)

You can add, subtract, multiply, and divide partial fractions. In addition, you can extract the parts of the decomposition. \(<a href="dbopnumberoffractionalterms.xhtml">numberOfFractionalTerms</a>\) computes the number of terms in the fractional part. This does not include the whole part of the fraction, which you get by calling \(<a href="dbopwholepart.xhtml">wholePart</a>\). In this example, the whole part is 0.

\(<ul><li><input type="submit" id="p4" class="subbut" onclick="handleFree(['p2','p4']);" value="numberOfFractionalTerms(f)" /> <div id="ansp4"><div></div></div></li></ul>\)
The operation \( \text{nthFractionalTerm} \) returns the individual terms in the decomposition. Notice that the object returned is a partial fraction itself.

\( \text{firstNumer} \) and \( \text{firstDenom} \) extract the numerator and denominator of the first term of the fraction.

Given two gaussian integers, you can decompose their quotient into a partial fraction.

To convert back to a quotient, simply use the conversion.

To conclude this section, we compute the decomposition of

\[
\frac{1}{(x + 1)(x + 2) (a + 3) (x + 4)}
\]

The polynomials in this object have type \( \text{UnivariatePolynomial} \).

We use the \( \text{primeFactor} \) operation to create the denominator in factored form directly.

These are the compact and expanded partial fractions for the quotient.
Also see <a href="db.xhtml?FullPartialFractionExpansion">FullPartialFractionExpansion</a> for examples of factor-free conversion of quotients to full partial fractions.

Issue the system command

Also see <a href="db.xhtml?PartialFraction">PartialFraction</a>.

umproblems.xhtml

One can show that if an integer of the form $2^{**k}-1$ is prime then $k$ must be prime.

<br/>
Suppose that \( k = m \cdot n \) is a non-trivial factorization. Then

\[
2^m = 1 \pmod{2^m - 1}
\]

\[
2^{m \cdot n} = 1 \pmod{2^m - 1}
\]

so \( 2^m - 1 \) is a non-trivial factor of \( 2^k - 1 \)

---

Problem

Find the smallest prime \( p \) such that \( 2^p - 1 \) is not prime

Answer

First, define a function:

\[
f(n: \text{NNI}): \text{INT} = 2^n - 1
\]

You can try factoring \( f(p) \) as \( p \) ranges through the set of primes. For example,

\[
factors f(7)
\]

This gets tedious after a while, so let’s use Axiom’s stream facility. A stream is essentially an infinite sequence. First, we create a stream consisting of the positive integers:

\[ints := [n \text{ for } n \text{ in } 1..]
\]

Now, we create a stream consisting of the primes:

\[
primes := [x \text{ for } x \text{ in } \text{ints } | \prime? x]
\]

Here is the 25th prime:
Next, create the stream of numbers of the form $2^p - 1$ with $p$ prime:

\[
\text{numbers} := [f(n) \text{ for } n \text{ in primes}]
\]

Finally, form the stream of factorizations of the elements of numbers:

\[
factors := [\text{factor } n \text{ for } n \text{ in numbers}]
\]

You can see that the fifth number in the stream ($2047 = 23 \times 89$) is the first one that has a non-trivial factorization. Since $2^{11} = 2048$, the solution to the problem is 11.

Here is another way to see that 2047 is the first number in the stream that is composite:

\[
\text{nums} := [x \text{ for } x \text{ in numbers | not prime? } x]
\]

You can see that the fifth number in the stream (2047 = 23 * 89) is the first one that has a non-trivial factorization. Since $2^{11} = 2048$, the solution to the problem is 11.

Here is another way to see that 2047 is the first number in the stream that is composite:

\[
\text{nums} := [x \text{ for } x \text{ in numbers | not prime? } x]
\]

Now have Axiom factor the numbers on this list:

\[
factors := [\text{factor } n \text{ for } n \text{ in numbers}]
\]

You can see that the fifth number in the stream ($2047 = 23 \times 89$) is the first one that has a non-trivial factorization. Since $2^{11} = 2048$, the solution to the problem is 11.

Here is another way to see that 2047 is the first number in the stream that is composite:

\[
\text{nums} := [x \text{ for } x \text{ in numbers | not prime? } x]
\]

You can see that the fifth number in the stream (2047 = 23 * 89) is the first one that has a non-trivial factorization. Since $2^{11} = 2048$, the solution to the problem is 11.

Here is another way to see that 2047 is the first number in the stream that is composite:

\[
\text{nums} := [x \text{ for } x \text{ in numbers | not prime? } x]
\]

Now have Axiom factor the numbers on this list:

\[
factors := [\text{factor } n \text{ for } n \text{ in numbers}]
\]

You can see that the fifth number in the stream ($2047 = 23 \times 89$) is the first one that has a non-trivial factorization. Since $2^{11} = 2048$, the solution to the problem is 11.
You can see that 41 is the smallest positive integer \( n \) such that \( n^2 - n + 41 \) is not prime.

---

numquaternions.xhtml

---

The domain constructor \(<a href="db.xhtml?Quaternion">Quaternion</a>\) implements quaternions over commutative rings.

The basic operation for creating quaternions is \(<a href="dbopquatern.xhtml">quatern</a>\). This is a quaternion over the rational numbers.

The four arguments are the real part, the \( i \) imaginary part, the \( j \) imaginary part, and the \( k \) imaginary part, respectively.

Because \( q \) is over the rationals (and nonzero), you can invert it.
The usual arithmetic (ring) operations are available.

In general, multiplication is not commutative.

There are no predefined constants for the imaginary i, j, and k parts, but you can easily define them.
These satisfy the normal identities.

\[ [i*i, j*j, k*k, i*j, j*k, k*i, q*i] \]

The norm is the quaternion times its conjugate.

\[ \text{norm } q \]

\[ \text{c:=conjugate } q \]

\[ q*c \]

For information on related topics, see <a href="db.xhtml?Complex">Complex</a> and <a href="db.xhtml?Octonion">Octonion</a>. You can also issue the system command

\[ )show Quaternion \]

to display the full list of operations defined by <a href="db.xhtml?Quaternion">Quaternion</a>.\getchunk{page foot}
The `<a href="db.xhtml?Fraction">Fraction</a>` domain implements quotients. The elements must belong to a domain of category `<a href="db.xhtml?IntegralDomain">IntegralDomain</a>`: multiplication must be commutative and the product of two non-zero elements must not be zero. This allows you to make fractions of most things you would think of, but don't expect to create a fraction of two matrices. The abbreviation for `<a href="db.xhtml?Fraction">Fraction</a>` is `<a href="db.xhtml?Fraction">FRAC</a>`. Use `<a href="dbopdivide.xhtml">/</a>` to create a fraction.

- `<input type="submit" id="p1" class="subbut" onclick="makeRequest('p1');" value="a:=11/12" />
  <div id="ansp1"><div></div></div>`
- `<input type="submit" id="p2" class="subbut" onclick="makeRequest('p2');" value="b:=23/24" />
  <div id="ansp2"><div></div></div>`

The standard arithmetic operations are available.

- `<input type="submit" id="p3" class="subbut" onclick="handleFree(['p1','p2','p3']);" value="3-a*b^2+a+b/a" />
  <div id="ansp3"><div></div></div>`

Extract the numerator and denominator by using `<a href="dbopnumer.xhtml">numer</a>` and `<a href="dbopdenom.xhtml">denom</a>`, respectively.

- `<input type="submit" id="p4" class="subbut" onclick="handleFree(['p1','p4']);" value="numer(a)" />
  <div id="ansp4"><div></div></div>`
- `<input type="submit" id="p5" class="subbut"`
CHAPTER 1. OVERVIEW

Operations like \(<a href="dbopmax.xhtml">max</a>\), \(<a href="dbopmin.xhtml">min</a>\), \(<a href="dbopnegativeq.xhtml">negative</a>\), \(<a href="dboppositiveq.xhtml">positive</a>\), and \(<a href="dbopzeroq.xhtml">zero</a>\) are all available if they are provided for the numerators and denominators. See \(<a href="numintegers.xhtml?Integer">Integer</a>\) for examples.

Don't expect a useful answer from \(<a href="dbopfactor.xhtml">factor</a>\), \(<a href="dbopgcd.xhtml">gcd</a>\), or \(<a href="dboplcm.xhtml">lcm</a>\) if you apply them to fractions.

Since all non-zero fractions are invertible, these operations have trivial definitions.

Use \(<a href="dbopmap.xhtml">map</a>\) to apply \(<a href="dbopfactor.xhtml">factor</a>\) to the numerator and denominator, which is probably what you mean.

Other forms of fractions are available, Use \(<a href="dbopcontinuedfraction.xhtml">continuedFraction</a>\) to create a continued fraction.
Use <a href="dboppartialfraction.xhtml">partialFraction</a> to create a partial fraction. 
See <a href="numcontinuedfractions.xhtml">continuedFraction</a> and <a href="numpartialfractions.xhtml">PartialFraction</a> for additional information and examples.

Use conversion to create alternative views of fractions with objects moved in and out of the numerator and denominator.

Conversion is discussed in detail in <a href="axbook/section-2.7.xhtml">Conversion</a>.

---

numrationalnumbers.xhtml

--- numrationalnumbers.xhtml ---
Like integers, rational numbers can be arbitrarily large. For example:

- \( 
\frac{61657^{10}}{999983^{12}} 
\)

Rational numbers will not be converted to decimals unless you explicitly ask Axiom to do so. To convert a rational number to a decimal, use the function \(<a href="dbopnumeric.xhtml">numeric</a>\). Here's an example:

- \( \frac{104348}{33215} \)
- \( numeric \( x \) \)

You can find the numerator and denominator of rational numbers using the functions \(<a href="dbopnumer.xhtml">numer</a>\) and \(<a href="dbopdenom.xhtml">denom</a>\), respectively.

To factor the numerator and denominator of a fraction, use the following command:

- \( factor(numer(x))/factor(denom(x)) \)
All rational numbers have repeating binary expansions. Operations to access the individual bits of a binary expansion can be obtained by converting the value to RadixExpansion(2). More examples of expansions are available in DecimalExpansion, HexadecimalExpansion, and RadixExpansion.

The expansion of a rational number is returned by the binary operation.

Arithmetic is exact.

The period of the expansion can be short or long...
or very long

These numbers are bona fide algebraic objects.

All rationals have repeating decimal expansions. Operations to access the individual digits of a decimal expansion can be obtained by converting the value to $\text{RadixExpansion}(10)$. 

---

All rationals have repeating decimal expansions. Operations to access the individual digits of a decimal expansion can be obtained by converting the value to $\text{RadixExpansion}(10)$.
The operation `<a href="dbopdecimal.xhtml">decimal</a>` is used to create this expansion of type `<a href="db.xhtml?DecimalExpansion">DecimalExpansion</a>`.

```
<input type="submit" id="p1" class="subbut" onclick="makeRequest('p1');"
    value="r:=decimal(22/7)" />
<div id="ansp1">div</div>
```

Arithmetic is exact.

```
<input type="submit" id="p2" class="subbut"
    onclick="handleFree(['p1','p2']);"
    value="r+decimal(6/7)" />
<div id="ansp2">div</div>
```

The period of the expansion can be short or long...

```
<input type="submit" id="p3" class="subbut"
    onclick="makeRequest('p3');"
    value="[decimal(1/i) for i in 350..354]" />
<div id="ansp3">div</div>
```

or very long

```
<input type="submit" id="p4" class="subbut"
    onclick="makeRequest('p4');"
    value="decimal(1/2049)" />
<div id="ansp4">div</div>
```

These numbers are bona fide algebraic objects.

```
<input type="submit" id="p5" class="subbut"
    onclick="makeRequest('p5');"
    value="p:=decimal(1/4)*x^2+decimal(2/3)*x+decimal(4/9)" />
<div id="ansp5">div</div>
```

```
<input type="submit" id="p6" class="subbut"
    onclick="handleFree(['p5','p6']);"
    value="q:=differentiate(p,x)" />
<div id="ansp6">div</div>
```

```
<input type="submit" id="p7" class="subbut"
    onclick="handleFree(['p5','p6','p7']);"
    value="g:=gcd(p,q)" />
<div id="ansp7">div</div>
```
 CHAPTER 1. OVERVIEW

More examples of expansions are available in
<a href="numrepeatingbinaryexpansions.xhtml">BinaryExpansion</a>,
<a href="numrepeatinghexexpansions.xhtml">HexadecimalExpansion</a>, and
<a href="db.xhtml?RadixExpansion">RadixExpansion</a>. Issue the system command
<ul>
  <li><input type="submit" id="p8" class="subbut"
    onclick="showcall('p8');"
    value="show RadixExpansion"/>
    <div id="ansp8"><div></div></div></li>
</ul>
to display the full list of operations defined by
<a href="db.xhtml?RadixExpansion">RadixExpansion</a>.

