The mpFormulaC Library and Toolbox Manual

Dietrich Hadler
Helge Hadler
Thomas Hadler

May 2015
Version 0.0.1

Original Authors of the GMP Manual: Torbjörn Granlund and the GMP Development Team
Original Authors of the MPIR Manual: William Hart and the MPIR Team
Original Authors of the MPFR Manual: Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, Philippe Théveny, Paul Zimmermann and the MPFR Team
Original Authors of the MPC Manual: Andreas Enge, Philippe Théveny and Paul Zimmermann
Original Authors of the MPFI Manual: Fabrice Rouillier, Nathalie Revol, Sylvain Chevillard, Hong Diep Nguyen and Christoph Lauter
Original Authors of the FLINT Manual: William Hart and the FLINT Team
Original Authors of the ARB Manual: Frederik Johanson and the ARB Team
Original Authors of the XCS-MPFI Manual: F. Blomquist

Subsequent modifications: Dietrich Hadler, Helge Hadler and Thomas Hadler

Preface

Arbitrary-precision arithmetic, also called bignum arithmetic, multiple precision arithmetic, or sometimes infinite-precision arithmetic, indicates that calculations are performed on numbers which digits of precision are limited only by the available memory of the host system.

Arbitrary precision is used in applications where the speed of arithmetic is not a limiting factor, or where precise results with very large numbers are required. It should not be confused with the symbolic computation provided by many computer algebra systems, which represent numbers by expressions such as , and can thus represent any computable number with infinite precision.

The mpFormulaC Library and Toolbox are based on a number of well-established libraries, which implement multiprecision arithmetic.

This manual is divided in various parts, which reflect different levels of confidence regarding the accuracy of the results.

Part II: Arbitrary Precision with Guaranteed Error Bounds.
Functions in this part come optionally with a guaranteed error bound, which can (in principal) be made arbitrarily small. Based mostly on GMP, MPFR, MPFI, MPC, FLINT, ARB and libmpdec.

Part III: Arbitrary Precision with Error Tracking.
Functions in this part include functions which do not guarantee an error bound, but provide error tracking. This includes a comprehensive selections of complex and real linear algebra functions, based on Eigen.

Part IV: Additional Functions. Work in progress

The use of these function various environments is described in some detail in the appendices:

Appendix A describes the interfaces to a number of popular programming languages and applications with built-in scripting languages.

If you want to re-build or change the library and/or toolbox, have a look at appendices C and D.

Finally, the mpFormula Library and Toolbox would not exist without the many authors and contributors of the underlying libraries. They are acknowledged in appendix E.

Dietrich Hadler
Helge Hadler
Thomas Hadler

Contents

List of Tables

List of Figures

Part I

Getting Started

Chapter 1

Introduction

1.1 Overview: Features and Setup

1.1.1 Features

The mpFormulaC distribution consists of two parts: the mpFormulaC Library and the mpFormulaC Toolbox.

1.1.2 The mpFormulaC Library

The mpFormulaC Library is a collection of numerical functions and procedures in multiprecision arithmetic. It is intended to be usable on multiple platforms (i.e. platforms supported by a recent version of the GNU Compiler Collection, e.g. Windows, GNU/Linux, Mac OS) and is provided in the form of source code, with interfaces to C and C++.
The following multi-precision types are supported:

- The MPZ arbitrary precision integer type of the GMP library.
- The MPQ arbitrary precision rational type of the GMP library.
- The MPD arbitrary precision decimal floating point type of the libmpdec library.
- The MPFR arbitrary precision real binary floating point type of the MPFR library.
- The MPC arbitrary precision complex binary floating point type of the MPC library.
- The MPFI arbitrary precision real binary interval arithmetic floating point type of the MPFI library.
- The MPFCI arbitrary precision complex binary interval arithmetic floating point type of the XSC-MPFI library.

In addition, the following hardware-based floating-point types are supported:

- The conventional single (32 bit) precision real binary floating point type (float in C).
- The conventional single (32 bit) precision complex binary floating point type (float in C).
- The conventional double (64 bit) precision real binary floating point type (double in C).

- The conventional double (64 bit) precision complex binary floating point type (double in C).
- The extended precision (80 bit) real binary floating point type of the Intel FPU.
- The extended precision (80 bit) complex binary floating point type of the Intel FPU.

All of these types are available as scalars, vectors, and matrices.

The mpFormulaC Library is based on GMP (Granlund & the GMP development team, 2013), MPFR (Fousse *et al.*, 2007), MPFI (Revol & Rouillier, 2005), MPC (Enge *et al.*, 2012), XSC-MPFI (Blomquist *et al.*, 2012), MPFRC++ (Holoborodko, 2008-2012), libmpdec (Krah, 2012), Eigen (Guennebaud *et al.*, 2010), Boost Math (Bristow *et al.*, 2013), Boost Random (Maurer & Watanabe, 2013).

1.1.3 The mpFormulaC Toolbox

The mpFormulaC Toolbox provides precompiled binaries for the Windows platform with multiple interfaces:

- A C interface: provides the most direct and efficient access to the numerical routines, and can be used as basis for other interfaces. Is intended to work with most C compilers, and can also be used from Objective C.
- A C++ interface: provides a rich set of multiprecision arithmetic functions, operators and procedures, which are accessible in a familiar syntax, thanks to operator overloading. Both 32 bit and 64 bit versions are provided. Is intended to work with most C++ compilers, and can also be used from Objective C++.
- A COM (Component Object Model) interface: multiprecision arithmetic functions and procedures, with arithmetic operators emulated as properties. Both 32 bit and 64 bit versions are provided. This interface makes the numerical routines available to all languages with COM support, including VBScript, JScript (Windows Script Host), Visual Basic for Applications, Visual Basic 6.0, OpenOffice Basic, Lua, Ruby, PHP CLI, Perl, Python, R (Statistical System) and Mathematica.
- A .NET Framework 4.0 interface: As for C++, arithmetic functions, operators and procedures are accessible in a familiar syntax. Both 32 bit and 64 bit versions are provided. This interface makes the numerical routines available to all languages with .NET Framework support, including VB.NET, C#, JScript 2010, F#, MS C++ (CLI), IronPython and Matlab.
- A Names Pipes and Command Line interface: this is designed to make sure that the calling application and the routines in the library are executed in separate processes, greatly enhancing stability.

1.1.4 System Requirement

This mpFormulaC Toolbox has the following system requirement:

- Microsoft Windows with Microsoft .NET Framework version 4.x (Full).

1.1.5 Installation

The mpFormulaC Toolbox can be downloaded from
https://github.com/DUHadler/.
Double-click on the downloaded file to start installation and then follow the instructions.
https://github.com/DUHadler/DUHTest1/tags.
https://github.com/DUHadler/DUHTest1/releases.

1.2 License

The mpFormulaC Toolbox is free software. It is licensed under the GNU Lesser General Public License, Version 3 (see appendix D.1.3). The manual for the mpFormulaC Library and Toolbox (this document) is licensed under the GNU Free Documentation License, Version 1.3 (see appendix D.1.5).

1.3 No Warranty

There is no warranty. See the GNU General Public License, Version 3 (see appendix D.1.4) for details.

Part II

GMP and related libraries

Chapter 2

GMP and related libraries: an overview

2.1 Integer Types and Fractions

2.2 FloatingPoint Types

2.2.1 Fixed Single Precision

The IEEE 754 standard specifies a binary32 as having:
Sign bit: 1 bit Exponent width: 8 bits Significand precision: 24 (23 explicitly stored) This gives from 6 to 9 significant decimal digits precision (if a decimal string with at most 6 significant decimal is converted to IEEE 754 single precision and then converted back to the same number of significant decimal, then the final string should match the original; and if an IEEE 754 single precision is converted to a decimal string with at least 9 significant decimal and then converted back to single, then the final number must match the original [3]).
Sign bit determines the sign of the number, which is the sign of the significand as well. Exponent is either an 8 bit signed integer from âĹŠ128 to 127 (2's Complement) or an 8 bit unsigned integer from 0 to 255 which is the accepted biased form in IEEE 754 binary32 definition. For this case an exponent value of 127 represents the actual zero.
The true significand includes 23 fraction bits to the right of the binary point and an implicit leading bit (to the left of the binary point) with value 1 unless the exponent is stored with all zeros. Thus only 23 fraction bits of the significand appear in the memory format but the total precision is 24 bits (equivalent to log10(224) âLĹ 7.225 decimal digits). See Kahan (1997).

2.2.2 Fixed Double Precision

Double-precision binary floating-point is a commonly used format on PCs, due to its wider range over single-precision floating point, in spite of its performance and bandwidth cost. As with single-precision floating-point format, it lacks precision on integer numbers when compared with an integer format of the same size. It is commonly known simply as double. The IEEE 754 standard specifies a binary64 as having:
Sign bit: 1 bit Exponent width: 11 bits Significand precision: 53 bits (52 explicitly stored) This gives from 15âĂŞ17 significant decimal digits precision. If a decimal string with at most 15 significant digits is converted to IEEE 754 double precision representation and then converted back to a string with the same number of significant digits, then the final string should match the original; and if an IEEE 754 double precision is converted to a decimal string with at least 17 significant digits and then converted back to double, then the final number must match the

original (see Kahan (1997)).

2.3 Arithmetic Operators

2.3.1 Addition

Operator +
Function **.Plus**(**a** As mpNum, **b** As mpNum) As mpNum Function **.PlusInt**(**a** As mpNum, **b** As Integer) As mpNum

The binary operator + is used to return the sum of the 2 operands a and b, and assign the result to c: c = a + b.

For languages not supporting operator overloading, the function .Plus can be used to achieve the same: c = a.Plus(b)

The function .PlusInt can be used if the second operand is an integer: c = a.PlusInt(b)

2.3.2 Substraction

Operator −
Function **.Minus**(**a** As mpNum, **b** As mpNum) As mpNum Function **.MinusInt**(**a** As mpNum, **b** As Integer) As mpNum

The binary operator − is used to return the difference of the 2 operands a and b, and assign the result to c: c = a - b.

For languages not supporting operator overloading, the function .Minus can be used to achieve the same: c = a.Minus(b)

The function .MinusInt can be used if the second operand is an integer: c = a.MinusInt(b)

2.3.3 Multiplication (Scalars, Vectors and Matrices)

Operator *
Function **.Times**(**a** As mpNum, **b** As mpNum) As mpNum Function **.TimesInt**(**a** As mpNum, **b** As Integer) As mpNum Function **.TimesMat**(**a** As mpNum, **b** As Integer) As mpNum Function **.DotProd**(**a** As mpNum, **b** As Integer) As mpNum Function **.LSH**(**a** As mpNum, **b** As Integer) As mpNum

The binary operator ∗ is used to return the product of the 2 operands a and b, and assign the result to c: c = a * b.

For languages not supporting operator overloading, the function .Times can be used to achieve the same: c = a.Times(b)

The function .TimesInt can be used if the second operand is an integer: c = a.TimesInt(b)

2.3.4 Scalar Division

Operator /
Function **.Div**(**a** As mpNum, **b** As mpNum) As mpNum Function **.DivInt**(**a** As mpNum, **b** As Integer) As mpNum Function **.RSH**(**a** As mpNum, **b** As Integer) As mpNum

The binary operator / is used to return the quotient of the 2 operands a and b, and assign the result to c: c = a / b.
For languages not supporting operator overloading, the function .Div can be used to achieve the same: c = a.Div(b)
The function .DivInt can be used if the second operand is an integer: c = a.DivInt(b)

2.3.5 Modulo

Operator **mod**
Function **.Mod**(**a** As mpNum, **b** As mpNum) As mpNum Function **.ModInt**(**a** As mpNum, **b** As Integer) As mpNum

The binary operator mod is used to return the modulo of the 2 operands a and b, and assign the result to c: c = a mod b.
For languages not supporting operator overloading, the function .Mod can be used to achieve the same: c = a.Mod(b)
The function .ModInt can be used if the second operand is an integer: c = a.ModInt(b)

2.3.6 Power

Operator ^
Function **.Pow**(**a** As mpNum, **b** As mpNum) As mpNum Function **.PowInt**(**a** As mpNum, **b** As Integer) As mpNum

The binary operator ^ is used to return a raised to the power of b, and assign the result to c: c = a ^ b.
For languages not supporting operator overloading, the function .Pow can be used to achieve the same: c = a.Pow(b)
The function .PowInt can be used if the second operand is an integer: c = a.PowInt(b)

2.4 Comparison Operators and Sorting

2.4.1 Equal

Operator = (VB.NET) Operator == (C#)
Function **.EQ**(**a** As mpNum, **b** As mpNum) As Boolean

The binary logical operator = returns TRUE if $a = b$ and FALSE otherwise, e.g.:
if (a = b) then

For languages not supporting operator overloading, the function .EQ can be used to achieve the same, e.g.:
if a.EQ(b) then

2.4.2 Greater or equal

Operator >=
Function **.GE**(**a** As mpNum, **b** As mpNum) As Boolean

The binary logical operator >= returns TRUE if $a \geq b$ and FALSE otherwise, e.g.:
if (a >= b) then
For languages not supporting operator overloading, the function .GE can be used to achieve the same, e.g.:
if a.GE(b) then

2.4.3 Greater than

Operator >
Function **.GT**(**a** As mpNum, **b** As mpNum) As Boolean

The binary logical operator > returns TRUE if $a > b$ and FALSE otherwise, e.g.:
if (a > b) then
For languages not supporting operator overloading, the function .GT can be used to achieve the same, e.g.:
if a.GT(b) then

2.4.4 Less or equal

Operator <=
Function **.LE**(**a** As mpNum, **b** As mpNum) As Boolean

The binary logical operator <= returns TRUE if $a \leq b$ and FALSE otherwise, e.g.:
if (a <= b) then
For languages not supporting operator overloading, the function .LE can be used to achieve the same, e.g.:
if a.LE(b) then

2.4.5 Less than

Operator <
Function **.LT**(**a** As mpNum, **b** As mpNum) As Boolean

The binary logical operator > returns TRUE if $a < b$ and FALSE otherwise, e.g.:
if (a < b) then
For languages not supporting operator overloading, the function .LT can be used to achieve the same, e.g.:
if a.LT(b) then

2.4.6 Not equal

Operator <> (VB.NET)
Operator ! = (C#)

Function **.NE**(**a** As mpNum, **b** As mpNum) As Boolean

The binary logical operator <> returns TRUE if $a \neq b$ and FALSE otherwise, e.g.:
if (a <> b) then
For languages not supporting operator overloading, the function .NE can be used to achieve the same, e.g.:
if a.NE(b) then

2.4.7 IsApproximate

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

2.4.8 IsSmall

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

2.5 Vectors, Matrices and Tables

2.5.1 Dimension (Vectors and Matrices)

Property **.Rows**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum
Property **.Cols**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum
Property **.Size**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2.5.2 Precision

Property **.Prec10**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum
Property **.Prec2**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2.5.3 Item

Property **.Item**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2.5.4 Row

Property **.Row**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum
Property **.TopRows**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum
Property **.MiddleRows**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum
Property **.BottomRows**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2.5.5 Column

Property **.Col**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum
Property **.LeftCols**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum
Property **.MiddleCols**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum

Property **.RightCols**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2.5.6 Matrix

Property **.FillLinearByStep**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum
Property **.SetRandomSymmetric**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum
Property **.Block**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum

Property **.TopLeftCorner**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum
Property **.TopRightCorner**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum
Property **.BottomLeftCorner**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum
Property **.BottomRightCorner**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum
Property **.Diagonal**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum
Property **.TriangularView**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum
Property **.Adjoint**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum
Property **.AsDiagonal**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2.5.7 Sorting

Function **.Sorted**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2.5.8 Table

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2.5.9 List of Tables

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Chapter 3

MPZ

3.0.10 Multiprecision Rational Numbers (GMP: MPQ)

The GMP reference is Granlund & the GMP development team (2013)
GMP is a free library for arbitrary precision arithmetic, operating on signed integers, rational numbers and floating point numbers. There is no practical limit on the precision except the ones implied by the available memory in the machine GMP runs on.
The GNU Multiple Precision Arithmetic Library (GMP) is a free library for arbitrary-precision arithmetic, operating on signed integers, rational numbers, and floating point numbers.[2] There are no practical limits to the precision except the ones implied by the available memory in the machine GMP runs on (operand dimension limit is 232-1 bits on 32-bit machines and 237 bits on 64-bit machines).[3] GMP has a rich set of functions, and the functions have a regular interface. The basic interface is for C but wrappers exist for other languages including Ada, C++, C#, OCaml, Perl, PHP, and Python. In the past, the Kaffe Java virtual machine used GMP to support Java built-in arbitrary precision arithmetic. This feature has been removed from recent releases, causing protests from people who claim that they used Kaffe solely for the speed benefits afforded by GMP.[4] As a result, GMP support has been added to GNU Classpath.[5]
The main target applications of GMP are cryptography applications and research, Internet security applications, and computer algebra systems.
GMP aims to be faster than any other bignum library for all operand sizes. Some important factors in doing this are:
Using full words as the basic arithmetic type. Using different algorithms for different operand sizes; algorithms that are faster for very big numbers are usually slower for small numbers. Highly optimized assembly language code for the most important inner loops, specialized for different processors. The first GMP release was made in 1991. It is constantly developed and maintained.[1]
GMP is part of the GNU project (although its website being off gnu.org may cause confusion), and is distributed under the GNU Lesser General Public License (LGPL).
GMP is used for integer arithmetic in many computer algebra systems such as Mathematica[6] and Maple.[7] It is also used in the Computational Geometry Algorithms Library (CGAL) because geometry algorithms tend to 'explode' when using ordinary floating point CPU math.[8]
GMP is needed to build the GNU Compiler Collection (GCC).[9]

3.0.10.1 mpz input

int mpz_set_str (mpz t rop, const char *str, int base) Set the value of rop from str, a null-terminated C string in base base. White space is allowed in the string, and is simply ignored. The base may vary from 2 to 62, or if base is 0, then the leading characters are used: 0x and 0X

for hexadecimal, 0b and 0B for binary, 0 for octal, or decimal otherwise. For bases up to 36, case is ignored; upper-case and lower-case letters have the same value. For bases 37 to 62, upper-case letter represent the usual 10..35 while lower-case letter represent 36..61. This function returns 0 if the entire string is a valid number in base base. Otherwise it returns âĹŠ1.

3.0.10.2 mpz output

char * mpz_get_str (char *str, int base, const mpz t op) Convert op to a string of digits in base base. The base argument may vary from 2 to 62 or from âĹŠ2 to âĹŠ36. For base in the range 2..36, digits and lower-case letters are used; for âĹŠ2..âĹŠ36, digits and upper-case letters are used; for 37..62, digits, upper-case letters, and lower-case letters (in that signifi cance order) are used.
If str is NULL, the result string is allocated using the current allocation function (see Chapter 13 [Custom Allocation], page 86). The block will be strlen(str)+1 bytes, that being exactly enough for the string and null-terminator. If str is not NULL, it should point to a block of storage large enough for the result, that being mpz_sizeinbase (op, base) + 2. The two extra bytes are for a possible minus sign, and the null-terminator. A pointer to the result string is returned, being either the allocated block, or the given str.
Book reference: Shoup (2009)
Background on discrete applied algebra: Hardy D.W. (2009)

3.1 Arithmetic Operators

3.1.1 Unary Minus

Function **intNeg**(***n*** *As mpNum*) As mpNum

The function intNeg returns $-n$

Parameter:
n: An Integer.

3.1.2 Addition

Function **intAdd**(***n1*** *As mpNum*, ***n2*** *As mpNum*) As mpNum

The function intAdd returns $n_1 + n_2$..

Parameters:
n1: An Integer.
n2: An Integer.

The function intAdd(n_1, n_2) returns the sum of n_1 and n_2:

$$\text{intAdd}(n_1, n_2) = n_1 + n_2. \tag{3.1.1}$$

3.1.3 Subtraction

Function **intSub**(***n1*** *As mpNum*, ***n2*** *As mpNum*) As mpNum

The function intSub returns $n_1 - n_2$..

Parameters:
n1: An Integer.
n2: An Integer.

The function intSub(n_1, n_2) returns the difference of n_1 and n_2:

$$\mathsf{intSub}(n_1, n_2) = n_1 - n_2. \tag{3.1.2}$$

3.1.4 Multiplication

Function **intMul**(***n1*** *As mpNum*, ***n2*** *As mpNum*) As mpNum

The function intMul returns $n_1 \times n_2$.

Parameters:
n1: An Integer.
n2: An Integer.

The function intMul(n_1, n_2) returns the product of n_1 and n_2:

$$\mathsf{intMul}(n_1, n_2) = n_1 \times n_2. \tag{3.1.3}$$

3.1.5 Fused-Multiply-Add fma

Function **intFma**(***n1*** *As mpNum*, ***n2*** *As mpNum*, ***n3*** *As mpNum*) As mpNum

The function intFma returns $(n_1 \times n_2) + n_3$.

Parameters:
n1: An Integer.
n2: An Integer.
n3: An Integer.

The function intFma(n_1, n_2, n_3) returns the product of n_1 and n_2, plus n_3:

$$\mathsf{intFma}(n_1, n_2, n_3) = (n_1 \times n_2) + n_3. \tag{3.1.4}$$

3.1.6 Fused-Multiply-Subtract fms

Function **intFms**(***n1*** *As mpNum*, ***n2*** *As mpNum*, ***n3*** *As mpNum*) As mpNum

The function intFms returns $(n_1 \times n_2) - n_3$.

Parameters:
n1: An Integer.
n2: An Integer.
n3: An Integer.

The function intFms(n_1, n_2, n_3) returns the product of n_1 and n_2, minus n_3:

$$\mathsf{intFms}(n_1, n_2, n_3) = (n_1 \times n_2) - n_3. \tag{3.1.5}$$

3.1.7 Multiplication by multiples of 2 (LSH)

Function **intLSH**(***n*** *As mpNum*, ***k*** *As mpNum*) As mpNum

The function intLSH returns the product of n and 2^k

Parameters:
n: An Integer.
k: An Integer.

The function intLSH(n, k) returns the product of n and 2^k:

$$\text{intLSH}(n, k) = n \times 2^k. \tag{3.1.6}$$

This operation can also be defined as a left shift by k bits.

3.1.8 Division by multiples of 2 (RSH)

Function **intRSH**(***n*** *As mpNum*, ***k*** *As mpNum*) As mpNum

The function intRSH returns the quotient of n and 2^k

Parameters:
n: An Integer.
k: An Integer.

The function intRSH(n, k) returns the quotient of n and 2^k:

$$\text{intRSH}(n, k) = n \div 2^k. \tag{3.1.7}$$

This operation can also be defined as a right shift by k bits.

3.1.9 Exact Division

Function **intDivExact**(***n*** *As mpNum*, ***d*** *As mpNum*) As mpNum

The function intDivExact returns n/d

Parameters:
n: An Integer.
d: An Integer.

Returns n/d. This function produces correct results only when it is known in advance that d divides n. This routine is much faster than the other division functions, and is the best choice when exact division is known to occur, for example reducing a rational to lowest terms.

3.1.10 Modulo Division

Function **intMod**(***n*** *As mpNum*, ***d*** *As mpNum*) As mpNum

The function intMod returns n mod d.

Parameters:

n: An Integer.
d: An Integer.

The sign of the divisor is ignored; the result is always non-negative.

3.2 Divisions, forming quotients and/or remainder

Division is undefined if the divisor is zero. Passing a zero divisor to the division or modulo functions (including the modular powering functions), will cause an intentional division by zero. This lets a program handle arithmetic exceptions in these functions the same way as for normal integer arithmetic.
The following routines calculate n divided by d, forming a quotient q and/or remainder r. For the 2exp functions, $d = 2^b$. The rounding is in three styles, each suiting different applications.

- cdiv rounds q up towards $+\infty$, and r will have the opposite sign to d. The c stands for "ceil".
- fdiv rounds q down towards $-\infty$, and r will have the same sign as d. The f stands for "floor".
- tdiv rounds q towards zero, and r will have the same sign as n. The t stands for "truncate".

In all cases q and r will satisfy $n = qd + r$, and r will satisfy $0 \leq |r| < |d|$. The q functions calculate only the quotient, the r functions only the remainder, and the qr functions calculate both. Note that for qr the same variable cannot be passed for both q and r, or results will be unpredictable.

3.2.1 Quotient only, rounded up

Function **intCDivQ**(***n*** *As mpNum,* ***d*** *As mpNum*) As mpNum

The function intCDivQ returns the quotient of n and d, rounded up towards $+\infty$.

Parameters:
n: An Integer.
d: An Integer.

$$\mathsf{intCDivQ}(n, d) = \lceil n \div d \rceil \,. \tag{3.2.1}$$

Function **intCDivQ2exp**(***n*** *As mpNum,* ***b*** *As mpNum*) As mpNum

The function intCDivQ2exp returns the quotient of n and 2^b, rounded up towards $+\infty$.

Parameters:
n: An Integer.
b: An Integer.

$$\mathsf{intCDivQ2exp}(n, b) = \left\lceil n \div 2^b \right\rceil \,. \tag{3.2.2}$$

3.2.2 Remainder only (Quotient rounded up)

Function **intCDivR**(***n*** *As mpNum*, ***d*** *As mpNum*) As mpNum

The function intCDivR returns the remainder, once the quotient of n and d, rounded up towards $+\infty$, has been obtained.

Parameters:
n: An Integer.
d: An Integer.

$$\text{intCDivR}(n, d) = n - d \times \lceil n \div d \rceil \,. \tag{3.2.3}$$

Function **intCDivR2exp**(***n*** *As mpNum*, ***b*** *As mpNum*) As mpNum

The function intCDivR2exp returns the remainder, once the quotient of n and 2^b, rounded up towards $+\infty$, has been obtained.

Parameters:
n: An Integer.
b: An Integer.

$$\text{intCDivR2exp}(n, d) = n - 2^b \times \left\lceil n \div 2^b \right\rceil \,. \tag{3.2.4}$$

3.2.3 Quotient and Remainder, Quotient rounded up

Function **intCDivQR**(***n*** *As mpNum*, ***d*** *As mpNum*) As mpNumList[2]

The function intCDivQR returns the quotient of n and d, rounded up towards $+\infty$, and the remainder.

Parameters:
n: An Integer.
d: An Integer.

intCDivQR[1] returns intCDivQ(n, k) as defined in equation 3.2.1, and intCDivQR[2] returns intCDivR(n, k) as defined in in equation 3.2.3.

3.2.4 Quotient only, rounded down

Function **intFDivQ**(***n*** *As mpNum*, ***d*** *As mpNum*) As mpNum

The function intFDivQ returns the quotient of n and d, rounded down towards $-\infty$.

Parameters:
n: An Integer.
d: An Integer.

$$\text{intFDivQ}(n, d) = \lfloor n \div d \rfloor \,. \tag{3.2.5}$$

Function **intFDivQ2exp**(***n*** *As mpNum*, ***b*** *As mpNum*) As mpNum

The function intFDivQ2exp returns the quotient of n and 2^b, rounded down towards $-\infty$.

Parameters:
n: An Integer.
b: An Integer.

$$\text{intFDivQ2exp}(n, d) = \left\lfloor n \div 2^b \right\rfloor . \tag{3.2.6}$$

3.2.5 Remainder only (Quotient rounded down)

Function **intFDivR**(***n*** *As mpNum*, ***d*** *As mpNum*) As mpNum

The function intFDivR returns the remainder, once the quotient of n and d, rounded down towards $-\infty$, has been obtained.

Parameters:
n: An Integer.
d: An Integer.

$$\text{intFDivR}(n, d) = n - d \times \lfloor n \div d \rfloor . \tag{3.2.7}$$

Function **intFDivR2exp**(***n*** *As mpNum*, ***b*** *As mpNum*) As mpNum

The function intFDivR2exp returns the remainder, once the quotient of n and 2^b, rounded down towards $-\infty$, has been obtained.

Parameters:
n: An Integer.
b: An Integer.

$$\text{intFDivR2exp}(n, d) = n - 2^b \times \left\lfloor n \div 2^b \right\rfloor . \tag{3.2.8}$$

3.2.6 Quotient and Remainder, Quotient rounded down

Function **intFDivQR**(***n*** *As mpNum*, ***d*** *As mpNum*) As mpNumList[2]

The function intFDivQR returns the quotient of n and d, rounded down towards $-\infty$, and the remainder.

Parameters:
n: An Integer.
d: An Integer.

intFDivQR[1] returns intFDivQ(n, k) as defined in equation 3.2.5, and intFDivQR[2] returns intFDivR(n, k) as defined in in equation 3.2.7.

3.2.7 Quotient only, Quotient truncated

Function **intTDivQ**(***n*** *As mpNum*, ***d*** *As mpNum*) As mpNum

The function intTDivQ returns the quotient of n and d, rounded towards zero.

Parameters:
n: An Integer.
d: An Integer.

$$\mathsf{intTDivQ}(n, d) = \lfloor n \div d \rfloor \,. \tag{3.2.9}$$

Function **intTDivQ2exp**(***n*** *As mpNum,* ***b*** *As mpNum*) As mpNum

The function intTDivQ2exp returns the quotient of n and 2^b, rounded towards zero.

Parameters:
n: An Integer.
b: An Integer.

$$\mathsf{intTDivQ2exp}(n, d) = \left\lfloor n \div 2^b \right\rfloor \,. \tag{3.2.10}$$

3.2.8 Remainder only (Quotient truncated)

Function **intTDivR**(***n*** *As mpNum,* ***d*** *As mpNum*) As mpNum

The function intTDivR returns the remainder, once the quotient of n and d,rounded towards zero, has been obtained.

Parameters:
n: An Integer.
d: An Integer.

$$\mathsf{intTDivR}(n, d) = n - d \times [n \div d] \,. \tag{3.2.11}$$

Function **intTDivR2exp**(***n*** *As mpNum,* ***b*** *As mpNum*) As mpNum

The function intTDivR2exp returns the remainder, once the quotient of n and 2^b, rounded towards zero, has been obtained.

Parameters:
n: An Integer.
b: An Integer.

$$\mathsf{intTDivR2exp}(n, d) = n - 2^b \times \left[n \div 2^b\right] \,. \tag{3.2.12}$$

3.2.9 Quotient and Remainder, Quotient truncated

Function **intTDivQr**(***n*** *As mpNum,* ***d*** *As mpNum*) As mpNumList[2]

The function intTDivQr returns the quotient of n and d, rounded towards zero, and the remainder.

Parameters:
n: An Integer.
d: An Integer.

intTDivQR[1] returns intTDivQ(n, k) as defined in equation 3.2.9, and intTDivQR[2] returns intTDivR(n, k) as defined in in equation 3.2.11.

3.3 Logical Operators

3.3.1 Bitwise AND

Function **intAND**(***n1*** *As mpNum*, ***n2*** *As mpNum*) As mpNum

The function intAND returns n_1 bitwise-and n_2.

Parameters:
n1: An Integer.
n2: An Integer.

3.3.2 Bitwise Inclusive OR

Function **intIOR**(***n1*** *As mpNum*, ***n2*** *As mpNum*) As mpNum

The function intIOR returns n_1 bitwise-inclusive-or n_2.

Parameters:
n1: An Integer.
n2: An Integer.

3.3.3 Bitwise Exclusive OR

Function **intXOR**(***n1*** *As mpNum*, ***n2*** *As mpNum*) As mpNum

The function intXOR returns n_1 bitwise-exclusive-or n_2.

Parameters:
n1: An Integer.
n2: An Integer.

3.4 Bit-Oriented Functions

3.4.1 Complement

Function **intComplement**(***n*** *As mpNum*) As mpNum

The function intComplement returns the one's complement of n.

Parameter:
n: An Integer.

3.4.2 Hamming Distance

Function **intHamDist**(***n1*** *As mpNum*, ***n2*** *As mpNum*) As mpNum

The function intHamDist returns the hamming distance between the two operands

Parameters:
n1: An Integer.
n2: An Integer.

If n_1 and n_2 are both ≥ 0 or both < 0, return the hamming distance between the two operands, which is the number of bit positions where n_1 and n_2 have different bit values. If one operand is ≥ 0 and the other < 0 then the number of bits different is infinite, and the return value is the largest possible mp_bitcnt_t.

3.4.3 Testing , setting, and clearing a Bit

Function **intTestBit**(***n*** *As mpNum,* ***k*** *As mpNum*) As mpNum

The function intTestBit returns 1 or 0 according to whether bit k in n is set or not.

Parameters:
n: An Integer.
k: An Integer.

Function **intComBit**(***n*** *As mpNum,* ***k*** *As mpNum*) As mpNum

The function intComBit returns n with the complement bit k set in n.

Parameters:
n: An Integer.
k: An Integer.

Function **intClearBit**(***n*** *As mpNum,* ***k*** *As mpNum*) As mpNum

The function intClearBit returns n with the bit k cleared in n.

Parameters:
n: An Integer.
k: An Integer.

Function **intSetBit**(***n*** *As mpNum,* ***k*** *As mpNum*) As mpNum

The function intSetBit returns n with the bit k set in n.

Parameters:
n: An Integer.
k: An Integer.

3.4.4 Scanning for 0 or 1

Function **intScan0**(***n*** *As mpNum,* ***k*** *As mpNum*) As mpNum

The function intScan0 returns the index of the found bit 0, starting from bit k.

Parameters:
n: An Integer.

k: An Integer.

Function **intScan1**(***n*** *As mpNum,* ***k*** *As mpNum*) As mpNum

---

The function intScan1 returns the index of the found bit 1, starting from bit k.

Parameters:
n: An Integer.
k: An Integer.

Scan n, starting from bit k, towards more significant bits, until the first 0 or 1 bit (respectively) is found. Return the index of the found bit.
If the bit at starting bit is already whatâĂŹs sought, then k is returned. If thereâĂŹs no bit found, then the largest possible mp_bitcnt_t is returned. This will happen in mpz_scan0 past the end of a negative number, or mpz_scan1 past the end of a nonnegative number.

3.4.5 Population Count

Function **intPopCount**(***n*** *As mpNum*) As mpNum

The function intPopCount returns the population count of n.

Parameter:
n: An Integer.

If $n \geq 0$, return the population count of n, which is the number of 1 bits in the binary representation. If $n < 0$, the number of 1s is infinite, and the return value is the largest possible mp_bitcnt_t.

3.5 Sign, Powers and Roots

3.5.1 Sign

Function **intSgn**(***n*** *As mpNum*) As mpNum

The function intSgn returns the sign of n.

Parameter:
n: An Integer.

3.5.2 Absolute value

Function **intAbs**(***n*** *As mpNum*) As mpNum

The function intAbs returns the absolute value of n.

Parameter:
n: An Integer.

3.5.3 Power Function: n^k; $n, k \in \mathbb{Z}$

Function **intPow**(***n*** *As mpNum*, ***k*** *As mpNum*) As mpNum

The function intPow returns the value of n^k. The case 0^0 yields 1.

Parameters:
n: An Integer.
k: An Integer.

3.5.4 Power Function modulo m: $n^k \mod m$; $m, n, k \in \mathbb{Z}$

Function **intPowMod**(***n*** *As mpNum*, ***k*** *As mpNum*, ***m*** *As mpNum*) As mpNum

The function intPowMod returns the value of $n^k \bmod m$.

Parameters:
n: An Integer.
k: An Integer.
m: An Integer.

Returns the value of $n^k \bmod m$; $m, n, k \in \mathbb{Z}$.
Negative k is supported if an inverse $n^{-1} \bmod m$ exists (see mpz_invert in Section 5.9 [Number Theoretic Functions], page 36). If an inverse does not exist then a divide by zero is raised.

3.5.5 Truncated integer part of the square root: $\lfloor\sqrt{n}\rfloor$

Function **intSqrt**(***n*** *As mpNum*) As mpNum

The function intSqrt returns the truncated integer part of the square root of n.

Parameter:
n: An Integer.

Returns $\lfloor\sqrt{m}\rfloor$, the truncated integer part of the square root of m.

3.5.6 Truncated integer part of the square root: $\lfloor\sqrt{m}\rfloor$, with remainder

Function **intSqrtRem**(***n*** *As mpNum*) As mpNumList[2]

The function intSqrtRem returns the truncated integer part of the square root of n, and the remainder.

Parameter:
n: An Integer.

intSqrtRem[1] returns s =intSqrt(n) as defined in section 3.5.5, and intSqrtRem[2] returns the remainder $(m - s^2)$, which will be zero if m is a perfect square.

3.5.7 Truncated integer part of the nth root: $\lfloor \sqrt[n]{m} \rfloor$

Function **intRoot**(***n*** *As mpNum*, ***m*** *As mpNum*) As mpNum

The function intRoot returns the truncated integer part of the n^{th} root of m

Parameters:
n: An Integer.
m: An Integer.

Returns $\lfloor \sqrt[n]{m} \rfloor$, the truncated integer part of the n^{th} root of m.

3.5.8 Truncated integer part of the nth root: $\lfloor \sqrt[n]{m} \rfloor$, with remainder

Function **intRootRem**(***n*** *As mpNum*, ***m*** *As mpNum*) As mpNumList[2]

The function intRootRem returns the truncated integer part of the n^{th} root of m, with remainder

Parameters:
n: An Integer.
m: An Integer.

intRootRem[1] returns $s =$intSqrt(m, n) as defined in section 3.5.7, and intRootRem[2] returns the remainder $(m - s^2)$.

3.6 Numbertheoretic Functions

3.6.1 Factorial

Function **intFactorial**(***n*** *As mpNum*) As mpNum

The function intFactorial returns $n!$, the factorial of n

Parameter:
n: An Integer.

3.6.2 Binomial Coefficient, Combinations

Function **intBinCoeff**(***n*** *As mpNum*, ***k*** *As mpNum*) As mpNum

The function intBinCoeff returns the binomial coefficient

Parameters:
n: An Integer.
k: An Integer.

Returns the binomial coefficient, $\binom{n}{k}$. Negative values of n are supported, using the identity

$$\binom{-n}{k} = (-1)^k \binom{n+k-1}{k}. \tag{3.6.1}$$

3.6.3 Next Prime

Function **intNextprime**(***n*** *As Integer*) As Integer

The function intNextprime returns the next prime greater than n.

Parameter:
n: An Integer.

Returns the next prime greater than n. This function uses a probabilistic algorithm to identify primes. The chance of a composite passing will be extremely small.

3.6.4 Greatest Common Divisor (GCD)

Function **intGcd**(***n1*** *As mpNum,* ***n2*** *As mpNum*) As mpNum

The function intGcd returns the greatest common divisor of n_1 and n_2

Parameters:
n1: An Integer.
n2: An Integer.

The result is always positive even if one or both input operands are negative. Except if both inputs are zero; then this function defines intGcd(0, 0) = 0.

3.6.5 Greatest Common Divisor, Extended

Function **intGcdExt**(***n1*** *As mpNum,* ***n2*** *As mpNum*) As mpNumList[3]

The function intGcdExt returns the extended greatest common divisor of n_1 and n_2

Parameters:
n1: An Integer.
n2: An Integer.

Set intGcdExt[1] = g to the greatest common divisor of a and b, and in addition set intGcdExt[2] = s and intGcdExt[3] = t to coefficients satisfying $as + bt = g$. The value in g is always positive, even if one or both of a and b are negative (or zero if both inputs are zero). The values in s and t are chosen such that normally, $|s| < |b|/(2g)$ and $|t| < |a|/(2g)$, and these relations define s and t uniquely. There are a few exceptional cases:

If $|a| = |b|$, then $s = 0, t = sgn(b)$.

Otherwise, $s = sgn(a)$ if $b = 0$ or $|b| = 2g$, and $t = sgn(b)$ if $a = 0$ or $|a| = 2g$.

In all cases, $s = 0$ if and only if $g = |b|$, i.e., if b divides a or $a = b = 0$.

3.6.6 Least Common Multiple (LCM)

Function **intLcm**(***n1*** *As mpNum,* ***n2*** *As mpNum*) As mpNum

The function intLcm returns the least common multiple of n_1 and n_2.

Parameters:

n1: An Integer.
n2: An Integer.

Returns the least common multiple of n_1 and n_2. The returned value is always positive, irrespective of the signs of n_1 and n_2. The returned value will be zero if either n_1 or n_2 is zero.

3.6.7 Inverse Modulus

Function **intInvertMod**(***n1*** *As mpNum,* ***n2*** *As mpNum*) As mpNum

The function intInvertMod returns the inverse of n_1 modulo n_2

Parameters:
n1: An Integer.
n2: An Integer.

Returns the inverse of n_1 modulo n_2. If the inverse exists, the indicator value is non-zero and the returned value will satisfy $0 < rop < |n_2|$. If an inverse does not exist the indicator value is zero and rop is undefined. The behaviour of this function is undefined when n_2 is zero.

3.6.8 Remove Factor

Function **intRemoveFactor**(***n*** *As mpNum,* ***f*** *As mpNum*) As mpNum

The function intRemoveFactor returns n with all occurrences of the factor f removed from n.

Parameters:
n: An Integer.
f: An Integer.

Remove all occurrences of the factor f from n and return the result in intRemoveFactor[1]. intRemoveFactor[2] contains how many such occurrences were removed.

3.6.9 Legendre Symbol

Function **intLegendreSymbol**(***a*** *As mpNum,* ***p*** *As mpNum*) As mpNum

The function intLegendreSymbol returns the Legendre symbol $\left(\frac{a}{p}\right)$.

Parameters:
a: An Integer.
p: An Integer.

Calculate the Legendre symbol $\left(\frac{a}{p}\right)$. This is defined only for p an odd positive prime, and for such p it is identical to the Jacobi symbol.

3.6.10 Jacobi Symbol

Function **intJacobiSymbol**(***a*** *As mpNum,* ***b*** *As mpNum*) As mpNum

The function intJacobiSymbol returns the Jacobi symbol $\left(\frac{a}{b}\right)$

Parameters:
a: An Integer.
b: An Integer.

Calculate the Jacobi symbol $\left(\frac{a}{b}\right)$. This is defined only for b odd.

3.6.11 Kronecker Symbol

Function **intKroneckerSymbol**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum

The function intKroneckerSymbol returns the Kronecker symbol $\left(\frac{a}{b}\right)$

Parameters:
a: An Integer.
b: An Integer.

Calculate the Jacobi symbol $\left(\frac{a}{b}\right)$ with the Kronecker extension $\left(\frac{a}{2}\right) = \left(\frac{2}{a}\right)$ when a odd, or when a odd, $\left(\frac{a}{2}\right) = 0$ when a even. When b is odd the Jacobi symbol and Kronecker symbol are identical.

3.6.12 Fibonacci Numbers

Function **intFibonacci**(***n*** *As mpNum*) As mpNum

The function intFibonacci returns the n^{th} Fibonacci number.

Parameter:
n: An Integer.

3.6.13 Lucas Numbers

Function **intLucas**(***n*** *As mpNum*) As mpNum

The function intLucas returns the n^{th} Lucas number.

Parameter:
n: An Integer.

3.7 Additional Numbertheoretic Functions

3.7.1 Pseudoprimes

An overview is provided by Grantham (2001).

Function **intIsBpswPrp**(***n*** *As mpNum*) As mpNum

The function intIsBpswPrp returns True if n is a Baillie-Pomerance-Selfridge-Wagstaff probable prime.

Parameter:
n: An Integer.

is_bpsw_prp(n) will return True if n is a Baillie-Pomerance-Selfridge-Wagstaff probable prime. A BPSW probable prime passes the is_strong_prp() test with base 2 and the is_selfridge_prp() test.

Function **intIsEulerPrp**(***n*** *As mpNum*, ***a*** *As mpNum*) As mpNum

---

The function intIsEulerPrp returns True if n is an Euler (also known as Solovay-Strassen) probable

Parameters:
n: An Integer.
a: An Integer.

is_euler_prp(n,a) will return True if n is an Euler (also known as Solovay-Strassen) probable prime to the base a.
Assuming: gcd(n, a) == 1 n is odd
Then an Euler probable prime requires:
a**((n-1)/2) == 1 (mod n)

Function **intIsExtraStrongLucasPrp**(***n*** *As mpNum*, ***p*** *As mpNum*) As mpNum

---

The function intIsExtraStrongLucasPrp returns True if n is an extra strong Lucas probable prime

Parameters:
n: An Integer.
p: An Integer.

is_extra_strong_lucas_prp(n,p) will return True if n is an extra strong Lucas probable prime with parameters (p,1). Assuming: n is odd D = p*p - 4, D != 0 gcd(n, 2*D) == 1 n = s*(2**r) + Jacobi(D,n), s odd
Then an extra strong Lucas probable prime requires:
lucasu(p,1,s) == 0 (mod n) or lucasv(p,1,s) == +/-2 (mod n) or lucasv(p,1,s*(2**t)) == 0 (mod n) for some t, 0 ¡= t ¡ r

---

Function **intIsFermatPrp**(***n*** *As mpNum*, ***a*** *As mpNum*) As mpNum

---

The function intIsFermatPrp returns True if n is a Fermat probable prime to the base a

Parameters:
n: An Integer.
a: An Integer.

is_fermat_prp(n,a) will return True if n is a Fermat probable prime to the base a.
Assuming: gcd(n,a) == 1 Then a Fermat probable prime requires: a**(n-1) == 1 (mod n)

Function **intIsFibonacciPrp**(***n*** *As mpNum*, ***p*** *As mpNum*, ***q*** *As mpNum*) As mpNum

---

The function intIsFibonacciPrp returns True if n is an Fibonacci probable prime with parameters (p,q).

Parameters:
n: An Integer.
p: An Integer.

q: An Integer.

is_fibonacci_prp(n,p,q) will return True if n is an Fibonacci probable prime with parameters (p,q).
Assuming: n is odd p ¿ 0, q = +/-1 p*p - 4*q != 0
Then a Fibonacci probable prime requires: lucasv(p,q,n) == p (mod n).

Function **intIsLucasPrp**(***n*** *As mpNum*, ***p*** *As mpNum*, ***q*** *As mpNum*) As mpNum

---

The function intIsLucasPrp returns True if n is a Lucas probable prime with parameters (p,q).

Parameters:
n: An Integer.
p: An Integer.
q: An Integer.

is_lucas_prp(n,p,q) will return True if n is a Lucas probable prime with parameters (p,q).
Assuming: n is odd D = p*p - 4*q, D != 0 gcd(n, 2*q*D) == 1
Then a Lucas probable prime requires:
lucasu(p,q,n - Jacobi(D,n)) == 0 (mod n)

Function **intIsSelfridgePrp**(***a*** *As mpNum*) As mpNum

The function intIsSelfridgePrp returns True if n is a Lucas probable prime with Selfidge parameters (p,q).

Parameter:
a: An Integer.

is_selfridge_prp(n) will return True if n is a Lucas probable prime with Selfidge parameters (p,q). The Selfridge parameters are chosen by finding the first element D in the sequence 5, -7, 9, -11, 13, ... such that Jacobi(D,n) == -1. Let p=1 and q = (1-D)/4 and then perform a Lucas probable prime test.

Function **intIsStrongBpswPrp**(***a*** *As mpNum*) As mpNum

The function intIsStrongBpswPrp returns True if n is a strong Baillie-Pomerance-Selfridge-Wagstaff probable prime

Parameter:
a: An Integer.

is_strong_bpsw_prp(n) will return True if n is a strong Baillie-Pomerance-Selfridge-Wagstaff probable prime. A strong BPSW probable prime passes the is_strong_prp() test with base 2 and the is_strongselfridge_prp() test.

Function **intIsStrongLucasPrp**(***n*** *As mpNum*, ***p*** *As mpNum*, ***q*** *As mpNum*) As mpNum

---

The function intIsStrongLucasPrp returns True if n is a strong Lucas probable prime with parameters (p,q).

Parameters:

n: An Integer.
p: An Integer.
q: An Integer.

is_strong_lucas_prp(n,p,q) will return True if n is a strong Lucas probable prime with parameters (p,q).
Assuming: n is odd D = p*p - 4*q, D != 0
gcd(n, 2*q*D) == 1 n = s*(2**r) + Jacobi(D,n), s odd Then a strong Lucas probable prime requires:
lucasu(p,q,s) == 0 (mod n) or lucasv(p,q,s*(2**t)) == 0 (mod n) for some t, 0 ¡= t ¡ r

---

Function **intIsStrongPrp**(***n*** *As mpNum,* ***a*** *As mpNum*) As mpNum

---

The function intIsStrongPrp returns True if n is an strong (also known as Miller-Rabin) probable prime

Parameters:
n: An Integer.
a: An Integer.

is_strong_prp(n,a) will return True if n is an strong (also known as Miller-Rabin) probable prime to the base a.
Assuming: gcd(n,a) == 1 n is odd n = s*(2**r) + 1, with s odd
Then a strong probable prime requires one of the following is true: a**s == 1 (mod n) or a**(s*(2**t)) == -1 (mod n) for some t, 0 ¡= t ¡ r.

Function **intIsStrongSelfridgePrp**(***a*** *As mpNum*) As mpNum

---

The function intIsStrongSelfridgePrp returns True if n is a strong Lucas probable prime with Selfidge parameters

Parameter:
a: An Integer.

is_strong_selfridge_prp(n) will return True if n is a strong Lucas probable prime with Selfidge parameters (p,q). The Selfridge parameters are chosen by finding the first element D in the sequence 5, -7, 9, -11, 13, ... such that Jacobi(D,n) == -1. Let p=1 and q = (1-D)/4 and then perform a strong Lucas probable prime test.

3.7.2 Lucas Sequences

An overview is provided by Joye & Quisquater (1996).

Function **intLucasU**(***p*** *As mpNum,* ***q*** *As mpNum,* ***k*** *As mpNum*) As mpNum

---

The function intLucasU returns the k-th element of the Lucas U sequence defined by p,q

Parameters:
p: An Integer.
q: An Integer.

k: An Integer.

lucasu(p,q,k) will return the k-th element of the Lucas U sequence defined by p,q. p*p - 4*q must not equal 0; k must be greater than or equal to 0.

Function **intLucasModU**(***p*** *As mpNum*, ***q*** *As mpNum*, ***k*** *As mpNum*, ***n*** *As mpNum*) As mpNum

---

The function intLucasModU returns the k-th element of the Lucas U sequence defined by p,q (mod n)

Parameters:

p: An Integer.

q: An Integer.

k: An Integer.

n: An Integer.

lucasu_mod(p,q,k,n) will return the k-th element of the Lucas U sequence defined by p,q (mod n). p*p - 4*q must not equal 0; k must be greater than or equal to 0; n must be greater than 0.

Function **intLucasV**(***p*** *As mpNum*, ***q*** *As mpNum*, ***k*** *As mpNum*) As mpNum

---

The function intLucasV returns the k-th element of the Lucas V sequence defined by p,q

Parameters:

p: An Integer.

q: An Integer.

k: An Integer.

lucasv(p,q,k) will return the k-th element of the Lucas V sequence defined by parameters (p,q). p*p - 4*q must not equal 0; k must be greater than or equal to 0.

Function **intLucasModV**(***p*** *As mpNum*, ***q*** *As mpNum*, ***k*** *As mpNum*, ***n*** *As mpNum*) As mpNum

---

The function intLucasModV returns the k-th element of the Lucas V sequence defined by p,q (mod n)

Parameters:

p: An Integer.

q: An Integer.

k: An Integer.

n: An Integer.

lucasv_mod(p,q,k,n) will return the k-th element of the Lucas V sequence defined by parameters (p,q) (mod n). p*p - 4*q must not equal 0; k must be greater than or equal to 0; n must be greater than 0.

3.8 Random Numbers

3.8.1 intUrandomb

Function **intUrandomb**(***n*** *As mpNum*) As mpNum

The function intUrandomb returns a uniformly distributed random integer in the range 0 to $2^n - 1$, inclusive.

Parameter:
n: An Integer.

3.8.2 intUrandomm

Function **intUrandomm**(***n*** *As mpNum*) As mpNum

The function intUrandomm returns a uniformly distributed random integer in the range 0 to $n - 1$, inclusive.

Parameter:
n: An Integer.

3.8.3 intRrandomb

Function **intRrandomb**(***n*** *As mpNum*) As mpNum

The function intRrandomb returns a random integer with long strings of zeros and ones in the binary representation.

Parameter:
n: An Integer.

Useful for testing functions and algorithms, since this kind of random numbers have proven to be more likely to trigger corner-case bugs. The random number will be in the range 0 to $2^n - 1$, inclusive.

3.9 Information Functions for Integers

3.9.1 Congruence: IsCongruent(*n*, *c*, *d*)

Function **IsCongruent**(***n*** *As mpNum*, ***d*** *As mpNum*, ***m*** *As mpNum*) As mpNum

The function IsCongruent returns TRUE if n is congruent to c modulo d, and FALSE otherwise.

Parameters:
n: An Integer.
d: An Integer.
m: An Integer.

Returns TRUE if n is congruent to c modulo d, and FALSE otherwise.
n is congruent to c mod d if there exists an integer q satisfying $n = c + qd$.

Unlike the other division functions, $d = 0$ is accepted and following the rule it can be seen that n and c are considered congruent mod 0 only when exactly equal.

3.9.2 Congruence 2n: IsCongruent2exp(*n*, *c*, *b*)

Function **IsCongruent2exp**(***n*** *As mpNum*, ***c*** *As mpNum*, ***b*** *As mpNum*) As mpNum

The function IsCongruent2exp returns TRUE if n is congruent to c modulo d, and FALSE otherwise.

Parameters:
n: An Integer.
c: An Integer.
b: An Integer.

n is congruent to c mod d if there exists an integer q satisfying $n = c + qd$.
Unlike the other division functions, $d = 0$ is accepted and following the rule it can be seen that n and c are considered congruent mod 0 only when exactly equal.

3.9.3 Primality Testing: IsProbablyPrime(*n*, *reps*)

Function **IsProbablyPrime**(***n*** *As mpNum*, ***reps*** *As mpNum*) As mpNum

The function IsProbablyPrime returns 2 if n is definitely prime, returns 1 if n is probably prime (without being certain), and returns 0 if n is definitely composite.

Parameters:
n: An Integer.
reps: An Integer.

This function does some trial divisions, then some Miller-Rabin probabilistic primality tests.
The argument *reps* controls how many such tests are done; a higher value will reduce the chances of a composite being returned as âĂIJprobably primeâĂİ. 25 is a reasonable number; a composite number will then be identified as a prime with a probability of less than 2^{-50}. Miller-Rabin and similar tests can be more properly called compositeness tests. Numbers which fail are known to be composite but those which pass might be prime or might be composite. Only a few composites pass, hence those which pass are considered probably prime.

3.9.4 Divisibility: IsDivisible(*n*, *d*)

Function **IsDivisible**(***n*** *As mpNum*, ***d*** *As mpNum*) As mpNum

The function IsDivisible returns TRUE if n is exactly divisible by d.

Parameters:
n: An Integer.
d: An Integer.

n is divisible by d if there exists an integer q satisfying $n = qd$.
Unlike the other division functions, $d = 0$ is accepted and following the rule it can be seen that only 0 is considered divisible by 0.

3.9.5 Divisibility by (2 pow b): IsDivisible2exp(n, b)

Function **IsDivisible2exp**(***n*** *As mpNum*, ***b*** *As mpNum*) As mpNum

The function IsDivisible2exp returns TRUE if n is exactly divisible by 2^b.

Parameters:
n: An Integer.
b: An Integer.

n is divisible by d if there exists an integer q satisfying $n = qd$.
Unlike the other division functions, $d = 0$ is accepted and following the rule it can be seen that only 0 is considered divisible by 0.

3.9.6 Perfect Power: IsPerfectPower(n)

Function **IsPerfectPower**(***n*** *As mpNum*) As mpNum

The function IsPerfectPower returns TRUE if n is a perfect power.

Parameter:
n: An Integer.

Returns TRUE if n is a perfect power, i.e., if there exist integers a and b, with $b > 1$, such that $n = a^b$. Under this definition both 0 and 1 are considered to be perfect powers. Negative values of n are accepted, but of course can only be odd perfect powers.

3.9.7 Perfect Square: IsPerfectSquare(n)

Function **IsPerfectSquare**(***n*** *As mpNum*) As mpNum

The function IsPerfectSquare returns non-zero if n is a perfect square.

Parameter:
n: An Integer.

Returns non-zero if n is a perfect square, i.e., if the square root of n is an integer. Under this definition both 0 and 1 are considered to be perfect squares.

Chapter 4

MPQ

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

4.0.7.1 mpq input

[Function] int mpq_set_str (mpq t rop, const char *str, int base) Set rop from a null-terminated string str in the given base.
The string can be an integer like âĂIJ41âĂİ or a fraction like âĂIJ41/152âĂİ. The fraction must be in canonical form (see Chapter 6 [Rational Number Functions], page 45), or if not then mpq_canonicalize must be called.
The numerator and optional denominator are parsed the same as in mpz_set_str (see Section 5.2 [Assigning Integers], page 30). White space is allowed in the string, and is simply ignored. The base can vary from 2 to 62, or if base is 0 then the leading characters are used:
0x or 0X for hex, 0b or 0B for binary, 0 for octal, or decimal otherwise. Note that this is done separately for the numerator and denominator, so for instance 0xEF/100 is 239/100, whereas 0xEF/0x100 is 239/256.
The return value is 0 if the entire string is a valid number, or âĹŠ1 if not.

4.0.7.2 mpq output

[Function] char * mpq_get_str (char *str, int base, const mpq t op) Convert op to a string of digits in base base. The base may vary from 2 to 36. The string will be of the form âĂŸnum/denâĂŹ, or if the denominator is 1 then just âĂŸnumâĂŹ. If str is NULL, the result string is allocated using the current allocation function (see Chapter 13 [Custom Allocation], page 86). The block will be strlen(str)+1 bytes, that being exactly enough for the string and null-terminator. If str is not NULL, it should point to a block of storage large enough for the result, that being mpz_sizeinbase (mpq_numref(op), base) + mpz_sizeinbase (mpq_denref(op), base) + 3 The three extra bytes are for a possible minus sign, possible slash, and the null-terminator. A pointer to the result string is returned, being either the allocated block, or the given str.

Chapter 5

MPD

The MPD reference is Krah (2012)
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

5.0.8 Multiprecision Decimal Numbers (MPD)

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

5.1 MPD Context

Explanation of decimal context
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim

rutrum.

Chapter 6

MPFR

6.0.1 Multiprecision with correct rounding (MPFR)

Provides a library for multiple-precision floating-point computation with exact rounding. The computation is both efficient and has a well-defined semantics. It copies the good ideas from the ANSI/IEEE-754 standard for double-precision floating-point arithmetic.
GNU MPFR (for GNU Multiple Precision Floating-Point Reliably[1]) is a portable C library for arbitrary-precision binary floating-point computation with correct rounding, based on GNU Multi-Precision Library. The computation is both efficient and has a well-defined semantics: the functions are completely specified on all the possible operands and the results do not depend on the platform. This is done by copying the ideas from the ANSI/IEEE-754 standard for fixed-precision floating-point arithmetic (correct rounding and exceptions, in particular). More precisely, its main features are:
Support for special numbers: signed zeros (âĹŠ0), infinities and not-a-number (a single NaN is currently supported). Each number has its own precision (in bits since MPFR uses radix 2). The floating-point results are correctly rounded to the precision of the target variable, in any of the four IEEE-754 rounding modes. Supported functions: MPFR implements all mathematical functions from C99: the logarithm and exponential in natural base, base 2 and base 10, the log(1+x) and exp(x)-1 functions (log1p and expm1), the six trigonometric and hyperbolic functions and their inverses, the gamma, zeta and error functions, the arithmetic geometric mean, the power (xy) function. All those functions are correctly rounded over their complete range. Subnormals are not supported, but can be emulated with the mpfr_subnormalize function. MPFR is not able to track the accuracy of numbers in a whole program or expression; this is not its goal. Interval arithmetic packages like MPFI, or Real RAM implementations like iRRAM, which may be based on MPFR, can do that for the user.

6.1 MPFR Context

MPFR is a portable library written in C for arbitrary precision arithmetic on floating-point numbers. Version 3.0.0 is used. MPFR is based on the GNU multiprecision library.

The MPFR code is portable, i.e. the result of any operation does not depend (or should not) on the machine word size mp_bits_per_limb (32 or 64 on most machines); the precision in bits can be set exactly to any valid value for each variable (including very small precision); MPFR provides the four rounding modes from the IEEE 754-1985 standard. In particular, with a precision of 53 bits, MPFR should be able to exactly reproduce all computations with double-precision machine floating-point numbers (e.g., double type in C, with a C implementation that rigorously

follows Annex F of the ISO C99 standard and FP_CONTRACT pragma set to OFF) on the four arithmetic operations and the square root, except the default exponent range is much wider and subnormal numbers are not implemented (but can be emulated).

There is one significant characteristic of floating-point numbers that has motivated a difference between this function class and other GNU MP function classes: the inherent inexactness of floating-point arithmetic. The user has to specify the precision for each variable. A computation that assigns a variable will take place with the precision of the assigned variable; the cost of that computation should not depend from the precision of variables used as input (on average).

The semantics of a calculation in MPFR is specified as follows: Compute the requested operation exactly (with "infinite accuracy"), and round the result to the precision of the destination variable, with the given rounding mode. The MPFR floating-point functions are intended to be a smooth extension of the IEEE 754-1985 arithmetic. The results obtained on one computer should not differ from the results obtained on a computer with a different word size.

MPFR does not keep track of the accuracy of a computation. This is left to the user or to a higher layer. As a consequence, if two variables are used to store only a few significant bits, and their product is stored in a variable with large precision, then MPFR will still compute the result with full precision.

6.1.1 Nomenclature and Types

A floating-point number as used in this chapter is an arbitrary precision significand (also called mantissa) with a limited precision exponent. The type for such objects is mp_fr. A floating-point number can have three special values: Not-a-Number (NaN) or plus or minus Infinity. NaN represents an uninitialized object, the result of an invalid operation (like 0 divided by 0), or a value that cannot be determined (like +Infinity minus +Infinity). Moreover, like in the IEEE 754-1985 standard, zero is signed, i.e. there are both +0 and -0; the behavior is the same as in the IEEE 754-1985 standard and it is generalized to the other functions supported by MPFR.

The precision is the number of bits used to represent the significand of a floating-point number. The precision can be any integer between mp_fr_PREC_MIN and mp_fr_PREC_MAX. In the current implementation, mp_fr_PREC_MIN is equal to 2.

Note: MPFR needs to increase the precision internally, in order to provide accurate results (and in particular, correct rounding). Do not attempt to set the precision to any value near mp_fr_PREC_MAX, otherwise MPFR will abort due to an assertion failure. Moreover, you may reach some memory limit on your platform, in which case the program may abort, crash or have undefined behavior.

The rounding mode specifies the way to round the result of a floating-point operation, in case the exact result can not be represented exactly in the destination significand; the corresponding VBA data type is mp_rnd_t.

A limb means the part of a multi-precision number that fits in a single word. (We chose this word because a limb of the human body is analogous to a digit, only larger, and containing several digits.) Normally a limb contains 32 or 64 bits.
There is only one class of functions in the MPFR library:
1. Functions for floating-point arithmetic, with names beginning with mp_fr_. The associated type is mp_fr_t.

6.1.2 Precision

Sub mp_fr_set_default_prec(prec As Long)
Set the default precision to be exactly prec bits. The precision of a variable means the number of bits used to store its significand. All subsequent calls to mp_fr_init will use this precision, but previously initialized variables are unaffected. This default precision is set to 53 bits initially. The precision can be any integer between mp_fr_PREC_MIN and mp_fr_PREC_MAX.

Function mp_fr_get_default_prec() As Long
Return the default MPFR precision in bits.

The following functions are useful for changing the precision during a calculation. A typical use would be for adjusting the precision gradually in iterative algorithms like Newton-Raphson, making the computation precision closely match the actual accurate part of the numbers.

Sub mp_fr_set_prec(x As mp_fr, prec As Long)
Reset the precision of x to be exactly prec bits, and set its value to NaN. The previous value stored in x is lost. It is equivalent to a call to mp_fr_clear(x) followed by a call to mp_fr_init2(x, prec), but more efficient as no allocation is done in case the current allocated space for the significand of x is enough. The precision prec can be any integer between mp_fr_PREC_MIN and mp_fr_PREC_MAX. In case you want to keep the previous value stored in x, use mp_fr_prec_round instead.

Function mp_fr_get_prec(x As mp_fr_t) As Long Return the precision actually used for assignments of x, i.e. the number of bits used to store its significand.

6.1.3 Rounding

The following four rounding modes are supported:

Enum mp_fr_rnd_t
mp_fr_RNDN: round to nearest.
mp_fr_RNDZ: round toward zero.
mp_fr_RNDU: round toward plus infinity.
mp_fr_RNDD: round toward minus infinity.
End Enum

The "round to nearest" mode works as in the IEEE 754-1985 standard: in case the number to be rounded lies exactly in the middle of two representable numbers, it is rounded to the one with the least significant bit set to zero. For example, the number 5/2, which is represented by (10.1) in binary, is rounded to (10.0) = 2 with a precision of two bits, and not to (11.0) = 3. This rule avoids the drift phenomenon mentioned by Knuth in volume 2 of The Art of Computer Programming (Section 4.2.2).

Most MPFR functions take as first argument the destination variable, as second and following arguments the input variables, as last argument a rounding mode, and have a return value of type mp_fr_status_t, called the ternary value.

Enum mp_fr_status_t
mp_fr_BELOW: the returned value is below the exact value.
mp_fr_EXACT: the returned value is exact.
mp_fr_ABOVE: the returned value is above the exact value.

End Enum

The value stored in the destination variable is correctly rounded, i.e. MPFR behaves as if it computed the result with an infinite precision, then rounded it to the precision of this variable. The input variables are regarded as exact (in particular, their precision does not affect the result).

As a consequence, in case of a non-zero real rounded result, the error on the result is less or equal to 1/2 ulp (unit in the last place) of the target in the rounding to nearest mode, and less than 1 ulp of the target in the directed rounding modes (a ulp is the weight of the least significant represented bit of the target after rounding).

If the ternary value is zero, it means that the value stored in the destination variable is the exact result of the corresponding mathematical function. If the ternary value is positive (resp. negative), it means the value stored in the destination variable is greater (resp. lower) than the exact result. For example with the GMP_RNDU rounding mode, the ternary value is usually positive, except when the result is exact, in which case it is zero. In the case of an infinite result, it is considered as inexact when it was obtained by overflow, and exact otherwise. A NaN result (Not-a-Number) always corresponds to an exact return value. The opposite of a returned ternary value is guaranteed to be representable in an int.

Unless documented otherwise, functions returning a 1 (or any other value specified in this manual) for special cases (like acos(0)) should return an overflow or an underflow if 1 is not representable in the current exponent range.

Sub mp_fr_set_default_rounding_mode([rnd As mp_fr_rnd_t = GMP_RNDN])
Set the default rounding mode to rnd. The default rounding mode is to nearest initially.

Function mp_fr_get_default_rounding_mode() As mp_fr_rnd_t
Get the default rounding mode.

Function mp_fr_prec_round(x As mp_fr, prec As Long, [rnd As mp_fr_rnd_t = GMP_RNDN]) As mp_fr_status_t
Round x according to rnd with precision prec, which must be an integer between mp_fr_PREC_MIN and mp_fr_PREC_MAX (otherwise the behavior is undefined). If prec is greater or equal to the precision of x, then new space is allocated for the significand, and it is filled with zeros. Otherwise, the significand is rounded to precision prec with the given direction. In both cases, the precision of x is changed to prec.

Function mp_fr_print_rnd_mode([rnd As mp_fr_rnd_t = GMP_RNDN]) As String
Return the input string (GMP RNDD, GMP RNDU, GMP RNDN, GMP RNDZ) corresponding to the rounding mode rnd .

6.1.4 Exponent

Function mp_fr_get_exp(x As mp_fr, [rnd As mp_fr_rnd_t = GMP_RNDN]) As Long
Get the exponent of x, assuming that x is a non-zero ordinary number and the significand is chosen in [1/2,1). The behavior for NaN, infinity or zero is undefined.

Function mp_fr_set_exp(x As mp_fr, e As Long, [rnd As mp_fr_rnd_t = GMP_RNDN]) As mp_fr_status_t
Set the exponent of x if e is in the current exponent range, and return 0 (even if x is not a non-zero ordinary number); otherwise, return a non-zero value. The significand is assumed to be in [1/2,1).

Function mp_fr_get_emin() As Long
Function mp_fr_get_emax() As Long
Return the (current) smallest and largest exponents allowed for a floating-point variable. The smallest positive value of a floating-point variable is Â · ÃŮ 2emin and the largest value has the form (1 âĹŠ Îţ) ÃŮ 2emax.

Function mp_fr_set_emin(exp As Long) As mp_fr_status_t
Function mp_fr_set_emax(exp As Long) As mp_fr_status_t
Set the smallest and largest exponents allowed for a floating-point variable. Return a non-zero value when exp is not in the range accepted by the implementation (in that case the smallest or largest exponent is not changed), and zero otherwise. If the user changes the exponent range, it is her/his responsibility to check that all current floating-point variables are in the new allowed range (for example using mp_fr_check_range), otherwise the subsequent behavior will be undefined, in the sense of the ISO C standard.

Function mp_fr_get_emin_min() As Long
Function mp_fr_get_emin_max() As Long
Function mp_fr_get_emax_min() As Long
Function mp_fr_get_emax_max() As Long
Return the minimum and maximum of the smallest and largest exponents allowed for mp_fr_set_emin and mp_fr_set_emax. These values are implementation dependent; it is possible to create a non portable program by writing mp_fr_set_emax(mp_fr_get_emax_max()) and mp_fr_set_emin(mp_fr_get_emin_mi since the values of the smallest and largest exponents become implementation dependent.

Function mp_fr_check_range(x As mp_fr, t As Long, [rnd As mp_fr_rnd_t = GMP_RNDN]) As mp_fr_status_t
This function forces x to be in the current range of acceptable values, t being the current ternary value: negative if x is smaller than the exact value, positive if x is larger than the exact value and zero if x is exact (before the call). It generates an underflow or an overflow if the exponent of x is outside the current allowed range; the value of t may be used to avoid a double rounding. This function returns zero if the rounded result is equal to the exact one, a positive value if the rounded result is larger than the exact one, a negative value if the rounded result is smaller than the exact one. Note that unlike most functions, the result is compared to the exact one, not the input value x, i.e. the ternary value is propagated.

Note: If x is an infinity and t is different from zero (i.e., if the rounded result is an inexact infinity), then the overflow flag is set. This is useful because mp_fr_check_range is typically called (at least in MPFR functions) after restoring the flags that could have been set due to internal computations.

Function mp_fr_subnormalize(x As mp_fr, t As Long, [rnd As mp_fr_rnd_t = GMP_RNDN]) As mp_fr_status_t
This function rounds x emulating subnormal number arithmetic: if x is outside the subnormal exponent range, it just propagates the ternary value t; otherwise, it rounds x to precision EXP(x)-emin+1 according to rounding mode rnd and previous ternary value t, avoiding double rounding problems. More precisely in the subnormal domain, denoting by e the value of emin, x is rounded in fixed-point arithmetic to an integer multiple of 2e âĹŠ 1; as a consequence, 1.5e âĹŠ 1 when t is zero is rounded to 2e with rounding to nearest.

PREC(x) is not modified by this function. rnd and t must be the used rounding mode for computing x and the returned ternary value when computing x. The subnormal exponent range

is from emin to emin+PREC(x)-1. If the result cannot be represented in the current exponent range (due to a too small emax), the behavior is undefined. Note that unlike most functions, the result is compared to the exact one, not the input value x, i.e. the ternary value is propagated. This is a preliminary interface.

This is an example of how to emulate double IEEE-754 arithmetic using MPFR:

```
{
mp_fr xa, xb;
int i;
volatile double a, b;
mp_fr_set_default_prec (53);
mp_fr_set_emin (-1073);
mp_fr_set_emax (1024);
mp_fr_init (xa); mp_fr_init (xb);
b = 34.3; mp_fr_set_d (xb, b, GMP_RNDN);
a = 0x1.1235P-1021; mp_fr_set_d (xa, a, GMP_RNDN);
a /= b;
i = mp_fr_div (xa, xa, xb, GMP_RNDN);
i = mp_fr_subnormalize (xa, i, GMP_RNDN);
mp_fr_clear (xa); mp_fr_clear (xb);
}
```

6.1.5 Status Flags

Sub mp_fr_clear_underflow()
Sub mp_fr_clear_overflow()
Sub mp_fr_clear_nanflag()
Sub mp_fr_clear_inexflag()
Sub mp_fr_clear_erangeflag()
Clear the underflow, overflow, invalid, inexact and erange flags.

Sub mp_fr_set_underflow()
Sub mp_fr_set_overflow()
Sub mp_fr_set_nanflag()
Sub mp_fr_set_inexflag()
Sub mp_fr_set_erangeflag()
Set the underflow, overflow, invalid, inexact and erange flags.

Sub mp_fr_clear_flags()
Clear all global flags (underflow, overflow, inexact, invalid, erange).

Function mp_fr_underflow_p() As Boolean
Function mp_fr_overflow_p() As Boolean
Function mp_fr_nanflag_p() As Boolean
Function mp_fr_inexflag_p() As Boolean
Function mp_fr_erangeflag_p() As Boolean
Return TRUE if the corresponding (underflow, overflow, invalid, inexact, erange) flag is set.

6.1.6 Exceptions

MPFR supports 5 exception types:

Underflow: An underflow occurs when the exact result of a function is a non-zero real number and the result obtained after the rounding, assuming an unbounded exponent range (for the rounding), has an exponent smaller than the minimum exponent of the current range. In the round-to-nearest mode, the halfway case is rounded toward zero. Note: This is not the single definition of the underflow. MPFR chooses to consider the underflow after rounding. The underflow before rounding can also be defined. For instance, consider a function that has the exact result 7 ÃŮ 2eâĹŠ4, where e is the smallest exponent (for a significand between 1/2 and 1) in the current range, with a 2-bit target precision and rounding toward plus infinity. The exact result has the exponent eâĹŠ1. With the underflow before rounding, such a function call would yield an underflow, as eâĹŠ1 is outside the current exponent range. However, MPFR first considers the rounded result assuming an unbounded exponent range. The exact result cannot be represented exactly in precision 2, and here, it is rounded to 0.5 ÃŮ 2e, which is representable in the current exponent range. As a consequence, this will not yield an underflow in MPFR.

Overflow: An overflow occurs when the exact result of a function is a non-zero real number and the result obtained after the rounding, assuming an unbounded exponent range (for the rounding), has an exponent larger than the maximum exponent of the current range. In the round-to-nearest mode, the result is infinite.

NaN: A NaN exception occurs when the result of a function is a NaN.

Inexact: An inexact exception occurs when the result of a function cannot be represented exactly and must be rounded.

Range error: A range exception occurs when a function that does not return a MPFR number (such as comparisons and conversions to an integer) has an invalid result (e.g. an argument is NaN in mp_fr_cmp or in a conversion to an integer).

MPFR has a global flag for each exception, which can be cleared, set or tested by functions described in Section 42.19 [Exception Related Functions].

Differences with the ISO C99 standard:
In C, only quiet NaNs are specified, and a NaN propagation does not raise an invalid exception. Unless explicitly stated otherwise, MPFR sets the NaN flag whenever a NaN is generated, even when a NaN is propagated (e.g. in NaN + NaN), as if all NaNs were signaling.
An invalid exception in C corresponds to either a NaN exception or a range error in MPFR.
The MPFR reference is Fousse *et al.* (2007)
The Brent reference is Brent & Zimmermann (2010)
The Holoborodko reference is Holoborodko (2008-2012)
The Wilkening reference is Wilkening (2008)
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis

nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

6.1.6.1 mpfr input

[Function] int mpfr_strtofr (mpfr t rop, const char *nptr, char **endptr, int base, mpfr rnd t rnd) Read a fl oating-point number from a string nptr in base base, rounded in the direction rnd; base must be either 0 (to detect the base, as described below) or a number from 2 to 62 (otherwise the behavior is undefi ned). If nptr starts with valid data, the result is stored in rop and *endptr points to the character just after the valid data (if endptr is not a null pointer); otherwise rop is set to zero (for consistency with strtod) and the value of nptr is stored in the location referenced by endptr (if endptr is not a null pointer). The usual ternary value is returned.
Parsing follows the standard C strtod function with some extensions. After optional leading whitespace, one has a subject sequence consisting of an optional sign (+ or -), and either numeric data or special data. The subject sequence is defi ned as the longest initial subsequence of the input string, starting with the fi rst non-whitespace character, that is of the expected form.
The form of numeric data is a non-empty sequence of signifi cand digits with an optional decimal point, and an optional exponent consisting of an exponent prefi x followed by an optional sign and a non-empty sequence of decimal digits. A signifi cand digit is eithera decimal digit or a Latin letter (62 possible characters), with A = 10, B = 11, . . ., Z = 35; case is ignored in bases less or equal to 36, in bases larger than 36, a = 36, b = 37, . . ., z = 61. The value of a signifi cand digit must be strictly less than the base. The decimal point can be either the one defi ned by the current locale or the period (the fi rst one is accepted for consistency with the C standard and the practice, the second one is accepted to allow the programmer to provide MPFR numbers from strings in a way that does not depend on the current locale). The exponent prefi x can be e or E for bases up to 10, or @ in any base; it indicates a multiplication by a power of the base. In bases 2 and 16, the exponent prefi x can also be p or P, in which case the exponent, called binary exponent, indicates a multiplication by a power of 2 instead of the base (there is a diff erence only for base 16); in base 16 for example 1p2 represents 4 whereas 1@2 represents 256. The value of an exponent is always written in base 10.
If the argument base is 0, then the base is automatically detected as follows. If the signifi cand starts with 0b or 0B, base 2 is assumed. If the signifi cand starts with 0x or 0X, base 16 is assumed. Otherwise base 10 is assumed.
Note: The exponent (if present) must contain at least a digit. Otherwise the possible exponent prefi x and sign are not part of the number (which ends with the signifi cand). Similarly,if 0b, 0B, 0x or 0X is not followed by a binary/hexadecimal digit, then the subject sequence stops at the character 0, thus 0 is read.
Special data (for infi nities and NaN) can be @inf@ or @nan@(n-char-sequence-opt), and if base âĽď 16, it can also be infinity, inf, nan or nan(n-char-sequence-opt), all case insensitive. A n-char-sequence-opt is a possibly empty string containing only digits, Latin letters and the underscore (0, 1, 2, . . ., 9, a, b, . . ., z, A, B, . . ., Z,). Note: one has an optional sign for all data, even NaN. For example, -@nAn@(This_Is_Not_17) is a valid representation for NaN in base 17.

6.1.6.2 mpfr output

char * mpfr_get_str (char *str, mpfr exp t *expptr, int b, size t n, mpfr t op, mpfr rnd t rnd) Convert op to a string of digits in base b, with rounding in the direction rnd, where n is either zero (see below) or the number of signifi cant digits output in the string; in the latter case, n must be greater or equal to 2. The base may vary from 2 to 62. If the input number is an

ordinary number, the exponent is written through the pointer expptr (for input 0, the current minimal exponent is written). The generated string is a fraction, with an implicit radix point immediately to the left of the fi rst digit. For example, the number âĹŠ3.1416 would be returned as "âĹŠ31416" in the string and 1 written at expptr. If rnd is to nearest, and op is exactly in the middle of two consecutive possible outputs, the one with an even signifi cand is chosen, where both signifi cands are considered with the exponent of op. Note that for an odd base, this may not correspond to an even last digit: for example with 2 digits in base 7, (14) and a half is rounded to (15) which is 12 in decimal, (16) and a half is rounded to (20) which is 14 in decimal, and (26) and a half is rounded to (26) which is 20 in decimal. If n is zero, the number of digits of the signifi cand is chosen large enough so that re-reading the printed value with the same precision, assuming both output and input use rounding to nearest, will recover the original value of op. More precisely, in most cases, the chosen precision of str is the minimal precision m depending only on p = PREC(op) and b that satisfi es the above property, i.e., m = 1 + dp log 2 log b e, with p replaced by pâĹŠ1 if b is a power of 2, but in some very rare cases, it might be m + 1 (the smallest case for bases up to 62 is when p equals 186564318007 for bases 7 and 49). If str is a null pointer, space for the signifi cand is allocated using the current allocation function, and a pointer to the string is returned. To free the returned string, you must use mpfr_free_str. If str is not a null pointer, it should point to a block of storage large enough for the signifi cand, i.e., at least max(n + 2, 7). The extra two bytes are for a possible minus sign, and for the terminating null character, and the value 7 accounts for -@Inf@ plus the terminating null character. A pointer to the string is returned, unless there is an error, in which case a null pointer is returned.