---

numrepeatinghexexpansions.xhtml

---

All rationals have repeating hexadecimals expansions. The operation
<a href="dbophex.xhtml">hex</a> returns these expansions of type
<a href="db.xhtml?HexadecimalExpansion">HexadecimalExpansion</a>. Operations to access the individual numerals of a hexadecimal expansion can be obtained by converting the value to
<a href="db.xhtml?RadixExpansion">RadixExpansion(16)</a>. More examples of expansions are available in
<a href="numrepeatingdecimals.xhtml">DecimalExpansion</a>,
<a href="numrepeatingbinaryexpansions.xhtml">BinaryExpansion</a>, and
<a href="db.xhtml?RadixExpansion">RadixExpansion</a>. This is a hexadecimal expansion of a rational number.

<ul>
  <li><input type="submit" id="p1" class="subbut" onclick="makeRequest('p1');"
    value="r:=hex(22/7)" />
    <div id="ansp1"><div></div></div></li>
</ul>
Arithmetic is exact.

The period of the expansion can be short or long...

or very long.

These numbers are bona fide algebraic objects.

Issue the system command
to display the full list of operations defined by
<a href="db.xhtml?HexadecimalExpansion">HexadecimalExpansion</a>.

\getchunk{page foot}

numromannumerals.xhtml

— numromannumerals.xhtml —

\getchunk{standard head}
  <script type="text/javascript">
  \getchunk{handlefreevars}
  \getchunk{axiom talker}
  </script>
  </head>
  <body onload="resetvars();">
  \getchunk{page head}
  <div align="center">Roman Numerals</div>
  <hr/>

The Roman numeral package was added to Axiom in MCMLXXXVI for use in
denoting higher order derivatives.

For example, let f be a symbolic operator.

<ul>
  <li>
    <input type="submit" id="p1" class="subbut" onclick="makeRequest('p1');"
      value="f := operator 'f" />
    <div id="ansp1"><div></div></div>
  </li>
</ul>

This is the seventh derivative of f with respect to x

<ul>
  <li>
    <input type="submit" id="p2" class="subbut" onclick="makeRequest('p2');"
      value="D(f, x, 7)" />
    <div id="ansp2"><div></div></div>
  </li>
</ul>

You can have integers printed as Roman numerals by declaring variables
to be of type

<a href="db.xhtml?RomanNumeral">RomanNumeral</a>
(abbreviation <a href="db.xhtml?RomanNumeral">ROMAN</a>).

<ul>
  <li>
    <input type="submit" id="p3" class="subbut" onclick="makeRequest('p3');"
      value="a := roman(1978-1965)" />
    <div id="ansp3"><div></div></div>
  </li>
</ul>

This package now has a small but devoted group of followers that claim
this domain has shown its efficacy in many other contexts. They claim
that Roman numerals are every bit as useful as ordinary integers. In a sense, they are correct, because Roman numerals form a ring and you can therefore construct polynomials with Roman numeral coefficients, matrices over Roman numerals, etc.

Was Fibonacci Italian or ROMAN?

You can also construct fractions with Roman numeral numerators and denominators, as this matrix Hilberticus illustrates.

Note that the inverse of the matrix has integral entries.

Unfortunately, the spoil-sports say that the fun stops when the numbers get big -- mostly because the Romans didn't establish conventions about representing very large numbers.
CHAPTER 1. OVERVIEW

You work it out!

Issue the system command

to display the full list of operations defined by 

ocwmit18085.xhtml

ocwmit18085.xhtml —

18.085 Mathematical Methods for Engineers I Course Notes

These are course notes based on the M.I.T. Open Courseware lectures by Gilbert Strang.
In applied mathematics we have 2 basic tasks:
<ul>
<li>Find the equations</li>
<li>Solve the equations</li>
</ul>
<h4>Positive Definite Matrices</h4>
Certain matrices occur frequently in applied math. These three matrices (K, T, and M) are canonical examples.
We have 3 3x3 matrices,
<pre>
K:Matrix(Integer):=[[2,-1,0],[-1,2,-1],[0,-1,2]]
</pre>
These matrices are similar and can be generalized to square matrices of order N, with n x n elements. All of these matrices have the same element along the diagonal. T (aka Top) differs from K in the first row. B (aka Both) differs from K in the first and last row. These represent different boundary conditions in the problem.
We can create K(n), T(n) and B(n) with the following commands:
k(n) ==
M := diagonalMatrix([2 for i in 1..n])
for i in 1..n-1 repeat M(i,i+1):=-1
for i in 1..n-1 repeat M(i+1,i):=-1
M::SquareMatrix(n,Fraction(Integer))
</pre>

<pre>
t(n) ==
M:=k(n)
N:=M::Matrix(Fraction(Integer))
qsetelt!(N,1,1,1)
N::SquareMatrix(n,Fraction(Integer))
</pre>

<pre>
b(n) ==
M:=k(n)
N:=M::Matrix(Fraction(Integer))
qsetelt!(N,1,1,1)
qsetelt!(N,n,n,1)
N::SquareMatrix(n,Fraction(Integer))
</pre>

K:=k(n) has a few key properties:
<ul>
<li> K is symmetric, that is K=K^T</li>
<li> K might be nonsingular, that is, it is invertible</li>
<li> K has a non-zero determinant</li>
<li> K is banded (main diagonal and neighbors)</li>
<li> K is tri-diagonal (main diagonal and nearest neighbors)</li>
<li> K is extremely sparse</li>
<li> K has constant diagonals, (shift invariant, time invariant)</li>
<li> K is Toeplitz (constant diagonal, shows up in filters)</li>
<li> K is good for Fourier analysis</li>
</ul>

The inverse of T:
If we look at the inverse of the T matrix we see:
<pre>
T^-1
+3  2  1+
|    |    |
|2  2  1|
|    |    |
+1  1  1+

Type: Matrix Fraction Integer
</pre>

Notice that these are all integers because the determinant of this matrix is 1
<pre>
determinant T
1

Type: Fraction Integer
We can check that this matrix is the inverse of T.

When computing the inverse the row pattern $[-1, 2, -1]$ is a "second difference". The first column of the inverse matrix is $[3, 2, 1]$ which is linear. When we take the second difference of a linear object we should get 0. Thus,

$$[-1, 2, -1] \cdot \text{MATRIX(INT)} \cdot [3, 2, 1]$$

0

Type: Matrix Integer

The third column of the T matrix is linear and constant. If we take the second difference of that we also find it is zero:

$$[-1, 2, -1] \cdot \text{MATRIX(INT)} \cdot [1, 1, 1]$$

0

Type: Matrix Integer

and the diagonal element of the unit matrix must be one. So the second difference of the second column is:

$$[-1, 2, -1] \cdot \text{MATRIX(INT)} \cdot [2, 2, 1]$$

1

Type: Matrix Integer

So these simple checks show that we're getting the correct row and column values for the identity matrix by multiplying T times its inverse.

The inverse of B

If we look for the inverse of the B matrix we can observe that the rows sum to zero which implies that it is not invertible. Thus it is singular.

K and T are positive definite. B is only positive semi-definite.

If we can find a vector that it takes to zero, that is if we can solve for x, y, z in:

$$
\begin{bmatrix}
+ 1 & - 1 & 0 & + & + & + & + & 0 & + \\
|   & |   & |   & |   & |   & |   & |   \\
- 1 & 2 & - 1 & | & y & = & | & 0 & | \\
|   & |   & |   & |   & |   & |   & |   \\
+ 0 & - 1 & 1 & + & + & z & + & + & 0 & + \\
\end{bmatrix}
$$

The constant vector $[1, 1, 1]$ solves this equation. When
the rows sum to zero we are adding each row by a constant
and thus we add each row times the constant one and we
get zeros. If the matrix takes some vector to zero it
cannot have an inverse since if

\[
B \mathbf{x} = 0
\]

and \( \mathbf{x} \) is not zero. If \( B \) had an inverse only \( \mathbf{x} = 0 \) would
solve the equation. Since \( \mathbf{x} = 1 \) solves the equation \( B \) has
no inverse. The vector \( \mathbf{x} \) is in the nullspace of \( B \). In
fact any constant vector, e.g. \([3 \ 3 \ 3]\) is in the nullspace.
Thus the nullspace of \( B \) is \( c \mathbf{x} \) for any constant \( c \).

When doing matrix multiplication one way to think about the
work is to consider the problem by columns. Thus in the
multiplication

\[
\begin{bmatrix}
+1 & -1 & 0 & + & x & + & 0 & + \\
| & | & | & | & | & | & | \\
|-1 & 2 & -1 & | & y & | & 0 & | \\
| & | & | & | & | & | & | \\
+0 & -1 & 1 & + & z & + & 0 & +
\end{bmatrix}
\]

we can think about this as

\[
\mathbf{x} \text{(first column)} + \mathbf{y} \text{(second column)} + \mathbf{z} \text{(third column)}.
\]

and for the constant vector \([1 \ 1 \ 1]\) this means that we
just need to sum the columns.

Alternatively this can be computed by thinking of the
multiplication as

\[
\begin{align*}
(\text{first row}) \times (\text{vector}) \\
(\text{second row}) \times (\text{vector}) \\
(\text{third row}) \times (\text{vector})
\end{align*}
\]

The inverse of \( K \)

Now we consider the \( K \) matrix we see the inverse

\[
K
\]

\[
+2 & -1 & 0 + \\
| & | & | \\
-1 & 2 & -1 | \\
| & | & | \\
+0 & -1 & 2 +
\]

Type: SquareMatrix(3,Fraction Integer)

\[
\text{kinv:=}K^{-1}
\]

\[
+3 & 1 & 1+
\]
We can take the determinant of \( k \)

\[
\text{determinant } K
\]

\[
4
\]

Type: \text{Fraction Integer}

Thus there is a constant \( 1/4 \) which can be factored out

\[
4 \cdot \text{kinv}
\]

Type: \text{SquareMatrix(3,Fraction Integer)}

Notice that the inverse is a symmetric matrix but not tri-diagonal.

The inverse is not a sparse matrix so much more computation would be involved when using the inverse.

In order to solve the system

\[
K \ u = f
\]

by elimination which implies multiplying and subtracting rows.

\[
K \ u = f \implies U \ u = f
\]

For the 2x2 case we see:

\[
+2 \ 1+ + f1 +
\]

\[
+2 \ -1+ +x+ +f1+ | +x+ | = | +x+ | = 1
\]

\[
+2 +y+ +f2+ | y+ | = | f2+f1|
\]

By multiplying row1 by \( 1/2 \) and adding it to row2 we create an upper triangular matrix \( U \). Since we chose \( K(1,1) \), the number 2
For K 2x2 above is symmetric and invertible (since the pivots are all non-zero).

For the K 3x3 case the pivots are 2, 3/2, and 4/3. (The next pivots would be 5/4, 6/5, etc. for larger matrices).

For the T 3x3 case the pivots are 1, 1, and 1.

For the B 3x3 case the third pivot would be zero.

Generalizing the matrix pivot operations

For the 2x2 case we see construct an elimination matrix E which we can use to pre-multiply by K to give us the upper triangular matrix U

\[ E K = U \]

In detail we see

\[
\begin{bmatrix}
+1 & 0 & +2 & -1 \\
| & +2 & -1 & | \\
|1 | & | & | = | 3 \\
| -1 | & +1 & 2 & |0 & - \\
+2 & + & + & 2 \\
\end{bmatrix}
\]

We wish to rewrite this as

\[ K = L U \]

The big 4 solve operations in Linear Algebra

- Elimination
- Gram-Schmidt Orthogonalization
- Eigenvalues
- Singular Value Decomposition

Each of these operations is described by a factorization of K. Elimination is written

\[ K = L U \]

where L is lower triangular and U is upper triangular.