6.2 Constants

6.2.1 Log2

Function **ConstLog2**() As mpNum

The function ConstLog2 returns the value of the natural logarithm of 2, $\ln(2) = 0.69314718055994....$ Implemented in double, MPFR and MPFI.

6.2.2 Pi

Function **Pi**() As mpNum

The function Pi returns the value of $\pi = 3.1415926535897932....$

6.2.3 Catalan

Function **Catalan**() As mpNum

The function Catalan returns the value of Catalan's constant, $G = 0.9159655941772190....$ Implemented in double, MPFR and MPFI.

6.2.4 Euler's Gamma

Function **EulerGamma**() As mpNum

The function EulerGamma returns the value of Euler's Gamma, $\gamma = 0.57721566490153286....$

Implemented in double, MPFR and MPFI.

6.2.5 Machine Epsilon

Function **MachineEpsilon**() As mpNum

The function MachineEpsilon returns the value of the Machine Epsilon in the current precision

6.2.6 MaxReal

Function **MaxReal**() As mpNum

The function MaxReal returns the value of the largest representable real number in the current precision.
Implemented in double, MPFR, MPFI.

6.2.7 MaxInteger

Function **MaxInteger**() As mpNum

The function MaxInteger returns the value of the largest representable integer in the current precision.
Implemented in double, MPFR, MPFI.

6.2.8 MinReal

Function **MinReal**() As mpNum

The function MinReal returns the value of the smallest representable positive real number in the current precision.
Implemented in double, MPFR, MPFI.

6.2.9 MinInteger

Function **MinInteger**() As mpNum

The function MinInteger returns the value of the smallest representable positive integer in the current precision.
Implemented in double, MPFR, MPFI.

6.2.10 Positive Infinity

Function **PosInf**() As mpNum

The function PosInf returns the value of the representation of $+\infty$ in the current precision.
Implemented in double, MPFR, MPFI.

6.2.11 Negative Infinity

Function **NegInf**() As mpNum

The function NegInf returns the value of the representation of $-\infty$ in the current precision.
Implemented in double, MPFR, MPFI.

6.2.12 Not-a-Number: NaN

Function **NaN**() As mpNum

The function NaN returns the value of the representation of Not a Number (NaN) in the current precision.
Implemented in double, MPFR, MPFI.

6.3 Sign, Powers and Roots

6.3.1 Sign

Function **Sign**(***x*** *As mpNum*) As mpNum

The function Sign returns the value of the sign of $x, \text{sign}(x)$.

Parameter:
x: A real number.

6.3.2 Copysign

Function **Copysign**(***x*** *As mpNum,* ***y*** *As mpNum*) As mpNum

The function Copysign returns $|x| \cdot \text{sign(y)}$.

Parameters:
x: A real number.
y: A real number.
Implemented in double, MPFR and MPFI.

6.3.3 Absolute Value: $|x| = \sqrt{x^2}$

Function **Abs**(***x*** *As mpNum*) As mpNum

The function Abs returns the absolute value of x, $|x| = \sqrt{x^2}$.

Parameter:
x: A real number.

6.3.4 Reciprocal: $1/x = x^{-1}$

Function **Reci**(***x*** *As mpNum*) As mpNum

The function Reci returns the absolute value of the reciprocal of $x, 1/x = x^{-1}$

Parameter:
x: A real number.
Implemented in double, MPFR and MPFI.

6.3.5 Square: x^2

Function **Square**(***x*** *As mpNum*) As mpNum

The function Square returns the absolute value of the square of x, x^2.

Parameter:
x: A real number.
Implemented in double, MPFR and MPFI.

6.3.6 Power Function with Integer Exponent: $x^k, k \in \mathbb{Z}$

Function **Power_k**(***x*** *As mpNum,* ***k*** *As mpNum*) As mpNum

The function Power_k returns the value of $x^k, k \in \mathbb{Z}$

Parameters:
x: A real number.
k: An integer.
Implemented in double, MPFR and MPFI.

6.3.7 Power Function with Real Exponent: $x^y, y \in \mathbb{R}$

Function **Power**(***x*** *As mpNum,* ***y*** *As mpNum*) As mpNum

The function Power returns the value of $x^y, y \in \mathbb{R}$.

Parameters:
x: A real number.
y: A real number.

6.3.8 Auxiliary Function $x^y - 1$

Function **Powm1**(***x*** *As mpNum,* ***y*** *As mpNum*) As mpNum

The function Powm1 returns the value of $x^y - 1, y \in \mathbb{R}$.

Parameters:
x: A real number.
y: A real number.
Implemented in double, MPFR and MPFI.

6.3.9 Auxiliary Function $x^2 + y^2$

Function **X2pY2**(***x*** *As mpNum*, ***y*** *As mpNum*) As mpNum

The function X2pY2 returns the value of $x^2 + y^2$.

Parameters:
x: A real number.
y: A real number.
Implemented in double, MPFR and MPFI.

6.3.10 Auxiliary Function $x^2 - y^2$

Function **X2mY2**(***x*** *As mpNum*, ***y*** *As mpNum*) As mpNum

The function X2mY2 returns the value of $x^2 - y^2$.

Parameters:
x: A real number.
y: A real number.
Implemented in double, MPFR and MPFI.

6.3.11 Square Root: $\sqrt{x}$

Function **Sqrt**(***x*** *As mpNum*) As mpNum

The function Sqrt returns the absolute value of the square root of x, $\sqrt{x}$.

Parameter:
x: A real number.

6.3.12 Square Root of a nonnegative Integer: $\sqrt{n}, n \in \mathbb{N}$

Function **Sqrt_n**(***x*** *As mpNum*) As mpNum

The function Sqrt_n returns the absolute value of the square root of a nonnegative Integer$n$, $\sqrt{n}$.

Parameter:
x: An integer.
Implemented in double, MPFR and MPFI.

6.3.13 Reciprocal Square Root: $1/\sqrt{x}$

Function **ReciSqrt**(***x*** *As mpNum*) As mpNum

The function ReciSqrt returns the absolute value of the reciprocal square root of x, $\sqrt{x}$.

Parameter:
x: A real number.
Implemented in double, MPFR and MPFI.

6.3.14 Cube Root: $\sqrt[3]{x}$

Function **Cbrt**(***x*** *As mpNum*) As mpNum

The function Cbrt returns the absolute value of the cube root of x, $\sqrt[3]{x}$.

Parameter:
x: A real number.
Implemented in double, MPFR and MPFI.

6.3.15 Auxiliary Function $\sqrt{x+1}-1$

Function **Sqrtp1m1**(***x*** *As mpNum*) As mpNum

The function Sqrtp1m1 returns the value of $\sqrt{x+1}-1$.

Parameter:
x: A real number.
Implemented in double, MPFR and MPFI.

6.3.16 Auxiliary Function $\sqrt{1+x^2}$

Function **Sqrt1px2**(***x*** *As mpNum*) As mpNum

The function Sqrt1px2 returns the value of $\sqrt{1+x^2}$.

Parameter:
x: A real number.
Implemented in double, MPFR and MPFI.

6.3.17 Auxiliary Function $\sqrt{1-x^2}$

Function **Sqrt1mx2**(***x*** *As mpNum*) As mpNum

The function Sqrt1mx2 returns the value of $\sqrt{1-x^2}$.

Parameter:
x: A real number.
Implemented in double, MPFR and MPFI.

6.3.18 Auxiliary Function $\sqrt{x^2-1}$

Function **Sqrtx2m1**(***x*** *As mpNum*) As mpNum

The function Sqrtx2m1 returns the value of $\sqrt{x^2-1}$.

Parameter:
x: A real number.
Implemented in double, MPFR and MPFI.

6.3.19 Auxiliary Function $\sqrt{x^2+y^2}$

Function **Hypot**(***x*** *As mpNum,* ***y*** *As mpNum*) As mpNum

The function Hypot returns the value of $\sqrt{x^2+y^2}$.

Parameters:
x: A real number.
y: A real number.
Implemented in double, MPFR and MPFI.

6.3.20 Nth Root: $\sqrt[n]{x}, n = 2, 3, ...$

Function **NthRoot**(***n*** *As mpNum,* ***y*** *As mpNum*) As mpNum

The function NthRoot returns the value of the n^{th} root of x, $\sqrt[n]{x}, n = 2, 3, ...$

Parameters:
n: An integer.
y: A real number.
Implemented in double, MPFR and MPFI.

6.4 Exponential, Logarithmic, and Lambert Functions

6.4.1 Exponential Function $e^x = \exp(x)$

Function **Exp**(***x*** *As mpNum*) As mpNum

The function Exp returns the value of the exponential function, $\exp(x) = e^x = \exp(x)$.

Parameter:
x: A real number.

6.4.2 Exponential Function $10^x = \exp_{10}(x)$

Function **Exp10**(***x*** *As mpNum*) As mpNum

The function Exp10 returns the value of the exponential function, $\text{exp10}(x) = 10^x = \exp_{10}(x)$.

Parameter:
x: A real number.
Implemented in double, MPFR and MPFI.

6.4.3 Exponential Function $2^x = \exp_2(x)$

Function **Exp2**(***x*** *As mpNum*) As mpNum

The function Exp2 returns the value of the exponential function, $\text{exp2}(x) = 2^x = \exp_2(x)$.

Parameter:

x: A real number.
Implemented in double, MPFR and MPFI.

6.4.4 Auxiliary Function $e^x - 1$

Function **Expm1**(***x*** *As mpNum*) As mpNum

The function Expm1 returns the value of the function $\mathrm{expm1}(x) = e^x - 1$.

Parameter:
x: A real number.
Implemented in double, MPFR and MPFI.

6.4.5 Auxiliary Function e^{x^2}

Function **Expx2**(***x*** *As mpNum*) As mpNum

The function Expx2 returns the value of the function $\mathrm{expx2}(x) = e^{x^2}$.

Parameter:
x: A real number.
Implemented in double, MPFR and MPFI.

6.4.6 Auxiliary Function $e^{x^2} - 1$

Function **Expx2m1**(***x*** *As mpNum*) As mpNum

The function Expx2m1 returns the value of the function $\mathrm{expx2m1}(x) = e^{x^2} - 1$.

Parameter:
x: A real number.
Implemented in double, MPFR and MPFI.

6.4.7 Auxiliary Function e^{-x^2}

Function **Expmx2**(***x*** *As mpNum*) As mpNum

The function Expmx2 returns the value of the function $\mathrm{expmx2}(x) = e^{-x^2}$.

Parameter:
x: A real number.
Implemented in double, MPFR and MPFI.

6.4.8 Auxiliary Function $e^{-x^2} - 1$

Function **Expmx2m1**(***x*** *As mpNum*) As mpNum

The function Expmx2m1 returns the value of the function $\mathrm{expmx2m1}(x) = e^{-x^2} - 1$.

Parameter:

x: A real number.
Implemented in double, MPFR and MPFI.

6.4.9 Natural logarithm $\ln(x) = \log_e(x)$

Function **Ln**(***x*** *As mpNum*) As mpNum

The function Ln returns the value of the natural logarithm $\ln(x) = \log_e(x)$.

Parameter:
x: A real number.
Implemented in double, MPFR and MPFI.

6.4.10 Auxiliary Function $\ln(1 + x)$

Function **Lnp1**(***x*** *As mpNum*) As mpNum

The function Lnp1 returns the value of the function $\ln(1 + x)$.

Parameter:
x: A real number.
Implemented in double, MPFR and MPFI.

6.4.11 Common (decadic) logarithm $\log_{10}(x)$

Function **Log10**(***x*** *As mpNum*) As mpNum

The function Log10 returns the value of the decadic logarithm $\log10(x) = \log_{10}(x)$.

Parameter:
x: A real number.

6.4.12 Binary logarithm $\log_2(x)$

Function **Log2**(***x*** *As mpNum*) As mpNum

The function Log2 returns the value of the binary logarithm $\log2(x) = \log_2(x)$.

Parameter:
x: A real number.

6.4.13 Logarithm to base b: $\log_b(x)$

Function **Log**(***x*** *As mpNum*) As mpNum

The function Log returns the value of the logarithm to base b: $\log b(x) = \log_b(x)$.

Parameter:
x: A real number.

6.4.14 Auxiliary Function $\ln(\cos(x))$

Function **LnCos**(***x*** *As mpNum*) As mpNum

The function LnCos returns the value of the logarithm of the cosine of x: $\mathrm{LnCos}(x) = \ln(\cos(x))$.

Parameter:
x: A real number.

6.4.15 Auxiliary Function $\ln(\sin(x))$

Function **LnSin**(***x*** *As mpNum*) As mpNum

The function LnSin returns the value of the logarithm of the sine of x: $\mathrm{LnSin}(x) = \ln(\sin(x))$

Parameter:
x: A real number.
Implemented in double, MPFR and MPFI.

6.4.16 Auxiliary Function $\ln\left(\sqrt{x^2+y^2}\right)$

Function **LnSqrtx2y2**(***x*** *As mpNum*, ***y*** *As mpNum*) As mpNum

The function LnSqrtx2y2 returns the value of the function $\mathrm{LnSqrtx2y2}(x) = \ln\left(\sqrt{x^2+y^2}\right)$.

Parameters:
x: A real number.
y: A real number.

6.4.17 Auxiliary Function $\ln\left(\sqrt{(x+1)^2+y^2}\right)$

Function **LnSqrtxp1T2y2**(***x*** *As mpNum*, ***y*** *As mpNum*) As mpNum

The function LnSqrtxp1T2y2 returns the value of the function $\mathrm{LnSqrtxp1T2y2}(x) = \ln\left(\sqrt{(x+1)^2+y^2}\right)$.

Parameters:
x: A real number.
y: A real number.

6.4.18 Lambert Functions $W_0(x)$ and $W_{-1}(x)$

The multivalued Lambert W function is defined as a solution of

$$W(x)e^{W(x)} = x. \tag{6.4.1}$$

This function has two real branches for $x < 0$ with a branch point at $x = -1/e$.
$\mathrm{LambertW0}(x) = W_0(x)$ is the principal branch with $W_0(x) \geq -1$ for $x < 0$, and
$\mathrm{LambertWm1}(x) = W_{-1}(x)$ is the other real branch with $W_{-1}(x) \leq -1$ for $x < 0$.

Function **LambertW0**(***x*** *As mpNum*) As mpNum

The function LambertW0 returns the value of the Lambert functions $W_0(x)$

Parameter:
x: A real number.

Function **LambertWm1**(***x*** *As mpNum*) As mpNum

The function LambertWm1 returns the value of the Lambert functions $W_{-1}(x)$

Parameter:
x: A real number.
See Corless *et al.* (1996)

6.5 Trigonometric Functions

6.5.1 Sine: $\sin(x)$

Function **Sin**(***x*** *As mpNum*) As mpNum

The function Sin returns the value of the sine of x, with x in radians.

Parameter:
x: A real number.

Function **SinDeg**(***x*** *As mpNum*) As mpNum

The function SinDeg returns the value of the sine of x, with x in degrees

Parameter:
x: A real number.

6.5.2 Cosine: $\cos(x)$

Function **Cos**(***x*** *As mpNum*) As mpNum

The function Cos returns the value of the cosine of x, with x in radians.

Parameter:
x: A real number.

Function **CosDeg**(***x*** *As mpNum*) As mpNum

The function CosDeg returns the value of the cosine of x, with x in degrees

Parameter:
x: A real number.

6.5.3 Tangent: $\tan(x)$

Function **Tan**(***x*** *As mpNum*) As mpNum

The function Tan returns the value of the tangent of x, with x in radians.

Parameter:
x: A real number.

Function **TanDeg**(***x*** *As mpNum*) As mpNum

The function TanDeg returns the value of the tangent of x, with x in degrees

Parameter:
x: A real number.

6.5.4 Cosecant: $\csc(x) = 1/\sin(x)$

Function **Csc**(***x*** *As mpNum*) As mpNum

The function Csc returns the value of the cosecant of x, with x in radians.

Parameter:
x: A real number.

Function **CscDeg**(***x*** *As mpNum*) As mpNum

The function CscDeg returns the value of the cosecant of x, with x in degrees

Parameter:
x: A real number.

6.5.5 Secant: $\sec(x) = 1/\cos(x)$

Function **Sec**(***x*** *As mpNum*) As mpNum

The function Sec returns the value of the secant of x, with x in radians.

Parameter:
x: A real number.

Function **SecDeg**(***x*** *As mpNum*) As mpNum

The function SecDeg returns the value of the secant of x, with x in degrees

Parameter:
x: A real number.

6.5.6 Cotangent: $\cot(x) = 1/\tan(x)$

Function **Cot**(***x*** *As mpNum*) As mpNum

The function Cot returns the value of the cotangent of x, with x in radians.

Parameter:
x: A real number.

Function **CotDeg**(***x*** *As mpNum*) As mpNum

The function CotDeg returns the value of the cotangent of x, with x in degrees

Parameter:
x: A real number.

6.5.7 Sinus Cardinal: $\mathrm{Sinc}_a(x)$

Function **Sinca**(***x*** *As mpNum*) As mpNum

The function Sinca returns the sinus cardinal function

Parameter:
x: A real number.

The sinus cardinal function is defined as

$$\mathrm{sinc}_a(x) = \sin\left(\frac{\pi x}{a}\right)\frac{a}{\pi x} \tag{6.5.1}$$

Implemented in double, MPFR and MPFI.

6.5.8 Hyperbolic Sine: $\sinh(x)$

Function **Sinh**(***x*** *As mpNum*) As mpNum

The function Sinh returns the value of the hyperbolic sine of x, with x in radians.

Parameter:
x: A real number.

Function **SinhDeg**(***x*** *As mpNum*) As mpNum

The function SinhDeg returns the value of the hyperbolic sine of x, with x in degrees

Parameter:
x: A real number.

6.5.9 Hyperbolic Cosine: $\cosh(x)$

Function **Cosh**(***x*** *As mpNum*) As mpNum

The function Cosh returns the value of the hyperbolic cosine of x, with x in radians.

Parameter:
x: A real number.

Function **CoshDeg**(***x*** *As mpNum*) As mpNum

The function CoshDeg returns the value of the hyperbolic cosine of x, with x in degrees

Parameter:
x: A real number.

6.5.10 Hyperbolic Tangent: $\tanh(x)$

Function **Tanh**(***x*** *As mpNum*) As mpNum

The function Tanh returns the value of the hyperbolic cosine of x, with x in radians.

Parameter:
x: A real number.

Function **TanhDeg**(***x*** *As mpNum*) As mpNum

The function TanhDeg returns the value of the hyperbolic cosine of x, with x in degrees

Parameter:
x: A real number.

6.5.11 Hyperbolic Cosecant: $\mathbf{csch}(x) = 1/\sinh(x)$

Function **Csch**(***x*** *As mpNum*) As mpNum

The function Csch returns the value of the hyperbolic cosecant of x, with x in radians.

Parameter:
x: A real number.

Function **CschDeg**(***x*** *As mpNum*) As mpNum

The function CschDeg returns the value of the hyperbolic cosecant of x, with x in degrees

Parameter:
x: A real number.

6.5.12 Hyperbolic Secant: $\mathbf{sech}(x) = 1/\cosh(x)$

Function **Sech**(***x*** *As mpNum*) As mpNum

The function Sech returns the value of the hyperbolic cosecant of x, with x in radians.

Parameter:
x: A real number.

Function **SechDeg**(***x*** *As mpNum*) As mpNum

The function SechDeg returns the value of the hyperbolic cosecant of x, with x in degrees

Parameter:
x: A real number.

6.5.13 Hyperbolic Cotangent: $\coth(x) = 1/\tanh(x)$

Function **Coth**(***x*** *As mpNum*) As mpNum

The function Coth returns the value of the hyperbolic cotangent of x, with x in radians.

Parameter:
x: A real number.

Function **CothDeg**(***x*** *As mpNum*) As mpNum

The function CothDeg returns the value of the hyperbolic cotangent of x, with x in degrees

Parameter:
x: A real number.

6.5.14 Hyperbolic Sinus Cardinal: $\mathbf{Sinhc}_a(x)$

Function **Sinhca**(***x*** *As mpNum*) As mpNum

The function Sinhca returns the hyperbolic sinus cardinal function.

Parameter:
x: A real number.

The hyperbolic sinus cardinal function is defined as

$$\operatorname{sinhc}_a(x) = \sinh\left(\frac{\pi x}{a}\right)\frac{a}{\pi x} \tag{6.5.2}$$

Implemented in double, MPFR and MPFI.

6.6 Inverse Trigonometric Functions

6.6.1 Arc-sine: $\mathbf{asin}(x)$

Function **Asin**(***x*** *As mpNum*) As mpNum

The function Asin returns the value of the arc-sine of x in radians.

Parameter:
x: A real number.

Function **AsinDeg**(***x*** *As mpNum*) As mpNum

The function AsinDeg returns the value of the arc-sine of x in degrees

Parameter:
x: A real number.

6.6.2 Arc-cosine: $\mathbf{acos}(x)$

Function **Acos**(***x*** *As mpNum*) As mpNum

The function Acos returns the value of the arc-cosine of x in radians.

Parameter:
x: A real number.

Function **AcosDeg**(***x*** *As mpNum*) As mpNum

The function AcosDeg returns the value of the arc-cosine of x in degrees

Parameter:
x: A real number.

6.6.3 Arc-tangent: atan(x)

Function **Atan**(***x*** *As mpNum*) As mpNum

The function Atan returns the value of the arc-tangent of x in radians.

Parameter:
x: A real number.

Function **AtanDeg**(***x*** *As mpNum*) As mpNum

The function AtanDeg returns the value of the arc-tangent of x in degrees

Parameter:
x: A real number.

6.6.4 Arc-tangent, version with 2 arguments: atan2(x, y)

Function **Atan2**(***x*** *As mpNum*, ***y*** *As mpNum*) As mpNum

---

The function Atan2 returns the value of the arc-tangent of x in radians.

Parameters:
x: A real number.
y: A real number.

Function **Atan2Deg**(***x*** *As mpNum*, ***y*** *As mpNum*) As mpNum

---

The function Atan2Deg returns the value of the arc-tangent of x in degrees

Parameters:
x: A real number.
y: A real number.

6.6.5 Arc-cotangent: acot(x)

Function **Acot**(***x*** *As mpNum*) As mpNum

The function Acot returns the value of the arc-cotangent of x in radians.

Parameter:

x: A real number.

Function **AcotDeg**(***x*** *As mpNum*) As mpNum

The function AcotDeg returns the value of the arc-cotangent of x in degrees

Parameter:
x: A real number.

6.6.6 Hyperbolic Arc-sine: asinh(x)

Function **Asinh**(***x*** *As mpNum*) As mpNum

The function Asinh returns the value of the hyperbolic arc-sine of x in radians.

Parameter:
x: A real number.

Function **AsinhDeg**(***x*** *As mpNum*) As mpNum

The function AsinhDeg returns the value of hyperbolic arc-sine of x in degrees

Parameter:
x: A real number.

6.6.7 Hyperbolic Arc-cosine: acosh(x)

Function **Acosh**(***x*** *As mpNum*) As mpNum

The function Acosh returns the value of the hyperbolic arc-cosine of x in radians.

Parameter:
x: A real number.

Function **AcoshDeg**(***x*** *As mpNum*) As mpNum

The function AcoshDeg returns the value of hyperbolic arc-cosine of x in degrees

Parameter:
x: A real number.

6.6.8 Hyperbolic Arc-tangent: atanh(x)

Function **Atanh**(***x*** *As mpNum*) As mpNum

The function Atanh returns the value of the hyperbolic arc-tangent of x in radians.

Parameter:
x: A real number.

Function **AtanhDeg**(***x*** *As mpNum*) As mpNum

The function AtanhDeg returns the value of hyperbolic arc-tangent of x in degrees

Parameter:
x: A real number.

6.6.9 Hyperbolic Arc-cotangent: acoth(x)

Function **Acoth**(***x*** *As mpNum*) As mpNum

The function Acoth returns the value of the hyperbolic arc-cotangent of x in radians.

Parameter:
x: A real number.

Function **AcothDeg**(***x*** *As mpNum*) As mpNum

The function AcothDeg returns the value of hyperbolic arc-cotangent of x in degrees

Parameter:
x: A real number.

6.7 Elementary Functions of Mathematical Physics

6.7.1 Bessel Function $J_0(x)$

Function **BesselJ0**(***x*** *As mpNum*) As mpNum

The function BesselJ0 returns $J_0(x)$, the Bessel function of the 1st kind, order zero.

Parameter:
x: A real number.

6.7.2 Bessel Function $J_1(x)$

Function **BesselJ1**(***x*** *As mpNum*) As mpNum

The function BesselJ1 returns $J_1(x)$, the Bessel function of the 1st kind, order one.

Parameter:
x: A real number.

6.7.3 Bessel Function $J_n(x)$

Function **BesselJn**(***x*** *As mpNum*, ***n*** *As mpNum*) As mpNum

The function BesselJn returns $J_n(x)$, the Bessel function of the 1st kind, order n.

Parameters:
x: A real number.
n: An Integer.

The Bessel function of the 1st kind, order n is defined as

$$J_n(x) = \left(\tfrac{1}{2}x\right)^n \sum_{k=0}^{\infty} (-1)^k \frac{\left(\tfrac{1}{4}x^2\right)^k}{k!(n+k)!}, \quad J_{-n}(x) = (-1)^n J_n(x). \tag{6.7.1}$$

6.7.4 Bessel Function $Y_0(x)$

Function **BesselY0**(***x*** *As mpNum*) As mpNum

The function BesselY0 returns $Y_0(x)$, the Bessel function of the second kind, order zero.

Parameter:
x: A real number.

6.7.5 Bessel Function $Y_1(x)$

Function **BesselY1**(***x*** *As mpNum*) As mpNum

The function BesselY1 returns $Y_1(x)$, the Bessel function of the second kind, order one.

Parameter:
x: A real number.

6.7.6 Bessel Function $Y_n(x)$

Function **BesselYn**(***x*** *As mpNum*, ***n*** *As mpNum*) As mpNum

The function BesselYn returns $Y_n(x)$, the Bessel function of the second kind, order n.

Parameters:
x: A real number.
n: An Integer.

The Bessel function of the second kind of order n is defined as

$$Y_n(x) = \frac{J_n(x)\cos(n\pi) - J_{-n}(x)}{\sin(n\pi)} \tag{6.7.2}$$

6.7.7 Error Function erf

Function **Erf**(***x*** *As mpNum*) As mpNum

The function Erf returns the value of the error function.

Parameter:
x: A real number.

The error function is defined by

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-x^2} dt, \tag{6.7.3}$$

6.7.8 Complementary Error Function

Function **Erfc**(***x*** *As mpNum*) As mpNum

The function Erfc returns the value of the complementary error function.

Parameter:
x: A real number.

The complementary error function is defined by

$$\operatorname{erfc}(x) = 1 - \operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-x^2} dt, \tag{6.7.4}$$

6.7.9 Gamma function $\Gamma(x)$

Function **Tgamma**(***x*** *As mpNum*) As mpNum

The function Tgamma returns the gamma function for $x \neq 0, -1, -2, \ldots$.

Parameter:
x: A real number.

The gamma function for $x \neq 0, -1, -2, \ldots$ is defined by

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt \quad (x > 0), \tag{6.7.5}$$

and by analytic continuation if $x < 0$, using the reflection formula

$$\Gamma(x)\Gamma(1 - x) = \pi / \sin(\pi x). \tag{6.7.6}$$

Function **Lgamma**(***x*** *As mpNum*) As mpNum

The function Lgamma returns the logarithm of the gamma function.

Parameter:
x: A real number.

This function computes $\ln|\Gamma(x)|$ for $x \neq 0, -1, -2, \ldots$. If $x < 0$ the function uses the logarithm of the reflection formula.

6.7.10 Pochhammer symbol

Function **Pochhammer**(***a*** *As mpNum,* ***x*** *As mpNum*) As mpNum

The function Pochhammer returns the Pochhammer symbol.

Parameters:
a: An integer.
x: An integer.

The Pochhammer symbol is defined as

$$(a)_x = \frac{\Gamma(a + x)}{\Gamma(a)}. \tag{6.7.7}$$

In the special case that $x = n$ is a positive integer, $(a)_n = a(a+1)(a+2)\cdots(a+n-1)$ is often called "'rising factorial"'. By convention $(a)_0 = 1$.

6.7.11 Beta Function B(a,b)

Function **Beta**(***a*** *As mpNum,* ***b*** *As mpNum*) As mpNum

The function Beta returns the Beta function.

Parameters:
a: A real number.
b: A real number.

This function computes $B(a,b)$ for $a, b \neq 0, -1, -2, \ldots$.

6.7.12 Logarithm of B(a, b)

Function **LnBetaBoost**(***a*** *As mpNum,* ***b*** *As mpNum*) As mpNum

The function LnBetaBoost returns the logarithm of the beta function $\ln B(a,b)|$ with $a, b \neq 0, -1, -2, \ldots$.

Parameters:
a: A real number.
b: A real number.
The alogorithm is implemented as in DiDonato & Morris (1986)

6.7.13 Normalised incomplete beta functions

Function **IBetaBoost**(***a*** *As mpNum,* ***b*** *As mpNum,* ***x*** *As mpNum*) As mpNum

The function IBetaBoost returns the normalised incomplete beta function.

Parameters:
a: A real number.
b: A real number.
x: A real number.

This function returns the normalised incomplete beta function $I_x(a,b)$ for $a > 0$, $b > 0$, and $0 \leq x \leq 1$:

$$I_x(a,b) = \frac{B_x(a,b)}{B(a,b)}, \quad B_x(a,b) = \int_0^x t^{a-1}(1-t)^{b-1}dt. \tag{6.7.8}$$

There are some special cases

$$I_0(a,b) = 0, \quad I_1(a,b) = 1, \quad I_x(a,1) = x^a, \tag{6.7.9}$$

and the symmetry relation $I_x(a,b) = 1 - I_{1-x}(b,a)$, which is used for $x > a/(a+b)$.

Function **IBetacBoost**(***a*** *As mpNum,* ***b*** *As mpNum,* ***x*** *As mpNum*) As mpNum

The function IBetacBoost returns the normalised complement of the incomplete beta function, $1 - I_x(a, b)$.

Parameters:
a: A real number.
b: A real number.
x: A real number.

6.7.14 Non-Normalised incomplete beta functions

Function **IBetaNonNormalizedBoost**(***a*** *As mpNum,* ***b*** *As mpNum,* ***x*** *As mpNum*) As mpNum

The function IBetaNonNormalizedBoost returns the non-normalised incomplete beta function.

Parameters:
a: A real number.
b: A real number.
x: A real number.

This function returns the non-normalised incomplete beta function $B_x(a, b)$ for $a > 0$, $b > 0$, and $0 \leq x \leq 1$:

$$B_x(a, b) = \int_0^x t^{a-1}(1 - t)^{b-1} dt. \tag{6.7.10}$$

There are some special cases

$$B_0(a, b) = 0, \quad B_1(a, b) = B(a, b), \quad B_x(a, 1) = \frac{x^a}{a}, \quad B_x(1, b) = \frac{1 - (1 - x)^b}{b}, \tag{6.7.11}$$

and the relation $B_{1-x}(a, b) = B(a, b) - B_x(b, a)$, which is used if $x > a/(a + b)$.

Function **IBetacNonNormalizedBoost**(***a*** *As mpNum,* ***b*** *As mpNum,* ***x*** *As mpNum*) As mpNum

The function IBetacNonNormalizedBoost returns the non-normalised complement of the incomplete beta function, $1 - B_x(a, b)$.

Parameters:
a: A real number.
b: A real number.
x: A real number.

6.7.15 Inverse normalised incomplete beta functions

Function **IBetaInvBoost**(***a*** *As mpNum,* ***b*** *As mpNum,* ***p*** *As mpNum*) As mpNum

The function IBetaInvBoost returns the inverse of the normalised incomplete beta function $I_x(a, b)$.

Parameters:
a: A real number.
b: A real number.
p: A real number.

This function calculates x such that $I_x(a, b) = p$. The input parameters are $a, b > 0$, $p \geq 0$, and $p + q = 1$.

Function **IBetacInvBoost**(***a*** *As mpNum*, ***b*** *As mpNum*, ***q*** *As mpNum*) As mpNum

---

The function IBetacInvBoost returns the inverse of the complement of the normalised incomplete beta function $1 - I_x(a, b)$.

Parameters:
a: A real number.
b: A real number.
q: A real number.

This function calculates x such that $1 - I_x(a, b) = q$. The input parameters are $a, b > 0$, $q \geq 0$, and $p + q = 1$.

Function **IBetaInvaBoost**(***x*** *As mpNum*, ***b*** *As mpNum*, ***p*** *As mpNum*) As mpNum

---

The function IBetaInvaBoost returns the parameter a of the normalised incomplete beta function $I_x(a, b)$, such that $I_x(a, b) = p$.

Parameters:
x: A real number.
b: A real number.
p: A real number.

Function **IBetacInvaBoost**(***x*** *As mpNum*, ***b*** *As mpNum*, ***q*** *As mpNum*) As mpNum

---

The function IBetacInvaBoost returns the parameter a of the complement of the normalised incomplete beta function $1 - I_x(a, b)$, such that $1 - I_x(a, b) = q$.

Parameters:
x: A real number.
b: A real number.
q: A real number.

Function **IBetaInvbBoost**(***x*** *As mpNum*, ***a*** *As mpNum*, ***p*** *As mpNum*) As mpNum

---

The function IBetaInvbBoost returns the parameter b of the normalised incomplete beta function $I_x(a, b)$, such that $I_x(a, b) = p$.

Parameters:
x: A real number.
a: A real number.
p: A real number.

Function **IBetacInvbBoost**(***x*** *As mpNum*, ***a*** *As mpNum*, ***q*** *As mpNum*) As mpNum

---

The function IBetacInvbBoost returns the parameter b of the complement of the normalised incomplete beta function $1 - I_x(a, b)$, such that $1 - I_x(a, b) = q$.

Parameters:
x: A real number.
a: A real number.
q: A real number.

6.7.16 Derivative of the Normalised Incomplete beta Function

Function **IBetaDerivativeBoost**(***x*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum

The function IBetaDerivativeBoost returns the partial derivative with respect to x of the incomplete beta function.

Parameters:
x: A real number.
a: A real number.
b: A real number.

The partial derivative with respect to x of the incomplete beta function is defined as:

$$\frac{\partial}{\partial x} I_x(a,b) = \frac{(1-x)^{b-1}x^{a-1}}{B(a,b)}. \tag{6.7.12}$$

PLACEHOLDER: Incomplete Betafunction

6.7.17 Riemann $\zeta(s)$ function

Function **RiemannZeta**(***s*** *As mpNum*) As mpNum

The function RiemannZeta returns the Riemann zeta function.

Parameter:
s: A real number.

The Riemann zeta function $\zeta(s)$ for $s \neq 1$ is defined as

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s}, \quad s > 1. \tag{6.7.13}$$

If $s < 0$, the reflection formula is used:

$$\zeta(s) = 2(2\pi)^{s-1} \sin\left(\tfrac{1}{2}\pi s\right) \Gamma(1-s)\zeta(1-s) \tag{6.7.14}$$

6.7.18 Dilogarithm Function

Function **Dilogarithm**(***x*** *As mpNum*) As mpNum

The function Dilogarithm returns the dilogarithm function $\mathrm{Li}_2(x)$.

Parameter:
x: A real number.

This function returns the dilogarithm function

$$\mathrm{dilog}(x) = \Re \mathrm{Li}_2(x) = -\Re \int_0^x \frac{\ln(1-t)}{t} dt. \tag{6.7.15}$$

Note that there is some confusion about the naming: some authors and/or computer algebra systems use $\mathrm{dilog}(x) = \mathrm{Li}_2(1-x)$ and then call $\mathrm{Li}_2(x)$ Spence function/integral or similar.

6.8 Integer and Remainder Related Functions

6.8.1 Nearest integer: Round(x)

ROUND rounds to the nearest representable integer, rounding halfway cases away from zero (as in the roundTiesToAway mode of IEEE 754-2008).
The returned indicator value is zero when the result is exact, positive when it is greater than the original value of op, and negative when it is smaller. More precisely, the returned value is 0 when op is an integer representable in rop, 1 or -1 when op is an integer that is not representable in rop, 2 or -2 when op is not an integer.
Note that mpfr_round is different from mpfr_rint called with the rounding to nearest mode (where halfway cases are rounded to an even integer or significand). Note also that no double rounding is performed; for instance, 10.5 (1010.1 in binary) is rounded by mpfr_rint with rounding to nearest to 12 (1100 in binary) in 2-bit precision, because the two enclosing numbers representable on two bits are 8 and 12, and the closest is 12. (If one first rounded to an integer, one would round 10.5 to 10 with even rounding, and then 10 would be rounded to 8 again with even rounding.)

6.8.2 Next higher or equal integer: Ceil(x)

CEILING rounds to the next higher or equal representable integer.

6.8.3 Next lower or equal integer: Floor(x)

FLOOR rounds to the next lower or equal representable integer.

6.8.4 Next integer, rounded toward zero: Trunc(x)

TRUNC rounds to the next representable integer toward zero.

6.8.5 Nearest integer, rounded in a given direction: Rint(x)

Function **Rint**(***x*** *As mpNum*) As mpNum

The function Rint returns the rounded value of x.

Parameter:
x: A real number.RoundingMode? mpNum? An integer.

Rint rounds to the nearest representable integer in the given direction RoundingMode.

6.8.6 Nearest integer, followed by rint: RintRound(x)

Function **RintRound**(*x As mpNum*) As mpNum

The function RintRound returns the rounded value of x, rounded to the nearest integer, rounding halfway cases away from zero.

Parameter:
x: A real number.

mpfr_rint_round rounds to the nearest integer, rounding halfway cases away from zero.
If the result is not representable, it is rounded in the direction rnd. The returned value is the ternary value associated with the considered round-to-integer function (regarded in the same way as any other mathematical function).
Contrary to mpfr_rint, those functions do perform a double rounding: first op is rounded to the nearest integer in the direction given by the function name, then this nearest integer (if not representable) is rounded in the given direction rnd.
For example, mpfr_rint_round with rounding to nearest and a precision of two bits rounds 6.5 to 7 (halfway cases away from zero), then 7 is rounded to 8 by the round-even rule, despite the fact that 6 is also representable on two bits, and is closer to 6.5 than 8.

6.8.7 Next higher or equal integer, followed by rint: RintCeil(x)

Function **RintCeil**(*x As mpNum*) As mpNum

The function RintCeil returns the rounded value of x, rounded to the next higher or equal integer.

Parameter:
x: A real number.

mpfr_rint_ceil rounds to the next higher or equal integer.

6.8.8 Next lower or equal integer, followed by rint: RintFloor(x)

Function **RintFloor**(*x As mpNum*) As mpNum

The function RintFloor returns the rounded value of x, rounded to the next lower or equal integer.

Parameter:
x: A real number.

mpfr_rint_floor rounds to the next lower or equal integer.

6.8.9 Next integer, rounded toward zero, followed by Rint: RintTrunc(x)

Function **RintTrunc**(*x As mpNum*) As mpNum

The function RintTrunc returns the rounded value of x, rounded to the next integer toward zero.

Parameter:
x: A real number.

mpfr_rint_trunc rounds to the next integer toward zero.

6.8.10 Fractional Part: Frac(x)

Function **Frac**(***x*** *As mpNum*) As mpNum

The function Frac returns the fractional part of x

Parameter:
x: A real number.

Returns the fractional part of op, having the same sign as op, rounded in the direction rnd (unlike in mpfr_rint, rnd affects only how the exact fractional part is rounded, not how the fractional part is generated).

6.8.11 Next integer rounded toward zero, with fractional part: Modf(x)

Function **Modf**(***x*** *As mpNum*) As mpNumList

The function Modf returns simultaneously the integer and fractional part of x

Parameter:
x: A real number.

Set simultaneously iop to the integral part of op and fop to the fractional part of op, rounded in the direction rnd with the corresponding precision of iop and fop (equivalent to mpfr_trunc(iop, op, rnd) and mpfr_frac(fop, op, rnd)). The variables iop and fop must be different. Return 0 iff both results are exact (see mpfr_sin_cos for a more detailed description of the return value).

6.8.12 Floating Point Modulo: Fmod(x, y)

Function **Fmod**(***x*** *As mpNum*, ***y*** *As mpNum*) As mpNum

The function Fmod returns the remainder of x/y

Parameters:
x: A real number.
y: A real number.

Returns the value of $x - ny, \quad n = \lfloor x/y \rfloor$, i.e. rounded according to the direction rnd, where n is the integer quotient of x divided by y, rounded toward zero. See also section 6.8.14.

6.8.13 Floating Point Remainder: Remainder(x, y)

Function **Remainder**(***x*** *As mpNum*, ***y*** *As mpNum*) As mpNum

The function Remainder returns the remainder of x/y

Parameters:
x: A real number.
y: A real number.

Returns the value of $x - ny$, $n = \text{round}(x/y)$, i.e. rounded according to the direction rnd, where n is the integer quotient of x divided by y, rounded toward zero. See also section 6.8.14.

6.8.14 Remainder and Quotient: Remquo(*x*, *y*)

Function **Remquo**(***x*** *As mpNum*, ***y*** *As mpNum*) As mpNum

The function Remquo returns the remainder of x/y

Parameters:
x: A real number.
y: A real number.

Returns the value of $x - ny$, rounded according to the direction rnd, where n is the integer quotient of x divided by y, defined as follows: n is rounded toward zero for mpfr_fmod, and to the nearest integer (ties rounded to even) for mpfr_remainder and mpfr_remquo.

Special values are handled as described in Section F.9.7.1 of the ISO C99 standard: If x is infinite or y is zero, r is NaN. If y is infinite and x is finite, r is x rounded to the precision of r. If r is zero, it has the sign of x. The return value is the ternary value corresponding to r.

Additionally, mpfr_remquo stores the low significant bits from the quotient n in $*q$ (more precisely the number of bits in a long minus one), with the sign of x divided by y (except if those low bits are all zero, in which case zero is returned). Note that x may be so large in magnitude relative to y that an exact representation of the quotient is not practical. The mpfr_remainder and mpfr_remquo functions are useful for additive argument reduction.

6.8.15 INT(*x*)

Rounds a number down to the nearest integer.

6.9 Miscellaneous Functions

6.9.1 Next representable value from *x* toward y: Nexttoward(*x*, *y*)

Function **Nexttoward**(***x*** *As mpNum*, ***y*** *As mpNum*) As mpNum

The function Nexttoward returns the next floating-point number (with the precision of x and the current exponent range) in the direction of y

Parameters:
x: A real number.
y: A real number.

If x or y is NaN, set x to NaN. If x and y are equal, x is unchanged. Otherwise, if x is different from y, replace x by the next floating-point number (with the precision of x and the current exponent range) in the direction of y (the infinite values are seen as the smallest and largest floating-point numbers). If the result is zero, it keeps the same sign. No underflow or overflow is generated.

6.9.2 Next representable value above x: Nextabove(x)

Function **Nextabove**(***x*** *As mpNum*) As mpNum

The function Nextabove returns the next floating-point number (with the precision of x and the current exponent range) in the direction of plus infinity.

Parameter:
x: A real number.

Equivalent to Nexttoward where y is plus infinity.

6.9.3 Next representable value below x: Nextbelow(x)

Function **Nextbelow**(***x*** *As mpNum*) As mpNum

The function Nextbelow returns the next floating-point number (with the precision of x and the current exponent range) in the direction of minus infinity.

Parameter:
x: A real number.

Equivalent to Nexttoward where y is minus infinity.

6.9.4 Significand and Exponent: Frexp(x)

Function **Frexp**(***x*** *As mpNum*) As mpNumList

The function Frexp returns returns simultaneously significand and exponent of x

Parameter:
x: A real number.

Set exp (formally, the value pointed to by exp) and y such that $0.5 \leq |y| < 1$ and $y \times 2^{exp}$ equals x rounded to the precision of y, using the given rounding mode. If x is zero, then y is set to a zero of the same sign and exp is set to 0. If x is NaN or an infinity, then y is set to the same value and exp is undefined.

6.9.5 Number generated from Significand and Exponent: Ldexp(x, y)

Function **Ldexp**(***x*** *As mpNum*, ***y*** *As mpNum*) As mpNum

The function Ldexp returns $x \cdot 2^y$

Parameters:
x: A real number.
y: A real number.

Returns the result of multiplying x (the significand) by 2 raised to the power of y (the exponent): $\text{Ldexp}(x, y) = x \cdot 2^y$.

6.9.6 Fused-Multiply-Add Fma

Function **Fma**(***a*** *As mpNum*, ***b*** *As mpNum*, ***c*** *As mpNum*) As mpNum

The function Fma returns $(a \times b) + c$.

Parameters:
a: A real number.
b: A real number.
c: A real number.

This function returns $(a \times b) + c$.

6.9.7 Fused-Multiply-Subtract Fms

Function **Fms**(***a*** *As mpNum*, ***b*** *As mpNum*, ***c*** *As mpNum*) As mpNum

The function Fms returns $(a \times b) - c$.

Parameters:
a: A real number.
b: A real number.
c: A real number.

This function returns $(a \times b) - c$.

6.10 Numerical Information Functions

6.10.1 Infinity (positive or negative): IsInf(x)

Function **IsInf**(***x*** *As mpNum*) As Boolean

The function IsInf returns TRUE if x is infinity (positive or negative), and FALSE otherwise.

Parameter:
x: A real number.

6.10.2 Integer: IsInteger(x)

Function **IsInteger**(***x*** *As mpNum*) As Boolean

The function IsInteger returns TRUE if x is an integer, and FALSE otherwise.

Parameter:
x: A real number.

6.10.3 Not-a-Number: IsNan(x)

Function **IsNan**(***x*** *As mpNum*) As Boolean

The function IsNan returns TRUE if x is an NaN (Not a Number), and FALSE otherwise.

Parameter:
x: A real number.

6.10.4 Negative Number: IsNeg(x)

Function **IsNeg**(***x*** *As mpNum*) As Boolean

The function IsNeg returns TRUE if x is negative, and FALSE otherwise.

Parameter:
x: A real number.

6.10.5 Non-Negative Number: IsNonNeg(x)

Function **IsNonNeg**(***x*** *As mpNum*) As Boolean

The function IsNonNeg returns TRUE if $x \geq 0$, and FALSE otherwise.

Parameter:
x: A real number.

6.10.6 Non-Positive Number: IsNonPos(x)

Function **IsNonPos**(***x*** *As mpNum*) As Boolean

The function IsNonPos returns TRUE if $x \leq 0$, and FALSE otherwise.

Parameter:
x: A real number.

6.10.7 Positive Number: IsPos(x)

Function **IsPos**(***x*** *As mpNum*) As Boolean

The function IsPos returns TRUE if $x > 0$, and FALSE otherwise.

Parameter:
x: A real number.

6.10.8 Regular Number: IsRegular(x)

Function **IsRegular**(***x*** *As mpNum*) As Boolean

The function IsRegular returns TRUE if x is an regular number (i.e. neither NaN nor an infinity nor zero), and FALSE otherwise.

Parameter:
x: A real number.

6.10.9 Unordered Comparison: IsUnordered(x, y)

Function **IsUnordered**(***x*** *As mpNum*, ***y*** *As mpNum*) As Boolean

The function IsUnordered returns TRUE if x or y is NaN (i.e. they cannot be compared), and FALSE otherwise.

Parameters:
x: A real number.
y: A real number.

6.10.10 Number is Zero: IsZero(x)

Function **IsZero**(***x*** *As mpNum*) As Boolean

The function IsZero returns TRUE if $x = 0$, and FALSE otherwise.

Parameter:
x: A real number.

Chapter 7

MPC

7.0.11 Complex Multiprecision Numbers (MPC)

GNU MPC is a C library for the arithmetic of complex numbers with arbitrarily high precision and correct rounding of the result. It extends the principles of the IEEE-754 standard for fixed precision real floating point numbers to complex numbers, providing well-defined semantics for every operation. At the same time, speed of operation at high precision is a major design goal
The MPC reference is Enge *et al.* (2012)
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

7.0.11.1 mpc input

[Function] int mpc_strtoc (mpc t rop, const char *nptr, char **endptr, int base, mpc rnd t rnd)
Read a complex number from a string nptr in base base, rounded to the precision of rop with the given rounding mode rnd. The base must be either 0 or a number from 2 to 36 (otherwise the behaviour is undefi ned). If nptr starts with valid data, the result is stored in rop, the usual inexact value is returned (see [Return Value], page 7) and, if endptr is not the null pointer, *endptr points to the character just after the valid data. Otherwise, rop is set to NaN + i * NaN, -1 is returned and, if endptr is not the null pointer, the value of nptr is stored in the location referenced by endptr.
The expected form of a complex number string is either a real number (an optional leading whitespace, an optional sign followed by a fl oating-point number), or a pair of real numbers in parentheses separated by whitespace. If a real number is read, the missing imaginary part is set to +0. The form of a fl oating-point number depends on the base and is described in the documentation of mpfr_strtofr in the GNU MPFR manual. For instance, ”3.1415926”, ”(1.25e+7 +.17)”, ”(@nan@ 2)” and ”(-0 -7)” are valid strings for base = 10. If base = 0, then a prefi x may be used to indicate the base in which the fl oating-point number is written. Use prefi x âĂŹ0bâĂŹ for binary numbers, prefi x âĂŹ0xâĂŹ for hexadecimal numbers, and no prefix

for decimal numbers. The real and imaginary part may then be written in diff erent bases. For instance, "(1.024e+3 +2.05e+3)" and "(0b1p+10 +0x802)" are valid strings for base=0 and represent the same value.

7.0.11.2 mpc output

char * mpc_get_str (int b, size t n, mpc t op, mpc rnd t rnd)
Convert op to a string containing its real and imaginary parts, separated by a space and enclosed in a pair of parentheses. The numbers are written in base b (which may vary from 2 to 36) and rounded according to rnd. The number of signifi cant digits, at least 2, is given by n. It is also possible to let n be zero, in which case the number of digits is chosen large enough so that re-reading the printed value with the same precision, assuming both output and input use rounding to nearest, will recover the original value of op. Note that mpc_get_str uses the decimal point of the current locale if available, and âĂŸ.âĂŹ otherwise.
The string is generated using the current memory allocation function (malloc by default, unless it has been modifi ed using the custom memory allocation interface of gmp); once it is not needed any more, it should be freed by calling mpc_free_str.
In the following, we assume $z = x + iy$, $z_1 = x_1 + iy_1$, $z_2 = x_2 + iy_2$, etc.

7.1 Conversion between Real and Complex Numbers

7.1.1 Building a Complex Number from Real Components

Function **cplxRect**(***x*** *As mpNum,* ***y*** *As mpNum*) As mpNum

The function cplxRect returns a complex number z build from the real components x and y as $z = x + iy$.

Parameters:
x: A real number.
y: A real number.

7.1.2 Real Component

Function **cplxReal**(*z As mpNum*) As mpNum

The function cplxReal returns the real component x of $z = x + iy$.

Parameter:
z: A complex number.

7.1.3 Imaginary Component

Function **cplxImag**(*z As mpNum*) As mpNum

The function cplxImag returns the imaginary component y of $z = x + iy$.

Parameter:
z: A complex number.

7.1.4 Absolute Value

Function **cplxAbs**(*z As mpNum*) As mpNum

The function cplxAbs returns the absolute value of $z = x + iy$

Parameter:
z: A complex number.

The absolute value of $z = x + iy$ is calculated as

$$|z| = \sqrt{x^2 + y^2}. \tag{7.1.1}$$

7.1.5 Argument

Function **cplxArg**(*z As mpNum*) As mpNum

The function cplxArg returns the argument of $z = x + iy$

Parameter:
z: A complex number.

The argument θ of $z = x + iy$, is defined such that

$$z = x + iy = |x + iy|e^{\theta} = |x + iy|(\cos(\theta) + i\sin(\theta)). \tag{7.1.2}$$

cplxArg(z) is calculated as

$$\text{cplxArg}(z) = \arctan\left(\frac{y}{x}\right) = \theta, \text{ where } \theta \in (-\pi; \pi]. \tag{7.1.3}$$

7.2 Unary and Arithmetic Operators

7.2.1 Unary Minus and Conjugate

Function **cplxNeg**(***z*** *As mpNum*) As mpNum

The function cplxNeg returns $-z = -x - iy$.

Parameter:
z: A complex number.

Function **cplxConj**(***z*** *As mpNum*) As mpNum

The function cplxConj returns the conjugate of z, $\overline{z} = x - iy$

Parameter:
z: A complex number.

7.2.2 Addition and Sum

Operator +

The operator + returns the sum of $z1$ and $z2$.

Function **cplxAdd**(***z1*** *As mpNum*, ***z2*** *As mpNum*) As mpNum

The function cplxAdd returns $-z = -x - iy$.

Parameters:
z1: A complex number.
z2: A complex number.

The function cplxAdd$(z1, z2)$ returns the sum of $z1$ and $z2$:

$$z_1 + z_2 = (x_1 + x_2) + i(y_1 + y_2). \tag{7.2.1}$$

Function **cplxSum**(***z*** *As mpNum[]*) As mpNum

The function cplxSum returns the sum of up to 255 complex numbers.

Parameter:
z: An array of complex numbers.

7.2.3 Subtraction

Operator −

The operator − returns the difference of $z1$ and $z2$.

Function **cplxSub**(***z1*** *As mpNum*, ***z2*** *As mpNum*) As mpNum

The function cplxSub returns the difference of $z1$ and $z2$.

Parameters:
z1: A complex number.
z2: A complex number.

The function cplxSub($z1, z2$) returns the difference of $z1$ and $z2$:

$$z_1 - z_2 = (x_1 - x_2) + i(y_1 - y_2). \tag{7.2.2}$$

7.2.4 Multiplication

Operator $\times$

The operator + returns the product of $z1$ and $z2$.

Function **cplxMul**(***z1*** *As mpNum,* ***z2*** *As mpNum*) As mpNum

The function cplxMul returns the product of $z1$ and $z2$.

Parameters:
z1: A complex number.
z2: A complex number.

The function cplxMul($z1, z2$) returns the product of $z1$ and $z2$:

$$z_1 \cdot z_2 = (x_1 x_2 - y_1 y_2) + i(x_1 y_2 + x_2 y_1). \tag{7.2.3}$$

Function **cplxProduct**(***z*** *As mpNum[]*) As mpNum

The function cplxProduct returns the product of up to 255 complex numbers.

Parameter:
z: An array of complex numbers.

7.2.5 Division

Operator /

The operator − returns the quotient of $z1$ and $z2$.

Function **cplxDiv**(***z1*** *As mpNum,* ***z2*** *As mpNum*) As mpNum

The function cplxDiv returns the difference of $z1$ and $z2$.

Parameters:
z1: A complex number.
z2: A complex number.

The function cplxDiv($z1, z2$) returns the quotient of $z1$ and $z2$:

$$\frac{z_1}{z_2} = \frac{x_1 x_2 + y_1 y_2 + i(x_2 y_1 - x_1 y_2)}{x_2^2 + y_2^2} \tag{7.2.4}$$

7.3 Roots and Power Functions

7.3.1 Square: z^2

Function **cplxSqr**(***z*** *As mpNum*) As mpNum

The function cplxSqr returns the square of z.

Parameter:
z: A complex number.

The function cplxSqr$(z1, z2)$ returns the square of z:

$$z^2 = x^2 - y^2 + i(2xy). \tag{7.3.1}$$

7.3.2 Power Function with Integer Exponent: $z^k, k \in \mathbb{Z}$

Function **cplxPower**(***z*** *As mpNum,* ***k*** *As mpNum*) As mpNum

The function cplxPower returns an integer power of z

Parameters:
z: A complex number.
k: An integer.

The function cplxPower(z, k) returns an integer power of z:

$$z^k = r^k \cos(k\theta) + i(r^k \sin(k\theta)), \quad k \in \mathbb{Z}, \tag{7.3.2}$$

where $r = \sqrt{x^2 + y^2}$, and $\theta = \arctan(y/x)$.

7.3.3 Power Function with Real Exponent: $z^a, a \in \mathbb{R}$

Function **cplxPowR**(***z*** *As mpNum,* ***a*** *As mpNum*) As mpNum

The function cplxPowR returns an real power of z

Parameters:
z: A complex number.
a: A real number.

The function cplxPowR(z, k) returns a real power of z:

$$z^a = r^a \cos(a\theta) + i(r^a \sin(a\theta)), \quad a \in \mathbb{R}, \tag{7.3.3}$$

where $r = \sqrt{x^2 + y^2}$, and $\theta = \arctan(y/x)$.

7.3.4 Power Function with Complex Exponent: $z_1^{z_2}, z_2 \in \mathbb{C}$

Function **cplxPowC**(***z1*** *As mpNum,* ***z2*** *As mpNum*) As mpNum

The function cplxPowC returns an complex power of z

Parameters:
z1: A complex number.
z2: A complex number.

The function cplxPowC(z, k) returns a complex power of z_1:

$$z_1^{z_2} = \exp(\ln(z_1)z_2), \quad z_1, z_2 \in \mathbb{C}. \tag{7.3.4}$$

7.3.5 Square Root: $\sqrt{z}$

Function **cplxSqrt**(*z As mpNum*) As mpNum

The function cplxSqrt returns the square root of z

Parameter:
z: A complex number.

The function cplxSqrt(z) returns the square root of z:

$$\sqrt{z} = \sqrt{r}\cos\left(\tfrac{1}{2}\theta\right) + i\sqrt{r}\sin\left(\tfrac{1}{2}\theta\right), \tag{7.3.5}$$

where $r = \sqrt{x^2+y^2}$, and $\theta = \arctan(y/x)$.

7.3.6 Nth Root: $\sqrt[n]{z}, n = 2, 3, ...$

Function **cplxNthRoot**(*z As mpNum, n As mpNum*) As mpNum

The function cplxNthRoot returns an integer power of z

Parameters:
z: A complex number.
n: An integer.

The function cplxNthRoot(z, n) returns the n^{th} root of z:

$$\sqrt[n]{z} = z^{1/n} = \sqrt[n]{r}\exp\left(\frac{i\theta}{n}\right), \quad n \in \mathbb{N}, \tag{7.3.6}$$

where $r = \sqrt{x^2+y^2}$, and $\theta = \arctan(y/x)$. This is the principal root if $-\pi < \theta \leq \pi$. The other roots are given by

$$\sqrt[n]{z} = \sqrt[n]{r}\exp\left(\frac{i(\theta+2\pi k)}{n}\right), \quad k = 1, 2, \ldots, n-1. \tag{7.3.7}$$

7.4 Exponential and Logarithmic Functions

7.4.1 Exponential Function $e^z = \exp(z)$

Function **cplxExp**(*z As mpNum*) As mpNum

The function cplxExp returns the complex exponential of z

Parameter:
z: A complex number.

The function cplxExp(z) returns the complex exponential function of z:

$$\exp(z) = e^x\cos(y) + ie^x\sin(y). \tag{7.4.1}$$

7.4.2 Exponential Function $10^z = \exp_{10}(z)$

Function **cplxExp10**(*z As mpNum*) As mpNum

The function cplxExp10 returns 10^z

Parameter:
z: A complex number.

The function cplxExp10(z) returns $10^z = \exp_{10}(z) = \exp(z \cdot \ln(10))$.

7.4.3 Exponential Function $2^z = \exp_2(z)$

Function **cplxExp2**(*z As mpNum*) As mpNum

The function cplxExp2 returns 2^z

Parameter:
z: A complex number.

The function cplxExp2(z) returns $2^z = \exp_2(z) = \exp(z \cdot \ln(2))$.

7.4.4 Natural logarithm $\ln(z)$

Function **cplxLn**(*z As mpNum*) As mpNum

The function cplxLn returns the complex natural logarithm of z

Parameter:
z: A complex number.

The function cplxLn(z) returns the complex natural logarithm of z:

$$\ln(z) = \log_e(z) = \ln(r) + i\theta, \tag{7.4.2}$$

where $r = \sqrt{x^2 + y^2}$, and $\theta = \arctan(y/x)$.

7.4.5 Common (decadic) logarithm $\log_{10}(z)$

Function **cplxLog10**(*z As mpNum*) As mpNum

The function cplxLog10 returns $\log_{10}(z)$

Parameter:
z: A complex number.

The function cplxLn(z) returns the complex natural logarithm of z:

$$\log_{10}(z) = \ln(z)/\ln(10). \tag{7.4.3}$$

7.4.6 Binary logarithm $\log_2(z)$

Function **cplxLog2**(*z As mpNum*) As mpNum

The function cplxLog2 returns $\log_2(z)$

Parameter:
z: A complex number.

The function cplxLn(z) returns the complex natural logarithm of z:

$$\log_2(z) = \ln(z)/\ln(2). \tag{7.4.4}$$

7.5 Trigonometric Functions

7.5.1 Sine: $\sin(z)$

Function **cplxSin**(*z As mpNum*) As mpNum

The function cplxSin returns complex sine of z

Parameter:
z: A complex number.

The function cplxSin(z) returns the complex sine of z:

$$\sin(z) = \sin(x)\cosh(y) + i\cos(x)\sinh(y). \tag{7.5.1}$$

7.5.2 Cosine: $\cos(z)$

Function **cplxCos**(*z As mpNum*) As mpNum

The function cplxCos returns complex cosine of z

Parameter:
z: A complex number.

The function cplxCos(z) returns the complex cosine of z:

$$\cos(z) = \cos(x)\cosh(y) - i\sin(x)\sinh(y). \tag{7.5.2}$$

7.5.3 Tangent: $\tan(z)$

Function **cplxTan**(*z As mpNum*) As mpNum

The function cplxTan returns complex tangent of z

Parameter:
z: A complex number.

The function cplxTan(z) returns the tangent tangent of z:

$$\tan(z) = \frac{\sin(z)}{\cos(z)} = \frac{\sin(2x) + i\sinh(2y)}{\cos(2x) + i\cosh(2y)} \tag{7.5.3}$$

7.5.4 Secant: $\sec(z) = 1/\cos(z)$

Function **cplxSec**(*z As mpNum*) As mpNum

The function cplxSec returns the complex secant of z

Parameter:
z: A complex number.

The function cplxSec(z) returns the complex secant of z:

$$\sec(z) = 1/\cos(z). \tag{7.5.4}$$

7.5.5 Cosecant: $\csc(z) = 1/\sin(z)$

Function **cplxCsc**(*z As mpNum*) As mpNum

The function cplxCsc returns the complex cosecant of z

Parameter:
z: A complex number.

The function cplxCsc(z) returns the complex cosecant of z:

$$\sec(z) = 1/\sin(z). \tag{7.5.5}$$

7.5.6 Cotangent: $\cot(z) = 1/\tan(z)$

Function **cplxCot**(*z As mpNum*) As mpNum

The function cplxCot returns the complex cotangent of z

Parameter:
z: A complex number.

The function cplxCot(z) returns the complex cotangent of z:

$$\cot(z) = \frac{\cos(z)}{\sin(z)} = \frac{\sin(2x) - i\sinh(2y)}{\cosh(2y) - i\cos(2x)} \tag{7.5.6}$$

7.5.7 Hyperbolic Sine: $\sinh(z)$

Function **cplxSinh**(*z As mpNum*) As mpNum

The function cplxSinh returns the complex hyperbolic sine of z

Parameter:
z: A complex number.

The function cplxSinh(z) returns the complex hyperbolic sine of z:

$$\sinh(z) = \sinh(x)\cos(y) + i\cosh(x)\sin(y). \tag{7.5.7}$$

7.5.8 Hyperbolic Cosine: $\cosh(z)$

Function **cplxCosh**(*z As mpNum*) As mpNum

The function cplxCosh returns the complex hyperbolic cosine of z

Parameter:
z: A complex number.

The function cplxCosh(z) returns the complex hyperbolic cosine of z:

$$\cosh(z) = \cosh(x)\cos(y) + i\sinh(x)\sin(y). \tag{7.5.8}$$

7.5.9 Hyperbolic Tangent: $\tanh(z)$

Function **cplxTanh**(*z As mpNum*) As mpNum

The function cplxTanh returns the complex hyperbolic tangent of z

Parameter:
z: A complex number.

The function cplxTanh(z) returns the complex hyperbolic tangent of z:

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{\sinh(2x) + i\sin(2y)}{\cosh(2x) + i\cos(2y)} \tag{7.5.9}$$

7.5.10 Hyperbolic Secant: sech$(x) = 1/\cosh(z)$

Function **cplxSech**(*z As mpNum*) As mpNum

The function cplxSech returns the complex hyperbolic secant of z

Parameter:
z: A complex number.

The function cplxSech(z) returns the complex hyperbolic secant of z:

$$\operatorname{sech}(z) = 1/\cosh(z). \tag{7.5.10}$$

7.5.11 Hyperbolic Cosecant: csch$(x) = 1/\sinh(z)$

Function **cplxCsch**(*z As mpNum*) As mpNum

The function cplxCsch returns the complex hyperbolic cosecant of z

Parameter:
z: A complex number.

The function cplxCsch(z) returns the complex hyperbolic cosecant of z:

$$\operatorname{csch}(z) = 1/\sinh(z). \tag{7.5.11}$$

7.5.12 Hyperbolic Cotangent: $\coth(x) = 1/\tanh(z)$

Function **cplxCoth**(*z As mpNum*) As mpNum

The function cplxCoth returns the complex hyperbolic cotangent of z

Parameter:
z: A complex number.

The function cplxCot(z) returns the complex hyperbolic cotangent of z:

$$\coth(z) = \frac{\cosh(z)}{\sinh(z)} = \frac{\sinh(2x) - i\sin(2y)}{\cosh(2x) - i\cos(2y)} \tag{7.5.12}$$

7.6 Inverse Trigonometric Functions

The formulas in section follow Olver *et al.* (2010), equations 4.23.34 - 4.23.38 for sections 7.6.1 - 7.6.3, equation 4.23.9 for section 7.6.4, and Abramowitz & Stegun. (1970), equations 4.6.14 - 4.6.19 for sections 7.6.5 - 7.6.8.

7.6.1 Arcsine: asin(z)

Function **cplxASin**(*z As mpNum*) As mpNum

The function cplxASin returns the inverse complex sine of z

Parameter:
z: A complex number.

The function cplxASin(z) returns the inverse complex sine of $z = x + iy$:

$$\arcsin(z) = \arcsin(\beta) + i\ln\left(\alpha + \sqrt{\alpha^2 - 1}\right), \quad \text{where} \tag{7.6.1}$$

$$\alpha = \tfrac{1}{2}\sqrt{(x+1)^2 + y^2} + \tfrac{1}{2}\sqrt{(x-1)^2 + y^2}, \tag{7.6.2}$$

$$\beta = \tfrac{1}{2}\sqrt{(x+1)^2 + y^2} - \tfrac{1}{2}\sqrt{(x-1)^2 + y^2}, \tag{7.6.3}$$

and $x \in [-1, 1]$.

7.6.2 Arccosine: acos(z)

Function **cplxACos**(***z*** *As mpNum*) As mpNum

The function cplxACos returns the inverse complex cosine of z

Parameter:
z: A complex number.

The function cplxACos(z) returns the inverse complex cosine of $z = x + iy$:

$$\arccos(z) = \arccos(\beta) - i \ln\left(\alpha + \sqrt{\alpha^2 - 1}\right), \quad \text{where} \tag{7.6.4}$$

α and β are defined in equations 7.6.2 and 7.6.3, and $x \in [-1, 1]$.

7.6.3 Arctangent: atan(z)

Function **cplxATan**(***z*** *As mpNum*) As mpNum

The function cplxATan returns the inverse complex tangent of z

Parameter:
z: A complex number.

The function cplxATan(z) returns the inverse complex tangent of $z = x + iy$:

$$\arctan(z) = \tfrac{1}{2}\arctan\left(\frac{2x}{1-x^2-y^2}\right) + \tfrac{1}{4}i\ln\left(\frac{x^2+(y+1)^2}{x^2+(y-1)^2}\right), \quad \text{where } |z| < 1. \tag{7.6.5}$$

7.6.4 Arccotangent: acot(z)

Function **cplxACot**(***z*** *As mpNum*) As mpNum

The function cplxACot returns the inverse complex cotangent of z

Parameter:
z: A complex number.

The function cplxACot(z) returns the inverse complex cotangent of z:

$$\operatorname{arccot}(z) = \arctan(1/z), \quad z \neq \pm i. \tag{7.6.6}$$

7.6.5 Inverse Hyperbolic Sine: asinh(z)

Function **cplxASinh**(***z*** *As mpNum*) As mpNum

The function cplxASinh returns the inverse complex hyperbolic sine of z

Parameter:
z: A complex number.

The function cplxASinh(z) returns the inverse complex hyperbolic sine of z:

$$\operatorname{arcsinh}(z) = -i \arcsin(iz), \tag{7.6.7}$$

where $\arcsin(z)$ is defined in section 7.6.1

7.6.6 Inverse Hyperbolic Cosine: acosh(z)

Function **cplxACosh**(*z As mpNum*) As mpNum

The function cplxACosh returns the inverse complex hyperbolic cosine of z

Parameter:
z: A complex number.

The function cplxACosh(z) returns the inverse complex hyperbolic cosine of z:

$$\operatorname{arccosh}(z) = \pm i \arccos(z), \tag{7.6.8}$$

where $\arccos(z)$ is defined in section 7.6.2

7.6.7 Inverse Hyperbolic Tangent: atanh(z)

Function **cplxATanh**(***z*** *As mpNum*) As mpNum

The function cplxATanh returns the inverse complex hyperbolic tangent of z

Parameter:
z: A complex number.

The function cplxATanh(z) returns the inverse complex hyperbolic tangent of z:

$$\operatorname{arctanh}(z) = -i \arctan(z), \tag{7.6.9}$$

where $\arctan(z)$ is defined in section 7.6.3

7.6.8 Inverse Hyperbolic Cotangent: acoth(z)

Function **cplxACoth**(***z*** *As mpNum*) As mpNum

The function cplxACoth returns the inverse complex hyperbolic cotangent of z

Parameter:
z: A complex number.

The function cplxACoth(z) returns the inverse complex hyperbolic cotangent of z:

$$\operatorname{arctanh}(z) = i \arctan(iz), \tag{7.6.10}$$

where $\arctan(z)$ is defined in section 7.6.4

Chapter 8

MPFI

8.0.9 Multiprecision Interval Arithmetic (MPFI)

MPFI (Multiple Precision Floating-Point Interval Library) is a library for arbitrary precision interval arithmetic with intervals represented using MPFR reliable floating-point numbers. It is based on the GNU MP library and on the MPFR library. The purpose of an arbitrary precision interval arithmetic is on the one hand to get guaranteed results, thanks to interval computation, and on the other hand to obtain accurate results, thanks to multiple precision arithmetic. The MPFI library is built upon MPFR to benefit from the correct roundings provided by MPFR, it portability, and its compliance with the IEEE 754 standard for floating-point arithmetic
References for MPFI: Revol & Rouillier (2005), Moore R. E. (2009), Hayes (2003), and Rump (1999)
References for C-XSC 2.0 Hofschuster & Krämer (2004)
Manual for MPFR/MPFI version: Blomquist *et al.* (2012)
C++ Toolbox for Verified Scientific Computing I: Hammer *et al.* (1995)
C++ Toolbox for Verified Scientific Computing II: Krämer *et al.* (1994) and Krämer *et al.* (2006)
PASCAL-XSC Language Reference: Klatte *et al.* (1991)
speziellen Funktionen der mathematischen Physik: Hofschuster (2000)
Integration: Wedner (2000)
A priori error estimates : Blomquist (2005)
Other papers: Blomquist *et al.* (2008b) and Blomquist *et al.* (2008a) with detailed description of extended complex interval arithmetic.
and Krämer *et al.* (2012)

8.0.9.1 mpfi input

int mpfi_set_str (mpfi t rop, char *s, int base)
Sets rop to the value of the string s, in base base (between 2 and 36), outward rounded to the precision of rop: op then belongs to rop. The exponent is read in decimal. The string is of the form âĂŸnumberâĂŹ or âĂŸ[number1 , number 2]âĂŹ. Each endpoint has the form âĂŸM@NâĂŹ or, if the base is 10 or less, alternatively âĂŸMeNâĂŹ or âĂŸMENâĂŹ. âĂŸMâĂŹ is the mantissa and âĂŸNâĂŹ is the exponent. The mantissa is always in the specified base. The exponent is in decimal. The argument base may be in the ranges 2 to 36.
This function returns 1 if the input is incorrect, and 0 otherwise.

8.0.9.2 mpfi output

size_t mpfi_out_str (FILE *stream, int base, size t n_digits, mpfi t op)

Outputs op on stdio stream stream, as a string of digits in base base. The output is an opening square bracket "[", followed by the lower endpoint, a separating comma, the upper endpoint and a closing square bracket "]".
For each endpoint, the output is performed by mpfr_out_str. The following piece of in- formation is taken from MPFR documentation. The base may vary from 2 to 36. For each endpoint, it prints at most n digits signifi cant digits, or if n digits is 0, the maximum number of digits accurately representable by op. In addition to the signifi cant digits, a decimal point at the right of the fi rst digit and a trailing exponent, in the formâĂŸ eNNNâĂŹ, are printed. If base is greater than 10, âĂŸ@âĂŹ will be used instead of âĂŸeâĂŹ as exponent delimiter.
Returns the number of bytes written, or if an error occurred, return 0.
As mpfi_out_str outputs an enclosure of the input interval, and as mpfi_inp_str provides an enclosure of the interval it reads, these functions are not reciprocal. More precisely, when they are called one after the other, the resulting interval contains the initial one, and this inclusion may be strict.

8.1 Information Functions for Intervals

8.1.1 IsEmpty

Function **IsEmpty**(***x*** *As mpNum*) As mpNum

The function IsEmpty returns TRUE if x is empty (its endpoints are in reverse order), and FALSE otherwise.

Parameter:
x: A real number.

Nothing is done in arithmetic or special functions to handle empty intervals: it is the responsibility of the user to avoid computing with empty intervals.

8.1.2 IsInside

Function **IsInside**(***x*** *As mpNum*, ***y*** *As mpNum*) As mpNum

The function IsInside returns TRUE if x is contained in y, and FALSE otherwise.

Parameters:
x: A real number.
y: A real number.

Returns FALSE if at least one argument is NaN or an invalid interval.

8.1.3 IsStrictlyInside

Function **IsStrictlyInside**(***x*** *As mpNum*, ***y*** *As mpNum*) As mpNum

The function IsStrictlyInside returns TRUE if the second interval y is contained in the interior of x, and FALSE otherwise.

Parameters:
x: A real number.

y: A real number.

8.1.4 IsStrictlyNeg

Function **IsStrictlyNeg**(***x*** *As mpNum*) As mpNum

The function IsStrictlyNeg returns TRUE if x contains only negative numbers, and FALSE otherwise.

Parameter:
x: A real number.

8.1.5 IsStrictlyPos

Function **IsStrictlyPos**(***x*** *As mpNum*) As mpNum

The function IsStrictlyPos returns TRUE if x contains only positive numbers, and FALSE otherwise.

Parameter:
x: A real number.

Chapter 9

MPFCI

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Part III

Eigen: Real and Complex Linear Algebra

Chapter 10

BLAS Support (based on Eigen)

The Eigen reference is Guennebaud *et al.* (2010)
The Basic Linear Algebra Subprograms (BLAS) define a set of fundamental operations on vectors and matrices which can be used to create optimized higher-level linear algebra functionality. Specifications for Level 1, Level 2 and Level 3 BLAS can be found in Dongarra *et al.* (1988, 1990); Lawson *et al.* (1979).
Based on BLAS in LAPACK, which is described in Anderson *et al.* (1999); Barker *et al.* (2001).

The library provides high-level interface for blas operations on vectors and matrices. This should satisfy the needs of most users. Note that currently matrices are implemented using dense-storage so the interface only includes the corresponding dense-storage blas functions. The full blas functionality for band-format and packed-format matrices will be available in later versions of the library.

There are three levels of blas operations,
Level 1: Vector operations, e.g. y = ax + y
Level 2: Matrix-vector operations, e.g. y = ÎśAx + Îšy
Level 3: Matrix-matrix operations, e.g. C = ÎśAB + C

Each routine has a name which specifies the operation, the type of matrices involved and their precisions. Some of the most common operations and their names are given below,

DOT scalar product, xT y
AXPY vector sum, Îśx + y
MV matrix-vector product, Ax
SV matrix-vector solve, inv(A)x
MM matrix-matrix product, AB
SM matrix-matrix solve, inv(A)B

The types of matrices are,
GE general
GB general band
SY symmetric
SB symmetric band
SP symmetric packed
HE hermitian
HB hermitian band
HP hermitian packed
TR triangular

TB triangular band
TP triangular packed
Each operation is defined for four precisions,
S single real
D double real
C single complex
Z double complex

Thus, for example, the name sgemm stands for single-precision general matrix-matrix multiply and zgemm stands for double-precision complex matrix-matrix multiply.
Book reference: Golub & Van Loan (1996)
Book reference: Bernstein (2009)
Book reference: Seber (2008)

10.1 BLAS Level 1 Support and related Functions

10.1.1 Vector-Vector Product

Function **RDot**(***x*** *As mpNum[]*, ***y*** *As mpNum[]*) As mpNum

The function RDot returns the real scalar product $\boldsymbol{x}^T\boldsymbol{y}$ for the real vectors $\boldsymbol{x}$ and $\boldsymbol{y}$.

Parameters:
x: A vector of real numbers.
y: A vector of real numbers.

Function **cplxDotu**(***x*** *As mpNum[]*, ***y*** *As mpNum[]*) As mpNum

The function cplxDotu returns the complex scalar product $\boldsymbol{x}^T\boldsymbol{y}$ for the complex vectors $\boldsymbol{x}$ and $\boldsymbol{y}$.

Parameters:
x: A vector of complex numbers.
y: A vector of complex numbers.

Function **cplxDotc**(***x*** *As mpNum[]*, ***y*** *As mpNum[]*) As mpNum

The function cplxDotc returns the complex conjugate scalar product $\boldsymbol{x}^H\boldsymbol{y}$ for the complex vectors $\boldsymbol{x}$ and $\boldsymbol{y}$.

Parameters:
x: A vector of complex numbers.
y: A vector of complex numbers.

10.1.2 Euclidian Norm

Function **RNrm2**(***x*** *As mpNum[]*, ***y*** *As mpNum[]*) As mpNum

The function RNrm2 returns the Euclidean norm $||\boldsymbol{x}||_2$ of the real vector $\boldsymbol{x}$.

Parameters:

x: A vector of real numbers.
y: A vector of real numbers.

Function **cplxNrm2**(***x*** *As mpNum[]*, ***y*** *As mpNum[]*) As mpNum

---

The function cplxNrm2 returns the Euclidean norm $||\boldsymbol{x}||_2$ of the complex vector $\boldsymbol{x}$.

Parameters:
x: A vector of complex numbers.
y: A vector of complex numbers.

10.1.3 Absolute Sum

Function **RAsum**(***x*** *As mpNum[]*, ***y*** *As mpNum[]*) As mpNum

---

The function RAsum returns the the absolute sum of the elements of the real vector $\boldsymbol{x}$.

Parameters:
x: A vector of real numbers.
y: A vector of real numbers.

Function **cplxAsum**(***x*** *As mpNum[]*, ***y*** *As mpNum[]*) As mpNum

---

The function cplxAsum returns the sum of the magnitudes of the real and imaginary parts of the complex vector $\boldsymbol{x}$.

Parameters:
x: A vector of complex numbers.
y: A vector of complex numbers.

10.1.4 Addition

Function **RAxpy**(α *As mpNum*, ***x*** *As mpNum[]*, ***y*** *As mpNum[]*) As mpNum

---

The function RAxpy returns the sum $\alpha\boldsymbol{x} + \boldsymbol{y}$ for the real scalar α and the real vectors $\boldsymbol{x}$ and $\boldsymbol{y}$.

Parameters:
α: A real scalar.
x: A vector of real numbers.
y: A vector of real numbers.

Function **cplxAxpy**(α *As mpNum*, ***x*** *As mpNum[]*, ***y*** *As mpNum[]*) As mpNum

---

The function cplxAxpy returns the sum $\alpha\boldsymbol{x} + \boldsymbol{y}$ for the complex scalar α and the complex vectors $\boldsymbol{x}$ and $\boldsymbol{y}$.

Parameters:
α: A complex scalar.
x: A vector of complex numbers.
y: A vector of complex numbers.

10.2 BLAS Level 2 Support

10.2.1 Matrix-Vector Product and Sum (General Matrix)

Function **RGemv**(***TransA*** *As Integer*, α *As mpNum*, ***A*** *As mpNum[,]*, ***x*** *As mpNum[]*, β *As mpNum*, ***y*** *As mpNum[]*) As mpNum

The function RGemv returns the matrix-vector product and sum for a general matrix.

Parameters:
TransA: An indicator specifying the multiplication.
α: A real scalar.
A: A matrix of real numbers.
x: A vector of real numbers.
β: A real scalar.
y: A vector of real numbers.

For the real scalars α and β, the real vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, and the real matrix $\boldsymbol{A}$, the function RGemv computes the matrix-vector product and sum

$$\mathsf{RGemv} = \begin{cases} \alpha\boldsymbol{A}\boldsymbol{x} + \beta\boldsymbol{y}, & \text{for } \mathsf{TransA} = \mathsf{mpBlasNoTrans}, \\ \alpha\boldsymbol{A}^T\boldsymbol{x} + \beta\boldsymbol{y}, & \text{for } \mathsf{TransA} = \mathsf{mpBlasTrans}. \end{cases} \tag{10.2.1}$$

Function **cplxGemv**(***TransA*** *As Integer*, α *As mpNum*, ***A*** *As mpNum[,]*, ***x*** *As mpNum[]*, β *As mpNum*, ***y*** *As mpNum[]*) As mpNum

The function cplxGemv returns the matrix-vector product and sum for a general matrix.

Parameters:
TransA: An indicator specifying the multiplication.
α: A complex scalar.
A: A matrix of complex numbers.
x: A vector of complex numbers.
β: A complex scalar.
y: A vector of complex numbers.

For the complex scalars α and β, the complex vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, and the complex matrix $\boldsymbol{A}$, the function cplxGemv computes the matrix-vector product and sum

$$\mathsf{cplxGemv} = \begin{cases} \alpha\boldsymbol{A}\boldsymbol{x} + \beta\boldsymbol{y}, & \text{for } \mathsf{TransA} = \mathsf{mpBlasNoTrans}, \\ \alpha\boldsymbol{A}^T\boldsymbol{x} + \beta\boldsymbol{y}, & \text{for } \mathsf{TransA} = \mathsf{mpBlasTrans}, \\ \alpha\boldsymbol{A}^H\boldsymbol{x} + \beta\boldsymbol{y}, & \text{for } \mathsf{TransA} = \mathsf{mpBlasConjTrans}. \end{cases} \tag{10.2.2}$$

10.2.2 Matrix-Vector Product (Triangular Matrix)

Function **RTrmv**(***Uplo*** *As Integer*, ***TransA*** *As Integer*, ***Diag*** *As Integer*, ***A*** *As mpNum[,]*, ***x*** *As mpNum[]*) As mpNum

The function RTrmv returns the matrix-vector product for a triangular matrix.

Parameters:
Uplo: An indicator specifying whether the upper or lower triangle will be used.
TransA: An indicator specifying the multiplication.
Diag: An indicator specifying the use of the diagonal.
A: A matrix of real numbers.
x: A vector of real numbers.

For the real vector $\boldsymbol{x}$ and the real triangular matrix $\boldsymbol{A}$, the function RTrmv computes the matrix-vector product

$$\mathsf{RTrmv} = \begin{cases} \boldsymbol{A}\boldsymbol{x}, & \text{for TransA} = \mathsf{mpBlasNoTrans}, \\ \boldsymbol{A}^T\boldsymbol{x}, & \text{for TransA} = \mathsf{mpBlasTrans}. \end{cases} \tag{10.2.3}$$

When Uplo is 0 then the upper triangle of $\boldsymbol{A}$ is used, and when Uplo is 1 then the lower triangle of $\boldsymbol{A}$ is used. If Diag is 0 then the diagonal of the matrix is used, but if Diag is 1 then the diagonal elements of the matrix $\boldsymbol{A}$ are taken as unity and are not referenced.

Function **cplxTrmv**(***Uplo*** *As Integer*, ***TransA*** *As Integer*, ***Diag*** *As Integer*, ***A*** *As mpNum[,]*, ***x*** *As mpNum[]*) As mpNum

The function cplxTrmv returns the matrix-vector product for a triangular matrix.

Parameters:
Uplo: An indicator specifying whether the upper or lower triangle will be used.
TransA: An indicator specifying the multiplication.
Diag: An indicator specifying the use of the diagonal.
A: A matrix of complex numbers.
x: A vector of complex numbers.

For the complex vector $\boldsymbol{x}$ and the complex triangular matrix $\boldsymbol{A}$, the function cplxTrmv computes the matrix-vector product

$$\mathsf{cplxTrmv} = \begin{cases} \boldsymbol{A}\boldsymbol{x}, & \text{for TransA} = \mathsf{mpBlasNoTrans}, \\ \boldsymbol{A}^T\boldsymbol{x}, & \text{for TransA} = \mathsf{mpBlasTrans}, \\ \boldsymbol{A}^H\boldsymbol{x}, & \text{for TransA} = \mathsf{mpBlasConjTrans}. \end{cases} \tag{10.2.4}$$

When Uplo is 0 then the upper triangle of $\boldsymbol{A}$ is used, and when Uplo is 1 then the lower triangle of $\boldsymbol{A}$ is used. If Diag is 0 then the diagonal of the matrix is used, but if Diag is 1 then the diagonal elements of the matrix $\boldsymbol{A}$ are taken as unity and are not referenced.

10.2.3 Inverse Matrix-Vector Product (Triangular Matrix)

Function **RTrsv**(***Uplo*** *As Integer,* ***TransA*** *As Integer,* ***Diag*** *As Integer,* ***A*** *As mpNum[,],* ***x*** *As mpNum[]*) As mpNum

The function RTrsv returns the inverse matrix-vector product for a triangular matrix.

Parameters:
Uplo: An indicator specifying whether the upper or lower triangle will be used.
TransA: An indicator specifying the multiplication.
Diag: An indicator specifying the use of the diagonal.
A: A matrix of real numbers.
x: A vector of real numbers.

For the real vector $\boldsymbol{x}$ and the real triangular matrix $\boldsymbol{A}$, the function RTrsv computes the matrix-vector product

$$\mathsf{RTrsv} = \begin{cases} \boldsymbol{A}^{-1}\boldsymbol{x}, & \text{for TransA} = \mathsf{mpBlasNoTrans}, \\ \left(\boldsymbol{A}^{T}\right)^{-1}\boldsymbol{x}, & \text{for TransA} = \mathsf{mpBlasTrans}. \end{cases} \tag{10.2.5}$$

When Uplo is 0 then the upper triangle of $\boldsymbol{A}$ is used, and when Uplo is 1 then the lower triangle of $\boldsymbol{A}$ is used. If Diag is 0 then the diagonal of the matrix is used, but if Diag is 1 then the diagonal elements of the matrix $\boldsymbol{A}$ are taken as unity and are not referenced.

Function **cplxTrsv**(***Uplo*** *As Integer,* ***TransA*** *As Integer,* ***Diag*** *As Integer,* ***A*** *As mpNum[,],* ***x*** *As mpNum[]*) As mpNum

The function cplxTrsv returns the inverse matrix-vector product for a triangular matrix.

Parameters:
Uplo: An indicator specifying whether the upper or lower triangle will be used.
TransA: An indicator specifying the multiplication.
Diag: An indicator specifying the use of the diagonal.
A: A matrix of complex numbers.
x: A vector of complex numbers. For the complex vector $\boldsymbol{x}$ and the complex triangular matrix $\boldsymbol{A}$, the function cplxTrsv computes the matrix-vector product

$$\mathsf{cplxTrsv} = \begin{cases} \boldsymbol{A}^{-1}\boldsymbol{x} & \text{for TransA} = \mathsf{mpBlasNoTrans}, \\ \left(\boldsymbol{A}^{T}\right)^{-1}\boldsymbol{x}, & \text{for TransA} = \mathsf{mpBlasTrans}, \\ \left(\boldsymbol{A}^{H}\right)^{-1}\boldsymbol{x}, & \text{for TransA} = \mathsf{mpBlasConjTrans}. \end{cases} \tag{10.2.6}$$

When Uplo is 0 then the upper triangle of $\boldsymbol{A}$ is used, and when Uplo is 1 then the lower triangle of $\boldsymbol{A}$ is used. If Diag is 0 then the diagonal of the matrix is used, but if Diag is 1 then the diagonal elements of the matrix $\boldsymbol{A}$ are taken as unity and are not referenced.

10.2.4 Matrix-Vector Product and Sum (Symmetric/Hermitian Matrix)

Function **RSymv**(***Uplo*** *As Integer*, α *As mpNum*, ***A*** *As mpNum[,]*, ***x*** *As mpNum[]*, β *As mpNum*, ***y*** *As mpNum[]*) As mpNum

The function RSymv returns the matrix-vector product and sum for a symmetric matrix.

Parameters:
Uplo: An indicator specifying whether the upper or lower triangle will be used.
α: A real scalar.
A: A matrix of real numbers.
x: A vector of real numbers.
β: A real scalar.
y: A vector of real numbers.

For the real scalars α and β, the real vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, and the real symmetric matrix $\boldsymbol{A}$, the function RSymv computes the matrix-vector product and sum

$$\mathsf{RSymv} = \alpha \boldsymbol{A}\boldsymbol{x} + \beta \boldsymbol{y}. \tag{10.2.7}$$

When Uplo is 0 then the upper triangle and diagonal of $\boldsymbol{A}$ are used, and when Uplo is 1 then the lower triangle and diagonal of $\boldsymbol{A}$ are used.

Function **cplxHemv**(***Uplo*** *As Integer*, α *As mpNum*, ***A*** *As mpNum[,]*, ***x*** *As mpNum[]*, β *As mpNum*, ***y*** *As mpNum[]*) As mpNum

The function cplxHemv returns the matrix-vector product and sum for a hermitian matrix.

Parameters:
Uplo: An indicator specifying whether the upper or lower triangle will be used.
α: A complex scalar.
A: A matrix of complex numbers.
x: A vector of complex numbers.
β: A complex scalar.
y: A vector of complex numbers.

For the complex scalars α and β, the complex vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, and the complex hermitian matrix $\boldsymbol{A}$, the function cplxHemv computes the matrix-vector product and sum

$$\mathsf{cplxHemv} = \alpha \boldsymbol{A}\boldsymbol{x} + \beta \boldsymbol{y}. \tag{10.2.8}$$

When Uplo is 0 then the upper triangle and diagonal of $\boldsymbol{A}$ are used, and when Uplo is 1 then the lower triangle and diagonal of $\boldsymbol{A}$ are used.
In cplxHemv, the imaginary elements of the diagonal are automatically assumed to be zero and are not referenced.

10.2.5 Rank-1 update (General Matrix)

Function **RGer**(α *As mpNum*, ***x*** *As mpNum[]*, ***y*** *As mpNum[]*, ***A*** *As mpNum[,]*) As mpNum

The function RGer returns the rank-1 update for a general matrix

Parameters:
α: A real scalar.
x: A vector of real numbers.
y: A vector of real numbers.
A: A matrix of real numbers.

For the real scalar α, the real vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, and the real general matrix $\boldsymbol{A}$, the function RGer computes the rank-1 update of the matrix $\boldsymbol{A}$, defined as

$$\mathsf{RGer} = \alpha \boldsymbol{x}\boldsymbol{y}^T + \boldsymbol{A}. \tag{10.2.9}$$

Function **cplxGeru**(α *As mpNum*, ***x*** *As mpNum[]*, ***y*** *As mpNum[]*, ***A*** *As mpNum[,]*) As mpNum

The function cplxGeru returns the rank-1 update for a general matrix

Parameters:
α: A complex scalar.
x: A vector of complex numbers.
y: A vector of complex numbers.
A: A matrix of complex numbers.

For the complex scalar α, the complex vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, and the complex general matrix $\boldsymbol{A}$, the function cplxGeru computes the rank-1 update of the matrix $\boldsymbol{A}$, defined as

$$\mathsf{cplxGeru} = \alpha \boldsymbol{x}\boldsymbol{y}^T + \boldsymbol{A}. \tag{10.2.10}$$

Function **cplxGerc**(α *As mpNum*, ***x*** *As mpNum[]*, ***y*** *As mpNum[]*, ***A*** *As mpNum[,]*) As mpNum

The function cplxGerc returns the rank-1 update for a general matrix

Parameters:
α: A complex scalar.
x: A vector of complex numbers.
y: A vector of complex numbers.
A: A matrix of complex numbers.

For the complex scalar α, the complex vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, and the complex general matrix $\boldsymbol{A}$, the function cplxGerc computes the rank-1 update of the matrix $\boldsymbol{A}$, defined as

$$\mathsf{cplxGerc} = \alpha \boldsymbol{x}\boldsymbol{y}^H + \boldsymbol{A}. \tag{10.2.11}$$

10.2.6 Rank-1 update (Symmetric/Hermitian Matrix)

Function **RSyr**(***Uplo*** *As Integer*, α *As mpNum*, ***x*** *As mpNum[]*, ***A*** *As mpNum[,]*) As mpNum

The function RSyr returns the Rank-1 update for a symmetric matrix.

Parameters:
Uplo: An indicator specifying whether the upper or lower triangle will be used.
α: A real scalar.
x: A vector of real numbers.
A: A matrix of real numbers.

For the real scalar α, the real vector $\boldsymbol{x}$, and the real symmetric matrix $\boldsymbol{A}$, the function RSyr computes the symmetric rank-1 update of the matrix $\boldsymbol{A}$, defined as

$$\mathsf{RSyr} = \alpha \boldsymbol{x}\boldsymbol{x}^T + \boldsymbol{A}. \tag{10.2.12}$$

Since the matrix $\boldsymbol{A}$ is symmetric, only its upper half or lower half need to be stored. When Uplo is 0 then the upper triangle and diagonal of $\boldsymbol{A}$ are used, and when Uplo is 1 then the lower triangle and diagonal of $\boldsymbol{A}$ are used.

Function **cplxHer**(***Uplo*** *As Integer*, α *As mpNum*, ***x*** *As mpNum[]*, ***A*** *As mpNum[,]*) As mpNum

The function cplxHer returns the Rank-1 update for a hermitian matrix.

Parameters:
Uplo: An indicator specifying whether the upper or lower triangle will be used.
α: A complex scalar.
x: A vector of complex numbers.
A: A matrix of complex numbers.

For the complex scalar α, the complex vector $\boldsymbol{x}$, and the complex hermitian matrix $\boldsymbol{A}$, the function cplxHer computes the hermitian rank-1 update of the matrix $\boldsymbol{A}$, defined as

$$\mathsf{cplxHer} = \alpha \boldsymbol{x}\boldsymbol{x}^H + \boldsymbol{A}. \tag{10.2.13}$$

Since the matrix $\boldsymbol{A}$ is hermitian, only its upper half or lower half need to be stored. When Uplo is 0 then the upper triangle and diagonal of $\boldsymbol{A}$ are used, and when Uplo is 1 then the lower triangle and diagonal of $\boldsymbol{A}$ are used. The imaginary elements of the diagonal are automatically set to zero.

10.2.7 Rank-2 update (Symmetric/Hermitian Matrix)

Function **RSyr2**(***Uplo*** *As Integer*, α *As mpNum*, ***x*** *As mpNum[]*, ***y*** *As mpNum[]*, ***A*** *As mpNum[,]*) As mpNum

The function RSyr2 returns the Rank-1 update for a symmetric matrix.

Parameters:
Uplo: An indicator specifying whether the upper or lower triangle will be used.
α: A real scalar.
x: A vector of real numbers.
y: A vector of real numbers.
A: A matrix of real numbers.

For the real scalar α, the real vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, and the real symmetric matrix $\boldsymbol{A}$, the function RSyr2 computes the symmetric rank-1 update of the matrix $\boldsymbol{A}$, defined as

$$\mathsf{RSyr2} = \alpha\boldsymbol{x}\boldsymbol{y}^T + \alpha\boldsymbol{y}\boldsymbol{x}^T + \boldsymbol{A}. \tag{10.2.14}$$

Since the matrix $\boldsymbol{A}$ is symmetric, only its upper half or lower half need to be stored. When Uplo is 0 then the upper triangle and diagonal of $\boldsymbol{A}$ are used, and when Uplo is 1 then the lower triangle and diagonal of $\boldsymbol{A}$ are used.

Function **cplxHer2**(***Uplo*** *As Integer*, α *As mpNum*, ***x*** *As mpNum[]*, ***y*** *As mpNum[]*, ***A*** *As mpNum[,]*) As mpNum

The function cplxHer2 returns the Rank-1 update for a hermitian matrix.

Parameters:
Uplo: An indicator specifying whether the upper or lower triangle will be used.
α: A complex scalar.
x: A vector of complex numbers.
y: A vector of complex numbers.
A: A matrix of complex numbers.

For the complex scalar α, the complex vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, and the complex hermitian matrix $\boldsymbol{A}$, the function cplxHer2 computes the hermitian rank-1 update of the matrix $\boldsymbol{A}$, defined as

$$\mathsf{cplxHer2} = \alpha\boldsymbol{x}\boldsymbol{y}^H + \alpha^*\boldsymbol{y}\boldsymbol{x}^H + \boldsymbol{A}. \tag{10.2.15}$$

Since the matrix $\boldsymbol{A}$ is hermitian, only its upper half or lower half need to be stored. When Uplo is 0 then the upper triangle and diagonal of $\boldsymbol{A}$ are used, and when Uplo is 1 then the lower triangle and diagonal of $\boldsymbol{A}$ are used. The imaginary elements of the diagonal are automatically set to zero.

10.3 BLAS Level 3 Support

10.3.1 Matrix-Matrix-Product and Sum (General Matrix A)

Function **RGemm**(***TransA*** *As Integer*, ***TransB*** *As Integer*, α *As mpNum*, ***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*, β *As mpNum*, ***C*** *As mpNum[,]*) As mpNum

The function RGemm returns the matrix-matrix product and sum for a general matrix.

Parameters:
TransA: An indicator specifying the multiplication.
TransB: An indicator specifying the multiplication.
α: A real scalar.
A: A matrix of real numbers.
B: A matrix of real numbers.
β: A real scalar.
C: A matrix of real numbers.

For the real scalars α and β, and the real matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, the function RGemm computes the matrix-matrix product and sum

$$\mathsf{RGemm} = \begin{cases} \alpha\boldsymbol{A}\boldsymbol{B} + \beta\boldsymbol{C}, & \text{for TransA} = \mathsf{mpBlasNoTrans}, \text{ TransB} = \mathsf{mpBlasNoTrans}, \\ \alpha\boldsymbol{A}\boldsymbol{B}^T + \beta\boldsymbol{C}, & \text{for TransA} = \mathsf{mpBlasNoTrans}, \text{ TransB} = \mathsf{mpBlasTrans}, \\ \alpha\boldsymbol{A}^T\boldsymbol{B} + \beta\boldsymbol{C}, & \text{for TransA} = \mathsf{mpBlasTrans}, \text{ TransB} = \mathsf{mpBlasNoTrans}, \\ \alpha\boldsymbol{A}^T\boldsymbol{B}^T + \beta\boldsymbol{C}, & \text{for TransA} = \mathsf{mpBlasTrans}, \text{ TransB} = \mathsf{mpBlasTrans}, \end{cases} \tag{10.3.1}$$

Function **cplxGemm**(***TransA*** *As Integer,* ***TransB*** *As Integer,* α *As mpNum,* ***A*** *As mpNum[,],* ***B*** *As mpNum[,],* β *As mpNum,* ***C*** *As mpNum[,]*) As mpNum

The function cplxGemm returns the matrix-matrix product and sum for a general matrix.

Parameters:
TransA: An indicator specifying the multiplication.
TransB: An indicator specifying the multiplication.
α: A real scalar.
A: A matrix of real numbers.
B: A matrix of real numbers.
β: A real scalar.
C: A matrix of real numbers.

For the complex scalars α and β, and the complex matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, the function cplxGemm computes the matrix-matrix product and sum

$$\text{cplxGemm} = \begin{cases} \alpha\boldsymbol{A}\boldsymbol{B} + \beta\boldsymbol{C}, & \text{for TransA} = \text{mpBlasNoTrans, TransB} = \text{mpBlasNoTrans,} \\ \alpha\boldsymbol{A}\boldsymbol{B}^T + \beta\boldsymbol{C}, & \text{for TransA} = \text{mpBlasNoTrans, TransB} = \text{mpBlasTrans,} \\ \alpha\boldsymbol{A}\boldsymbol{B}^H + \beta\boldsymbol{C}, & \text{for TransA} = \text{mpBlasNoTrans, TransB} = \text{mpBlasConjTrans,} \\ \alpha\boldsymbol{A}^T\boldsymbol{B} + \beta\boldsymbol{C}, & \text{for TransA} = \text{mpBlasTrans, TransB} = \text{mpBlasNoTrans,} \\ \alpha\boldsymbol{A}^T\boldsymbol{B}^T + \beta\boldsymbol{C}, & \text{for TransA} = \text{mpBlasTrans, TransB} = \text{mpBlasTrans,} \\ \alpha\boldsymbol{A}^T\boldsymbol{B}^H + \beta\boldsymbol{C}, & \text{for TransA} = \text{mpBlasTrans, TransB} = \text{mpBlasConjTrans,} \\ \alpha\boldsymbol{A}^H\boldsymbol{B} + \beta\boldsymbol{C}, & \text{for TransA} = \text{mpBlasConjTrans, TransB} = \text{mpBlasNoTrans,} \\ \alpha\boldsymbol{A}^H\boldsymbol{B}^T + \beta\boldsymbol{C}, & \text{for TransA} = \text{mpBlasConjTrans, TransB} = \text{mpBlasTrans,} \\ \alpha\boldsymbol{A}^H\boldsymbol{B}^H + \beta\boldsymbol{C}, & \text{for TransA} = \text{mpBlasConjTrans, TransB} = \text{mpBlasConjTrans,} \end{cases} \tag{10.3.2}$$

10.3.2 Matrix-Matrix-Product and Sum (Symmetric/Hermitian Matrix A)

Function **RSymm**(***Side*** *As Integer,* ***Uplo*** *As Integer,* α *As mpNum,* ***A*** *As mpNum[,],* ***B*** *As mpNum[,],* β *As mpNum,* ***C*** *As mpNum[,]*) As mpNum

The function RSymm returns the matrix-matrix product and sum for a symmetric matrix.

Parameters:
Side: An indicator specifying the order of the multiplication.
Uplo: An indicator specifying whether the upper or lower triangle will be used.
α: A real scalar.
A: A matrix of real numbers.
B: A matrix of real numbers.
β: A real scalar.
C: A matrix of real numbers.

For the real scalars α and β, the real symmetric matrix $\boldsymbol{A}$, and the real general matrices $\boldsymbol{B}$ and $\boldsymbol{C}$, the function RSymm computes the matrix-matrix product and sum

$$\mathsf{RSymm} = \begin{cases} \alpha\boldsymbol{AB} + \beta\boldsymbol{C}, & \text{for Side} = \mathsf{mpBlasLeft} \\ \alpha\boldsymbol{BA} + \beta\boldsymbol{C}, & \text{for Side} = \mathsf{mpBlasRight} \end{cases} \tag{10.3.3}$$

When Uplo is 0 then the upper triangle and diagonal of $\boldsymbol{A}$ are used, and when Uplo is 1 then the lower triangle and diagonal of $\boldsymbol{A}$ are used.

Function **cplxSymm**(***Side*** *As Integer,* ***Uplo*** *As Integer,* α *As mpNum,* ***A*** *As mpNum[,],* ***B*** *As mpNum[,],* β *As mpNum,* ***C*** *As mpNum[,]*) As mpNum

The function cplxSymm returns the matrix-matrix product and sum for a symmetric matrix.

Parameters:
Side: An indicator specifying the order of the multiplication.
Uplo: An indicator specifying whether the upper or lower triangle will be used.
α: A complex scalar.
A: A matrix of complex numbers.
B: A matrix of complex numbers.
β: A complex scalar.
C: A matrix of complex numbers.

For the complex scalars α and β, the complex symmetric matrix $\boldsymbol{A}$, and the complex general matrices $\boldsymbol{B}$ and $\boldsymbol{C}$, the function cplxSymm computes the matrix-matrix product and sum

$$\mathsf{cplxSymm} = \begin{cases} \alpha\boldsymbol{AB} + \beta\boldsymbol{C}, & \text{for Side} = \mathsf{mpBlasLeft} \\ \alpha\boldsymbol{BA} + \beta\boldsymbol{C}, & \text{for Side} = \mathsf{mpBlasRight} \end{cases} \tag{10.3.4}$$

When Uplo is 0 then the upper triangle and diagonal of $\boldsymbol{A}$ are used, and when Uplo is 1 then the lower triangle and diagonal of $\boldsymbol{A}$ are used.

Function **cplxHemm**(***Side*** *As Integer,* ***Uplo*** *As Integer,* α *As mpNum,* ***A*** *As mpNum[,],* ***B*** *As mpNum[,],* β *As mpNum,* ***C*** *As mpNum[,]*) As mpNum

The function cplxHemm returns the matrix-matrix product and sum for a hermitian matrix.

Parameters:
Side: An indicator specifying the order of the multiplication.
Uplo: An indicator specifying whether the upper or lower triangle will be used.
α: A complex scalar.
A: A matrix of complex numbers.
B: A matrix of complex numbers.
β: A complex scalar.
C: A matrix of complex numbers.

For the complex scalars α and β, the complex hermitian matrix $\boldsymbol{A}$, and the complex general matrices $\boldsymbol{B}$ and $\boldsymbol{C}$, the function cplxHemm computes the matrix-matrix product and sum

$$\text{cplxHemm} = \begin{cases} \alpha\boldsymbol{AB} + \beta\boldsymbol{C}, & \text{for Side} = \text{mpBlasLeft} \\ \alpha\boldsymbol{BA} + \beta\boldsymbol{C}, & \text{for Side} = \text{mpBlasRight} \end{cases} \tag{10.3.5}$$

When Uplo is 0 then the upper triangle and diagonal of $\boldsymbol{A}$ are used, and when Uplo is 1 then the lower triangle and diagonal of $\boldsymbol{A}$ are used. The imaginary elements of the diagonal are automatically assumed to be zero and are not referenced.

10.3.3 Matrix-Matrix-Product (Triangular Matrix A)

Function **RTrmm**(***Side*** *As Integer,* ***Uplo*** *As Integer,* ***TransA*** *As Integer,* α *As mpNum,* ***A*** *As mpNum[,],* ***B*** *As mpNum[,]*) As mpNum

The function RTrmm returns the matrix-matrix produc for a triangular matrix.

Parameters:
Side: An indicator specifying the order of the multiplication.
Uplo: An indicator specifying whether the upper or lower triangle will be used.
TransA: An indicator specifying the multiplication.
α: A real scalar.
A: A matrix of real numbers.
B: A matrix of real numbers.

For the real scalar α, the real triangular matrix $\boldsymbol{A}$, and the real general matrix $\boldsymbol{B}$, the function RTrmm computes the matrix-matrix product

$$\mathsf{RTrmm} = \begin{cases} \alpha\boldsymbol{AB}, & \text{for Side} = \mathsf{mpBlasLeft}, \text{ TransA} = \mathsf{mpBlasNoTrans}, \\ \alpha\boldsymbol{BA}, & \text{for Side} = \mathsf{mpBlasRight}, \text{ TransA} = \mathsf{mpBlasNoTrans}, \\ \alpha\boldsymbol{A}^T\boldsymbol{B}, & \text{for Side} = \mathsf{mpBlasLeft}, \text{ TransA} = \mathsf{mpBlasTrans}, \\ \alpha\boldsymbol{BA}^T, & \text{for Side} = \mathsf{mpBlasRight}, \text{ TransA} = \mathsf{mpBlasTrans}, \end{cases} \tag{10.3.6}$$

When Uplo is 0 then the upper triangle of $\boldsymbol{A}$ is used, and when Uplo is 1 then the lower triangle of $\boldsymbol{A}$ is used. If Diag is 0 then the diagonal of $\boldsymbol{A}$ is used, but if Diag is 1 then the diagonal elements of the matrix $\boldsymbol{A}$ are taken as unity and are not referenced.

Function **cplxTrmm**(***Side*** *As Integer,* ***Uplo*** *As Integer,* ***TransA*** *As Integer,* α *As mpNum,* ***A*** *As mpNum[,],* ***B*** *As mpNum[,]*) As mpNum

The function cplxTrmm returns the matrix-matrix product for a triangular matrix.

Parameters:
Side: An indicator specifying the order of the multiplication.
Uplo: An indicator specifying whether the upper or lower triangle will be used.
TransA: An indicator specifying the multiplication.
α: A complex scalar.
A: A matrix of complex numbers.
B: A matrix of complex numbers.

For the complex scalar α, the complex triangular matrix $\boldsymbol{A}$, and the complex general matrix $\boldsymbol{B}$, the function cplxTrmm computes the matrix-matrix product

$$\mathsf{cplxTrmm} = \begin{cases} \alpha\boldsymbol{AB}, & \text{for Side} = \mathsf{mpBlasLeft}, \text{ TransA} = \mathsf{mpBlasNoTrans}, \\ \alpha\boldsymbol{BA}, & \text{for Side} = \mathsf{mpBlasRight}, \text{ TransA} = \mathsf{mpBlasNoTrans}, \\ \alpha\boldsymbol{A}^T\boldsymbol{B}, & \text{for Side} = \mathsf{mpBlasLeft}, \text{ TransA} = \mathsf{mpBlasTrans}, \\ \alpha\boldsymbol{BA}^T, & \text{for Side} = \mathsf{mpBlasRight}, \text{ TransA} = \mathsf{mpBlasTrans}, \\ \alpha\boldsymbol{A}^H\boldsymbol{B}, & \text{for Side} = \mathsf{mpBlasLeft}, \text{ TransA} = \mathsf{mpBlasConjTrans}, \\ \alpha\boldsymbol{BA}^H, & \text{for Side} = \mathsf{mpBlasRight}, \text{ TransA} = \mathsf{mpBlasConjTrans}, \end{cases} \tag{10.3.7}$$

When Uplo is 0 then the upper triangle of $\boldsymbol{A}$ is used, and when Uplo is 1 then the lower triangle of $\boldsymbol{A}$ is used. If Diag is 0 then the diagonal of $\boldsymbol{A}$ is used, but if Diag is 1 then the diagonal elements of the matrix $\boldsymbol{A}$ are taken as unity and are not referenced.

10.3.4 Inverse Matrix-Matrix-Product (Triangular Matrix A)

Function **RTrsm**(***Side*** *As Integer,* ***Uplo*** *As Integer,* ***TransA*** *As Integer,* α *As mpNum,* ***A*** *As mpNum[,],* ***B*** *As mpNum[,]*) As mpNum

The function RTrsm returns the inverse matrix-matrix product for a triangular matrix.

Parameters:
Side: An indicator specifying the order of the multiplication.
Uplo: An indicator specifying whether the upper or lower triangle will be used.
TransA: An indicator specifying the multiplication.
α: A real scalar.
A: A matrix of real numbers.
B: A matrix of real numbers.

For the real scalar α, the real triangular matrix $\boldsymbol{A}$, and the real general matrix $\boldsymbol{B}$, the function RTrsm computes the matrix-matrix product

$$\mathsf{RTrsm} = \begin{cases} \alpha \boldsymbol{A}^{-1}\boldsymbol{B}, & \text{for Side} = \mathsf{mpBlasLeft}, \text{ TransA} = \mathsf{mpBlasNoTrans}, \\ \alpha \boldsymbol{B}\boldsymbol{A}^{-1}, & \text{for Side} = \mathsf{mpBlasRight}, \text{ TransA} = \mathsf{mpBlasNoTrans}, \\ \alpha \left(\boldsymbol{A}^{T}\right)^{-1}\boldsymbol{B}, & \text{for Side} = \mathsf{mpBlasLeft}, \text{ TransA} = \mathsf{mpBlasTrans}, \\ \alpha \boldsymbol{B}\left(\boldsymbol{A}^{T}\right)^{-1}, & \text{for Side} = \mathsf{mpBlasRight}, \text{ TransA} = \mathsf{mpBlasTrans}, \end{cases} \tag{10.3.8}$$

When Uplo is 0 then the upper triangle of $\boldsymbol{A}$ is used, and when Uplo is 1 then the lower triangle of $\boldsymbol{A}$ is used. If Diag is 0 then the diagonal of $\boldsymbol{A}$ is used, but if Diag is 1 then the diagonal elements of the matrix $\boldsymbol{A}$ are taken as unity and are not referenced.

Function **cplxTrsm**(***Side*** *As Integer,* ***Uplo*** *As Integer,* ***TransA*** *As Integer,* α *As mpNum,* ***A*** *As mpNum[,],* ***B*** *As mpNum[,]*) As mpNum

The function cplxTrsm returns the inverse matrix-matrix product for a triangular matrix.

Parameters:
Side: An indicator specifying the order of the multiplication.
Uplo: An indicator specifying whether the upper or lower triangle will be used.
TransA: An indicator specifying the multiplication.
α: A complex scalar.
A: A matrix of complex numbers.
B: A matrix of complex numbers.

For the complex scalar α, the complex triangular matrix $\boldsymbol{A}$, and the complex general matrix $\boldsymbol{B}$, the function cplxTrsm computes the matrix-matrix product

$$\mathsf{cplxTrsm} = \begin{cases} \alpha \boldsymbol{A}^{-1}\boldsymbol{B}, & \text{for Side} = \mathsf{mpBlasLeft}, \text{ TransA} = \mathsf{mpBlasNoTrans}, \\ \alpha \boldsymbol{B}\boldsymbol{A}^{-1}, & \text{for Side} = \mathsf{mpBlasRight}, \text{ TransA} = \mathsf{mpBlasNoTrans}, \\ \alpha \left(\boldsymbol{A}^{T}\right)^{-1}\boldsymbol{B}, & \text{for Side} = \mathsf{mpBlasLeft}, \text{ TransA} = \mathsf{mpBlasTrans}, \\ \alpha \boldsymbol{B}\left(\boldsymbol{A}^{T}\right)^{-1}, & \text{for Side} = \mathsf{mpBlasRight}, \text{ TransA} = \mathsf{mpBlasTrans}, \\ \alpha \left(\boldsymbol{A}^{H}\right)^{-1}\boldsymbol{B}, & \text{for Side} = \mathsf{mpBlasLeft}, \text{ TransA} = \mathsf{mpBlasConjTrans}, \\ \alpha \boldsymbol{B}\left(\boldsymbol{A}^{H}\right)^{-1}, & \text{for Side} = \mathsf{mpBlasRight}, \text{ TransA} = \mathsf{mpBlasConjTrans}, \end{cases} \tag{10.3.9}$$

When Uplo is 0 then the upper triangle of $\boldsymbol{A}$ is used, and when Uplo is 1 then the lower triangle of $\boldsymbol{A}$ is used. If Diag is 0 then the diagonal of $\boldsymbol{A}$ is used, but if Diag is 1 then the diagonal elements of the matrix $\boldsymbol{A}$ are taken as unity and are not referenced.

10.3.5 Rank-k update (Symmetric/Hermitian Matrix C))

Function **Rsyrk**(***Uplo*** *As Integer,* ***Trans*** *As Integer,* α *As mpNum,* ***A*** *As mpNum[,],* β *As mpNum,* ***C*** *As mpNum[,]*) As mpNum

The function Rsyrk returns a rank-k update for a symmetric matrix.

Parameters:
Uplo: An indicator specifying whether the upper or lower triangle will be used.
Trans: An indicator specifying the multiplication.
α: A real scalar.
A: A matrix of real numbers.
β: A real scalar.
C: A matrix of real numbers.

For the real scalars α and β, the real symmetric matrix $\boldsymbol{C}$, and the real general matrix $\boldsymbol{A}$, the function Rsyrk computes a rank-k update of the symmetric matrix $\boldsymbol{C}$, defined as

$$\mathsf{Rsyrk} = \begin{cases} \alpha \boldsymbol{A}\boldsymbol{A}^T + \beta \boldsymbol{C}, & \text{for } \mathsf{Trans} = \mathsf{mpNoTrans} \\ \alpha \boldsymbol{A}^T\boldsymbol{A} + \beta \boldsymbol{C}, & \text{for } \mathsf{Trans} = \mathsf{mpTrans} \end{cases} \tag{10.3.10}$$

Since the matrix $\boldsymbol{C}$ is symmetric, only its upper half or lower half need to be stored. When Uplo is 0 then the upper triangle and diagonal of $\boldsymbol{C}$ are used, and when Uplo is 1 then the lower triangle and diagonal of $\boldsymbol{C}$ are used.

Function **cplxSyrk**(***Uplo*** *As Integer,* ***Trans*** *As Integer,* α *As mpNum,* ***A*** *As mpNum[,],* β *As mpNum,* ***C*** *As mpNum[,]*) As mpNum

The function cplxSyrk returns a rank-k update for a symmetric matrix.

Parameters:
Uplo: An indicator specifying whether the upper or lower triangle will be used.
Trans: An indicator specifying the multiplication.
α: A complex scalar.
A: A matrix of complex numbers.
β: A complex scalar.
C: A matrix of complex numbers.

For the complex scalars α and β, the complex symmetric matrix $\boldsymbol{C}$, and the complex general matrix $\boldsymbol{A}$, the function cplxSyrk computes a rank-k update of the complex matrix $\boldsymbol{C}$, defined as

$$\mathsf{cplxSyrk} = \begin{cases} \alpha \boldsymbol{A}\boldsymbol{A}^T + \beta \boldsymbol{C}, & \text{for } \mathsf{Trans} = \mathsf{mpNoTrans} \\ \alpha \boldsymbol{A}^T\boldsymbol{A} + \beta \boldsymbol{C}, & \text{for } \mathsf{Trans} = \mathsf{mpTrans} \end{cases} \tag{10.3.11}$$

Since the matrix $\boldsymbol{C}$ is symmetric, only its upper half or lower half need to be stored. When Uplo is 0 then the upper triangle and diagonal of $\boldsymbol{C}$ are used, and when Uplo is 1 then the lower triangle and diagonal of $\boldsymbol{C}$ are used.

Function **cplxHerk**(***Uplo*** *As Integer,* ***Trans*** *As Integer,* α *As mpNum,* ***A*** *As mpNum[,],* β *As mpNum,* ***C*** *As mpNum[,]*) As mpNum

The function cplxHerk returns a rank-k update for a hermitian matrix.

Parameters:
Uplo: An indicator specifying whether the upper or lower triangle will be used.
Trans: An indicator specifying the multiplication.
α: A complex scalar.
A: A matrix of complex numbers.
β: A complex scalar.
C: A matrix of complex numbers.

For the complex scalars α and β, the complex hermitian matrix $\boldsymbol{C}$, and the complex general matrix $\boldsymbol{A}$, the function cplxHerk computes a rank-k update of the hermitian matrix $\boldsymbol{C}$, defined as

$$\text{cplxHerk} = \begin{cases} \alpha \boldsymbol{A}\boldsymbol{A}^H + \beta \boldsymbol{C}, & \text{for Trans} = \text{mpNoTrans} \\ \alpha \boldsymbol{A}^H \boldsymbol{A} + \beta \boldsymbol{C}, & \text{for Trans} = \text{mpTrans} \end{cases} \tag{10.3.12}$$

Since the matrix $\boldsymbol{C}$ is hermitian, only its upper half or lower half need to be stored. When Uplo is 0 then the upper triangle and diagonal of $\boldsymbol{C}$ are used, and when Uplo is 1 then the lower triangle and diagonal of $\boldsymbol{C}$ are used.

10.3.6 Rank-2k update (Symmetric/Hermitian Matrix C)

Function **Rsyr2k**(***Uplo*** *As Integer,* ***Trans*** *As Integer,* α *As mpNum,* ***A*** *As mpNum[,],* ***B*** *As mpNum[,],* β *As mpNum,* ***C*** *As mpNum[,])* As mpNum

The function Rsyr2k returns a rank-k update for a symmetric matrix.

Parameters:
Uplo: An indicator specifying whether the upper or lower triangle will be used.
Trans: An indicator specifying the multiplication.
α: A real scalar.
A: A matrix of real numbers.
B: A matrix of real numbers.
β: A real scalar.
C: A matrix of real numbers.

For the real scalars α and β, the real symmetric matrix $\boldsymbol{C}$, and the real general matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, the function Rsyr2k computes a rank-k update of the symmetric matrix $\boldsymbol{C}$, defined as

$$\mathsf{Rsyr2k} = \begin{cases} \alpha\boldsymbol{A}\boldsymbol{B}^T + \alpha\boldsymbol{B}\boldsymbol{A}^T + \beta\boldsymbol{C}, & \text{for } \mathsf{Trans} = \mathsf{mpNoTrans} \\ \alpha\boldsymbol{A}^T\boldsymbol{B} + \alpha\boldsymbol{B}^T\boldsymbol{A} + \beta\boldsymbol{C}, & \text{for } \mathsf{Trans} = \mathsf{mpTrans} \end{cases} \tag{10.3.13}$$

Since the matrix $\boldsymbol{C}$ is symmetric/hermitian, only its upper half or lower half need to be stored. When Uplo is 0 then the upper triangle and diagonal of $\boldsymbol{C}$ are used, and when Uplo is 1 then the lower triangle and diagonal of $\boldsymbol{C}$ are used.

Function **cplxSyr2k**(***Uplo*** *As Integer,* ***Trans*** *As Integer,* α *As mpNum,* ***A*** *As mpNum[,],* ***B*** *As mpNum[,],* β *As mpNum,* ***C*** *As mpNum[,])* As mpNum

The function cplxSyr2k returns a rank-k update for a symmetric matrix.

Parameters:
Uplo: An indicator specifying whether the upper or lower triangle will be used.
Trans: An indicator specifying the multiplication.
α: A complex scalar.
A: A matrix of complex numbers.
B: A matrix of complex numbers.
β: A complex scalar.
C: A matrix of complex numbers.

For the complex scalars α and β, the complex symmetric matrix $\boldsymbol{C}$, and the complex general matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, the function cplxSyrk computes a rank-k update of the complex matrix $\boldsymbol{C}$, defined as

$$\mathsf{cplxSyrk} = \begin{cases} \alpha\boldsymbol{A}\boldsymbol{B}^T + \alpha\boldsymbol{B}\boldsymbol{A}^T + \beta\boldsymbol{C}, & \text{for } \mathsf{Trans} = \mathsf{mpNoTrans} \\ \alpha\boldsymbol{A}^T\boldsymbol{B} + \alpha\boldsymbol{B}^T\boldsymbol{A} + \beta\boldsymbol{C}, & \text{for } \mathsf{Trans} = \mathsf{mpTrans} \end{cases} \tag{10.3.14}$$

Since the matrix $\boldsymbol{C}$ is symmetric/hermitian, only its upper half or lower half need to be stored. When Uplo is 0 then the upper triangle and diagonal of $\boldsymbol{C}$ are used, and when Uplo is 1 then the lower triangle and diagonal of $\boldsymbol{C}$ are used.

Function **cplxHer2k**(***Uplo*** *As Integer,* ***Trans*** *As Integer,* α *As mpNum,* ***A*** *As mpNum[,],* ***B*** *As mpNum[,],* β *As mpNum,* ***C*** *As mpNum[,]*) As mpNum

The function cplxHer2k returns a rank-k update for a hermitian matrix.

Parameters:
Uplo: An indicator specifying whether the upper or lower triangle will be used.
Trans: An indicator specifying the multiplication.
α: A complex scalar.
A: A matrix of complex numbers.
B: A matrix of complex numbers.
β: A complex scalar.
C: A matrix of complex numbers.

For the complex scalars α and β, the complex hermitian matrix $\boldsymbol{C}$, and the complex general matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, the function cplxHer2k computes a rank-k update of the hermitian matrix $\boldsymbol{C}$, defined as

$$\text{cplxHer2k} = \begin{cases} \alpha \boldsymbol{A}\boldsymbol{B}^H + \alpha \boldsymbol{B}\boldsymbol{A}^H + \beta \boldsymbol{C}, & \text{for Trans} = \text{mpNoTrans} \\ \alpha \boldsymbol{A}^H\boldsymbol{B} + \alpha \boldsymbol{B}^H\boldsymbol{A} + \beta \boldsymbol{C}, & \text{for Trans} = \text{mpTrans} \end{cases} \tag{10.3.15}$$

Since the matrix $\boldsymbol{C}$ is symmetric/hermitian, only its upper half or lower half need to be stored. When Uplo is 0 then the upper triangle and diagonal of $\boldsymbol{C}$ are used, and when Uplo is 1 then the lower triangle and diagonal of $\boldsymbol{C}$ are used.

Chapter 11

Linear Solvers (based on Eigen)

Book reference: Golub & Van Loan (1996)

11.1 Cholesky Decomposition without Pivoting

11.1.1 Decomposition

Function **DecompCholeskyLLT**(***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*, ***UpLo*** *As Integer*, ***Output*** *As String*) As mpNumList

---

The function DecompCholeskyLLT returns the Cholesky decomposition $A = LL^* = U^*U$ of a matrix.

Parameters:
A: the real matrix of which we are computing the LL^T Cholesky decomposition.
B: A vector or matrix of real numbers.
UpLo: the triangular part that will be used for the decompositon: Lower (default) or Upper. The other triangular part won't be read.
Output: A string specifying the output options.

Function **cplxDecompCholeskyLLT**(***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*, ***UpLo*** *As Integer*, ***Output*** *As String*) As mpNumList

---

The function cplxDecompCholeskyLLT returns the Cholesky decomposition $A = LL^* = U^*U$ of a matrix.

Parameters:
A: the complex matrix of which we are computing the LL^T Cholesky decomposition.
B: A vector or complex of real numbers.
UpLo: the triangular part that will be used for the decompositon: Lower (default) or Upper. The other triangular part won't be read.
Output: A string specifying the output options.

These functions perform a LL^T Cholesky decomposition of a symmetric, positive definite matrix A such that $A = LL^* = U^*U$, where L is lower triangular. While the Cholesky decomposition is particularly useful to solve selfadjoint problems like $D^*Dx = b$, for that purpose, we recommend the Cholesky decomposition without square root which is more stable and even faster.

Nevertheless, this standard Cholesky decomposition remains useful in many other situations like generalised eigen problems with hermitian matrices.
Remember that Cholesky decompositions are not rank-revealing. This LLT decomposition is only stable on positive definite matrices, use LDLT instead for the semidefinite case. Also, do not use a Cholesky decomposition to determine whether a system of equations has a solution.

LLT¡ MatrixType, _UpLo ¿ & compute (const MatrixType & a)
Computes / recomputes the Cholesky decomposition $A = LL^* = U^*U$ of matrix
Returns: a reference to *this

ComputationInfo info ( ) const: Reports whether previous computation was successful.
Returns: Success if computation was succesful, NumericalIssue if the matrix.appears to be negative.

Traits::MatrixL **matrixL** ( ) const
Returns: a view of the lower triangular matrix L

const MatrixType& **matrixLLT** ( ) const inline
Returns: the LLT decomposition matrix
TODO: document the storage layout

Traits::MatrixU **matrixU** ( ) const
Returns: a view of the upper triangular matrix U

LLT¡_MatrixType,_UpLo¿ **rankUpdate** (const VectorType & v, const RealScalar & sigma)
Performs a rank one update (or dowdate) of the current decomposition. If $A = LL^*$ before the rank one update, then after it we have $LL^* = A + \sigma \times vv^*$ where v must be a vector of same dimension. References Eigen::NumericalIssue, and Eigen::Success.

MatrixType **reconstructedMatrix** ( ) const
Returns: the matrix represented by the decomposition, i.e., it returns the product: LL^*. This function is provided for debug purpose.

const internal::solve_retval¡LLT, Rhs¿ **solve** (const MatrixBase¡ Rhs ¿ & b) const
Returns: the solution x of $Ax = b$ using the current decomposition of A. Since this LLT class assumes anyway that the matrix A is invertible, the solution theoretically exists and is unique regardless of b.

11.1.2 Linear Solver

Function **SolveCholeskyLLT**(***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*, ***UpLo*** *As Integer*) As mpNum[]

The function SolveCholeskyLLT returns the solution x of $Ax = b$, based on a Cholesky decomposition.

Parameters:
A: A symmetric positive definite real matrix.
B: A real vector or matrix.
UpLo: the triangular part that will be used for the decompositon: Lower (default) or Upper. The other triangular part won't be read.

Function **cplxSolveCholeskyLLT**(***A*** *As mpNum[,],* ***B*** *As mpNum[,],* ***UpLo*** *As Integer*) As mpNum[]

The function cplxSolveCholeskyLLT returns the solution x of $Ax = b$, based on a Cholesky decomposition.

Parameters:
A: A symmetric positive definite complex matrix.
B: A complex vector or matrix.
UpLo: the triangular part that will be used for the decompositon: Lower (default) or Upper. The other triangular part won't be read.

11.1.3 Matrix Inversion

Function **InvertCholeskyLLT**(***A*** *As mpNum[,],* ***UpLo*** *As Integer*) As mpNum[]

The function InvertCholeskyLLT returns A^{-1}, the inverse of A, based on a Cholesky decomposition.

Parameters:
A: A symmetric positive definite real matrix.
UpLo: the triangular part that will be used for the decompositon: Lower (default) or Upper. The other triangular part won't be read.

Function **cplxInvertCholeskyLLT**(***A*** *As mpNum[,],* ***UpLo*** *As Integer*) As mpNum[]

The function cplxInvertCholeskyLLT returns A^{-1}, the inverse of A, based on a Cholesky decomposition.

Parameters:
A: A symmetric positive definite complex matrix.
UpLo: the triangular part that will be used for the decompositon: Lower (default) or Upper. The other triangular part won't be read.

11.1.4 Determinant

Function **DetCholeskyLLT**(***A*** *As mpNum[,],* ***UpLo*** *As Integer*) As mpNum

The function DetCholeskyLLT returns $|A|$, the determinant of A, based on a Cholesky decomposition.

Parameters:
A: A symmetric positive definite real matrix.
UpLo: the triangular part that will be used for the decompositon: Lower (default) or Upper. The other triangular part won't be read.

Function **cplxDetCholeskyLLT**(***A*** *As mpNum[,],* ***UpLo*** *As Integer*) As mpNum

The function cplxDetCholeskyLLT returns $|A|$, the determinant of A, based on a Cholesky decomposition.

Parameters:

A: A symmetric positive definite complex matrix.
UpLo: the triangular part that will be used for the decompositon: Lower (default) or Upper. The other triangular part won't be read.

11.1.5 Example

Example:

```
MatrixXd A(3,3);
A << 4,-1,2, -1,6,0, 2,0,5;
cout << "The matrix A is" << endl << A << endl;
LLT<MatrixXd> lltOfA(A);
// compute the Cholesky decomposition of AMatrixXd L = lltOfA.matrixL();
// retrieve factor L in the decomposition
// The previous two lines can also be written as "L = A.llt().matrixL()"
cout << "The Cholesky factor L is" << endl << L << endl;
cout << "To check this, let us compute L * L.transpose()" << endl;
cout << L * L.transpose() << endl;
cout << "This should equal the matrix A" << endl;
```

```
Output:
The matrix A is
4 -1  2
-1  6  0
2  0  5
The Cholesky factor L is
2     0     0
-0.5   2.4     0
1 0.209  1.99
To check this, let us compute L * L.transpose()
4 -1  2
-1  6  0
2  0  5
This should equal the matrix A
```

11.2 Cholesky Decomposition with Pivoting

11.2.1 Decomposition

Function **DecompCholeskyLDLT**(***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*, ***UpLo*** *As Integer*, ***Output*** *As String*) As mpNumList

---

The function DecompCholeskyLDLT returns the Cholesky decomposition with pivoting of $A = LL^* = U^*U$.

Parameters:
A: the real matrix of which we are computing the LL^T Cholesky decomposition.
B: A vector or matrix of real numbers.
UpLo: the triangular part that will be used for the decompositon: Lower (default) or Upper. The other triangular part won't be read.
Output: A string specifying the output options.

Function **cplxDecompCholeskyLDLT**(***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*, ***UpLo*** *As Integer*, ***Output*** *As String*) As mpNumList

---

The function cplxDecompCholeskyLDLT returns the Cholesky decomposition with pivoting of $A = LL^* = U^*U$.

Parameters:
A: the complex matrix of which we are computing the LL^T Cholesky decomposition.
B: A vector or complex of real numbers.
UpLo: the triangular part that will be used for the decompositon: Lower (default) or Upper. The other triangular part won't be read.
Output: A string specifying the output options.

Perform a robust Cholesky decomposition of a positive semidefinite or negative semidefinite matrix such that $A = P^T LDL^*P$, where P is a permutation matrix, L is lower triangular with a unit diagonal and D is a diagonal matrix.
The decomposition uses pivoting to ensure stability, so that L will have zeros in the bottom right $\text{rank}(A) - n$ submatrix. Avoiding the square root on D also stabilizes the computation.
Remember that Cholesky decompositions are not rank-revealing. Also, do not use a Cholesky decomposition to determine whether a system of equations has a solution.

LDLT¡ MatrixType, _UpLo ¿ & **compute** (const MatrixType & a)
Compute / recompute the LDLT decomposition $A = LDL^* = U^*DU$ of matrix

ComputationInfo **info** ( ) const
Reports whether previous computation was successful.
Returns Success if computation was succesful, NumericalIssue if the matrix.appears to be negative.

bool **isNegative** (void) const
Returns true if the matrix is negative (semidefinite)

bool **isPositive** ( ) const
Returns true if the matrix is positive (semidefinite)

Traits::MatrixL **matrixL** ( ) const

Returns a view of the lower triangular matrix L

const MatrixType& **matrixLDLT** ( ) const
Returns the internal LDLT decomposition matrix TODO: document the storage layout

Traits::MatrixU **matrixU** ( ) const
Returns a view of the upper triangular matrix U

LDLT¡MatrixType,_UpLo¿& **rankUpdate** (const MatrixBase¡ Derived ¿ & w, const typename NumTraits¡ typename MatrixType::Scalar ¿::Real & sigma)
Update the LDLT decomposition: given $A = LDL^T$, efficiently compute the decomposition of $A + \sigma ww^T$.
Parameters: w a vector to be incorporated into the decomposition. sigma a scalar, +1 for updates and -1 for "downdates," which correspond to removing previously-added column vectors. Optional; default value is +1.

MatrixType **reconstructedMatrix** ( ) const
Returns the matrix represented by the decomposition, i.e., it returns the product: $P^T LDL^* P$. This function is provided for debug purpose.

void **setZero** ( )
Clear any existing decomposition

const internal::solve_retval¡LDLT, Rhs¿ **solve** (const MatrixBase¡ Rhs ¿ & b) const
Returns a solution x of $Ax = b$ using the current decomposition of A.
This function also supports in-place solves using the syntax x = decompositionObject.solve(x) .
This method just tries to find as good a solution as possible. If you want to check whether a solution exists or if it is accurate, just call this function to get a result and then compute the error of this result, or use MatrixBase::isApprox() directly, for instance like this:
bool a_solution_exists = (A*result).isApprox(b, precision);
This method avoids dividing by zero, so that the non-existence of a solution doesn't by itself mean that you'll get inf or nan values.
More precisely, this method solves $Ax = b$ using the decomposition $A = P^T LDL^* P$ by solving the systems $P^T y_1 = b, LY_2 = y_1, Dy_3 = y_2$, $L^* y_4 = y_3$ and $Px = y_4$ in succession. If the matrix A is singular, then D will also be singular (all the other matrices are invertible). In that case, the least-square solution of $Dy_3 = y_2$ is computed. This does not mean that this function computes the least-square solution of $Ax = b$ if A is singular.

const TranspositionType& **transpositionsP** ( ) const
Returns the permutation matrix P as a transposition sequence.

Diagonal¡const MatrixType¿ **vectorD** ( ) const
Returns the coefficients of the diagonal matrix D

11.2.2 Linear Solver

Function **SolveCholeskyLDLT**(***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*, ***UpLo*** *As Integer*) As mpNum[]

The function SolveCholeskyLDLT returns the solution x of $Ax = b$, based on a Cholesky decomposition with pivoting.

Parameters:
A: A symmetric positive definite real matrix.

B: A real vector or matrix.
UpLo: the triangular part that will be used for the decompositon: Lower (default) or Upper. The other triangular part won't be read.

Function **cplxSolveCholeskyLDLT**(***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*, ***UpLo*** *As Integer*) As mpNum[]

The function cplxSolveCholeskyLDLT returns the solution x of $Ax = b$, based on a Cholesky decomposition with pivoting.

Parameters:
A: A symmetric positive definite complex matrix.
B: A complex vector or matrix.
UpLo: the triangular part that will be used for the decompositon: Lower (default) or Upper. The other triangular part won't be read.

11.2.3 Matrix Inversion

Function **InvertCholeskyLDLT**(***A*** *As mpNum[,]*, ***UpLo*** *As Integer*) As mpNum[]

The function InvertCholeskyLDLT returns A^{-1}, the inverse of A, based on a Cholesky decomposition with pivoting.

Parameters:
A: A symmetric positive definite real matrix.
UpLo: the triangular part that will be used for the decompositon: Lower (default) or Upper. The other triangular part won't be read.

Function **cplxInvertCholeskyLDLT**(***A*** *As mpNum[,]*, ***UpLo*** *As Integer*) As mpNum[]

The function cplxInvertCholeskyLDLT returns A^{-1}, the inverse of A, based on a Cholesky decomposition with pivoting.

Parameters:
A: A symmetric positive definite complex matrix.
UpLo: the triangular part that will be used for the decompositon: Lower (default) or Upper. The other triangular part won't be read.

11.2.4 Determinant

Function **DetCholeskyLDLT**(***A*** *As mpNum[,]*, ***UpLo*** *As Integer*) As mpNum

---

The function DetCholeskyLDLT returns $|A|$, the determinant of A, based on a Cholesky decomposition.

Parameters:
A: A symmetric positive definite real matrix.
UpLo: the triangular part that will be used for the decompositon: Lower (default) or Upper. The other triangular part won't be read.

Function **cplxDetCholeskyLDLT**(***A*** *As mpNum[,]*, ***UpLo*** *As Integer*) As mpNum

The function cplxDetCholeskyLDLT returns $|A|$, the determinant of A, based on a Cholesky decomposition.

Parameters:
A: A symmetric positive definite complex matrix.
UpLo: the triangular part that will be used for the decompositon: Lower (default) or Upper. The other triangular part won't be read.

11.2.5 Example

Example:

```
MatrixXd A(3,3);
A << 4,-1,2, -1,6,0, 2,0,5;
cout << "The matrix A is" << endl << A << endl;
LLT<MatrixXd> lltOfA(A);
// compute the Cholesky decomposition of AMatrixXd L = lltOfA.matrixL();
// retrieve factor L in the decomposition
// The previous two lines can also be written as "L = A.llt().matrixL()"
cout << "The Cholesky factor L is" << endl << L << endl;
cout << "To check this, let us compute L * L.transpose()" << endl;
cout << L * L.transpose() << endl;
cout << "This should equal the matrix A" << endl;
```

```
Output:
The matrix A is
4 -1  2
-1  6  0
2  0  5
The Cholesky factor L is
2     0     0
-0.5   2.4     0
1 0.209  1.99
To check this, let us compute L * L.transpose()
4 -1  2
-1  6  0
2  0  5
This should equal the matrix A
```

11.3 LU Decomposition with partial Pivoting

11.3.1 Decomposition

Function **DecompPartialPivLU**(***A*** *As mpNum[,],* ***B*** *As mpNum[,],* ***Output*** *As String*) As mpNumList

The function DecompPartialPivLU returns the LU decomposition with partial pivoting of $A = PLU$.

Parameters:
A: the square real matrix of which we are computing the LU decomposition.
B: A vector or matrix of real numbers.
Output: A string specifying the output options.

Function **cplxDecompPartialPivLU**(***A*** *As mpNum[,],* ***B*** *As mpNum[,],* ***Output*** *As String*) As mpNumList

The function cplxDecompPartialPivLU returns the LU decomposition with partial pivoting of $A = PLU$.

Parameters:
A: the square complex matrix of which we are computing the LU decomposition.
B: A vector or complex of real numbers.
Output: A string specifying the output options.

This class represents a LU decomposition of a square invertible matrix, with partial pivoting: the matrix A is decomposed as $A = PLU$ where L is unit-lower-triangular, U is upper-triangular, and P is a permutation matrix.
Typically, partial pivoting LU decomposition is only considered numerically stable for square invertible matrices. Thus LAPACK's dgesv and dgesvx require the matrix to be square and invertible. The present class does the same. It will assert that the matrix is square, but it won't (actually it can't) check that the matrix is invertible: it is your task to check that you only use this decomposition on invertible matrices.
The guaranteed safe alternative, working for all matrices, is the full pivoting LU decomposition, provided by class FullPivLU.
This is not a rank-revealing LU decomposition. Many features are intentionally absent from this class, such as rank computation. If you need these features, use class FullPivLU.
This LU decomposition is suitable to invert invertible matrices. It is what MatrixBase::inverse() uses in the general case. On the other hand, it is not suitable to determine whether a given matrix is invertible.
The data of the LU decomposition can be directly accessed through the methods matrixLU(), permutationP().
Returns the determinant of the matrix of which *this is the LU decomposition. It has only linear complexity (that is, O(n) where n is the dimension of the square matrix) as the LU decomposition has already been computed.
Warning: a determinant can be very big or small, so for matrices of large enough dimension, there is a risk of overflow/underflow.See Also MatrixBase::determinant()

const internal::solve_retval¡PartialPivLU,typename MatrixType::IdentityReturnType¿ **inverse** (

) const
Returns the inverse of the matrix of which *this is the LU decomposition.
Warning: The matrix being decomposed here is assumed to be invertible. If you need to check for invertibility, use class FullPivLU instead.

const MatrixType& **matrixLU** ( ) const
Returns the LU decomposition matrix: the upper-triangular part is U, the unit-lower-triangular part is L (at least for square matrices; in the non-square case, special care is needed, see the documentation of class FullPivLU).

const PermutationType& **permutationP** ( ) const
Returns the permutation matrix P.

MatrixType **reconstructedMatrix** ( ) const
Returns the matrix represented by the decomposition, i.e., it returns the product: $P^{-1}LU$. This function is provided for debug purpose.

const internal::solve_retval¡PartialPivLU, Rhs¿ **solve** (const MatrixBase¡ Rhs ¿ & b) const
This method returns the solution x to the equation $Ax = b$, where A is the matrix of which *this is the LU decomposition.
Parameters: b the right-hand-side of the equation to solve. Can be a vector or a matrix, the only requirement in order for the equation to make sense is that b.rows()==A.rows(), where A is the matrix of which *this is the LU decomposition.
Returns the solution.

11.3.2 Linear Solver

Function **SolvePartialPivLU**(***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*) As mpNum[]

The function SolvePartialPivLU returns the solution x of $Ax = b$, based on a LU decomposition with partial pivoting.

Parameters:
A: A square real matrix.
B: A real vector or matrix.

Function **cplxSolvePartialPivLU**(***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*) As mpNum[]

The function cplxSolvePartialPivLU returns the solution x of $Ax = b$, based on a LU decomposition with partial pivoting.

Parameters:
A: A square complex matrix.
B: A complex vector or matrix.

11.3.3 Matrix Inversion

Function **InvertPartialPivLU**(***A*** *As mpNum[,]*) As mpNum[]

The function InvertPartialPivLU returns A^{-1}, the inverse of A, based on a LU decomposition with partial pivoting.

Parameter:
A: A square real matrix.

Function **cplxInvertPartialPivLU**(***A*** *As mpNum[,]*) As mpNum[]

The function cplxInvertPartialPivLU returns A^{-1}, the inverse of A, based on a LU decomposition with partial pivoting.

Parameter:
A: A square complex matrix.

11.3.4 Determinant

Function **DetPartialPivLU**(***A*** *As mpNum[,]*) As mpNum

The function DetPartialPivLU returns $|A|$, the determinant of A, based on a LU decomposition with partial pivoting.

Parameter:
A: A square real matrix.

Function **cplxDetPartialPivLU**(***A*** *As mpNum[,]*) As mpNum

---

The function cplxDetPartialPivLU returns $|A|$, the determinant of A, based on a LU decomposition with partial pivoting.

Parameter:
A: A square complex matrix.

11.3.5 Example

Example:

```
MatrixXd A = MatrixXd::Random(3,3);
MatrixXd B = MatrixXd::Random(3,2);
cout << "Here is the invertible matrix A:" << endl << A << endl;
cout << "Here is the matrix B:" << endl << B << endl;
MatrixXd X = A.lu().solve(B);
cout << "Here is the (unique) solution X to the equation AX=B:"
<< endl << X << endl;
cout << "Relative error: " << (A*X-B).norm() / B.norm() << endl;
```

```
Output:
Here is the invertible matrix A:
0.68  0.597  -0.33
-0.211  0.823  0.536
0.566 -0.605 -0.444
Here is the matrix B:
0.108   -0.27
-0.0452  0.0268
```

```
0.258   0.904
Here is the (unique) solution X to the equation AX=B:
0.609   2.68
-0.231  -1.57
0.51   3.51
Relative error: 3.28e-16
```

11.4 LU Decomposition with full Pivoting

11.4.1 Decomposition

Function **DecompFullPivLU**(***A*** *As mpNum[,],* ***B*** *As mpNum[,],* ***Output*** *As String*) As mpNumList

The function DecompFullPivLU returns the LU decomposition with full pivoting of $A = PLUQ$.

Parameters:
A: the square real matrix of which we are computing the LU decomposition.
B: A vector or matrix of real numbers.
Output: A string specifying the output options.

Function **cplxDecompFullPivLU**(***A*** *As mpNum[,],* ***B*** *As mpNum[,],* ***Output*** *As String*) As mpNumList

The function cplxDecompFullPivLU returns the LU decomposition with full pivoting of $A = PLUQ$.

Parameters:
A: the square complex matrix of which we are computing the LU decomposition.
B: A vector or complex of real numbers.
Output: A string specifying the output options.

This class represents a LU decomposition of any matrix, with complete pivoting: the matrix A is decomposed as $A = PLUQ$ where L is unit-lower-triangular, U is upper-triangular, and P and Q are permutation matrices. This is a rank-revealing LU decomposition. The eigenvalues (diagonal coefficients) of U are sorted in such a way that any zeros are at the end.
This decomposition provides the generic approach to solving systems of linear equations, computing the rank, invertibility, inverse, kernel, and determinant. This LU decomposition is very stable and well tested with large matrices. However there are use cases where the SVD decomposition is inherently more stable and/or flexible. For example, when computing the kernel of a matrix, working with the SVD allows to select the smallest singular values of the matrix, something that the LU decomposition doesn't see.
The data of the LU decomposition can be directly accessed through the methods matrixLU(), permutationP(), permutationQ().
Computes the LU decomposition of the given matrix.
Parameters: matrix the matrix of which to compute the LU decomposition. It is required to be nonzero.
Returns: a reference to *this Referenced by FullPivLU¡ MatrixType ¿::FullPivLU().

internal::traits¡ MatrixType ¿::Scalar **determinant** ( ) const
Returns the determinant of the matrix of which *this is the LU decomposition. It has only linear complexity (that is, O(n) where n is the dimension of the square matrix) as the LU decomposition has already been computed.
For fixed-size matrices of size up to 4, MatrixBase::determinant() offers optimized paths.
Warning: a determinant can be very big or small, so for matrices of large enough dimension, there is a risk of overflow/underflow.

Index **dimensionOfKernel** ( ) const

Returns the dimension of the kernel of the matrix of which *this is the LU decomposition. Note: This method has to determine which pivots should be considered nonzero. For that, it uses the threshold value that you can control by calling setThreshold(const RealScalar&). References FullPivLU¡ MatrixType ¿::rank().

const internal::image_retval¡FullPivLU¿ **image** (const MatrixType & originalMatrix) const
Returns the image of the matrix, also called its column-space. The columns of the returned matrix will form a basis of the kernel.
Parameters: originalMatrix the original matrix, of which *this is the LU decomposition. The reason why it is needed to pass it here, is that this allows a large optimization, as otherwise this method would need to reconstruct it from the LU decomposition.
Note: If the image has dimension zero, then the returned matrix is a column-vector filled with zeros. This method has to determine which pivots should be considered nonzero. For that, it uses the threshold value that you can control by calling setThreshold(const RealScalar&).
Example:

```
Matrix3d m;m << 1,1,0,  1,3,2,   0,1,1;
cout << "Here is the matrix m:" << endl << m << endl;
cout << "Notice that the middle column is the sum of the two others, so the "
<< "columns are linearly dependent." << endl;
cout << "Here is a matrix whose columns have the same span but are linearly
    independent:"
<< endl << m.fullPivLu().image(m) << endl;
```

```
Output:
Here is the matrix m:
1 1 0
1 3 2
0 1 1
Notice that the middle column is the sum of the two others, so the columns are linearly d
Here is a matrix whose columns have the same span but are linearly independent:
1 1
3 1
1 0
```

const internal::solve_retval¡FullPivLU,typename MatrixType::IdentityReturnType¿ **inverse** ( ) const
Returns the inverse of the matrix of which *this is the LU decomposition. Note: If this matrix is not invertible, the returned matrix has undefined coefficients. Use isInvertible() to first determine whether this matrix is invertible.

bool **isInjective** ( ) const
Returns true if the matrix of which *this is the LU decomposition represents an injective linear map, i.e. has trivial kernel; false otherwise. Note: This method has to determine which pivots should be considered nonzero. For that, it uses the threshold value that you can control by calling setThreshold(const RealScalar&). References FullPivLU¡ MatrixType ¿::rank().

bool **isInvertible** ( ) const
Returns true if the matrix of which *this is the LU decomposition is invertible. Note: This method has to determine which pivots should be considered nonzero. For that, it uses the threshold

value that you can control by calling setThreshold(const RealScalar&). References FullPivLU¡ MatrixType ¿::isInjective().

bool **isSurjective** ( ) const
Returns true if the matrix of which *this is the LU decomposition represents a surjective linear map; false otherwise. Note: This method has to determine which pivots should be considered nonzero. For that, it uses the threshold value that you can control by calling setThreshold(const RealScalar&). References FullPivLU¡ MatrixType ¿::rank().

const internal::kernel_retval¡FullPivLU¿ **kernel** ( ) const
Returns the kernel of the matrix, also called its null-space. The columns of the returned matrix will form a basis of the kernel. Note: If the kernel has dimension zero, then the returned matrix is a column-vector filled with zeros. This method has to determine which pivots should be considered nonzero. For that, it uses the threshold value that you can control by calling setThreshold(const RealScalar&).
Example:

```
MatrixXf m = MatrixXf::Random(3,5);
cout << "Here is the matrix m:" << endl << m << endl;
MatrixXf ker = m.fullPivLu().kernel();
cout << "Here is a matrix whose columns form a basis of the kernel of m:"
<< endl << ker << endl;cout << "By definition of the kernel, m*ker is zero:"
<< endl << m*ker << endl;
```

```
Output:
Here is the matrix m:
0.68   0.597   -0.33   0.108   -0.27
-0.211   0.823   0.536 -0.0452  0.0268
0.566  -0.605  -0.444   0.258   0.904
Here is a matrix whose columns form a basis of the kernel of m:
-0.219   0.763
0.00335  -0.447
0        1
1        0
-0.145  -0.285
By definition of the kernel, m*ker is zero:
-1.12e-08  1.49e-08
-1.4e-09 -4.05e-08
1.49e-08 -2.98e-08
```

const MatrixType& **matrixLU** ( ) const
Returns the LU decomposition matrix: the upper-triangular part is U, the unit-lower-triangular part is L (at least for square matrices; in the non-square case, special care is needed, see the documentation of class FullPivLU).

RealScalar **maxPivot** ( ) const
Returns the absolute value of the biggest pivot, i.e. the biggest diagonal coefficient of U.

Index **nonzeroPivots** ( ) const

Returns the number of nonzero pivots in the LU decomposition. Here nonzero is meant in the exact sense, not in a fuzzy sense. So that notion isn't really intrinsically interesting, but it is still useful when implementing algorithms.

const PermutationPType& **permutationP** ( ) const inline
Returns the permutation matrix P

const PermutationQType& **permutationQ** ( ) const
Returns the permutation matrix Q

Index **rank** ( ) const
Returns the rank of the matrix of which *this is the LU decomposition. Note: This method has to determine which pivots should be considered nonzero. For that, it uses the threshold value that you can control by calling setThreshold(const RealScalar&). References FullPivLU¡ MatrixType ¿::threshold().

MatrixType **reconstructedMatrix** ( ) const
Returns the matrix represented by the decomposition, i.e., it returns the product: $P^{-1}LUQ^{-1}$. This function is provided for debug purpose.

FullPivLU& **setThreshold** (const RealScalar & threshold)
Allows to prescribe a threshold to be used by certain methods, such as rank(), who need to determine when pivots are to be considered nonzero. This is not used for the LU decomposition itself. When it needs to get the threshold value, Eigen calls threshold(). By default, this uses a formula to automatically determine a reasonable threshold. Once you have called the present method setThreshold(const RealScalar&), your value is used instead.
Parameters: threshold The new value to use as the threshold.
A pivot will be considered nonzero if its absolute value is strictly greater than where maxpivot is the biggest pivot. If you want to come back to the default behavior, call setThreshold(Default_t) References FullPivLU¡ MatrixType ¿::threshold().

FullPivLU& **setThreshold** (Default_t)
Allows to come back to the default behavior, letting Eigen use its default formula for determining the threshold. You should pass the special object Eigen::Default as parameter here. lu.setThreshold(Eigen::Default); See the documentation of setThreshold(const RealScalar&).

const internal::solve_retval¡FullPivLU, Rhs¿ **solve** (const MatrixBase¡ Rhs ¿ & b) const
Returns a solution x to the equation $Ax = b$, where A is the matrix of which *this is the LU decomposition. Parameters: b the right-hand-side of the equation to solve. Can be a vector or a matrix, the only requirement in order for the equation to make sense is that b.rows()==A.rows(), where A is the matrix of which *this is the LU decomposition.
Returns a solution. This method just tries to find as good a solution as possible. If you want to check whether a solution exists or if it is accurate, just call this function to get a result and then compute the error of this result, or use MatrixBase::isApprox() directly, for instance like this:

```
bool a\_solution\_exists = (A*result).isApprox(b, precision);
```

This method avoids dividing by zero, so that the non-existence of a solution doesn't by itself mean that you'll get inf or nan values. If there exists more than one solution, this method will arbitrarily choose one. If you need a complete analysis of the space of solutions, take the one solution obtained by this method and add to it elements of the kernel, as determined by kernel(). Example:

```
Matrix<float,2,3> m = Matrix<float,2,3>::Random();
Matrix2f y = Matrix2f::Random();
cout << "Here is the matrix m:" << endl << m << endl;
cout << "Here is the matrix y:" << endl << y << endl;
Matrix<float,3,2> x = m.fullPivLu().solve(y);
if((m*x).isApprox(y))
{  cout << "Here is a solution x to the equation mx=y:" << endl << x << endl;}
else  cout << "The equation mx=y does not have any solution." << endl;
```

```
Output:
Here is the matrix m:
0.68  0.566  0.823
-0.211  0.597 -0.605
Here is the matrix y:
-0.33 -0.444
0.536  0.108
Here is a solution x to the equation mx=y:
0      0
0.291 -0.216
-0.6 -0.391
```

RealScalar **threshold** ( ) const
Returns the threshold that will be used by certain methods such as rank(). See the documentation of **setThreshold**(const RealScalar&).

11.4.2 Linear Solver

Function **SolveFullPivLU**(***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*) As mpNum[]

The function SolveFullPivLU returns the solution x of $Ax = b$, based on a LU decomposition with full pivoting.

Parameters:
A: A square real matrix.
B: A real vector or matrix.

Function **cplxSolveFullPivLU**(***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*) As mpNum[]

The function cplxSolveFullPivLU returns the solution x of $Ax = b$, based on a LU decomposition with full pivoting.

Parameters:
A: A square complex matrix.
B: A complex vector or matrix.

11.4.3 Matrix Inversion

Function **InvertFullPivLU**(***A*** *As mpNum[,]*) As mpNum[]

The function InvertFullPivLU returns A^{-1}, the inverse of A, based on a LU decomposition with full pivoting.

Parameter:
A: A square real matrix.

Function **cplxInvertFullPivLU**(***A*** *As mpNum[,]*) As mpNum[]

The function cplxInvertFullPivLU returns A^{-1}, the inverse of A, based on a LU decomposition with full pivoting.

Parameter:
A: A square complex matrix.

11.4.4 Determinant

Function **DetFullPivLU**(***A*** *As mpNum[,]*) As mpNum

The function DetFullPivLU returns $|A|$, the determinant of A, based on a LU decomposition with full pivoting.

Parameter:
A: A square real matrix.

Function **cplxDetFullPivLU**(***A*** *As mpNum[,]*) As mpNum

---

The function cplxDetFullPivLU returns $|A|$, the determinant of A, based on a LU decomposition with full pivoting.

Parameter:
A: A square complex matrix.

11.5 QR Decomposition without Pivoting

11.5.1 Decomposition

Function **DecompQR**(***A*** *As mpNum[,],* ***B*** *As mpNum[,],* ***Output*** *As String*) As mpNumList

The function DecompQR returns the QR decomposition without pivoting of $A = QR$.

Parameters:
A: the square real matrix of which we are computing the LU decomposition.
B: A vector or matrix of real numbers.
Output: A string specifying the output options.

Function **cplxDecompQR**(***A*** *As mpNum[,],* ***B*** *As mpNum[,],* ***Output*** *As String*) As mpNumList

The function cplxDecompQR returns the QR decomposition without pivoting of $A = QR$.

Parameters:
A: the square complex matrix of which we are computing the LU decomposition.
B: A vector or complex of real numbers.
Output: A string specifying the output options.

This class performs a QR decomposition of a matrix A into matrices Q and R such that

$$A = QR \tag{11.5.1}$$

by using Householder transformations. Here, Q a unitary matrix and R an upper triangular matrix. The result is stored in a compact way compatible with LAPACK. Note that no pivoting is performed. This is not a rank-revealing decomposition. If you want that feature, use FullPivHouseholderQR or ColPivHouseholderQR instead.
This Householder QR decomposition is faster, but less numerically stable and less feature-rich than FullPivHouseholderQR or ColPivHouseholderQR.
Member Function DocumentationMatrixType::RealScalar **absDeterminant** ( ) const
Returns the absolute value of the determinant of the matrix of which *this is the QR decomposition. It has only linear complexity (that is, O(n) where n is the dimension of the square matrix) as the QR decomposition has already been computed. Note: This is only for square matrices. Warning: a determinant can be very big or small, so for matrices of large enough dimension, there is a risk of overflow/underflow. One way to work around that is to use logAbsDeterminant() instead. See Also logAbsDeterminant(), MatrixBase::determinant()

HouseholderQR¡ MatrixType ¿ & **compute** (const MatrixType & matrix)
Performs the QR factorization of the given matrix matrix. The result of the factorization is stored into *this, and a reference to *this is returned. See Also: class HouseholderQR, HouseholderQR(const MatrixType&)

const HCoeffsType& **hCoeffs** ( ) const
Returns a const reference to the vector of Householder coefficients used to represent the factor Q. For advanced uses only.