Thus we need a matrix L which when multiplied by U gives K. The required matrix is the inverse of the E matrix above since

\[ E K = U \]
Given the matrix operations above we had

\[ E K = U \]

\[ E = \begin{bmatrix} +1 & 0 \\ -1 & 1 \\ +2 & 0 \end{bmatrix} = L \]

and the inverse of \( E \) is the same matrix with a minus sign in the second row, thus:

\[ E = \begin{bmatrix} +1 & 0 \\ -1 & 1 \\ +2 & 0 \end{bmatrix} = L \]

Making the matrices symmetric

We would like to preserve the symmetry property which we can do with a further decomposition of \( LU \) as follows:

\[ L U = L D U' \]

So now we have 3 matrices; \( L \) is the lower triangular, \( D \) is symmetric and contains the pivots, and \( U' \) is upper triangular and is the transpose of the lower. So the real form we have is

\[ T \]

\[ L D L \]
This result will always be symmetric. We can check this by taking its transpose. If we get the same matrix we must have a symmetric matrix. So the transpose of

\[
( \begin{array}{ccc} 
L & D & L \\
T & T & T \\
T & T & T \\
\end{array} )
= \begin{array}{ccc} 
L & D & L \\
L & D & L \\
L & D & L \\
\end{array}
\]

Positive Definite Matrices

There are several ways to recognize a positive definite matrix. First, it must be symmetric. The "positive" aspect comes from the pivots, all of which must be positive. Note that \( T \) is also positive definite. \( B \) is positive semi-definite because one of the pivots is zero. So

- positive definite \( \Rightarrow \) all pivots \( > 0 \)
- positive semi-definite \( \Rightarrow \) all pivots \( \geq 0 \)

When all the pivots are positive then all the eigenvalues are positive.

So a positive definite matrix \( K \) and any non-zero vector \( X \)

\[
X^T K X > 0
\]

\( X \) transpose is just a row and \( X \) is just a column.

ocwmit18085lecture2.xhtml

— ocwmit18085lecture2.xhtml —

\getchunk{standard head}

\getchunk{page head}

One-dimensional Applications: \( A = \) Difference Matrix

\getchunk{page head}

\getchunk{page foot}
outputfunctions.xhtml

--- outputfunctions.xhtml ---

```html
<script type="text/javascript">
  getchunk{handlefreevars}
  axiom talker
</script>

A number of operations exist for specifying how numbers of type `Float` are to be displayed. By default, spaces are inserted every ten digits in the output for readability. (Not that you cannot include spaces in the input form of a floating point number, though you can use underscores.)

Output spacing can be modified with the `outputSpacing` operation. This inserts no spaces and then displays the value of `x`.

```html
<ul>
  <li>
    <input type="submit" id="p1" class="subbut" onclick="makeRequest('p1');"
      value="outputSpacing 0; x:=sqrt 0.2" />
    <div id="ansp1"></div>
  </li>
</ul>

Issue this to have the spaces inserted every 5 digits.

```html
<ul>
  <li>
    <input type="submit" id="p2" class="subbut" onclick="handleFree(['p1','p2']);"
      value="outputSpacing 5; x" />
    <div id="ansp2"></div>
  </li>
</ul>
```
By default, the system displays floats in either fixed format or scientific format, depending on the magnitude of the number.

A particular format may be requested with the operations \texttt{outputFloating} and \texttt{outputFixed}.

Additionally, you can ask for \( n \) digits to be displayed after the decimal point.

The \texttt{outputGeneral} function resets the output printing to the default behavior.
pagelist.xhtml

--- pagelist.xhtml ---

<head>
<body>
pagelist not implemented
</body>
</head>

pagematrix.xhtml

--- pagematrix.xhtml ---

<head>
<body>
pagematrix not implemented
</body>
</head>

pageonedimensionalarray.xhtml

--- pageonedimensionalarray.xhtml ---

<head>
<body>
pageonedimensionalarray not implemented
</body>
</head>

pageset.xhtml

--- pageset.xhtml ---

<head>
<body>
CHAPTER 1. OVERVIEW

pagetable.xhtml

— pagetable.xhtml —

pagepermanent.xhtml

— pagepermanent.xhtml —

pagesquarematrix.xhtml

— pagesquarematrix.xhtml —

pagetwodimensionalarray.xhtml

— pagetwodimensionalarray.xhtml —
The \texttt{TwoDimensionalArray} is used for storing data in a two-dimensional data structure indexed by row and column. Such an array is a homogeneous data structure in that all the entries of the array must belong to the same Axiom domain (although see \texttt{The Any Domain}). Each array has a fixed number of rows and columns specified by the user and arrays are not extensible. In Axiom, the indexing of two-dimensional arrays is one-based. This means that both the "first" row of an array and the "first" column of an array are given the index 1. Thus, the entry in the upper left corner of an array is in position \((1,1)\).

The operation \texttt{new} creates an array with a specified number of rows and columns and fills the components of that array with a specified entry. The arguments of this operation specify the number of rows, the number of columns, and the entry. This creates a five-by-four array of integers, all of whose entries are zero.

The entries of this array can be set to other integers using the operation \texttt{setelt}.

Issue this to set the element in the upper left corner of this array to 17.

Now the first element of the array is 17.

The entries of this array can be set to other integers using the operation \texttt{setelt}.

Issue this to set the element in the upper left corner of this array to 17.

Now the first element of the array is 17.
 Likewise, elements of an array are extracted using the operation \(<a href="dbopelt.xhtml">elt</a>\).

\[
\begin{enumerate}
  \item \text{input type="submit" id="p4" class="subbut" onclick="handleFree(['p1','p2','p4']);" value="elt(arr,1,1)" /}
  \text{<div id="ansp4">\text{</div>}</div>
\end{enumerate}
\]

Another way to use these two operations is as follows. This sets the element in position (3,2) of the array to 15.

\[
\begin{enumerate}
  \item \text{input type="submit" id="p5" class="subbut" onclick="handleFree(['p1','p2','p5']);" value="arr(3,2):=15" /}
  \text{<div id="ansp5">\text{</div>}</div>
\end{enumerate}
\]

This extracts the element in position (3,2) of the array.

\[
\begin{enumerate}
  \item \text{input type="submit" id="p6" class="subbut" onclick="handleFree(['p1','p2','p5','p6']);" value="arr(3,2)" /}
  \text{<div id="ansp6">\text{</div>}</div>
\end{enumerate}
\]

The operation \(<a href="dbopelt.xhtml">elt</a>\) and \(<a href="dbopsetelt.xhtml">setelt</a>\) come equipped with an error check which verifies that the indices are in the proper ranges. For example, the above array has five rows and four columns, so if you ask for the entry in position (6,2) with \(arr(6,2)\) Axiom displays an error message. If there is no need for an error check, you can call the operations \(<a href="dbopqelt.xhtml">qelt</a>\) and \(<a href="dbopqseteltbang.xhtml">qsetelt!</a>\) which provide the same functionality but without the error check. Typically, these operations are called in well-tested programs.

The operations \(<a href="dboprow.xhtml">row</a>\) and \(<a href="dbopcolumn.xhtml">column</a>\) extract rows and columns, respectively, and return objects of \(<a href="db.xhtml?OneDimensionalArray">OneDimensionalArray</a>\) with the same underlying element type.
You can determine the dimensions of an array by calling the operations \( \text{nrows} \) and \( \text{ncols} \), which return the number of rows and columns, respectively.

To apply an operation to every element of an array, use \( \text{map} \). This creates a new array. This expression negates every element.

This creates an array where all the elements are doubled.

To change the array destructively, use \( \text{map!} \) instead of \( \text{map} \).

If you need to make a copy of an array, use \( \text{copy} \).
Use <a href="dbopmember.xhtml">member?</a> to see if a given element is in an array.

To see how many times an element appears in an array, use <a href="dbopcount.xhtml">count</a>.

For more information about the operations available for
For more information on related topics, see
<a href="pagematrix.xhtml">Matrix</a> and
<a href="pageonedimensionalarray.xhtml">OneDimensionalArray</a>.
CHAPTER 1. OVERVIEW

These operations can also be used to combine polynomials. Try the following:

As you can see from the above examples, the variables are ordered by defaults

That is, z is the main variable, then y and so on in reverse alphabetical order. You can redefine this ordering (for display purposes) with the <a href="dbopsetvariableorder.xhtml">setVariableOrder</a>. For example, the following makes a the main variable, then b, and so on:
Now compare the way polynomials are displayed:

To return to the system's default ordering, use a href="dbopresetvariableorder.xhtml">resetVariableOrder</a>.

Polynomial coefficients can be pulled out using the function a href="dbopcoefficient.xhtml">coefficient</a>. For example:

will give you the coefficient of x**2 in the polynomial q. Try these commands:
Coefficients of monomials can be obtained as follows:

This will return the coefficient of \(x^2z\) in the polynomial \(q^2\). Also,

will return the coefficient of \(x^2y^2\) in the polynomial \(r(x,y)\).
Polynomials with integer coefficients can be factored.

\[ v := (4x^3 + 2y^2 + 1)(12x^5 - (1/2)x^3 + 12) \]

\[ \text{factor } v \]
Also, Axiom can factor polynomials with rational number coefficients

\begin{verbatim}
<input type="submit" id="p3" class="subbut"
onclick="makeRequest('p3');"
value="w:=(4*x^3+(2/3)*x^2+1)*(12*x^5-(1/2)*x^3+12)" />
<div id="ansp3">\div</div></div>
</li>
</ul>

\begin{verbatim}
<input type="submit" id="p4" class="subbut"
onclick="handleFree(['p3','p4']);"
value="factor w" />
<div id="ansp4">\div</div>
</li>
</ul>
\getchunk{page foot}

---

polyfactorization2.xhtml

--- polyfactorization2.xhtml ---

\getchunk{standard head}
\getchunk{handlefreevars}
\getchunk{axiom talker}
\getchunk{axiom talker}
</script>
</head>
<body onload="resetvars();">
\getchunk{page head}
<div align="center">Finite Field Coefficients</div>
<hr/>
Polynomials with coefficients in a finite field can also be factored.
<ul>
<li>
<input type="submit" id="p1" class="subbut"
onclick="makeRequest('p1');"
value="u:=3*x^4+2*x^2+15*x+18" />
<div id="ansp1">\div</div>
</li>
</ul>
The integers mod p, where p is prime, and extensions of these fields.
<ul>
<li>
<input type="submit" id="p2" class="subbut"
onclick="handleFree(['p1','p2']);"
value="factor u" />
<div id="ansp2">\div</div>
</li>
</ul>
Convert this to have coefficients in the finite field with 19^3 elements. See <a href="axbook/section-8.11.xhtml">FiniteFields</a> for more information about finite fields.

<p id="ansp3"><div></div></p>

Note that the second argument to factor can be a list of algebraic extensions to factor over.

Here, aa and bb are symbolic roots of polynomials.

Polynomials with coefficients in simple algebraic extensions of the rational numbers can be factored.
This factors $x^2+3$ over the integers.

Factor the same polynomial over the field obtained by adjoining $aa$ to the rational numbers.

Factor $x^6+108$ over the same field.

Factor again over the field obtained by adjoining both $aa$ and $bb$ to the rational numbers.
Since fractions of polynomials form a field, every element (other than zero) divides any other, so there is no useful notion of irreducible factors. Thus the `<a href="dbopfactor.xhtml">factor</a>` operation is not very useful for fractions of polynomials.

Instead, there is a specific operation `<a href="dbopfactorfraction.xhtml">factorFraction</a>` that separately factors the numerator and denominator and returns a fraction of the factored results.