HouseholderSequenceType **householderQ** (void) const
This method returns an expression of the unitary matrix Q as a sequence of Householder transformations. The returned expression can directly be used to perform matrix products. It can also

be assigned to a dense Matrix object. Here is an example showing how to recover the full or thin matrix Q, as well as how to perform matrix products using operator*:
Example:

```
MatrixXf A(MatrixXf::Random(5,3)), thinQ(MatrixXf::Identity(5,3)),
Q;A.setRandom();
HouseholderQR<MatrixXf> qr(A);
Q = qr.householderQ();
thinQ = qr.householderQ() * thinQ;
std::cout << "The complete unitary matrix Q is:\n" << Q << "\n\n";
std::cout << "The thin matrix Q is:\n" << thinQ << "\n\n";
```

```
Output:
The complete unitary matrix Q is:
-0.676   0.0793    0.713  -0.0788   -0.147
-0.221   -0.322    -0.37   -0.366   -0.759
-0.353   -0.345   -0.214    0.841  -0.0518
0.582   -0.462    0.555    0.176   -0.329
-0.174   -0.747 -0.00907   -0.348    0.539

The thin matrix Q is:
-0.676   0.0793    0.713
-0.221   -0.322    -0.37
-0.353   -0.345   -0.214
0.582   -0.462    0.555
-0.174   -0.747 -0.00907
```

MatrixType::RealScalar **logAbsDeterminant** ( ) const
Returns the natural log of the absolute value of the determinant of the matrix of which *this is the QR decomposition. It has only linear complexity (that is, O(n) where n is the dimension of the square matrix) as the QR decomposition has already been computed. Note: This is only for square matrices. This method is useful to work around the risk of overflow/underflow that's inherent to determinant computation.See Also absDeterminant(), MatrixBase::determinant()

const MatrixType& **matrixQR** ( ) const
Returns a reference to the matrix where the Householder QR decomposition is stored in a LAPACK-compatible way.
const internal::solve_retval¡HouseholderQR, Rhs¿ **solve** (const MatrixBase¡ Rhs ¿ & b) const
This method finds a solution x to the equation $Ax = b$, where A is the matrix of which *this is the QR decomposition, if any exists. Parameters: b the right-hand-side of the equation to solve. Returns a solution. Note: The case where b is a matrix is not yet implemented. Also, this code is space inefficient.This method just tries to find as good a solution as possible. If you want to check whether a solution exists or if it is accurate, just call this function to get a result and then compute the error of this result, or use MatrixBase::isApprox() directly, for instance like this:

```
bool a\_solution\_exists = (A*result).isApprox(b, precision);
```

This method avoids dividing by zero, so that the non-existence of a solution doesn't by itself mean that you'll get inf or nan values. If there exists more than one solution, this method will arbitrarily choose one.
Example:

```
typedef Matrix<float,3,3> Matrix3x3;
Matrix3x3 m = Matrix3x3::Random();
Matrix3f y = Matrix3f::Random();
cout << "Here is the matrix m:" << endl << m << endl;
cout << "Here is the matrix y:" << endl << y << endl;
Matrix3f x;
x = m.householderQr().solve(y);
assert(y.isApprox(m*x));
cout << "Here is a solution x to the equation mx=y:" << endl << x << endl;
```

```
Output:
Here is the matrix m:
0.68  0.597  -0.33
-0.211  0.823  0.536
0.566 -0.605 -0.444
Here is the matrix y:
0.108   -0.27   0.832
-0.0452  0.0268   0.271
0.258   0.904   0.435
Here is a solution x to the equation mx=y:
0.609   2.68   1.67
-0.231  -1.57 0.0713
0.51   3.51   1.05
```

11.5.2 Linear Solver

Function **SolveQR**(***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*) As mpNum[]

The function SolveQR returns the solution x of $Ax = b$, based on a QR decomposition without pivoting.

Parameters:
A: A square real matrix.
B: A real vector or matrix.

Function **cplxSolveQR**(***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*) As mpNum[]

The function cplxSolveQR returns the solution x of $Ax = b$, based on a QR decomposition without pivoting.

Parameters:
A: A square complex matrix.
B: A complex vector or matrix.

11.5.3 Matrix Inversion

Function **InvertQR**(***A*** *As mpNum[,]*) As mpNum[]

The function InvertQR returns A^{-1}, the inverse of A, based on a QR decomposition without pivoting.

Parameter:
A: A square real matrix.

Function **cplxInvertQR**(***A*** *As mpNum[,]*) As mpNum[]

The function cplxInvertQR returns A^{-1}, the inverse of A, based on a QR decomposition without pivoting.

Parameter:
A: A square complex matrix.

11.5.4 Determinant

Function **DetQR**(***A*** *As mpNum[,]*) As mpNum

The function DetQR returns $|A|$, the determinant of A, based on a QR decomposition without pivoting.

Parameter:
A: A square real matrix.

Function **cplxDetQR**(***A*** *As mpNum[,]*) As mpNum

The function cplxDetQR returns $|A|$, the determinant of A, based on a QR decomposition without pivoting.

Parameter:
A: A square complex matrix.

11.5.5 Example

Example:

```
Example
```

11.6 QR Decomposition with column Pivoting

11.6.1 Decomposition

Function **DecompColPivQR**(***A*** *As mpNum[,],* ***B*** *As mpNum[,],* ***Output*** *As String*) As mpNumList

The function DecompColPivQR returns the QR decomposition with column-pivoting of $A = QR$.

Parameters:
A: the square real matrix of which we are computing the LU decomposition.
B: A vector or matrix of real numbers.
Output: A string specifying the output options.

Function **cplxDecompColPivQR**(***A*** *As mpNum[,],* ***B*** *As mpNum[,],* ***Output*** *As String*) As mpNumList

The function cplxDecompColPivQR returns the QR decomposition with column-pivoting of $A = QR$.

Parameters:
A: the square complex matrix of which we are computing the LU decomposition.
B: A vector or complex of real numbers.
Output: A string specifying the output options.

This class performs a rank-revealing QR decomposition of a matrix A into matrices P, Q and R such that

$$AP = QR \tag{11.6.1}$$

by using Householder transformations. Here, P is a permutation matrix, Q a unitary matrix and R an upper triangular matrix.
This decomposition performs column pivoting in order to be rank-revealing and improve numerical stability.
It is slower than HouseholderQR, and faster than FullPivHouseholderQR.
Member Function DocumentationMatrixType::RealScalar **absDeterminant** ( ) const
Returns the absolute value of the determinant of the matrix of which *this is the QR decomposition. It has only linear complexity (that is, O(n) where n is the dimension of the square matrix) as the QR decomposition has already been computed. Note: This is only for square matrices. Warning: a determinant can be very big or small, so for matrices of large enough dimension, there is a risk of overflow/underflow. One way to work around that is to use logAbsDeterminant() instead. See Also logAbsDeterminant(), MatrixBase::determinant()

const PermutationType& **colsPermutation** ( ) const
Returns a const reference to the column permutation matrix

ColPivHouseholderQR¡ MatrixType ¿ & **compute** (const MatrixType & matrix)
Performs the QR factorization of the given matrix matrix. The result of the factorization is stored into *this, and a reference to *this is returned. See Also class ColPivHouseholderQR, ColPivHouseholderQR(const MatrixType&)

Index **dimensionOfKernel** ( ) const
Returns the dimension of the kernel of the matrix of which *this is the QR decomposition. Note: This method has to determine which pivots should be considered nonzero. For that, it uses

the threshold value that you can control by calling setThreshold(const RealScalar&). References ColPivHouseholderQR¡ MatrixType ¿::rank().

const HCoeffsType& **hCoeffs** ( ) const
Returns a const reference to the vector of Householder coefficients used to represent the factor Q. For advanced uses only.

ColPivHouseholderQR¡ MatrixType ¿::HouseholderSequenceType **householderQ** (void) const
Returns the matrix Q as a sequence of householder transformations

ComputationInfo **info** ( ) const
Reports whether the QR factorization was succesful. Note: This function always returns Success. It is provided for compatibility with other factorization routines.

const internal::solve_retval¡ColPivHouseholderQR, typename MatrixType::IdentityReturnType¿ **inverse** ( ) const
Returns the inverse of the matrix of which *this is the QR decomposition. Note: If this matrix is not invertible, the returned matrix has undefined coefficients. Use isInvertible() to first determine whether this matrix is invertible.

bool **isInjective** ( ) const
Returns true if the matrix of which *this is the QR decomposition represents an injective linear map, i.e. has trivial kernel; false otherwise. Note: This method has to determine which pivots should be considered nonzero. For that, it uses the threshold value that you can control by calling setThreshold(const RealScalar&). References ColPivHouseholderQR¡ MatrixType ¿::rank().

bool **isInvertible** ( ) const
Returns true if the matrix of which *this is the QR decomposition is invertible. Note: This method has to determine which pivots should be considered nonzero. For that, it uses the threshold value that you can control by calling setThreshold(const RealScalar&). References ColPivHouseholderQR¡ MatrixType ¿::isInjective(), and ColPivHouseholderQR¡ MatrixType ¿::isSurjective().

bool **isSurjective** ( ) const
Returns true if the matrix of which *this is the QR decomposition represents a surjective linear map; false otherwise. Note: This method has to determine which pivots should be considered nonzero. For that, it uses the threshold value that you can control by calling setThreshold(const RealScalar&). References ColPivHouseholderQR¡ MatrixType ¿::rank().

MatrixType::RealScalar **logAbsDeterminant** ( ) const
Returns the natural log of the absolute value of the determinant of the matrix of which *this is the QR decomposition. It has only linear complexity (that is, O(n) where n is the dimension of the square matrix) as the QR decomposition has already been computed. Note: This is only for square matrices. This method is useful to work around the risk of overflow/underflow that's inherent to determinant computation. See Also absDeterminant(), MatrixBase::determinant()

const MatrixType& **matrixQR** ( ) const
Returns a reference to the matrix where the Householder QR decomposition is stored

const MatrixType& **matrixR** ( ) const
Returns a reference to the matrix where the result Householder QR is stored Warning: The strict lower part of this matrix contains internal values. Only the upper triangular part should be referenced. To get it, use matrixR().template triangularView¡Upper¿() For rank-deficient matrices, use matrixR().topLeftCorner(rank(), rank()).template triangularView¡Upper¿() RealScalar

maxPivot ( ) const
Returns the absolute value of the biggest pivot, i.e. the biggest diagonal coefficient of R.

Index **nonzeroPivots** ( ) const
Returns the number of nonzero pivots in the QR decomposition. Here nonzero is meant in the exact sense, not in a fuzzy sense. So that notion isn't really intrinsically interesting, but it is still useful when implementing algorithms. See Also rank() .

Index **rank** ( ) const
Returns the rank of the matrix of which *this is the QR decomposition. Note: This method has to determine which pivots should be considered nonzero. For that, it uses the threshold value that you can control by calling setThreshold(const RealScalar&). References ColPivHouseholderQR¡ MatrixType ¿::threshold().

ColPivHouseholderQR& **setThreshold** (const RealScalar & threshold)
Allows to prescribe a threshold to be used by certain methods, such as rank(), who need to determine when pivots are to be considered nonzero. This is not used for the QR decomposition itself. When it needs to get the threshold value, Eigen calls threshold(). By default, this uses a formula to automatically determine a reasonable threshold. Once you have called the present method setThreshold(const RealScalar&), your value is used instead. Parameters: threshold The new value to use as the threshold.
A pivot will be considered nonzero if its absolute value is strictly greater than where maxpivot is the biggest pivot. If you want to come back to the default behavior, call setThreshold(Default_t) References ColPivHouseholderQR¡ MatrixType ¿::threshold().

ColPivHouseholderQR& **setThreshold** (Default_t)
Allows to come back to the default behavior, letting Eigen use its default formula for determining the threshold. You should pass the special object Eigen::Default as parameter here. qr.setThreshold(Eigen::Default); See the documentation of setThreshold(const RealScalar&).

const internal::solve_retval¡ColPivHouseholderQR, Rhs¿ **solve** (const MatrixBase¡ Rhs ¿ & b) const
This method finds a solution x to the equation Ax=b, where A is the matrix of which *this is the QR decomposition, if any exists. Parameters: b the right-hand-side of the equation to solve.
Returns a solution. Note: The case where b is a matrix is not yet implemented. Also, this code is space inefficient.This method just tries to find as good a solution as possible. If you want to check whether a solution exists or if it is accurate, just call this function to get a result and then compute the error of this result, or use MatrixBase::isApprox() directly, for instance like this:

```
bool a\_solution\_exists = (A*result).isApprox(b, precision);
```

This method avoids dividing by zero, so that the non-existence of a solution doesn't by itself mean that you'll get inf or nan values. If there exists more than one solution, this method will arbitrarily choose one.
Example:

```
Matrix3f m = Matrix3f::Random();
Matrix3f y = Matrix3f::Random();
cout << "Here is the matrix m:" << endl << m << endl;
cout << "Here is the matrix y:" << endl << y << endl;Matrix3f x;
x = m.colPivHouseholderQr().solve(y);
assert(y.isApprox(m*x));
```

```
cout << "Here is a solution x to the equation mx=y:" << endl << x << endl;
```

```
Output:
Here is the matrix m:
0.68  0.597  -0.33
-0.211  0.823  0.536
0.566 -0.605 -0.444
Here is the matrix y:
0.108   -0.27   0.832
-0.0452  0.0268   0.271
0.258   0.904   0.435
Here is a solution x to the equation mx=y:
0.609   2.68   1.67
-0.231  -1.57 0.0713
0.51   3.51   1.05
```

RealScalar threshold ( ) const inline
Returns the threshold that will be used by certain methods such as rank(). See the documentation of setThreshold(const RealScalar&). Referenced by ColPivHouseholderQR¡ MatrixType ¿::rank(), and ColPivHouseholderQR¡ MatrixType ¿::setThreshold().

11.6.2 Linear Solver

Function **SolveColPivQR**(***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*) As mpNum[]

The function SolveColPivQR returns the solution x of $Ax = b$, based on a QR decomposition with column-pivoting.

Parameters:
A: A square real matrix.
B: A real vector or matrix.

Function **cplxSolveColPivQR**(***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*) As mpNum[]

The function cplxSolveColPivQR returns the solution x of $Ax = b$, based on a QR decomposition with column-pivoting.

Parameters:
A: A square complex matrix.
B: A complex vector or matrix.

11.6.3 Matrix Inversion

Function **InvertColPivQR**(***A*** *As mpNum[,]*) As mpNum[]

The function InvertColPivQR returns A^{-1}, the inverse of A, based on a QR decomposition with column-pivoting.

Parameter:
A: A square real matrix.

Function **cplxInvertColPivQR**(***A*** *As mpNum[,]*) As mpNum[]

The function cplxInvertColPivQR returns A^{-1}, the inverse of A, based on a QR decomposition with column-pivoting.

Parameter:
A: A square complex matrix.

11.6.4 Determinant

Function **DetColPivQR**(***A*** *As mpNum[,]*) As mpNum

The function DetColPivQR returns $|A|$, the determinant of A, based on a QR decomposition with column-pivoting.

Parameter:
A: A square real matrix.

Function **cplxDetColPivQR**(***A*** *As mpNum[,]*) As mpNum

The function cplxDetColPivQR returns $|A|$, the determinant of A, based on a QR decomposition with column-pivoting.

Parameter:
A: A square complex matrix.

11.6.5 Example

Example:

```
Example
```

11.7 QR Decomposition with full Pivoting

11.7.1 Decomposition

Function **DecompFullPivQR**(***A*** *As mpNum[,],* ***B*** *As mpNum[,],* ***Output*** *As String*) As mpNumList

The function DecompFullPivQR returns the QR decomposition with full pivoting of $AP = QR$.

Parameters:
A: the square real matrix of which we are computing the LU decomposition.
B: A vector or matrix of real numbers.
Output: A string specifying the output options.

Function **cplxDecompFullPivQR**(***A*** *As mpNum[,],* ***B*** *As mpNum[,],* ***Output*** *As String*) As mpNumList

The function cplxDecompFullPivQR returns the QR decomposition with full pivoting of $AP = QR$.

Parameters:
A: the square complex matrix of which we are computing the LU decomposition.
B: A vector or complex of real numbers.
Output: A string specifying the output options.

This class performs a rank-revealing QR decomposition of a matrix A into matrices P, Q and R such that

$$AP = QR \tag{11.7.1}$$

by using Householder transformations. Here, P is a permutation matrix, Q a unitary matrix and R an upper triangular matrix.
This decomposition performs a very prudent full pivoting in order to be rank-revealing and achieve optimal numerical stability. The trade-off is that it is slower than HouseholderQR and ColPivHouseholderQR.
Member Function DocumentationMatrixType::RealScalar **absDeterminant** ( ) const
Returns the absolute value of the determinant of the matrix of which *this is the QR decomposition. It has only linear complexity (that is, O(n) where n is the dimension of the square matrix) as the QR decomposition has already been computed. Note: This is only for square matrices. Warning: a determinant can be very big or small, so for matrices of large enough dimension, there is a risk of overflow/underflow. One way to work around that is to use logAbsDeterminant() instead. See Also logAbsDeterminant(), MatrixBase::determinant()

const PermutationType& **colsPermutation** ( ) const
Returns a const reference to the column permutation matrix

FullPivHouseholderQR¡ MatrixType ¿ & **compute** (const MatrixType & matrix)
Performs the QR factorization of the given matrix matrix. The result of the factorization is stored into *this, and a reference to *this is returned. See Also class FullPivHouseholderQR, FullPivHouseholderQR(const MatrixType&)

Index **dimensionOfKernel** ( ) const
Returns the dimension of the kernel of the matrix of which *this is the QR decomposition. Note: This method has to determine which pivots should be considered nonzero. For that, it uses

the threshold value that you can control by calling setThreshold(const RealScalar&). References FullPivHouseholderQR¡ MatrixType ¿::rank().

const HCoeffsType& **hCoeffs** ( ) const
Returns a const reference to the vector of Householder coefficients used to represent the factor Q. For advanced uses only.

const internal::solve_retval¡FullPivHouseholderQR, typename MatrixType::IdentityReturnType¿ **inverse** ( ) const
Returns the inverse of the matrix of which *this is the QR decomposition. Note: If this matrix is not invertible, the returned matrix has undefined coefficients. Use isInvertible() to first determine whether this matrix is invertible.

bool **isInjective** ( ) const
Returns true if the matrix of which *this is the QR decomposition represents an injective linear map, i.e. has trivial kernel; false otherwise. Note: This method has to determine which pivots should be considered nonzero. For that, it uses the threshold value that you can control by calling setThreshold(const RealScalar&). References FullPivHouseholderQR¡ MatrixType ¿::rank().

bool **isInvertible** ( ) const
Returns true if the matrix of which *this is the QR decomposition is invertible. Note: This method has to determine which pivots should be considered nonzero. For that, it uses the threshold value that you can control by calling setThreshold(const RealScalar&). References FullPivHouseholderQR¡ MatrixType ¿::isInjective(), and FullPivHouseholderQR¡ MatrixType ¿::isSurjective().

bool **isSurjective** ( ) const
Returns true if the matrix of which *this is the QR decomposition represents a surjective linear map; false otherwise. Note: This method has to determine which pivots should be considered nonzero. For that, it uses the threshold value that you can control by calling setThreshold(const RealScalar&). References FullPivHouseholderQR¡ MatrixType ¿::rank(). Referenced by FullPivHouseholderQR¡ MatrixType ¿::isInvertible().

MatrixType::RealScalar **logAbsDeterminant** ( ) const
Returns the natural log of the absolute value of the determinant of the matrix of which *this is the QR decomposition. It has only linear complexity (that is, O(n) where n is the dimension of the square matrix) as the QR decomposition has already been computed. Note: This is only for square matrices. This method is useful to work around the risk of overflow/underflow that's inherent to determinant computation.See Also absDeterminant(), MatrixBase::determinant()

FullPivHouseholderQR¡ MatrixType ¿::MatrixQReturnType **matrixQ** (void) const
Returns Expression object representing the matrix Q

const MatrixType& **matrixQR** ( ) const
Returns a reference to the matrix where the Householder QR decomposition is stored

RealScalar **maxPivot** ( ) const
Returns the absolute value of the biggest pivot, i.e. the biggest diagonal coefficient of U.

Index **nonzeroPivots** ( ) const
Returns the number of nonzero pivots in the QR decomposition. Here nonzero is meant in the exact sense, not in a fuzzy sense. So that notion isn't really intrinsically interesting, but it is still useful when implementing algorithms. See Also rank().

Index **rank** ( ) const
Returns the rank of the matrix of which *this is the QR decomposition. Note: This method has to determine which pivots should be considered nonzero. For that, it uses the threshold value that you can control by calling setThreshold(const RealScalar&). References FullPivHouseholderQR¡ MatrixType ¿::threshold().

const IntDiagSizeVectorType& **rowsTranspositions** ( ) const
Returns a const reference to the vector of indices representing the rows transpositions

FullPivHouseholderQR& **setThreshold** (const RealScalar & threshold)
Allows to prescribe a threshold to be used by certain methods, such as rank(), who need to determine when pivots are to be considered nonzero. This is not used for the QR decomposition itself. When it needs to get the threshold value, Eigen calls threshold(). By default, this uses a formula to automatically determine a reasonable threshold. Once you have called the present method setThreshold(const RealScalar&), your value is used instead. Parameters threshold The new value to use as the threshold.
A pivot will be considered nonzero if its absolute value is strictly greater than where maxpivot is the biggest pivot. If you want to come back to the default behavior, call setThreshold(Default_t) References FullPivHouseholderQR¡ MatrixType ¿::threshold().

FullPivHouseholderQR& **setThreshold** (Default_t)
Allows to come back to the default behavior, letting Eigen use its default formula for determining the threshold. You should pass the special object Eigen::Default as parameter here. qr.setThreshold(Eigen::Default); See the documentation of setThreshold(const RealScalar&).

const internal::solve_retval¡FullPivHouseholderQR, Rhs¿ **solve** (const MatrixBase¡ Rhs ¿ & b) const
This method finds a solution x to the equation Ax=b, where A is the matrix of which *this is the QR decomposition. Parameters: b the right-hand-side of the equation to solve.
Returns the exact or least-square solution if the rank is greater or equal to the number of columns of A, and an arbitrary solution otherwise. Note: The case where b is a matrix is not yet implemented. Also, this code is space inefficient.This method just tries to find as good a solution as possible. If you want to check whether a solution exists or if it is accurate, just call this function to get a result and then compute the error of this result, or use MatrixBase::isApprox() directly, for instance like this:

```
bool a\_solution\_exists = (A*result).isApprox(b, precision);
```

This method avoids dividing by zero, so that the non-existence of a solution doesn't by itself mean that you'll get inf or nan values. If there exists more than one solution, this method will arbitrarily choose one.
Example:

```
Matrix3f m = Matrix3f::Random();
Matrix3f y = Matrix3f::Random();
cout << "Here is the matrix m:" << endl << m << endl;
cout << "Here is the matrix y:" << endl << y << endl;
Matrix3f x;x = m.fullPivHouseholderQr().solve(y);
assert(y.isApprox(m*x));
cout << "Here is a solution x to the equation mx=y:" << endl << x << endl;
```

```
Output:
Here is the matrix m:
0.68  0.597  -0.33
-0.211  0.823  0.536
0.566 -0.605 -0.444
Here is the matrix y:
0.108   -0.27   0.832
-0.0452  0.0268   0.271
0.258   0.904   0.435
Here is a solution x to the equation mx=y:
0.609   2.68   1.67
-0.231  -1.57 0.0713
0.51   3.51   1.05
```

RealScalar **threshold** ( ) const
Returns the threshold that will be used by certain methods such as rank(). See the documentation of setThreshold(const RealScalar&). Referenced by FullPivHouseholderQR¡ MatrixType ¿::rank(), and FullPivHouseholderQR¡ MatrixType ¿::setThreshold().

11.7.2 Linear Solver

Function **SolveFullPivQR**(***A*** *As mpNum[,],* ***B*** *As mpNum[,]*) As mpNum[]

The function SolveFullPivQR returns the solution x of $Ax = b$, based on a QR decomposition with full pivoting.

Parameters:
A: A square real matrix.
B: A real vector or matrix.

Function **cplxSolveFullPivQR**(***A*** *As mpNum[,],* ***B*** *As mpNum[,]*) As mpNum[]

The function cplxSolveFullPivQR returns the solution x of $Ax = b$, based on a QR decomposition with full pivoting.

Parameters:
A: A square complex matrix.
B: A complex vector or matrix.

11.7.3 Matrix Inversion

Function **InvertFullPivQR**(***A*** *As mpNum[,]*) As mpNum[]

The function InvertFullPivQR returns A^{-1}, the inverse of A, based on a QR decomposition with full pivoting.

Parameter:
A: A square real matrix.

Function **cplxInvertFullPivQR**(***A*** *As mpNum[,]*) As mpNum[]

The function cplxInvertFullPivQR returns A^{-1}, the inverse of A, based on a QR decomposition with full pivoting.

Parameter:
A: A square complex matrix.

11.7.4 Determinant

Function **DetFullPivQR**(***A*** *As mpNum[,]*) As mpNum

The function DetFullPivQR returns $|A|$, the determinant of A, based on a QR decomposition with full pivoting.

Parameter:
A: A square real matrix.

Function **cplxDetFullPivQR**(***A*** *As mpNum[,]*) As mpNum

The function cplxDetFullPivQR returns $|A|$, the determinant of A, based on a QR decomposition with full pivoting.

Parameter:
A: A square complex matrix.

11.7.5 Example

Example:

```
Example
```

11.8 Singular Value Decomposition

Two-sided Jacobi SVD decomposition of a rectangular matrix.
Parameters
MatrixType the type of the matrix of which we are computing the SVD decomposition
QRPreconditioner this optional parameter allows to specify the type of QR decomposition that will be used internally for the R-SVD step for non-square matrices. See discussion of possible values below.
SVD decomposition consists in decomposing any n-by-p matrix A as a product

$$A = USV^* \tag{11.8.1}$$

where U is a n-by-n unitary, V is a p-by-p unitary, and S is a n-by-p real positive matrix which is zero outside of its main diagonal; the diagonal entries of S are known as the singular values of A and the columns of U and V are known as the left and right singular vectors of A respectively. Singular values are always sorted in decreasing order.
This JacobiSVD decomposition computes only the singular values by default. If you want U or V, you need to ask for them explicitly.
You can ask for only thin U or V to be computed, meaning the following. In case of a rectangular n-by-p matrix, letting m be the smaller value among n and p, there are only m singular vectors; the remaining columns of U and V do not correspond to actual singular vectors. Asking for thin U or V means asking for only their m first columns to be formed. So U is then a n-by-m matrix, and V is then a p-by-m matrix. Notice that thin U and V are all you need for (least squares) solving.
This JacobiSVD class is a two-sided Jacobi R-SVD decomposition, ensuring optimal reliability and accuracy. The downside is that it's slower than bidiagonalizing SVD algorithms for large square matrices; however its complexity is still where n is the smaller dimension and p is the greater dimension, meaning that it is still of the same order of complexity as the faster bidiagonalizing R-SVD algorithms. In particular, like any R-SVD, it takes advantage of non-squareness in that its complexity is only linear in the greater dimension.
If the input matrix has inf or nan coefficients, the result of the computation is undefined, but the computation is guaranteed to terminate in finite (and reasonable) time.
The possible values for QRPreconditioner are:

- ColPivHouseholderQRPreconditioner is the default. In practice it's very safe. It uses column-pivoting QR.
- FullPivHouseholderQRPreconditioner, is the safest and slowest. It uses full-pivoting QR. Contrary to other QRs, it doesn't allow computing thin unitaries.
- HouseholderQRPreconditioner is the fastest, and less safe and accurate than the pivoting variants. It uses non-pivoting QR. This is very similar in safety and accuracy to the bidiagonalization process used by bidiagonalizing SVD algorithms (since bidiagonalization is inherently non-pivoting). However the resulting SVD is still more reliable than bidiagonalizing SVDs because the Jacobi-based iterarive process is more reliable than the optimized bidiagonal SVD iterations.
- NoQRPreconditioner allows not to use a QR preconditioner at all. This is useful if you know that you will only be computing JacobiSVD decompositions of square matrices. Non-square matrices require a QR preconditioner. Using this option will result in faster compilation and smaller executable code. It won't significantly speed up computation, since JacobiSVD is always checking if QR preconditioning is needed before applying it anyway.

11.8.1 Decomposition

Function **DecompJacobiSVD**(***A*** *As mpNum[,],* ***B*** *As mpNum[,],* ***computationOptions*** *As Integer,* ***Output*** *As String*) As mpNumList

The function DecompJacobiSVD returns the Cholesky decomposition $A = LL^* = U^*U$ of a matrix.

Parameters:
A: the real matrix of which we are computing the LL^T Cholesky decomposition.
B: A vector or matrix of real numbers.
computationOptions: An optional parameter allowing to specify if you want full or thin U or V unitaries to be computed.
Output: A string specifying the output options.

Function **cplxDecompJacobiSVD**(***A*** *As mpNum[,],* ***B*** *As mpNum[,],* ***computationOptions*** *As Integer,* ***Output*** *As String*) As mpNumList

The function cplxDecompJacobiSVD returns the Cholesky decomposition $A = LL^* = U^*U$ of a matrix.

Parameters:
A: the complex matrix of which we are computing the LL^T Cholesky decomposition.
B: A vector or complex of real numbers.
computationOptions: An optional parameter allowing to specify if you want full or thin U or V unitaries to be computed.
Output: A string specifying the output options.

Member Function DocumentationJacobiSVD¡ MatrixType, QRPreconditioner ¿ & **compute** (const MatrixType & matrix, unsigned int computationOptions)
Method performing the decomposition of given matrix using custom options. Parameters: matrix the matrix to decompose
computationOptions: optional parameter allowing to specify if you want full or thin U or V unitaries to be computed. By default, none is computed. This is a bit-field, the possible bits are ComputeFullU, ComputeThinU, ComputeFullV, ComputeThinV.
Thin unitaries are only available if your matrix type has a Dynamic number of columns (for example MatrixXf). They also are not available with the (non-default) FullPivHouseholderQR preconditioner. References JacobiRotation¡ Scalar ¿::transpose().

JacobiSVD& **compute** (const MatrixType & matrix)
Method performing the decomposition of given matrix using current options. Parameters: matrix the matrix to decompose
This method uses the current computationOptions, as already passed to the constructor or to compute(const MatrixType&, unsigned int).

bool **computeU** ( ) const
Returns true if U (full or thin) is asked for in this SVD decomposition

bool **computeV** ( ) const
Returns true if V (full or thin) is asked for in this SVD decomposition

const MatrixUType& **matrixU** ( ) const

Returns the U matrix. For the SVD decomposition of a n-by-p matrix, letting m be the minimum of n and p, the U matrix is n-by-n if you asked for ComputeFullU, and is n-by-m if you asked for ComputeThinU.
The m first columns of U are the left singular vectors of the matrix being decomposed.
This method asserts that you asked for U to be computed.

const MatrixVType& **matrixV** ( ) const
Returns the V matrix. For the SVD decomposition of a n-by-p matrix, letting m be the minimum of n and p, the V matrix is p-by-p if you asked for ComputeFullV, and is p-by-m if you asked for ComputeThinV. The m first columns of V are the right singular vectors of the matrix being decomposed. This method asserts that you asked for V to be computed.

Index **nonzeroSingularValues** ( ) const
Returns the number of singular values that are not exactly 0

const SingularValuesType& **singularValues** ( ) const
Returns the vector of singular values. For the SVD decomposition of a n-by-p matrix, letting m be the minimum of n and p, the returned vector has size m. Singular values are always sorted in decreasing order.

const internal::solve_retval¡JacobiSVD, Rhs¿ **solve** (const MatrixBase¡ Rhs ¿ & b) const
Returns a (least squares) solution of using the current SVD decomposition of A. Parameters: b the right-hand-side of the equation to solve.
Note: Solving requires both U and V to be computed. Thin U and V are enough, there is no need for full U or V. SVD solving is implicitly least-squares. Thus, this method serves both purposes of exact solving and least-squares solving. In other words, the returned solution is guaranteed to minimize the Euclidean norm $||Ax - b||$.

11.8.2 Linear Solver

Function **SolveJacobiSVD**(***A*** *As mpNum[,],* ***B*** *As mpNum[,]*) As mpNum[]

The function SolveJacobiSVD returns the solution x of $Ax = b$, based on a singular value decomposition.

Parameters:
A: A symmetric positive definite real matrix.
B: A real vector or matrix.

Function **cplxSolveJacobiSVD**(***A*** *As mpNum[,],* ***B*** *As mpNum[,]*) As mpNum[]

The function cplxSolveJacobiSVD returns the solution x of $Ax = b$, based on a singular value decomposition.

Parameters:
A: A symmetric positive definite complex matrix.
B: A complex vector or matrix.

11.8.3 Matrix Inversion

Function **InvertJacobiSVD**(***A*** *As mpNum[,]*) As mpNum[]

The function InvertJacobiSVD returns A^{-1}, the inverse of A, based on a singular value decomposition.

Parameter:
A: A square real matrix.

Function **cplxInvertJacobiSVD**(***A*** *As mpNum[,]*) As mpNum[]

The function cplxInvertJacobiSVD returns A^{-1}, the inverse of A, based on a singular value decomposition.

Parameter:
A: A square complex matrix.

11.8.4 Determinant

Function **DetJacobiSVD**(***A*** *As mpNum[,]*) As mpNum

The function DetJacobiSVD returns $|A|$, the determinant of A, based on a singular value decomposition.

Parameter:
A: A square real matrix.

Function **cplxDetJacobiSVD**(***A*** *As mpNum[,]*) As mpNum

---

The function cplxDetJacobiSVD returns $|A|$, the determinant of A, based on a singular value decomposition.

Parameter:
A: A square complex matrix.

11.8.5 Example

Example:
Here's an example demonstrating basic usage:

```
MatrixXf m = MatrixXf::Random(3,2);
cout << "Here is the matrix m:" << endl << m << endl;
JacobiSVD<MatrixXf> svd(m, ComputeThinU | ComputeThinV);
cout << "Its singular values are:" << endl << svd.singularValues() << endl;
cout << "Its left singular vectors are the columns of the thin U matrix:" << endl <<
    svd.matrixU() << endl;
cout << "Its right singular vectors are the columns of the thin V matrix:" << endl <<
    svd.matrixV() << endl;
Vector3f rhs(1, 0, 0);
cout << "Now consider this rhs vector:" << endl << rhs << endl;
cout << "A least-squares solution of m*x = rhs is:" << endl << svd.solve(rhs) << endl;
```

```
Output:
Here is the matrix m:
```

```
0.68  0.597
-0.211  0.823
0.566 -0.605
Its singular values are:
1.19
0.899
Its left singular vectors are the columns of the thin U matrix:
0.388   0.866
0.712 -0.0634
-0.586   0.496
Its right singular vectors are the columns of the thin V matrix:
-0.183  0.983
0.983  0.183
Now consider this rhs vector:
1
0
0
A least-squares solution of m*x = rhs is:
0.888
0.496
```

11.9 Householder Transformations

Reference
Detailed Description
This module provides Householder transformations.

HouseholderSequence
Convenience function for constructing a Householder sequence of Householder reflections acting on subspaces with decreasing size.
Returns A HouseholderSequence constructed from the specified arguments.

HouseholderSequence(OnTheRight)
Convenience function for constructing a Householder sequence.
Returns A HouseholderSequence constructed from the specified arguments.
This function differs from householderSequence() in that the template argument OnTheSide of the constructed HouseholderSequence is set to OnTheRight, instead of the default OnTheLeft.

11.9.1 Overview

This class represents a product sequence of Householder reflections where the first Householder reflection acts on the whole space, the second Householder reflection leaves the one-dimensional subspace spanned by the first unit vector invariant, the third Householder reflection leaves the two-dimensional subspace spanned by the first two unit vectors invariant, and so on up to the last reflection which leaves all but one dimensions invariant and acts only on the last dimension. Such sequences of Householder reflections are used in several algorithms to zero out certain parts of a matrix.

Indeed, the methods HessenbergDecomposition::matrixQ(), Tridiagonalization::matrixQ(), HouseholderQR::householderQ(), and ColPivHouseholderQR::householderQ() all return a HouseholderSequence.

More precisely, the Householder sequence represents an $n \times n$ matrix H of the form $H = \prod_{i=0}^{n-1} H_i$ where the i-th Householder reflection is $H_i = I - h_i v_i v_i^*$. The i-th Householder coefficient H_i is a scalar and the i-th Householder vector v_i is a vector of the form

$$v_i = [\underbrace{0, \ldots, 0}_{i-1 \text{ zeros}}, 1, \underbrace{*, \ldots, *}_{n-i \text{ arbitrary entries}}]. \tag{11.9.1}$$

The last $n - i$ entries of v_i are called the essential part of the Householder vector.
Typical usages are listed below, where H is a HouseholderSequence:

```
A.applyOnTheRight(H);           // A = A * H
A.applyOnTheLeft(H);            // A = H * A
A.applyOnTheRight(H.adjoint()); // A = A * H^*
A.applyOnTheLeft(H.adjoint()); // A = H^* * A
MatrixXd Q = H;                 // conversion to a dense matrix
```

In addition to the adjoint, you can also apply the inverse (=adjoint), the transpose, and the conjugate operators.

11.9.2 Constructor

Parameters:

[in] v Matrix containing the essential parts of the Householder vectors
[in] h Vector containing the Householder coefficients
Constructs the Householder sequence with coefficients given by h and vectors given by v. The i-th Householder coefficient h_i is given by h(i) and the essential part of the i-th Householder vector v_i is given by v(k,i) with k > i (the subdiagonal part of the i-th column). If v has fewer columns than rows, then the Householder sequence contains as many Householder reflections as there are columns.
Example:

```
Matrix3d v = Matrix3d::Random();
cout << "The matrix v is:" << endl;
cout << v << endl;
Vector3d v0(1, v(1,0), v(2,0));
cout << "The first Householder vector is: v\_0 = " << v0.transpose() << endl;
Vector3d v1(0, 1, v(2,1));
cout << "The second Householder vector is: v\_1 = " << v1.transpose() << endl;
Vector3d v2(0, 0, 1);
cout << "The third Householder vector is: v\_2 = " << v2.transpose() << endl;
Vector3d h = Vector3d::Random();
cout << "The Householder coefficients are: h = " << h.transpose() << endl;
Matrix3d H0 = Matrix3d::Identity() - h(0) * v0 * v0.adjoint();
cout << "The first Householder reflection is represented by H\_0 = " << endl;
cout << H0 << endl;Matrix3d H1 = Matrix3d::Identity() - h(1) * v1 * v1.adjoint();
cout << "The second Householder reflection is represented by H\_1 = " << endl;cout <<
    H1 << endl;
Matrix3d H2 = Matrix3d::Identity() - h(2) * v2 * v2.adjoint();
cout << "The third Householder reflection is represented by H\_2 = " << endl;
cout << H2 << endl;cout << "Their product is H\_0 H\_1 H\_2 = " << endl;
cout << H0 * H1 * H2 << endl;HouseholderSequence<Matrix3d, Vector3d> hhSeq(v, h);
Matrix3d hhSeqAsMatrix(hhSeq);
cout << "If we construct a HouseholderSequence from v and h" << endl;
cout << "and convert it to a matrix, we get:" << endl;cout << hhSeqAsMatrix << endl;
```

```
Output:
The matrix v is:
0.68  0.597  -0.33
-0.211  0.823  0.536
0.566 -0.605 -0.444
The first Householder vector is: v\_0 =      1 -0.211  0.566
The second Householder vector is: v\_1 =      0      1 -0.605
The third Householder vector is: v\_2 = 0 0 1
The Householder coefficients are: h =   0.108 -0.0452   0.258
The first Householder reflection is represented by H\_0 =
0.892  0.0228 -0.0611
0.0228   0.995  0.0129
-0.0611  0.0129   0.965
The second Householder reflection is represented by H\_1 =
1       0       0
0    1.05 -0.0273
0 -0.0273    1.02
```

```
The third Householder reflection is represented by H\_2 =
1     0     0
0     1     0
0     0 0.742
Their product is H\_0 H\_1 H\_2 =
0.892  0.0255 -0.0466
0.0228    1.04 -0.0105
-0.0611 -0.0129   0.728
If we construct a HouseholderSequence from v and h
and convert it to a matrix, we get:
0.892  0.0255 -0.0466
0.0228    1.04 -0.0105
-0.0611 -0.0129   0.728
```

11.9.3 Member Function Documentation

Index **cols** (void) const
Number of columns of transformation viewed as a matrix.
Returns Number of columns. This equals the dimension of the space that the transformation acts on.

const **HouseholderSequence**
Returns a reference to the derived object

const EssentialVectorType **essentialVector** (Index k) const
Essential part of a Householder vector.
Parameters: [in] k Index of Householder reflection
Returns: Vector containing non-trivial entries of k-th Householder vector
This function returns the essential part of the Householder vector v_i. This is a vector of length $n-i$ containing the last $n-i$ entries of the vector

$$v_i = [\underbrace{0, \ldots, 0}_{i-1 \text{ zeros}}, 1, \underbrace{*, \ldots, *}_{n-i \text{ arbitrary entries}}]. \tag{11.9.2}$$

The index i equals k + shift(), corresponding to the k-th column of the matrix v passed to the constructor.

Matrix_type_times_scalar_type
Computes the product of a Householder sequence with a matrix.
Parameters: [in] other Matrix being multiplied.
Returns Expression object representing the product. This function computes HM where H is the Householder sequence represented by *this and M is the matrix other.

Index **rows** (void) const
Number of rows of transformation viewed as a matrix.
Returns Number of rows
This equals the dimension of the space that the transformation acts on.

HouseholderSequence& **setLength** (Index length)
Sets the length of the Householder sequence.
Parameters: [in] length New value for the length.

By default, the length n of the Householder sequence $H = H_0 H_1 \dots H_{n-1}$ is set to the number of columns of the matrix v passed to the constructor, or the number of rows if that is smaller. After this function is called, the length equals length.

HouseholderSequence& **setShift** (Index shift)
Sets the shift of the Householder sequence.
Parameters: [in] shift New value for the shift.
By default, a HouseholderSequence object represents $H = H_0 H_1 \dots H_{n-1}$ and the i-th column of the matrix v passed to the constructor corresponds to the i-th Householder reflection. After this function is called, the object represents $H = H_{\text{shift}} H_{\text{shift}+1} \dots H_{n-1}$ and the i-th column of v corresponds to the (shift+i)-th Householder reflection.

HouseholderSequence& **setTrans** (bool trans)
Sets the transpose flag.
Parameters: [in] trans New value of the transpose flag.
By default, the transpose flag is not set. If the transpose flag is set, then this object represents $H^T = H_{n-1}^T \dots H_1^T H_0^T$ instead of $H = H_0 H_1 \dots H_{n-1}$.

Index **size** ( ) const
Returns the number of coefficients, which is rows()*cols().

Chapter 12

Eigensystems, (based on Eigen)

Book reference: Golub & Van Loan (1996)

12.1 Symmetric/Hermitian Eigensystems

A matrix A is selfadjoint if it equals its adjoint. For real matrices, this means that the matrix is symmetric: it equals its transpose. This class computes the eigenvalues and eigenvectors of a selfadjoint matrix. These are the scalars λ and vectors v such that $Av = \lambda v$. The eigenvalues of a selfadjoint matrix are always real. If D is a diagonal matrix with the eigenvalues on the diagonal, and V is a matrix with the eigenvectors as its columns, then $A = VDV^{-1}$ (for selfadjoint matrices, the matrix V is always invertible). This is called the eigendecomposition.
The algorithm exploits the fact that the matrix is selfadjoint, making it faster and more accurate than the general purpose eigenvalue algorithms implemented in EigenSolver and ComplexEigenSolver.
Only the lower triangular part of the input matrix is referenced.
Call the function compute() to compute the eigenvalues and eigenvectors of a given matrix. Alternatively, you can use the SelfAdjointEigenSolver(const MatrixType, int) constructor which computes the eigenvalues and eigenvectors at construction time. Once the eigenvalue and eigenvectors are computed, they can be retrieved with the eigenvalues() and eigenvectors() functions.
The documentation for SelfAdjointEigenSolver(const MatrixType, int) contains an example of the typical use of this class.
To solve the generalized eigenvalue problem $Av = \lambda Bv$ and the likes, see the class GeneralizedSelfAdjointEigenSolver.

12.1.1 Real Symmetric Matrices

Function **EigenSymm**(***A*** *As mpNum[,]*) As mpNum

The function EigenSymm returns the eigenvalues of a real symmetric matrix.

Parameter:
A: the real matrix of which we are computing the eigenvalues.

Function **EigenSymmv**(***A*** *As mpNum[,]*) As mpNum

The function EigenSymmv returns the eigenvalues and eigenvectors of a real symmetric matrix.

Parameter:
A: the real matrix of which we are computing the eigenvalues.

Member Function DocumentationSelfAdjointEigenSolver¡ MatrixType ¿ & **compute** (const MatrixType & matrix, int options = ComputeEigenvectors)
Computes eigendecomposition of given matrix. Parameters: [in] matrix Selfadjoint matrix whose eigendecomposition is to be computed. Only the lower triangular part of the matrix is referenced. [in] options Can be ComputeEigenvectors (default) or EigenvaluesOnly.
Returns Reference to *this.
This function computes the eigenvalues of matrix. The eigenvalues() function can be used to retrieve them. If options equals ComputeEigenvectors, then the eigenvectors are also computed and can be retrieved by calling eigenvectors().
This implementation uses a symmetric QR algorithm. The matrix is first reduced to tridiagonal form using the Tridiagonalization class. The tridiagonal matrix is then brought to diagonal form with implicit symmetric QR steps with Wilkinson shift. Details can be found in Section 8.3 of Golub & Van Loan (1996).
The cost of the computation is about $9n^3$ if the eigenvectors are required and $4n^3/3$ if they are not required.
This method reuses the memory in the SelfAdjointEigenSolver object that was allocated when the object was constructed, if the size of the matrix does not change.
Example:

```
SelfAdjointEigenSolver<MatrixXf> es(4);
MatrixXf X = MatrixXf::Random(4,4);
MatrixXf A = X + X.transpose();
es.compute(A);
cout << "The eigenvalues of A are: " << es.eigenvalues().transpose() << endl;
es.compute(A + MatrixXf::Identity(4,4)); // re-use es to compute eigenvalues of A+I
cout << "The eigenvalues of A+I are: " << es.eigenvalues().transpose() << endl;
```

```
Output:
The eigenvalues of A are:  -1.58 -0.473   1.32   2.46
The eigenvalues of A+I are: -0.581  0.527   2.32   3.46
```

See Also SelfAdjointEigenSolver(const MatrixType&, int) References Eigen::ComputeEigenvectors, Eigen::NoConvergence, and Eigen::Success. Referenced by SelfAdjointEigenSolver¡ _MatrixType ¿::SelfAdjointEigenSolver().

SelfAdjointEigenSolver¡ MatrixType ¿ & **computeDirect** (const MatrixType & matrix, int options = ComputeEigenvectors)
Computes eigendecomposition of given matrix using a direct algorithm. This is a variant of compute(const MatrixType&, int options) which directly solves the underlying polynomial equation. Currently only 3x3 matrices for which the sizes are known at compile time are supported (e.g., Matrix3d). This method is usually significantly faster than the QR algorithm but it might also be less accurate. It is also worth noting that for 3x3 matrices it involves trigonometric operations which are not necessarily available for all scalar types. See Also compute(const MatrixType&, int options)

const RealVectorType& **eigenvalues** ( ) const
Returns the eigenvalues of given matrix. Returns A const reference to the column vector containing the eigenvalues. Precondition: The eigenvalues have been computed before. The eigenvalues

are repeated according to their algebraic multiplicity, so there are as many eigenvalues as rows in the matrix. The eigenvalues are sorted in increasing order.
Example:

```
MatrixXd ones = MatrixXd::Ones(3,3);
SelfAdjointEigenSolver<MatrixXd> es(ones);
cout << "The eigenvalues of the 3x3 matrix of ones are:"
<< endl << es.eigenvalues() << endl;
```

```
Output:
The eigenvalues of the 3x3 matrix of ones are:
-3.09e-16
0
3
```

See Also eigenvectors(), MatrixBase::eigenvalues()

const MatrixType& **eigenvectors** ( ) const
Returns the eigenvectors of given matrix. Returns A const reference to the matrix whose columns are the eigenvectors. Precondition: The eigenvectors have been computed before.
Column k of the returned matrix is an eigenvector corresponding to eigenvalue number k as returned by eigenvalues(). The eigenvectors are normalized to have (Euclidean) norm equal to one. If this object was used to solve the eigenproblem for the selfadjoint matrix A, then the matrix returned by this function is the matrix V in the eigendecomposition $A = VDV^{-1}$.
Example:

```
MatrixXd ones = MatrixXd::Ones(3,3);
SelfAdjointEigenSolver<MatrixXd> es(ones);
cout << "The first eigenvector of the 3x3 matrix of ones is:"
<< endl << es.eigenvectors().col(1) << endl;
```

```
Output:
The first eigenvector of the 3x3 matrix of ones is:
0
-0.707
0.707
```

See Also eigenvalues()

ComputationInfo info ( ) const
Reports whether previous computation was successful. Returns: Success if computation was succesful, NoConvergence otherwise.

MatrixType **operatorInverseSqrt** ( ) const

Function **MatSymmInverseSqrt**(***A*** *As mpNum[,]*) As mpNum

---

The function MatSymmInverseSqrt returns the inverse matrix square root of a real symmetric matrix.

Parameter:

A: the real matrix of which we are computing the eigenvalues.
Computes the inverse square root of the matrix. Returns the inverse positive-definite square root of the matrix Precondition: The eigenvalues and eigenvectors of a positive-definite matrix have been computed before. This function uses the eigendecomposition $A = VDV^{-1}$ to compute the inverse square root as $VD^{-1/2}V^{-1}$. This is cheaper than first computing the square root with operatorSqrt() and then its inverse with MatrixBase::inverse().
Example:

```
MatrixXd X = MatrixXd::Random(4,4);
MatrixXd A = X * X.transpose();
cout << "Here is a random positive-definite matrix, A:" << endl << A << endl << endl;
SelfAdjointEigenSolver<MatrixXd> es(A);
cout << "The inverse square root of A is: " << endl;
cout << es.operatorInverseSqrt() << endl;
cout << "We can also compute it with operatorSqrt() and inverse(). That yields: " <<
    endl;
cout << es.operatorSqrt().inverse() << endl;
```

```
Output:
Here is a random positive-definite matrix, A:
1.41 -0.697 -0.111  0.508
-0.697  0.423 0.0991   -0.4
-0.111 0.0991   1.25  0.902
0.508   -0.4  0.902    1.4

The inverse square root of A is:
1.88   2.78 -0.546  0.605
2.78   8.61   -2.3   2.74
-0.546   -2.3   1.92  -1.36
0.605   2.74  -1.36   2.18
We can also compute it with operatorSqrt() and inverse(). That yields:
1.88   2.78 -0.546  0.605
2.78   8.61   -2.3   2.74
-0.546   -2.3   1.92  -1.36
0.605   2.74  -1.36   2.18
```

See Also operatorSqrt(), MatrixBase::inverse(), MatrixFunctions Module

MatrixType **operatorSqrt** ( ) const

Function **MatSymmSqrt**(***A*** *As mpNum[,]*) As mpNum

The function MatSymmSqrt returns the matrix square root of a real symmetric matrix.

Parameter:
A: the real matrix of which we are computing the eigenvalues.
Computes the positive-definite square root of the matrix. Returns the positive-definite square root of the matrix Precondition: The eigenvalues and eigenvectors of a positive-definite matrix have been computed before. The square root of a positive-definite matrix A is the positive-definite matrix whose square equals A. This function uses the eigendecomposition $A = VDV^{-1}$ to compute the square root as $A^{1/2} = VD^{1/2}V^{-1}$.
Example:

```
MatrixXd X = MatrixXd::Random(4,4);
MatrixXd A = X * X.transpose();
cout << "Here is a random positive-definite matrix, A:" << endl << A << endl << endl;
SelfAdjointEigenSolver<MatrixXd> es(A);
MatrixXd sqrtA = es.operatorSqrt();
cout << "The square root of A is: " << endl << sqrtA << endl;
cout << "If we square this, we get: " << endl << sqrtA*sqrtA << endl;
```

```
Output:
Here is a random positive-definite matrix, A:
1.41 -0.697 -0.111  0.508
-0.697  0.423 0.0991   -0.4
-0.111 0.0991   1.25  0.902
0.508   -0.4  0.902    1.4

The square root of A is:
1.09  -0.432 -0.0685     0.2
-0.432   0.379   0.141  -0.269
-0.0685   0.141       1   0.468
0.2  -0.269   0.468    1.04
If we square this, we get:
1.41 -0.697 -0.111  0.508
-0.697  0.423 0.0991   -0.4
-0.111 0.0991   1.25  0.902
0.508   -0.4  0.902    1.4
```

See Also: operatorInverseSqrt(), MatrixFunctions Module

Member Data Documentationconst int **m_maxIterations**
Maximum number of iterations. The algorithm terminates if it does not converge within m_maxIterations * n iterations, where n denotes the size of the matrix. This value is currently set to 30 (copied from LAPACK).

12.1.2 Complex Hermitian Matrices

Function **cplxEigenHerm**(***A*** *As mpNum[,]*) As mpNum

The function cplxEigenHerm returns the eigenvalues of a complex hermitian matrix.

Parameter:
A: the complex hermitian matrix of which we are computing the eigenvalues.

Function **cplxEigenHermv**(***A*** *As mpNum[,]*) As mpNum

The function cplxEigenHermv returns the eigenvalues and eigenvectors of a complex hermitian matrix.

Parameter:
A: the complex hermitian matrix of which we are computing the eigenvalues.

12.1.2.1 Example

Example:

```
MatrixXd X = MatrixXd::Random(5,5);
MatrixXd A = X + X.transpose();
cout << "Here is a random symmetric 5x5 matrix, A:" << endl << A << endl << endl;
SelfAdjointEigenSolver<MatrixXd> es(A);
cout << "The eigenvalues of A are:" << endl << es.eigenvalues() << endl;
cout << "The matrix of eigenvectors, V, is:" << endl << es.eigenvectors() << endl <<
    endl;
double lambda = es.eigenvalues()[0];
cout << "Consider the first eigenvalue, lambda = " << lambda << endl;
VectorXd v = es.eigenvectors().col(0);
cout << "If v is the corresponding eigenvector, then lambda * v = " << endl << lambda
    * v << endl;
cout << "... and A * v = " << endl << A * v << endl << endl;
MatrixXd D = es.eigenvalues().asDiagonal();
MatrixXd V = es.eigenvectors();
cout << "Finally, V * D * V^(-1) = " << endl << V * D * V.inverse() << endl;
```

```
Output:
Here is a random symmetric 5x5 matrix, A:
1.36 -0.816  0.521   1.43 -0.144
-0.816 -0.659  0.794 -0.173 -0.406
0.521  0.794 -0.541  0.461  0.179
1.43 -0.173  0.461  -1.43  0.822
-0.144 -0.406  0.179  0.822  -1.37

The eigenvalues of A are:
-2.65
-1.77
-0.745
0.227
2.29
The matrix of eigenvectors, V, is:
0.326 -0.0984  -0.347  0.0109   0.874
0.207  -0.642  -0.228  -0.662  -0.232
-0.0495   0.629   0.164   -0.74   0.164
-0.721  -0.397   0.402  -0.115   0.385
0.573  -0.156   0.799  0.0256  0.0858

Consider the first eigenvalue, lambda = -2.65
If v is the corresponding eigenvector, then lambda * v =
-0.865
-0.55
0.131
1.91
-1.52
... and A * v =
```

```
-0.865
-0.55
0.131
1.91
-1.52

Finally, V * D * V^(-1) =
1.36 -0.816  0.521   1.43 -0.144
-0.816 -0.659  0.794 -0.173 -0.406
0.521  0.794 -0.541  0.461  0.179
1.43 -0.173  0.461  -1.43  0.822
-0.144 -0.406  0.179  0.822  -1.37
```

12.2 General (Nonsymmetric) Eigensystems

Computes eigenvalues and eigenvectors of general matrices.
This is defined in the Eigenvalues module.
MatrixType the type of the matrix of which we are computing the eigendecomposition; this is expected to be an instantiation of the Matrix class template. Currently, only real matrices are supported.
The eigenvalues and eigenvectors of a matrix A are scalars λ and vectors v such that $Av = \lambda v$. If D is a diagonal matrix with the eigenvalues on the diagonal, and V is a matrix with the eigenvectors as its columns, then $AV = VD$. The matrix V is almost always invertible, in which case we have $A = VDV^{-1}$. This is called the eigendecomposition. The eigenvalues and eigenvectors of a matrix may be complex, even when the matrix is real. However, we can choose real matrices V and D satisfying $AV = VD$, just like the eigendecomposition, if the matrix D is not required to be diagonal, but if it is allowed to have blocks of the form

$$\begin{pmatrix} u & v \\ -v & u \end{pmatrix} \tag{12.2.1}$$

(where u and v are real numbers) on the diagonal. These blocks correspond to complex eigenvalue pairs $u \pm iv$. We call this variant of the eigendecomposition the pseudo-eigendecomposition.
Call the function compute() to compute the eigenvalues and eigenvectors of a given matrix. Alternatively, you can use the EigenSolver(const MatrixType, bool) constructor which computes the eigenvalues and eigenvectors at construction time. Once the eigenvalue and eigenvectors are computed, they can be retrieved with the eigenvalues() and eigenvectors() functions. The pseudoEigenvalueMatrix() and pseudoEigenvectors() methods allow the construction of the pseudo-eigendecomposition.
The documentation for EigenSolver(const MatrixType, bool) contains an example of the typical use of this class.
See Also
MatrixBase::eigenvalues(), class ComplexEigenSolver, class SelfAdjointEigenSolver

12.2.1 Real Nonsymmetric Matrices

Function **EigenNonsymm**(***A*** *As mpNum[,]*) As mpNum[]

The function EigenNonsymm returns the eigenvalues of a real general (non-symmetric) matrix.

Parameter:
A: the real general (non-symmetric) matrix of which we are computing the eigenvalues.

Function **EigenNonsymmv**(***A*** *As mpNum[,]*) As mpNumList[2]

The function EigenNonsymmv returns the eigenvalues and eigenvectors of a real general (non-symmetric) matrix.

Parameter:
A: the real general (non-symmetric) matrix of which we are computing the eigenvalues.

Function **PseudoEigenNonsymm**(***A*** *As mpNum[,]*) As mpNum[]

The function PseudoEigenNonsymm returns the pseudoeigenvalues of a real general (non-symmetric) matrix.

Parameter:
A: the real general (non-symmetric) matrix of which we are computing the pseudoeigenvalues.

Function **PseudoEigenNonsymmv**(***A*** *As mpNum[,]*) As mpNumList[2]

The function PseudoEigenNonsymmv returns the pseudoeigenvalues and pseudoeigenvectors of a real general (non-symmetric) matrix.

Parameter:
A: the real general (non-symmetric) matrix of which we are computing the pseudoeigenvalues and pseudoeigenvectors.

Member Function DocumentationEigenSolver¡ MatrixType ¿ & **compute** (const MatrixType & matrix, bool computeEigenvectors = true)
Computes eigendecomposition of given matrix. Parameters:
[in] matrix Square matrix whose eigendecomposition is to be computed.
[in] computeEigenvectors If true, both the eigenvectors and the eigenvalues are computed; if false, only the eigenvalues are computed.
Returns: Reference to *this
This function computes the eigenvalues of the real matrix matrix. The eigenvalues() function can be used to retrieve them. If computeEigenvectors is true, then the eigenvectors are also computed and can be retrieved by calling eigenvectors().
The matrix is first reduced to real Schur form using the RealSchur class. The Schur decomposition is then used to compute the eigenvalues and eigenvectors. The cost of the computation is dominated by the cost of the Schur decomposition, which is very approximately $25n^3$ (where n is the size of the matrix) if computeEigenvectors is true, and $10n^3$ if computeEigenvectors is false.
This method reuses of the allocated data in the EigenSolver object.
Example:

```
EigenSolver<MatrixXf> es;
MatrixXf A = MatrixXf::Random(4,4);
es.compute(A, /* computeEigenvectors = */ false);
cout << "The eigenvalues of A are: " << es.eigenvalues().transpose() << endl;
es.compute(A + MatrixXf::Identity(4,4), false); // re-use es to compute eigenvalues
    of A+I
cout << "The eigenvalues of A+I are: " << es.eigenvalues().transpose() << endl;
```

```
Output:
The eigenvalues of A are:    (0.755,0.528)   (0.755,-0.528)  (-0.323,0.0965) (-0.323,-0.0
The eigenvalues of A+I are:    (1.75,0.528)   (1.75,-0.528)  (0.677,0.0965) (0.677,-0.096
```

const EigenvalueType& **eigenvalues** ( ) const
Returns the eigenvalues of given matrix. Returns: A const reference to the column vector containing the eigenvalues. Precondition: Either the constructor EigenSolver(const MatrixType&,bool) or the member function compute(const MatrixType&, bool) has been called before. The eigenvalues are repeated according to their algebraic multiplicity, so there are as many eigenvalues as rows in the matrix. The eigenvalues are not sorted in any particular order.

Example:

```
MatrixXd ones = MatrixXd::Ones(3,3);
EigenSolver<MatrixXd> es(ones, false);
cout << "The eigenvalues of the 3x3 matrix of ones are:" << endl << es.eigenvalues()
    << endl;
```

```
Output:
The eigenvalues of the 3x3 matrix of ones are:
(-5.31e-17,0)
(3,0)
(0,0)
```

EigenSolver¡ MatrixType ¿::EigenvectorsType **eigenvectors** ( ) const
Returns the eigenvectors of given matrix. Returns Matrix whose columns are the (possibly complex) eigenvectors. Precondition: Either the constructor EigenSolver(const MatrixType&,bool) or the member function compute(const MatrixType&, bool) has been called before, and computeEigenvectors was set to true (the default). Column k of the returned matrix is an eigenvector corresponding to eigenvalue number k as returned by eigenvalues(). The eigenvectors are normalized to have (Euclidean) norm equal to one. The matrix returned by this function is the matrix V in the eigendecomposition $A = VDV^{-1}$, if it exists.
Example:

```
MatrixXd ones = MatrixXd::Ones(3,3);
EigenSolver<MatrixXd> es(ones);
cout << "The first eigenvector of the 3x3 matrix of ones is:"
<< endl << es.eigenvectors().col(1) << endl;
```

```
Output:
The first eigenvector of the 3x3 matrix of ones is:
(0.577,0)
(0.577,0)
(0.577,0)
```

MatrixType **pseudoEigenvalueMatrix** ( ) const
Returns the block-diagonal matrix in the pseudo-eigendecomposition. Returns A block-diagonal matrix.
Precondition: Either the constructor EigenSolver(const MatrixType&,bool) or the member function compute(const MatrixType&, bool) has been called before. The matrix D returned by this function is real and block-diagonal. The blocks on the diagonal are either 1-by-1 or 2-by-2 blocks of the form $\begin{pmatrix} u & v \\ -v & u \end{pmatrix}$. These blocks are not sorted in any particular order. The matrix D and the matrix V returned by pseudoEigenvectors() satisfy $AV = VD$.
See Also pseudoEigenvectors() for an example, eigenvalues()

const MatrixType& **pseudoEigenvectors** ( ) const
Returns the pseudo-eigenvectors of given matrix. Returns Const reference to matrix whose columns are the pseudo-eigenvectors.

Precondition: Either the constructor EigenSolver(const MatrixType&,bool) or the member function compute(const MatrixType&, bool) has been called before, and computeEigenvectors was set to true (the default). The real matrix V returned by this function and the block-diagonal matrix D returned by pseudoEigenvalueMatrix() satisfy $AV = VD$.
Example:

```
MatrixXd A = MatrixXd::Random(6,6);
cout << "Here is a random 6x6 matrix, A:" << endl << A << endl << endl;
EigenSolver<MatrixXd> es(A);
MatrixXd D = es.pseudoEigenvalueMatrix();
MatrixXd V = es.pseudoEigenvectors();
cout << "The pseudo-eigenvalue matrix D is:" << endl << D << endl;
cout << "The pseudo-eigenvector matrix V is:" << endl << V << endl;
cout << "Finally, V * D * V^(-1) = " << endl << V * D * V.inverse() << endl;
```

```
Output:
Here is a random 6x6 matrix, A:
0.68   -0.33   -0.27  -0.717  -0.687  0.0259
-0.211   0.536  0.0268   0.214  -0.198   0.678
0.566  -0.444   0.904  -0.967   -0.74   0.225
0.597   0.108   0.832  -0.514  -0.782  -0.408
0.823 -0.0452   0.271  -0.726   0.998   0.275
-0.605   0.258   0.435   0.608  -0.563  0.0486

The pseudo-eigenvalue matrix D is:
0.049   1.06      0      0      0      0
-1.06  0.049      0      0      0      0
0      0  0.967      0      0      0
0      0      0  0.353      0      0
0      0      0      0  0.618  0.129
0      0      0      0 -0.129  0.618
The pseudo-eigenvector matrix V is:
-0.571   -0.888   -0.066    -1.13     17.2    -3.54
0.263   -0.204   -0.869     0.21     9.73     10.7
-0.827   -0.352    0.209   0.0871    -9.75    -4.17
-1.15   0.0535  -0.0857   -0.971     9.36    -4.53
-0.485    0.258    0.436    0.337    -9.74    -2.21
0.206    0.353   -0.426 -0.00873   -0.942     2.98
Finally, V * D * V^(-1) =
0.68   -0.33   -0.27  -0.717  -0.687  0.0259
-0.211   0.536  0.0268   0.214  -0.198   0.678
0.566  -0.444   0.904  -0.967   -0.74   0.225
0.597   0.108   0.832  -0.514  -0.782  -0.408
0.823 -0.0452   0.271  -0.726   0.998   0.275
-0.605   0.258   0.435   0.608  -0.563  0.0486
```

12.2.1.1 Example

Example:

```
MatrixXd A = MatrixXd::Random(6,6);
cout << "Here is a random 6x6 matrix, A:" << endl << A << endl << endl;
EigenSolver<MatrixXd> es(A);
cout << "The eigenvalues of A are:" << endl << es.eigenvalues() << endl;
cout << "The matrix of eigenvectors, V, is:" << endl << es.eigenvectors() << endl <<
   endl;
complex<double> lambda = es.eigenvalues()[0];
cout << "Consider the first eigenvalue, lambda = " << lambda << endl;
VectorXcd v = es.eigenvectors().col(0);
cout << "If v is the corresponding eigenvector, then lambda * v = " << endl << lambda
   * v << endl;
cout << "... and A * v = " << endl << A.cast<complex<double> >() * v << endl << endl;
MatrixXcd D = es.eigenvalues().asDiagonal();MatrixXcd V = es.eigenvectors();
cout << "Finally, V * D * V^(-1) = " << endl << V * D * V.inverse() << endl;
```

```
Output:
Here is a random 6x6 matrix, A:
0.68   -0.33   -0.27  -0.717  -0.687  0.0259
-0.211   0.536  0.0268   0.214  -0.198   0.678
0.566  -0.444   0.904  -0.967   -0.74   0.225
0.597   0.108   0.832  -0.514  -0.782  -0.408
0.823 -0.0452   0.271  -0.726   0.998   0.275
-0.605   0.258   0.435   0.608  -0.563  0.0486

The eigenvalues of A are:
(0.049,1.06)
(0.049,-1.06)
(0.967,0)
(0.353,0)
(0.618,0.129)
(0.618,-0.129)
The matrix of eigenvectors, V, is:
(-0.292,-0.454)   (-0.292,0.454)      (-0.0607,0)       (-0.733,0)    (0.59,-0.122)     (
(0.134,-0.104)    (0.134,0.104)       (-0.799,0)        (0.136,0)    (0.335,0.368)   (0.3
(-0.422,-0.18)    (-0.422,0.18)        (0.192,0)       (0.0563,0)  (-0.335,-0.143)   (-0.
(-0.589,0.0274) (-0.589,-0.0274)      (-0.0788,0)       (-0.627,0)   (0.322,-0.156)    (0
(-0.248,0.132)  (-0.248,-0.132)        (0.401,0)        (0.218,0)  (-0.335,-0.076)   (-0.
(0.105,0.18)    (0.105,-0.18)       (-0.392,0)     (-0.00564,0)  (-0.0324,0.103) (-0.0324

Consider the first eigenvalue, lambda = (0.049,1.06)
If v is the corresponding eigenvector, then lambda * v =
(0.466,-0.331)
(0.117,0.137)
(0.17,-0.456)
(-0.0578,-0.622)
(-0.152,-0.256)
(-0.186,0.12)
... and A * v =
```

```
(0.466,-0.331)
(0.117,0.137)
(0.17,-0.456)
(-0.0578,-0.622)
(-0.152,-0.256)
(-0.186,0.12)

Finally, V * D * V^(-1) =
(0.68,1.9e-16)    (-0.33,4.82e-17)   (-0.27,-2.37e-16)    (-0.717,1.6e-16)   (-0.687,-2.2
(-0.211,2.22e-16)    (0.536,4.16e-17)  (0.0268,-2.98e-16)           (0.214,0)   (-0.198,6
(0.566,1.22e-15)   (-0.444,1.11e-16)   (0.904,-4.61e-16)  (-0.967,-3.61e-16)    (-0.74,7.
(0.597,1.6e-15)    (0.108,1.84e-16)    (0.832,-5.6e-16)  (-0.514,-4.44e-16)   (-0.782,1.2
(0.823,-8.33e-16) (-0.0452,-2.71e-16)    (0.271,5.53e-16)   (-0.726,7.77e-16)   (0.998,-2
(-0.605,1.03e-15)    (0.258,1.91e-16)    (0.435,-4.6e-16)   (0.608,-6.38e-16)   (-0.563,1
```

Computes eigenvalues and eigenvectors of general complex matrices.
This is defined in the Eigenvalues module.
MatrixType the type of the matrix of which we are computing the eigendecomposition; this is expected to be an instantiation of the Matrix class template.
The eigenvalues and eigenvectors of a matrix A are scalars λ and vectors v such that $Av = \lambda v$. If D is a diagonal matrix with the eigenvalues on the diagonal, and V is a matrix with the eigenvectors as its columns, then $AV = VD$. The matrix V is almost always invertible, in which case we have $A = VDV^{-1}$. This is called the eigendecomposition. The main function in this class is compute(), which computes the eigenvalues and eigenvectors of a given function. The documentation for that function contains an example showing the main features of the class.
See Also
class EigenSolver, class SelfAdjointEigenSolver

12.2.2 Complex Nonsymmetric Matrices

Function **cplxEigenNonsymm**(***A*** *As mpNum[,]*) As mpNum[]

The function cplxEigenNonsymm returns the eigenvalues of a complex general (non-symmetric) matrix.

Parameter:
A: the complex general (non-symmetric) matrix of which we are computing the eigenvalues.

Function **cplxEigenNonsymmv**(***A*** *As mpNum[,]*) As mpNumList[2]

The function cplxEigenNonsymmv returns the eigenvalues and eigenvectors of a complex general (non-symmetric) matrix.

Parameter:
A: the complex general (non-symmetric) matrix of which we are computing the eigenvalues.

Member Function DocumentationComplexEigenSolver¡ MatrixType ¿ & compute (const Matrix-Type & matrix, bool computeEigenvectors = true)

Computes eigendecomposition of given matrix. Parameters: [in] matrix Square matrix whose eigendecomposition is to be computed. [in] computeEigenvectors If true, both the eigenvectors and the eigenvalues are computed; if false, only the eigenvalues are computed.
Returns: Reference to *this
This function computes the eigenvalues of the complex matrix matrix. The eigenvalues() function can be used to retrieve them. If computeEigenvectors is true, then the eigenvectors are also computed and can be retrieved by calling eigenvectors(). The matrix is first reduced to Schur form using the ComplexSchur class. The Schur decomposition is then used to compute the eigenvalues and eigenvectors. The cost of the computation is dominated by the cost of the Schur decomposition, which is $O(n^3)$ where n is the size of the matrix.
Example:

```
MatrixXcf A = MatrixXcf::Random(4,4);
cout << "Here is a random 4x4 matrix, A:" << endl << A << endl << endl;
ComplexEigenSolver<MatrixXcf> ces;
ces.compute(A);
cout << "The eigenvalues of A are:" << endl << ces.eigenvalues() << endl;
cout << "The matrix of eigenvectors, V, is:" << endl << ces.eigenvectors() << endl <<
    endl;
complex<float> lambda = ces.eigenvalues()[0];
cout << "Consider the first eigenvalue, lambda = " << lambda << endl;
VectorXcf v = ces.eigenvectors().col(0);
cout << "If v is the corresponding eigenvector, then lambda * v = " << endl << lambda
    * v << endl;
cout << "... and A * v = " << endl << A * v << endl << endl;
cout << "Finally, V * D * V^(-1) = "
<< endl
<< ces.eigenvectors() * ces.eigenvalues().asDiagonal() * ces.eigenvectors().inverse()
<< endl;
```

```
Output:
Here is a random 4x4 matrix, A:
(-0.211,0.68)  (0.108,-0.444)   (0.435,0.271) (-0.198,-0.687)
(0.597,0.566) (0.258,-0.0452)  (0.214,-0.717)  (-0.782,-0.74)
(-0.605,0.823)  (0.0268,-0.27) (-0.514,-0.967)  (-0.563,0.998)
(0.536,-0.33)   (0.832,0.904)  (0.608,-0.726)  (0.678,0.0259)

The eigenvalues of A are:
(0.137,0.505)
(-0.758,1.22)
(1.52,-0.402)
(-0.691,-1.63)
The matrix of eigenvectors, V, is:
(-0.246,-0.106)     (0.418,0.263)   (0.0417,-0.296)    (-0.122,0.271)
(-0.205,-0.629)    (0.466,-0.457)    (0.244,-0.456)      (0.247,0.23)
(-0.432,-0.0359) (-0.0651,-0.0146)    (-0.191,0.334)   (0.859,-0.0877)
(-0.301,0.46)    (-0.41,-0.397)     (0.623,0.328)    (-0.116,0.195)

Consider the first eigenvalue, lambda = (0.137,0.505)
If v is the corresponding eigenvector, then lambda * v =
```

```
(0.0197,-0.139)
(0.29,-0.19)
(-0.0412,-0.223)
(-0.274,-0.0891)
... and A * v =
(0.0197,-0.139)
(0.29,-0.19)
(-0.0412,-0.223)
(-0.274,-0.0891)

Finally, V * D * V^(-1) =
(-0.211,0.68)  (0.108,-0.444)   (0.435,0.271) (-0.198,-0.687)
(0.597,0.566) (0.258,-0.0452)  (0.214,-0.717)  (-0.782,-0.74)
(-0.605,0.823)  (0.0268,-0.27) (-0.514,-0.967)  (-0.563,0.998)
(0.536,-0.33)   (0.832,0.904)  (0.608,-0.726)  (0.678,0.0259)
```

References Eigen::Success. Referenced by ComplexEigenSolver¡ _MatrixType ¿::ComplexEigenSolver().

const EigenvalueType& **eigenvalues** ( ) const
Returns the eigenvalues of given matrix. Returns A const reference to the column vector containing the eigenvalues.
Precondition: Either the constructor ComplexEigenSolver(const MatrixType& matrix, bool) or the member function compute(const MatrixType& matrix, bool) has been called before to compute the eigendecomposition of a matrix. This function returns a column vector containing the eigenvalues. Eigenvalues are repeated according to their algebraic multiplicity, so there are as many eigenvalues as rows in the matrix. The eigenvalues are not sorted in any particular order.
Example:

```
MatrixXcf ones = MatrixXcf::Ones(3,3);
ComplexEigenSolver<MatrixXcf> ces(ones, /* computeEigenvectors = */ false);
cout << "The eigenvalues of the 3x3 matrix of ones are:"
<< endl << ces.eigenvalues() << endl;
```

```
Output:
The eigenvalues of the 3x3 matrix of ones are:
(0,-0)
(0,0)
(3,0)
```

const EigenvectorType& **eigenvectors** ( ) const
Returns the eigenvectors of given matrix. Returns A const reference to the matrix whose columns are the eigenvectors.
Precondition: Either the constructor ComplexEigenSolver(const MatrixType& matrix, bool) or the member function compute(const MatrixType& matrix, bool) has been called before to compute the eigendecomposition of a matrix, and computeEigenvectors was set to true (the default). This function returns a matrix whose columns are the eigenvectors. Column k is an eigenvector corresponding to eigenvalue number k as returned by eigenvalues(). The eigenvectors are normalized to have (Euclidean) norm equal to one. The matrix returned by this function is the matrix V in the eigendecomposition $A = VDV^{-1}$, if it exists.

Example:

```
MatrixXcf ones = MatrixXcf::Ones(3,3);
ComplexEigenSolver<MatrixXcf> ces(ones);
cout << "The first eigenvector of the 3x3 matrix of ones is:"
<< endl << ces.eigenvectors().col(1) << endl;
```

```
Output:
The first eigenvector of the 3x3 matrix of ones is:
(0.154,0)
(-0.772,0)
(0.617,0)
```

ComputationInfo **info** ( ) const
Reports whether previous computation was successful. Returns : Success if computation was succesful, NoConvergence otherwise. References ComplexSchur¡ _MatrixType ¿::info().

12.3 Generalized Eigensystems

Computes eigenvalues and eigenvectors of the generalized selfadjoint eigen problem. This is defined in the Eigenvalues module.
MatrixType the type of the matrix of which we are computing the eigendecomposition; this is expected to be an instantiation of the Matrix class template.
This class solves the generalized eigenvalue problem $Av = \lambda Bv$. In this case, the matrix A should be selfadjoint and the matrix B should be positive definite.
Only the lower triangular part of the input matrix is referenced.
Call the function compute() to compute the eigenvalues and eigenvectors of a given matrix. Alternatively, you can use the GeneralizedSelfAdjointEigenSolver(const MatrixType, const MatrixType, int) constructor which computes the eigenvalues and eigenvectors at construction time. Once the eigenvalue and eigenvectors are computed, they can be retrieved with the eigenvalues() and eigenvectors() functions.

GeneralizedSelfAdjointEigenSolver (const MatrixType & matA, const MatrixType & matB, int options = ComputeEigenvectors—Ax_lBx)
Constructor; computes generalized eigendecomposition of given matrix pencil. Parameters:
[in] matA Selfadjoint matrix in matrix pencil. Only the lower triangular part of the matrix is referenced.
[in] matB Positive-definite matrix in matrix pencil. Only the lower triangular part of the matrix is referenced.
[in] options A or-ed set of flags ComputeEigenvectors,EigenvaluesOnly — Ax_lBx,ABx_lx,BAx_lx. Default is ComputeEigenvectors—Ax_lBx.
This constructor calls compute(const MatrixType&, const MatrixType&, int) to compute the eigenvalues and (if requested) the eigenvectors of the generalized eigenproblem $Ax = \lambda Bx$ with matA the selfadjoint matrix A and matB the positive definite matrix B. Each eigenvector x satisfies the property $x^8Bx = 1$. The eigenvectors are computed if options contains ComputeEigenvectors. In addition, the two following variants can be solved via options:
âĂćABx_lx: $ABx = \lambda x$
âĂćBAx_lx: $BAx = \lambda x$.
Example:

```
MatrixXd X = MatrixXd::Random(5,5);
MatrixXd A = X + X.transpose();
cout << "Here is a random symmetric matrix, A:" << endl << A << endl;
X = MatrixXd::Random(5,5);
MatrixXd B = X * X.transpose();
cout << "and a random postive-definite matrix, B:" << endl << B << endl << endl;
GeneralizedSelfAdjointEigenSolver<MatrixXd> es(A,B);
cout << "The eigenvalues of the pencil (A,B) are:" << endl << es.eigenvalues() <<
    endl;
cout << "The matrix of eigenvectors, V, is:" << endl << es.eigenvectors() << endl <<
    endl;
double lambda = es.eigenvalues()[0];
cout << "Consider the first eigenvalue, lambda = " << lambda << endl;
VectorXd v = es.eigenvectors().col(0);
cout << "If v is the corresponding eigenvector, then A * v = "
<< endl << A * v << endl;
cout << "... and lambda * B * v = " << endl << lambda * B * v << endl << endl;
```

```
Output:
Here is a random symmetric matrix, A:
1.36 -0.816  0.521   1.43 -0.144
-0.816 -0.659  0.794 -0.173 -0.406
0.521  0.794 -0.541  0.461  0.179
1.43 -0.173  0.461  -1.43  0.822
-0.144 -0.406  0.179  0.822  -1.37
and a random postive-definite matrix, B:
0.132  0.0109 -0.0512  0.0674  -0.143
0.0109    1.68    1.13   -1.12   0.916
-0.0512    1.13     2.3   -2.14    1.86
0.0674   -1.12   -2.14    2.69   -2.01
-0.143   0.916    1.86   -2.01    1.68

The eigenvalues of the pencil (A,B) are:
-227
-3.9
-0.837
0.101
54.2
The matrix of eigenvectors, V, is:
-14.2    1.03 -0.0766  0.0273   -8.36
-0.0546   0.115  -0.729  -0.478   0.374
9.23  -0.624  0.0165  -0.499    3.01
-7.88    -1.3  -0.225  -0.109   -3.85
-20.8  -0.805   0.567  0.0828   -8.73

Consider the first eigenvalue, lambda = -227
If v is the corresponding eigenvector, then A * v =
-22.8
28.8
-19.8
-21.9
25.9
... and lambda * B * v =
-22.8
28.8
-19.8
-21.9
25.9
```

12.3.1 Real Generalized Symmetric-Definite Eigensystems

Function **EigenGensymm**(***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*) As mpNum

The function EigenGensymm returns the eigenvalues of a real Generalized Symmetric-Definite Eigensystem.

Parameters:

A: Selfadjoint matrix in matrix pencil. Only the lower triangular part of the matrix is referenced.
B: Positive-definite matrix in matrix pencil. Only the lower triangular part of the matrix is referenced.

Function **EigenGensymmv**(***A*** *As mpNum[,]*, ***B*** *As mpNum[,]*) As mpNum

---

The function EigenGensymmv returns the eigenvalues and eigenvectors of a real Generalized Symmetric-Definite Eigensystem.

Parameters:
A: Selfadjoint matrix in matrix pencil. Only the lower triangular part of the matrix is referenced.
B: Positive-definite matrix in matrix pencil. Only the lower triangular part of the matrix is referenced.

Member Function Documentation
GeneralizedSelfAdjointEigenSolver¡ MatrixType ¿ & compute (const MatrixType & matA, const MatrixType & matB, int options = ComputeEigenvectors—Ax_lBx)
Computes generalized eigendecomposition of given matrix pencil. Parameters [in] matA Self-adjoint matrix in matrix pencil. Only the lower triangular part of the matrix is referenced. [in] matB Positive-definite matrix in matrix pencil. Only the lower triangular part of the matrix is referenced. [in] options A or-ed set of flags ComputeEigenvectors,EigenvaluesOnly — Ax_lBx,ABx_lx,BAx_lx. Default is ComputeEigenvectors—Ax_lBx.
Returns Reference to *this According to options, this function computes eigenvalues and (if requested) the eigenvectors of one of the following three generalized eigenproblems:
âĂćAx_lBx: $Ax = \lambda Bx$
âĂćABx_lx: $ABx = \lambda x$
âĂćBAx_lx: $BAx = \lambda x$
with matA the selfadjoint matrix A and matB the positive definite matrix B. In addition, each eigenvector satisfies the property $x^*Bx = 1$.The eigenvalues() function can be used to retrieve the eigenvalues. If options contains ComputeEigenvectors, then the eigenvectors are also computed and can be retrieved by calling eigenvectors().
The implementation uses LLT to compute the Cholesky decomposition $B = LL^*$ and computes the classical eigendecomposition of the selfadjoint matrix $L^{-1}A(L^*)^{-1}$ if options contains Ax_lBx and of L^*AL otherwise. This solves the generalized eigenproblem, because any solution of the generalized eigenproblem $Ax = \lambda Bx$ corresponds to a solution $L^{-1}A(L^*)^{-1}(L^*x) = \lambda(L^*x)$ of the eigenproblem for $L^{-1}A(L^*)^{-1}$. Similar statements can be made for the two other variants.
Example:

```
MatrixXd X = MatrixXd::Random(5,5);
MatrixXd A = X * X.transpose();
X = MatrixXd::Random(5,5);
MatrixXd B = X * X.transpose();
GeneralizedSelfAdjointEigenSolver<MatrixXd> es(A,B,EigenvaluesOnly);
cout << "The eigenvalues of the pencil (A,B) are:" << endl << es.eigenvalues() <<
    endl;
es.compute(B,A,false);
cout << "The eigenvalues of the pencil (B,A) are:" << endl << es.eigenvalues() <<
    endl;
```

Output:

```
The eigenvalues of the pencil (A,B) are:
0.0289
0.299
2.11
8.64
2.08e+03
The eigenvalues of the pencil (B,A) are:
0.000481
0.116
0.473
3.34
34.6
```

SelfAdjointEigenSolver¡ MatrixType ¿ & compute (const MatrixType & matrix, int options = ComputeEigenvectors)
Computes eigendecomposition of given matrix. Parameters [in] matrix Selfadjoint matrix whose eigendecomposition is to be computed. Only the lower triangular part of the matrix is referenced. [in] options Can be ComputeEigenvectors (default) or EigenvaluesOnly.
Returns Reference to *this
This function computes the eigenvalues of matrix. The eigenvalues() function can be used to retrieve them. If options equals ComputeEigenvectors, then the eigenvectors are also computed and can be retrieved by calling eigenvectors().
This implementation uses a symmetric QR algorithm. The matrix is first reduced to tridiagonal form using the Tridiagonalization class. The tridiagonal matrix is then brought to diagonal form with implicit symmetric QR steps with Wilkinson shift. Details can be found in Section 8.3 of Golub & Van Loan (1996). The cost of the computation is about $9n^3$ if the eigenvectors are required and $4n^3/3$ if they are not required.
This method reuses the memory in the SelfAdjointEigenSolver object that was allocated when the object was constructed, if the size of the matrix does not change.
Example:

```
SelfAdjointEigenSolver<MatrixXf> es(4);
MatrixXf X = MatrixXf::Random(4,4);
MatrixXf A = X + X.transpose();es.compute(A);
cout << "The eigenvalues of A are: " << es.eigenvalues().transpose() << endl;
es.compute(A + MatrixXf::Identity(4,4)); // re-use es to compute eigenvalues of A+I
cout << "The eigenvalues of A+I are: " << es.eigenvalues().transpose() << endl;
```

```
Output:
The eigenvalues of A are:  -1.58 -0.473   1.32   2.46
The eigenvalues of A+I are: -0.581  0.527   2.32   3.46
```

SelfAdjointEigenSolver¡ MatrixType ¿ & **computeDirect** (const MatrixType & matrix, int options = ComputeEigenvectors)
Computes eigendecomposition of given matrix using a direct algorithm. This is a variant of compute(const MatrixType&, int options) which directly solves the underlying polynomial equation. Currently only 3x3 matrices for which the sizes are known at compile time are supported (e.g., Matrix3d). This method is usually significantly faster than the QR algorithm but it might also be less accurate. It is also worth noting that for 3x3 matrices it involves trigonometric operations

which are not necessarily available for all scalar types. See Also compute(const MatrixType&, int options)

const RealVectorType& **eigenvalues** ( ) const
Returns the eigenvalues of given matrix. Returns A const reference to the column vector containing the eigenvalues. Precondition The eigenvalues have been computed before. The eigenvalues are repeated according to their algebraic multiplicity, so there are as many eigenvalues as rows in the matrix. The eigenvalues are sorted in increasing order.
Example:

```
MatrixXd ones = MatrixXd::Ones(3,3);
SelfAdjointEigenSolver<MatrixXd> es(ones);
cout << "The eigenvalues of the 3x3 matrix of ones are:"
<< endl << es.eigenvalues() << endl;
```

```
Output:
The eigenvalues of the 3x3 matrix of ones are:
-3.09e-16
0
3
```

const MatrixType& eigenvectors ( ) const inlineinherited
Returns the eigenvectors of given matrix. Returns A const reference to the matrix whose columns are the eigenvectors. Precondition The eigenvectors have been computed before. Column k of the returned matrix is an eigenvector corresponding to eigenvalue number k as returned by eigenvalues(). The eigenvectors are normalized to have (Euclidean) norm equal to one. If this object was used to solve the eigenproblem for the selfadjoint matrix A, then the matrix returned by this function is the matrix V in the eigendecomposition $A = VDV^{-1}$.
Example:

```
MatrixXd ones = MatrixXd::Ones(3,3);
SelfAdjointEigenSolver<MatrixXd> es(ones);
cout << "The first eigenvector of the 3x3 matrix of ones is:"
<< endl << es.eigenvectors().col(1) << endl;
```

```
Output:
The first eigenvector of the 3x3 matrix of ones is:
0
-0.707
0.707
```

ComputationInfo **info** ( ) const
Reports whether previous computation was successful. Returns Success if computation was succesful, NoConvergence otherwise.

Member Data Documentationconst int m_maxIterations staticinherited
Maximum number of iterations. The algorithm terminates if it does not converge within m_maxIterations * n iterations, where n denotes the size of the matrix. This value is currently set to 30 (copied from LAPACK).

12.3.2 Complex Hermitian Generalized Symmetric-Definite Eigensystems

Function **cplxEigenGenherm**(***A*** *As mpNum[,],* ***B*** *As mpNum[,]*) As mpNum

The function cplxEigenGenherm returns the eigenvalues of a Complex Hermitian Generalized Symmetric-Definite Eigensystem.

Parameters:
A: Selfadjoint matrix in matrix pencil. Only the lower triangular part of the matrix is referenced.
B: Positive-definite matrix in matrix pencil. Only the lower triangular part of the matrix is referenced.

Function **cplxEigenGenhermv**(***A*** *As mpNum[,],* ***B*** *As mpNum[,]*) As mpNum

The function cplxEigenGenhermv returns the eigenvalues and eigenvectors of a Complex Hermitian Generalized Symmetric-Definite Eigensystem.

Parameters:
A: Selfadjoint matrix in matrix pencil. Only the lower triangular part of the matrix is referenced.
B: Positive-definite matrix in matrix pencil. Only the lower triangular part of the matrix is referenced.

GeneralizedSelfAdjointEigenSolver (const MatrixType & matA, const MatrixType & matB, int options = ComputeEigenvectors—Ax_lBx)
Constructor; computes generalized eigendecomposition of given matrix pencil. Parameters:
[in] matA Selfadjoint matrix in matrix pencil. Only the lower triangular part of the matrix is referenced.
[in] matB Positive-definite matrix in matrix pencil. Only the lower triangular part of the matrix is referenced.
[in] options A or-ed set of flags ComputeEigenvectors,EigenvaluesOnly — Ax_lBx,ABx_lx,BAx_lx. Default is ComputeEigenvectors—Ax_lBx.
This constructor calls compute(const MatrixType&, const MatrixType&, int) to compute the eigenvalues and (if requested) the eigenvectors of the generalized eigenproblem $Ax = \lambda Bx$ with matA the selfadjoint matrix A and matB the positive definite matrix B. Each eigenvector x satisfies the property $x^8Bx = 1$. The eigenvectors are computed if options contains ComputeEigenvectors. In addition, the two following variants can be solved via options:
âĂćABx_lx: $ABx = \lambda x$
âĂćBAx_lx: $BAx = \lambda x$.

12.3.3 Real Generalized Nonsymmetric Eigensystem

Function **EigenGenNonsymm**(***A*** *As mpNum[,],* ***B*** *As mpNum[,]*) As mpNum

The function EigenGenNonsymm returns the eigenvalues of a real Generalized Non-Symmetric Eigensystem.

Parameters:
A: Selfadjoint matrix in matrix pencil. Only the lower triangular part of the matrix is referenced.
B: Positive-definite matrix in matrix pencil. Only the lower triangular part of the matrix is referenced.

Function **EigenGenNonsymmv**(***A*** *As mpNum[,],* ***B*** *As mpNum[,]*) As mpNum

The function EigenGenNonsymmv returns the eigenvalues and eigenvectors of a real Generalized Non-Symmetric Eigensystem.

Parameters:
A: Selfadjoint matrix in matrix pencil. Only the lower triangular part of the matrix is referenced.
B: Positive-definite matrix in matrix pencil. Only the lower triangular part of the matrix is referenced.
Computes the generalized eigenvalues and eigenvectors of a pair of general (nonsymmetric) matrices.
This is defined in the Eigenvalues module.
MatrixType the type of the matrices of which we are computing the eigen-decomposition; this is expected to be an instantiation of the Matrix class template. Currently, only real matrices are supported.
The generalized eigenvalues and eigenvectors of a matrix pair A and B are scalars λ and vectors v such that $Av = \lambda Bv$. If D is a diagonal matrix with the eigenvalues on the diagonal, and V is a matrix with the eigenvectors as its columns, then $AV = BVD$. The matrix V is almost always invertible, in which case we have $A = BVDV^{-1}$. This is called the generalized eigen-decomposition.
The generalized eigenvalues and eigenvectors of a matrix pair may be complex, even when the matrices are real. Moreover, the generalized eigenvalue might be infinite if the matrix B is singular. To workaround this difficulty, the eigenvalues are provided as a pair of complex α and real β such that: $\lambda_i = \alpha_i/\beta_i$. If β_i is (nearly) zero, then one can consider the well defined left eigenvalue $\mu = \beta_i/\alpha_i$ such that: $\mu_i A v_i = B v_i$, or even $\mu_i u_i^T A = u_i^T B$ where u_i is called the left eigenvector.
Call the function compute() to compute the generalized eigenvalues and eigenvectors of a given matrix pair.
Alternatively, you can use the GeneralizedEigenSolver(const MatrixType, const MatrixType, bool) constructor which computes the eigenvalues and eigenvectors at construction time. Once the eigenvalue and eigenvectors are computed, they can be retrieved with the eigenvalues() and eigenvectors() functions.

12.3.3.1 Member Function Documentation

ComplexVectorType **alphas** ( ) const
Returns A const reference to the vectors containing the alpha values
This vector permits to reconstruct the j-th eigenvalues as alphas(i)/betas(j).

VectorType **betas** ( ) const
Returns A const reference to the vectors containing the beta values This vector permits to reconstruct the j-th eigenvalues as alphas(i)/betas(j).

GeneralizedEigenSolver¡ MatrixType ¿ & **compute** (const MatrixType & A, const MatrixType & B, bool computeEigenvectors = true)
Computes generalized eigendecomposition of given matrix. Parameters
[in] A Square matrix whose eigendecomposition is to be computed.
[in] B Square matrix whose eigendecomposition is to be computed.
[in] computeEigenvectors If true, both the eigenvectors and the eigenvalues are computed; if false, only the eigenvalues are computed.
Returns Reference to *this
This function computes the eigenvalues of the real matrix matrix. The eigenvalues() function can be used to retrieve them. If computeEigenvectors is true, then the eigenvectors are also computed and can be retrieved by calling eigenvectors().
The matrix is first reduced to real generalized Schur form using the RealQZ class. The generalized Schur decomposition is then used to compute the eigenvalues and eigenvectors.
The cost of the computation is dominated by the cost of the generalized Schur decomposition.
This method reuses of the allocated data in the GeneralizedEigenSolver object.

EigenvalueType **eigenvalues** ( ) const
Returns an expression of the computed generalized eigenvalues. Returns An expression of the column vector containing the eigenvalues.
It is a shortcut for this-¿alphas().cwiseQuotient(this-¿betas()); Not that betas might contain zeros. It is therefore not recommended to use this function, but rather directly deal with the alphas and betas vectors.
Precondition:
Either the constructor GeneralizedEigenSolver(const MatrixType&,const MatrixType&,bool) or the member function compute(const MatrixType&,const MatrixType&,bool) has been called before.
The eigenvalues are repeated according to their algebraic multiplicity, so there are as many eigenvalues as rows in the matrix. The eigenvalues are not sorted in any particular order.

GeneralizedEigenSolver& **setMaxIterations** (Index maxIters)
Sets the maximal number of iterations allowed.

12.3.3.2 Example

Here is an usage example of this class:
Example:

```
GeneralizedEigenSolver<MatrixXf> ges;
MatrixXf A = MatrixXf::Random(4,4);
MatrixXf B = MatrixXf::Random(4,4);
ges.compute(A, B);
cout << "The (complex) numerators of the generalized eigenvalues are: "
<< ges.alphas().transpose() << endl;
cout << "The (real) denominatore of the generalized eigenvalues are: "
<< ges.betas().transpose() << endl;
cout << "The (complex) generalized eigenvalues are (alphas./beta): "
<< ges.eigenvalues().transpose() << endl;
```

```
Output:
The (complex) numerators of the generalized eigenvalues are:
(0.644,0.795) (0.644,-0.795)     (-0.398,0)      (-1.12,0)

The (real) denominatore of the generalized eigenvalues are:
1.51  1.51 -1.25 0.746

The (complex) generalized eigenvalues are (alphas./beta):
(0.427,0.528) (0.427,-0.528)     (0.318,-0)       (-1.5,0)
```

12.4 Decompositions

12.4.1 Tridiagonalization

Tridiagonal decomposition of a selfadjoint matrix.
This is defined in the Eigenvalues module.
MatrixType the type of the matrix of which we are computing the tridiagonal decomposition; this is expected to be an instantiation of the Matrix class template.
This class performs a tridiagonal decomposition of a selfadjoint matrix A such that: $A = QTQ^*$ where Q is unitary and T a real symmetric tridiagonal matrix. A tridiagonal matrix is a matrix which has nonzero elements only on the main diagonal and the first diagonal below and above it. The Hessenberg decomposition of a selfadjoint matrix is in fact a tridiagonal decomposition. This class is used in SelfAdjointEigenSolver to compute the eigenvalues and eigenvectors of a selfadjoint matrix.
Call the function compute() to compute the tridiagonal decomposition of a given matrix. Alternatively, you can use the Tridiagonalization(const MatrixType) constructor which computes the tridiagonal Schur decomposition at construction time. Once the decomposition is computed, you can use the matrixQ() and matrixT() functions to retrieve the matrices Q and T in the decomposition.
The documentation of Tridiagonalization(const MatrixType) contains an example of the typical use of this class.
Example:

```
MatrixXd X = MatrixXd::Random(5,5);
MatrixXd A = X + X.transpose();
cout << "Here is a random symmetric 5x5 matrix:" << endl << A << endl << endl;
Tridiagonalization<MatrixXd> triOfA(A);
MatrixXd Q = triOfA.matrixQ();
cout << "The orthogonal matrix Q is:" << endl << Q << endl;
MatrixXd T = triOfA.matrixT();
cout << "The tridiagonal matrix T is:" << endl << T << endl << endl;
cout << "Q * T * Q^T = " << endl << Q * T * Q.transpose() << endl;
```

```
Output:
Here is a random symmetric 5x5 matrix:
1.36 -0.816  0.521   1.43 -0.144
-0.816 -0.659  0.794 -0.173 -0.406
0.521  0.794 -0.541  0.461  0.179
1.43 -0.173  0.461  -1.43  0.822
-0.144 -0.406  0.179  0.822  -1.37

The orthogonal matrix Q is:
1        0        0        0        0
0   -0.471    0.127   -0.671   -0.558
0    0.301   -0.195    0.437   -0.825
0    0.825   0.0459   -0.563 -0.00872
0  -0.0832   -0.971   -0.202   0.0922
The tridiagonal matrix T is:
1.36   1.73      0      0      0
```

```
1.73   -1.2 -0.966      0      0
0 -0.966  -1.28  0.214      0
0      0  0.214  -1.69  0.345
0      0      0  0.345  0.164

Q * T * Q^T =
1.36 -0.816  0.521   1.43 -0.144
-0.816 -0.659  0.794 -0.173 -0.406
0.521  0.794 -0.541  0.461  0.179
1.43 -0.173  0.461  -1.43  0.822
-0.144 -0.406  0.179  0.822  -1.37
```

12.4.1.1 Member Function Documentation

Member Function Documentation

Tridiagonalization& **compute** (const MatrixType & matrix)
Computes tridiagonal decomposition of given matrix. Parameters: [in] matrix Selfadjoint matrix whose tridiagonal decomposition is to be computed.
Returns Reference to *this
The tridiagonal decomposition is computed by bringing the columns of the matrix successively in the required form using Householder reflections. The cost is flops, where denotes the size of the given matrix. This method reuses of the allocated data in the Tridiagonalization object, if the size of the matrix does not change.
Example:

```
Tridiagonalization<MatrixXf> tri;
MatrixXf X = MatrixXf::Random(4,4);
MatrixXf A = X + X.transpose();
tri.compute(A);
cout << "The matrix T in the tridiagonal decomposition of A is: " << endl;
cout << tri.matrixT() << endl;
tri.compute(2*A); // re-use tri to compute eigenvalues of 2A
cout << "The matrix T in the tridiagonal decomposition of 2A is: " << endl;
cout << tri.matrixT() << endl;
```

```
Output:
The matrix T in the tridiagonal decomposition of A is:
1.36 -0.704      0      0
-0.704 0.0147   1.71      0
0   1.71  0.856  0.641
0      0  0.641 -0.506
The matrix T in the tridiagonal decomposition of 2A is:
2.72  -1.41      0      0
-1.41 0.0294   3.43      0
0   3.43   1.71   1.28
0      0   1.28  -1.01
```

Tridiagonalization¡ MatrixType ¿::DiagonalReturnType **diagonal** ( ) const

Returns the diagonal of the tridiagonal matrix T in the decomposition. Returns expression representing the diagonal of T Precondition Either the constructor Tridiagonalization(const MatrixType&) or the member function compute(const MatrixType&) has been called before to compute the tridiagonal decomposition of a matrix.
Example:

```
MatrixXcd X = MatrixXcd::Random(4,4);
MatrixXcd A = X + X.adjoint();
cout << "Here is a random self-adjoint 4x4 matrix:" << endl << A << endl << endl;
Tridiagonalization<MatrixXcd> triOfA(A);
MatrixXd T = triOfA.matrixT();
cout << "The tridiagonal matrix T is:" << endl << T << endl << endl;
cout << "We can also extract the diagonals of T directly ..." << endl;
VectorXd diag = triOfA.diagonal();
cout << "The diagonal is:" << endl << diag << endl;
VectorXd subdiag = triOfA.subDiagonal();
cout << "The subdiagonal is:" << endl << subdiag << endl;
```

```
Output:
Here is a random self-adjoint 4x4 matrix:
(-0.422,0)  (0.705,-1.01) (-0.17,-0.552) (0.338,-0.357)
(0.705,1.01)      (0.515,0) (0.241,-0.446)   (0.05,-1.64)
(-0.17,0.552)  (0.241,0.446)      (-1.03,0)  (0.0449,1.72)
(0.338,0.357)    (0.05,1.64) (0.0449,-1.72)       (1.36,0)

The tridiagonal matrix T is:
-0.422  -1.45      0      0
-1.45   1.01  -1.42      0
0  -1.42    1.8   -1.2
0      0   -1.2  -1.96

We can also extract the diagonals of T directly ...
The diagonal is:
-0.422
1.01
1.8
-1.96
The subdiagonal is:
-1.45
-1.42
-1.2
```

See Also matrixT(), subDiagonal()

CoeffVectorType **householderCoefficients** ( ) const
Returns the Householder coefficients. Returns a const reference to the vector of Householder coefficients
Precondition: Either the constructor Tridiagonalization(const MatrixType&) or the member function compute(const MatrixType&) has been called before to compute the tridiagonal decompo-

sition of a matrix. The Householder coefficients allow the reconstruction of the matrix Q in the tridiagonal decomposition from the packed data.
Example:

```
Matrix4d X = Matrix4d::Random(4,4);
Matrix4d A = X + X.transpose();
cout << "Here is a random symmetric 4x4 matrix:" << endl << A << endl;
Tridiagonalization<Matrix4d> triOfA(A);
Vector3d hc = triOfA.householderCoefficients();
cout << "The vector of Householder coefficients is:" << endl << hc << endl;
```

```
Output:
Here is a random symmetric 4x4 matrix:
1.36   0.612   0.122   0.326
0.612   -1.21  -0.222   0.563
0.122  -0.222 -0.0904    1.16
0.326   0.563    1.16    1.66
The vector of Householder coefficients is:
1.87
1.24
0
```

See Also packedMatrix(), Householder module

HouseholderSequenceType **matrixQ** ( ) const
Returns the unitary matrix Q in the decomposition.
Returns object representing the matrix Q
Precondition:
Either the constructor Tridiagonalization(const MatrixType&) or the member function compute(const MatrixType&) has been called before to compute the tridiagonal decomposition of a matrix.
This function returns a light-weight object of template class HouseholderSequence. You can either apply it directly to a matrix or you can convert it to a matrix of type MatrixType.
See Also Tridiagonalization(const MatrixType&) for an example, matrixT(), class HouseholderSequence

MatrixTReturnType **matrixT** ( ) const
Returns an expression of the tridiagonal matrix T in the decomposition.
Returns expression object representing the matrix T
Precondition:
Either the constructor Tridiagonalization(const MatrixType&) or the member function compute(const MatrixType&) has been called before to compute the tridiagonal decomposition of a matrix.
Currently, this function can be used to extract the matrix T from internal data and copy it to a dense matrix object. In most cases, it may be sufficient to directly use the packed matrix or the vector expressions returned by diagonal() and subDiagonal() instead of creating a new dense copy matrix with this function.

const MatrixType& **packedMatrix** ( ) const
Returns the internal representation of the decomposition.

Returns a const reference to a matrix with the internal representation of the decomposition.
Precondition:
Either the constructor Tridiagonalization(const MatrixType&) or the member function compute(const MatrixType&) has been called before to compute the tridiagonal decomposition of a matrix.
The returned matrix contains the following information:
âĂć the strict upper triangular part is equal to the input matrix A.
âĂć the diagonal and lower sub-diagonal represent the real tridiagonal symmetric matrix T.
âĂć the rest of the lower part contains the Householder vectors that, combined with Householder coefficients returned by householderCoefficients(), allows to reconstruct the matrix Q as $Q = H_{N-1} \dots H_1 H_0$. Here, the matrices H_i are the Householder transformations $H_i = (I - h_i v_i v_i^T)$ where h_i is the ith Householder coefficient and v_i is the Householder vector defined by $v_i = [0, \dots, 0, 1, M(i+2, i), \dots, M(N-1, i)]^T$ with M the matrix returned by this function.See LAPACK for further details on this packed storage.
Example:

```
Matrix4d X = Matrix4d::Random(4,4);
Matrix4d A = X + X.transpose();
cout << "Here is a random symmetric 4x4 matrix:" << endl << A << endl;
Tridiagonalization<Matrix4d> triOfA(A);
Matrix4d pm = triOfA.packedMatrix();
cout << "The packed matrix M is:" << endl << pm << endl;
cout << "The diagonal and subdiagonal corresponds to the matrix T, which is:"
<< endl << triOfA.matrixT() << endl;
```

```
Output:
Here is a random symmetric 4x4 matrix:
1.36   0.612   0.122   0.326
0.612   -1.21  -0.222   0.563
0.122  -0.222 -0.0904    1.16
0.326   0.563    1.16    1.66
The packed matrix M is:
1.36  0.612  0.122  0.326
-0.704 0.0147 -0.222  0.563
0.0925   1.71  0.856   1.16
0.248  0.785  0.641 -0.506
The diagonal and subdiagonal corresponds to the matrix T, which is:
1.36 -0.704      0      0
-0.704 0.0147   1.71      0
0   1.71  0.856  0.641
0      0  0.641 -0.506
```

See Also householderCoefficients()

Tridiagonalization¡ MatrixType ¿::SubDiagonalReturnType **subDiagonal** ( ) const
Returns the subdiagonal of the tridiagonal matrix T in the decomposition.
Returns expression representing the subdiagonal of T
Precondition: Either the constructor Tridiagonalization(const MatrixType&) or the member function compute(const MatrixType&) has been called before to compute the tridiagonal decomposition of a matrix.

12.4.2 Hessenberg Decomposition

Reduces a square matrix to Hessenberg form by an orthogonal similarity transformation.
This is defined in the Eigenvalues module.
MatrixType the type of the matrix of which we are computing the Hessenberg decomposition
This class performs an Hessenberg decomposition of a matrix A.
In the real case, the Hessenberg decomposition consists of an orthogonal matrix Q and a Hessenberg matrix H such that $A = QHQ^T$. An orthogonal matrix is a matrix whose inverse equals its transpose ($Q^{-1} = Q^T$). A Hessenberg matrix has zeros below the subdiagonal, so it is almost upper triangular.
The Hessenberg decomposition of a complex matrix is $A = QHQ^*$ with Q unitary (that is, $Q^{-1} = Q^*$).
Call the function compute() to compute the Hessenberg decomposition of a given matrix. Alternatively, you can use the HessenbergDecomposition(const MatrixType) constructor which computes the Hessenberg decomposition at construction time. Once the decomposition is computed, you can use the matrixH() and matrixQ() functions to construct the matrices H and Q in the decomposition.
The documentation for matrixH() contains an example of the typical use of this class.
See Also
class ComplexSchur, class Tridiagonalization, QR Module

12.4.2.1 Member Function Documentation

HessenbergDecomposition& compute (const MatrixType & matrix)
Computes Hessenberg decomposition of given matrix. Parameters [in] matrix Square matrix whose Hessenberg decomposition is to be computed.
Returns Reference to *this The Hessenberg decomposition is computed by bringing the columns of the matrix successively in the required form using Householder reflections (see, e.g., Algorithm 7.4.2 in Golub & Van Loan (1996)). The cost is $10n^3/3$ flops, where n denotes the size of the given matrix. This method reuses of the allocated data in the HessenbergDecomposition object.
Example:

```
MatrixXcf A = MatrixXcf::Random(4,4);
HessenbergDecomposition<MatrixXcf> hd(4);
hd.compute(A);
cout << "The matrix H in the decomposition of A is:" << endl << hd.matrixH() << endl;
hd.compute(2*A); // re-use hd to compute and store decomposition of 2A
cout << "The matrix H in the decomposition of 2A is:" << endl << hd.matrixH() << endl;
```

```
Output:
The matrix H in the decomposition of A is:
(-0.211,0.68)      (0.346,0.216)  (-0.688,0.00979)     (0.0451,0.584)
(-1.45,0) (-0.0574,-0.0123)     (-0.196,0.385)      (0.395,0.389)
(0,0)           (1.68,0)    (-0.397,-0.552)     (0.156,-0.241)
(0,0)              (0,0)           (1.56,0)     (0.876,-0.423)
The matrix H in the decomposition of 2A is:
(-0.422,1.36)     (0.691,0.431)   (-1.38,0.0196)     (0.0902,1.17)
(-2.91,0) (-0.115,-0.0246)     (-0.392,0.77)     (0.791,0.777)
(0,0)          (3.36,0)     (-0.795,-1.1)    (0.311,-0.482)
(0,0)             (0,0)          (3.12,0)     (1.75,-0.846)
```

const CoeffVectorType& **householderCoefficients** ( ) const
Returns the Householder coefficients. Returns a const reference to the vector of Householder coefficients Precondition Either the constructor HessenbergDecomposition(const MatrixType&) or the member function compute(const MatrixType&) has been called before to compute the Hessenberg decomposition of a matrix. The Householder coefficients allow the reconstruction of the matrix in the Hessenberg decomposition from the packed data. See Also packedMatrix(), Householder module

MatrixHReturnType **matrixH** ( ) const
Constructs the Hessenberg matrix H in the decomposition. Returns expression object representing the matrix H Precondition Either the constructor HessenbergDecomposition(const MatrixType&) or the member function compute(const MatrixType&) has been called before to compute the Hessenberg decomposition of a matrix. The object returned by this function constructs the Hessenberg matrix H when it is assigned to a matrix or otherwise evaluated. The matrix H is constructed from the packed matrix as returned by packedMatrix(): The upper part (including the subdiagonal) of the packed matrix contains the matrix H. It may sometimes be better to directly use the packed matrix instead of constructing the matrix H.
Example:

```
Matrix4f A = MatrixXf::Random(4,4);
cout << "Here is a random 4x4 matrix:" << endl << A << endl;
HessenbergDecomposition<MatrixXf> hessOfA(A);
MatrixXf H = hessOfA.matrixH();
cout << "The Hessenberg matrix H is:" << endl << H << endl;
MatrixXf Q = hessOfA.matrixQ();
cout << "The orthogonal matrix Q is:" << endl << Q << endl;
cout << "Q H Q^T is:" << endl << Q * H * Q.transpose() << endl;
```

```
Output:
Here is a random 4x4 matrix:
0.68   0.823  -0.444   -0.27
-0.211  -0.605   0.108  0.0268
0.566   -0.33 -0.0452   0.904
0.597   0.536   0.258   0.832
The Hessenberg matrix H is:
0.68  -0.691  -0.645   0.235
0.849   0.836  -0.419   0.794
0  -0.469  -0.547 -0.0731
0       0  -0.559  -0.107
The orthogonal matrix Q is:
1       0       0       0
0  -0.249  -0.958   0.144
0   0.667  -0.277  -0.692
0   0.703 -0.0761   0.707
Q H Q^T is:
0.68   0.823  -0.444   -0.27
-0.211  -0.605   0.108  0.0268
0.566   -0.33 -0.0452   0.904
0.597   0.536   0.258   0.832
```

See Also matrixQ(), packedMatrix()

HouseholderSequenceType **matrixQ** ( ) const
Reconstructs the orthogonal matrix Q in the decomposition. Returns object representing the matrix Q Precondition Either the constructor HessenbergDecomposition(const MatrixType&) or the member function compute(const MatrixType&) has been called before to compute the Hessenberg decomposition of a matrix. This function returns a light-weight object of template class HouseholderSequence. You can either apply it directly to a matrix or you can convert it to a matrix of type MatrixType. See Also matrixH() for an example, class HouseholderSequence

const MatrixType& **packedMatrix** ( ) const
Returns the internal representation of the decomposition. Returns a const reference to a matrix with the internal representation of the decomposition.
Precondition Either the constructor HessenbergDecomposition(const MatrixType&) or the member function compute(const MatrixType&) has been called before to compute the Hessenberg decomposition of a matrix.
The returned matrix contains the following information:
âĂć the upper part and lower sub-diagonal represent the Hessenberg matrix H
âĂć the rest of the lower part contains the Householder vectors that, combined with Householder coefficients returned by householderCoefficients(), allows to reconstruct the matrix Q as $Q = H\_N - 1 \ldots H\_1 H\_0$. Here, the matrices H_i are the Householder transformations $H\_i = (I - h\_i v\_i v\_i^T)$ where h_i is the ith Householder coefficient and v_i is the Householder vector defined by $v\_i = [0, \ldots, 0, 1, M(i+2, i), \ldots, M(N-1, i)]^T$ with M the matrix returned by this function.See LAPACK for further details on this packed storage.
Example:

```
Matrix4d A = Matrix4d::Random(4,4);
cout << "Here is a random 4x4 matrix:" << endl << A << endl;
HessenbergDecomposition<Matrix4d> hessOfA(A);
Matrix4d pm = hessOfA.packedMatrix();
cout << "The packed matrix M is:" << endl << pm << endl;
cout << "The upper Hessenberg part corresponds to the matrix H, which is:"
<< endl << hessOfA.matrixH() << endl;
Vector3d hc = hessOfA.householderCoefficients();
cout << "The vector of Householder coefficients is:" << endl << hc << endl;
```

```
Output:
Here is a random 4x4 matrix:
0.68   0.823  -0.444   -0.27
-0.211  -0.605   0.108  0.0268
0.566   -0.33 -0.0452   0.904
0.597   0.536   0.258   0.832
The packed matrix M is:
0.68  -0.691  -0.645   0.235
0.849   0.836  -0.419   0.794
-0.534  -0.469  -0.547 -0.0731
-0.563   0.344  -0.559  -0.107
The upper Hessenberg part corresponds to the matrix H, which is:
0.68  -0.691  -0.645   0.235
0.849   0.836  -0.419   0.794
```

```
0  -0.469  -0.547 -0.0731
0       0  -0.559  -0.107
The vector of Householder coefficients is:
1.25
1.79
0
```

See Also householderCoefficients()

12.4.3 Real QZ Decomposition

Performs a real QZ decomposition of a pair of square matrices.
This is defined in the Eigenvalues module.
MatrixType the type of the matrix of which we are computing the real QZ decomposition; this is expected to be an instantiation of the Matrix class template.
Given a real square matrices A and B, this class computes the real QZ decomposition: $A = QSZ$, $B = QTZ$ where Q and Z are real orthogonal matrixes, T is upper-triangular matrix, and S is upper quasi-triangular matrix. An orthogonal matrix is a matrix whose inverse is equal to its transpose, $U^{-1} = U^T$. A quasi-triangular matrix is a block-triangular matrix whose diagonal consists of 1-by-1 blocks and 2-by-2 blocks where further reduction is impossible due to complex eigenvalues.
The eigenvalues of the pencil $A - zB$ can be obtained from 1x1 and 2x2 blocks on the diagonals of S and T.
Call the function compute() to compute the real QZ decomposition of a given pair of matrices. Alternatively, you can use the RealQZ(const MatrixType B, const MatrixType B, bool computeQZ) constructor which computes the real QZ decomposition at construction time. Once the decomposition is computed, you can use the matrixS(), matrixT(), matrixQ() and matrixZ() functions to retrieve the matrices S, T, Q and Z in the decomposition. If computeQZ==false, some time is saved by not computing matrices Q and Z.
Example:

```
MatrixXf A = MatrixXf::Random(4,4);
MatrixXf B = MatrixXf::Random(4,4);
RealQZ<MatrixXf> qz(4); // preallocate space for 4x4 matrices
qz.compute(A,B); // A = Q S Z, B = Q T Z// print original matrices and result of
    decomposition
cout << "A:\n" << A << "\n" << "B:\n" << B << "\n";
cout << "S:\n" << qz.matrixS() << "\n" << "T:\n" << qz.matrixT() << "\n";
cout << "Q:\n" << qz.matrixQ() << "\n" << "Z:\n" << qz.matrixZ() << "\n";// verify
    precision
cout << "\nErrors:" << "\n|A-QSZ|: "
<< (A-qz.matrixQ()*qz.matrixS()*qz.matrixZ()).norm()
<< ", |B-QTZ|: " << (B-qz.matrixQ()*qz.matrixT()*qz.matrixZ()).norm()
<< "\n|QQ* - I|: " << (qz.matrixQ()*qz.matrixQ().adjoint() -
    MatrixXf::Identity(4,4)).norm()
<< ", |ZZ* - I|: " << (qz.matrixZ()*qz.matrixZ().adjoint() -
    MatrixXf::Identity(4,4)).norm() << "\n";
```

```
Output:
A:
0.68   0.823  -0.444   -0.27
-0.211  -0.605   0.108  0.0268
0.566   -0.33 -0.0452   0.904
0.597   0.536   0.258   0.832
B:
0.271 -0.967 -0.687  0.998
0.435 -0.514 -0.198 -0.563
-0.717 -0.726  -0.74 0.0259
0.214  0.608 -0.782  0.678
```

```
S:
0.927 -0.928   0.643 -0.227
-0.594   0.36  0.146 -0.606
0      0  -0.398 -0.164
0      0      0  -1.12
T:
1.51  0.278 -0.238  0.501
0  -1.04  0.519 -0.239
0      0  -1.25  0.438
0      0      0  0.746
Q:
0.603  0.011  0.552  0.576
-0.142  0.243  0.761 -0.585
0.092 -0.958  0.152 -0.223
0.78  0.149 -0.306 -0.526
Z:
0.284    0.26  -0.696   0.606
-0.918  -0.108   -0.38  0.0406
-0.269   0.783   0.462    0.32
-0.0674  -0.555   0.398   0.727
Errors:
|A-QSZ|: 1.13e-06, |B-QTZ|: 1.81e-06
|QQ* - I|: 1.01e-06, |ZZ* - I|: 7.02e-07
```

Note The implementation is based on the algorithm in Golub & Van Loan (1996), and Moler & Stewart (1973).

12.4.3.1 Member Function Documentation

RealQZ¡ MatrixType ¿ & **compute** (const MatrixType & A, const MatrixType & B, bool computeQZ = true)
Computes QZ decomposition of given matrix. Parameters [in] A Matrix A. [in] B Matrix B. [in] computeQZ If false, A and Z are not computed.
Returns Reference to *this References Eigen::NoConvergence, and Eigen::Success. Referenced by RealQZ¡ MatrixType ¿::RealQZ().

ComputationInfo **info** ( ) const
Reports whether previous computation was successful. Returns Success if computation was succesful, NoConvergence otherwise.

const MatrixType& **matrixQ** ( ) const
Returns matrix Q in the QZ decomposition. Returns A const reference to the matrix Q.

const MatrixType& **matrixS** ( ) const
Returns matrix S in the QZ decomposition. Returns A const reference to the matrix S.

const MatrixType& **matrixT** ( ) const
Returns matrix S in the QZ decomposition. Returns A const reference to the matrix S.

const MatrixType& **matrixZ** ( ) const
Returns matrix Z in the QZ decomposition. Returns A const reference to the matrix Z.

RealQZ& **setMaxIterations** (Index maxIters)
Sets the maximal number of iterations allowed to converge to one eigenvalue or decouple the problem. Referenced by GeneralizedEigenSolver¡ _MatrixType ¿::setMaxIterations().

12.4.4 Real Schur Decomposition

Performs a real Schur decomposition of a square matrix. This is defined in the Eigenvalues module.
MatrixType the type of the matrix of which we are computing the real Schur decomposition; this is expected to be an instantiation of the Matrix class template.
Given a real square matrix A, this class computes the real Schur decomposition: $A = UTU^T$ where U is a real orthogonal matrix and T is a real quasi-triangular matrix. An orthogonal matrix is a matrix whose inverse is equal to its transpose, $U^{-1} = U^T$. A quasi-triangular matrix is a block-triangular matrix whose diagonal consists of 1-by-1 blocks and 2-by-2 blocks with complex eigenvalues. The eigenvalues of the blocks on the diagonal of T are the same as the eigenvalues of the matrix A, and thus the real Schur decomposition is used in EigenSolver to compute the eigendecomposition of a matrix.
Call the function compute() to compute the real Schur decomposition of a given matrix. Alternatively, you can use the RealSchur(const MatrixType, bool) constructor which computes the real Schur decomposition at construction time. Once the decomposition is computed, you can use the matrixU() and matrixT() functions to retrieve the matrices U and T in the decomposition.
The documentation of RealSchur(const MatrixType, bool) contains an example of the typical use of this class.
See Also
class ComplexSchur, class EigenSolver, class ComplexEigenSolver
Example:

```
MatrixXd A = MatrixXd::Random(6,6);
cout << "Here is a random 6x6 matrix, A:" << endl << A << endl << endl;
RealSchur<MatrixXd> schur(A);
cout << "The orthogonal matrix U is:" << endl << schur.matrixU() << endl;
cout << "The quasi-triangular matrix T is:" << endl << schur.matrixT() << endl <<
    endl;
MatrixXd U = schur.matrixU();
MatrixXd T = schur.matrixT();
cout << "U * T * U^T = " << endl << U * T * U.transpose() << endl;
```

```
Output:
Here is a random 6x6 matrix, A:
0.68   -0.33   -0.27  -0.717  -0.687  0.0259
-0.211   0.536  0.0268   0.214  -0.198   0.678
0.566  -0.444   0.904  -0.967   -0.74   0.225
0.597   0.108   0.832  -0.514  -0.782  -0.408
0.823 -0.0452   0.271  -0.726   0.998   0.275
-0.605   0.258   0.435   0.608  -0.563  0.0486

The orthogonal matrix U is:
0.348  -0.754 0.00435  -0.351  0.0145   0.432
-0.16  -0.266  -0.747   0.457  -0.366  0.0571
0.505  -0.157  0.0746   0.644   0.518  -0.177
0.703   0.324  -0.409  -0.349  -0.187  -0.275
0.296   0.372    0.24   0.324  -0.379   0.684
-0.126   0.305   -0.46  -0.161   0.647   0.485
The quasi-triangular matrix T is:
```

```
-0.2   -1.83   0.864   0.271    1.09    0.14
0.647   0.298 -0.0536   0.676  -0.288   0.023
0       0   0.967  -0.201  -0.429   0.847
0       0       0   0.353   0.602   0.694
0       0       0       0   0.572   -1.03
0       0       0       0  0.0184   0.664

U * T * U^T =
0.68   -0.33   -0.27  -0.717  -0.687  0.0259
-0.211   0.536  0.0268   0.214  -0.198   0.678
0.566  -0.444   0.904  -0.967   -0.74   0.225
0.597   0.108   0.832  -0.514  -0.782  -0.408
0.823 -0.0452   0.271  -0.726   0.998   0.275
-0.605   0.258   0.435   0.608  -0.563  0.0486
```

12.4.4.1 Member Function Documentation

Member Function DocumentationRealSchur¡ MatrixType ¿ & **compute** (const MatrixType & matrix, bool computeU = true)
Computes Schur decomposition of given matrix.
Parameters
[in] matrix Square matrix whose Schur decomposition is to be computed.
[in] computeU If true, both T and U are computed; if false, only T is computed.
Returns Reference to *this
The Schur decomposition is computed by first reducing the matrix to Hessenberg form using the class HessenbergDecomposition. The Hessenberg matrix is then reduced to triangular form by performing Francis QR iterations with implicit double shift. The cost of computing the Schur decomposition depends on the number of iterations; as a rough guide, it may be taken to be flops if computeU is true and flops if computeU is false.
Example:

```
MatrixXf A = MatrixXf::Random(4,4);
RealSchur<MatrixXf> schur(4);
schur.compute(A, /* computeU = */ false);
cout << "The matrix T in the decomposition of A is:" << endl << schur.matrixT() <<
    endl;
schur.compute(A.inverse(), /* computeU = */ false);
cout << "The matrix T in the decomposition of A^(-1) is:" << endl << schur.matrixT()
    << endl;
```

```
Output:
The matrix T in the decomposition of A is:
0.523 -0.698  0.148  0.742
0.475  0.986 -0.793  0.721
0      0  -0.28  -0.77
0      0 0.0145 -0.367
The matrix T in the decomposition of A^(-1) is:
-3.06 -4.57 -6.05  5.39
0.168 -2.62 -3.33  3.86
```

```
0      0 0.434  0.56
0      0 -1.06  1.35
```

See Also compute(const MatrixType&, bool, Index) Referenced by RealSchur¡ MatrixType ¿::RealSchur().

RealSchur& **computeFromHessenberg** (const HessMatrixType & matrixH, const OrthMatrixType & matrixQ, bool computeU)
Computes Schur decomposition of a Hessenberg matrix $H = ZTZ^T$. Parameters
[in] matrixH Matrix in Hessenberg form H
[in] matrixQ orthogonal matrix Q that transform a matrix A to H : $A = QHQ^T$
computeU Computes the matriX U of the Schur vectors
Returns Reference to *this This routine assumes that the matrix is already reduced in Hessenberg form matrixH using either the class HessenbergDecomposition or another mean. It computes the upper quasi-triangular matrix T of the Schur decomposition of H When computeU is true, this routine computes the matrix U such that $A = UTU^T = (QZ)T(QZ)^T = QHQ^T$ where A is the initial matrix NOTE Q is referenced if computeU is true; so, if the initial orthogonal matrix is not available, the user should give an identity matrix (Q.setIdentity()) See Also compute(const MatrixType&, bool)

ComputationInfo **info** ( ) const
Reports whether previous computation was successful. Returns Success if computation was succesful, NoConvergence otherwise.

const MatrixType& **matrixT** ( ) const
Returns the quasi-triangular matrix in the Schur decomposition. Returns A const reference to the matrix T. Precondition Either the constructor RealSchur(const MatrixType&, bool) or the member function compute(const MatrixType&, bool) has been called before to compute the Schur decomposition of a matrix. See Also RealSchur(const MatrixType&, bool) for an example

const MatrixType& **matrixU** ( ) const
Returns the orthogonal matrix in the Schur decomposition. Returns A const reference to the matrix U. Precondition Either the constructor RealSchur(const MatrixType&, bool) or the member function compute(const MatrixType&, bool) has been called before to compute the Schur decomposition of a matrix, and computeU was set to true (the default value). See Also RealSchur(const MatrixType&, bool) for an example

RealSchur& **setMaxIterations** (Index maxIters)
Sets the maximum number of iterations allowed. If not specified by the user, the maximum number of iterations is m_maxIterationsPerRow times the size of the matrix. Referenced by EigenSolver¡ _MatrixType ¿::setMaxIterations().

Member Data Documentationconst int **m_maxIterationsPerRow**
Maximum number of iterations per row. If not otherwise specified, the maximum number of iterations is this number times the size of the matrix. It is currently set to 40.

12.4.5 Complex Schur Decomposition

Performs a complex Schur decomposition of a real or complex square matrix.
This is defined in the Eigenvalues module.
MatrixType the type of the matrix of which we are computing the Schur decomposition; this is expected to be an instantiation of the Matrix class template.
Given a real or complex square matrix A, this class computes the Schur decomposition: $A = UTU^*$ where U is a unitary complex matrix, and T is a complex upper triangular matrix. The diagonal of the matrix T corresponds to the eigenvalues of the matrix A.
Call the function compute() to compute the Schur decomposition of a given matrix. Alternatively, you can use the ComplexSchur(const MatrixType, bool) constructor which computes the Schur decomposition at construction time. Once the decomposition is computed, you can use the matrixU() and matrixT() functions to retrieve the matrices U and V in the decomposition.
See Also
class RealSchur, class EigenSolver, class ComplexEigenSolver

12.4.5.1 Member Function Documentation

ComplexSchur¡ MatrixType ¿ & **compute** (const MatrixType & matrix, bool computeU = true)
Computes Schur decomposition of given matrix. Parameters [in] matrix Square matrix whose Schur decomposition is to be computed. [in] computeU If true, both T and U are computed; if false, only T is computed.
Returns Reference to *this
The Schur decomposition is computed by first reducing the matrix to Hessenberg form using the class HessenbergDecomposition. The Hessenberg matrix is then reduced to triangular form by performing QR iterations with a single shift. The cost of computing the Schur decomposition depends on the number of iterations; as a rough guide, it may be taken on the number of iterations; as a rough guide, it may be taken to be complex flops, or complex flops if computeU is false.
Example:

```
MatrixXcf A = MatrixXcf::Random(4,4);
ComplexSchur<MatrixXcf> schur(4);
schur.compute(A);
cout << "The matrix T in the decomposition of A is:" << endl << schur.matrixT() <<
    endl;
schur.compute(A.inverse());
cout << "The matrix T in the decomposition of A^(-1) is:" << endl << schur.matrixT()
    << endl;
```

```
Output:
The matrix T in the decomposition of A is:
(-0.691,-1.63)  (0.763,-0.144) (-0.104,-0.836) (-0.462,-0.378)
(0,0)   (-0.758,1.22)  (-0.65,-0.772)  (-0.244,0.113)
(0,0)           (0,0)   (0.137,0.505) (0.0687,-0.404)
(0,0)           (0,0)           (0,0)   (1.52,-0.402)
The matrix T in the decomposition of A^(-1) is:
(0.501,-1.84)    (-1.01,-0.984)       (0.636,1.3)    (-0.676,0.352)
(0,0)   (-0.369,-0.593)     (0.0733,0.18) (-0.0658,-0.0263)
(0,0)             (0,0)    (-0.222,0.521)    (-0.191,0.121)
```

```
(0,0)           (0,0)           (0,0)      (0.614,0.162)
```

See Also compute(const MatrixType&, bool, Index) References ComplexSchur¡ _MatrixType ¿::computeFromHessenberg(), and Eigen::Success. Referenced by ComplexSchur¡ MatrixType ¿::ComplexSchur().

ComplexSchur& **computeFromHessenberg** (const HessMatrixType & matrixH, const OrthMatrixType & matrixQ, bool computeU = true)
Compute Schur decomposition from a given Hessenberg matrix.
Parameters
[in] matrixH Matrix in Hessenberg form H
[in] matrixQ orthogonal matrix Q that transform a matrix A to H : $A = QHQ^T$
computeU Computes the matriX U of the Schur vectors
Returns Reference to *this
This routine assumes that the matrix is already reduced in Hessenberg form matrixH using either the class HessenbergDecomposition or another mean. It computes the upper quasi-triangular matrix T of the Schur decomposition of H When computeU is true, this routine computes the matrix U such that $A = UTU^T = (QZ)T(QZ)^T = QHQ^T$ where A is the initial matrix
NOTE Q is referenced if computeU is true; so, if the initial orthogonal matrix is not available, the user should give an identity matrix (Q.setIdentity()) See Also compute(const MatrixType&, bool) Referenced by ComplexSchur¡ _MatrixType ¿::compute().

ComputationInfo **info** ( ) const
Reports whether previous computation was successful. Returns Success if computation was succesful, NoConvergence otherwise. Referenced by ComplexEigenSolver¡ _MatrixType ¿::info().

const ComplexMatrixType& **matrixT** ( ) const
Returns the triangular matrix in the Schur decomposition.
Returns A const reference to the matrix T.
It is assumed that either the constructor ComplexSchur(const MatrixType& matrix, bool computeU) or the member function compute(const MatrixType& matrix, bool computeU) has been called before to compute the Schur decomposition of a matrix.
Note that this function returns a plain square matrix. If you want to reference only the upper triangular part, use: schur.matrixT().triangularView¡Upper¿()
Example:

```
MatrixXcf A = MatrixXcf::Random(4,4);
cout << "Here is a random 4x4 matrix, A:"
<< endl << A << endl << endl;ComplexSchur<MatrixXcf> schurOfA(A, false); // false
   means do not compute U
cout << "The triangular matrix T is:"
<< endl << schurOfA.matrixT() << endl;
```

```
Output:
Here is a random 4x4 matrix, A:
(-0.211,0.68)  (0.108,-0.444)   (0.435,0.271) (-0.198,-0.687)
(0.597,0.566) (0.258,-0.0452)  (0.214,-0.717)  (-0.782,-0.74)
(-0.605,0.823)  (0.0268,-0.27) (-0.514,-0.967)  (-0.563,0.998)
(0.536,-0.33)   (0.832,0.904)  (0.608,-0.726)  (0.678,0.0259)
```

```
The triangular matrix T is:
(-0.691,-1.63)  (0.763,-0.144) (-0.104,-0.836) (-0.462,-0.378)
(0,0)   (-0.758,1.22)  (-0.65,-0.772)  (-0.244,0.113)
(0,0)           (0,0)   (0.137,0.505) (0.0687,-0.404)
(0,0)           (0,0)           (0,0)   (1.52,-0.402)
```

const ComplexMatrixType& **matrixU** ( ) const
Returns the unitary matrix in the Schur decomposition. Returns A const reference to the matrix U. It is assumed that either the constructor ComplexSchur(const MatrixType& matrix, bool computeU) or the member function compute(const MatrixType& matrix, bool computeU) has been called before to compute the Schur decomposition of a matrix, and that computeU was set to true (the default value).
Example:

```
MatrixXcf A = MatrixXcf::Random(4,4);
cout << "Here is a random 4x4 matrix, A:" << endl << A << endl << endl;
ComplexSchur<MatrixXcf> schurOfA(A);
cout << "The unitary matrix U is:" << endl << schurOfA.matrixU() << endl;
```

```
Output:
Here is a random 4x4 matrix, A:
(-0.211,0.68)  (0.108,-0.444)   (0.435,0.271) (-0.198,-0.687)
(0.597,0.566) (0.258,-0.0452)  (0.214,-0.717)  (-0.782,-0.74)
(-0.605,0.823)  (0.0268,-0.27) (-0.514,-0.967)  (-0.563,0.998)
(0.536,-0.33)   (0.832,0.904)  (0.608,-0.726)  (0.678,0.0259)

The unitary matrix U is:
(-0.122,0.271)   (0.354,0.255)    (-0.7,0.321) (0.0909,-0.346)
(0.247,0.23)  (0.435,-0.395)   (0.184,-0.38)  (0.492,-0.347)
(0.859,-0.0877)  (0.00469,0.21) (-0.256,0.0163)   (0.133,0.355)
(-0.116,0.195) (-0.484,-0.432)  (-0.183,0.359)   (0.559,0.231)
```

ComplexSchur& **setMaxIterations** (Index maxIters)
Sets the maximum number of iterations allowed. If not specified by the user, the maximum number of iterations is m_maxIterationsPerRow times the size of the matrix. Referenced by ComplexEigenSolver¡ _MatrixType ¿::setMaxIterations().

Member Data Documentationconst int m
textbfmaxIterationsPerRow
Maximum number of iterations per row. If not otherwise specified, the maximum number of iterations is this number times the size of the matrix. It is currently set to 30.

12.5 Matrix Functions

Matrix functions are defined as follows. Suppose that f is an entire function (that is, a function on the complex plane that is everywhere complex differentiable). Then its Taylor series

$$f(0) + f'(0)x + \frac{f''(0)}{2}x^2 + \frac{f'''(0)}{3!}x^3 + \cdots \tag{12.5.1}$$

converges to $f(x)$. In this case, we can define the matrix function by the same series:

$$f(M) = f(0) + f'(0)M + \frac{f''(0)}{2}M^2 + \frac{f'''(0)}{3!}M^3 + \cdots \tag{12.5.2}$$

12.5.1 Matrix Square Root

Function **MatSqrt**(***M*** *As mpNum[,]*) As mpNum

The function MatSqrt returns an expression representing the matrix square root of the real matrix M.

Parameter:
M: the real matrix of which we are computing the matrix square root.

Function **cplxMatSqrt**(***M*** *As mpNum[,]*) As mpNum

The function cplxMatSqrt returns an expression representing the matrix square root of the complex matrix M.

Parameter:
M: the complex matrix of which we are computing the matrix square root.

Compute the matrix square root.
Parameters
[in] M invertible matrix whose square root is to be computed.
Returns: expression representing the matrix square root of M.
The matrix square root of M is the matrix $M^{1/2}$ whose square is the original matrix; so if $S = M^{1/2}$ then $S^2 = M$.
In the real case, the matrix M should be invertible and it should have no eigenvalues which are real and negative (pairs of complex conjugate eigenvalues are allowed). In that case, the matrix has a square root which is also real, and this is the square root computed by this function.
The matrix square root is computed by first reducing the matrix to quasi-triangular form with the real Schur decomposition. The square root of the quasi-triangular matrix can then be computed directly. The cost is approximately $25n^3$ real flops for the real Schur decomposition and n^3 real flops for the remainder (though the computation time in practice is likely more than this indicates).
Details of the algorithm can be found in Higham (1987).
If the matrix is positive-definite symmetric, then the square root is also positive-definite symmetric. In this case, it is best to use SelfAdjointEigenSolver::operatorSqrt() to compute it.
In the complex case, the matrix M should be invertible; this is a restriction of the algorithm. The square root computed by this algorithm is the one whose eigenvalues have an argument in the interval $\left(-\frac{1}{2}\pi, \frac{1}{2}\pi\right]$. This is the usual branch cut.

The computation is the same as in the real case, except that the complex Schur decomposition is used to reduce the matrix to a triangular matrix. The theoretical cost is the same. Details are in Björck & Hammarling (1983).
Example: The following program checks that the square root of

$$\begin{pmatrix} \cos\left(\frac{1}{3}\pi\right) & -\sin\left(\frac{1}{3}\pi\right) \\ \sin\left(\frac{1}{3}\pi\right) & \cos\left(\frac{1}{3}\pi\right) \end{pmatrix} \tag{12.5.3}$$

corresponding to a rotation over 60 degrees, is a rotation over 30 degrees:

$$\begin{pmatrix} \cos\left(\frac{1}{6}\pi\right) & -\sin\left(\frac{1}{6}\pi\right) \\ \sin\left(\frac{1}{6}\pi\right) & \cos\left(\frac{1}{6}\pi\right) \end{pmatrix} \tag{12.5.4}$$

```
#include <unsupported/Eigen/MatrixFunctions>
#include <iostream>

using namespace Eigen;

int main()
{
const double pi = std::acos(-1.0);

MatrixXd A(2,2);
A << cos(pi/3), -sin(pi/3),
sin(pi/3), cos(pi/3);
std::cout << "The matrix A is:\n" << A << "\n\n";
std::cout << "The matrix square root of A is:\n" << A.sqrt() << "\n\n";
std::cout << "The square of the last matrix is:\n"
<< A.sqrt() * A.sqrt() << "\n";
}
```

```
Output:
The matrix A is:
0.5 -0.866025
0.866025       0.5

The matrix square root of A is:
0.866025     -0.5
0.5 0.866025

The square of the last matrix is:
0.5 -0.866025
0.866025       0.5
```

12.5.2 Matrix Exponential

Function **MatExp**(***M*** *As mpNum[,]*) As mpNum

The function MatExp returns an expression representing the matrix exponential of the real matrix M.

Parameter:
M: the real matrix of which we are computing the matrix exponential.

Function **cplxMatExp**(***M*** *As mpNum[,]*) As mpNum

The function cplxMatExp returns an expression representing the matrix exponential of the complex matrix M.

Parameter:
M: the complex matrix of which we are computing the matrix exponential.

Compute the matrix exponential.
Parameters: [in] M matrix whose exponential is to be computed.
Returns: expression representing the matrix exponential of M.
The matrix exponential of M is defined by

$$\exp(M) = \sum_{k=0}^{\infty} \frac{M^k}{k!}. \tag{12.5.5}$$

The matrix exponential can be used to solve linear ordinary differential equations: the solution of $y' = My$ with the initial condition $y(0) = y_0$ is given by $y(t) = \exp(M)y_0$. The cost of the computation is approximately $20n^3$ for matrices of size n. The number 20 depends weakly on the norm of the matrix.
The matrix exponential is computed using the scaling-and-squaring method combined with Padé approximation. The matrix is first rescaled, then the exponential of the reduced matrix is computed approximant, and then the rescaling is undone by repeated squaring. The degree of the Padé approximant is chosen such that the approximation error is less than the round-off error. However, errors may accumulate during the squaring phase.
Details of the algorithm can be found in Higham (2005).
Example: The following program checks that

$$\exp\begin{pmatrix} 0 & \frac{1}{4}\pi & 0 \\ -\frac{1}{4}\pi & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{2}\sqrt{2} & -\frac{1}{2}\sqrt{2} & 0 \\ \frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{12.5.6}$$

This corresponds to a rotation of $\frac{1}{4}\pi$ radians around the z-axis.

```
#include <unsupported/Eigen/MatrixFunctions>
#include <iostream>

using namespace Eigen;

int main()
{
const double pi = std::acos(-1.0);
```

```
MatrixXd A(3,3);
A << 0,   -pi/4, 0,
pi/4, 0,    0,
0,    0,    0;
std::cout << "The matrix A is:\n" << A << "\n\n";
std::cout << "The matrix exponential of A is:\n"
<< A.exp() << "\n\n";
}
```

```
Output:
The matrix A is:
0 -0.785398         0
0.785398         0         0
0         0         0

The matrix exponential of A is:
0.707107 -0.707107         0
0.707107  0.707107         0
0         0         1
```

Note: M has to be a matrix of real or complex.

12.5.3 Matrix Logarithm

Function **MatLog**(***M*** *As mpNum[,]*) As mpNum

The function MatLog returns an expression representing the matrix logarithm of the real matrix M.

Parameter:
M: the real matrix of which we are computing the matrix logarithm.

Function **cplxMatLog**(***M*** *As mpNum[,]*) As mpNum

The function cplxMatLog returns an expression representing the matrix logarithm of the complex matrix M.

Parameter:
M: the complex matrix of which we are computing the matrix logarithm.

Compute the matrix logarithm.
Parameters: [in] M invertible matrix whose logarithm is to be computed.
Returns: expression representing the matrix logarithm root of M.
The matrix logarithm of M is a matrix X such that $\exp(X) = M$ where exp denotes the matrix exponential. As for the scalar logarithm, the equation $\exp(X) = M$ may have multiple solutions; this function returns a matrix whose eigenvalues have imaginary part in the interval $(-\pi, \pi]$.
In the real case, the matrix M should be invertible and it should have no eigenvalues which are real and negative (pairs of complex conjugate eigenvalues are allowed). In the complex case, it only needs to be invertible.

This function computes the matrix logarithm using the Schur-Parlett algorithm as implemented by MatrixBase::matrixFunction(). The logarithm of an atomic block is computed by MatrixLogarithmAtomic, which uses direct computation for 1-by-1 and 2-by-2 blocks and an inverse scaling-and-squaring algorithm for bigger blocks, with the square roots computed by MatrixBase::sqrt(). Details of the algorithm can be found in Section 11.6.2 of Higham (2008).
Example: The following program checks that

$$\log \begin{pmatrix} \frac{1}{2}\sqrt{2} & -\frac{1}{2}\sqrt{2} & 0 \\ \frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & \frac{1}{4}\pi & 0 \\ -\frac{1}{4}\pi & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{12.5.7}$$

This corresponds to a rotation of $\frac{1}{4}\pi$ radians around the z-axis. This is the inverse of the example used in the documentation of exp().

```
#include <unsupported/Eigen/MatrixFunctions>
#include <iostream>

using namespace Eigen;

int main()
{
using std::sqrt;
MatrixXd A(3,3);
A << 0.5*sqrt(2), -0.5*sqrt(2), 0,
0.5*sqrt(2), 0.5*sqrt(2), 0,
0,           0,           1;
std::cout << "The matrix A is:\n" << A << "\n\n";
std::cout << "The matrix logarithm of A is:\n" << A.log() << "\n";
}
```

```
Output:
The matrix A is:
0.707107 -0.707107         0
0.707107  0.707107         0
0         0         1

The matrix logarithm of A is:
-1.11022e-16    -0.785398            0
0.785398 -1.11022e-16            0
0            0            0
```

12.5.4 Matrix raised to arbitrary real power

Function **MatPow**(***M*** *As mpNum[,]*, ***p*** *As mpNum*) As mpNum

The function MatPow returns an expression representing the matrix power of the real matrix M.

Parameters:
M: M base of the matrix power, should be a square matrix.
p: exponent of the matrix power, should be real.

Function **cplxMatPow**(***M*** *As mpNum[,]*, ***p*** *As mpNum*) As mpNum

The function cplxMatPow returns an expression representing the matrix power of the complex matrix M.

Parameters:
M: M base of the matrix power, should be a square matrix.
p: exponent of the matrix power, should be real.

MatrixBase::pow()
Compute the matrix raised to arbitrary real power. const MatrixPowerReturnValue¡Derived¿ MatrixBase¡Derived¿::pow(RealScalar p) constParameters [in] M base of the matrix power, should be a square matrix. [in] p exponent of the matrix power, should be real.
The matrix power M^p is defined as $\exp(p\log(M))$, where exp denotes the matrix exponential, and log denotes the matrix logarithm.
The matrix M should meet the conditions to be an argument of matrix logarithm. If p is not of the real scalar type of M, it is casted into the real scalar type of M.
This function computes the matrix power using the Schur-Padé algorithm as implemented by class MatrixPower. The exponent is split into integral part and fractional part, where the fractional part is in the interval $(-1, 1)$. The main diagonal and the first super-diagonal is directly computed. Details of the algorithm can be found in Higham & Lin (2011).
Example: The following program checks that

$$\begin{pmatrix} \cos(1) & -\sin(1) & 0 \\ \sin(1) & \cos(1) & 0 \\ 0 & 0 & 1 \end{pmatrix}^{\frac{1}{4}\pi} = \begin{pmatrix} \frac{1}{2}\sqrt{2} & -\frac{1}{2}\sqrt{2} & 0 \\ \frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{12.5.8}$$

This corresponds to $\frac{1}{4}\pi$ rotations of 1 radian around the z-axis.

```
#include <unsupported/Eigen/MatrixFunctions>
#include <iostream>

using namespace Eigen;

int main()
{
const double pi = std::acos(-1.0);
Matrix3d A;
A << cos(1), -sin(1), 0,
sin(1), cos(1), 0,
0 ,     0 , 1;
std::cout << "The matrix A is:\n" << A << "\n\n"
```

```
"The matrix power A^(pi/4) is:\n" << A.pow(pi/4) << std::endl;
return 0;
}
```

```
Output:
The matrix A is:
0.540302 -0.841471         0
0.841471  0.540302         0
0         0         1

The matrix power A^(pi/4) is:
0.707107 -0.707107         0
0.707107  0.707107         0
0         0         1
```

MatrixBase::pow() is user-friendly. However, there are some circumstances under which you should use class MatrixPower directly. MatrixPower can save the result of Schur decomposition, so it's better for computing various powers for the same matrix. Example:

```
#include <unsupported/Eigen/MatrixFunctions>
#include <iostream>

using namespace Eigen;

int main()
{
Matrix4cd A = Matrix4cd::Random();
MatrixPower<Matrix4cd> Apow(A);

std::cout << "The matrix A is:\n" << A << "\n\n"
"A^3.1 is:\n" << Apow(3.1) << "\n\n"
"A^3.3 is:\n" << Apow(3.3) << "\n\n"
"A^3.7 is:\n" << Apow(3.7) << "\n\n"
"A^3.9 is:\n" << Apow(3.9) << std::endl;
return 0;
}
```

```
Output:
The matrix A is:
(-0.211234,0.680375)   (0.10794,-0.444451)   (0.434594,0.271423) (-0.198111,-0.686642)
(0.59688,0.566198) (0.257742,-0.0452059)  (0.213938,-0.716795) (-0.782382,-0.740419)
(-0.604897,0.823295) (0.0268018,-0.270431) (-0.514226,-0.967399)  (-0.563486,0.997849)
(0.536459,-0.329554)    (0.83239,0.904459)  (0.608354,-0.725537)  (0.678224,0.0258648)

A^3.1 is:
(2.80575,-0.607662) (-1.16847,-0.00660555)    (-0.760385,1.01461)   (-0.38073,-0.106512)
(1.4041,-3.61891)     (1.00481,0.186263)   (-0.163888,0.449419)   (-0.388981,-1.22629)
(-2.07957,-1.58136)     (0.825866,2.25962)     (5.09383,0.155736)    (0.394308,-1.63034)
```

```
(-0.818997,0.671026)  (2.11069,-0.00768024)    (-1.37876,0.140165)    (2.50512,-0.854429)

A^3.3 is:
(2.83571,-0.238717) (-1.48174,-0.0615217)  (-0.0544396,1.68092) (-0.292699,-0.621726)
(2.0521,-3.58316)    (0.87894,0.400548)  (0.738072,-0.121242)   (-1.07957,-1.63492)
(-3.00106,-1.10558)     (1.52205,1.92407)    (5.29759,-1.83562)  (-0.532038,-1.50253)
(-0.491353,-0.4145)     (2.5761,0.481286)  (-1.21994,0.0367069)    (2.67112,-1.06331)

A^3.7 is:
(1.42126,0.33362)   (-1.39486,-0.560486)      (1.44968,2.47066)   (-0.324079,-1.75879)
(2.65301,-1.82427)   (0.357333,-0.192429)      (2.01017,-1.4791)    (-2.71518,-2.35892)
(-3.98544,0.964861)     (2.26033,0.554254)     (3.18211,-5.94352)    (-2.22888,0.128951)
(0.944969,-2.14683)      (3.31345,1.66075) (-0.0623743,-0.848324)        (2.3897,-1.863)

A^3.9 is:
(0.0720766,0.378685) (-0.931961,-0.978624)      (1.9855,2.34105)  (-0.530547,-2.17664)
(2.40934,-0.265286)  (0.0299975,-1.08827)    (1.98974,-2.05886)   (-3.45767,-2.50235)
(-3.71666,2.3874)        (2.054,-0.303)   (0.844348,-7.29588)    (-2.59136,1.57689)
(1.87645,-2.38798)     (3.52111,2.10508)    (0.799055,-1.6122)    (1.93452,-2.44408)
```

12.5.5 Matrix General Function

Function **MatGeneralFunction**(***M*** *As mpNum[,]*, ***f*** *As mpFunction*) As mpNum

The function MatGeneralFunction returns an expression representing f applied to the real matrix M.

Parameters:
M: argument of matrix function, should be a square matrix.
f: f an entire function; f(x,n) should compute the n-th derivative of f at x.

Function **cplxMatGeneralFunction**(***M*** *As mpNum[,]*, ***f*** *As mpFunction*) As mpNum

The function cplxMatGeneralFunction returns an expression representing f applied to the complex matrix M.

Parameters:
M: argument of matrix function, should be a square matrix.
f: f an entire function; f(x,n) should compute the n-th derivative of f at x.

Compute a matrix function.
Parameters
[in] M argument of matrix function, should be a square matrix.
[in] f an entire function; f(x,n) should compute the n-th derivative of f at x.
Returns expression representing f applied to M.
Suppose that M is a matrix whose entries have type Scalar. Then, the second argument, f, should be a function with prototype
ComplexScalar f(ComplexScalar, int)
where ComplexScalar = std::complex¡Scalar¿ if Scalar is real (e.g., float or double) and ComplexScalar = Scalar if Scalar is complex.
The return value of f(x,n) should be $f^{(n)}(x)$, the n-th derivative of f at x.
This routine uses the algorithm described in Davies & Higham (2003).
The actual work is done by the MatrixFunction class. Example: The following program checks that

$$\exp\begin{pmatrix} 0 & \frac{1}{4}\pi & 0 \\ -\frac{1}{4}\pi & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{2}\sqrt{2} & -\frac{1}{2}\sqrt{2} & 0 \\ \frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{12.5.9}$$

This corresponds to a rotation of $\frac{1}{4}\pi$ radians around the z-axis. This is the same example as used in the documentation of exp().

```
#include <unsupported/Eigen/MatrixFunctions>
#include <iostream>

using namespace Eigen;

std::complex<double> expfn(std::complex<double> x, int)
{
return std::exp(x);
}
```

```
int main()
{
const double pi = std::acos(-1.0);

MatrixXd A(3,3);
A << 0,   -pi/4, 0,
pi/4, 0,    0,
0,    0,    0;

std::cout << "The matrix A is:\n" << A << "\n\n";
std::cout << "The matrix exponential of A is:\n"
<< A.matrixFunction(expfn) << "\n\n";
}
```

```
Output:
The matrix A is:
0 -0.785398         0
0.785398         0         0
0         0         0

The matrix exponential of A is:
0.707107 -0.707107         0
0.707107  0.707107         0
0         0         1
```

Note that the function expfn is defined for complex numbers x, even though the matrix A is over the reals. Instead of expfn, we could also have used StdStemFunctions::exp:
A.matrixFunction(StdStemFunctions¡std::complex¡double¿ ¿::exp, &B);

12.5.6 Matrix Sine

Function **MatSin**(***M*** *As mpNum[,]*) As mpNum

The function MatSin returns an expression representing the matrix sine of the real matrix M.

Parameter:
M: the real matrix of which we are computing the matrix sine.

Function **cplxMatSin**(***M*** *As mpNum[,]*) As mpNum

The function cplxMatSin returns an expression representing the matrix sine of the complex matrix M.

Parameter:
M: the complex matrix of which we are computing the matrix sine.

Compute the matrix sine.
Parameters: [in] M a square matrix.
Returns: expression representing $\sin(M)$.

This function calls matrixFunction() with StdStemFunctions::sin().
Example:

```
#include <unsupported/Eigen/MatrixFunctions>
#include <iostream>

using namespace Eigen;

int main()
{
MatrixXd A = MatrixXd::Random(3,3);
std::cout << "A = \n" << A << "\n\n";

MatrixXd sinA = A.sin();
std::cout << "sin(A) = \n" << sinA << "\n\n";

MatrixXd cosA = A.cos();
std::cout << "cos(A) = \n" << cosA << "\n\n";

// The matrix functions satisfy sin^2(A) + cos^2(A) = I,
// like the scalar functions.
std::cout << "sin^2(A) + cos^2(A) = \n" << sinA*sinA + cosA*cosA << "\n\n";
}
```

Output:

```
A =
0.680375   0.59688 -0.329554
-0.211234  0.823295  0.536459
0.566198 -0.604897 -0.444451

sin(A) =
0.679919    0.4579 -0.400612
-0.227278  0.821913    0.5358
0.570141 -0.676728 -0.462398

cos(A) =
0.927728  -0.530361  -0.110482
0.00969246   0.889022  -0.137604
-0.132574   -0.04289    1.16475

sin^2(A) + cos^2(A) =
1  4.44089e-16  1.94289e-16
6.38378e-16            1  5.55112e-16
0 -6.10623e-16            1
```

12.5.7 Matrix Cosine

Function **MatCos**(***M*** *As mpNum[,]*) As mpNum

The function MatCos returns an expression representing the matrix cosine of the real matrix M.

Parameter:
M: the real matrix of which we are computing the matrix cosine.

Function **cplxMatCos**(***M*** *As mpNum[,]*) As mpNum

The function cplxMatCos returns an expression representing the matrix cosine of the complex matrix M.

Parameter:
M: the complex matrix of which we are computing the matrix cosine.

Compute the matrix cosine.
Parameters: [in] M a square matrix.
Returns expression representing $\cos(M)$.
This function calls matrixFunction() with StdStemFunctions::cos().
See Also sin() for an example.

12.5.8 Matrix Hyperbolic Sine

Function **MatSinh**(***M*** *As mpNum[,]*) As mpNum

The function MatSinh returns an expression representing the matrix hyperbolic sine of the real matrix M.

Parameter:
M: the real matrix of which we are computing the matrix hyperbolic sine.

Function **cplxMatSinh**(***M*** *As mpNum[,]*) As mpNum

The function cplxMatSinh returns an expression representing the matrix hyperbolic sine of the complex matrix M.

Parameter:
M: the complex matrix of which we are computing the matrix hyperbolic sine.

Compute the matrix hyperbolic sine.
Parameters: [in] M a square matrix.
Returns: expression representing $\sinh(M)$.
This function calls matrixFunction() with StdStemFunctions::sinh().
Example:

```
#include <unsupported/Eigen/MatrixFunctions>
#include <iostream>

using namespace Eigen;
```

```
int main()
{
MatrixXf A = MatrixXf::Random(3,3);
std::cout << "A = \n" << A << "\n\n";

MatrixXf sinhA = A.sinh();
std::cout << "sinh(A) = \n" << sinhA << "\n\n";

MatrixXf coshA = A.cosh();
std::cout << "cosh(A) = \n" << coshA << "\n\n";

// The matrix functions satisfy cosh^2(A) - sinh^2(A) = I,
// like the scalar functions.
std::cout << "cosh^2(A) - sinh^2(A) = \n" << coshA*coshA - sinhA*sinhA
<< "\n\n";
}
```

```
Output:
A =
0.680375    0.59688 -0.329554
-0.211234   0.823295   0.536459
0.566198 -0.604897 -0.444451

sinh(A) =
0.682534   0.739989 -0.256871
-0.194928   0.826512   0.537546
0.562584   -0.53163 -0.425199

cosh(A) =
1.07817     0.567068     0.132125
-0.00418614      1.11649     0.135361
0.128891    0.0659989     0.851201

cosh^2(A) - sinh^2(A) =
1             0    8.9407e-08
1.29454e-07             1 -2.98023e-08
0 -2.83122e-07             1
```

12.5.9 Matrix Hyberbolic Cosine

Function **MatCosh**(***M*** *As mpNum[,]*) As mpNum

The function MatCosh returns an expression representing the matrix hyperbolic cosine of the real matrix M.