You can also use `<a href="dbopmap.xhtml">map</a>`. This expression applies the `<a href="dbopfactor.xhtml">factor</a>` operation to the numerator and denominator.
You can compute the greatest common divisor of two polynomials using the function `<a href="dbopgcd.xhtml">gcd</a>`. Here’s an example:

```
<ul>
  <li>
    <input type="submit" id="p1" class="subbut"
        onclick="makeRequest('p1');"
        value="p:=3*x^8+2*x^7+6*x^2+7*x+2" />
    <div id="ansp1"><div></div></div>
  </li>
  <li>
    <input type="submit" id="p2" class="subbut"
        onclick="makeRequest('p2');"
        value="q:=2*x^13+9*x^7+2*x^6+10*x+5" />
    <div id="ansp2"><div></div></div>
  </li>
  <li>
    <input type="submit" id="p3" class="subbut"
        onclick="handleFree(['p1','p2','p3']);"
        value="gcd(p,q)" />
    <div id="ansp3"><div></div></div>
  </li>
</ul>
```

You could also see that p and q have a factor in common by using the function `<a href="dbopresultant.xhtml">resultant</a>`:

```
<ul>
  <li>
    <input type="submit" id="p4" class="subbut"
        onclick="handleFree(['p1','p2','p4']);"
        value="resultant(p,q,x)" />
    <div id="ansp4"><div></div></div>
  </li>
</ul>
```

The resultant of two polynomials vanishes precisely when they have a factor in common. (In the example above we specified the variable with which we wanted to compute the resultant because the polynomials could have involved variables other than x.)
polynomialpage.xhtml

--- polynomialpage.xhtml ---

\getchunk{standard head}
\getchunk{page head}
<div align="center">Polynomials</div>
<hr />
<table>
<tr>
<td>
<a href="polybasicfunctions.xhtml">Basic Functions</a>
</td>
<td>
Create and manipulate polynomials
</td>
</tr>
<tr>
<td>
<a href="polysubstitutions.xhtml">Substitutions</a>
</td>
<td>
Evaluate Polynomials
</td>
</tr>
<tr>
<td>
<a href="polyfactorization.xhtml">Factorization</a>
</td>
<td>
Factor in different contexts
</td>
</tr>
<tr>
<td>
<a href="polygcdandfriends.xhtml">GCD and Friends</a>
</td>
<td>
Greatest Common Divisors, Resultants, and Discriminants
</td>
</tr>
<tr>
<td>
<a href="polyroots.xhtml">Roots</a>
</td>
<td>
Work with and solve for roots
</td>
</tr>
<table>
<thead>
<tr>
<th></th>
<th>&lt;a href=&quot;polyspecificatypes.xhtml&quot;&gt;Specific Types&lt;/a&gt;</th>
<th>More specific information</th>
</tr>
</thead>
</table>

---

polyroots.xhtml

— polyroots.xhtml —

---

<table>
<thead>
<tr>
<th></th>
<th>Using a Single Root of a Polynomial</th>
<th>Working with a single root of a polynomial</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Using All Roots of a Polynomial</td>
<td>Working with all the roots of a polynomial</td>
</tr>
<tr>
<td></td>
<td>Solution of a Single Polynomial Equation</td>
<td>Finding the roots of one polynomial</td>
</tr>
</tbody>
</table>
Use \texttt{rootOf(p,x)} to get a symbolic root of a polynomial. The call \texttt{rootOf(p,x)} returns a root of \( p(x) \).

This creates an algebraic number \( a \), which is a root of the polynomial returned in symbolic form.

\begin{itemize}
\item \texttt{\texttt{aa:=rootOf(a^4+1,a)}}
\end{itemize}

To find the algebraic relation that defines \( a \), use
\begin{itemize}
\item \texttt{\texttt{definingPolynomial aa}}
\end{itemize}
You can use a in any further expression, including a nested
\(<a href="dboprootof.xhtml">rootOf</a>\).

\(<ul>
\(\langle li\rangle\)
\(<input type="submit" id="p3" class="subbut"
onclick="handleFree(['p1','p3']);"
value="bb:=rootOf(b^2-aa-1,b)"/>
\(<div id="ansp3">\langle/\div\rangle\)</li> \(<\;/ul\>)

Higher powers of the roots are automatically reduced during calculations.

\(<ul>
\(\langle li\rangle\)
\(<input type="submit" id="p4" class="subbut"
onclick="handleFree(['p1','p3','p4']);"
value="g:=aa+bb"/>
\(<div id="ansp4">\langle/\div\rangle\)</li> \(<\;/ul\>)

The operation \(<a href="dbopzeroof.xhtml">zeroOf</a>\) is similar to \(<a href="dboprootof.xhtml">rootOf</a>\), except that it may express the root using radicals in some cases.

\(<ul>
\(\langle li\rangle\)
\(<input type="submit" id="p6" class="subbut"
onclick="makeRequest('p6');"
value="rootOf(c^2+c+1,c)"/>
\(<div id="ansp6">\langle/\div\rangle\)</li> \(<\;/ul\>)

\(<ul>
\(\langle li\rangle\)
\(<input type="submit" id="p7" class="subbut"
onclick="makeRequest('p7');"
value="zeroOf(d^2+d+1,d)"/>
\(<div id="ansp7">\langle/\div\rangle\)</li> \(<\;/ul\>)

\(<ul>
\(\langle li\rangle\)
\(<input type="submit" id="p8" class="subbut"
onclick="makeRequest('p8');"
value="rootOf(e^5-2,e)"/>
\(<div id="ansp8">\langle/\div\rangle\)</li> \(<\;/ul\>)

\(<ul>
\(\langle li\rangle\)
\(<input type="submit" id="p9" class="subbut"
onclick="makeRequest('p9');"
value="zeroOf(f^5-2,f)"/>
\(<div id="ansp9">\langle/\div\rangle\)</li> \(<\;/ul\>)
Compute all the roots of \( x^4 + 1 \).

- \( \text{rootsOf}(x^4 + 1) \)

As a side effect, the variables \( %x_0 \), \( %x_1 \), and \( %x_2 \) are bound to the first three roots of \( x^4 + 1 \).

Although they all satisfy \( x^4 + 1 = 0 \), \( %x_0 \), \( %x_1 \), and \( %x_2 \) are different algebraic numbers. To find the algebraic relation that defines each of them, use \(<a href="dbopdefiningpolynomial.xhtml">definingPolynomial</a>\).
Corresponding to the pair of operations \( \text{rootOf} \) and \( \text{zeroOf} \) in Solution of a Single Polynomial Equation, there is an operation \( \text{zerosOf} \) that, like \( \text{rootsOf} \), computes all the roots of a given polynomial, but which expresses some of them in terms of radicals.

As you see, only one implicit algebraic number was created (%y1), and its defining equation is this. The other three roots are expressed in radicals.
Solution of a Single Polynomial Equation

Axiom can solve polynomial equations producing either approximate or exact solutions. Exact solutions are either members of the ground field or can be presented symbolically as roots of irreducible polynomials.

This returns one rational root along with an irreducible polynomial describing the other solutions

If you want solutions expressed in terms of radicals you would use this instead.

The \texttt{solve} command always returns a value but \texttt{radicalSolve} returns only the solutions that it is able to express in terms of radicals.

If the polynomial equation has rational coefficients you can ask for
approximations to its real roots by calling solve with a second argument that specifies the "precision" epsilon. This means that each approximation will be within plus or minus epsilon of the actual result.

Notice that the type of second argument controls the type of the result.

If you give a floating point precision you get a floating point result. If you give the precision as a ration number you get a rational result.

If you want approximate complex results you should use the command \texttt{complexSolve} that takes the same precision argument epsilon.

Each approximation will be within plus or minus epsilon of the actual result in each of the real and imaginary parts.

Note that if you omit the = from the first argument Axiom generates an equation by equating the first argument to zero. Also, when only one variable is present in the equation, you do not need to specify the variable to be solved for, that is, you can omit the second argument.

Axiom can also solve equations involving rational functions. Solutions where the denominator vanishes are discarded.
Given a system of equations of rational functions with exact coefficients

\[
p_1(x_1,\ldots,x_n) \\
\ldots \\
p_m(x_1,\ldots,x_n)
\]

Axiom can find numeric or symbolic solutions. The system is first split into irreducible components, then for each component, a triangular system of equations is found that reduces the problem to sequential solutions of univariate polynomials resulting from substitution of partial solutions from the previous stage.

\[
q_1(x_1,\ldots,x_n) \\
\ldots \\
q_m(x_n)
\]

Symbolic solutions can be presented using "implicit" algebraic numbers defined as roots of irreducible polynomials or in terms of radicals. Axiom can also find approximations to the real or complex roots of a system of polynomial equations to any user specified accuracy.

The operation [a href="dbopsolve.xhtml">solve</a> for systems is used in a way similar to [a href="dbopsolve.xhtml">solve</a> for single equations. Instead of a polynomial equation, one has to give a list of equations and instead of a single variable to solve for, a list of variables. For solutions of single equations see [a href="axbook/section-8.5.xhtml#subsec-8.5.2">Solution of a Single Polynomial Equation</a>.
Use the operation `<a href="dbopsolve.xhtml">solve</a>` if you want implicitly presented solutions.

Use `<a href="dbopradialsolve.xhtml">radicalSolve</a>` if you want your solutions expressed in terms of radicals.

To get numeric solutions you only need to give the list of equations and the precision desired. The list of variables would be redundant information since there can be no parameters for the numerical solver.

If the precision is expressed as a floating point number you get results expressed as floats.

To get complex numeric solutions, use the operation `<a href="dbopcomplexsolve.xhtml">complexSolve</a>`, which takes the same arguments as in the real case.

It is also possible to solve systems of equations in rational functions over the rational numbers. Note that `[x=0.0, a=0.0]` is not returned as
a solution since the denominator vanishes there.

When solving equations with denominators, all solutions where the denominator vanishes are discarded.

—— polyspecifictypes.xhtml ——

polyspecifictypes.xhtml

The Specific Polynomial Types

Polynomial

The general type

UnivariatePolynomial

One variable polynomials
<table>
<thead>
<tr>
<th>MultivariatePolynomial</th>
<th>Multiple variable polynomials, recursive structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>DistributedMultivariatePolynomial</td>
<td>Multiple variable polynomials, non-recursive structure</td>
</tr>
</tbody>
</table>

The domain constructor <a href="db.xhtml?Polynomial">Polynomial</a> (abbreviation: <a href="db.xhtml?Polynomial">POLY</a>) provides polynomials with an arbitrary number of unspecified variables. It is used to create the default polynomial domains in Axiom. Here the coefficients are integers.

```html
<input type="submit" id="p1" class="subbut"
onclick="makeRequest('p1');"
value="x+1" />
<div id="ansp1"><div></div></div>
```
Here the coefficients have type `<a href="db.xhtml?Float">Float</a>`.

And here we have a polynomial in two variables with coefficients which have type `<a href="dbfractioninteger.xhtml">Fraction Integer</a>`

The representation of objects of domains created by `<a href="db.xhtml?Polynomial">Polynomial</a>` is that of recursive univariate polynomials. (The term univariate means "one variable". The term multivariate means "possibly more than one variable".) This recursive structure is sometimes obvious from the display of a polynomial.

In this example, you see that the polynomial is stored as a polynomial in y with coefficients that are polynomials in x with integer coefficients. In fact, you really don't need to worry about the representation unless you are working on an advanced application where it is critical. The polynomial types created from `<a href="db.xhtml?DistributedMultivariatePolynomial">DistributedMultivariatePolynomial</a>` and `<a href="db.xhtml?XDistributedPolynomial">XDistributedPolynomial</a>` (discussed in `<a href="axbook/section-9.16.xhtml">"DistributedMultivariatePolynomial"</a>" are stored and displayed in a non-recursive manner. You see a "flat" display of the above polynomial by converting to one of those types.
We will demonstrate many of the polynomial facilities by using two polynomials with integer coefficients. By default, the interpreter expands polynomial expressions, even if they are written in a factored format.

```
<input type="submit" id="p6" class="subbut"
   onclick="makeRequest('p6');"
   value="p:=(y-1)^2*x*z" />
<div id="ansp6">\(p\)</div></div>
</li>
</ul>
See <a href="axbook/section-9.22.xhtml">Factored</a> to see how to create objects in factored form directly.