Parameter:
M: the real matrix of which we are computing the matrix hyperbolic cosine.

Function **cplxMatCosh**(***M*** *As mpNum[,]*) As mpNum

The function cplxMatCosh returns an expression representing the matrix hyperbolic cosine of the complex matrix M.

Parameter:
M: the complex matrix of which we are computing the matrix hyperbolic cosine.

Compute the matrix hyberbolic cosine.
Parameters: [in] M a square matrix.
Returns expression representing $\cosh(M)$.
This function calls matrixFunction() with StdStemFunctions::cosh().
See Also sinh() for an example.

Chapter 13

Polynomials (based on Eigen)

13.1 Polynomial Evaluation

The functions described here evaluate the polynomial $c_0+c_1x+c_2x^2+\ldots+c_{n-1}x^{n-1}$ using Horner's method for stability.

13.1.1 Polynomial Evaluation, Real Coefficients and Argument

Function **PolynomialEvaluation**(***x*** *As mpNum*, ***c*** *As mpNum[]*) As mpNum

The function PolynomialEvaluation returns the value of a polynomial for the real variable x with real coefficients c.

Parameters:
x: A real number.
c: A vector of real coefficients.

13.1.2 Polynomial Evaluation, Complex Coefficients and Argument

Function **cplxPolynomialEvaluation**(***z*** *As mpNum*, ***c*** *As mpNum[]*) As mpNum

The function cplxPolynomialEvaluation returns the value of a polynomial for the complex variable z with complex coefficients c.

Parameters:
z: A complex number.
c: A vector of complex coefficients.

13.1.3 Examples

```
Sub DemoPolyComplexEvalComplex()
Dim c() As mp_complex, n As Long
Dim x As mp_complex, y As mp_complex
n = 4
x.Real = 3.54: x.Imag = 2.66
ReDim c(0 To n - 1)
c(0).Real = 2: c(1).Real = 5: c(2).Real = 4: c(3).Real = 7
```

```
c(0).Imag = 2: c(1).Imag = 5: c(2).Imag = 4: c(3).Imag = 7
y = mp_complex_poly_complex_eval(c(0), n, x)
Debug.Print "x: ", x.Real, x.Imag, "y:", y.Real, y.Imag
End Sub
```

The output of the program is,

```
x:      3.54 + 2.66i
y:   -830.84176  + 482.955648i
```

13.2 Quadratic Equations

13.2.1 Quadratic Equation, Real Coefficients and Zeros

Function **QuadraticEquation**(***a*** *As mpNum*, ***b*** *As mpNum*, ***c*** *As mpNum*) As mpNum[]

The function QuadraticEquation returns a real vector containing the real roots of the quadratic equation.

Parameters:
a: A real number.
b: A real number.
c: A real number.

This function returns a real vector containing the real roots of the quadratic equation

$$a + bx + cx^2 = 0 \tag{13.2.1}$$

where the coefficients a, b, c are all real.
The roots are returned in ascending order. If no real roots exist, then the function returns NaN. The case of coincident roots is not considered special. For example $(x-1)^2 = 0$ will have two roots, which happen to have exactly equal values.
The number of roots found depends on the sign of the discriminant $b^2 - 4ac$. This will be subject to rounding and cancellation errors when computed in mp_real precision, and will also be subject to errors if the coefficients of the polynomial are inexact. These errors may cause a discrete change in the number of roots. However, for polynomials with small integer coefficients the discriminant can always be computed exactly.

13.2.2 Quadratic Equation, Complex Coefficients and Zeros

Function **cplxQuadraticEquation**(***a*** *As mpNum*, ***b*** *As mpNum*, ***c*** *As mpNum*) As mpNum[]

The function cplxQuadraticEquation returns a complex vector containing the complex roots of the quadratic equation.

Parameters:
a: A real or complex number.
b: A real or complex number.
c: A real or complex number.

This function returns a complex vector containing the complex roots of the quadratic equation

$$a + bz + cz^2 = 0, \tag{13.2.2}$$

where the coefficients a, b, c can be either real or complex.
The roots are returned in ascending order, sorted first by their real components and then by their imaginary components.

13.3 Cubic Equations

13.3.1 Cubic Equation, Real Coefficients and Zeros

Function **CubicEquation**(***a*** *As mpNum,* ***b*** *As mpNum,* ***c*** *As mpNum,* ***d*** *As mpNum*) As mpNum[]

The function CubicEquation returns a real vector containing the real roots of the cubic equation.

Parameters:
a: A real number.
b: A real number.
c: A real number.
d: A real number.

This function returns a real vector containing the real roots of the cubic equation

$$a + bx + cx^2 + dx^3 = 0 \tag{13.3.1}$$

where the coefficients a, b, c, d are all real.
The roots (either one or three) are returned in ascending order. The case of coincident roots is not considered special. For example, the equation $(x-1)^3 = 0$ will have three roots with exactly equal values. As in the quadratic case, finite precision may cause equal or closely-spaced real roots to move off the real axis into the complex plane, leading to a discrete change in the number of real roots.

13.3.2 Cubic Equation, Complex Coefficients and Zeros

Function **cplxCubicEquation**(***a*** *As mpNum,* ***b*** *As mpNum,* ***c*** *As mpNum,* ***d*** *As mpNum*) As mpNum[]

The function cplxCubicEquation returns a complex vector containing the complex roots of the cubic equation.

Parameters:
a: A real or complex number.
b: A real or complex number.
c: A real or complex number.
d: A real or complex number.

This function returns a complex vector containing the complex roots of the cubic equation

$$a + bz + cz^2 + dz^3 = 0, \tag{13.3.2}$$

where the coefficients a, b, c, d can be either real or complex.
The roots are returned in ascending order, sorted first by their real components and then by their imaginary components.

13.4 Quartic Equations

13.4.1 Quartic Equation, Real Coefficients and Zeros

Function **QuarticEquation**(***a*** *As mpNum*, ***b*** *As mpNum*, ***c*** *As mpNum*, ***d*** *As mpNum*, ***e*** *As mpNum*) As mpNum[]

The function QuarticEquation returns a real vector containing the real roots of the quartic equation.

Parameters:
a: A real number.
b: A real number.
c: A real number.
d: A real number.
e: A real number.

This function returns a real vector containing the real roots of the quartic equation

$$a + bx + cx^2 + dx^3 + ex^4 = 0 \tag{13.4.1}$$

where the coefficients a, b, c, d, e are all real.
The roots are returned in ascending order. If no real roots exist, then the function returns NaN.

13.4.2 Quartic Equation, Complex Coefficients and Zeros

Function **cplxQuarticEquation**(***a*** *As mpNum*, ***b*** *As mpNum*, ***c*** *As mpNum*, ***d*** *As mpNum*, ***e*** *As mpNum*) As mpNum[]

The function cplxQuarticEquation returns a complex vector containing the complex roots of the quartic equation.

Parameters:
a: A real or complex number.
b: A real or complex number.
c: A real or complex number.
d: A real or complex number.
e: A real or complex number.

This function returns a complex vector containing the complex roots of the quartic equation

$$a + bz + cz^2 + dz^3 + ez^4 = 0 \tag{13.4.2}$$

where the coefficients a, b, c, d, e can be either real or complex.
The roots are returned in ascending order, sorted first by their real components and then by their imaginary components.

13.5 General Polynomial Equations

13.5.1 General Polynomial Equation, Real Coefficients and Zeros

Function **GeneralPolynomialEquation**(***a*** *As mpNum[]*) As mpNum[]

The function GeneralPolynomialEquation returns a real vector containing the real roots of the general real polynomial.

Parameter:
a: The real coefficients of the polynomial.

This function computes the real roots of the general real polynomial

$$P(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} \tag{13.5.1}$$

using balanced-QR reduction of the companion matrix (see Edelman & Murakami (1995). The coefficient of the highest order term must be non-zero. The roots (if any) are returned as real vector.

13.5.2 General Polynomial Equation, Complex Coefficients and Zeros

Function **cplxGeneralPolynomialEquation**(*c As mpNum[]*) As mpNum[]

The function cplxGeneralPolynomialEquation returns a complex vector containing the complex roots of the general complex polynomial.

Parameter:
c: The complex coefficients of the polynomial.

This function computes the complex roots of the general complex polynomial

$$P(z) = c_0 + c_1 z + c_2 z^2 + \ldots + c_{n-1} z^{n-1} \tag{13.5.2}$$

using balanced-QR reduction of the companion matrix (see Edelman & Murakami (1995). The coefficient of the highest order term must be non-zero. The $n-1$ roots are returned as a complex vector. The function returns mp_SUCCESS if all the roots are found. If the QR reduction does not converge, the error handler is invoked with an error code of mp_EFAILED. Note that due to finite precision, roots of higher multiplicity are returned as a cluster of simple roots with reduced accuracy. The solution of polynomials with higher-order roots requires specialized algorithms that take the multiplicity structure into account (see e.g. Zeng (2004, 2005)
To demonstrate the use of the general polynomial solver we will take the polynomial $P(x) = x^5 - 1$ which has the following roots,

$$1, e^{2\pi i/5}, e^{4\pi i/5}, e^{6\pi i/5}, e^{8\pi i/5}. \tag{13.5.3}$$

```
Sub DemoComplexSolve()
Dim a() As mp_real, z() As mp_complex, w As mp_poly_complex_workspace
Dim n As Long, i As Long, status As Long
n = 6
ReDim a(0 To n - 1)
ReDim z(0 To n - 2)
a(0) = -1: a(1) = 0: a(2) = 0: a(3) = 0: a(4) = 0: a(5) = 1
w = mp_poly_complex_workspace_alloc(n)
status = mp_poly_complex_solve(a(), n, w, z())
Call mp_poly_complex_workspace_free(w)
For i = 0 To n - 2
Debug.Print z(i).Real, z(i).Imag
```

```
Next i
End Sub
```

The output of the program is

```
z0 = -0.809016994374947451 +0.587785252292473137
z1 = -0.809016994374947451 -0.587785252292473137
z2 = +0.309016994374947451 +0.951056516295153642
z3 = +0.309016994374947451 -0.951056516295153642
z4 = +1.000000000000000000 +0.000000000000000000
```

which agrees with the analytic result, $z_n = e^{2n\pi i/5}$.

Chapter 14

Fast Fourier Transform (based on Eigen)

14.1 Discrete Fourier Transforms

In this section, we provide precise mathematical definitions for the transforms that FFTW computes. These transform definitions are fairly standard, but some authors follow slightly different conventions for the normalization of the transform (the constant factor in front) and the sign of the complex exponent. We begin by presenting the one-dimensional (1d) transform definitions, and then give the straightforward extension to multi-dimensional transforms.
A good introduction is given in Arndt (2011), chapter 21.
Another Reference is Kammler (2008).
Based on the EIGEN implementation of KISSFFT.

14.1.1 1d Complex Discrete Fourier Transform (DFT)

Function **FFTW_FORWARD**(***X*** *As mpNum[]*) As mpNum[]

The function FFTW_FORWARD returns a complex vector containing the forward complex discrete Fourier transform of X.

Parameter:
X: A complex vector.

The forward (FFTW_FORWARD) discrete Fourier transform (DFT) of a 1d complex array X of size n computes an array Y, where:

$$Y_k = \sum_{j=0}^{n-1} X_j e^{-2\pi jk\sqrt{-1}/n}. \tag{14.1.1}$$

Function **FFTW_BACKWARD**(***X*** *As mpNum[]*) As mpNum[]

The function FFTW_BACKWARD returns a complex vector containing the backward complex discrete Fourier transform of X.

Parameter:
X: A complex vector.
The backward (FFTW_BACKWARD) DFT computes:

$$Y_k = \sum_{j=0}^{n-1} X_j e^{2\pi jk\sqrt{-1}/n}. \tag{14.1.2}$$

FFTW computes an unnormalized transform, in that there is no coefficient in front of the summation in the DFT. In other words, applying the forward and then the backward transform will multiply the input by n.
From above, an FFTW_FORWARD transform corresponds to a sign of -1 in the exponent of the DFT. Note also that we use the standard âĂIJin-orderâĂİ output ordering − the k-th output corresponds to the frequency k/n (or k/T, where T is your total sampling period).
For those who like to think in terms of positive and negative frequencies, this means that the positive frequencies are stored in the first half of the output and the negative frequencies are stored in backwards order in the second half of the output. (The frequency $-k/n$ is the same as the frequency $(n-k)/n$.)

14.1.2 1d Real-data DFT

Function **FFTW_R2C**(***X*** *As mpNum[]*) As mpNum[]

The function FFTW_R2C returns a complex vector containing the forward complex discrete Fourier transform of X.

Parameter:
X: A real vector.

The real-input (r2c) DFT in FFTW computes the *forward* transform Y of the size n real array X, exactly as defined above, i.e.

$$Y_k = \sum_{j=0}^{n-1} X_j e^{-2\pi jk\sqrt{-1}/n}. \tag{14.1.3}$$

This output array Y can easily be shown to possess the "Hermitian" symmetry $Y_k = Y^*_{n-k}$, where we take Y to be periodic so that $Y_n = Y_0$. As a result of this symmetry, half of the output Y is redundant (being the complex conjugate of the other half), and so the 1d r2c transforms only output elements $0 \ldots n/2$ of Y ($n/2+1$ complex numbers), where the division by 2 is rounded down. Moreover, the Hermitian symmetry implies that Y_0 and, if n is even, the $Y_{n/2}$ element, are purely real. So, for the R2HC r2r transform, these elements are not stored in the halfcomplex output format.

Function **FFTW_C2R**(***X*** *As mpNum[]*) As mpNum[]

The function FFTW_C2R returns a real vector containing the backward discrete Fourier transform of X.

Parameter:
X: A complex hermitian vector.
The c2r and H2RC r2r transforms compute the backward DFT of the *complex* array X with Hermitian symmetry, stored in the r2c/R2HC output formats, respectively, where the backward transform is defined exactly as for the complex case:

$$Y_k = \sum_{j=0}^{n-1} X_j e^{2\pi jk\sqrt{-1}/n}. \tag{14.1.4}$$

The outputs Y of this transform can easily be seen to be purely real, and are stored as an array of real numbers. Like FFTWâĂŹs complex DFT, these transforms are unnormalized. In other words, applying the real-to-complex (forward) and then the complex-to-real (backward) transform will multiply the input by n.

14.1.3 1d Real-even DFTs (DCTs)

The Real-even symmetry DFTs in FFTW are exactly equivalent to the unnormalized forward (and backward) DFTs as defined above, where the input array X of length N is purely real and is also even symmetry. In this case, the output array is likewise real and even symmetry.
For the case of REDFT00, this even symmetry means that $X_j = X_{N-j}$, where we take X to be periodic so that $X_N = X_0$. Because of this redundancy, only the first n real numbers are actually stored, where $N = 2(n-1)$.
The proper definition of even symmetry for REDFT10, REDFT01, and REDFT11 transforms is somewhat more intricate because of the shifts by 1/2 of the input and/or output. Because of the even symmetry, however, the sine terms in the DFT all cancel and the remaining cosine terms are written explicitly below. This formulation often leads people to call such a transform a discrete cosine transform (DCT), although it is really just a special case of the DFT.
In each of the definitions below, we transform a real array X of length n to a real array Y of length n:

14.1.3.1 REDFT00 (DCT-I)

Function **FFTW_REDFT00**(***X*** *As mpNum[]*) As mpNum[]

The function FFTW_REDFT00 returns a real vector containing the REDFT00 transform (type-I DCT) transform of X.

Parameter:
X: A real vector.

An REDFT00 transform (type-I DCT) in FFTW is defined by:

$$Y_k = X_0 + (-1)^k X_{n-1} + 2\sum_{j=1}^{n-2} X_j \cos[\pi jk/(n-1)]. \tag{14.1.5}$$

Note that this transform is not defined for $n = 1$. For $n = 2$, the summation term above is dropped as you might expect.

14.1.3.2 REDFT10 (DCT-II)

Function **FFTW_REDFT10**(***X*** *As mpNum[]*) As mpNum[]

The function FFTW_REDFT10 returns a real vector containing the REDFT10 transform (type-II DCT) transform of X.

Parameter:
X: A real vector.

An REDFT10 transform (type-II DCT, sometimes called "the" DCT) in FFTW is defined by:

$$Y_k = 2\sum_{j=0}^{n-1} X_j \cos[\pi(j+1/2)k/n]. \tag{14.1.6}$$

14.1.3.3 REDFT01 (DCT-III)

Function **FFTW_REDFT01**(***X*** *As mpNum[]*) As mpNum[]

The function FFTW_REDFT01 returns a real vector containing the REDFT01 transform (type-III DCT) transform of X.

Parameter:
X: A real vector.

An REDFT01 transform (type-III DCT) in FFTW is defined by:

$$Y_k = X_0 + 2\sum_{j=1}^{n-1} X_j \cos[\pi j(k+1/2)/n]. \tag{14.1.7}$$

In the case of $n = 1$, this reduces to $Y_0 = X_0$. Up to a scale factor (see below), this is the inverse of REDFT10 ("the" DCT), and so the REDFT01 (DCT-III) is sometimes called the "IDCT".

14.1.3.4 REDFT11 (DCT-IV)

Function **FFTW_REDFT11**(***X*** *As mpNum[]*) As mpNum[]

The function FFTW_REDFT11 returns a real vector containing the REDFT11 transform (type-IV DCT) transform of X.

Parameter:
X: A real vector.

An REDFT11 transform (type-IV DCT) in FFTW is defined by:

$$Y_k = 2\sum_{j=0}^{n-1} X_j \cos[\pi(j+1/2)(k+1/2)/n]. \tag{14.1.8}$$

14.1.3.5 Inverses and Normalization

These definitions correspond directly to the unnormalized DFTs used elsewhere in FFTW (hence the factors of 2 in front of the summations). The unnormalized inverse of REDFT00 is REDFT00, of REDFT10 is REDFT01 and vice versa, and of REDFT11 is REDFT11. Each unnormalized inverse results in the original array multiplied by N, where N is the logical DFT size. For REDFT00, $N = 2(n-1)$ (note that $n = 1$ is not defined); otherwise, $N = 2n$.
In defining the discrete cosine transform, some authors also include additional factors of $\sqrt{2}$ (or its inverse) multiplying selected inputs and/or outputs. This is a mostly cosmetic change that makes the transform orthogonal, but sacrifices the direct equivalence to a symmetric DFT.

14.1.4 1d Real-odd DFTs (DSTs)

The Real-odd symmetry DFTs in FFTW are exactly equivalent to the unnormalized forward (and backward) DFTs as defined above, where the input array X of length N is purely real and is also odd symmetry. In this case, the output array is odd symmetry and purely imaginary.
For the case of RODFT00, this odd symmetry means that $X_j = -X_{N-j}$, where we take X to be periodic so that $X_N = X_0$. Because of this redundancy, only the first n real numbers starting at $j = 1$ are actually stored (the $j = 0$ element is zero)„ where $N = 2(n+1)$.

The proper definition of odd symmetry for RODFT10, RODFT01, and RODFT11 transforms is somewhat more intricate because of the shifts by 1/2 of the input and/or output. Because of the odd symmetry, however, the cosine terms in the DFT all cancel and the remaining sine terms are written explicitly below. This formulation often leads people to call such a transform a discrete sine transform (DCT), although it is really just a special case of the DFT.
In each of the definitions below, we transform a real array X of length n to a real array Y of length n:

14.1.4.1 RODFT00 (DST-I)

Function **FFTW_RODFT00**(***X*** *As mpNum[]*) As mpNum[]

The function FFTW_RODFT00 returns a real vector containing the RODFT00 transform (type-I DST) transform of X.

Parameter:
X: A real vector.
An RODFT00 transform (type-I DST) in FFTW is defined by:

$$Y_k = 2\sum_{j=0}^{n-1} X_j \sin[\pi(j+1)(k+1)/(n+1)]. \tag{14.1.9}$$

14.1.4.2 RODFT10 (DST-II)

Function **FFTW_RODFT10**(***X*** *As mpNum[]*) As mpNum[]

The function FFTW_RODFT10 returns a real vector containing the RODFT10 transform (type-II DST) transform of X.

Parameter:
X: A real vector.
An RODFT10 transform (type-II DST) in FFTW is defined by:

$$Y_k = 2\sum_{j=0}^{n-1} X_j \sin[\pi(j+1/2)(k+1/2)/n]. \tag{14.1.10}$$

14.1.4.3 RODFT01 (DST-III)

Function **FFTW_RODFT01**(***X*** *As mpNum[]*) As mpNum[]

The function FFTW_RODFT01 returns a real vector containing the RODFT01 transform (type-III DST) transform of X.

Parameter:
X: A real vector.
An RODFT01 transform (type-III DST) in FFTW is defined by:

$$Y_k = (-1)^k X_{n-1} + 2\sum_{j=0}^{n-2} X_j \sin[\pi(j+1)(k+1/2)/n]. \tag{14.1.11}$$

In the case of $n = 1$, this reduces to $Y_0 = X_0$.

14.1.4.4 RODFT11 (DST-IV)

Function **FFTW_RODFT11**(***X*** *As mpNum[]*) As mpNum[]

The function FFTW_RODFT11 returns a real vector containing the RODFT11 transform (type-IV DST) transform of X.

Parameter:
X: A real vector.
An RODFT11 transform (type-IV DST) in FFTW is defined by:

$$Y_k = 2\sum_{j=0}^{n-1} X_j \sin[\pi(j+1/2)(k+1/2)/n]. \tag{14.1.12}$$

14.1.4.5 Inverses and Normalization

These definitions correspond directly to the unnormalized DFTs used elsewhere in FFTW (hence the factors of 2 in front of the summations). The unnormalized inverse of RODFT00 is RODFT00, of RODFT10 is RODFT01 and vice versa, and of RODFT11 is RODFT11. Each unnormalized inverse results in the original array multiplied by N, where N is the logical DFT size. For RODFT00, $N = 2(n+1)$; otherwise, $N = 2n$.
In defining the discrete sine transform, some authors also include additional factors of $\sqrt{2}$ (or its inverse) multiplying selected inputs and/or outputs. This is a mostly cosmetic change that makes the transform orthogonal, but sacrifices the direct equivalence to a symmetric DFT.

Part IV

Boost: Special Functions

Chapter 15

RandomNumbers

Random numbers are required in a number of different problem domains, such as

- numerics (simulation, Monte-Carlo integration)
- games (non-deterministic enemy behavior)
- security (key generation)
- testing (random coverage in white-box tests)

The Boost Random Number Generator Library provides a framework for random number generators with well-defined properties so that the generators can be used in the demanding numerics and security domains. For a general introduction to random numbers in numerics, see Press *et al.* (2007), Chapter 7.
Depending on the requirements of the problem domain, different variations of random number generators are appropriate:
This is based on the Boost Random Library Maurer & Watanabe (2013).

15.1 Definitions

15.1.1 Random Device

Class random_device models a non-deterministic random number generator . It uses one or more implementation-defined stochastic processes to generate a sequence of uniformly distributed non-deterministic random numbers. For those environments where a non-deterministic random number generator is not available, class random_device must not be implemented. See Eastlake *et al.* (1994) for further discussions.
Implementation Note for Windows: On the Windows operating system, token is interpreted as the name of a cryptographic service provider. By default random_device uses MS_DEF_PROV.

15.1.2 Uniform Random Number Generator

A uniform random number generator is a NumberGenerator that provides a sequence of random numbers uniformly distributed on a given range. The range can be compile-time fixed or available (only) after run-time construction of the object. The tight lower bound of some (finite) set S is the (unique) member l in S, so that for all v in S, $l \leq v$ holds. Likewise, the tight upper bound of some (finite) set S is the (unique) member u in S, so that for all v in S, $v \leq u$ holds.

For integer generators (i.e. integer T), the generated values x fulfill $\min() \leq x \leq \max()$, for non-integer generators (i.e. non-integer T), the generated values x fulfill $\min() \leq x < \max()$.

15.1.3 Pseudo-Random Number Generator

A pseudo-random number generator is a UniformRandomNumberGenerator which provides a deterministic sequence of pseudo-random numbers, based on some algorithm and internal state. Linear congruential and inversive congruential generators are examples of such pseudo-random number generators. Often, these generators are very sensitive to their parameters. In order to prevent wrong implementations from being used, an external testsuite should check that the generated sequence and the validation value provided do indeed match. Knuth (1997) gives an extensive overview on pseudo-random number generation. The descriptions for the specific generators contain additional references.

15.2 The Random Number Generator Interface

15.2.1 Sampling

This is a place holder reference for Excel Sampling.

15.3 Random number generator algorithms

This library provides several pseudo-random number generators. The quality of a pseudo random number generator crucially depends on both the algorithm and its parameters. This library implements the algorithms as class templates with template value parameters, hidden in namespace boost::random. Any particular choice of parameters is represented as the appropriately specializing typedef in namespace boost.
Pseudo-random number generators should not be constructed (initialized) frequently during program execution, for two reasons. First, initialization requires full initialization of the internal state of the generator. Thus, generators with a lot of internal state (see below) are costly to initialize. Second, initialization always requires some value used as a ”seed” for the generated sequence. It is usually difficult to obtain several good seed values. For example, one method to obtain a seed is to determine the current time at the highest resolution available, e.g. microseconds or nanoseconds. When the pseudo- random number generator is initialized again with the then-current time as the seed, it is likely that this is at a near- constant (non-random) distance from the time given as the seed for first initialization. The distance could even be zero if the resolution of the clock is low, thus the generator re-iterates the same sequence of random numbers. For some applications, this is inappropriate.

Function **SaveDefaultRngState**(***FName*** *As String*) As Boolean

---

The function SaveDefaultRngState returns a boolean value: TRUE if the state was successfully save, FALSE otherwise

Parameter:
FName: A String, containing the full path of the file.

Function **LoadDefaultRngState**(***FName*** *As String*) As Boolean

---

The function LoadDefaultRngState returns a boolean value: TRUE if the state was successfully loaded, FALSE otherwise

Parameter:
FName: A String, containing the full path of the file.

Note that all pseudo-random number generators described below are CopyConstructible and Assignable. Copying or assigning a generator will copy all its internal state, so the original and the copy will generate the identical sequence of random numbers. Often, such behavior is not wanted.
The following table gives an overview of some characteristics of the generators. The cycle length is a rough estimate of the quality of the generator; the approximate relative speed is a performance measure, higher numbers mean faster random number generation.

15.3.1 Minimal Standard

The specialization minstd_rand0 was originally suggested in Lewis *et al.* (1969)
It is examined more closely together with minstd_rand in Park & Miller (1988).
The specialization minstd_rand was suggested in Park & Miller (1988).

15.3.2 rand48

Class rand48 models a pseudo-random number generator . It uses the linear congruential algorithm with the parameters a = 0x5DEECE66D, c = 0xB, m = 2**48. It delivers identical results to the lrand48() function available on some systems (assuming lcong48 has not been called).
It is only available on systems where uint64_t is provided as an integral type, so that for example static in-class constants and/or enum definitions with large uint64_t numbers work.

15.3.3 Ecuyer 1988

The specialization ecuyer1988 was suggested in L'Ecuyer (1988)

15.3.4 Knuth b

The specialization knuth_b is specified by the C++ standard. It is described in Knuth (1981)

15.3.5 Kreutzer 1986

the specialization kreutzer1986 was suggested in Kreutzer (1986)

15.3.6 Tauss 88

The specialization taus88 was suggested in L'Ecuyer (1996)

15.3.7 Hellekalek 1995

The specialization hellekalek1995 was suggested in Hellekalek (1995)

15.3.8 Mersenne-Twister 11213b

The specializations mt11213b and mt19937 are from Matsumoto & Nishimura (1998)

15.3.9 Mersenne-Twister 19937

The specializations mt11213b and mt19937 are from Matsumoto & Nishimura (1998)

15.3.10 Mersenne-Twister 19937 64

The specializations mt11213b and mt19937 are from Matsumoto & Nishimura (1998). adapted for 64 bit. The recursion is similar but different, so the output is totally different from the 32-bit versions.

15.3.11 Lagged Fibonacci Generators

The specializations lagged_fibonacci607 ... lagged_fibonacci44497 use well tested lags. See Brent (1992a)
The lags used here can be found in Brent (1992b).

15.3.12 Ranlux Generators

The ranlux family of generators are described in Luescher (1994).
The levels are given in James (1994).

15.4 Random number distributions

15.4.1 Uniform, small integer

discrete uniform distribution on a small set of integers (much smaller than the range of the underlying generator) .
The distribution function uniform_smallint models a random distribution . On each invocation, it returns a random integer value uniformly distributed in the set of integer numbers $\{min, min + 1, min + 2, ..., max\}$. It assumes that the desired range $(max - min + 1)$ is small compared to the range of the underlying source of random numbers and thus makes no attempt to limit quantization errors.
Let $r_{out} = (max - min + 1)$ be the desired range of integer numbers, and let r_{base} be the range of the underlying source of random numbers. Then, for the uniform distribution, the theoretical probability for any number i in the range r_{out} will be $p_{out} = \frac{1}{r_{out}}$. Likewise, assume a uniform distribution on for the underlying source of random numbers, i.e. . Let denote the random distribution generated by uniform_smallint. Then the sum over all i in of shall not exceed .
The template parameter IntType shall denote an integer-like value type.
[Note] Note
The property above is the square sum of the relative differences in probabilities between the desired uniform distribution and the generated distribution . The property can be fulfilled with the calculation , as follows: Let . The base distribution on is folded onto the range . The numbers $i < r$ have assigned numbers of the base distribution, the rest has only . Therefore, for $i < r$ and otherwise. Substituting this in the above sum formula leads to the desired result.

Note: The upper bound for is . Regarding the upper bound for the square sum of the relative quantization error of , it seems wise to either choose so that or ensure that is divisible by .

15.4.2 Uniform, integer

discrete uniform distribution on a set of integers; the underlying generator may be called several times to gather enough randomness for the output.
The class template uniform_int_distribution models a random distribution . On each invocation, it returns a random integer value uniformly distributed in the set of integers $\{min, min+1, min+2, ..., max\}$.
The template parameter IntType shall denote an integer-like value type.

15.4.3 Uniform, 01

continuous uniform distribution on the range $[0, 1)$; important basis for other distributions.
The distribution function uniform_01 models a random distribution . On each invocation, it returns a random floating-point value uniformly distributed in the range [0..1).
The template parameter RealType shall denote a float-like value type with support for binary operators +, −, and /.
Note: The current implementation is buggy, because it may not fill all of the mantissa with random bits. I'm unsure how to fill a (to-be-invented) boost::bigfloat class with random bits efficiently. It's probably time for a traits class.

15.4.4 Uniform, Real

continuous uniform distribution on some range [min, max) of real numbers
The class template uniform_real_distribution models a random distribution . On each invocation, it returns a random floating-point value uniformly distributed in the range $[min..max)$.

15.4.5 Discrete

discrete distribution with specific probabilities (rolling an unfair dice).
The class discrete_distribution models a random distribution . It produces integers in the range $[0, n)$ with the probability of producing each value is specified by the parameters of the distribution.
Constructs a discrete_distribution from a std::initializer_list. If the initializer_list is empty, equivalent to the default constructor. Otherwise, the values of the initializer_list represent weights for the possible values of the distribution. For example, given the distribution

```
* discrete_distribution<> dist{1, 4, 5};
*
```

The probability of a 0 is 1/10, the probability of a 1 is 2/5, the probability of a 2 is 1/2, and no other values are possible.

15.4.6 Piecewise constant

Constructs a piecewise_constant_distribution from two iterator ranges containing the interval boundaries and the interval weights. If there are less than two boundaries, then this is equivalent to the default constructor and creates a single interval, $[0, 1)$.

The values of the interval boundaries must be strictly increasing, and the number of weights must be one less than the number of interval boundaries. If there are extra weights, they are ignored. For example,

```
* double intervals[] = { 0.0, 1.0, 4.0 };
* double weights[] = { 1.0, 1.0 };
* piecewise_constant_distribution<> dist(
*     &intervals[0], &intervals[0] + 3, &weights[0]);
```

The distribution has a 50% chance of producing a value between 0 and 1 and a 50% chance of producing a value between 1 and 4.

15.4.7 Piecewise linear

Constructs a piecewise_linear_distribution from two iterator ranges containing the interval boundaries and the weights at the boundaries. If there are fewer than two boundaries, then this is equivalent to the default constructor and creates a distribution that produces values uniformly distributed in the range [0, 1).
The values of the interval boundaries must be strictly increasing, and the number of weights must be equal to the number of interval boundaries. If there are extra weights, they are ignored.
For example,

```
* double intervals[] = { 0.0, 1.0, 2.0 };
* double weights[] = { 0.0, 1.0, 0.0 };
* piecewise_constant_distribution<> dist(
*     &intervals[0], &intervals[0] + 3, &weights[0]);
*
```

produces a triangle distribution.

15.4.8 Triangle

Instantiations of triangle_distribution model a random distribution.
A triangle_distribution has three parameters, a, b, and c, which are the smallest, the most probable and the largest values of the distribution respectively.

15.4.9 Uniform on Sphere

Instantiations of class template uniform_on_sphere model a random distribution . Such a distribution produces random numbers uniformly distributed on the unit sphere of arbitrary dimension dim.

Chapter 16

Special Functions (based on Boost)

Boost: Maddock & Kormanyos (2013)
The standard referencea are Olver *et al.* (2010), and Temme (1996), and Press *et al.* (2007)
See also Gil *et al.* (2007) and Gil *et al.* (2011)
See also Cuyt *et al.* (2008)

16.1 Gamma and Beta Functions

Detailed review of the gamma function can be found in Pugh (2004) and Luschny (2012).
The implementation is based on Bristow *et al.* (2013)

16.1.1 Gamma function $\Gamma(x)$

Function **TgammaBoost**(***x*** *As mpNum*) As mpNum

The function TgammaBoost returns the gamma function for $x \neq 0, -1, -2, \ldots$.

Parameter:
x: A real number.

The gamma function for $x \neq 0, -1, -2, \ldots$ is defined by

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt \quad (x > 0), \tag{16.1.1}$$

and by analytic continuation if $x < 0$, using the reflection formula

$$\Gamma(x)\Gamma(1-x) = \pi / \sin(\pi x). \tag{16.1.2}$$

16.1.2 Logarithm of $\Gamma(x)$

Function **LgammaBoost**(***x*** *As mpNum*) As mpNum

The function LgammaBoost returns the logarithm of the gamma function.

Parameter:
x: A real number.

This function computes $\ln|\Gamma(x)|$ for $x \neq 0, -1, -2, \ldots$. If $x < 0$ the function uses the logarithm of the reflection formula.

16.1.3 Auxiliary function $\Gamma(x)/\Gamma(x+\delta)$

Function **TgammaDeltaRatioBoost**(***x*** *As mpNum*, δ *As mpNum*) As mpNum

The function TgammaDeltaRatioBoost returns the ratio of gamma functions.

Parameters:
x: A real number.
δ: A real number.

This functions returns the ratio of gamma functions in the form

$$\frac{\Gamma(a)}{\Gamma(a+\delta)} \tag{16.1.3}$$

Note that the result is calculated accurately even when δ is small compared to a: indeed even if $a+\delta \approx a$. The function is typically used when a is large and δ is very small.

16.1.4 Digamma function $\psi(x)$

Function **DigammaBoost**(***x*** *As mpNum*) As mpNum

The function DigammaBoost returns the digamma function for $x \neq 0, -1, -2, \ldots$.

Parameter:
x: A real number.

The digamma or ψ function is defined as

$$\psi(x) = \frac{d(\ln\Gamma(x))}{dx} = \frac{\Gamma'(x)}{\Gamma(x)}, \quad x \neq 0, -1, -2, \ldots \tag{16.1.4}$$

If $x < 0$ it is transformed to positive values with the reflection formula

$$\psi(1-x) = \psi(x) + \pi\cot(\pi x) \tag{16.1.5}$$

and for $0 < x < 12$ the recurrence formula

$$\psi(x+1) = \psi(x) + \frac{1}{x} \tag{16.1.6}$$

16.1.5 Ratio of Gamma Functions

Function **TgammaratioBoost**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum

The function TgammaratioBoost returns the ratio of gamma functions.

Parameters:
a: A real number.
b: A real number.

This functions returns the ratio of gamma functions in the form

$$\frac{\Gamma(a)}{\Gamma(b)} \tag{16.1.7}$$

16.1.6 Normalised incomplete gamma functions

Function **GammaPBoost**(***a*** *As mpNum*, ***x*** *As mpNum*) As mpNum

The function GammaPBoost returns the normalised incomplete gamma function $P(a, x)$.

Parameters:
a: A real number.
x: A real number.

The normalised incomplete gamma function $P(a, x)$ is defined as

$$P(a, x) = \frac{1}{\Gamma(a)} \int_0^x t^{a-1} e^{-t} dt \tag{16.1.8}$$

for $a \geq 0$ and $x \geq 0$.

Function **GammaQBoost**(***a*** *As mpNum*, ***x*** *As mpNum*) As mpNum

The function GammaQBoost returns the normalised incomplete gamma function $Q(a, x)$.

Parameters:
a: A real number.
x: A real number.
Boost references are Temme (1979) and Temme (1994)

The normalised incomplete gamma function $Q(a, x)$ is defined as

$$Q(a, x) = \frac{1}{\Gamma(a)} \int_x^\infty t^{a-1} e^{-t} dt \tag{16.1.9}$$

for $a \geq 0$ and $x \geq 0$.

16.1.7 Non-Normalised incomplete gamma functions

Function **NonNormalisedGammaPBoost**(***a*** *As mpNum*, ***x*** *As mpNum*) As mpNum

The function NonNormalisedGammaPBoost returns the non-normalised incomplete gamma function $\Gamma(a, x)$.

Parameters:
a: A real number.
x: A real number.

The non-normalised incomplete gamma function $\Gamma(a, x)$ is defined as

$$\Gamma(a, x) = \int_0^x t^{a-1} e^{-t} dt \tag{16.1.10}$$

for $a \geq 0$ and $x \geq 0$.

Function **NonNormalisedGammaQBoost**(***a*** *As mpNum*, ***x*** *As mpNum*) As mpNum

The function NonNormalisedGammaQBoost returns the non-normalised incomplete gamma function $\gamma(a, x)$.

Parameters:
a: A real number.
x: A real number.

The non-normalised incomplete gamma function $\gamma(a, x)$ is defined as

$$\gamma(a, x) = \int_x^\infty t^{a-1} e^{-t} dt \tag{16.1.11}$$

for $a \geq 0$ and $x \geq 0$.
Note: in Boost, the functions are referred to as TgammaLower and TgammaUpper.

16.1.8 Inverse normalised incomplete gamma functions

Function **GammaPinvBoost**(***a*** *As mpNum,* ***p*** *As mpNum*) As mpNum

The function GammaPinvBoost returns the inverse of the normalised incomplete gamma function $P(a, x)$.

Parameters:
a: A real number.
p: A real number.

This function returns the inverse normalised incomplete gamma function, i.e. it calculates x with $P(a, x) = p$. The input parameters are $a > 0$, $p \geq 0$, and $p + q = 1$.

Function **GammaQinvBoost**(***a*** *As mpNum,* ***q*** *As mpNum*) As mpNum

The function GammaQinvBoost returns the inverse of the normalised incomplete gamma function $Q(a, x)$.

Parameters:
a: A real number.
q: A real number.

This function returns the inverse normalised incomplete gamma function, i.e. it calculates x with $Q(a, x) = q$. The input parameters are $a > 0$, $q \geq 0$, and $p + q = 1$.

Function **GammaPinvaBoost**(***x*** *As mpNum,* ***p*** *As mpNum*) As mpNum

The function GammaPinvaBoost returns the parameter a of the normalised incomplete gamma function $P(a, x)$, such that $P(a, x) = p$.

Parameters:
x: A real number.
p: A real number.

Function **GammaQinvaBoost**(***x*** *As mpNum,* ***q*** *As mpNum*) As mpNum

The function GammaQinvaBoost returns the parameter a of the normalised incomplete gamma function $Q(a, x)$, such that $Q(a, x) = q$.

Parameters:
x: A real number.
q: A real number.

16.1.9 Derivative of the normalised incomplete gamma function

Function **GammaPDerivativeBoost**(***a*** *As mpNum*, ***x*** *As mpNum*) As mpNum

The function GammaPDerivativeBoost returns the partial derivative with respect to x of the incomplete gamma function $P(a, x)$.

Parameters:
a: A real number.
x: A real number.

The partial derivative with respect to x of the incomplete gamma function $P(a, x)$ is defined as:

$$\frac{\partial}{\partial x} P(a, x) = \frac{e^{-x} x^{a-1}}{\Gamma(a)}. \tag{16.1.12}$$

16.2 Factorials and Binomial Coefficient

16.2.1 Factorial

Function **FactorialBoost**(***n*** *As mpNum*) As mpNum

The function FactorialBoost returns the factorial $n! = \Gamma(n + 1) = n \times (n - 1) \times \cdots \times 1$.

Parameter:
n: An integer.

16.2.2 Double Factorial

Function **DoubleFactorialBoost**(***n*** *As mpNum*) As mpNum

The function DoubleFactorialBoost returns the double factorial $n!!$.

Parameter:
n: An integer.

For even $n < 0$ the result is ∞. For positive n the double factorial is defined as

$$n!! = \begin{cases} 1 \cdot 3 \cdot 5 \cdots n & \text{if } n \text{ is odd.} \\ 2 \cdot 4 \cdot 6 \cdots n & \text{if } n \text{ is even.} \end{cases} \tag{16.2.1}$$

16.2.3 Rising Factorial

Function **RisingFactorialBoost**(***n*** *As mpNum*, ***i*** *As mpNum*) As mpNum

The function RisingFactorialBoost returns the rising factorial of x and i.

Parameters:

n: An integer.
i: An integer.

Returns the rising factorial of x and i:

$$\text{RisingFactorial}(n, i) = n(n+1)(n+2)(n+3)\cdots(n+i-1) \tag{16.2.2}$$

or

$$(n)_i = \frac{\Gamma(n+i)}{\Gamma(n)}. \tag{16.2.3}$$

Note that both n and i can be negative as well as positive.

16.2.4 Falling Factorial

Function **FallingFactorialBoost**(***n*** *As mpNum*, ***i*** *As mpNum*) As mpNum

The function FallingFactorialBoost returns the falling factorial of x and i.

Parameters:
n: An integer.
i: An integer.

The falling factorial of x and i is defined as:

$$\text{FallingFactorial}(n, i) = n(n-1)(n-2)(n-3)...(n-i+1) \tag{16.2.4}$$

Note that this function is only defined for positive i, hence the unsigned second argument. Argument n can be either positive or negative however.

16.2.5 Binomial coefficient

Function **BinomialCoefficientBoost**(***n*** *As mpNum*, ***k*** *As mpNum*) As mpNum

The function BinomialCoefficientBoost returns the binomial coefficient.

Parameters:
n: An integer.
k: An integer.

The binomial coefficient ("n choose k") is defined as

$$\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k(k-1)\cdots(1)} \tag{16.2.5}$$

for $k \geq 0$.

16.3 Beta Functions

16.3.1 Beta function B(a, b)

Function **BetaBoost**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNum

The function BetaBoost returns the beta function.

Parameters:
a: A real number.
b: A real number.

The beta function is defined as

$$B(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} \tag{16.3.1}$$

where $\Gamma(\cdot)$ denotes the Gamma function (see section 16.1). The reference is DiDonato & Morris (1992)

16.4 Error Function and Related Functions

16.4.1 Error Function erf

Function **ErfBoost**(*x As mpNum*) As mpNum

The function ErfBoost returns the value of the error function.

Parameter:
x: A real number.

The error function is defined by

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-x^2} dt, \tag{16.4.1}$$

16.4.2 Complementary Error Function

Function **ErfcBoost**(*x As mpNum*) As mpNum

The function ErfcBoost returns the value of the complementary error function.

Parameter:
x: A real number.

16.4.3 Inverse Function of erf

Function **ErfInvBoost**(*x As mpNum*) As mpNum

The function ErfInvBoost returns the functional inverse of $\mathrm{erf}(x)$

Parameter:
x: A real number.

Tthe functional inverse of $\mathrm{erf}(x)$ is defined by

$$\mathrm{erf}(\mathrm{erf\_inv}(x)) = x, \quad -1 < x < 1. \tag{16.4.2}$$

16.4.4 Inverse Function of erfc

Function **ErfcInvBoost**(***x*** *As mpNum*) As mpNum

The function ErfcInvBoost returns the functional inverse of erfc(x)

Parameter:
x: A real number.

The functional inverse of erfc(x) is defined by

$$\text{erfc}(\text{erfc\_inv}(x)) = x, \quad 0 < x < 2. \tag{16.4.3}$$

16.5 Polynomials

16.5.1 Legendre Polynomials/Functions

Function **LegendrePBoost**(***l*** *As mpNum*, ***x*** *As mpNum*) As mpNum

The function LegendrePBoost returns $P_l(x)$, the Legendre functions of the first kind.

Parameters:
l: An Integer.
x: A real number.

These functions return $P_l(x)$, the Legendre functions of the first kind, also called Legendre polynomials if $l \geq 0$ and $|x| \leq 1$. The Legendre polynomials are orthogonal on the interval (âĹŠ1, 1) with $w(x) = 1$. If $l \geq 0$ the function uses the recurrence relation for varying degree from [1, 8.5.3]:

$$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ (l+1)P_{l+1}(x) &= (2l+1)P_l(x) - lP_{l-1}(x). \end{aligned} \tag{16.5.1}$$

and for negative l the result is $P_l(x) = P_{-l-1}(x)$.

Function **LegendrePNextBoost**(***l*** *As mpNum*, ***x*** *As mpNum*, ***Pl*** *As mpNum*, ***Plm1*** *As mpNum*) As mpNum

The function LegendrePNextBoost returns the Legendre function of the first kind of degree $l + 1$, using the results for degree l and $l - 1$.

Parameters:
l: An Integer. The degree of the last polynomial calculated.
x: A real number. The abscissa value.
Pl: A real number. The value of the polynomial evaluated at degree l.
Plm1: A real number. The value of the polynomial evaluated at degree $l - 1$.

This function implements the recursion relation given in equation 16.5.1

16.5.2 Associated Legendre Polynomials/Functions

Function **AssociatedLegendrePlmBoost**(***l*** *As mpNum*, ***m*** *As mpNum*, ***x*** *As mpNum*) As mpNum

The function AssociatedLegendrePlmBoost returns $L_n^m(x)$, the associated Legendre polynomials of degree $l \geq 0$ and order $m \geq 0$.

Parameters:
l: An Integer.
m: An Integer.
x: A real number.

This function returns $L_n^m(x)$, the associated Legendre polynomials of degree $l \geq 0$ and order $m \geq 0$, defined for $m > 0, |x| < 1$ as

$$P_l^m(x) = (-1)^m(1-x^2)^{m/2}\frac{d^m}{dx^m}P_l(x). \tag{16.5.2}$$

The following recursion relation holds:

$$(l-m+1)P_{l+1}^m(x) = (2l+1)xP_l^m(x) - (l+m+1)P_{l-1}^m(x). \tag{16.5.3}$$

Function **AssociatedLegendrePlmNextBoost**(***l*** *As mpNum*, ***m*** *As mpNum*, ***x*** *As mpNum*, ***Pl*** *As mpNum*, ***Plm1*** *As mpNum*) As mpNum

---

The function AssociatedLegendrePlmNextBoost returns the Legendre function of the first kind of degree $l+1$, using the results for degree l and $l-1$.

Parameters:
l: An Integer. The degree of the last polynomial calculated.
m: An Integer. The order of the Associated Polynomial.
x: A real number. The abscissa value.
Pl: A real number. The value of the polynomial evaluated at degree l.
Plm1: A real number. The value of the polynomial evaluated at degree $l-1$.

This function implements the recursion relation given in equation 16.5.3

16.5.3 Legendre Functions of the Second Kind

Function **LegendreQBoost**(***l*** *As mpNum*, ***x*** *As mpNum*) As mpNum

---

The function LegendreQBoost returns $Q_l(x)$, the Legendre functions of the second kind of degree $l \geq 0$ and $|x| \neq 1$.

Parameters:
l: An Integer.
x: A real number.

These functions return $Q_l(x)$, the Legendre functions of the second kind of degree $l \geq 0$ and $|x| \neq 1$, defined as

$$\begin{aligned} Q_0(x) &= \frac{1}{2}\ln\frac{1+x}{1-x} \\ Q_1(x) &= \frac{x}{2}\ln\frac{1+x}{1-x} - 1 \\ (k+1)Q_{k+1}(x) &= (2k+1)xQ_k(x) - kQ_{k-1}(x). \end{aligned} \tag{16.5.4}$$

16.5.4 Laguerre Polynomials

Function **LaguerreLBoost**(***n*** *As mpNum*, ***x*** *As mpNum*) As mpNum

---

The function LaguerreLBoost returns $L_n(x)$, the Laguerre polynomials of degree $n \geq 0$.

Parameters:
n: An Integer.
x: A real number.

This function returns $L_n(x)$, the Laguerre polynomials of degree $n \geq 0$. The Laguerre polynomials are just special cases of the generalized Laguerre polynomials

$$L_n(x) = L_n^{(0)}(x). \tag{16.5.5}$$

The standard recurrence formulas are used:

$$\begin{aligned} L_0(x) &= 1 \\ L_1(x) &= -x + 1 \\ nLn(x) &= (2n - 1 - x)L_{n-1}(x) - (n-1)L_{n-2}(x). \end{aligned} \tag{16.5.6}$$

Function **LaguerreLNextBoost**(***n*** *As mpNum*, ***x*** *As mpNum*, ***Ln*** *As mpNum*, ***Lnm1*** *As mpNum*) As mpNum

---

The function LaguerreLNextBoost returns the Laguerre polynomial of the first kind of degree $n+1$, using the results for degree n and $n-1$.

Parameters:
n: An Integer. The degree of the last polynomial calculated.
x: A real number. The abscissa value.
Ln: A real number. The value of the polynomial evaluated at degree n.
Lnm1: A real number. The value of the polynomial evaluated at degree $n-1$.

This function implements the recursion relation given in equation 16.5.6

16.5.5 Associated Laguerre Polynomials

Function **AssociatedLaguerreBoost**(***n*** *As mpNum*, ***m*** *As mpNum*, ***x*** *As mpNum*) As mpNum

---

The function AssociatedLaguerreBoost returns $L_n^m(x)$, the associated Laguerre polynomials of degree $n \geq 0$ and order $m \geq 0$.

Parameters:
n: An Integer.
m: An Integer.
x: A real number.

This function returns $L_n^m(x)$, the associated Laguerre polynomials of degree $n \geq 0$ and order $m \geq 0$, defined as

$$L_n^m(x) = (-1)^m \frac{d^m}{dx^m} L_{n+m}(x). \tag{16.5.7}$$

The standard recurrence formulas are used:

$$\begin{aligned} L_0^a(x) &= 1 \\ L_1^a(x) &= -x + 1 + a \\ nL_n^a(x) &= (2n + a - 1 - x)L_{n-1}^a(x) - (n + a - 1)L_{n-2}^a(x). \end{aligned} \tag{16.5.8}$$

Function **AssociatedLaguerreLNextBoost**(***n*** *As mpNum*, ***m*** *As mpNum*, ***x*** *As mpNum*, ***Ln*** *As mpNum*, ***Lnm1*** *As mpNum*) As mpNum

---

The function AssociatedLaguerreLNextBoost returns the associated Laguerre polynomial of the first kind of degree $n+1$, using the results for degree n and $n-1$.

Parameters:

n: An Integer. The degree of the last polynomial calculated.
m: An Integer. The order of the Associated Polynomial.
x: A real number. The abscissa value.
Ln: A real number. The value of the polynomial evaluated at degree n.
Lnm1: A real number. The value of the polynomial evaluated at degree $n-1$.

This function implements the recursion relation given in equation 16.5.8

16.5.6 Hermite Polynomials

Function **HermiteHBoost**(***n*** *As mpNum*, ***x*** *As mpNum*) As mpNum

The function HermiteHBoost returns $H_n(x)$, the Hermite polynomial of degree $n \geq 0$.

Parameters:
n: An Integer.
x: A real number.

These functions return $H_n(x)$, the Hermite polynomial of degree $n \geq 0$. The H_n are orthogonal on the interval $(-\infty, \infty)$, with respect to the weight function $w(x) = e^{-x^2}$. They are computed with the standard recurrence formulas [1, 22.7.13]:

$$\begin{aligned} H_0(x) &= 1 \\ H_1(x) &= 2x \\ H_n(x) &= 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x). \end{aligned} \tag{16.5.9}$$

Function **HermiteHNextBoost**(***n*** *As mpNum*, ***x*** *As mpNum*, ***Hn*** *As mpNum*, ***Hnm1*** *As mpNum*) As mpNum

The function HermiteHNextBoost returns the Hermite polynomial of degree $n+1$, using the results for degree n and $n-1$.

Parameters:
n: An Integer. The degree of the last polynomial calculated.
x: A real number. The abscissa value.
Hn: A real number. The value of the polynomial evaluated at degree n.
Hnm1: A real number. The value of the polynomial evaluated at degree $n-1$.

This function implements the recursion relation given in equation 16.5.9

16.5.7 Spherical Harmonic Functions

Function **SphericalHarmonicBoost**(***l*** *As mpNumList[2]*, ***m*** *As mpNum*, θ *As mpNum*, ϕ *As mpNum*) As mpNum

The function SphericalHarmonicBoost returns the real and imaginary parts of the spherical harmonic function $Y_{lm}(\theta, \phi)$.

Parameters:
l: An Integer.

m: An Integer.
θ: A real number.
ϕ: A real number.

The procedures return the real and imaginary parts of the spherical harmonic function $Y_{lm}(\theta, \phi)$. These functions are closely related to the associated Legendre polynomials:

$$Y_{lm}(\theta, \phi) = \sqrt{\frac{(2l+1)(l-m)!}{4\pi(l+m)!}} P_l^m(\cos(\theta)) e^{im\phi} \tag{16.5.10}$$

16.6 Bessel Functions of Real Order

16.6.1 Bessel Function $J_\nu(x)$

Function **BesselJBoost**(***x*** *As mpNum*, ν *As mpNum*) As mpNum

The function BesselJBoost returns $J_\nu(z)$, the Bessel function of the first kind of real order ν.

Parameters:
x: A real number.
ν: A real number.

$J_\nu(z)$, the Bessel function of the first kind of order ν, is defined as

$$J_\nu(x) = \left(\tfrac{1}{2}x\right)^\nu \sum_{k=0}^{\infty} (-1)^k \frac{(x^2/4)^k}{k!\Gamma(\nu + k + 1)} \tag{16.6.1}$$

16.6.2 Bessel Function $Y_\nu(x)$

Function **BesselYBoost**(***x*** *As mpNum*, ν *As mpNum*) As mpNum

The function BesselYBoost returns $Y_\nu(z)$, the Bessel function of the second kind of order ν.

Parameters:
x: A real number.
ν: A real number.

$Y_\nu(z)$, the Bessel function of the second kind of order ν, is defined as

$$Y_\nu(x) = \frac{J_\nu(x)\cos(\nu\pi) - J_{-\nu}(x)}{\sin(\nu\pi)} \tag{16.6.2}$$

16.7 Modified Bessel Functions of Real Order

16.7.1 Bessel Function $I_\nu(x)$

Function **BesselIBoost**(***x*** *As mpNum*, ν *As mpNum*) As mpNum

The function BesselIBoost returns the modified Bessel function $I_\nu(z)$ of the first kind of order ν.

Parameters:

x: A real number.
ν: A real number.

This function returns the modified Bessel function $I_\nu(z)$ of the first kind of order ν, defined as

$$I_\nu(z) = \frac{z}{2}\sum_{j=0}^{\infty}\frac{(z^2/4)^j}{j!\Gamma(\nu+j+1)} \tag{16.7.1}$$

16.7.2 Bessel Function $K_\nu(x)$

Function **BesselKBoost**(***x*** *As mpNum, ν As mpNum*) As mpNum

The function BesselKBoost returns $K_\nu(x)$, the modified Bessel function of the second kind of order ν.

Parameters:
x: A real number.
ν: A real number.

This function returns $K_\nu(z)$, the modified Bessel function of the second kind of order ν, defined as

$$K_\nu(x) = \frac{\pi}{2}\frac{I_{-\nu}(x)) - I_\nu(x)}{\sin(\nu\pi)} \tag{16.7.2}$$

16.8 Spherical Bessel Functions

16.8.1 Spherical Bessel function $j_n(x)$

Function **BesselSphericaljBoost**(***x*** *As mpNum, ν As mpNum*) As mpNum

The function BesselSphericaljBoost returns $j_n(x)$, the spherical Bessel function of the 1st kind, order n.

Parameters:
x: A real number.
ν: A real number.

The function $j_n(x)$, the spherical Bessel function of the 1st kind, order n, is defined as

$$j_n(x) = \sqrt{\tfrac{1}{2}\pi/x}J_{n+\frac{1}{2}}(x), \quad (x \le 0), \quad \text{and } j_n(-x) = (-1)^n j_n(x). \tag{16.8.1}$$

16.8.2 Spherical Bessel function $y_n(x)$

Function **BesselSphericalyBoost**(***x*** *As mpNum, ν As mpNum*) As mpNum

The function BesselSphericalyBoost returns $j_n(x)$, the spherical Bessel function of the 1st kind, order n.

Parameters:
x: A real number.
ν: A real number.

The function $y_n(x)$, the spherical Bessel function of the second kind, order n, $x \neq 0$, is defined as

$$y_n(x) = \sqrt{\tfrac{1}{2}\pi/x}Y_{n+\frac{1}{2}}(x), \quad (x > 0), \quad \text{and } y_n(-x) = (-1)^{n+1}y_n(x). \tag{16.8.2}$$

16.9 Hankel Functions

16.9.1 Hankel Function of the First Kind

Function **cplxHankel1Boost**(***x*** *As mpNum, ν As mpNum*) As mpNum

The function cplxHankel1Boost returns the Hankel function of the first kind $H_\nu^{(1)}(x)$.

Parameters:
x: A real number.
ν: A real number.

This routine returns the Hankel function of the first kind $H_\nu^{(1)}(x)$, defined as

$$H_\nu^{(1)}(x) = J_\nu(x) + iY_\nu(x). \tag{16.9.1}$$

16.9.2 Hankel Function of the Second Kind

Function **cplxHankel2Boost**(***x*** *As mpNum, ν As mpNum*) As mpNum

The function cplxHankel2Boost returns the Hankel function of the second kind $H_\nu^{(2)}(x)$.

Parameters:
x: A real number.
ν: A real number.

This routine returns the Hankel function of the second kind $H_\nu^{(2)}(x)$, defined as

$$H_\nu^{(2)}(x) = J_\nu(x) - iY_\nu(x). \tag{16.9.2}$$

16.9.3 Spherical Hankel Function of the First Kind

Function **cplxHankelSph1Boost**(***x*** *As mpNum, ν As mpNum*) As mpNum

The function cplxHankelSph1Boost returns the spherical Hankel function of the first kind $h_\nu^{(1)}(x)$.

Parameters:
x: A real number.
ν: A real number.

This routine returns the spherical Hankel function of the first kind $h_\nu^{(1)}(x)$, defined as

$$h_\nu^{(1)} = \sqrt{\frac{\pi}{2x}}H^{(1)}_{\nu+\frac{1}{2}}(x). \tag{16.9.3}$$

16.9.4 Spherical Hankel Function of the Second Kind

Function **cplxHankelSph2Boost**(*x As mpNum, ν As mpNum*) As mpNum

The function cplxHankelSph2Boost returns the spherical Hankel function of the second kind $h_\nu^{(2)}(x)$.

Parameters:
x: A real number.
ν: A real number.

This routine returns the spherical Hankel function of the second kind $h_\nu^{(2)}(x)$, defined as

$$h_\nu^{(2)} = \sqrt{\frac{\pi}{2x}} H^{(2)}_{\nu+\frac{1}{2}}(x). \tag{16.9.4}$$

16.10 Airy Functions

In this section let $z = (2/3)|x|^{3/2}$. For large negative x the Airy functions and the Scorer function $Gi(x)$ have asymptotic expansions oscillating with $\cos(z+\pi/4)$ or $\sin(z+\pi/4)$, see Abramowitz and Stegun [1, 10.4.60, 10.4.64, 10.4.87]; therefore the phase information becomes totally unreliable for $x < \hat{a}\acute{L}\check{S}(2/epsx)^{2/3}$, and the relative error increases strongly for x less than the square root.

16.10.1 Airy Function Ai(x)

Function **AiryAiBoost**(*x As mpNum*) As mpNum

The function AiryAiBoost returns the Airy function $\mathrm{Ai}(x)$.

Parameter:
x: A real number.

The Airy function $\mathrm{Ai}(x)$ is defined as

$$\mathrm{Ai}(x) = \frac{1}{\pi}\sqrt{\frac{x}{3}} K_{1/3}(z), \quad (x > 0) \tag{16.10.1}$$

$$\mathrm{Ai}(x) = \frac{1}{3^{2/3}\Gamma(2/3)}, \quad (x = 0) \tag{16.10.2}$$

$$\mathrm{Ai}(x) = \tfrac{1}{2}\sqrt{-x}\left(J_{1/3}(z) - \frac{1}{\sqrt{3}} Y_{1/3}(z)\right), \quad (x < 0) \tag{16.10.3}$$

16.10.2 Airy Function Ai$'(x)$

Function **AiryAiDerivativeBoost**(*x As mpNum*) As mpNum

The function AiryAiDerivativeBoost returns the Airy function $\mathrm{Ai}'(x)$.

Parameter:
x: A real number.

This routine returns the Airy function $\mathrm{Ai}'(x)$, defined as

$$\mathrm{Ai}'(x) = \frac{x}{\pi\sqrt{3}} K_{2/3}(z), \quad (x > 0) \tag{16.10.4}$$

$$\mathrm{Ai}'(x) = \frac{1}{-(3^{2/3})\Gamma(1/3)}, \quad (x = 0) \tag{16.10.5}$$

$$\mathrm{Ai}'(x) = -\frac{x}{2}\left(J_{2/3}(z) + \frac{1}{\sqrt{3}}Y_{2/3}(z)\right), \quad (x < 0) \tag{16.10.6}$$

16.10.3 Airy Function Bi(x)

Function **AiryBiBoost**(***x*** *As mpNum*) As mpNum

The function AiryBiBoost returns the Airy function $\mathrm{Bi}(x)$.

Parameter:
x: A real number.

This routine returns the Airy function $\mathrm{Bi}(x)$, defined as

$$\mathrm{Bi}(x) = \sqrt{x}\left(\frac{2}{\sqrt{3}}I_{1/3}(z) + \frac{1}{\pi}K_{1/3}(z)\right), \quad (x > 0) \tag{16.10.7}$$

$$\mathrm{Bi}(x) = \frac{1}{3^{1/6}\Gamma(2/3)}, \quad (x = 0) \tag{16.10.8}$$

$$\mathrm{Bi}(x) = -\tfrac{1}{2}\sqrt{-x}\left(\frac{1}{\sqrt{3}}J_{1/3}(z) + Y_{1/3}(z)\right), \quad (x < 0) \tag{16.10.9}$$

16.10.4 Airy Function Bi$'(x)$

Function **AiryBiDerivativeBoost**(***x*** *As mpNum*) As mpNum

The function AiryBiDerivativeBoost returns the Airy function $\mathrm{Bi}'(x)$.

Parameter:
x: A real number.

This routine returns the Airy function $\mathrm{Bi}'(x)$, defined as

$$\mathrm{Bi}'(x) = x\left(\frac{2}{\sqrt{3}}I_{2/3}(z) + \frac{1}{\pi}K_{2/3}(z)\right), \quad (x > 0) \tag{16.10.10}$$

$$\mathrm{Bi}(x) = \frac{3^{1/6}}{\Gamma(1/3)}, \quad (x = 0) \tag{16.10.11}$$

$$\mathrm{Bi}(x) = -\tfrac{x}{2}\left(\frac{1}{\sqrt{3}}J_{2/3}(z) - Y_{2/3}(z)\right), \quad (x < 0) \tag{16.10.12}$$

16.11 Carlson-style Elliptic Integrals

The Carlson style elliptic integrals are a complete alternative group to the classical Legendre style integrals. They are symmetric and the numerical calculation is usually performed by duplication as described in Carlson & Gustafson (1994) and Carlson (1995).

16.11.1 Degenerate elliptic integral RC

Function **CarlsonRCBoost**(***x*** *As mpNum*, ***y*** *As mpNum*) As mpNum

The function CarlsonRCBoost returns the value of the of Carlson's degenerate elliptic integral R_C.

Parameters:
x: A real number.
y: A real number.

This function computes the value of the of Carlson's degenerate elliptic integral R_C for $x \geq 0$, $y \neq 0$:

$$R_C(x, y) = R_F(x, y, y) = \frac{1}{2}\int_0^\infty (t + x)^{-1/2}(t + y)^{-1}dt. \tag{16.11.1}$$

16.11.2 Integral of the 1st kind RF

Function **CarlsonRFBoost**(***x*** *As mpNum*, ***y*** *As mpNum*, ***z*** *As mpNum*) As mpNum

The function CarlsonRFBoost returns the value of the of Carlson's elliptic integral R_F of the first kind.

Parameters:
x: A real number.
y: A real number.
z: A real number.

This function computes the value of the of Carlson's elliptic integral R_F of the first kind

$$R_F(x, y, z) = \frac{1}{2}\int_0^\infty ((t + x)(t + y)(t + z))^{-1/2}dt. \tag{16.11.2}$$

with $x, y, z \geq 0$, at most one may be zero.

16.11.3 Integral of the 2nd kind RD

Function **CarlsonRDBoost**(***x*** *As mpNum*, ***y*** *As mpNum*, ***z*** *As mpNum*) As mpNum

The function CarlsonRDBoost returns the value of the of Carlson's elliptic integral R_D of the second kind.

Parameters:
x: A real number.
y: A real number.
z: A real number.

This function computes the value of the of Carlson's elliptic integral R_D of the second kind

$$R_D(x, y, z) = R_J(x, y, z, z) = \frac{3}{2}\int_0^\infty ((t + x)(t + y))^{-1/2}(t + z)^{-3/2}dt. \tag{16.11.3}$$

with $z > 0$, $x, y \geq 0$, at most one of x, y may be zero.

16.11.4 Integral of the 3rd kind RJ

Function **CarlsonRJBoost**(***x*** *As mpNum*, ***y*** *As mpNum*, ***z*** *As mpNum*, ***r*** *As mpNum*) As mpNum

The function CarlsonRJBoost returns the value of the of Carlson's elliptic integral R_J of the third kind.

Parameters:
x: A real number.
y: A real number.
z: A real number.
r: A real number.

This function computes the value of the of Carlson's elliptic integral R_J of the third kind

$$R_J(x, y, z, r) = \frac{3}{2}\int_0^\infty ((t+x)(t+y)(t+z))^{-1/2}(t+r)^{-1}dt. \tag{16.11.4}$$

with $x, y, z \geq 0$, at most one of may be zero, and $r \neq 0$.

16.12 Legendre-style Elliptic Integrals

16.12.1 Complete elliptic integral of the 1st kind

Function **CompleteLegendreEllint1Boost**(***k*** *As mpNum*) As mpNum

The function CompleteLegendreEllint1Boost returns the value of the complete elliptic integral of the first kind.

Parameter:
k: A real number.

This function computes the value of the complete elliptic integral of the first kind $K(k)$ with $|k| < 1$

$$K(k) = \int_0^{\pi/2} \frac{dt}{\sqrt{1 - k^2 \sin^2 t}}. \tag{16.12.1}$$

16.12.2 Complete elliptic integral of the 2nd kind

Function **CompleteLegendreEllint2Boost**(***k*** *As mpNum*) As mpNum

The function CompleteLegendreEllint2Boost returns the value of the complete elliptic integral of the second kind.

Parameter:
k: A real number.

This function computes the value of the complete elliptic integral of the second kind $E(k)$ with $|k| \leq 1$

$$E(k) = \int_0^{\pi/2} \sqrt{1 - k^2 \sin^2 t}. \tag{16.12.2}$$

16.12.3 Complete elliptic integral of the 3rd kind

Function **CompleteLegendreEllint3Boost**(ν *As mpNum*, ***k*** *As mpNum*) As the value of the complete elliptic integral of the third kind.

The function CompleteLegendreEllint3Boost returns

Parameters:
ν: A real number.
k: A real number.

This function computes the value of the complete elliptic integral of the third kind $\Pi(\nu, k)$ with $|k| <, \nu \neq 1$

$$\Pi(\nu, k) = \int_0^{\pi/2} \frac{dt}{(1 - \nu \sin^2 t)\sqrt{1 - k^2 \sin^2 t}}. \tag{16.12.3}$$

16.12.4 Legendre elliptic integral of the 1st kind

Function **LegendreEllint1Boost**(ϕ *As mpNum*, ***k*** *As mpNum*) As mpNum

The function LegendreEllint1Boost returns the value of the incomplete Legendre elliptic integral of the first kind.

Parameters:
ϕ: A real number.
k: A real number.

This function computes the value of the incomplete Legendre elliptic integral of the first kind

$$F(\phi, k) = \int_0^{\phi} \frac{dt}{\sqrt{1 - k^2 \sin^2 t}}. \tag{16.12.4}$$

with $|k \sin \phi| \leq 1$.

16.12.5 Legendre elliptic integral of the 2nd kind

Function **LegendreEllint2Boost**(ϕ *As mpNum*, ***k*** *As mpNum*) As mpNum

The function LegendreEllint2Boost returns the value of the incomplete Legendre elliptic integral of the second kind.

Parameters:
ϕ: A real number.
k: A real number.

This function computes the value of the incomplete Legendre elliptic integral of the second kind

$$E(\phi, k) = \int_0^{\phi} \sqrt{1 - k^2 \sin^2 t}. \tag{16.12.5}$$

with $|k \sin \phi| \leq 1$.

16.12.6 Legendre elliptic integral of the 3rd kind

Function **LegendreEllint3Boost**(ϕ *As mpNum*, ν *As mpNum*, ***k*** *As mpNum*) As mpNum

The function LegendreEllint3Boost returns the value of the incomplete Legendre elliptic integral of the third kind.

Parameters:
ϕ: A real number.
ν: A real number.
k: A real number.

This function computes the value of the incomplete Legendre elliptic integral of the third kind

$$\Pi(\phi, \nu, k) = \int_0^{\phi} \frac{dt}{(1 - \nu \sin^2 t)\sqrt{1 - k^2 \sin^2 t}}. \tag{16.12.6}$$

with $|k \sin \phi| \leq 1$.

16.13 Jacobi Elliptic Functions

These procedures return the Jacobi elliptic functions sn, cn, dn for argument x and complementary parameter m_c. A convenient implicit definition of the functions is

$$x = \int_0^{\mathrm{sn}} \frac{dt}{\sqrt{(1 - t^2)(1 - k^2 t^2)}}, \quad \mathrm{sn}^2 + \mathrm{cn}^2 = 1, \quad k^2 \mathrm{sn}^2 + \mathrm{cn}^2 = 1 \tag{16.13.1}$$

with $k^2 = 1 - m_c$. There are a lot of equivalent definition of the Jacobi elliptic functions, e.g. with the Jacobi amplitude function (see e.g. Olver *et al.* (2010) [30, 22.16.11/12])

$$\mathrm{sn}(x, k) = \sin(\mathrm{am}(x, k)),$$
$$\mathrm{cn}(x, k) = \cos(\mathrm{am}(x, k)),$$

or with Jacobi theta functions (cf. [Olver *et al.* (2010), 22.2]).

16.13.1 Jacobi elliptic function sn

Function **JacobiSNBoost**(***x*** *As mpNum*, ***k*** *As mpNum*) As mpNum

The function JacobiSNBoost returns the Jacobi elliptic function $\mathrm{sn}(x, k)$.

Parameters:
x: A real number.
k: A real number.

16.13.2 Jacobi elliptic function cn

Function **JacobiCNBoost**(***x*** *As mpNum*, ***k*** *As mpNum*) As mpNum

The function JacobiCNBoost returns the Jacobi elliptic function $\mathrm{cn}(x, k)$.

Parameters:
x: A real number.
k: A real number.

16.13.3 Jacobi elliptic function dn

Function **JacobiDNBoost**(***x*** *As mpNum*, ***k*** *As mpNum*) As mpNum

The function JacobiDNBoost returns the Jacobi elliptic function $\mathrm{dn}(x, k)$.

Parameters:
x: A real number.
k: A real number.

16.14 Zeta Functions

16.14.1 Riemann $\zeta(s)$ function

Function **RiemannZetaBoost**(***s*** *As mpNum*) As mpNum

The function RiemannZetaBoost returns the Riemann zeta function.

Parameter:
s: A real number.

The Riemann zeta function $\zeta(s)$ for $s \neq 1$ is defined as

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s}, \quad s > 1. \tag{16.14.1}$$

If $s < 0$, the reflection formula is used:

$$\zeta(s) = 2(2\pi)^{s-1} \sin\left(\tfrac{1}{2}\pi s\right) \Gamma(1-s)\zeta(1-s) \tag{16.14.2}$$

16.15 Exponential Integral and Related Integrals

16.15.1 Exponential Integral E1

Function **ExponentialIntegralE1Boost**(***x*** *As mpNum*) As mpNum

The function ExponentialIntegralE1Boost returns the exponential integral $\mathrm{E}_1(x)$.

Parameter:
x: A real number.

The exponential integral $\mathrm{E}_1(x)$ for $x \neq 0$ is defined as

$$\mathrm{E}_1(x) = \int_1^{\infty} \frac{e^{-xt}}{t} dt, \tag{16.15.1}$$

For $x < 0$ the integral is calculated as $\mathrm{E}_1(x) = -\mathrm{Ei}(-x)$.

16.15.2 Exponential Integral Ei

Function **ExponentialIntegralEiBoost**(***x*** *As mpNum*) As mpNum

The function ExponentialIntegralEiBoost returns the exponential integral $\mathrm{Ei}(x)$.

Parameter:
x: A real number.

The exponential integral $\mathrm{Ei}(x)$ for $x \neq 0$ is defined as

$$\mathrm{Ei}(x) = -PV \int_{-x}^{\infty} \frac{e^{-t}}{t} dt = PV \int_{-\infty}^{x} \frac{e^{t}}{t} dt, \tag{16.15.2}$$

For $x < 0$ the integral is calculated as $\mathrm{Ei}(x) = -\mathrm{E}_1(-x)$.

16.15.3 Exponential Integrals En

Function **ExponentialIntegralEnBoost**(***x*** *As mpNum*, ***n*** *As mpNum*) As mpNum

The function ExponentialIntegralEnBoost returns the exponential integral $\mathrm{E}_n(x)$.

Parameters:
x: A real number.
n: A real number.

The exponential integrals $\mathrm{E}_n(x)$ of integer order is defined as

$$\mathrm{E}_n(x) = \int_{1}^{\infty} \frac{e^{-xt}}{t^n} dt, \quad (n \geq 0). \tag{16.15.3}$$

For $x < 0$ the integral is calculated as $\mathrm{Ei}(x) = -\mathrm{E}_1(-x)$.

16.16 Basic Functions

Function **SinPiBoost**(***x*** *As mpNum*) As mpNum

The function SinPiBoost returns the value of the sine of πx, with x in radians.

Parameter:
x: A real number.

Function **CosPiBoost**(***x*** *As mpNum*) As mpNum

The function CosPiBoost returns the value of the cosine of πx, with x in radians.

Parameter:
x: A real number.

16.16.1 Auxiliary Function $\ln(1 + x)$

Function **Lnp1Boost**(***x*** *As mpNum*) As mpNum

The function Lnp1Boost returns the value of the function $\ln(1 + x)$.

Parameter:
x: A real number.
In Boost, this function is called Log1p.

16.16.2 Auxiliary Function $e^x - 1$

Function **Expm1Boost**(***x*** *As mpNum*) As mpNum

The function Expm1Boost returns the value of the function $\text{expm1}(x) = e^x - 1$.

Parameter:
x: A real number.

16.16.3 Cube Root: $\sqrt[3]{x}$

Function **CbrtBoost**(***x*** *As mpNum*) As mpNum

The function CbrtBoost returns the absolute value of the cube root of x, $\sqrt[3]{x}$.

Parameter:
x: A real number.

16.16.4 Auxiliary Function $\sqrt{x+1} - 1$

Function **Sqrtp1m1Boost**(***x*** *As mpNum*) As mpNum

The function Sqrtp1m1Boost returns the value of $\sqrt{x+1} - 1$.

Parameter:
x: A real number.

16.16.5 Auxiliary Function $x^y - 1$

Function **Powm1Boost**(***x*** *As mpNum*, ***y*** *As mpNum*) As mpNum

The function Powm1Boost returns the value of $x^y - 1, y \in \mathbb{R}$.

Parameters:
x: A real number.
y: A real number.

16.16.6 Auxiliary Function $\sqrt{x^2 + y^2}$

Function **HypotBoost**(***x*** *As mpNum*, ***y*** *As mpNum*) As mpNum

The function HypotBoost returns the value of $\sqrt{x^2 + y^2}$.

Parameters:
x: A real number.
y: A real number.

16.17 Sinus Cardinal Function and Hyperbolic Sinus Cardinal Functions

Function **SincaBoost**(***x*** *As mpNum*) As mpNum

The function SincaBoost returns the sinus cardinal function

Parameter:
x: A real number.

The sinus cardinal function is defined as

$$\mathrm{sinc}_a(x) = \sin\left(\frac{\pi x}{a}\right)\frac{a}{\pi x} \tag{16.17.1}$$

16.17.1 Hyperbolic Sinus Cardinal: $\mathbf{Sinhc}_a(x)$

Function **SinhcaBoost**(***x*** *As mpNum*) As mpNum

The function SinhcaBoost returns the hyperbolic sinus cardinal function.

Parameter:
x: A real number.

The hyperbolic sinus cardinal function is defined as

$$\mathrm{sinhc}_a(x) = \sinh\left(\frac{\pi x}{a}\right)\frac{a}{\pi x} \tag{16.17.2}$$

16.18 Inverse Hyperbolic Functions

16.18.1 Hyperbolic Arc-cosine: $\mathbf{acosh}(x)$

Function **AcoshBoost**(***x*** *As mpNum*) As mpNum

The function AcoshBoost returns the value of the hyperbolic arc-cosine of x in radians.

Parameter:
x: A real number.

16.18.2 Hyperbolic Arc-sine: $\mathbf{asinh}(x)$

Function **AsinhBoost**(***x*** *As mpNum*) As mpNum

The function AsinhBoost returns the value of the hyperbolic arc-sine of x in radians.

Parameter:
x: A real number.

16.18.3 Hyperbolic Arc-tangent: atanh(x)

Function **AtanhBoost**(***x*** *As mpNum*) As mpNum

The function AtanhBoost returns the value of the hyperbolic arc-tangent of x in radians.

Parameter:
x: A real number.