```
<input type="submit" id="p7" class="subbut"
   onclick="makeRequest('p7');"
   value="q:=(y-1)*x*(z+5)" />
<div id="ansp7">\(q\)</div></div>
</li>
</ul>
The fully factored form can be recovered by using

```
<input type="submit" id="p8" class="subbut"
   onclick="handleFree(['p7','p8']);"
   value="factor(q)" />
<div id="ansp8">\(\text{factor}(q)\)</div></div>
</li>
</ul>
This is the same name used for the operation to factor integer. Such reuse of names is called <a href="#p36465">overloading</a> and makes it much easier to think of solving problems in general ways. Axiom facilities for factoring polynomials created with <a href="db.xhtml?Polynomial">Polynomial</a> are currently restricted to the integer and rational number coefficients cases. There are more complete facilities for factoring univariate polynomials (see <a href="axbook/section-8.2.xhtml">Polynomial Factorization</a>)

The standard arithmetic operations are available for polynomials.

```
<input type="submit" id="p9" class="subbut"
   onclick="handleFree(['p6','p7','p9']);"
   value="p-q^2" />
<div id="ansp9">\(p-q^2\)</div></div>
</li>
</ul>
The operation <a href="dbopgcd.xhtml">gcd</a> is used to compute the greatest common divisor of two polynomials.
In the case of \( p \) and \( q \), the \( \gcd \) is obvious from their definitions. We factor the \( \gcd \) to show this relationship better.

The least common multiple is computed by using \( \text{lcm} \)

Use \( \text{content} \) to compute the greatest common divisor of the coefficients of the polynomial.

Many of the operations on polynomials require you to specify a variable. For example, \( \text{resultant} \) requires you to give the variable in which the polynomials should be expressed. This computes the resultant of the values of \( p \) and \( q \), considering them as polynomials in the variable \( z \). They do not share a root when thought of as polynomials in \( z \).

This value is 0 because as polynomials in \( x \) the polynomials have a common root.
CHAPTER 1. OVERVIEW

The data type used for the variables created by Polynomial is Symbol. As mentioned above, the representation used by Polynomial is recursive and so there is a main variable for nonconstant polynomials. The operation makeVariable returns this variable. The return type is actually a union of Symbol and "failed".

The latter branch of the union is used if the polynomial has no variables, that is, is a constant.

The complete list of variables actually used in a particular polynomial is returned by variables. For constant polynomials, this list is empty.

The degree operation returns the degree of a polynomial in a specific variable.
If you give a list of variables for the second argument, a list of the degrees in those variables is returned.

The minimum degree of a variable in a polynomial is computed using \( \text{minimumDegree} \).

The total degree of a polynomial is returned by \( \text{totalDegree} \).

It is often convenient to think of a polynomial as a leading monomial plus the remaining terms, using the operation \( \text{leadingMonomial} \).
The <a href="reductum.xhtml">reductum</a> operation returns a polynomial consisting of the sum of the monomials after the first.

These have the obvious relationship that the original polynomial is equal to the leading monomial plus the reductum.

The value returned by <a href="leadingmonomial.xhtml">leadingMonomial</a> includes the coefficient of that term. This is extracted by using <a href="leadingcoefficient.xhtml">leadingCoefficient</a> on the original polynomial.

The operation <a href="eval.xhtml">eval</a> is used to substitute a value for a variable in a polynomial.

This value may be another variable, a constant or a polynomial.
Actually, all the things being substituted are just polynomials, some more trivial than others.

Derivatives are computed using the <a href="dbopd.xhtml">D</a> operation.

The first argument is the polynomial and the second is the variable.

Even if the polynomial has only one variable, you must specify it.

Integration of polynomials is similar and the <a href="dbopintegrate.xhtml">integrate</a> operation is used.

Integration requires that the coefficients support division. Consequently, Axiom converts polynomials over the integers to polynomials over the rational numbers before integrating them.
It is not possible, in general, to divide two polynomials. In our example using polynomials over the integers, the operation \[\text{monicDivide}\] divides a polynomial by a monic polynomial (that is, a polynomial with leading coefficient equal to 1). The result is a record of the quotient and remainder of the division. You must specify the variable in which to express the polynomial.

The selectors of the components of the record are quotient and remainder. Issue this to extract the remainder:

Now that we can extract the components, we can demonstrate the relationship among them and the arguments to our original expression

If the \(/\) operator is used with polynomials, a fraction object is created. In this example, the result is an object of type \(\text{Fraction Polynomial Integer}\).
If you use rational numbers as polynomial coefficients, the resulting object is of type Polynomial Fraction Integer.

This can be converted to a fraction of polynomials and back again, if required.

To convert the coefficients to floating point, map the numeric operation on the coefficients of the polynomial.

For more information on related topics, see UnivariatePolynomial, MultivariatePolynomial, and DistributedMultivariatePolynomial.

You can also issue the system command to display the full list of operations defined by Polynomial.
CHAPTER 1. OVERVIEW

polyspecificotypes2.xhtml

— polyspecificotypes2.xhtml —

The domain constructor
<a href="db.xhtml?UnivariatePolynomial">UnivariatePolynomial</a> creates domains of univariate polynomials in a specified variable. For example, the domain UP(a1,POLY FRAC INT) provides polynomials in the single variable a1 whose coefficients are general polynomials with rational number coefficients.

Restriction:
Axiom does not allow you to create types where
<a href="db.xhtml?UnivariatePolynomial">UnivariatePolynomial</a> is contained in the coefficient type of
<a href="db.xhtml?Polynomial">Polynomial</a>. Therefore, UP(x,POLY INT) is legal but POLY UP(x,INT) is not.

UP(x,INT) is the domain of polynomials in the single variable x with integer coefficients.

Restriction:
Axiom does not allow you to create types where
<a href="db.xhtml?UnivariatePolynomial">UnivariatePolynomial</a> is contained in the coefficient type of
<a href="db.xhtml?Polynomial">Polynomial</a>. Therefore, UP(x,POLY INT) is legal but POLY UP(x,INT) is not.

UP(x,INT) is the domain of polynomials in the single variable x with integer coefficients.

1. 
<input type="submit" id="p1" class="subbut"
onclick="makeRequest('p1');"
value="(p,q):UP(x,INT)" />
<div id="ansp1"></div>

2. 
<input type="submit" id="p2" class="subbut"
onclick="makeRequest('p2');"
value="p:=(3*x-1)^2*2*(2*x+8)" />
<div id="ansp2"></div>

3. 
<input type="submit" id="p3" class="subbut"
onclick="makeRequest('p3');"
value="q:=(1-6*x+9*x^2)^2" />
<div id="ansp3"></div>
The usual arithmetic operations are available for univariate polynomials.

The operation \(<a href="dbopleadingcoefficient.xhtml">leadingCoefficient</a>\) extracts the coefficient of the term of highest degree.

The operation \(<a href="dbopdegree.xhtml">degree</a>\) returns the degree of the polynomial. Since the polynomial has only one variable, the variable is not supplied to operations like \(<a href="dbopdegree.xhtml">degree</a>\).

The reductum of the polynomial, the polynomial obtained by subtracting the term of highest order, is returned by \(<a href="dbopreductum.xhtml">reductum</a>\).

The operation \(<a href="dbopgcd.xhtml">gcd</a>\) computes the greatest common divisor of two polynomials.
The operation `<a href="dboplcm.xhtml">lcm</a>` computes the least common multiple.

```html
<ul>
  <li>
    <input type="submit" id="p9" class="subbut"
      onclick="handleFree(['p1','p2','p3','p9']);"
      value="lcm(p,q)" />
    <div id="ansp9"></div></li>
</ul>
```

The operation `<a href="dbopresultant.xhtml">resultant</a>` computes the resultant of two univariate polynomials. In the case of p and q, the resultant is 0 because they share a common root.

```html
<ul>
  <li>
    <input type="submit" id="p10" class="subbut"
      onclick="handleFree(['p1','p2','p3','p10']);"
      value="resultant(p,q)" />
    <div id="ansp10"></div></li>
</ul>
```

To compute the derivative of a univariate polynomial with respect to its variable, use `<a href="dbopd.xhtml">D</a>`.

```html
<ul>
  <li>
    <input type="submit" id="p11" class="subbut"
      onclick="handleFree(['p1','p2','p11']);"
      value="D p" />
    <div id="ansp11"></div></li>
</ul>
```

Univariate polynomials can also be used as if they were functions. To evaluate a univariate polynomial at some point, apply the polynomial to the point.

```html
<ul>
  <li>
    <input type="submit" id="p12" class="subbut"
      onclick="handleFree(['p1','p2','p12']);"
      value="p(2)" />
    <div id="ansp12"></div></li>
</ul>
```

The same syntax is used for composing two univariate polynomials, i.e. substituting one polynomial for the variable in another. This substitutes q for the variable in p.

```html
<ul>
  <li>
    <input type="submit" id="p13" class="subbut"
      onclick="handleFree(['p1','p2','p3','p13']);"
      value="p(q)" />
    <div id="ansp13"></div></li>
</ul>
```

This substitutes p for the variable in q.
To obtain a list of coefficients of the polynomial, use \(\text{coefficients}\).

From this you can use \(\text{gcd}\) and \(\text{reduce}\) to compute the contents of the polynomial.

Alternatively (and more easily), you can just call \(\text{content}\).

Note that the operation \(\text{coefficients}\) omits the zero coefficients from the list. Sometimes it is useful to convert a univariate polynomial to a vector whose i-th position contains the degree i-1 coefficient of the polynomial.

To get a complete vector of coefficients, use the operation \(\text{vectorise}\), which takes a univariate polynomial and an integer denoting the length of the desired vector.
It is common to want to do something to every term of a polynomial, creating a new polynomial in the process. This is a function for iterating across the terms of a polynomial, squaring each term.

We can demonstrate squareTerms on p.

When the coefficients of the univariate polynomial belong to a field, (for example, when the coefficients are rational numbers, as opposed to integers. The important property of a field is that non-zero elements can be divided and produce another element. The quotient of the integers 2 and 3 is not another integer.) It is possible to compute quotients and remainders.
When the coefficients are rational numbers or rational expressions, the operation \( \textit{quo} \) computes the quotient of two polynomials.

The operation \( \textit{rem} \) computes the remainder.

The operation \( \textit{divide} \) can be used to return a record of both components.

Now we check the arithmetic.

It is also possible to integrate univariate polynomials when the coefficients belong to a field.
One application of univariate polynomials is to see expressions in terms of a specific variable. We start with a polynomial in a1 whose coefficients are quotients of polynomials in b1 and b2.

Since in this case we are not talking about using multivariate polynomials in only two variables, we use `<a href="db.xhtml?Polynomial">Polynomial</a>`. We also use `<a href="db.xhtml?Fraction">Fraction</a>` because we want fractions.

We push all the variables into a single quotient of polynomials.

Alternatively, we can view this as a polynomial in the variable. This is a mode-directed conversion: You indicate as much of the structure as you care about and let Axiom decide on the full type and how to do the transformation.

See `<a href="axbook/section-8.2.xhtml">Polynomial Factorization</a>` for a discussion of the factorization facilities in Axiom for univariate polynomials. For more information on related topics, see
The domain constructor <a href="db.xhtml?MultivariatePolynomial">MultivariatePolynomial</a> is similar to <a href="db.xhtml?Polynomial">Polynomial</a> except that it specifies the variables to be used. <a href="db.xhtml?MultivariatePolynomial">MultivariatePolynomial</a> are available for <a href="db.xhtml?MultivariatePolynomial">MultivariatePolynomial</a>. The abbreviation for <a href="db.xhtml?MultivariatePolynomial">MultivariatePolynomial</a> is <a href="db.xhtml?MultivariatePolynomial">MPOLY</a>. The type expressions
<pre>
MultivariatePolynomial([x,y],Integer)
</pre>
and
<pre>
MPOLY([x,y],INT)
</pre>
refer to the domain of multivariate polynomials in the variables x and y where the coefficients are restricted to be integers. The first variable specified is the main variable and the display of the polynomial reflects
CHAPTER 1. OVERVIEW

this. This polynomial appears with terms in descending powers of the variable x.

It is easy to see a different variable ordering by doing a conversion.

You can use other, unspecified variables, by using

Conversions can be used to re-express such polynomials in terms of the other variables. For example, you can first push all the variables into a polynomial with integer coefficients.

Now pull out the variables of interest.

Restriction: Axiom does not allow you to create types where MultivariatePolynomial is contained in the coefficient type of MultivariatePolynomial.
Therefore, 
<pre>
MPOLY([x,y],POLY INT)
</pre>
is legal but this is not: 
<pre>
POLY MPOLY([x,y],INT)n
</pre>

Multivariate polynomials may be combined with univariate polynomials to create types with special structures.
<ul>
<li><input type="submit" id="p6" class="subbut"
onclick="makeRequest('p6');"
value="q:UP(x,FRAC MPOLY([y,z],INT)):=(x^2-x*(z+1)/y+2)^2" />
<div id="ansp6"><div></div></div></li>
</ul>
This is a polynomial in x whose coefficients are quotients of polynomials in y and z. Use conversions for the structural rearrangements. z does not appear in a denominator and so it can be made the main variable.
<ul>
<li><input type="submit" id="p7" class="subbut"
onclick="handleFree(['p6','p7']);"
value="q::UP(z,FRAC MPOLY([x,y],INT))" />
<div id="ansp7"><div></div></div></li>
</ul>
Or you can make a multivariate polynomial in x and z whose coefficients are fractions in polynomials in y
<ul>
<li><input type="submit" id="p8" class="subbut"
onclick="handleFree(['p6','p8']);"
value="q::MPOLY([x,z],FRAC UP(y,INT))" />
<div id="ansp8"><div></div></div></li>
</ul>
A conversion like 
<pre>
q::MPOLY([x,y],FRAC UP(z,INT))
</pre>
is not possible in this example because y appears in the denominator of a fraction. As you can see, Axiom provides extraordinary flexibility in the manipulation and display of expressions via its conversion facility. 

For more information on related topics, see
<a href="polyspecificatypes1.xhtml">Polynomial</a>,
<a href="polyspecificatypes2.xhtml">UnivariatePolynomial</a>, and
<a href="polyspecificatypes4.xhtml">DistributedMultivariatePolynomial</a>. Issue the system command
<ul>
CHAPTER 1. OVERVIEW

...to display the full list of operations defined by
<code>a href="db.xhtml?MultivariatePolynomial">MultivariatePolynomial</a>.</code>

---

polyspecificitytypes4.xhtml

--- polyspecificitytypes4.xhtml ---

The construction
<code>a href="db.xhtml?DistributedMultivariatePolynomial">DMP</a> orders its monomials lexicographically while
<code>a href="db.xhtml?HomogeneousDistributedMultivariatePolynomial">HDMPP</a> orders them by total order refined by reverse lexicographic order.

---

<code>a href="db.xhtml?DistributedMultivariatePolynomial">DMP</a> orders its monomials lexicographically while
<code>a href="db.xhtml?HomogeneousDistributedMultivariatePolynomial">HDMPP</a> orders them by total order refined by reverse lexicographic order.
These constructors are mostly used in Groebner basis calculations.

Note that we get a different Groebner basis when we use the `HDMP` polynomials, as expected.

`GeneralDistributedMultivariatePolynomial` is somewhat more flexible in the sense that as well as accepting a list of variables to specify the variable ordering, it also takes a predicate on exponent vectors to specify the term ordering. With this polynomial type the user can experiment with the effect of using completely arbitrary term orderings. This flexibility
is mostly important for algorithms such as Groebner basis calculations which can be very sensitive to term orderings.

For more information on related topics, see
<a href="axbook/section-1.8.xhtml">Polynomials</a>,
<a href="axbook/section-2.7.xhtml">Conversion</a>,
<a href="polyspecificity.xhtml">Polynomial</a>,
<a href="polyspecificity2.xhtml">UnivariatePolynomial</a> and
<a href="polyspecificity3.xhtml">MultivariatePolynomial</a>,
Issue the system command
<ul>
<li><input type="submit" id="p9" class="subbut"
onclick="showcall('p9');"
value="show DistributedMultivariatePolynomial"/>
</li>
</ul>
to display the full list of operations defined by
<a href="db.xhtml?DistributedMultivariatePolynomial">DistributedMultivariatePolynomial</a>
and
\getchunk{page foot}

polysubstitutions.xhtml

— polysubstitutions.xhtml —

\getchunk{standard head}
\getchunk{handlefreevars}
\getchunk{axiom talker}
</script>
</head>
<body onload="resetvars();">
\getchunk{page head}
\getchunk{page head}

The function <a href="dbopeval.xhtml">eval</a> is used to substitute values into polynomials. Here's an example of how to use it:
<ul>
<li><input type="submit" id="p1" class="subbut"
onclick="makeRequest('p1');"
value="p:=x^2+y^2"/>
</li>
</ul>
<ul>
<li><input type="submit" id="p2" class="subbut"
onclick="handleFree(['p1','p2'])"
value="eval(p,x=5)"/>
</li>
</ul>
This example would give you the value of the polynomial \( p \) at 5. You can also substitute into polynomials with several variables. First, specify the polynomial, then give a list of the bindings of the form

\[
\text{variable} = \text{value}
\]

For examples:

Here \( x \) was replaced by \( a+b \), and \( y \) was replaced by \( c+d \).

Substitution is done "in parallel". That is, Axiom takes \( q(x,y) \) and returns \( q(y,x) \).

You can also substitute numerical values for some or all of the variables.
CHAPTER 1. OVERVIEW

puiseuxseries.xhtml

— puiseuxseries.xhtml —

\getchunk{standard head}
<\script type="text/javascript">
function commandline(arg) {
  myfunc = document.getElementById('function').value;
  myivar = document.getElementById('ivar').value;
 mypvar = document.getElementById('pvar').value;
  myevar = document.getElementById('evar').value;
  myival = document.getElementById('ival').value;
  mysval = document.getElementById('sval').value;
  ans = 'series('+myivar+'+->'+myfunc+','+mypvar+'='+myevar+','+
    myival+'...','+mysval+');
  alert(ans);
  return(ans);
}
\getchunk{showfullanswer}
\getchunk{axiom talker}
\</script>
\getchunk{page head}
<\table>
  <\tr>
    <\td>
      Enter the formula for the general coefficient of the series:
    \</td>
  \</tr>
  <\tr>
    <\td>
      <input type="text" id="function" size="80" tabindex="10"
        value="(-1)^((3*n-4)/6)/factorial(n-1/3)="/n>
    \</td>
  \</tr>
  <\tr>
    <\td>
      Enter the index variable for your formula:
      <input type="text" id="ivar" size="10" tabindex="20" value="n="/n>
    \</td>
  \</tr>
  <\tr>
    <\td>
      Enter the power series variable:
      <input type="text" id="pvar" size="10" tabindex="30" value="x="/n>
    \</td>
  \</tr>
  <\tr>
    <\td>
      Enter the point about which to expand:
      <input type="text" id="evar" size="10" tabindex="40" value="0="/n>
    \</td>
  \</tr>
\</table>
For Puiseux Series, the exponent of the power series variable ranges from an initial value, an arbitrary rational number, to plus infinity; the step size is any positive rational number.

Enter the initial value of the index (a rational number): <input type="text" id="ival" size="10" tabindex="50" value="4/3"/>

Enter the step size (a positive rational number): <input type="text" id="sval" size="10" tabindex="60" value="2"/>