Chapter 17

Distribution Functions

17.1 Introduction to Distribution Functions

This is a citation Walck (2007), and some more.
This is a citation Van Hauwermeiren & Vose (2009), and some more.
This is a citation Rinne (2008), and some more.
This is a citation Johnson *et al.* (1994.), and some more.
This is a citation Johnson *et al.* (1995.), and some more.
See also Monahan (2011)
See also Lange (2010)
See also Chernick (2008)
See also Cheney & Kincaid (2008)

17.1.1 Continuous Distribution Functions

Continuous random number distributions are defined by a probability density function, $p(x)$, such that the probability of x occurring in the infinitesimal range x to $x + dx$ is $p\ dx$. The cumulative distribution function for the lower tail $P(x)$ gives the probability of a variate taking a value less than x, and the cumulative distribution function for the upper tail $Q(x)$ gives the probability of a variate taking a value greater than x.
The upper and lower cumulative distribution functions are related by $P(x) + Q(x) = 1$ and satisfy $0 \leq P(x) \leq 1, 0 \leq Q(x) \leq 1$. The inverse cumulative distributions, $x = P - 1(P)$ and $x = Q - 1(Q)$ give the values of x which correspond to a specific value of P or Q. They can be used to find confidence limits from probability values.

17.1.2 Discrete Distribution Functions

For discrete distributions the probability of sampling the integer value k is given by $p(k)$. The cumulative distribution for the lower tail $P(k)$ of a discrete distribution is defined as the sum over the allowed range of the distribution less than or equal to k. The cumulative distribution for the upper tail of a discrete distribution $Q(k)$ is defined as the sum of probabilities for all values greater than k. These two definitions satisfy the identity $P(k) + Q(k) = 1$. If the range of the distribution is 1 to n inclusive then $P(n) = 1$, $Q(n) = 0$ while $P(1) = p(1)$, $Q(1) = 1 - p(1)$.

17.1.3 Commonly Used Function Types

17.1.3.1 Functions returning pdf, CDF, and related information

These functions have the form ?Dist(x; [Parameters;], OutputString). Here
"?" is a placeholder for the name of the distribution,
"x" is the value for which we want to calculate the pdf, CDF etc,
"[Parameters;]" denote any parameters (like degrees of freedom) of the distribution, and
"OutputString" specifies the computed results which will be returned. This can be any of the following:

- **pdf**: the probability density function
- **P**: the cumulative distribution function (CDF)
- **Q**: the complement of cumulative distribution function (CDF)
- **logpdf**: the logarithm of the probability density function
- **logP**: the logarithm of the cumulative distribution function (CDF)
- **logQ**: the logarithm of the complement of cumulative distribution function (CDF)
- **h**: hazard function
- **H**: cumulative hazard function

As an example, for Student's t-distribution, a "T" is used to specify the name of the distibution, and there is just one distribution parameter, ν, the degrees of freedom. Therefore, the function has the form

TDist(x As nmNum; ν As mpNum, OutputString As String) As mpNumList,

and an actual call to the function, requesting the pdf, CDF, and the complement of the CDF for $x = 2.3$ and $\nu = 22$ could be

```
Result = TDist(2.3, 22, "pdf + P + Q")
mp.Print Result
```

which produces the output

```
pdf: 0.434234342343434
P: 0.943453463453453
Q: 0.054564564564236
```

17.1.3.2 Functions returning Quantiles

These functions have the form ?DistInv(Prob; [Parameters;], OutputString). Here
"?" is a placeholder for the name of the distribution,
"Prob" sets the target values for P and Q,
"[Parameters;]" denote any parameters (like degrees of freedom) of the distribution, and
"OutputString" specifies the computed results which will be returned. This can be any of the following:

- **PInv**: the inverse of the cumulative distribution function (CDF). For discrete distribution, this will be outwardly rounded
- **QInv**: the inverse of the complement of the cumulative distribution function (CDF). For discrete distribution, this will be outwardly rounded
- **P**: the value of the cumulative distribution function (CDF), which has actually been achieved
- **Q**: the value of the complement of the cumulative distribution function (CDF), which has actually been achieved

As an example, for Student's t-distribution, a "T" is used to specify the name of the distibution, and there is just one distribution parameter, ν, the degrees of freedom. Therefore, the function has the form

TDistInv(*Prob* As mpNum; ν As mpNum, OutputString As String) As mpNumList,

and an actual call to the function, requesting the inverse of the complement of the CDF for $Prob = 0.01$ and $\nu = 22$ could be

```
Result = TDistInv(0,01, 22, "QInv")
mp.Print Result
```

which produces the output

```
QInv: 2.943453463453453
```

17.1.3.3 Functions returning moments and related information

These functions have the form ?DistInfo([Parameters;], OutputString). Here
"?" is a placeholder for the name of the distribution,
"[Parameters;]" denote any parameters (like degrees of freedom) of the distribution, and
"OutputString" specifies the computed results which will be returned. This can be any of the following:

- **range**: Returns the valid range of the random variable over distribution dist.
- **support**:
- **mode**: Returns the mode of the distribution dist. This function may return a domain_error if the distribution does not have a defined mode.
- **median**: Returns the median of the distribution dist.
- **mean**: Returns the mean of the distribution dist. This function may return a domain_error if the distribution does not have a defi ned mean (for e xample the Cauchy distribution).
- **stdev**: Returns the standard deviation of distribution dist. This function may return a domain_error if the distribution does not have a defined standard deviation.
- **variance**: Returns the variance of the distribution dist. This function may return a domain_error if the distribution does not have a defi ned v ariance.
- **skewness**: Returns the skewness of the distribution dist. This function may return a domain_error if the distribution does not have a defined skewness.
- **kurtosis**: Returns the 'proper' kurtosis (normalized fourth moment) of the distribution dist.
- **kurtosis excess**: Returns the kurtosis excess of the distribution dist. kurtosis excess = kurtosis - 3

As an example, for Student's t-distribution, a "T" is used to specify the name of the distibution, and there is just one distribution parameter, ν, the degrees of freedom. Therefore, the function has the form

TDistInfo(ν As mpNum, OutputString As String) As mpNumList,

and an actual call to the function, requesting the mean, varaince, skewness and kurtosis with $\nu = 22$ could be

```
Result = TDistInfo(22, "mean + variance + skewness + kurtosis")
mp.Print Result
```

which produces the output

```
mean: 0.434234342343434
variance: 0.943453463453453
skewness: 0.054564564564236
kurtosis: 0.6054564564564236
```

17.1.3.4 Functions returning Sample Size estimates

These functions have the form ?SampleSize(Alpha; Beta; ModifiedNoncentrality; [Parameters;], OutputString). Here
"?" is a placeholder for the name of the distribution,
"Alpha" specifies the confidence level (or Type I error),
"Beta" specifies the Type I error (or 1 − Power),
"ModifiedNoncentrality" specifies the (modified) noncentrality parameter of the distribution in a form which does not depend on sample size (which may require a modification compared to the conventional form for stating the noncentrlaity parameter),
"[Parameters;]" denote any additional parameters of the distribution (if any) which are not a function of the sample size, and
"OutputString" specifies the computed results which will be returned. This can be any of the following:

- **ExactN**: returns an "exact", i.e. typically non-integer sample size estimate
- **UpperN**: upper integer sample size estimate
- **LowerN**: lower integer sample size estimate
- **UpperNPower**: actual power when using UpperN
- **LowerNPower**: actual power when using LowerN

As an example, for the noncentral t-distribution, the prefix "NoncentralT" is used to specify the name of the distibution. The distribution parameter ν, the degrees of freedom, which depends on the sample size, and is therefore not included in the parameter list of this function. The modified noncentrality parameter is called $\tilde{\rho} = \Delta/\sigma$. Therefore, the function has the form

NoncentralTSampleSize(α As mpNum, β As mpNum, $\tilde{\rho}$ As mpNum, OutputString As String) As mpNumList

and an actual call to the function, requesting an upper sample size estimate (and actual power) for $\alpha = 0.95$, $\beta = 0.1$, and $\tilde{\rho} = \Delta/\sigma = 0.6$ would be

```
Result = NoncentralTSampleSize(0.95, 0.1, 0.6, "UpperN + UpperNPower")
mp.Print Result
```

which produces the output

```
UpperN: 26
UpperNPower: 0.92435435
```

17.1.3.5 Functions related to noncentrality parameters

These functions have the form ?Noncentrality(Alpha; Noncentrality; [Parameters;], OutputString). Here
"?" is a placeholder for the name of the distribution,
"Alpha" specifies the confidence level (or Type I error),
"Noncentrality" specifies the noncentrality parameter of the distribution,
"[Parameters;]" denote any additional parameters of the distribution, and
"OutputString" specifies the computed results which will be returned. This can be any of the following:

- **UpperCI**: upper confidence interval
- **LowerCI**: lower confidence interval
- **TwoSidedCI**: two-sided confidence interval

As an example, for the noncentral t-distribution, the prefix "NoncentralT" is used to specify the name of the distibution. The noncentrality parameter is δ, and the other distribution parameter is ν, the degrees of freedom. Therefore, the function has the form

NoncentralTNoncentrality(α As mpNum, δ As mpNum, ν As mpNum, OutputString As String) As mpNumList

and an actual call to the function, requesting an upper confidence interval for δ with $\alpha = 0.95$, $\delta = 0.6$ and $\nu = 22$ would be

```
Result = NoncentralTNoncentrality(0.95, 0.6, 22, "UpperCI")
mp.Print Result
```

which produces the output

```
UpperCI: 0.7546534
```

17.1.3.6 Functions returning Random numbers

These functions have the form ?DistRan(Size; [Parameters;], Generator, OutputString). Here
"?" is a placeholder for the name of the distribution,
"Size" specifies the size of the random sample,
"[Parameters;]" denote any parameters (like degrees of freedom) of the distribution, and
"Generator" specifies the pseudo random generator which will be used to produce the random sample,
"OutputString" specifies the computed results which will be returned. This can be any of the following:

- **Unsorted**: produces unsorted output
- **Ascending**: output sorted in ascending order
- **Descending**: output sorted in descending order
- **Histogram**(k): output grouped in histogram format, with k buckets
- **HistogramCDF**(k): cumulated output grouped in histogram format, with k buckets

As an example, for Student's t-distribution, a "T" is used to specify the name of the distibution, and there is just one distribution parameter, ν, the degrees of freedom. Therefore, the function has the form

TDistRan($Size$ As Integer; ν As mpNum, Generator As String, OutputString As String) As mpNumList,

and an actual call to the function, requesting a random sample of $Size = 10000$ of a t-distribution with $\nu = 22$, using the default pseudo-random number generator, sorting output in ascending order could be

```
Result = TDistRan(10000, 22, "Default", "Ascending")
mp.Plot Result
```

which produces the output

```
QInv: 2.943453463453453
```

17.2 Beta-Distribution

17.2.1 Definition

If X_1 an X_2 is are independent random variables following χ^2-distribution with $2a$ and $2b$ degrees of freedom respectively, then the distribution of the ratio $\frac{X_1}{X_1+X_2}$ is said to follow a Beta-distribution with a and b degrees of freedom.
See Tretter & Walster (1979)

17.2.2 Density and CDF

Function **BetaDist**(***x*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function BetaDist returns returns pdf, CDF and related information for the central Beta-distribution

Parameters:
x: A real number
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.2.2.1 and 17.2.2.2.

17.2.2.1 Density

The pdf of a variable following a central Beta-distribution with a and b degrees of freedom is given by

$$f_{\text{Beta}}(a,b,x) = \frac{1}{B(a,b)} x^{a-1}(1-x)^{b-1} \tag{17.2.1}$$

where $B(a,b)$ denotes the beta function (see section 6.7.11).

17.2.2.2 CDF: General formulas

The cdf of a variable following a central Beta-distribution with a and b degrees of freedom is given by

$$\Pr[X \le x] = F_{\text{Beta}}(a,b,x) = \int_0^x f_{\text{Beta}}(a,b,t)dt \tag{17.2.2}$$

17.2.2.3 Exact cdf as continued fraction

The following representation as continued fraction is used (Peizer 1968, .1428 and 1452):

$$I(a,b,x) = \binom{n}{a} p^{b-1} q^a \frac{1}{(1+u_1/(v_1+u_2/(v2+u3/(v_3+\cdots))))}, \quad \text{where} \tag{17.2.3}$$

$$p=(1-x), \quad q=x, \quad n=a+b-1, \quad u_1=\frac{-(b-1)q}{p}, \quad u_{2j}=\frac{j(n+j)q}{p},$$

$$u_{2j+1}=\frac{-(a+j)(b-j-1)q}{p}, \quad v_j=a+j, \quad j=1,2,\ldots$$

17.2.3 Quantiles

Function **BetaDistInv**(***Prob*** *As mpNum*, ***m*** *As mpNum*, ***n*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function BetaDistInv returns returns quantiles and related information for the the central Beta-distribution

Parameters:
Prob: A real number between 0 and 1.
m: A real number greater 0, representing the numerator degrees of freedom
n: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*).

17.2.4 Properties

Function **BetaDistInfo**(***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function BetaDistInfo returns returns moments and related information for the central Beta-distribution

Parameters:
a: A real number greater 0, representing the degrees of freedom
b: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given in section 17.13.4.

17.2.4.1 Moments: algorithms and formulas

The raw moments are given by:

$$E^h(W) = \frac{\Gamma(a+h)\Gamma(a+b)}{\Gamma(a)\Gamma(a+b+h)} \tag{17.2.4}$$

The raw moments of the power of a beta vairiable are given by:

$$E^h(W^s) = \frac{\Gamma(a+hs)\Gamma(a+b)}{\Gamma(a)\Gamma(a+b+hs)} \tag{17.2.5}$$

17.2.4.2 Recurrences

$$I(a,b;x) = 1 - I(b,a;1-x) \tag{17.2.6}$$
$$I(a,b;x) = \binom{n}{a} x^a (1-x)^{b-1} + I(a+1,b-1;x) \tag{17.2.7}$$
$$I(a,b;x) = \binom{n}{a} x^a (1-x)^{b} + I(a+1,b;x) \tag{17.2.8}$$
$$I(a,b+1;x) = \binom{n}{a} x^a (1-x)^{b} + I(a,b;x) \tag{17.2.9}$$
$$I(a,b;x) = \binom{n}{a+b} x^a (1-x)^{b} \frac{a}{a+b-x} + I(a+1,b+1;x) \tag{17.2.10}$$
$$I(a,b;x) = F\left(2a, 2b, \frac{nx}{m-mx}\right) \tag{17.2.11}$$

17.2.5 Random Numbers

Function **BetaDistRandom**(***Size*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function BetaDistRandom returns returns random numbers following a central Beta-distribution

Parameters:
Size: A positive integer up to 10^7
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given below.

17.2.5.1 Random Numbers: algorithms and formulas

In order to obtain random numbers from a Beta distribution we first single out a few special cases. For $p = 1$ and/or $q = 1$ we may easily solve the equation $F(x) = \xi$ where $F(x))$ is the cumulative function and ξ a uniform random number between zero and one. In these cases

$$p = 1 \Rightarrow x = 1 - \xi^{1/q}$$
$$q = 1 \Rightarrow x = \xi^{1/q}$$

For p and q half-integers we may use the relation to the chi-square distribution by forming the ratio $\frac{y_m}{y_m+y_n}$ with y_m and y_n two independent random numbers from chi-square distributions with $m = 2p$ and $n = 2q$ degrees of freedom, respectively.
Yet another way of obtaining random numbers from a Beta distribution valid when p and q are both integers is to take the l^{th} out of k $(1 \leq l \leq k)$ independent uniform random numbers between zero and one (sorted in ascending order). Doing this we obtain a Beta distribution with parameters $p = l$ and $q = k + 1 - l$. Conversely, if we want to generate random numbers from a Beta distribution with integer parameters p and q we could use this technique with $l = p$ and $k = p + q - 1$. This last technique implies that for low integer values of p and q simple code may be used, e.g. for $p = 2$ and $q = 1$ we may simply take $\max(\xi_1, \xi_2)$ i.e. the maximum of two uniform random numbers (Walck, 2007).

17.3 Binomial Distribution

These functions return PMF and CDF of the (discrete) binomial distribution with number of trials $n \geq 0$ and success probability $0 \leq p \leq 1$.

17.3.1 Density and CDF

Function **BinomialDist**(***x*** *As mpNum*, ***n*** *As mpNum*, ***p*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function BinomialDist returns returns pdf, CDF and related information for the central Binomial-distribution

Parameters:
x: The number of successes in trials.
n: The number of independent trials.
p: The probability of success on each trial
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.3.1.1 and 17.3.1.2.

17.3.1.1 Density

$$f_{\text{Bin}}(n,k;p) = \binom{n}{k} p^k (1-p)^{n-k} = f_{\text{Beta}}(k+1, n-k+1, p)/(n+1) \tag{17.3.1}$$

17.3.1.2 CDF

$$F_{\text{Bin}}(n,k;p) = I_{1-p}(n-k, k+1) = ibeta(n-k, k+1, 1-p) \tag{17.3.2}$$

17.3.2 Quantiles

Function **BinomialDistInv**(***Prob*** *As mpNum*, ***n*** *As mpNum*, ***p*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function BinomialDistInv returns returns quantiles and related information for the the central binomial-distribution

Parameters:
Prob: A real number between 0 and 1.
n: The number of Bernoulli trials.
p: The probability of a success on each trial.
Output: A string describing the output choices

See section 17.1.3.2 for the options for *Prob* and *Output*).

17.3.3 Properties

Function **BinomialDistInfo**(***n*** *As mpNum*, ***p*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function BinomialDistInfo returns returns moments and related information for the central Binomial-distribution

Parameters:
n: The number of Bernoulli trials.
p: The probability of a success on each trial.
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given in section 17.13.4.

17.3.3.1 Moments: algorithms and formulas

$$\mu_r^{'} = \sum_{i=0}^{r} \binom{n}{i} \left(\sum_{j=0}^{i} \binom{i}{j} (-1)^j (i-j)^r \right) \tag{17.3.3}$$

$$\mu_1 = np \tag{17.3.4}$$

$$\mu_2 = np(1-p) = npq \tag{17.3.5}$$

$$\mu_3 = npq(q-p) \tag{17.3.6}$$

$$\mu_4 = 3(npq)^3 + npq(1-6pq) \tag{17.3.7}$$

17.3.4 Random Numbers

Function **BinomialDistRandom**(***Size*** *As mpNum*, ***n*** *As mpNum*, ***p*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function BinomialDistRandom returns returns random numbers following a central Binomial-distribution

Parameters:
Size: A positive integer up to 10^7
n: The number of Bernoulli trials.
p: The probability of a success on each trial.
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given below.

17.3.4.1 Random Numbers: algorithms and formulas

In order to obtain random numbers from a Binomial distribution we first single out a few special cases. For $p = 1$ and/or $q = 1$ we may easily solve the equation $F(x) = \xi$ where $F(x))$ is the cumulative function and ξ a uniform random number between zero and one. In these cases

$$p = 1 \Rightarrow x = 1 - \xi^{1/q}$$
$$q = 1 \Rightarrow x = \xi^{1/q}$$

For p and q half-integers we may use the relation to the chi-square distribution by forming the ratio $\frac{y_m}{y_m+y_n}$ with y_m and y_n two independent random numbers from chi-square distributions with $m = 2p$ and $n = 2q$ degrees of freedom, respectively.
Yet another way of obtaining random numbers from a Beta distribution valid when p and q are both integers is to take the l^{th} out of k $(1 \leq l \leq k)$ independent uniform random numbers between zero and one (sorted in ascending order). Doing this we obtain a Beta distribution with parameters $p = l$ and $q = k + 1 - l$. Conversely, if we want to generate random numbers from a Beta distribution with integer parameters p and q we could use this technique with $l = p$ and $k = p + q - 1$. This last technique implies that for low integer values of p and q simple code may be used, e.g. for $p = 2$ and $q = 1$ we may simply take $\max(\xi_1, \xi_2)$ i.e. the maximum of two uniform random numbers (Walck, 2007).

17.4 Chi-Square Distribution

17.4.1 Definition

Let $X_1, X_2, \ldots, X_n$ be independent and identically distributed random variables each following a normal distribution with mean zero and unit variance. Then $\chi^2 = \sum_{j=1}^{n} X_j$ is said to follow a χ^2-distribution with n degress of freedom.

17.4.2 Density and CDF

Function **CDist**(***x*** *As mpNum*, ***n*** *As mpNum*, ***Output*** *As String*) As mpNumList

---

The function CDist returns returns pdf, CDF and related information for the central χ^2-distribution

Parameters:
x: A real number
n: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.4.2.1 and 17.4.2.2.

17.4.2.1 Density

The density of a central chi-square variable with n degrees of freedom is given by

$$f_{\chi^2}(n,x) = \frac{1}{2^{n/2}\Gamma(n/2)} x^{(n-2)/2} e^{-x/2}. \tag{17.4.1}$$

17.4.2.2 CDF: General formulas

The cdf of a central chi-square variable with n degrees of freedom is given by

$$\Pr\left[\chi^2 \leq x\right] = F_{\chi^2}(n,x) = \int_0^x f_{\chi^2}(n,t)dt \tag{17.4.2}$$

17.4.2.3 CDF: Continued fraction

For real $n > 0$, the CDF can be calculated using continued fraction (Peizer & Pratt, 1968).
If $(n-1) \leq x$ let $1 - F_{\chi^2}(n,x))$ be a right tail chi square probability. Then

$$1 - F_{\chi^2}(n,x) = f_{\chi^2}(n,x) \frac{1}{(1 + u_1/(v_1 + u_2/(v_2 + u_3/(v_3 + \ldots))))} \tag{17.4.3}$$

where $M = \frac{1}{2}x$, $b = \frac{1}{2}n$, $u_{2j-1} = j - b$, $v_{2j-1} = M$, $u_{2j} = j$, $v_{2j} = 1$, $j = 1, 2, \ldots$
If $(n-1) > x$ let $F_{\chi^2}(n,x)$ be a left tail chi square probability. Then

$$F_{\chi^2}(n,x) = f_{\chi^2}(n,x) \frac{m}{b} \frac{1}{(1 + u_1/(v_1 + u_2/(v_2 + u_3/(v_3 + \ldots))))} \tag{17.4.4}$$

where $M = \frac{1}{2}x$, $b = \frac{1}{2}n$, $u_1 = -M$, $u_{2j} = jM$, $u_{2j+1} = -(b+j)M$, $v_j = b + j$, $j = 1, 2, \ldots$

17.4.3 Quantiles

Function **CDistInv**(***Prob*** *As mpNum*, ***n*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function CDistInv returns quantiles and related information for the the central χ^2-distribution

Parameters:
Prob: A real number between 0 and 1.
n: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*).

17.4.4 Properties

Function **CDistInfo**(***n*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function CDistInfo returns moments and related information for the central χ^2-distribution

Parameters:
n: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given in section 17.4.4.

17.4.5 Random Numbers

Function **CDistRan**(***Size*** *As mpNum*, ***n*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function CDistRan returns random numbers following a central χ^2-distribution

Parameters:
Size: A positive integer up to 10^7
n: A real number greater 0, representing the degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given in section 17.4.5.

As we saw above the sum of n independent standard normal random variables gave a chi-square distribution with n degrees of freedom. This may be used as a technique to produce pseudorandom numbers from a chi-square distribution. This required a generator for standard normal random numbers and may be quite slow. However, if we make use of the Box-Muller transformation in order to obtain the standard normal random numbers we may simplify the calculations. Adding n such squared random numbers implies that

$$y_{2k} = -2\ln(\xi_1 \cdot \xi_2 \cdot \ldots \cdot \xi_k)$$
$$y_{2k+1} = -2\ln(\xi_1 \cdot \xi_2 \cdot \ldots \cdot \xi_k) - 2\ln(\xi_{k+1})[\cos(2\pi\xi_{k+2})]^2$$

for k a positive integer will be distributed as chi-square variable with even or odd number of degrees of freedom. In this manner a lot of unnecessary operations are avoided. Since the chi-square distribution is a special case of the Gamma distribution we may also use a generator for this distribution.

17.4.6 Wishart Matrix

See Gleser (1976)

17.5 Exponential Distribution

These functions return PDF, CDF, and ICDF of the exponential distribution with location a, rate $\alpha > 0$, and the support interval $(a, +\infty)$:

17.5.1 Density and CDF

Function **ExponentialDist**(***x*** *As mpNum*, ***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function ExponentialDist returns returns pdf, CDF and related information for the central Exponential distribution

Parameters:
x: The value of the distribution.
lambda: The parameter of the distribution.
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.2.2.1 and 17.2.2.2.

17.5.1.1 Density

$$f(x) = \alpha \exp(-\alpha(x - a)) \tag{17.5.1}$$

17.5.1.2 CDF

$$F(x) = 1 - \exp(-\alpha(x - a)) = \text{expm1}(-\alpha(x - a)) \tag{17.5.2}$$

17.5.2 Quantiles

Function **ExponentialDistInv**(***Prob*** *As mpNum*, ***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function ExponentialDistInv returns returns quantiles and related information for the the central Exponential distribution

Parameters:
Prob: A real number between 0 and 1.
lambda: The number of Bernoulli trials.
Output: A string describing the output choices

See section 17.1.3.2 for the options for *Prob* and *Output*).

$$F^{-1}(y) = a - \ln 1p(-y)/\alpha \tag{17.5.3}$$

17.5.3 Properties

Function **ExponentialDistInfo**(***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function ExponentialDistInfo returns returns moments and related information for the central t-distribution

Parameters:
lambda: A real number greater 0, representing the parameter of the distribution
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given in section 17.13.4.

17.5.3.1 Moments and cumulants

The mean or expected value of an exponentially distributed random variable X with rate parameter λ is given by

$$E[X] = \frac{1}{\lambda} \tag{17.5.4}$$

The variance of X is given by

$$E[X] = \frac{1}{\lambda^2} \tag{17.5.5}$$

so the standard deviation is equal to the mean.
The moments of X, for $n = 1, 2, ...$, are given by

$$E[X^n] = \frac{n!}{\lambda^n} \tag{17.5.6}$$

17.5.4 Random Numbers

Function **ExponentialDistRandom**(***Size*** *As mpNum*, ***lambda*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function ExponentialDistRandom returns returns random numbers following a central Beta-distribution

Parameters:
Size: A positive integer up to 10^7
lambda: A real number greater 0, representing the numerator degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given in section 17.6.5.

17.5.4.1 Random Numbers: algorithms and formulas

Random numbers can be generated using the inversion formula.

17.6 Fisher's F-Distribution

17.6.1 Definition

If X_1 an X_2 is are independent random variables following χ^2-distribution with m and n degrees of freedom respectively, then the distribution of the ratio $F = \frac{X_1/m}{X_2/n}$ is said to follow a F-distribution with m and n degrees of freedom.

17.6.2 Density and CDF

Function **FDist**(***x*** *As mpNum*, ***m*** *As mpNum*, ***n*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function FDist returns returns pdf, CDF and related information for the central F-distribution

Parameters:
x: A real number
m: A real number greater 0, representing the numerator degrees of freedom
n: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.6.2.1 and 17.6.2.2.

17.6.2.1 Density

The density of a variable following a central F-distribution with m and n degrees of freedom is given by

$$f_F(m,n,x) = \frac{m^{m/2}n^{n/2}}{B(m/2,n/2)}x^{(m-2)/2}(n+mx)^{-(m+n)/2} \tag{17.6.1}$$

17.6.2.2 CDF: General formulas

The cdf of a variable following a central F-distribution with m and n degrees of freedom is given by

$$\Pr[X \leq x] = F_F(m,n,x) = \int_0^x f(m,n,t)dt \tag{17.6.2}$$

17.6.3 Quantiles

Function **FDistInv**(***Prob*** *As mpNum*, ***m*** *As mpNum*, ***n*** *As mpNum*) As mpNumList

The function FDistInv returns returns quantiles and related information for the the central t-distribution

Parameters:
Prob: A real number between 0 and 1.
m: A real number greater 0, representing the numerator degrees of freedom
n: A real number greater 0, representing the denominator degrees of freedomOutput? String? A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*).

17.6.4 Properties

Function **FDistInfo**(***m*** *As mpNum*, ***n*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function FDistInfo returns returns moments and related information for the central t-distribution

Parameters:
m: A real number greater 0, representing the numerator degrees of freedom
n: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given in section 17.6.4.

17.6.5 Random Numbers

Function **FDistRan**(***Size*** *As mpNum*, ***m*** *As mpNum*, ***n*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function FDistRan returns returns random numbers following a central F-distribution

Parameters:
Size: A positive integer up to 10^7
m: A real number greater 0, representing the numerator degrees of freedom
n: A real number greater 0, representing the denominator degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given in section 17.6.5.

17.6.5.1 Random Numbers: algorithms and formulas

Following the definition the quantity $F = \frac{y_m/m}{y_n/n}$ where y_n and y_m are two variables distributed according to the chi-square distribution with n and m degrees of freedom respectively follows the F-distribution. We may thus use this relation inserting random numbers from chi-square distributions (see section ...).

17.7 Gamma (and Erlang) Distribution

These functions return PDF, CDF, and ICDF of the gamma distribution with shape $a > 0$, scale $b > 0$, and the support interval $(0, +\infty)$.
A gamma distribution with shape $a \in \mathbb{N}$ is called Erlang distribution.

17.7.1 Density and CDF

Function **GammaDist**(***x*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function GammaDist returns returns pdf, CDF and related information for the central Gamma-distribution

Parameters:
x: A real number
a: A real number greater 0, a parameter to the distribution
b: A real number greater 0, a parameter to the distribution
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.2.2.1 and 17.2.2.2.

17.7.1.1 Density

$$f(x;a,b) = \frac{x^{a-1}e^{-x/b}}{\Gamma(a)b^a} \tag{17.7.1}$$

17.7.1.2 CDF: General formulas

$$F(x;a,b) = P(a,x/b) = igammap(a,x/b) \tag{17.7.2}$$

17.7.2 Quantiles

Function **GammaDistInv**(***Prob*** *As mpNum*, ***m*** *As mpNum*, ***n*** *As mpNum*) As mpNumList

The function GammaDistInv returns returns quantiles and related information for the the central Gamma-distribution

Parameters:
Prob: A real number between 0 and 1.
m: A real number greater 0, a parameter to the distribution
n: A real number greater 0, a parameter to the distributionOutput? String? A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*).

$$F^{-1}(y) = b \cdot igammapInv(a,y) \tag{17.7.3}$$

17.7.3 Properties

Function **GammaDistInfo**(***a*** *As mpNum*, ***b*** *As mpNum*) As mpNumList

The function GammaDistInfo returns returns moments and related information for the central Gamma-distribution

Parameters:
a: A real number greater 0, representing the degrees of freedom
b: A real number greater 0, representing the degrees of freedomOutput? String? A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given in section 17.13.4.

17.7.3.1 Moments

The algebraic moments are given by (Wolfram)

$$\mu'_r = \frac{b^r \Gamma(a+r)}{\Gamma(a)} \tag{17.7.4}$$

17.7.4 Random Numbers

Function **GammaDistRandom**(***Size*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function GammaDistRandom returns returns random numbers following a central Beta-distribution

Parameters:
Size: A positive integer up to 10^7
a: A real number greater 0, a parameter to the distribution
b: A real number greater 0, a parameter to the distribution
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given below.

17.7.4.1 Random Numbers: algorithms and formulas

In the case of an Erlangian distribution (b a positive integer) we obtain a random number by adding b independent random numbers from an exponential distribution i.e.
$x = -\ln(\xi_1 \cdot \xi_2 \cdot \ldots \cdot \xi_b)/a$
where all the ξ_i are uniform random numbers in the interval from zero to one. Note that care must be taken if b is large in which case the product of uniform random numbers may become zero due to machine precision. In such cases simply divide the product in pieces and add the logarithms afterwards.

17.7.4.2 General case

In a more general case we use the so called Johnk's algorithm

1. Denote the integer part of b with i and the fractional part with f and put $r = 0$. Let ξ denote uniform random numbers in the interval from zero to one.
2. If $i > 0$ then put $r = -\ln(\xi_1 \cdot \xi_2 \cdot \ldots \cdot \xi_i)$.
3. If $f = 0$ then go to 7.
4. Calculate $w_1 = \xi_{i+1}^{1/f}$ and $w_1 = \xi_{i+2}^{1/(1-f)}$.
5. If $w_1 + w_2 > 1$ then go back to iv.
6. Put $r = r - \ln(\xi_{i+3}) \cdot \frac{w_1}{w_1+w_2}$.
7. Quit with $r = r/a$.

17.8 Hypergeometric Distribution

See Upton (1982), Harkness & Katz (1964)
See Ling & Pratt (1984)
See Knüsel & Michalk (1987)
See also Conlon & Thomas (1993)
See also Casagrande *et al.* (1978)

17.8.1 Definition

These functions return PMF and CDF of the (discrete) hypergeometric distribution; the PMF gives the probability that among n randomly chosen samples from a container with n_1 type1 objects and n_2 type2 objects there are exactly k type1 objects.

17.8.2 Density and CDF

Function **HypergeometricDist**(***x*** *As mpNum*, ***n*** *As mpNum*, ***M*** *As mpNum*, ***N*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function HypergeometricDist returns returns pdf, CDF and related information for the central hypergeometric distribution

Parameters:
x: The number of successes in the sample.
n: The size of the sample.
M: The number of successes in the population
N: The population size
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.2.2.1 and 17.2.2.2.

17.8.2.1 Density

$$f(k) = \frac{\binom{n_1}{k}\binom{n_2}{n-k}}{\binom{n_1+n_2}{n}}, \quad (n, n_1, n_2 \geq 0; n \leq n_1 + n_2). \tag{17.8.1}$$

$f(k)$ is computed with the R trick [39], which replaces the binomial coefficients by binomial PMFs with $p = n/(n1 + n2)$.

17.8.2.2 CDF

There is no explicit formula for the CDF, it is calculated as $\sum f(i)$, using the lower tail if $k < nn_1/(n_1+n_2)$ and the upper tail otherwise with one value of the PMF and the recurrence formulas:

$$f(k+1) = \frac{(n_1 - k)(n - k)}{(k+1)(n_2 - n + k + 1)} f(k) \tag{17.8.2}$$

$$f(k-1) = \frac{k(n_2 - n + k)}{(n_1 - k + 1)(n - k + 1)} f(k) \tag{17.8.3}$$

17.8.3 Quantiles

Function **HypergeometricDistInv**(***Prob*** *As mpNum*, ***n*** *As mpNum*, ***M*** *As mpNum*, ***N*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function HypergeometricDistInv returns returns quantiles and related information for the the central hypergeometric distribution

Parameters:
Prob: A real number between 0 and 1.
n: The size of the sample.
M: The number of successes in the population
N: The population size
Output: A string describing the output choices

See section 17.1.3.2 for the options for *Prob* and *Output*).

17.8.4 Sample Size

See Guenther (1974)

17.8.5 Properties

Function **HypergeometricDistInfo**(***n*** *As mpNum*, ***M*** *As mpNum*, ***N*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function HypergeometricDistInfo returns returns moments and related information for the central hypergeometric distribution

Parameters:
n: The size of the sample.
M: The number of successes in the population
N: The population size
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given in section 17.13.4.

17.8.5.1 Moments

$$\mu_1 = nP \tag{17.8.4}$$

$$\mu_2 = nPQ\frac{N-n}{N-1} \tag{17.8.5}$$

$$\mu_3 = nPQ(Q-P)\frac{(N-n)(N-2n)}{(N-1)(N-2)} \tag{17.8.6}$$

$$\kappa_4 = \frac{6nP^2Q^2(N-n)}{N-1}\frac{n(N-n)(5N-6)-N(N-1)}{(N-2)(N-3)} \tag{17.8.7}$$

17.8.6 Random Numbers

Function **HypergeometricDistRandom**(***Size*** *As mpNum*, ***n*** *As mpNum*, ***M*** *As mpNum*, ***N*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function HypergeometricDistRandom returns returns random numbers following a central hypergeometric distribution

Parameters:
Size: A positive integer up to 10^7
n: The size of the sample.
M: The number of successes in the population
N: The population size
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given below.

17.9 Lognormal Distribution

17.9.1 Definition

These functions return PDF, CDF, and ICDF of the lognormal distribution with location a, scale $b > 0$, and the support interval $(0, +\infty)$:
A log-normal (or lognormal) distribution is a continuous probability distribution of a random variable whose logarithm is normally distributed. Thus, if the random variable is log-normally distributed, then has a normal distribution. Likewise, if has a normal distribution, then has a log-normal distribution. A random variable which is log-normally distributed takes only positive real values.
In a log-normal distribution X, the parameters denoted μ and σ are, respectively, the mean and standard deviation of the variable's natural logarithm (by definition, the variable's logarithm is normally distributed), which means

$$X = e^{\mu+\sigma Z} \tag{17.9.1}$$

with Z a standard normal variable.
This relationship is true regardless of the base of the logarithmic or exponential function. If $\log_a(Y)$ is normally distributed, then so is $\log_b(Y)$, for any two positive numbers a, $b \neq 1$. Likewise, if e^X is log-normally distributed, then so is a^X, where is a positive number $\neq 1$.
On a logarithmic scale, μ and σ can be called the location parameter and the scale parameter, respectively.
In contrast, the mean, standard deviation, and variance of the non-logarithmized sample values are respectively denoted m, $s.d.$, and v in this article. The two sets of parameters can be related as

$$\mu = \ln\left(\frac{m^2}{\sqrt{v+m^2}}\right), \quad \sigma = \sqrt{\ln\left(1+\frac{v}{m^2}\right)} \tag{17.9.2}$$

17.9.2 Density and CDF

Function **LogNormalDist**(***x*** *As mpNum*, ***mean*** *As mpNum*, ***stdev*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function LogNormalDist returns returns pdf, CDF and related information for the Lognormal-distribution

Parameters:
x: A real number
mean: A real number greater 0, representing the mean of the distribution
stdev: A real number greater 0, representing the standard deviation of the distribution
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.2.2.1 and 17.2.2.2.

17.9.2.1 Density

$$f(x) = \frac{1}{bx\sqrt{2\pi}} \exp\left(-\frac{(\ln(x)-a)^2}{2b^2}\right) \tag{17.9.3}$$

17.9.2.2 CDF

$$F(x) = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{\ln(x)-a}{b\sqrt{2}}\right)\right) \tag{17.9.4}$$

17.9.3 Quantiles

Function **LognormalDistInv**(***Prob*** *As mpNum*, ***mean*** *As mpNum*, ***stdev*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function LognormalDistInv returns returns quantiles and related information for the the Lognormal-distribution

Parameters:
Prob: A real number between 0 and 1.
mean: A real number greater 0, representing the mean of the distribution
stdev: A real number greater 0, representing the standard deviation of the distribution
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*).

17.9.3.1 Quantiles: algorithms and formulas

$$F^{-1}(y) = \exp(a + b \cdot \text{normstdinv}(y)) \tag{17.9.5}$$

17.9.4 Properties

Function **LognormalDistInfo**(***mean*** *As mpNum*, ***stdev*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function LognormalDistInfo returns returns moments and related information for the central Lognormal-distribution

Parameters:
mean: A real number greater 0, representing the mean of the distribution
stdev: A real number greater 0, representing the standard deviation of the distribution
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given in section 17.13.4.

17.9.4.1 Moments: algorithms and formulas

Algebraic moments of the log-normal distribution are given by

$$\mu'_k = e^{k\mu + k^2\sigma^2/2} \tag{17.9.6}$$

17.9.5 Random Numbers

Function **LognormalRandom**(***Size*** *As mpNum*, ***mean*** *As mpNum*, ***stdev*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function LognormalRandom returns returns random numbers following a central Beta-distribution

Parameters:
Size: A positive integer up to 10^7
mean: A real number greater 0, representing the mean of the distribution
stdev: A real number greater 0, representing the standard deviation of the distribution
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given in section 17.6.5.

17.9.5.1 Random Numbers: algorithms and formulas

The most straightforward way of achieving random numbers from a log-normal distribution is to generate a random number u from a normal distribution with mean μ and standard deviation σ and construct $r = e^u$.

17.10 Negative Binomial Distribution

These functions return PMF and CDF of the (discrete) negative binomial distribution with target for number of successful trials $r > 0$ and success probability $0 \le p \le 1$.
If $r = n$ is a positive integer the name Pascal distribution is used, and for $r = 1$ it is called geometric distribution.
See Ong & Lee (1979) for information on the noncentral negative binomial distribution

17.10.1 Density and CDF

Function **NegativeBinomialDist**(***x*** *As mpNum*, ***r*** *As mpNum*, ***p*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function NegativeBinomialDist returns returns pdf, CDF and related information for the central negative binomial distribution

Parameters:
x: The number of failures in trials.
r: The threshold number of successes.
p: The probability of a success
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.3.1.1 and 17.3.1.2.

17.10.1.1 Density

$$f_{\text{NegBin}}(r,k;p) = \frac{\Gamma(k+r)}{k!\Gamma(r)}p^r(1-p)^k = \frac{p}{r+k}f_{\text{Beta}}(r,k+1,p) \tag{17.10.1}$$

17.10.1.2 CDF

$$F_{\text{NegBin}}(r,k;p) = I_{1-p}(r,k+1) = ibeta(r,k+1,1-p) \tag{17.10.2}$$

17.10.2 Quantiles

Function **NegativeBinomialDistInv**(***Prob*** *As mpNum*, ***r*** *As mpNum*, ***p*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function NegativeBinomialDistInv returns returns quantiles and related information for the the central binomial-distribution

Parameters:
Prob: A real number between 0 and 1.
r: The threshold number of successes.
p: The probability of a success
Output: A string describing the output choices

See section 17.1.3.2 for the options for *Prob* and *Output*).

17.10.3 Properties

Function **NegativeBinomialDistInfo**(***r*** *As mpNum*, ***p*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function NegativeBinomialDistInfo returns returns moments and related information for the central Binomial-distribution

Parameters:

r: The threshold number of successes.
p: The probability of a success
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given in section 17.13.4.

17.10.3.1 Moments: algorithms and formulas

$$\mu_1 = np \tag{17.10.3}$$

$$\mu_2 = np(1-p) = npq \tag{17.10.4}$$

$$\mu_3 = npq(q+p) \tag{17.10.5}$$

$$\mu_4 = npq(3npq + 6pq + 1) \tag{17.10.6}$$

17.10.3.2 Recurrence relations

The following recurrence relations hold:

$$f_{\text{NegBin}}(r, k+1; p) = \frac{(r+k)(1-p)}{k+1} f_{\text{NegBin}}(r, k; p) \tag{17.10.7}$$

$$f_{\text{NegBin}}(r, k-1; p) = \frac{k}{(r+k-1)(1-p)} f_{\text{NegBin}}(r, k; p) \tag{17.10.8}$$

17.10.4 Random Numbers

Function **NegativeBinomialDistRandom**(***Size*** *As mpNum*, ***r*** *As mpNum*, ***p*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function NegativeBinomialDistRandom returns returns random numbers following a central Binomial-distribution

Parameters:
Size: A positive integer up to 10^7
r: The threshold number of successes.
p: The probability of a success
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given below.

17.10.4.1 Random Numbers: algorithms and formulas

Random numbers from a negative binomial distribution can be obtained using the algorithms outline for the beta distribution.

17.11 Normal Distribution

17.11.1 Definition

A random variable is said to follow a normal distribution with mean μ and variance σ^2, if its pdf is given by 17.11.1. It is said to follow a standardized normal distribution if its pdf is given by 17.11.2.

17.11.2 Density and CDF

Function **NDist**(***x*** *As mpNum*, ***mean*** *As mpNum*, ***stdev*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function NDist returns returns pdf, CDF and related information for the normal-distribution

Parameters:
x: A real number
mean: A real number greater 0, representing the mean of the distribution
stdev: A real number greater 0, representing the standard deviation of the distribution
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.2.2.1 and 17.2.2.2.

17.11.2.1 Density

This functions returns the pdf of the normal distribution with mean μ and variance σ^2, which is given by

$$f_N(x;\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{17.11.1}$$

The pdf of the standardized normal distribution with mean 0 and variance 1 is given by

$$\phi(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2}, \tag{17.11.2}$$

These two functions are related by

$$f_N(x;\mu,\sigma^2) = \frac{1}{\sigma}\phi\left(\frac{x-\mu}{\sigma}\right), \text{and} \quad \phi(u) = \sigma f_N(\mu+\sigma u) \tag{17.11.3}$$

17.11.2.2 CDF

This functions returns the cdf of the normal distribution with mean μ and variance σ^2, which is given by

$$F_N(x;\mu,\sigma^2) = \int_{-\infty}^{x} f_N(v)dv \tag{17.11.4}$$

The cdf of the standardized normal distribution with mean 0 and variance 1 is given by

$$\Phi(u) = \int_{-\infty}^{u} \phi(w)dw \tag{17.11.5}$$

These two functions are related by

$$F_N(x;\mu,\sigma^2) = \Phi\left(\frac{x-\mu}{\sigma}\right), \text{and} \quad \Phi(u) = F_N(\mu+\sigma u) \tag{17.11.6}$$

17.11.3 Quantiles

These functions return the quantile of the normal distribution with mean μ and variance σ^2, $F_N^{-1}(\alpha;\mu,\sigma^2)$, or the standardized normal distribution with mean 0 and variance 1, $\Phi^{-1}(\alpha)$.

Function **NDistInv**(***Prob*** *As mpNum*, ***mean*** *As mpNum*, ***stdev*** *As mpNum*, ***Output*** *As String*) As mpNumList

---

The function NDistInv returns returns quantiles and related information for the the Lognormal-distribution

Parameters:
Prob: A real number between 0 and 1.
mean: A real number greater 0, representing the mean of the distribution
stdev: A real number greater 0, representing the standard deviation of the distribution
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*).

17.11.3.1 Quantiles: algorithms and formulas

$$F^{-1}(y) = \exp(a + b \cdot \text{normstdinv}(y)) \tag{17.11.7}$$

17.11.4 Properties

Function **NormalDistInfo**(***mean*** *As mpNum*, ***stdev*** *As mpNum*, ***Output*** *As String*) As mpNumList

---

The function NormalDistInfo returns returns moments and related information for the central Lognormal-distribution

Parameters:
mean: A real number greater 0, representing the mean of the distribution
stdev: A real number greater 0, representing the standard deviation of the distribution
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given in section 17.13.4.

17.11.4.1 Moments: algorithms and formulas

$\kappa_1 = \mu$
$\kappa_2 = \sigma^2$
$\kappa_r = 0$ for $r \geq 3$.

17.11.4.2 Differential Equation

Let $Z^{(m)}$ denote the m^{th} derivative of $Z(x)$. Then (Abramowitz & Stegun., 1970)

$$Z^{(1)} = -xZ(x) \tag{17.11.8}$$

$$Z^{(m+2)} + xZ^{(m+1)} + (m+1)Z^{(m)} = 0 \tag{17.11.9}$$

17.11.5 Random Numbers

Function **NormalRandom**(***Size*** *As mpNum*, ***mean*** *As mpNum*, ***stdev*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function NormalRandom returns returns random numbers following a central Beta-distribution

Parameters:
Size: A positive integer up to 10^7
mean: A real number greater 0, representing the mean of the distribution
stdev: A real number greater 0, representing the standard deviation of the distribution
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given in section 17.6.5.

17.11.5.1 Random Numbers: algorithms and formulas

Let $Z_1 \sim Re(0;1), Z_2 \sim Re(0,1)$ be independent random variables. Then

$X_1 = \sqrt{-2\ln Z_1}\cos(2\pi Z_2)$ and $X_2 = \sqrt{-2\ln Z_1}\sin(2\pi Z_2)$ are $\sim No(0;1)$.

It is also possible to directly use $\Phi^{-1}(\alpha)$.

17.12 Poisson Distribution

17.12.1 Definition

The Poisson distribution is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time and/or space if these events occur with a known average rate and independently of the time since the last event. The following functions return PMF and CDF of the Poisson distribution with mean $\mu \geq 0$.

17.12.2 Density and CDF

Function **PoissonDist**(***x*** *As mpNum*, ***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function PoissonDist returns returns pdf, CDF and related information for the Poisson distribution

Parameters:
x: A real number
lambda: A real number greater 0, representing the degrees of freedom

Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.4.2.1 and 17.4.2.2.

17.12.2.1 Density

$$f(k) = \frac{\mu^k}{k!} e^{-\mu} = sfcIgprefix(1 + k, \mu) \tag{17.12.1}$$

17.12.2.2 CDF

$$F(k) = e^{-\mu} \sum_{i=0}^{k} \frac{\mu^i}{i!} = igammaq(1 + k, \mu) \tag{17.12.2}$$

17.12.3 Quantiles

Function **PoissonDistInv**(***Prob*** *As mpNum*, ***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function PoissonDistInv returns quantiles and related information for the the Poisson distribution

Parameters:
Prob: A real number between 0 and 1.
lambda: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*). Algorithms and formulas are given in section 17.12.3.1.

17.12.3.1 Quantiles: algorithms and formulas

The algorithms follow the one for the chisquare distribution.

17.12.4 Properties

Function **PoissonDistInfo**(***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function PoissonDistInfo returns moments and related information for the Poisson distribution

Parameters:
lambda: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given in section 17.4.4.

17.12.4.1 Moments and Cumulants

The momemts and cumulants are given by

$$\kappa_r = \lambda \tag{17.12.3}$$

$$\mu_1 = \mu_2 = \mu_3 = \lambda \tag{17.12.4}$$

$$\mu_4 = 3\lambda^2 + \lambda \tag{17.12.5}$$

17.12.5 Random Numbers

Function **PoissonDistRan**(***Size*** *As mpNum*, ***lambda*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function PoissonDistRan returns random numbers following a Poisson distribution

Parameters:
Size: A positive integer up to 10^7
lambda: A real number greater 0, representing the degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given in section 17.4.5.

17.13 Student's t-Distribution

17.13.1 Definition

If X is a random variable following a normal distribution with mean zero and variance unity and χ^2 is a random variable following an independent χ^2-distribution with n degrees of freedom, then the distribution of the ratio $\frac{X}{\sqrt{\chi^2/n}}$ is called Student's t-distribution with n degrees of freedom

17.13.2 Density and CDF

Function **TDist**(***x*** *As mpNum*, ***n*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function TDist returns returns pdf, CDF and related information for the central t-distribution

Parameters:
x: A real number
n: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.13.2.1 and 17.13.2.2.

17.13.2.1 Density

The density of a variable following a central Student's t-distribution with n degrees of freedom is given by

$$f_t(n,x) = \frac{\Gamma((n+1)/2)}{\sqrt{n\pi}\Gamma(n/2)}\left(\frac{n}{n+t^2}\right)^{(n+1)/2} \tag{17.13.1}$$

where $\Gamma(\cdot)$ denotes the Gamma function (see section 6.7.9.)

17.13.2.2 CDF: General formulas

The cdf of a variable following a central t-distribution with n degrees of freedom is defined as

$$\Pr[X \leq x] = F_t(n, x) = \int_0^x f_t(n, t)dt \tag{17.13.2}$$

The cdf of the central t-distribution is calculated for any positive degrees of freedom n using the relationships

$$2F_t(n, x) = F_F(1, n; x^2), \quad x \leq 0 \tag{17.13.3}$$

$$F_t(n, x) - F_t(n, -x) = F_F(1, n; x^2), \quad x \geq 0 \tag{17.13.4}$$

$$F_t(n, x) = 1 - F_t(n, -x) \tag{17.13.5}$$

where $F_F(1, n, x^2)$ denotes the cdf of the central F-distribution with 1 and n of freedom (see section 17.6.2.2).

17.13.3 Quantiles

Function **TDistInv**(***Prob*** *As mpNum*, ***n*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function TDistInv returns returns quantiles and related information for the the central t-distribution

Parameters:
Prob: A real number between 0 and 1.
n: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*).

17.13.4 Properties

Function **TDistInfo**(***n*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function TDistInfo returns returns moments and related information for the central t-distribution

Parameters:
n: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given in section 17.13.4.

17.13.4.1 Moments: algorithms and formulas

The algebraic moments (defined for $n > r$) are given by

$$\mu'_r = \left(\tfrac{1}{2}n\right)^{r/2} \frac{\Gamma\left(\frac{1}{2}(n-r)\right)}{\Gamma\left(\frac{1}{2}n\right)}. \tag{17.13.6}$$

17.13.5 Random Numbers

Function **TDistRan**(***Size*** *As mpNum*, ***n*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function TDistRan returns returns random numbers following a central t-distribution

Parameters:
Size: A positive integer up to 10^7
n: A real number greater 0, representing the degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given in section 17.13.5.

17.13.5.1 Random Numbers: algorithms and formulas

Following the definition we may define a random number t from a t-distribution, using random numbers from a normal and a chi-square distribution, as $t = \frac{z}{\sqrt{y_n/n}}$, where z is a standard normal and y_n a chi-squared variable with n degrees of freedom. To obtain random numbers from these distributions see the appropriate sections.

17.13.6 Behrens-Fisher Problem

See Golhar (1972)

17.14 Weibull Distribution

These functions return PDF, CDF, and ICDF of the Weibull distribution with shape parameter a and scale $b > 0$ and the support interval $(0, +\infty)$:

17.14.1 Density and CDF

Function **WeibullDist**(***x*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function WeibullDist returns returns pdf, CDF and related information for the Weibull distribution

Parameters:
x: A real number
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.2.2.1 and 17.2.2.2.

17.14.1.1 Density

$$f(x) = \frac{x}{b^2} \exp\left(-\frac{x^2}{2b^2}\right) \exp(-(x/b)^a) \tag{17.14.1}$$

17.14.1.2 CDF

$$F(x) = 1 - \exp\left(-(x/b)^a\right) = -\text{expm1}\left(-(x/b)^a\right) \tag{17.14.2}$$

17.14.2 Quantiles

Function **WeibullDistInv**(***Prob*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function WeibullDistInv returns returns quantiles and related information for the the central Beta-distribution

Parameters:
Prob: A real number between 0 and 1.
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*).

$$F^{-1}(y) = b(-\text{ln1p}(-y))^{1/a} \tag{17.14.3}$$

17.14.3 Properties

Function **WeibullDistInfo**(***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function WeibullDistInfo returns returns moments and related information for the central Beta-distribution

Parameters:
a: A real number greater 0, representing the degrees of freedom
b: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given in section 17.13.4.

17.14.3.1 Moments: algorithms and formulas

$$\mu_r' = \sum_{j=0}^{r} \binom{r}{j} \Gamma\left(\frac{r-j}{c} + 1\right) b^{r-j} \tag{17.14.4}$$

$$\mu_1 = b\Gamma\left(\frac{1}{c} + 1\right) \tag{17.14.5}$$

$$\mu_2 = b^2 \left[\Gamma\left(\frac{1}{c} + 1\right) \Gamma^2\left(\frac{1}{c} + 1\right)\right] \tag{17.14.6}$$

See Rinne (2008) for further details.

17.14.4 Random Numbers

Function **WeibullDistRandom**(***Size*** *As mpNum,* ***a*** *As mpNum,* ***b*** *As mpNum,* ***Generator*** *As String,* ***Output*** *As String*) As mpNumList

The function WeibullDistRandom returns returns random numbers following a central Beta-distribution

Parameters:
Size: A positive integer up to 10^7
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given in section 17.6.5.

17.15 Bernoulli Distribution

The Bernoulli distribution is a discrete distribution of the outcome of a single trial with only two results, 0 (failure) or 1 (success), with a probability of success p. The Bernoulli distribution is the simplest building block on which other discrete distributions of sequences of independent Bernoulli trials can be based. The Bernoulli is the binomial distribution ((k = 1, p)) with only one trial.

17.15.1 Density and CDF

Function **BernoulliDistBoost**(***k*** *As mpNum,* ***p*** *As mpNum,* ***Output*** *As String*) As mpNumList

The function BernoulliDistBoost returns returns pdf, CDF and related information for the central t-distribution

Parameters:
k: A real number, 0 or 1
p: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.15.1.1 and 17.15.1.2.

17.15.1.1 Density

$$f(x) = \begin{cases} q = 1 - p & \text{for } k = 0 \\ p & \text{for } k = 1. \end{cases} \tag{17.15.1}$$

17.15.1.2 CDF

$$F(x) = \begin{cases} 0 & \text{for } k = 0 \\ q & \text{for } k = 0 \\ 1 & \text{for } k = 1. \end{cases} \tag{17.15.2}$$

17.15.2 Quantiles

Function **BernoulliDistInvBoost**(***Prob*** *As mpNum*, ***p*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function BernoulliDistInvBoost returns returns quantiles and related information for the the central t-distribution

Parameters:
Prob: A real number between 0 and 1.
p: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*).

17.15.2.1 Quantiles: Algorithm

Using the relation: cdf $= 1 - p$ for $k = 0$, else 1.

17.15.3 Properties

Function **BernoulliDistInfoBoost**(***p*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function BernoulliDistInfoBoost returns returns moments and related information for the central t-distribution

Parameters:
p: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given in section 17.15.3.

17.15.3.1 Moments: algorithms and formulas

$$\mu_r^{'} = \sum_{i=0}^{r-1} \binom{r}{i} (-1)^i p^{i+1} + (-p)^r \tag{17.15.3}$$

$$\mu_1 = p \tag{17.15.4}$$

$$\mu_2 = pq \tag{17.15.5}$$

$$\mu_3 = pq(1 - 2p) \tag{17.15.6}$$

$$\mu_4 = pq(1 - 3pq) \tag{17.15.7}$$

17.15.4 Random Numbers

Function **BernoulliDistRandomBoost**(***Size*** *As mpNum*, ***p*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function BernoulliDistRandomBoost returns returns random numbers following a central Binomial-distribution

Parameters:
Size: A positive integer up to 10^7
p: The probability of a success on each trial.
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given below.

17.16 Cauchy Distribution

17.16.1 Density and CDF

Function **CauchyDistBoost**(***x*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function CauchyDistBoost returns returns pdf, CDF and related information for the Cauchy distribution

Parameters:
x: A real number
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*.

17.16.1.1 Density

$$f(x) = \frac{1}{\pi(1 + ((x-a)/b)^2)} \tag{17.16.1}$$

17.16.1.2 CDF

$$F(x) = \frac{1}{2} + \frac{1}{\pi}\arctan\left(\frac{x-a}{b}\right) \tag{17.16.2}$$

17.16.2 Quantiles

Function **CauchyDistInvBoost**(***Prob*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function CauchyDistInvBoost returns returns quantiles and related information for the Cauchy distribution

Parameters:
Prob: A real number between 0 and 1.
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*). Algorithms and formulas are given below.

$$F^{-1}(y) = \begin{cases} a - b/\tan(\pi y), & y < 0.5, \\ a, & y = 0.5, \\ a - b/\tan(\pi(1-y)) & y > 0.5. \end{cases} \tag{17.16.3}$$

17.16.3 Properties

Function **CauchyDistInfoBoost**(***a*** *As mpNum,* ***b*** *As mpNum,* ***Output*** *As String*) As mpNumList

The function CauchyDistInfoBoost returns returns moments and related information for the Cauchy distribution

Parameters:
a: A real number greater 0, representing the degrees of freedom
b: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*.
All the usual non-member accessor functions that are generic to all distributions are supported: Cumulative Distribution Function, Probability Density Function, Quantile, Hazard Function, Cumulative Hazard Function, mean, median, mode, variance, standard deviation, skewness, kurtosis, kurtosis_excess, range and support. Note however that the Cauchy distribution does not have a mean, standard deviation, etc. See mathematically undefined function to control whether these should fail to compile with a BOOST_STATIC_ASSERTION_FAILURE, which is the default. Alternately, the functions mean, standard deviation, variance, skewness, kurtosis and kurtosis_excess will all return a domain_error if called.

17.16.4 Random Numbers

Function **CauchyDistRandomBoost**(***Size*** *As mpNum,* ***a*** *As mpNum,* ***b*** *As mpNum,* ***Generator*** *As String,* ***Output*** *As String*) As mpNumList

The function CauchyDistRandomBoost returns returns random numbers following a Cauchy distribution

Parameters:
Size: A positive integer up to 10^7
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given in below.

17.17 Extreme Value (or Gumbel) Distribution

These functions return PDF, CDF, and ICDF of the Extreme Value Type I distribution with location a, scale $b > 0$, and the support interval $(-\infty, +\infty)$:

17.17.1 Density and CDF

Function **ExtremevalueDistBoost**(***x*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function ExtremevalueDistBoost returns returns pdf, CDF and related information for the Extreme Value distribution

Parameters:
x: A real number
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.17.1.1 and 17.17.1.2.

17.17.1.1 Density

$$f(x) = \frac{e^{-(x-a)/b}}{b} e^{e^{-(x-a)/b}} \tag{17.17.1}$$

17.17.1.2 CDF

$$F(x) = e^{e^{-(x-a)/b}} \tag{17.17.2}$$

17.17.2 Quantiles

Function **ExtremevalueDistInvBoost**(***Prob*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function ExtremevalueDistInvBoost returns returns quantiles and related information for the the Extreme Value distribution

Parameters:
Prob: A real number between 0 and 1.
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*). Algorithms and formulas are given below.

$$F^{-1}(y) = a - \ln(-\ln(y)) \tag{17.17.3}$$

17.17.3 Properties

Function **ExtremevalueDistInfoBoost**(***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function ExtremevalueDistInfoBoost returns returns moments and related information for the Extreme Value distribution

Parameters:
a: A real number greater 0, representing the degrees of freedom
b: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*.

17.17.4 Random Numbers

Function **ExtremevalueDistRandomBoost**(***Size*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function ExtremevalueDistRandomBoost returns returns random numbers following a Extreme Value distribution

Parameters:
Size: A positive integer up to 10^7
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given in below.

17.18 Geometric Distribution

Geometric distribution: it is used when there are exactly two mutually exclusive outcomes of a Bernoulli trial: these outcomes are labelled "success" and "failure". For Bernoulli trials each with success fraction p, the geometric distribution gives the probability of observing k trials (failures, events, occurrences, or arrivals) before the first success.

17.18.1 Density and CDF

Function **GeometricDistBoost**(***k*** *As mpNum*, ***p*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function GeometricDistBoost returns returns pdf, CDF and related information for the Geometric distribution

Parameters:

k: A real number
p: A real number greater 0, representing the numerator degrees of freedom
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.18.1.1 and 17.18.1.2.

17.18.1.1 Density

$$f(k;p) = p(1-p)^k \tag{17.18.1}$$

17.18.1.2 CDF

$$F(k;p) = 1-(1-p)^{k+1} \tag{17.18.2}$$

17.18.2 Quantiles

Function **GeometricDistInvBoost**(***Prob*** *As mpNum*, ***p*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function GeometricDistInvBoost returns returns quantiles and related information for the Geometric distribution

Parameters:
Prob: A real number between 0 and 1.
p: A real number greater 0, representing the numerator degrees of freedom
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*). Algorithms and formulas are given below.

$$F^{-1}(x;p) = \frac{\mathrm{log1p}(-x)}{\mathrm{log1p}(-p)} - 1 \tag{17.18.3}$$

17.18.3 Properties

Function **GeometricDistInfoBoost**(***p*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function GeometricDistInfoBoost returns returns moments and related information for the Geometric distribution

Parameters:
p: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*.

17.18.4 Random Numbers

Function **GeometricDistRandomBoost**(***Size*** *As mpNum*, ***p*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function GeometricDistRandomBoost returns returns random numbers following a Geometric distribution

Parameters:
Size: A positive integer up to 10^7
p: A real number greater 0, representing the denominator degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given in below.

17.19 Inverse Chi Squared Distribution

17.19.1 Definition

The inverse-chi-squared distribution (or inverted-chi-square distribution[1]) is the probability distribution of a random variable whose multiplicative inverse (reciprocal) has a chi-squared distribution. It is also often defined as the distribution of a random variable whose reciprocal divided by its degrees of freedom is a chi-squared distribution. That is, if X has the chi-squared distribution with ν degrees of freedom, then according to the first definition, $1/X$ has the inverse-chi-squared distribution with ν degrees of freedom; while according to the second definition, ν/X has the inverse-chi-squared distribution with ν degrees of freedom.
The inverse-chi-squared distribution is a special case of a inverse-gamma distribution with ν (degrees of freedom), shape (α) and scale (β), where $\alpha = \nu/2$ and $\beta = 1/2$.

17.19.2 Density and CDF

Function **InverseChiSquaredDistBoost**(***x*** *As mpNum,* ***n*** *As mpNum,* ***Output*** *As String*) As mpNumList

---

The function InverseChiSquaredDistBoost returns returns pdf, CDF and related information for the inverse-chi-squared -distribution

Parameters:
x: A real number
n: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*.

17.19.2.1 Density

The first definition yields a probability density function given by

$$f(x;\nu) = \frac{2^{-\nu/2}}{\Gamma(\nu/2)} x^{-\nu/2-1} e^{-1/(2x)} \tag{17.19.1}$$

while the second definition yields the density function

$$f(x;\nu) = \frac{(\nu/2)^{\nu/2}}{\Gamma(\nu/2)} x^{-\nu/2-1} e^{-\nu/(2x)} \tag{17.19.2}$$

In both cases, $x > 0$ and ν is the degrees of freedom parameter. Further, Γ is the gamma function. Both definitions are special cases of the scaled-inverse-chi-squared distribution. For the first definition the variance of the distribution is $\sigma = 1/\nu$, while for the second definition $\sigma = 1$.

17.19.2.2 CDF

$$F(x;\nu) = \frac{1}{\Gamma(\nu/2)}\Gamma\left(\frac{\nu}{2}, \frac{1}{2x}\right) \tag{17.19.3}$$

17.19.3 Quantiles

Function **InverseChiSquaredDistInvBoost**(***Prob*** *As mpNum*, ***n*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function InverseChiSquaredDistInvBoost returns quantiles and related information for the inverse-chi-squared distribution

Parameters:
Prob: A real number between 0 and 1.
n: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*).

$$F^{-1}(prob;\nu) = \beta//gamma - q - inv(\alpha, p) \tag{17.19.4}$$

17.19.4 Properties

Function **InverseChiSquaredDistInfoBoost**(***n*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function InverseChiSquaredDistInfoBoost returns moments and related information for the inverse-chi-squared distribution

Parameters:
n: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given below.

17.19.4.1 Moments and Cumulants

$$\mu_1 = \frac{\nu}{\nu - 2} \text{for } \nu > 2. \tag{17.19.5}$$

17.19.5 Random Numbers

Function **InverseChiSquaredDistRanBoost**(***Size*** *As mpNum*, ***n*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function InverseChiSquaredDistRanBoost returns random numbers following a inverse-chi-squared distribution

Parameters:
Size: A positive integer up to 10^7

n: A real number greater 0, representing the degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given below.

17.20 Inverse Gamma Distribution

17.20.1 Definition

In probability theory and statistics, the inverse gamma distribution is a two-parameter family of continuous probability distributions on the positive real line, which is the distribution of the reciprocal of a variable distributed according to the gamma distribution. Perhaps the chief use of the inverse gamma distribution is in Bayesian statistics, where the distribution arises as the marginal posterior distribution for the unknown variance of a normal distribution if an uninformative prior is used; and as an analytically tractable conjugate prior if an informative prior is required.
However, it is common among Bayesians to consider an alternative parametrization of the normal distribution in terms of the precision, defined as the reciprocal of the variance, which allows the gamma distribution to be used directly as a conjugate prior. Other Bayesians prefer to parametrize the inverse gamma distribution differently, as a scaled inverse chi-squared distribution

17.20.2 Density and CDF

Function **InverseGammaDistBoost**(***x*** *As mpNum,* ***a*** *As mpNum,* ***b*** *As mpNum,* ***Output*** *As String*) As mpNumList

The function InverseGammaDistBoost returns returns pdf, CDF and related information for the inverse gamma distribution

Parameters:
x: A real number
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.20.2.1 and 17.20.2.2.

17.20.2.1 Density

The inverse gamma distribution's probability density function is defined over the support $x > 0$

$$f(x;\alpha,\beta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} x^{-\alpha-1} \exp\left(-\frac{\beta}{x}\right) \tag{17.20.1}$$

with shape parameter α and scale parameter β.

17.20.2.2 CDF

The cumulative distribution function is the regularized gamma function

$$F(x;\alpha,\beta) = \frac{\Gamma(\alpha),\beta/x}{\Gamma(\alpha)} = Q\left(\alpha,-\frac{\beta}{x}\right) \tag{17.20.2}$$

where the numerator is the upper incomplete gamma function and the denominator is the gamma function. Many math packages allow you to compute Q, the regularized gamma function, directly.

17.20.3 Quantiles

Function **InverseGammaDistInvBoost**(***Prob*** *As mpNum*, ***m*** *As mpNum*, ***n*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function InverseGammaDistInvBoost returns returns quantiles and related information for the the inverse gamma distribution

Parameters:
Prob: A real number between 0 and 1.
m: A real number greater 0, representing the numerator degrees of freedom
n: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*). Algorithms and formulas are given below.

$$F^{-1}(prob;\nu) = \beta//gamma - q - inv(\alpha,p) \tag{17.20.3}$$

17.20.4 Properties

Function **InverseGammaDistInfoBoost**(***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function InverseGammaDistInfoBoost returns returns moments and related information for the inverse gamma distribution

Parameters:
a: A real number greater 0, representing the degrees of freedom
b: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given below.

17.20.4.1 Moments and Cumulants

$$\mu_1 = \frac{\nu}{\nu-2} \text{for } \nu > 2. \tag{17.20.4}$$

17.20.5 Random Numbers

Function **InverseGammaDistRanBoost**(***Size*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function InverseGammaDistRanBoost returns returns random numbers following a inverse gamma distribution

Parameters:
Size: A positive integer up to 10^7
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given below.

17.21 Inverse Gaussian (or Wald) Distribution

17.21.1 Definition

In probability theory, the inverse Gaussian distribution (also known as the Wald distribution) is a two-parameter family of continuous probability distributions with mean μ and shape parameter λ and support on $(0, \infty)$. As λ tends to infinity, the inverse Gaussian distribution becomes more like a normal (Gaussian) distribution. The inverse Gaussian distribution has several properties analogous to a Gaussian distribution. The name can be misleading: it is an "inverse" only in that its cumulant generating function (logarithm of the characteristic function) is the inverse of the cumulant generating function of a Gaussian random variable.
While the Gaussian describes a Brownian Motion's level at a fixed time, the inverse Gaussian describes the distribution of the time a Brownian Motion with positive drift takes to reach a fixed positive level.
See also http://en.wikipedia.org/wiki/Inverse_Gaussian_distribution.

17.21.2 Density and CDF

Function **InverseGaussianDistBoost**(***x*** *As mpNum*, ***mu*** *As mpNum*, ***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function InverseGaussianDistBoost returns returns pdf, CDF and related information for the inverse Gaussian distribution

Parameters:
x: A real number
mu: A real number greater 0, representing the numerator degrees of freedom
lambda: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.21.2.1 and 17.21.2.2.

17.21.2.1 Density

$$f(x;\mu,\lambda) = \sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(\frac{-\lambda(x-\mu)^2}{2\mu^2 x}\right) \tag{17.21.1}$$

17.21.2.2 CDF

$$F(x;\mu,\lambda) = \Phi\left(\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu}-1\right)\right) + \exp\left(\frac{2\lambda}{\mu}\right)\Phi\left(-\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu}+1\right)\right) \tag{17.21.2}$$

17.21.3 Quantiles

Function **InverseGaussianDistInvBoost**(***Prob*** *As mpNum*, ***mu*** *As mpNum*, ***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function InverseGaussianDistInvBoost returns returns quantiles and related information for the the inverse Gaussian distribution

Parameters:
Prob: A real number between 0 and 1.
mu: A real number greater 0, representing the numerator degrees of freedom
lambda: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*). Algorithms and formulas are given below.

$$F^{-1}(prob;\nu) = \beta//gamma - q - inv(\alpha,p) \tag{17.21.3}$$

17.21.4 Properties

Function **InverseGaussianDistInfoBoost**(***mu*** *As mpNum*, ***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function InverseGaussianDistInfoBoost returns returns moments and related information for the inverse Gaussian distribution

Parameters:
mu: A real number greater 0, representing the degrees of freedom
lambda: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given below.

17.21.4.1 Moments and Cumulants

$$\mu_1 = \frac{\nu}{\nu-2} \text{for } \nu > 2. \tag{17.21.4}$$

17.21.5 Random Numbers

Function **InverseGaussianDistRanBoost**(***Size*** *As mpNum*, ***mu*** *As mpNum*, ***lambda*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function InverseGaussianDistRanBoost returns returns random numbers following a inverse Gaussian distribution

Parameters:
Size: A positive integer up to 10^7
mu: A real number greater 0, representing the numerator degrees of freedom
lambda: A real number greater 0, representing the denominator degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given below.

17.22 Laplace Distribution

These functions return PDF, CDF, and ICDF of the Laplace distribution with location a, scale $b > 0$, and the support interval $(-\infty, +\infty)$:

17.22.1 Density and CDF

Function **LaplaceDistBoost**(***x*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function LaplaceDistBoost returns returns pdf, CDF and related information for the Laplace distribution

Parameters:
x: A real number
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.22.1.1 and 17.22.1.2.

17.22.1.1 Density

$$f(x) = \exp(-|x-a|/b)/(2b) \tag{17.22.1}$$

17.22.1.2 CDF

$$F(x) = \begin{cases} \frac{1}{2} - \frac{1}{2}\text{expm1}\left(-\frac{x-a}{b}\right) & x \geq a \\ \frac{1}{2}\exp\left(-\frac{x-a}{b}\right) & x < a. \end{cases} \tag{17.22.2}$$

17.22.2 Quantiles

Function **LaplaceDistInvBoost**(***Prob*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function LaplaceDistInvBoost returns returns quantiles and related information for the the Laplace distribution

Parameters:
Prob: A real number between 0 and 1.
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*). Algorithms and formulas are given below.

$$F^{-1}(y) = \begin{cases} a + b\ln(2y), & y < 0.5, \\ a - b\ln(2(1-y)) & y > 0.5. \end{cases} \tag{17.22.3}$$

17.22.3 Properties

Function **LaplaceDistInfoBoost**(***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function LaplaceDistInfoBoost returns returns moments and related information for the Laplace distribution

Parameters:
a: A real number greater 0, representing the degrees of freedom
b: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*.

17.22.4 Random Numbers

Function **LaplaceDistRanBoost**(***Size*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function LaplaceDistRanBoost returns returns random numbers following a Laplace distribution

Parameters:
Size: A positive integer up to 10^7
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given in below.

17.23 Logistic Distribution

17.23.1 Definition

These functions return PDF, CDF, and ICDF of the logistic distribution with location a, scale $b > 0$, and the support interval $(-\infty, +\infty)$:

17.23.2 Density and CDF

Function **LogisticDistBoost**(***x*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function LogisticDistBoost returns returns pdf, CDF and related information for the Logistic distribution

Parameters:
x: A real number
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.23.2.1 and 17.23.2.2.

17.23.2.1 Density

$$f(x) = \frac{1}{b} \frac{\exp\left(-\frac{x-a}{b}\right)}{\left(1 + \exp\left(-\frac{x-a}{b}\right)\right)^2} \tag{17.23.1}$$

17.23.2.2 CDF

$$F(x) = \frac{1}{1 + \exp\left(-\frac{x-a}{b}\right)} \tag{17.23.2}$$

17.23.3 Quantiles

Function **LogisticDistInvBoost**(***Prob*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function LogisticDistInvBoost returns returns quantiles and related information for the the Logistic distribution

Parameters:
Prob: A real number between 0 and 1.
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*). Algorithms and formulas are given below.

$$F^{-1}(y) = a - b \ln\left((1-y)/y\right) \tag{17.23.3}$$

17.23.4 Properties

Function **LogisticDistInfoBoost**(***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function LogisticDistInfoBoost returns returns moments and related information for the Logistic distribution

Parameters:
a: A real number greater 0, representing the degrees of freedom
b: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*.

17.23.5 Random Numbers

Function **LogisticDistRanBoost**(***Size*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function LogisticDistRanBoost returns returns random numbers following a Logistic distribution

Parameters:
Size: A positive integer up to 10^7
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given in below.

17.24 Pareto Distribution

17.24.1 Definition

These functions return PDF, CDF, and ICDF of the Pareto distribution with minimum (real) value $k > 0$, shape $a > 0$, and the support interval $(k, +\infty)$: This is a reference: Wikipedia contributors (2013)

17.24.2 Density and CDF

Function **ParetoDistBoost**(***x*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function ParetoDistBoost returns returns pdf, CDF and related information for the Pareto distribution

Parameters:
x: A real number
a: A real number greater 0, representing the numerator degrees of freedom

b: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.24.2.1 and 17.24.2.2.

17.24.2.1 Density

$$f(x) = \frac{a}{x}\left(\frac{k}{x}\right)^a \tag{17.24.1}$$

17.24.2.2 CDF

$$F(x) = 1 - \left(\frac{k}{x}\right)^a = -\text{powm1}(k/x, a) \tag{17.24.2}$$

17.24.3 Quantiles

Function **ParetoDistInvBoost**(***Prob*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function ParetoDistInvBoost returns returns quantiles and related information for the the Pareto distribution

Parameters:
Prob: A real number between 0 and 1.
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*). Algorithms and formulas are given below.

$$F^{-1}(y) = k(1-y)^{-1/a} \tag{17.24.3}$$

17.24.4 Properties

Function **ParetoDistInfoBoost**(***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function ParetoDistInfoBoost returns returns moments and related information for the Pareto distribution

Parameters:
a: A real number greater 0, representing the degrees of freedom
b: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*.

17.24.5 Random Numbers

Function **ParetoDistRanBoost**(***Size*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function ParetoDistRanBoost returns returns random numbers following a Pareto distribution

Parameters:
Size: A positive integer up to 10^7
a: A real number greater 0, representing the numerator degrees of freedom
b: A real number greater 0, representing the denominator degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given in below.

17.25 Raleigh Distribution

17.25.1 Definition

These functions return PDF, CDF, and ICDF of the Rayleigh distribution with scale $b > 0$ and the support interval $(0, +\infty)$:

17.25.2 Density and CDF

Function **RaleighDistBoost**(***x*** *As mpNum*, ***n*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function RaleighDistBoost returns returns pdf, CDF and related information for the Raleigh distribution

Parameters:
x: A real number
n: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.25.3 and 17.25.4.

17.25.3 Density

$$f(x) = \frac{x}{b^2} \exp\left(-\frac{x^2}{2b^2}\right) \tag{17.25.1}$$

17.25.4 CDF

$$F(x) = 1 - \exp\left(-\frac{x^2}{2b^2}\right) = -\text{expm1}\left(-\frac{x^2}{2b^2}\right) \tag{17.25.2}$$

17.25.5 Quantiles

Function **RaleighDistInvBoost**(***Prob*** *As mpNum*, ***n*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function RaleighDistInvBoost returns quantiles and related information for the Raleigh distribution

Parameters:
Prob: A real number between 0 and 1.
n: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*). Algorithms and formulas are below.

$$F^{-1}(y) = b\sqrt{-2 \cdot \ln1p(-y)} \tag{17.25.3}$$

17.25.6 Properties

Function **RaleighDistInfoBoost**(***n*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function RaleighDistInfoBoost returns moments and related information for the Raleigh distribution

Parameters:
n: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given below.

17.25.6.1 Moments and Cumulants

$$\mu_1 = s\sqrt{\pi/2} \tag{17.25.4}$$

17.25.7 Random Numbers

Function **RaleighDistRanBoost**(***Size*** *As mpNum*, ***n*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function RaleighDistRanBoost returns random numbers following a Raleigh distribution

Parameters:
Size: A positive integer up to 10^7
n: A real number greater 0, representing the degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given below.

17.26 Triangular Distribution

17.26.1 Definition

These functions return PDF, CDF, and ICDF of the triangular distribution on the support interval $[a, b]$ with finite $a < b$ and mode $c, a \leq c \leq b$.

17.26.2 Density and CDF

Function **TriangularDistBoost**(***x*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***c*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function TriangularDistBoost returns returns pdf, CDF and related information for the triangular distribution

Parameters:
x: A real number.
a: The left border parameter.
b: The right border parameter.
c: The mode parameter.
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.26.2.1 and 17.26.2.2.