---

reallimit.xhtml

--- reallimit.xhtml ---

```javascript
function commandline(arg) {
    var myform = document.getElementById("form2");
    var myfunct = myform.expr.value;
    var myvar = myform.vars.value;
    var mypoint = ";
    // decide what the limit point should be
    var finite = document.getElementById('finite').checked;
    if (finite == true)
        mypoint = document.getElementById('fpoint').value;
    if (document.getElementById('plus').checked == true)
        mypoint = "\pm\infty";
    if (document.getElementById('minus').checked == true)
        mypoint = "\mp\infty";
    // decide what the limit statement is
    if (document.getElementById('both').checked == true)
        ans = 'limit('+myfunct+','+myvar+'='+mypoint+')';
    else if (document.getElementById('right').checked == true) {
        if (finite == true)
            ans = 'limit('+myfunct+','+myvar+'='+mypoint+","+mypoint+',"right")';
        else {
```
```javascript
ans = 'limit('+myform.expr.value+','+myvar+'='+mypoint+');
};
// note: ignore direction if limit is %minutInfinity
if (document.getElementById('left').checked == true) {
    if (finite == true) {
        ans = 'limit('+myform.expr.value+','+myvar+'='+mypoint+","+"left")';
    } else {
        ans = 'limit('+myform.expr.value+','+myvar+'='+mypoint+');'
    };
};
return(ans);
\getchunk{showfullanswer}
\getchunk{axiom talker}
</script>
</head>
<body>
\getchunk{page head}
<form id="form2">
Enter the function you want to compute the limit of:<br/>
<input type="text" id="expr" tabindex="10" size="50"
    value="x*sin(1/x)"/>
Enter the name of the variable:<br/>
<input type="text" id="vars" tabindex="20" value="x"/><br/>
<input type="radio" id="finite" tabindex="30" checked="checked"
    name="point"/>
A finite point
<input type="text" id="fpoint" tabindex="20" value="0"/>
<input type="radio" id="plus" tabindex="40" name="point"/>
%plusInfinity<br/>
<input type="radio" id="minus" tabindex="50" name="point"/>
%minusInfinity<br/>
Compute the limit from:<br/>
<input type="radio" id="both" tabindex="60" name="direction" checked="checked"/>
both directions<br/>
<input type="radio" id="right" tabindex="70" name="direction"/>
the right<br/>
<input type="radio" id="left" tabindex="80" name="direction"/>
the left<br/>
</form>
\getchunk{continue button}
\getchunk{answer field}
\getchunk{page foot}
refsearchpage.xhtml
— refsearchpage.xhtml —
The November 2007 release of Axiom contains

- New MathML output mode. This mode allows Axiom to output expressions using standard MathML format. This complements the existing ability to output Fortran, IBM script, Latex, OpenMath, and algebra formats.
- Ninety-five domains have been documented for the )help command. Type )help to see the list.
- New regression tests were added to improve the release testing.
- Hyperdoc can now be restarted. Type )hd
- Testing has begun against Spiegel’s Mathematical Handbook from the Schaum’s Outline Series. These tests include Axiom’s solutions and have uncovered mistakes in the published text.

Bug fixes

- integrate((z\(a+1\))^b,z) no longer loops infinitely.
- laplace(log(z),z,w) returns "failed" instead of crashing.
- solve(z\=z,z) returns the correct answer.

Additional information sources:
CHAPTER 1. OVERVIEW

<!-- Online information is available here -->

The changelog file contains specific file-by-file changes.

-- rootpage.xhtml --

books/tanglec books/bookvol11.pamphlet rootpage.xhtml > rootpage.xhtml

-- rootpage.xhtml --
<td>Read Volume 0 -- The Jenks/Sutor Book</td>
</tr>
<tr>
<td>
<a href="tutorial.xhtml">
<b>Axiom Tutorial</b>
</a>
</td>
<td>Read Volume 1 -- The Tutorial</td>
</tr>
<tr>
<td>
<a href="topreferencepage.xhtml">
<b>Reference</b>
</a>
</td>
<td>Scan on-line documentation for AXIOM</td>
</tr>
<tr>
<td>
<a href="topicspage.xhtml">
<b>Topics</b>
</a>
</td>
<td>Learn how to use Axiom, by topic</td>
</tr>
<tr>
<td>
<a href="man0page.xhtml">
<b>Browser</b>
</a>
</td>
<td>Browse through the AXIOM library</td>
</tr>
<tr>
<td>
<a href="topexamplepage.xhtml">
<b>Examples</b>
</a>
</td>
<td>See examples of use of the library</td>
</tr>
<tr>
<td>
<a href="topsettingspage.xhtml">
<b>Settings</b>
</a>
</td>
<td>Display and change the system environment</td>
</tr>
<tr>
<td>
<a href="releasenotes.xhtml">
<b>What's New</b>
</a>
</td>
Enhancements in this version of Axiom

<table>
<thead>
<tr>
<th>Fonts</th>
<th>Test Axiom Fonts in your Browser</th>
</tr>
</thead>
</table>

Create a series by

- **Expansion**
  Expand a function in a series around a point

- **Taylor Series**
  Series where the exponent ranges over the integers from a non-negative integer value to plus infinity by an arbitrary positive integer step size.

- **Laurent Series**
Series where the exponent ranges from an arbitrary integer value to plus infinity by an arbitrary positive integer step size.

Series where the exponent ranges from an arbitrary rational value to plus infinity by an arbitrary positive rational number step size.
**CHAPTER 1. OVERVIEW**

Enter the power series variable:
<input type="text" id="var" size="10" tabindex="20" value="x"/>

Expand around the point:
<input type="text" id="point" size="10" tabindex="30" value="\pi/2"/>

---

**solve.xhtml**

--- solve.xhtml ---

What do you want to solve?
<table>
<tr>
<td>
<a href="solvelinearequations.xhtml">
A System of Linear Equations in equation form
</a>
</td>
</tr>
<tr>
<td>
<a href="solvelinearmatrix.xhtml">
A System of Linear Equations in matrix form
</a>
</td>
</tr>
<tr>
<td>
<a href="solvesystempolynomials.xhtml">
A System of Polynomial Equations
</a>
</td>
</tr>
</table>
solvelinearequations.xhtml

"--- solvelinearequations.xhtml ---"

<script type="text/javascript">
<![CDATA[

function indeps(i) {
    var ans="";
    for (var j = 0 ; j < i ; j++) {
        ans=ans+'x'+j
        if (j != (i - 1)) ans=ans+',';
    }
    return(ans);
}

function equation(i) {
    var ans="";
    for (var j = 0 ; j < i ; j++) {
        ans=ans+Math.floor(Math.random()*100)+'*x'+j;
        if (j != (i - 1)) ans=ans+'+';
    }
    ans=ans="="+Math.floor(Math.random()*100);
    return(ans);
}

function byelement() {
    // find out how many rows and columns, must be positive and nonzero
    var rcnt = parseInt(document.getElementById('rowcnt').value);
    if (rcnt <= 0) {
        alert("Rows must be positive and non-zero -- defaulting to 1");
        rcnt = 1;
        document.getElementById('rowcnt').value=1;
        return(false);
    }
    // remove the question and the buttons
    var quest = document.getElementById('question');
    var clicks = document.getElementById('clicks');
    quest.removeChild(clicks);
    // write "Elements"
    var tbl = document.getElementById('form2');
    var tblsize = tbl.rows.length;
    var row = tbl.insertRow(tblsize);

]]>"
var thecell = row.insertCell(0);
var tnode = document.createTextNode("Enter the equations:");
thecell.appendChild(tnode);

// create input boxes for the matrix values
for (var i = 0 ; i < rcnt ; i++) {
    tblsize = tblsize + 1;
    row = tbl.insertRow(tblsize);
    thecell = row.insertCell(0);
    tnode = document.createTextNode('equation '+i+': ');
    thecell.appendChild(tnode);
    thecell = row.insertCell(1);
    tnode = document.createElement('input');
    tnode.type = 'text';
    tnode.name = 'a'+i;
    tnode.id = 'a'+i;
    tnode.size=50;
    tnode.value=equation(rcnt);
    tnode.tabindex=20+i;
    thecell.appendChild(tnode);
}

// insert the request for the unknown
tblsize = tblsize + 1;
row = tbl.insertRow(tblsize);
thecell = row.insertCell(0);
tnode = document.createTextNode("Enter the unknowns (comma separated): ");
thecell.appendChild(tnode);
thecell = row.insertCell(1);
tnode = document.createElement('input');
tnode.type = 'text';
tnode.name = 'unk';
tnode.id = 'unk';

// insert a blank line
row = tbl.insertRow(tblsize);
thecell = row.insertCell(0);

// insert the continue button
var contnode = document.createElement('center');
contnode.appendChild(tnode);
return(false);
}

function commandline(arg) { var rcnt = parseInt(document.getElementById('rowcnt').value);
var cmdhead = 'solve(';
var cmdtail = '])';
var listbody = '[';
for (var j = 0 ; j < rcnt ; j++) {
    var aj = document.getElementById('a'+j).value;
    listbody = listbody+aj;
    if (j != (rcnt - 1)) listbody = listbody+',';
}
listbody = listbody+']';
cmdhead = cmdhead+listbody;
var ans = cmdhead+',['+document.getElementById('unk').value+cmdtail;
alert(ans);
return(ans);
}

getchunk{showfullanswer}
getchunk{axiom talker}
</script>
</head>
<body>
getchunk{page head}
<table id="form2">
<tr>
<td>
Enter the number of equations:
<input type="text" id="rowcnt" tabindex="10" size="10" value="2"/>
</td>
</tr>
</table>
<div id="question">
<div id="clicks">
<center>
<input type="button" value="Continue" onclick="byelement();"/>
</center>
</div>
</div>
getchunk{answer field}
getchunk{page foot}

solvelinearmatrix.xhtml

— solvelinearmatrix.xhtml —

getchunk{standard head}
<script type="text/javascript">
<!CDATA[

function byformula() {
    // find out how many rows and columns, must be positive and nonzero
    var rcnt = parseInt(document.getElementById('rowcnt').value);
    if (rcnt <= 0) {
        alert("Rows must be positive and non-zero -- defaulting to 1");
    } else {  

}]
</script>
rcnt = 1;
document.getElementById('rowcnt').value=1;
return(false);
}
var ccnt = parseInt(document.getElementById('colcnt').value);
if (ccnt <= 0) {
    alert("Columns must be positive and non-zero -- defaulting to 1");
    ccnt = 1;
document.getElementById('colcnt').value=1;
    return(false);
}

// remove the question and the buttons
var quest = document.getElementById('question');
var clicks = document.getElementById('clicks');
quest.removeChild(clicks);
var tbl = document.getElementById('form2');
var tblsize = tbl.rows.length;

// make the row variable question
// row variable left cell
var row = tbl.insertRow(tblsize);
var cell = row.insertCell(0);
var tnode = document.createTextNode("Enter the row variable");
cell.appendChild(tnode);

// row variable right cell
cell = row.insertCell(1);
tnode = document.createElement('input');
tnode.type = 'text';
tnode.name = 'rowvar';
tnode.id = 'rowvar';
tnode.size=10;
tnode.value='i';
tnode.tabindex=21;
cell.appendChild(tnode);

// make the column variable question
// column variable left cell
tblsize = tblsize + 1;
row = tbl.insertRow(tblsize);

// column variable right cell
cell = row.insertCell(1);
tnode = document.createElement('input');
tnode.type = 'text';
tnode.name = 'colvar';
tnode.id = 'colvar';
tnode.size=10;
tnode.value='j';
tnode.tabindex=22;
cell.appendChild(tnode);
cell = row.insertCell(0);
tnode = document.createTextNode("Enter the formulas for the elements");
cell.appendChild(tnode);
    // formula input field
tblsize = tblsize + 1;
row = tbl.insertRow(tblsize);
cell = row.insertCell(0);
tnode = document.createElement('input');
tnode.type = 'text';
tnode.name = 'formula1';
tnode.id = 'formula1';
tnode.size=50;
tnode.value = '1/(x-i-j-1)';
tnode.tabindex=23;
cell.appendChild(tnode);
tblsize = tblsize + 1;
row = tbl.insertRow(tblsize);
cell = row.insertCell(0);
tnode = document.createTextNode("Enter the vector, one per row:");
cell.appendChild(tnode);
    // formula input field
tblsize = tblsize + 1;
row = tbl.insertRow(tblsize);
cell = row.insertCell(0);
tnode = document.createElement('input');
tnode.type = 'text';
tnode.name = 'vec1';
tnode.id = 'vec1';
tnode.size=70;
tnode.value = '3,5';
tnode.tabindex=24;
cell.appendChild(tnode);
    // insert the continue button
tblsize = tblsize + 1;
row = tbl.insertRow(tblsize);
cell = row.insertCell(0);
tnode = document.createElement('input');
tnode.type = 'button';
tnode.id = 'contbutton';
tnode.value = 'Continue';
tnode.setAttribute("onclick","makeRequest('formula');");
tnode.tabindex=24;
cell.appendChild(tnode);
return(false);
}

function byelement() {
    // find out how many rows and columns, must be positive and nonzero
    var rcnt = parseInt(document.getElementById('rowcnt').value);
    if (rcnt <= 0) {
        alert("Rows must be positive and non-zero -- defaulting to 1");
        rcnt = 1;
document.getElementById('rowcnt').value=1;
return(false);
}
var ccnt = parseInt(document.getElementById('colcnt').value);
if (ccnt <= 0) {
    alert("Columns must be positive and non-zero -- defaulting to 1");
    ccnt = 1;
    document.getElementById('colcnt').value=1;
    return(false);
}

// remove the question and the buttons
var quest = document.getElementById('question');
var clicks = document.getElementById('clicks');
quest.removeChild(clicks);

// write "Elements"
var tbl = document.getElementById('form2');
var tblsize = tbl.rows.length;
var row = tbl.insertRow(tblsize);
var thecell = row.insertCell(0);
var tnode = document.createTextNode("Elements");
thecell.appendChild(tnode);

// create input boxes for the matrix values
for (var i = 0 ; i < rcnt ; i++) {
    row = tbl.insertRow(tblsize);
    for (var j = 0 ; j < ccnt ; j++) {
        thecell = row.insertCell(j);
        tnode = document.createElement('input');
tnode.type = 'text';
tnode.name = 'a'+i+'c'+j;
tnode.id = 'a'+i+'c'+j;
tnode.size=10;
tnode.tabindex=20+(i*10)+j;
        thecell.appendChild(tnode);
    }
    thecell = row.insertCell(j);
    tnode = document.createTextNode(' = ');
    thecell.appendChild(tnode);
    thecell = row.insertCell(j+1);
    tnode = document.createElement('input');
tnode.type = 'text';
tnode.name = 'k'+i;
tnode.id = 'k'+i;
tnode.size=10;
tnode.value='0';
tnode.tabindex=20+(i*10)+j+10;
    thecell.appendChild(tnode);
}

// insert a blank line
row = tbl.insertRow(tblsize);
thecell = row.insertCell(0);
tnode = document.createTextNode("");
theccell.appendChild(tnode);

// insert the continue button
var centnode = document.createElement('center');
tbl.parentNode.appendChild(centnode);
tnode = document.createElement('input');
 tnnode.type = 'button';
 tnnode.id = 'contbutton';
 tnnode.value = 'Continue';
 tnnode.setAttribute("onclick","makeRequest('element');");
 centnode.appendChild(tnnode);
 return(false);
}

function commandline(arg) {
 if (arg == 'element') {
  var rcnt = parseInt(document.getElementById('rowcnt').value);
  var ccnt = parseInt(document.getElementById('colcnt').value);
  // get the right side vector into list form
  var vecbody = '[';
  var homogeneous = true;
  for (var k = 0 ; k < rcnt ; k++) {
    var ki = document.getElementById('k'+k).value;
    // is it homogeneous?
    if (parseInt(ki) != 0) homogeneous = false;
    vecbody = vecbody+ki;
    if (k != (rcnt - 1)) vecbody = vecbody+',';
  }
  vecbody = vecbody+']';
  alert('vecbody=('+vecbody+');
  // get the matrix elements, make them into lists of lists
  var listbody = '];
  for (var i = 0 ; i < rcnt ; i++) {
    var listbody = listbody+'[';
    for (var j = 0 ; j < ccnt ; j++) {
      var aij = document.getElementById('a'+i+'c'+j).value;
      listbody = listbody+aij;
      if (j != (ccnt - 1)) listbody = listbody+',';
    }
    listbody = listbody+']';
    if (i != (rcnt - 1)) listbody = listbody+',';
  }
  var matcmd = 'matrix([['+listbody+']]);
  alert('matcmd=('+matcmd+');
  // now we decide whether to compute the nullSpace or solve
  if (homogeneous == true)
    cmd = 'nullSpace('+matcmd+');
  else
    cmd = 'solve('+matcmd+','+vecbody+');
  alert(cmd);
  return(cmd);
} else {
  var rcnt = parseInt(document.getElementById('rowcnt').value);
  var ccnt = parseInt(document.getElementById('colcnt').value);
  var vec = '['+document.getElementById('vec1').value+'];
  var cmdhead = 'matrix([[[';
  var cmdtail = ']);
  var formula = document.getElementById('formula1').value;
  var rowv = document.getElementById('rowvar').value;
  var colv = document.getElementById('colvar').value;
var cmd = cmdhead+formula+' for '+colv+' in 1..'+ccnt+' ]'+
' for '+rowv+' in 1..'+rcnt+cmdtail;

return(cmd);
}
}
]
]
\getchunk{showfullanswer}
\getchunk{axiom talker}

</script>
</head>
<body>
\getchunk{page head}
Enter the size of the matrix:
<table id="form2">
<tr>
<td size="10">Rows</td>
<td><input type="text" id="rowcnt" tabindex="10" size="10" value="2"></td>
</tr>
<tr>
<td>Columns</td>
<td><input type="text" id="colcnt" tabindex="20" size="10" value="3"></td>
</tr>
</table>
<div id="question">
<div id="clicks">
How would you like to enter the matrix elements?
<center>
<input type="button" value="By Formula" onclick="byformula();"/>
<input type="button" value="By Element" onclick="byelement();"/>
</center>
</div>
</div>
\getchunk{answer field}
\getchunk{page foot}

_____
solvesinglepolynomial.xhtml

— solvesinglepolynomial.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
solvesinglepolynomial.xhtml not implemented
\getchunk{page foot}

_____
solvesystempolynomials.xhtml

— solvesystempolynomials.xhtml —

\getchunk{standard head}
</head>
<body>
\getchunk{page head}
solvesystempolynomials.xhtml not implemented
\getchunk{page foot}

summation.xhtml

— summation.xhtml —

\getchunk{standard head}
<script type="text/javascript">
function commandline(arg) {
    var myform = document.getElementById("form2");
    return('sum('+myform.expr.value+','+myform.vars.value+'='+
        myform.lower.value+'..'+myform.upper.value+');');
}
\getchunk{showfullanswer}
\getchunk{axiom talker}
</script>
\getchunk{page head}
<body>
\getchunk{page head}
<form id="form2">
Enter the function you want to sum:<br/>
<input type="text" id="expr" tabindex="10" size="50" value="i^3"/>
<br/>
Enter the summation index:<br/>
<input type="text" id="vars" tabindex="20" value="i" size="5"/>
<br/>
Enter the limits of the sum: From:<br/>
<input type="text" id="lower" tabindex="30" value="1" size="5"/>
<br/>To:<br/>
<input type="text" id="upper" tabindex="40" value="n" size="5"/>
</form>
\getchunk{continue button}
\getchunk{answer field}
\getchunk{page foot}

systemvariables.xhtml

— systemvariables.xhtml —

\getchunk{standard head}
CHAPTER 1. OVERVIEW

systemvariables not implemented

---

taylorseries.xhtml

--- taylorseries.xhtml ---

<script type="text/javascript">
function commandline(arg) {
    myfunc = document.getElementById('function').value;
    myivar = document.getElementById('ivar').value;
    mypvar = document.getElementById('pvar').value;
    myevar = document.getElementById('evar').value;
    myival = document.getElementById('ival').value;
    mysval = document.getElementById('sval').value;
    ans = 'series('+myivar+'+->'+myfunc+','+mypvar+'='+myevar+','+
        myival+'..,'+mysval+');
    alert(ans);
    return(ans);
}
</script>
<table>
<tr>
    <td>
        Enter the formula for the general coefficient of the series:
    </td>
</tr>
<tr>
    <td>
        <input type="text" id="function" size="80" tabindex="10" value="1/factorial(i)"/>
    </td>
</tr>
<tr>
    <td>
        Enter the index variable for your formula:
        <input type="text" id="ivar" size="10" tabindex="20" value="i"/>
    </td>
</tr>
<tr>
    <td>
        Enter the power series variable:
    </td>
</tr>
</table>
For Taylor Series, the exponent of the power series variable ranges from an initial value, an arbitrary non-negative integer, to plus infinity; the step size is any positive integer.
<table>
<thead>
<tr>
<th>Examples of Axiom Operations, by topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers</td>
</tr>
<tr>
<td>Polynomials</td>
</tr>
<tr>
<td>Functions</td>
</tr>
<tr>
<td>Solving Equations</td>
</tr>
<tr>
<td>Calculus</td>
</tr>
<tr>
<td>Linear Algebra</td>
</tr>
<tr>
<td>Graphics</td>
</tr>
<tr>
<td>Algebra</td>
</tr>
<tr>
<td>Cryptography</td>
</tr>
</tbody>
</table>

---

topicspage.xhtml

— topicspage.xhtml —

<td><a href="numberspage.xhtml">Numbers</a></td>
<td>A look at different types of numbers</td>
</tr>
<tr>
<td><a href="polynomialpage.xhtml">Polynomials</a></td>
<td>Polynomials in Axiom</td>
</tr>
<tr>
<td><a href="functionpage.xhtml">Functions</a></td>
<td>Built-in and user-defined functions</td>
</tr>
<tr>
<td><a href="equationpage.xhtml">Solving Equations</a></td>
<td>Facilities for solving equations</td>
</tr>
<tr>
<td><a href="calculuspage.xhtml">Calculus</a></td>
<td>Using Axiom to do calculus</td>
</tr>
<tr>
<td><a href="linalgpage.xhtml">Linear Algebra</a></td>
<td>Axiom's linear algebra facilities</td>
</tr>
<tr>
<td><a href="graphicspage.xhtml">Graphics</a></td>
<td>Axiom's graphics facilities</td>
</tr>
<tr>
<td><a href="algebrapage.xhtml">Algebra</a></td>
<td>Axiom's abstract algebra facilities</td>
</tr>
<tr>
<td><a href="cryptopage.xhtml">Cryptography</a></td>
<td>Alasdair McAndrew's Cryptography Course Notes</td>
</tr>
System commands are used to perform Axiom environment management and change Axiom system variables.

<table>
<thead>
<tr>
<th>Commands</th>
<th>System commands that control your environment.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Settings</td>
<td>Change an Axiom variable.</td>
</tr>
</tbody>
</table>

tutorial.xhtml

— tutorial.xhtml —

tutorial not implemented
uglangpage.xhtml

---

---

ugsyiscmdpage.xhtml

---

---

usersguidepage.xhtml

---

---

rcm3720.input

---

---

str2lst(str) == [ord(str.i) - 65 for i in 1..#str]

lst2str(lst) == concat [char(lst.i+65)::String for i in 1..#lst]

str2num(str) ==
local strlst
strlst:=[ord(str.i) for i in 1..#str]
return wholeRadix(strlst)$RadixExpansion(256)::INT

num2str(n) ==
local tmp
tmp:=wholeRagits(n::RadixExpansion(256))
return concat [char(tmp.i)::String for i in 1..#tmp]

superIncreasing?(lst) ==
reduce(/\,[lst.i>reduce(+,[lst.j for j in 1..i-1]) for i in 2..#lst])

siSolve(lst,n) ==
local res,m,i
if not superIncreasing?(lst) then error "The list is not super-increasing"
m := n
res := [0 for i in 1..#lst]
for i in #lst..1 by -1 repeat
  if lst.i <= m then
    res.i := 1
    m := m - lst.i
  if m = 0 then return res
error "Unsolvable"

subsetsum(L:List(INT),N:INT):List(INT) ==
local x,Y
if N=0 then return([])
if N<0 or #L=0 then return([-1])
for x in L repeat
  Y:=subsetsum(remove(x,L),N)
  if Y=[-1] then return(Y)
  Y:=subsetsum(remove(x,L),N-x)
  if Y=[-1] then return(cons(x,Y))
return([-1])

signatures.txt

--- signatures.txt ---

RSA ---
n = 2^137-1 e = 17
message = "This is my text."
signature = 68767027465671577191073128495082796700768
n = (6^67-1)/5 e = 17
message = "Please feed my dog!"
signature = 17032150984563519936051049192595664358435909788852633

Rabin -----n = (3^59-1)/2
message = "Leave now."
signature =
n = (7^47-1)/6
message = "Arrive Thursday."
signature = 1894797231225344140197834472714111895509

El Gamal --------
p = next prime after 2^150
a = 2
B = 136985158577406312693119161120024351761244461
message = "Leave AT ONCE!"
signature r = 138908052505754392111976715361069425353578198
s = 1141326468070168229982976133801721430306004477

DSS ---
p = next prime after 2^170
q = 143441505468590696209
g = 672396402136852996799074813867123583326389281120278
B = 1394256880659564848116770226045673904445792389839
message = "Now's your chance!"
signature r = 64609209464638355801
s = 13824808741200493330

---

--- strang.input ---

rowmatrix(r:List(Fraction(Integer))):Matrix(Fraction(Integer)) ==
[r]::Matrix(Fraction(Integer))

columnmatrix(c:List(Fraction(Integer))):Matrix(Fraction(Integer)) ==
[[i] for i in c]::Matrix(Fraction(Integer))

k(n) ==
M := diagonalMatrix([2 for i in 1..n])
for i in 1..n-1 repeat M(i,i+1):=-1
for i in 1..n-1 repeat M(i+1,i):=-1
M::SquareMatrix(n,Fraction(Integer))

t(n) ==
M:=k(n)
N:=M::Matrix(Fraction(Integer))
qsetelt!(N,1,1,1)
N::SquareMatrix(n,Fraction(Integer))

b(n) ==
M:=k(n)
N:=M::Matrix(Fraction(Integer))
qsetelt!(N,1,1,1)
qsetelt!(N,n,n,1)
N::SquareMatrix(n,Fraction(Integer))

K:=k(3)
966

CHAPTER 1. OVERVIEW

T:=t(3)
B:=b(3)

———-

bitmaps/axiom1.bitmap
— axiom1.bitmap —
#define axiom_width 270
#define axiom_height 100
static char axiom_bits[] =
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x80, 0xff, 0x01,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0xff, 0x03, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xf8,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0xe0, 0xff, 0x0f,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0xff, 0x03, 0x00, 0x00,

{
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x7e,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0xff,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,

0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0xf8,
0x00,
0x00,
0x1f,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,

0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x3c,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0xff,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,

0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0xf8,
0x00,
0x00,
0x1f,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,

0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0xff,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,

0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0xf0,
0x00,
0x00,
0x1f,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,

0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0xff,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,

0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0xc0,
0x00,
0x00,
0x1f,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x80,
0x00,


CHAPTER 1. OVERVIEW

0x3f, 0x00, 0xff, 0x1f, 0x00, 0x00, 0x00, 0x0f, 0x01, 0x00, 0x80, 0xff,
0x03, 0xff, 0x00, 0x80, 0x7f, 0x00, 0x80, 0xff, 0x03, 0xe0, 0x0f, 0x00,
0x00, 0x80, 0xff, 0x0f, 0x00, 0x00, 0x0f, 0x0f, 0x00, 0x8f, 0xff, 0x0f,
0x00, 0x80, 0xff, 0x0f, 0x00, 0x00, 0x0f, 0x0f, 0x00, 0x8f, 0xff, 0x0f,
0x00, 0x80, 0xff, 0x0f, 0x00, 0x00, 0x0f, 0x0f, 0x00, 0x8f, 0xff, 0x0f,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
CHAPTER 1. OVERVIEW

0x00, 0x00, 0x00, 0xff, 0xff, 0xfe, 0xff, 0x0f, 0xff, 0x03, 0x00,
0x08, 0xff, 0x3f, 0x00, 0x0f, 0x07, 0xff, 0xff, 0x1f, 0xff, 0x03, 0x00,
0x0f, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x10, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x11, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x12, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x13, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x14, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x15, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x16, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x17, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
License

--Copyright (c) 2007 Arthur C. Ralts
--All rights reserved.
--
--Redistribution and use in source and binary forms, with or without
--modification, are permitted provided that the following conditions are
--met:
--
--- Redistributions of source code must retain the above copyright
-- notice, this list of conditions and the following disclaimer.
--
--- Redistributions in binary form must reproduce the above copyright
-- notice, this list of conditions and the following disclaimer in
-- the documentation and/or other materials provided with the
-- distribution.
--
--- Neither the name of Arthur C. Ralts nor the
-- names of its contributors may be used to endorse or promote products
-- derived from this software without specific prior written permission.
--
--THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
--IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
--TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
--PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
--OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
--EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
--PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
--PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
--LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
--NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
--SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Bibliography