17.26.2.1 Density

$$f(x) = \begin{cases} 0 & x < a \\ \frac{2(x-a)}{(b-a)(c-a)} & a \leq x < c \\ \frac{2}{b-a} & x = c \\ \frac{2(b-x)}{(b-a)(b-c)} & c < x \leq b \\ 0 & x > b \end{cases} \tag{17.26.1}$$

17.26.2.2 CDF

$$F(x) = \begin{cases} 0 & x < a \\ \frac{(x-a)^2}{(b-a)(c-a)} & a \leq x < c \\ \frac{c-a}{b-a} & x = c \\ 1 - \frac{(b-x)^2}{(b-a)(b-c)} & c < x \leq b \\ 1 & x > b \end{cases} \tag{17.26.2}$$

17.26.3 Quantiles

Function **TriangularDistInvBoost**(***Prob*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***c*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function TriangularDistInvBoost returns returns quantiles and related information for the the triangular distribution

Parameters:
Prob: A real number between 0 and 1.
a: The left border parameter.
b: The right border parameter.
c: The mode parameter.
Output: A string describing the output choices

See section 17.1.3.2 for the options for *Prob* and *Output*).

$$F^{-1}(y) = \begin{cases} a + \sqrt{(b-a)(c-a)y} & y < t \\ c & y = t \\ b - \sqrt{(b-a)(b-c)(1-y)} & y > t \end{cases} \tag{17.26.3}$$

where $t = (c-a)/(b-a)$.

17.26.4 Properties

Function **TriangularDistInfoBoost**(***a*** *As mpNum*, ***b*** *As mpNum*, ***c*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function TriangularDistInfoBoost returns returns moments and related information for the triangular distribution

Parameters:
a: The left border parameter.
b: The right border parameter.
c: The mode parameter.
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given below.

17.26.4.1 Moments

$$\mu_1 = \frac{a+b+c}{3} \tag{17.26.4}$$

$$\mu_2 = \frac{a^2+b^2+c^2-ab-ac-bc}{18} \tag{17.26.5}$$

$$\gamma_1 = \frac{\sqrt{2}(a+b-2c)(2a-b-c)(a-2b+c)}{5(a^2+b^2+c^2-ab-ac-bc)^{3/2}} \tag{17.26.6}$$

$$\gamma_2 = -\frac{3}{5} \tag{17.26.7}$$

17.26.5 Random Numbers

Function **TriangularDistRanBoost**(***Size*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***c*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function TriangularDistRanBoost returns returns random numbers following a triangular distribution

Parameters:
Size: A positive integer up to 10^7
a: The left border parameter.
b: The right border parameter.
c: The mode parameter.
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given below.

17.27 Uniform Distribution

17.27.1 Definition

These functions return PDF, CDF, and ICDF of the uniform distribution on the support interval $[a, b]$ with finite $a < b$:

17.27.2 Density and CDF

Function **UniformDistBoost**(***x*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

---

The function UniformDistBoost returns returns pdf, CDF and related information for the uniform distribution

Parameters:
x: A real number
a: The left border parameter.
b: The right border parameter.
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 17.27.2.1 and 17.27.2.2.

17.27.2.1 Density

$$f(x) = \frac{1}{b-a} \tag{17.27.1}$$

17.27.2.2 CDF

$$F(x) = \frac{x-a}{b-a} \tag{17.27.2}$$

17.27.3 Quantiles

Function **UniformDistInvBoost**(***Prob*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

---

The function UniformDistInvBoost returns returns quantiles and related information for the the uniform distribution

Parameters:
Prob: A real number between 0 and 1.
a: The left border parameter.
b: The right border parameter.
Output: A string describing the output choices

See section 17.1.3.2 for the options for *Prob* and *Output*). Algorithms and formulas are given below.

$$F^{-1}(y) = a + y(b - a) \tag{17.27.3}$$

17.27.4 Properties

Function **UniformDistInfoBoost**(***a*** *As mpNum*, ***b*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function UniformDistInfoBoost returns returns moments and related information for the uniform distribution

Parameters:
a: A real number greater 0, representing the degrees of freedom
b: A real number greater 0, representing the degrees of freedom
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*.

17.27.4.1 Moments

$$\mu_1 = \frac{a+b}{2} \tag{17.27.4}$$

$$\mu_2 = \frac{(b-a)^2}{12} \tag{17.27.5}$$

$$\gamma_1 = 0 \tag{17.27.6}$$

$$\gamma_2 = -\frac{6}{5} \tag{17.27.7}$$

17.27.5 Random Numbers

Function **UniformDistRanBoost**(***Size*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function UniformDistRanBoost returns returns random numbers following a uniform distribution

Parameters:
Size: A positive integer up to 10^7
a: A real number greater 0, representing the degrees of freedom
b: A real number greater 0, representing the degrees of freedom
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given in below.

Chapter 18

Noncentral Distribution Functions (based on Boost)

18.1 Noncentral Beta-Distribution

18.1.1 Definition

If X_1 an X_2 are independent random variables, X_1 following a non-central χ^2-distribution with noncentrality parameter λ and $2a$ degrees of freedom, and X_2 following a χ^2-distribution with $2b$ degrees of freedom, then the distribution of the ratio $B = \frac{X_1}{X_1+X_2}$ is said to follow a non-central Beta-distribution with a and b degrees of freedom.
Note: The univariate version of the noncentral distribution of Wilks Lambda: GLM is equivalent to $W = 1 - B$
See Tiwari & Yang (1997)

18.1.2 Density and CDF

Function **NoncentralBetaDistBoost**(***x*** *As mpNum*, ***m*** *As mpNum*, ***n*** *As mpNum*, ***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

---

The function NoncentralBetaDistBoost returns returns pdf, CDF and related information for the central Beta-distribution

Parameters:
x: A real number
m: A real number greater 0, representing the numerator degrees of freedom
n: A real number greater 0, representing the denominator degrees of freedom
lambda: A real number greater 0, representing the noncentrality parameter
Output: A string describing the output choices

18.1.2.1 Density

The density function of the noncentral beta-Distribution is given by (Wang & Gray, 1993): this needs to be checked, see Paolella 2007:

$$f_{\text{Beta'}}(x; n_1, n_2, \lambda) = e^{-\lambda/2} f_B(x; n_1, n_2)_1F_1\left(\tfrac{1}{2}(m+n), \tfrac{1}{2}n, \tfrac{nx\lambda}{2(m+nx)}\right) \tag{18.1.1}$$

18.1.2.2 CDF: Infinite Series

The cdf can be calculated using the following infinite series Benton & Krishnamoorthy (2003):

$$\Pr[F \leq x] = F_{B'}(x; a, b, \lambda) = e^{-\lambda/2} \sum_{j=0}^{\infty} \frac{(\lambda/2)^j}{j!} I(a + j, b, x) \tag{18.1.2}$$

18.1.3 Quantiles

Function **NoncentralBetaDistInvBoost**(***Prob*** *As mpNum*, ***m*** *As mpNum*, ***n*** *As mpNum*, ***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function NoncentralBetaDistInvBoost returns returns quantiles and related information for the the noncentral Beta-distribution

Parameters:
Prob: A real number between 0 and 1.
m: A real number greater 0, representing the numerator degrees of freedom
n: A real number greater 0, representing the denominator degrees of freedom
lambda: A real number greater 0, representing the noncentrality parameter
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*). Algorithms and formulas are given below.

18.1.4 Properties

Function **NoncentralBetaDistInfoBoost**(***m*** *As mpNum*, ***n*** *As mpNum*, ***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function NoncentralBetaDistInfoBoost returns returns moments and related information for the noncentral Beta-distribution

Parameters:
m: A real number greater 0, representing the numerator degrees of freedom
n: A real number greater 0, representing the denominator degrees of freedom
lambda: A real number greater 0, representing the noncentrality parameter
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given below.

18.1.4.1 Moments of the non-central Beta-distribution

Currently, Boost does not calculate the moments of the noncentral beta distribution.

18.1.5 Random Numbers

Function **NoncentralBetaDistRanBoost**(***Size*** *As mpNum*, ***m*** *As mpNum*, ***n*** *As mpNum*, ***lambda*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function NoncentralBetaDistRanBoost returns returns random numbers following a noncentral Beta-distribution

Parameters:
Size: A positive integer up to 10^7
m: A real number greater 0, representing the numerator degrees of freedom
n: A real number greater 0, representing the denominator degrees of freedom
lambda: A real number greater 0, representing the noncentrality parameter
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given below.

Random numbers from a non-central Beta-distribution with integer or half-integer pâĹŠ and qâĹŠvalues is easily obtained using the definition above i.e. by using a random number from a non-central chi-square distribution and another from a (central) chi-square distribution.

18.2 Noncentral Chi-Square Distribution

18.2.1 Definition

Let $X_1, X_2, \ldots, X_n$ be independent and identically distributed random variables each following a normal distribution with mean μ_j and unit variance. Then $\chi^2 = \sum_{j=1}^{n} X_j$ is said to follow a non-central χ^2-distribution with n degress of freedom and noncentrality parameter $\lambda = \sum_{j=1}^{n}(\mu_j - \mu)$.

18.2.2 Density and CDF

Function **NoncentralCDistBoost**(***x*** *As mpNum*, ***n*** *As mpNum*, ***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function NoncentralCDistBoost returns returns pdf, CDF and related information for the noncentral χ^2-distribution

Parameters:
x: A real number
n: A real number greater 0, representing the degrees of freedom
lambda: A real number greater 0, representing the noncentrality parameter
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*.

18.2.2.1 CDF: General formulas

The cdf of a noncentral chi-square variable with n degrees of freedom and λ is given by

$$\Pr\left[\chi^2 \leq x\right] = F_{\chi^2}\left(n, x; \lambda\right) = \int_0^x f_{\chi^2}\left(n, t; \lambda\right) dt \tag{18.2.1}$$

18.2.2.2 CDF: Infinite series in terms of the central cdf

The cdf of a noncentral chi-square variable with n degrees of freedom and λ is given by

$$F_{\chi^2}\left(n, x; \lambda\right) = e^{-\lambda/2} \sum_{j=0}^{\infty} \frac{(\lambda/2)^j}{j!} F_{\chi^2}\left(n + 2 + j, x\right) \tag{18.2.2}$$

18.2.2.3 CDF: Infinite series in terms of the central pdf

Ding (1992) gives the following representation (this is used by Boost for small lambda):

$$F_{\chi^2}(n, x; \lambda) = 2e^{-\lambda/2} \sum_{i=0}^{\infty} f_{\chi^2}(n + 2 + 2i, x) \left(\sum_{k=0}^{i} \frac{(\lambda/2)^k}{k!} \right) \tag{18.2.3}$$

18.2.3 Quantiles

Function **NoncentralCDistInvBoost**(***Prob*** *As mpNum*, ***n*** *As mpNum*, ***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function NoncentralCDistInvBoost returns quantiles and related information for the noncentral χ^2-distribution

Parameters:
Prob: A real number between 0 and 1.
n: A real number greater 0, representing the degrees of freedom
lambda: A real number greater 0, representing the noncentrality parameter
Output: A string describing the output choices

See section 17.1.3.2 for the options for *Prob* and *Output*). Algorithms and formulas are given below

The quantile is approximated as

$$\chi^2_{n,\lambda,\alpha} \approx (1 + b)\chi^2_{n_1,\lambda,\alpha}, \quad \text{where } n_1 = \frac{(n + \lambda)^2}{n + 2\lambda}, \quad b = \frac{\lambda}{n + \lambda} \tag{18.2.4}$$

18.2.4 Properties

Function **NoncentralCDistInfoBoost**(***n*** *As mpNum*, ***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function NoncentralCDistInfoBoost returns moments and related information for the noncentral χ^2-distribution

Parameters:
n: A real number greater 0, representing the degrees of freedom
lambda: A real number greater 0, representing the noncentrality parameter
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given below.

18.2.4.1 Moments and Cumulants

The cumulants of noncentral $\chi^2$are given by

$$\kappa_r(n, \lambda) = 2^{r-1}(r - 1)!(n + r\lambda) \tag{18.2.5}$$

18.2.5 Random Numbers

Function **NoncentralCDistRanBoost**(***Size*** *As mpNum*, ***n*** *As mpNum*, ***lambda*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function NoncentralCDistRanBoost returns random numbers following a noncentral χ^2-distribution

Parameters:
Size: A positive integer up to 10^7
n: A real number greater 0, representing the degrees of freedom
lambda: A real number greater 0, representing the noncentrality parameter
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given below
Random numbers from a non-central chi-square distribution is easily obtained using the deïnĄnition above by e.g.

1. Put $\mu = \sqrt{\lambda/n}$
2. Sum n random numbers from a normal distribution with mean μ and variance unity. Note that this is not a unique choice. The only requirement is that $\lambda = \sum \mu_i^2$.
3. Return the sum as a random number from a non-central chi-square distribution with n degrees of freedom and non-central parameter λ.

This ought to be sufficient for most applications but if needed more efficient techniques may easily be developed e.g. using more general techniques.

18.3 NonCentral F-Distribution

18.3.1 Definition

If X_1 an X_2 are independent random variables, X_1 following a non-central χ^2-distribution with noncentrality parameter λ and m degrees of freedom, and X_2 following a χ^2-distribution with m degrees of freedom, then the distribution of the ratio $F = \frac{X_1/m}{X_2/n}$ is said to follow a non-central F-distribution with noncentrality parameter λ and m and n degrees of freedom.

18.3.2 Density and CDF

Function **NoncentralFDistBoost**(***x*** *As mpNum*, ***m*** *As mpNum*, ***n*** *As mpNum*, ***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function NoncentralFDistBoost returns returns pdf, CDF and related information for the noncentral F-distribution

Parameters:
x: A real number
m: A real number greater 0, representing the numerator degrees of freedom
n: A real number greater 0, representing the denominator degrees of freedom

lambda: A real number greater 0, representing the noncentrality parameter
Output: A string describing the output choices

18.3.2.1 Density

18.3.2.2 CDF (singly noncentral: Infinite Series

The cdf of a variable following a (singly) noncentral F-distribution with n and m degrees of freedom and noncentrality parameter λ_1 and is given by

$$\Pr[F \leq x] = F_{F'}(x; m, n, \lambda) = e^{-\lambda} \sum_{j=0}^{\infty} \frac{(\lambda/2)^j}{j!} F(m + 2j, n, x) \tag{18.3.1}$$

18.3.3 Quantiles

Function **NoncentralFDistInvBoost**(***Prob*** *As mpNum*, ***m*** *As mpNum*, ***n*** *As mpNum*, ***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function NoncentralFDistInvBoost returns returns quantiles and related information for the the noncentral F-distribution

Parameters:
Prob: A real number between 0 and 1.
m: A real number greater 0, representing the numerator degrees of freedom
n: A real number greater 0, representing the denominator degrees of freedom
lambda: A real number greater 0, representing the noncentrality parameter
Output: A string describing the output choices
See section 17.1.3.2 for the options for *Prob* and *Output*). Algorithms and formulas are given below.

18.3.4 Properties

Function **NoncentralFDistInfoBoost**(***m*** *As mpNum*, ***n*** *As mpNum*, ***lambda*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function NoncentralFDistInfoBoost returns returns moments and related information for the noncentral F-distribution

Parameters:
m: A real number greater 0, representing the numerator degrees of freedom
n: A real number greater 0, representing the denominator degrees of freedom
lambda: A real number greater 0, representing the noncentrality parameter
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given below.

18.3.4.1 Moments (singly noncentral

The algebraic moments (defined for $f_2 > 2r$) are given by

$$\mu'_r = \frac{\Gamma(\frac{1}{2}f_1 + r) - \Gamma(\frac{1}{2}f_2 - r)}{\Gamma(\frac{1}{2}f_2)} \sum_{j=0}^{r} \binom{r}{j} \frac{\frac{1}{2}\lambda f_1)^j}{\Gamma(\frac{1}{2}f_1 + j)}, \quad \text{for } f_2 > 2r. \tag{18.3.2}$$

The first four raw moments (defined for $n > 2k$) are given by

$$\mu'_1 = \frac{n}{m}\frac{m+\lambda}{n-2}$$

$$\mu'_2 = \left(\frac{n}{m}\right)^2 \frac{\lambda^2 + (2\lambda+m)(m+2)}{(n-2)(n-4)}$$

$$\mu'_3 = \left(\frac{n}{m}\right)^3 \frac{\lambda^3 + 3(m+4)\lambda^2 + (3\lambda+m)(m+4)(m+2)}{(n-2)(n-4)(n-6)}$$

$$\mu'_4 = \left(\frac{n}{m}\right)^4 \frac{\lambda^3 + 4(m+6)\lambda^3 + 6(m+6)(m+4)\lambda^2 + (4\lambda+m)(m+6)(m+4)(m+2)}{(n-2)(n-4)(n-6)(n-8)}$$

18.3.4.2 Relationships to other distributions (singly noncentral)

$$F_{F'}(x; m, n, \lambda) = F_B\left(\tfrac{1}{2}n, \tfrac{1}{2}m, \frac{mx}{mx+n}; \lambda\right) \tag{18.3.3}$$

18.3.5 Random Numbers

Function **NoncentralFDistRanBoost**(***Size*** *As mpNum,* ***m*** *As mpNum,* ***n*** *As mpNum,* ***lambda*** *As mpNum,* ***Generator*** *As String,* ***Output*** *As String*) As mpNumList

The function NoncentralFDistRanBoost returns returns random numbers following a noncentral F-distribution

Parameters:
Size: A positive integer up to 10^7
m: A real number greater 0, representing the numerator degrees of freedom
n: A real number greater 0, representing the denominator degrees of freedom
lambda: A real number greater 0, representing the noncentrality parameter
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given below.
Random numbers from a non-central F-distribution is easily obtained using the definition in terms of the ratio between two independent random numbers from central and non-central chi-square distributions. This ought to be sufficient for most applications but if needed more efficient techniques may easily be developed e.g. using more general techniques.

18.4 Noncentral Student's t-Distribution

18.4.1 Definition

If X is a random variable following a normal distribution with mean δ and variance unity and χ^2 is a random variable following an independent χÂř2-distribution with n degrees of freedom, then the distribution of the ratio $\frac{X}{\sqrt{\chi^2/n}}$ is called noncentral t-distribution with n degrees of freedom and noncentrality parameter δ.

18.4.2 Density and CDF

Function **NoncentralTDistBoost**(***x*** *As mpNum*, ***n*** *As mpNum*, ***delta*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function NoncentralTDistBoost returns returns pdf, CDF and related information for the noncentral t-distribution

Parameters:
x: A real number
n: A real number greater 0, representing the degrees of freedom
delta: A real number greater 0, representing the noncentrality parameter
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 18.4.2 and 18.4.2.1.

18.4.2.1 Density (singly noncentral): Infinite series

The pdf of a variable following a noncentral t-distribution with n degrees of freedom and noncentrality parameter δ is given by (Bristow *et al.*, 2013)

$$f_{t'}(n,x,\delta) = \frac{nt}{n^2+2nt^2+t^4} + \frac{1}{2}\sum_{i=0}^{\infty} P_i I'_x\left(i+\frac{1}{2},\frac{n}{2}\right) + \frac{\delta}{\sqrt{2}} Q_i I'_x\left(i+1,\frac{n}{2}\right), \quad \text{and} \tag{18.4.1}$$

$I'_x(\cdot,\cdot)$ denotes the derivative of the (normalized) incomplete beta function (see section 6.7.16), and P_i and Q_i are defined in equation 18.4.4.

18.4.2.2 CDF (singly noncentral): Infinite series

The cdf of a variable following a noncentral t-distribution with n degrees of freedom and noncentrality parameter δ is given by (Benton & Krishnamoorthy, 2003; Bristow *et al.*, 2013)

$$F_{t'}(n,x,\delta) = \Phi(-\delta) + \frac{1}{2}\sum_{i=0}^{\infty} P_i I_x\left(i+\frac{1}{2},\frac{n}{2}\right) + \frac{\delta}{\sqrt{2}} Q_i I_x\left(i+1,\frac{n}{2}\right), \quad \text{and} \tag{18.4.2}$$

$$1 - F_{t'}(n,x,\delta) = \frac{1}{2}\sum_{i=0}^{\infty} P_i I_y\left(\frac{n}{2},i+\frac{1}{2}\right) + \frac{\delta}{\sqrt{2}} Q_i I_y\left(\frac{n}{2},i+1\right), \quad \text{where} \tag{18.4.3}$$

$$\lambda = \tfrac{1}{2}\delta^2; \quad P_i = \frac{e^{-\lambda}\lambda^i}{i!}; \quad Q_i = \frac{e^{-\lambda}\lambda^i}{\Gamma(i+3/2)}; \quad x = \frac{t^2}{n+t^2}; \quad y = 1-x, \tag{18.4.4}$$

18.4.3 Quantiles

Function **NoncentralTDistInvBoost**(***Prob*** *As mpNum*, ***n*** *As mpNum*, ***delta*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function NoncentralTDistInvBoost returns quantiles and related information for the noncentral t-distribution

Parameters:
Prob: A real number between 0 and 1.

n: A real number greater 0, representing the degrees of freedom
delta: A real number greater 0, representing the noncentrality parameter
Output: A string describing the output choices

See section 17.1.3.2 for the options for *Prob* and *Output*). Algorithms and formulas are given below:

18.4.4 Properties

Function **NoncentralTDistInfoBoost**(***n*** *As mpNum*, ***delta*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function NoncentralTDistInfoBoost returns moments and related information for the noncentral t-distribution

Parameters:
n: A real number greater 0, representing the degrees of freedom
delta: A real number greater 0, representing the noncentrality parameter
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given below.

18.4.4.1 Moments (singly noncentral)

The algebraic moments (defined for $n > r$) are given by

$$\mu'_r = \left(\tfrac{1}{2}n\right)^{r/2} \frac{\Gamma\left(\frac{1}{2}(n-r)\right)}{\Gamma\left(\frac{1}{2}n\right)} \sum_{i=0}^{[r/2]_G} \binom{r}{2i} \frac{(2i)!}{2^i i!} \delta^{r-2i}. \tag{18.4.5}$$

The first four raw moments are given by

$$E(t) = \delta\sqrt{\tfrac{1}{2}n}\frac{\Gamma\left(\frac{1}{2}(n-1)\right)}{\Gamma\left(\frac{1}{2}n\right)} \tag{18.4.6}$$

$$E(t^2) = (\delta^2+1)\frac{n}{n-2} \tag{18.4.7}$$

$$E(t^3) = \delta(\delta^2+3)\sqrt{\tfrac{1}{8}n^3}\frac{\Gamma\left(\frac{1}{2}(n-3)\right)}{\Gamma\left(\frac{1}{2}n\right)} \tag{18.4.8}$$

$$E(t^4) = (\delta^4+6\delta^2+3)\frac{n^2}{(n-2)(n-4)} \tag{18.4.9}$$

18.4.5 Random Numbers

Function **NoncentralTDistRanBoost**(***Size*** *As mpNum*, ***n*** *As mpNum*, ***delta*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function NoncentralTDistRanBoost returns random numbers following a noncentral t-distribution

Parameters:
Size: A positive integer up to 10^7

n: A real number greater 0, representing the degrees of freedom
delta: A real number greater 0, representing the noncentrality parameter
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given below:

Random numbers from a non-central t-distribution is easily obtained using the definition in terms of the ratio between two independent random numbers from a normal and a central chi-square distribution. This ought to be sufficient for most applications but if needed more efficient techniques may easily be developed e.g. using more general techniques.

18.5 Skew Normal Distribution

18.5.1 Definition

In probability theory and statistics, the skew normal distribution is a continuous probability distribution that generalises the normal distribution to allow for non-zero skewness.
See also http://en.wikipedia.org/wiki/Skew_normal_distribution.

18.5.2 Density and CDF

Function **SkewNormalDistBoost**(***x*** *As mpNum,* ***a*** *As mpNum,* ***b*** *As mpNum,* ***c*** *As mpNum,* ***Output*** *As String*) As mpNumList

The function SkewNormalDistBoost returns returns pdf, CDF and related information for the skew normal distribution

Parameters:
x: A real number.
a: The location parameter.
b: The scale parameter
c: The shape parameter
Output: A string describing the output choices

See section 17.1.3.1 for the options for *Output*. Algorithms and formulas are given in sections 18.5.2.1 and 18.5.2.2.

18.5.2.1 Density

The probability density function with location parameter ξ, scale parameter ω, and shape parameter α is

$$f(\xi;\omega,\alpha) = \frac{2}{\omega}\phi\left(\frac{x-\xi}{\omega}\right)\Phi\left(\alpha\left(\frac{x-\xi}{\omega}\right)\right) \tag{18.5.1}$$

The probability density function with location parameter a, scale parameter b, and shape parameter c is

$$f(x;a,b,c) = \frac{2}{b}\phi\left(\frac{x-a}{b}\right)\Phi\left(c\left(\frac{x-a}{b}\right)\right) \tag{18.5.2}$$

18.5.2.2 CDF

$$F(\xi;\omega,\alpha) = \Phi\left(\frac{x-\xi}{\omega}\right) - 2T\left(\frac{x-\xi}{\omega},\alpha\right) \tag{18.5.3}$$

$$F(a,b,c) = \Phi\left(\frac{x-a}{b}\right) - 2T\left(\frac{x-a}{b},c\right) \tag{18.5.4}$$

18.5.3 Quantiles

Function **SkewNormalDistInvBoost**(***Prob*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***c*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function SkewNormalDistInvBoost returns returns quantiles and related information for the the skew normal distribution

Parameters:
Prob: A real number between 0 and 1.
a: The location parameter.
b: The scale parameter
c: The shape parameter
Output: A string describing the output choices

See section 17.1.3.2 for the options for *Prob* and *Output*). The quantile is determined using an iterative alogorithm.

18.5.4 Properties

Function **SkewNormalDistInfoBoost**(***a*** *As mpNum*, ***b*** *As mpNum*, ***c*** *As mpNum*, ***Output*** *As String*) As mpNumList

The function SkewNormalDistInfoBoost returns returns moments and related information for the skew normal distribution

Parameters:
a: The location parameter.
b: The scale parameter
c: The shape parameter
Output: A string describing the output choices

See section 17.1.3.3 for the options for *Output*. Algorithms and formulas are given below.

18.5.4.1 Moments

$$\mu_1 = a + bd\sqrt{\frac{2}{\pi}}, \quad \text{where } d = \frac{c}{\sqrt{1+c^2}} \tag{18.5.5}$$

$$\mu_2 = b^2\left(1 - \frac{2d^2}{\pi}\right) \tag{18.5.6}$$

$$\gamma_1 = \frac{4-\pi}{2}\frac{\left(d\sqrt{2/\pi}\right)^3}{(1-2d^2/\pi)^{3/2}} \tag{18.5.7}$$

$$\gamma_2 = 2(\pi - 3)\frac{\left(d\sqrt{2/\pi}\right)^4}{(1 - 2d^2/\pi)^2} \tag{18.5.8}$$

18.5.5 Random Numbers

Function **SkewNormalDistRanBoost**(***Size*** *As mpNum*, ***a*** *As mpNum*, ***b*** *As mpNum*, ***c*** *As mpNum*, ***Generator*** *As String*, ***Output*** *As String*) As mpNumList

The function SkewNormalDistRanBoost returns returns random numbers following a skew normal distribution

Parameters:
Size: A positive integer up to 10^7
a: The location parameter.
b: The scale parameter
c: The shape parameter
Generator: A string describing the random generator
Output: A string describing the output choices

See section 17.1.3.6 for the options for *Size*, *Generator* and *Output*. Algorithms and formulas are given below.

18.6 Owen's T-Function

18.6.1 Owen's T-Function

Function **TOwenBoost**(***h*** *As mpNum*, ***a*** *As mpNum*) As mpNum

The function TOwenBoost returns Owen's T-Function

Parameters:
h: A real number.
a: A real number.

Owen's T-Function is defined as (Owen, 1956):

$$T(h, a) = \frac{1}{2\pi}\int_0^a \frac{\exp\left[-\frac{1}{2}h^2(1 + x^2)\right]}{1 + x^2}dx \tag{18.6.1}$$

The implementation uses the algorithm descibed in Patefield & Tand (2000).

Part V

ODE and NLOPT

Chapter 19

Ordinary Differential Equations

The procedures in this chapter are based on Boost.Numeric.Odeint (see Ahnert & Mulansky (2013)), a library for solving initial value problems (IVP) of ordinary differential equations. Mathematically, these problems are formulated as follows:
$x'(t) = f(x,t), x(0) = x0$.
x and f can be vectors and the solution is some function $x(t)$ fulfilling both equations above. In the following we will refer to $x'(t)$ also $dxdt$ which is also our notation for the derivative in the source code.
Ordinary differential equations occur nearly everywhere in natural sciences. For example, the whole Newtonian mechanics are described by second order differential equations. Be sure, you will find them in every discipline. They also occur if partial differential equations (PDEs) are discretized. Then, a system of coupled ordinary differential occurs, sometimes also referred as lattices ODEs.
Numerical approximations for the solution $x(t)$ are calculated iteratively. The easiest algorithm is the Euler scheme, where starting at $x(0)$ one finds $x(dt) = x(0) + dtf(x(0), 0)$. Now one can use $x(dt)$ and obtain $x(2dt)$ in a similar way and so on. The Euler method is of order 1, that means the error at each step is $\approx dt2$. This is, of course, not very satisfactory, which is why the Euler method is rarely used for real life problems and serves just as illustrative example.
The main focus of odeint is to provide numerical methods implemented in a way where the algorithm is completely independent on the data structure used to represent the state x. In doing so, odeint is applicable for a broad variety of situations and it can be used with many other libraries. Besides the usual case where the state is defined as a std::vector or a boost::array, we provide native support for the following libraries:
General Literature includes:
General information about numerical integration of ordinary differential equations:
Press *et al.* (2007)
Hairer *et al.* (2009)
Hairer & Wanner (2010)
Symplectic integration of numerical integration:
Hairer *et al.* (2006)
Leimkuhler & Reich (2005)
Special symplectic methods:
Yoshida (1990)
McLachlan (1995)
Special systems:
Fermi-Pasta-Ulam nonlinear lattice oscillations
Pikovsky *et al.* (2001)

19.1 Defining the ODE System

The routines solve the general n-dimensional first-order system,

$$\frac{dy_i(t)}{dt} = f_i(t, y_1(t), ...y_n(t)) \tag{19.1.1}$$

for $i = 1, ..., n$. The stepping functions rely on the vector of derivatives f_i and the Jacobian matrix, $J_{ij} = \partial f_i(t, y(t))/\partial y_j$. A system of equations is defined using the system datatype.

19.2 Stepping Functions

Solving ordinary differential equation numerically is usually done iteratively, that is a given state of an ordinary differential equation is iterated forward $x(t) - > x(t + dt) - > x(t + 2dt)$. The steppers in odeint perform one single step. The most general stepper type is described by the Stepper concept. The stepper concepts of odeint are described in detail in section Concepts, here we briefly present the mathematical and numerical details of the steppers. The Stepper has two versions of the do_step method, one with an in-place transform of the current state and one with an out-of-place transform:
do_step(sys , inout , t , dt)
do_step(sys , in , t , out , dt)
The first parameter is always the system function - a function describing the ODE. In the first version the second parameter is the step which is here updated in-place and the third and the fourth parameters are the time and step size (the time step). After a call to do_step the state inout is updated and now represents an approximate solution of the ODE at time t+dt. In the second version the second argument is the state of the ODE at time t, the third argument is t, the fourth argument is the approximate solution at time t+dt which is filled by do_step and the fifth argument is the time step. Note that these functions do not change the time t.
System functions
Up to now, we have nothing said about the system function. This function depends on the stepper. For the explicit Runge-Kutta steppers this function can be a simple callable object hence a simple (global) C-function or a functor. The parameter syntax is $sys(x, dxdt, t)$ and it is assumed that it calculates $dx/dt = f(x, t)$. The function structure in most cases looks like:
void sys(const state_type & /*x*/ , state_type & /*dxdt*/ , const double /*t*/) // ...
Other types of system functions might represent Hamiltonian systems or systems which also compute the Jacobian needed in implicit steppers. For information which stepper uses which system function see the stepper table below. It might be possible that odeint will introduce new system types in near future. Since the system function is strongly related to the stepper type, such an introduction of a new stepper might result in a new type of system function.
Explicit steppers A first specialization are the explicit steppers. Explicit means that the new state of the ode can be computed explicitly from the current state without solving implicit equations. Such steppers have in common that they evaluate the system at time t such that the result of f(x,t) can be passed to the stepper. In odeint, the explicit stepper have two additional methods
Which steppers should be used in which situation
odeint provides a quite large number of different steppers such that the user is left with the question of which stepper fits his needs. Our personal recommendations are:
runge_kutta_dopri5 is maybe the best default stepper. It has step size control as well as dense-output functionality. Simple create a dense-output stepper by make_dense_output(1.0e-6 , 1.0e-5 , runge_kutta_dopri5¡ state_type ¿()). runge_kutta4 is a good stepper for constant step sizes. It

is widely used and very well known. If you need to create artificial time series this stepper should be the first choice. 'runge_kutta_fehlberg78' is similar to the 'runge_kutta4' with the advantage that it has higher precision. It can also be used with step size control. adams_bashforth_moulton is very well suited for ODEs where the r.h.s. is expensive (in terms of computation time). It will calculate the system function only once during each step.

19.2.1 Explicit Euler

In mathematics and computational science, the Euler method is a first-order numerical procedure for solving ordinary differential equations (ODEs) with a given initial value. It is the most basic explicit method for numerical integration of ordinary differential equations and is the simplest RungeâĂŞKutta method. The Euler method is named after Leonhard Euler, who treated it in his book Institutionum calculi integralis (published 1768âĂŞ70).[1]
The Euler method is a first-order method, which means that the local error (error per step) is proportional to the square of the step size, and the global error (error at a given time) is proportional to the step size. The Euler method often serves as the basis to construct more complicated methods.

19.2.2 Modified Midpoint

In numerical analysis, a branch of applied mathematics, the midpoint method is a one-step method for numerically solving the differential equation,
$y'(t) = f(t, y(t), y(t_0) = y_0$
and is given by the formula
$y_{n+1} = y_n + hf(t_n + h/2, y_n + (h/2)f(t_n, y_n)$,
for $n = 0, 1, 2, ...$ Here, h is the step size - a small positive number, $t_n = t_0 + nh$ and y_n is the computed approximate value of $y(t_n)$. The midpoint method is also known as the modified Euler method.[1]
The name of the method comes from the fact that in the formula above the function f is evaluated at $t = t_n + h/2$ which is the midpoint between t_n at which the value of $y(t)$ is known and t_{n+1} at which the value of $y(t)$ needs to be found.
The local error at each step of the midpoint method is of order $O(h^3)$, giving a global error of order $O(h^2)$. Thus, while more computationally intensive than Euler's method, the midpoint method generally gives more accurate results.
The method is an example of a class of higher-order methods known as Runge-Kutta methods.

19.2.3 Runge-Kutta 4

In numerical analysis, the RungeâĂŞKutta methods are an important family of implicit and explicit iterative methods, which are used in temporal discretization for the approximation of solutions of ordinary differential equations. These techniques were developed around 1900 by the German mathematicians C. Runge and M. W. Kutta.
See the article on numerical methods for ordinary differential equations for more background and other methods. See also List of RungeâĂŞKutta methods.
One member of the family of RungeâĂŞKutta methods is often referred to as "RK4", "classical RungeâĂŞKutta method" or simply as "the RungeâĂŞKutta method".
Let an initial value problem be specified as follows.

Here, y is an unknown function (scalar or vector) of time t which we would like to approximate; we are told that , the rate at which y changes, is a function of t and of y itself. At the initial time the corresponding y-value is . The function f and the data , are given.

19.2.4 Cash-Karp

In numerical analysis, the CashâĂŞKarp method is a method for solving ordinary differential equations (ODEs). It was proposed by Professor Jeff R. Cash [1] from Imperial College London and Alan H. Karp from IBM Scientific Center. The method is a member of the RungeâĂŞKutta family of ODE solvers. More specifically, it uses six function evaluations to calculate fourth- and fifth-order accurate solutions. The difference between these solutions is then taken to be the error of the (fourth order) solution. This error estimate is very convenient for adaptive stepsize integration algorithms. Other similar integration methods are Fehlberg (RKF) and DormandâĂŞPrince (RKDP).

J. R. Cash, A. H. Karp. "A variable order Runge-Kutta method for initial value problems with rapidly varying right-hand sides", ACM Transactions on Mathematical Software 16: 201-222, 1990. doi:10.1145/79505.79507.

Shampine, Lawrence F. (1986), "Some Practical Runge-Kutta Formulas", Mathematics of Computation (American Mathematical Society) 46 (173): 135âĂŞ150, doi:10.2307/2008219, JSTOR 2008219 .

19.2.5 Dormand-Prince 5

In numerical analysis, the DormandâĂŞPrince method, or DOPRI method, is an explicit method for solving ordinary differential equations (Dormand & Prince, 1980). The method is a member of the RungeâĂŞKutta family of ODE solvers. More specifically, it uses six function evaluations to calculate fourth- and fifth-order accurate solutions. The difference between these solutions is then taken to be the error of the (fourth-order) solution. This error estimate is very convenient for adaptive stepsize integration algorithms. Other similar integration methods are Fehlberg (RKF) and CashâĂŞKarp (RKCK).

The DormandâĂŞPrince method has seven stages, but it uses only six function evaluations per step because it has the FSAL (First Same As Last) property: the last stage is evaluated at the same point as the first stage of the next step. Dormand and Prince choose the coefficients of their method to minimize the error of the fifth-order solution. This is the main difference with the Fehlberg method, which was constructed so that the fourth-order solution has a small error. For this reason, the DormandâĂŞPrince method is more suitable when the higher-order solution is used to continue the integration, a practice known as local extrapolation (Shampine 1986; Hairer, NÃÿrsett & Wanner 2008, pp. 178âĂŞ179).

DormandâĂŞPrince is currently the default method in MATLAB and GNU Octave's ode45 solver and is the default choice for the Simulink's model explorer solver. A Fortran free software implementation of the algorithm called DOPRI5 is also available.[1]

19.2.6 Fehlberg 78

In mathematics, the RungeâĂŞKuttaâĂŞFehlberg method (or Fehlberg method) is an algorithm in numerical analysis for the numerical solution of ordinary differential equations. It was developed by the German mathematician Erwin Fehlberg and is based on the large class of RungeâĂŞKutta methods.

The novelty of Fehlberg's method is that it is an embedded method from the Runge-Kutta family, meaning that identical function evaluations are used in conjunction with each other to create methods of varying order and similar error constants. The method presented in Fehlberg's 1969 paper has been dubbed the RKF45 method, and is a method of order O(h4) with an error estimator of order O(h5).[1] By performing one extra calculation, the error in the solution can be estimated and controlled by using the higher-order embedded method that allows for an adaptive stepsize to be determined automatically.
Erwin Fehlberg (1970). "Klassische Runge-Kutta-Formeln vierter und niedrigerer Ordnung mit Schrittweiten-Kontrolle und ihre Anwendung auf WÃđrmeleitungsprobleme," Computing (Arch. Elektron. Rechnen), vol. 6, pp. 61âĂŞ71. doi:10.1007/BF02241732

19.2.7 Adams-Bashforth

Three families of linear multistep methods are commonly used: AdamsâĂŞBashforth methods, AdamsâĂŞMoulton methods, and the backward differentiation formulas (BDFs).
AdamsâĂŞBashforth methods[edit]The AdamsâĂŞBashforth methods are explicit methods. The coefficients are and , while the are chosen such that the methods has order s (this determines the methods uniquely).
The AdamsâĂŞBashforth methods with s = 1, 2, 3, 4, 5 are (Hairer, NÃÿrsett & Wanner 1993, ÂğIII.1; Butcher 2003, p. 103):

19.2.8 Adams-Moulton

The AdamsâĂŞMoulton methods are similar to the AdamsâĂŞBashforth methods in that they also have and . Again the b coefficients are chosen to obtain the highest order possible. However, the AdamsâĂŞMoulton methods are implicit methods. By removing the restriction that , an s-step AdamsâĂŞMoulton method can reach order , while an s-step AdamsâĂŞBashforth methods has only order s.
The AdamsâĂŞMoulton methods with s = 0, 1, 2, 3, 4 are (Hairer, NÃÿrsett & Wanner 1993, ÂğIII.1; Quarteroni, Sacco & Saleri 2000):

19.2.9 Adams-Bashforth-Moulton

The methods of Euler, Heun, Taylor and Runge-Kutta are called single-step methods because they use only the information from one previous point to compute the successive point, that is, only the initial point is used to compute and in general is needed to compute . After several points have been found it is feasible to use several prior points in the calculation. The Adams-Bashforth-Moulton method uses in the calculation of . This method is not self-starting; four initial points , , , and must be given in advance in order to generate the points .
A desirable feature of a multistep method is that the local truncation error (L. T. E.) can be determined and a correction term can be included, which improves the accuracy of the answer at each step. Also, it is possible to determine if the step size is small enough to obtain an accurate value for , yet large enough so that unnecessary and time-consuming calculations are eliminated. If the code for the subroutine is fine-tuned, then the combination of a predictor and corrector requires only two function evaluations of f(t,y) per step.
See also: http://mathfaculty.fullerton.edu/mathews//n2003/AdamsBashforthMod.html

19.2.10 Controlled Runge-Kutta

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

19.2.11 Dense Output Runge-Kutta

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

19.2.12 Bulirsch-Stoer

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

19.2.13 Bulirsch-Stoer Dense Output

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

19.2.14 Implicit Euler

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

19.2.15 Rosenbrock 4

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et

nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

19.2.16 Controlled Rosenbrock 4

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

19.2.17 Dense Output Rosenbrock 4

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

19.2.18 Symplectic Euler

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

19.2.19 Symplectic RKN McLachlan

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

19.3 Integrate functions: Evolution

Integrate functions perform the time evolution of a given ODE from some starting time t0 to a given end time t1 and starting at state x0 by subsequent calls of a given stepper's do_step function.
Additionally, the user can provide an observer to analyze the state during time evolution. There are five different integrate functions which have different strategies on when to call the observer function during integration.

All of the integrate functions except integrate_n_steps can be called with any stepper following one of the stepper concepts: Stepper , Error Stepper , Controlled Stepper , Dense Output Stepper. Depending on the abilities of the stepper, the integrate functions make use of step-size control or dense output.

Chapter 20

Nonlinear Root-finding, Minimization and Optimization

20.1 One-Dimensional Root-finding

The root bracketing algorithms described in this section require an initial interval which is guaranteed to contain a rootâĂŤif a and b are the endpoints of the interval then f(a) must differ in sign from f(b). This ensures that the function crosses zero at least once in the interval. If a valid initial interval is used then these algorithm cannot fail, provided the function is well-behaved. Note that a bracketing algorithm cannot find roots of even degree, since these do not cross the x-axis.

20.1.1 Bisection

bisection [Solver] The bisection algorithm is the simplest method of bracketing the roots of a function. It is the slowest algorithm provided by the library, with linear convergence. On each iteration, the interval is bisected and the value of the function at the midpoint is calculated. The sign of this value is used to determine which half of the interval does not contain a root. That half is discarded to give a new, smaller interval containing the root. This procedure can be continued indefinitely until the interval is sufficiently small. At any time the current estimate of the root is taken as the midpoint of the interval.

20.1.2 False Position

falsepos [Solver] The false position algorithm is a method of finding roots based on linear interpolation. Its convergence is linear, but it is usually faster than bisection. On each iteration a line is drawn between the endpoints (a, f(a)) and (b, f(b)) and the point where this line crosses the x-axis taken as a âĂIJmidpointâĂİ. The value of the function at this point is calculated and its sign is used to determine which side of the interval does not contain a root. That side is discarded to give a new, smaller interval containing the root. This procedure can be continued indefinitely until the interval is sufficiently small. The best estimate of the root is taken from the linear interpolation of the interval on the current iteration.

20.1.3 Brent-Dekker

brent [Solver] The Brent-Dekker method (referred to here as BrentâĂŹs method) combines an interpolation strategy with the bisection algorithm. This produces a fast algorithm which is still

robust. On each iteration BrentâĂŹs method approximates the function using an interpolating curve. On the first iteration this is a linear interpolation of the two endpoints. For subsequent iterations the algorithm uses an inverse quadratic fit to the last three points, for higher accuracy. The intercept of the interpolating curve with the x-axis is taken as a guess for the root. If it lies within the bounds of the current interval then the interpolating point is accepted, and used to generate a smaller interval. If the interpolating point is not accepted then the algorithm falls back to an ordinary bisection step. The best estimate of the root is taken from the most recent interpolation or bisection.

20.1.4 Newton

Root Finding Algorithms using Derivatives The root polishing algorithms described in this section require an initial guess for the location of the root. There is no absolute guarantee of convergenceâĂŤthe function must be suitable for this technique and the initial guess must be sufficiently close to the root for it to work. When these conditions are satisfied then convergence is quadratic. These algorithms make use of both the function and its derivative.
newton [Derivative Solver] NewtonâĂŹs Method is the standard root-polishing algorithm. The algorithm begins with an initial guess for the location of the root. On each iteration, a line tangent to the function f is drawn at that position. The point where this line crosses the x-axis becomes the new guess. The iteration is defined by the following sequence,
$xi + 1 = xi â ŁŠ f(xi) f â ĂŠ (xi)$
NewtonâĂŹs method converges quadratically for single roots, and linearly for multiple roots.

20.1.5 Secant

secant [Derivative Solver] The secant method is a simplified version of NewtonâĂŹs method which does not require the computation of the derivative on every step. On its first iteration the algorithm begins with NewtonâĂŹs method, using the derivative to compute a first step,
$x1 = x0 â ŁŠ f(x0) f â ĂŠ (x0)$
Subsequent iterations avoid the evaluation of the derivative by replacing it with a numerical estimate, the slope of the line through the previous two points,
$i + 1 = xi â ŁŠ f(xi) f â ĂŠ est$
where
$f â ĂŠ est = f(xi) â ŁŠ f(xi â ŁŠ 1) xi â ŁŠ xi â ŁŠ 1$
When the derivative does not change significantly in the vicinity of the root the secant method gives a useful saving. Asymptotically the secant method is faster than NewtonâĂŹs method whenever the cost of evaluating the derivative is more than 0.44 times the cost of evaluating the function itself. As with all methods of computing a numerical derivative the estimate can suffer from cancellation errors if the separation of the points becomes too small. On single roots, the method has a convergence of order (1 + â ŁŽ5)/2 (approximately 1.62). It converges linearly for multiple roots.

20.1.6 Steffenson

steffenson [Derivative Solver] The Steffenson Method1 provides the fastest convergence of all the routines. It combines the basic Newton algorithm with an Aitken âĂIJdelta-squaredâĂİ acceleration. If the Newton iterates are xi then the acceleration procedure generates a new sequence Ri,
$Ri = xi â ŁŠ (xi + 1 â ŁŠ xi) 2 (xi + 2 â ŁŠ 2xi + 1 + xi)$

which converges faster than the original sequence under reasonable conditions. The new sequence requires three terms before it can produce its first value so the method returns accelerated values on the second and subsequent iterations. On the first iteration it returns the ordinary Newton estimate. The Newton iterate is also returned if the denominator of the acceleration term ever becomes zero. As with all acceleration procedures this method can become unstable if the function is not well-behaved.

20.2 One-Dimensional Minimization

20.2.1 Minimization: goldensection

goldensection [Minimizer] The golden section algorithm is the simplest method of bracketing the minimum of a function. It is the slowest algorithm provided by the library, with linear convergence. On each iteration, the algorithm first compares the subintervals from the endpoints to the current minimum. The larger subinterval is divided in a golden section (using the famous ratio $(3\hat{a}\acute{L}\check{S}\sqrt{5})/2 = 0.3189660...$) and the value of the function at this new point is calculated. The new value is used with the constraint $f(a\hat{a}\check{A}\check{s}) > f(x\hat{a}\check{A}\check{s}) < f(b\hat{a}\check{A}\check{s})$ to a select new interval containing the minimum, by discarding the least useful point. This procedure can be continued indefinitely until the interval is sufficiently small. Choosing the golden section as the bisection ratio can be shown to provide the fastest convergence for this type of algorithm.

20.2.2 Minimization: Brent-Dekker

brent [Minimizer] The Brent minimization algorithm combines a parabolic interpolation with the golden section algorithm. This produces a fast algorithm which is still robust. The outline of the algorithm can be summarized as follows: on each iteration Brentâ$\check{A}\acute{Z}$s method approximates the function using an interpolating parabola through three existing points. The minimum of the parabola is taken as a guess for the minimum. If it lies within the bounds of the current interval then the interpolating point is accepted, and used to generate a smaller interval. If the interpolating point is not accepted then the algorithm falls back to an ordinary golden section step. The full details of Brentâ$\check{A}\acute{Z}$s method include some additional checks to improve convergence.

20.2.3 Minimization: Brent-Dekker-Gill-Murray

quad_golden [Minimizer] This is a variant of Brentâ$\check{A}\acute{Z}$s algorithm which uses the safeguarded step-length algorithm of Gill and Murray.

20.3 Procedures based on MINPACK

Reference for MINPACK: Moré *et al.* (1980).
Detailed Description
#include ¡unsupported/Eigen/NonLinearOptimization¿
This module provides implementation of two important algorithms in non linear optimization. In both cases, we consider a system of non linear functions. Of course, this should work, and even work very well if those functions are actually linear. But if this is so, you should probably better use other methods more fitted to this special case.
One algorithm allows to find an extremum of such a system (Levenberg Marquardt algorithm) and the second one is used to find a zero for the system (Powell hybrid "dogleg" method).

This code is a port of MINPACK . Minpack is a very famous, old, robust and well-reknown package, written in fortran. Those implementations have been carefully tuned, tested, and used for several decades.
The original fortran code was automatically translated using f2c in C, then c++, and then cleaned by several different authors.
Finally, we ported this code to Eigen, creating classes and API coherent with Eigen. When possible, we switched to Eigen implementation, such as most linear algebra (vectors, matrices, stable norms).
Doing so, we were very careful to check the tests we setup at the very beginning, which ensure that the same results are found.
Tests
The tests are placed in the file unsupported/test/NonLinear.cpp.
There are two kinds of tests : those that come from examples bundled with cminpack. They guaranty we get the same results as the original algorithms (value for 'x', for the number of evaluations of the function, and for the number of evaluations of the jacobian if ever).
Other tests were added by myself at the very beginning of the process and check the results for levenberg-marquardt using the reference data on NIST. Since then i've carefully checked that the same results were obtained when modifiying the code. Please note that we do not always get the exact same decimals as they do, but this is ok : they use 128bits float, and we do the tests using the C type 'double', which is 64 bits on most platforms (x86 and amd64, at least). I've performed those tests on several other implementations of levenberg-marquardt, and (c)minpack performs VERY well compared to those, both in accuracy and speed.
The documentation for running the tests is on the wiki http://eigen.tuxfamily.org/index.php?title=Tests
API : overview of methods
Both algorithms can use either the jacobian (provided by the user) or compute an approximation by themselves (actually using Eigen Numerical differentiation module). The part of API referring to the latter use 'NumericalDiff' in the method names (exemple: LevenbergMarquardt.minimizeNumericalDiff())
The methods LevenbergMarquardt.lmder1()/lmdif1()/lmstr1() and HybridNonLinearSolver.hybrj1()/hybrd are specific methods from the original minpack package that you probably should NOT use until you are porting a code that was previously using minpack. They just define a 'simple' API with default values for some parameters.
All algorithms are provided using Two APIs :
one where the user inits the algorithm, and uses '*OneStep()' as much as he wants : this way the caller have control over the steps
one where the user just calls a method (optimize() or solve()) which will handle the loop: init + loop until a stop condition is met. Those are provided for convenience.
As an example, the method LevenbergMarquardt::minimize() is implemented as follow :

```
Status LevenbergMarquardt<FunctorType,Scalar>::minimize(FVectorType &x, const int
    mode)
{
Status status = minimizeInit(x, mode);
do {
status = minimizeOneStep(x, mode);
} while (status==Running);
return status;
}
```

The easiest way to understand how to use this module is by looking at the many examples in the file unsupported/test/NonLinearOptimization.cpp.

20.3.1 Multidimensional Rootfinding: Powell Hybrid

This is a modified version of PowellâĂŹs Hybrid method as implemented in the hybrj algorithm in minpack. The Hybrid algorithm retains the fast convergence of NewtonâĂŹs method but will also reduce the residual when NewtonâĂŹs method is unreliable. The algorithm uses a generalized trust region to keep each step under control. In order to be accepted a proposed new position xâĂš must satisfy the condition $|D(x$âĂšâĹŠ$x)| < \delta$, where D is a diagonal scaling matrix and δ is the size of the trust region. The components of D are computed internally, using the column norms of the Jacobian to estimate the sensitivity of the residual to each component of x. This improves the behavior of the algorithm for badly scaled functions. On each iteration the algorithm first determines the standard Newton step by solving the system $Jdx =$ âĹŠf. If this step falls inside the trust region it is used as a trial step in the next stage. If not, the algorithm uses the linear combination of the Newton and gradient directions which is predicted to minimize the norm of the function while

$$dx = \text{âĹŠ}\alpha J^{\text{âĹŠ}1} f(x)\text{âĹŠ}\beta\nabla|f(x)|^2. \tag{20.3.1}$$

staying inside the trust region,
This combination of Newton and gradient directions is referred to as a dogleg step. The proposed step is now tested by evaluating the function at the resulting point, xâĂš. If the step reduces the norm of the function sufficiently then it is accepted and size of the trust region is increased. If the proposed step fails to improve the solution then the size of the trust region is decreased and another trial step is computed. The speed of the algorithm is increased by computing the changes to the Jacobian approximately, using a rank-1 update. If two successive attempts fail to reduce the residual then the full Jacobian is recomputed. The algorithm also monitors the progress of the solution and returns an error if several steps fail to make any improvement.

20.3.2 Nonlinear LeastSquares: Levenberg-Marquardt

The minimization algorithms described in this section make use of both the function and its derivative. They require an initial guess for the location of the minimum. There is no absolute guarantee of convergenceâĂŤthe function must be suitable for this technique and the initial guess must be sufficiently close to the minimum for it to work.
lmsder [Derivative Solver] This is a robust and efficient version of the Levenberg-Marquardt algorithm as implemented in the scaled lmder routine in minpack. Minpack was written by Jorge J. MorÂťe, Burton S. Garbow and Kenneth E. Hillstrom.
The algorithm uses a generalized trust region to keep each step under control. In order to be accepted a proposed new position xâĂš must satisfy the condition $|D(x$âĂšâĹŠ$x)| < \delta$, where D is a diagonal scaling matrix and δ is the size of the trust region. The components of D are computed internally, using the column norms of the Jacobian to estimate the sensitivity of the residual to each component of x. This improves the behavior of the algorithm for badly scaled functions.
On each iteration the algorithm attempts to minimize the linear system $|F + Jp|$ subject to the constraint $|Dp| <$. The solution to this constrained linear system is found using the Levenberg-Marquardt method.
The proposed step is now tested by evaluating the function at the resulting point, xâĂš. If the step reduces the norm of the function sufficiently, and follows the predicted behavior of the

function within the trust region, then it is accepted and the size of the trust region is increased. If the proposed step fails to improve the solution, or differs significantly from the expected behavior within the trust region, then the size of the trust region is decreased and another trial step is computed. The algorithm also monitors the progress of the solution and returns an error if the changes in the solution are smaller than the machine precision. The possible error codes are,

20.4 Procedures based on NLOPT: Overview

Reference to NLOPT is is Johnson (2012)
Nomenclature
Each algorithm in NLopt is identified by a named constant, which is passed to the NLopt routines in the various languages in order to select a particular algorithm. These constants are mostly of the form NLOPT_G,LN,D_xxxx, where G/L denotes global/local optimization and N/D denotes derivative-free/gradient-based algorithms, respectively.
For example, the NLOPT_LN_COBYLA constant refers to the COBYLA algorithm (described below), which is a local (L) derivative-free (N) optimization algorithm.
Two exceptions are the MLSL and augmented Lagrangian algorithms, denoted by NLOPT_G_MLSL and NLOPT_AUGLAG, since whether or not they use derivatives (and whether or not they are global, in AUGLAG's case) is determined by what subsidiary optimization algorithm is specified. Many of the algorithms have several variants, which are grouped together below.
Comparing algorithms
For any given optimization problem, it is a good idea to compare several of the available algorithms that are applicable to that problemâĂŤin general, one often finds that the "best" algorithm strongly depends upon the problem at hand.
However, comparing algorithms requires a little bit of care because the function-value/parameter tolerance tests are not all implemented in exactly the same way for different algorithms. So, for example, the same fractional $10^{âĹš4}$ tolerance on the function value might produce a much more accurate minimum in one algorithm compared to another, and matching them might require some experimentation with the tolerances.
Instead, a more fair and reliable way to compare two different algorithms is to run one until the function value is converged to some value fA, and then run the second algorithm with the $minf_max$ termination test set to $minf_max = fA$. That is, ask how long it takes for the two algorithms to reach the same function value.
Better yet, run some algorithm for a really long time until the minimum fM is located to high precision. Then run the different algorithms you want to compare with the termination test: $minf_max = fM + \Delta f$. That is, ask how long it takes for the different algorithms to obtain the minimum to within an absolute tolerance Δf, for some Δf. (This is totally different from using the $ftol_abs$ termination test, because the latter uses only a crude estimate of the error in the function values, and moreover the estimate varies between algorithms.)

20.5 NLOPT: Global optimization

All of the global-optimization algorithms currently require you to specify bound constraints on all the optimization parameters. Of these algorithms, only ISRES and ORIG_DIRECT support nonlinear inequality constraints, and only ISRES supports nonlinear equality constraints. (However, any of them can be applied to nonlinearly constrained problems by combining them with the augmented Lagrangian method below.)

Something you should consider is that, after running the global optimization, it is often worthwhile to then use the global optimum as a starting point for a local optimization to "polish" the optimum to a greater accuracy. (Many of the global optimization algorithms devote more effort to searching the global parameter space than in finding the precise position of the local optimum accurately.)

20.5.1 DIRECT and DIRECT-L

DIRECT is the DIviding RECTangles algorithm for global optimization, described in Jones *et al.* (1993)
and DIRECT-L is the "locally biased" variant proposed by Gablonsky & Kelley (2001)
These is are deterministic-search algorithms based on systematic division of the search domain into smaller and smaller hyperrectangles. The Gablonsky version makes the algorithm "more biased towards local search" so that it is more efficient for functions without too many local minima. NLopt contains several implementations of both of these algorithms. I would tend to try NLOPT_GN_DIRECT_L first; YMMV.
First, it contains a from-scratch re-implementation of both algorithms, specified by the constants NLOPT_GN_DIRECT and NLOPT_GN_DIRECT_L, respectively.
Second, there is a slightly randomized variant of DIRECT-L, specified by NLOPT_GLOBAL_DIRECT_L_RA which uses some randomization to help decide which dimension to halve next in the case of nearties.
The DIRECT and DIRECT-L algorithms start by rescaling the bound constraints to a hypercube, which gives all dimensions equal weight in the search procedure. If your dimensions do not have equal weight, e.g. if you have a "long and skinny" search space and your function varies at about the same speed in all directions, it may be better to use unscaled variants of these algorthms, which are specified as NLOPT_GLOBAL_DIRECT_NOSCAL, NLOPT_GLOBAL_DIRECT_L_NOSCAL, and NLOPT_GLOBAL_DIRECT_L_RAND_NOSCAL, respectively. However, the unscaled variations make the most sense (if any) with the original DIRECT algorithm, since the design of DIRECT-L to some extent relies on the search region being a hypercube (which causes the subdivided hyperrectangles to have only a small set of side lengths).
Finally, NLopt also includes separate implementations based on the original Fortran code by Gablonsky et al. (1998-2001), which are specified as NLOPT_GN_ORIG_DIRECT and NLOPT_GN_ORIG_D
These implementations have a number of hard-coded limitations on things like the number of function evaluations; I removed several of these limitations, but some remain. On the other hand, there seem to be slight differences between these implementations and mine; most of the time, the performance is roughly similar, but occasionally Gablonsky's implementation will do significantly better than mine or vice versa.
Most of the above algorithms only handle bound constraints, and in fact require finite bound constraints (they are not applicable to unconstrained problems). They do not handle arbitrary nonlinear constraints. However, the ORIG versions by Gablonsky et al. include some support for arbitrary nonlinear inequality constraints.

20.5.2 Controlled Random Search (CRS) with local mutation

My implementation of the "controlled random search" (CRS) algorithm (in particular, the CRS2 variant) with the "local mutation" modification, as defined by: Kaelo & Ali (2006).
The original CRS2 algorithm was described by: Price (1978, 1983)
The CRS algorithms are sometimes compared to genetic algorithms, in that they start with a random "population" of points, and randomly "evolve" these points by heuristic rules. In this case, the "evolution" somewhat resembles a randomized Nelder-Mead algorithm. The published

results for CRS seem to be largely empirical; limited analytical results about its convergence were derived in Hendrix *et al.* (2001)
The initial population size for CRS defaults to 10ÃŮ(n+1) in n dimensions, but this can be changed with the nlopt_set_stochastic_population function; the initial population must be at least n+1.
Only bound-constrained problems are supported by this algorithm.
CRS2 with local mutation is specified in NLopt as NLOPT_GN_CRS2_LM.

20.5.3 MLSL (Multi-Level Single-Limkage)

This is my implementation of the "Multi-Level Single-Linkage" (MLSL) algorithm for global optimization by a sequence of local optimizations from random starting points, proposed by: Rinnooy Kan & Timmer (1987a,b)
We also include a modification of MLSL use a Sobol' low-discrepancy sequence (LDS) instead of pseudorandom numbers, which was argued to improve the convergence rate by: Kucherenko & Sytsko (2005)
In either case, MLSL is a "multistart" algorithm: it works by doing a sequence of local optimizations (using some other local optimization algorithm) from random or low-discrepancy starting points. MLSL is distinguished, however by a "clustering" heuristic that helps it to avoid repeated searches of the same local optima, and has some theoretical guarantees of finding all local optima in a finite number of local minimizations.
The local-search portion of MLSL can use any of the other algorithms in NLopt, and in particular can use either gradient-based (D) or derivative-free algorithms (N) The local search uses the derivative/nonderivative algorithm set by nlopt_opt_set_local_optimizer.
LDS-based MLSL with is specified as NLOPT_G_MLSL_LDS, while the original non-LDS original MLSL (using pseudo-random numbers, currently via the Mersenne twister algorithm) is indicated by NLOPT_G_MLSL. In both cases, you must specify the local optimization algorithm (which can be gradient-based or derivative-free) via nlopt_opt_set_local_optimizer.
Note: If you do not set a stopping tolerance for your local-optimization algorithm, MLSL defaults to ftol_rel=10âĹŠ15 and xtol_rel=10âĹŠ7 for the local searches. Note that it is perfectly reasonable to set a relatively large tolerance for these local searches, run MLSL, and then at the end run another local optimization with a lower tolerance, using the MLSL result as a starting point, to "polish off" the optimum to high precision.
By default, each iteration of MLSL samples 4 random new trial points, but this can be changed with the nlopt_set_population function.
Only bound-constrained problems are supported by this algorithm.

20.5.4 StoGO

This is an algorithm adapted from the code downloaded from
StoGO global optimization library (link broken as of Nov. 2009, and the software seems absent from the author's web site) by Madsen et al. StoGO is a global optimization algorithm that works by systematically dividing the search space (which must be bound-constrained) into smaller hyperrectangles via a branch-and-bound technique, and searching them by a gradient-based local-search algorithm (a BFGS variant), optionally including some randomness (hence the "Sto", which stands for "stochastic" I believe).
StoGO is written in C++, which means that it is only included when you compile the C++ algorithms enabled, in which case (on Unix) you must link to -lnlopt_cxx instead of -lnlopt.

StoGO is specified within NLopt by NLOPT_GD_STOGO, or NLOPT_GD_STOGO_RAND for the randomized variant.
Some references on StoGO are: Gudmundsson (1998), Madsen *et al.* (1998), Zertchaninov & Madsen (1998)
Only bound-constrained problems are supported by this algorithm.

20.5.5 ISRES (Improved Stochastic Ranking Evolution Strategy)

This is my implementation of the "Improved Stochastic Ranking Evolution Strategy" (ISRES) algorithm for nonlinearly-constrained global optimization (or at least semi-global; although it has heuristics to escape local optima, I'm not aware of a convergence proof), based on the method described in: Runarsson & Yao (2005)
It is a refinement of an earlier method described in: Runarsson & Yao (2000)
This is an independent implementation by S. G. Johnson (2009) based on the papers above. Runarsson also has his own Matlab implemention available from his web page here.
The evolution strategy is based on a combination of a mutation rule (with a log-normal step-size update and exponential smoothing) and differential variation (a NelderâĂŞMead-like update rule). The fitness ranking is simply via the objective function for problems without nonlinear constraints, but when nonlinear constraints are included the stochastic ranking proposed by Runarsson and Yao is employed. The population size for ISRES defaults to 20ÃŮ(n+1) in n dimensions, but this can be changed with the nlopt_set_stochastic_population function.
This method supports arbitrary nonlinear inequality and equality constraints in addition to the bound constraints, and is specified within NLopt as NLOPT_GN_ISRES.

20.6 NLOPT: Local derivative-free optimization

Of these algorithms, only COBYLA currently supports arbitrary nonlinear inequality and equality constraints; the rest of them support bound-constrained or unconstrained problems only. (However, any of them can be applied to nonlinearly constrained problems by combining them with the augmented Lagrangian method below.)

20.6.1 COBYLA (Constrained Optimization BY Linear Approximations)

This is a derivative of Powell's implementation of the COBYLA (Constrained Optimization BY Linear Approximations) algorithm for derivative-free optimization with nonlinear inequality and equality constraints, by M. J. D. Powell, described in: Powell (1994)
and reviewed in: Powell (1998)
It constructs successive linear approximations of the objective function and constraints via a simplex of n+1 points (in n dimensions), and optimizes these approximations in a trust region at each step.
The original code itself was written in Fortran by Powell and was converted to C in 2004 by Jean-Sebastien Roy (js@jeannot.org) for the SciPy project. The version in NLopt was based on Roy's C version, downloaded from:
http://www.jeannot.org/ js/code/index.en.html#COBYLA NLopt's version is slightly modified in a few ways. First, we incorporated all of the NLopt termination criteria. Second, we added explicit support for bound constraints (although the original COBYLA could handle bound constraints as linear constraints, it would sometimes take a step that violated the bound constraints).

Third, we allow COBYLA to increase the trust-region radius if the predicted improvement was approximately right and the simplex is OK, following a suggestion in the SAS manual for PROC NLP that seems to improve convergence speed. Fourth, we pseudo-randomize simplex steps in COBYLA algorithm, improving robustness by avoiding accidentally taking steps that don't improve conditioning (which seems to happen sometimes with active bound constraints); the algorithm remains deterministic (a deterministic seed is used), however. Also, we support unequal initial-step sizes in the different parameters (by the simple expedient of internally rescaling the parameters proportional to the initial steps), which is important when different parameters have very different scales.
(The underlying COBYLA code only supports inequality constraints. Equality constraints are automatically transformed into pairs of inequality constraints, which in the case of this algorithm seems not to cause problems.)
It is specified within NLopt as NLOPT_LN_COBYLA.

20.6.2 BOBYQA

This is an algorithm derived from the BOBYQA subroutine of M. J. D. Powell, converted to C and modified for the NLopt stopping criteria. BOBYQA performs derivative-free bound-constrained optimization using an iteratively constructed quadratic approximation for the objective function. See: Powell (2009)
(Because BOBYQA constructs a quadratic approximation of the objective, it may perform poorly for objective functions that are not twice-differentiable.)
The NLopt BOBYQA interface supports unequal initial-step sizes in the different parameters (by the simple expedient of internally rescaling the parameters proportional to the initial steps), which is important when different parameters have very different scales.
This algorithm, specified in NLopt as NLOPT_LN_BOBYQA, largely supersedes the NEWUOA algorithm below, which is an earlier version of the same idea by Powell.

20.6.3 NEWUOA + bound constraints

This is an algorithm derived from the NEWUOA subroutine of M. J. D. Powell, converted to C and modified for the NLopt stopping criteria. I also modified the code to include a variant, NEWUOA-bound, that permits efficient handling of bound constraints. This algorithm is largely superseded by BOBYQA (above).
The original NEWUOA performs derivative-free unconstrained optimization using an iteratively constructed quadratic approximation for the objective function. See: Powell (2004)
(Because NEWUOA constructs a quadratic approximation of the objective, it may perform poorly for objective functions that are not twice-differentiable.)
The original algorithm is specified in NLopt as NLOPT_LN_NEWUOA, and only supports unconstrained problems. For bound constraints, my variant is specified as NLOPT_LN_NEWUOA_BOUND.
In the original NEWUOA algorithm, Powell solved the quadratic subproblems (in routines TRSAPP and BIGLAG) in a spherical trust region via a truncated conjugate-gradient algorithm. In my bound-constrained variant, we use the MMA algorithm for these subproblems to solve them with both bound constraints and a spherical trust region. In principle, we should also change the BIGDEN subroutine in a similar way (since BIGDEN also approximately solves a trust-region subproblem), but instead I just truncated its result to the bounds (which probably gives suboptimal convergence, but BIGDEN is called only very rarely in practice).
Shortly after my addition of bound constraints to NEWUOA, Powell released his own version of NEWUOA modified for bound constraints as well as some numerical-stability and convergence

enhancements, called BOBYQA. NLopt now incorporates BOBYQA as well, and it seems to largely supersede NEWUOA.
Note: NEWUOA requires the dimension n of the parameter space to be ≥ 2, i.e. the implementation does not handle one-dimensional optimization problems.

20.6.4 PRAXIS (Principal AXIS)

"PRAXIS" gradient-free local optimization via the "principal-axis method" of Richard Brent, based on a C translation of Fortran code downloaded from Netlib:
http://netlib.org/opt/praxis The original Fortran code was written by Richard Brent and made available by the Stanford Linear Accelerator Center, dated 3/1/73. The appropriate reference seems to be: Brent (1972)
Specified in NLopt as NLOPT_LN_PRAXIS
This algorithm was originally designed for unconstrained optimization. In NLopt, bound constraints are "implemented" in PRAXIS by the simple expedient of returning infinity (Inf) when the constraints are violated (this is done automaticallyâĂŤyou don't have to do this in your own function). This seems to work, more-or-less, but appears to slow convergence significantly. If you have bound constraints, you are probably better off using COBYLA or BOBYQA.

20.6.5 Nelder-Mead Simplex

My implementation of almost the original Nelder-Mead simplex algorithm (specified in NLopt as NLOPT_LN_NELDERMEAD), as described in: Nelder & Mead (1965)
This method is simple and has demonstrated enduring popularity, despite the later discovery that it fails to converge at all for some functions (and examples may be constructed in which it converges to point that is not a local minimum). Anecdotal evidence suggests that it often performs well even for noisy and/or discontinuous objective functions. I would tend to recommend the Subplex method (below) instead, however.
The main change compared to the 1965 paper is that I implemented explicit support for bound constraints, using essentially the method proposed in: Box (1965)
and later reviewed in: Richardson & Kuester (1973)
Whenever a new point would lie outside the bound constraints, Box advocates moving it "just inside" the constraints by some fixed "small" distance of 10âĹŠ8 or so. I couldn't see any advantage to using a fixed distance inside the constraints, especially if the optimum is on the constraint, so instead I move the point exactly onto the constraint in that case. The danger with implementing bound constraints in this way (or by Box's method) is that you may collapse the simplex into a lower-dimensional subspace. I'm not aware of a better way, however. In any case, this collapse of the simplex is somewhat ameliorated by restarting, such as when Nelder-Mead is used within the Subplex algorithm below.

20.6.6 Sbplx (based on Subplex)

This is my re-implementation of Tom Rowan's "Subplex" algorithm. As Rowan expressed a preference that other implementations of his algorithm use a different name, I called my implementation "Sbplx" (referred to in NLopt as NLOPT_LN_SBPLX).
Subplex (a variant of Nelder-Mead that uses Nelder-Mead on a sequence of subspaces) is claimed to be much more efficient and robust than the original Nelder-Mead, while retaining the latter's facility with discontinuous objectives, and in my experience these claims seem to be true in many

cases. (However, I'm not aware of any proof that Subplex is globally convergent, and perhaps it may fail for some objectives like Nelder-Mead; YMMV.)
I used the description of Rowan's algorithm in his PhD thesis: Rowan (1990)
I would have preferred to use Rowan's original implementation, posted by him on Netlib: http://www.netlib.org/opt/subplex.tgz Unfortunately, the legality of redistributing or modifying this code is unclear, because it lacks anything resembling a license statement. After some friendly emails with Rowan in which he promised to consider providing a clear open-source/free-software license, I lost touch with him and his old email address now seems invalid.
Since the algorithm is not too complicated, however, I just rewrote it. There seem to be slight differences between the behavior of my implementation and his (probably due to different choices of initial subspace and other slight variations, where his paper was ambiguous), but the number of iterations to converge on my test problems seems to be quite close (within Âś10% of the number of function evaluations for most problems).
The only major difference between my implementation and Rowan's, as far as I can tell, is that I implemented explicit support for bound constraints (via the method in the Box paper as described above). This seems to be a big improvement in the case where the optimum lies against one of the constraints.

20.7 NLOPT: Local gradient-based optimization

Of these algorithms, only MMA and SLSQP support arbitrary nonlinear inequality constraints, and only SLSQP supports nonlinear equality constraints; the rest support bound-constrained or unconstrained problems only. (However, any of them can be applied to nonlinearly constrained problems by combining them with the augmented Lagrangian method below.)

20.7.1 MMA (Method of Moving Asymptotes) and CCSA

My implementation of the globally-convergent method-of-moving-asymptotes (MMA) algorithm for gradient-based local optimization, including nonlinear inequality constraints (but not equality constraints), specified in NLopt as NLOPT_LD_MMA, as described in: Svanberg (2002)
This is an improved CCSA ("conservative convex separable approximation") variant of the original MMA algorithm published by Svanberg in 1987, which has become popular for topology optimization. (Note: "globally convergent" does not mean that this algorithm converges to the global optimum; it means that it is guaranteed to converge to some local minimum from any feasible starting point.)
At each point x, MMA forms a local approximation using the gradient of f and the constraint functions, plus a quadratic "penalty" term to make the approximations "conservative" (upper bounds for the exact functions). The precise approximation MMA forms is difficult to describe in a few words, because it includes nonlinear terms consisting of a poles at some distance from x (outside of the current trust region), almost a kind of Pade approximant. The main point is that the approximation is both convex and separable, making it trivial to solve the approximate optimization by a dual method. Optimizing the approximation leads to a new candidate point x. The objective and constraints are evaluated at the candidate point. If the approximations were indeed conservative (upper bounds for the actual functions at the candidate point), then the process is restarted at the new x. Otherwise, the approximations are made more conservative (by increasing the penalty term) and re-optimized.
(If you contact Professor Svanberg, he has been willing in the past to graciously provide you with his original code, albeit under restrictions on commercial use or redistribution. The MMA

implementation in NLopt, however, is completely independent of Svanberg's, whose code we have not examined; any bugs are my own, of course.)
I also implemented another CCSA algorithm from the same paper, NLOPT_LD_CCSAQ: instead of constructing local MMA approximations, it constructs simple quadratic approximations (or rather, affine approximations plus a quadratic penalty term to stay conservative). This is the ccsa_quadratic code. It seems to have similar convergence rates to MMA for most problems, which is not surprising as they are both essentially similar. However, for the quadratic variant I implemented the possibility of preconditioning: including a user-supplied Hessian approximation in the local model. It is easy to incorporate this into the proof in Svanberg's paper, and to show that global convergence is still guaranteed as long as the user's "Hessian" is positive semidefinite, and it practice it can greatly improve convergence if the preconditioner is a good approximation for the real Hessian (at least for the eigenvectors of the largest eigenvalues).

20.7.2 SLSQP

Specified in NLopt as NLOPT_LD_SLSQP, this is a sequential quadratic programming (SQP) algorithm for nonlinearly constrained gradient-based optimization (supporting both inequality and equality constraints), based on the implementation by Dieter Kraft and described in:
Kraft (1988, 1994)
(I believe that SLSQP stands for something like "Sequential Least-Squares Quadratic Programming," because the problem is treated as a sequence of constrained least-squares problems, but such a least-squares problem is equivalent to a QP.) The algorithm optimizes successive second-order (quadratic/least-squares) approximations of the objective function (via BFGS updates), with first-order (affine) approximations of the constraints.
The Fortran code was obtained from the SciPy project, who are responsible for obtaining permission to distribute it under a free-software (3-clause BSD) license.
The code was modified for inclusion in NLopt by S. G. Johnson in 2010, with the following changes. The code was converted to C and manually cleaned up. It was modified to be reentrant (preserving the reverse-communication interface but explicitly saving the state in a data structure). The reverse-communication interface was wrapped with an NLopt-style interface, with NLopt stopping conditions. The inexact line search was modified to evaluate the functions including gradients for the first step, since this removes the need to evaluate the function+gradient a second time for the same point in the common case when the inexact line search concludes after a single step; this is motivated by the fact that NLopt's interface combines the function and gradient computations. Since roundoff errors sometimes pushed SLSQP's parameters slightly outside the bound constraints (not allowed by NLopt), we added checks to force the parameters within the bounds. We fixed a bug in the LSEI subroutine (use of uninitialized variables) for the case where the number of equality constraints equals the dimension of the problem. The LSQ subroutine was modified to handle infinite lower/upper bounds (in which case those constraints are omitted).
Note: Because the SLSQP code uses dense-matrix methods (ordinary BFGS, not low-storage BFGS), it requires $O(n^2)$ storage and $O(n^3)$ time in n dimensions, which makes it less practical for optimizing more than a few thousand parameters

20.7.3 Low-storage BFGS

This algorithm in NLopt (specified by NLOPT_LD_LBFGS), is based on a Fortran implementation of the low-storage BFGS algorithm written by Prof. Ladislav Luksan, and graciously posted online under the GNU LGPL at:

http://www.uivt.cas.cz/ luksan/subroutines.html The original L-BFGS algorithm, based on variable-metric updates via Strang recurrences, was described by the papers:

Nocedal (1980) and Liu & Nocedal (1989).

I converted Prof. Luksan's code to C with the help of f2c, and made a few minor modifications (mainly to include the NLopt termination criteria).

One of the parameters of this algorithm is the number M of gradients to "remember" from previous optimization steps: increasing M increases the memory requirements but may speed convergence. NLopt sets M to a heuristic value by default, but this can be changed by the set_vector_storage function.

20.7.4 Preconditioned truncated Newton

This algorithm in NLopt, is based on a Fortran implementation of a preconditioned inexact truncated Newton algorithm written by Prof. Ladislav Luksan, and graciously posted online under the GNU LGPL at:

http://www.uivt.cas.cz/ luksan/subroutines.html

NLopt includes several variations of this algorithm by Prof. Luksan. First, a variant preconditioned by the low-storage BFGS algorithm with steepest-descent restarting, specified as NLOPT_LD_TNEWTON_PRECOND_RESTART. Second, simplified versions NLOPT_LD_TNEWTON_PR (same without restarting), NLOPT_LD_TNEWTON_RESTART (same without preconditioning), and NLOPT_LD_TNEWTON (same without restarting or preconditioning).

The algorithms are based on the ones described by: Dembo & Steihaug (1982)

I converted Prof. Luksan's code to C with the help of f2c, and made a few minor modifications (mainly to include the NLopt termination criteria).

One of the parameters of this algorithm is the number M of gradients to "remember" from previous optimization steps: increasing M increases the memory requirements but may speed convergence. NLopt sets M to a heuristic value by default, but this can be changed by the set_vector_storage function.

20.7.5 Shifted limited-memory variable-metric

This algorithm in NLopt, is based on a Fortran implementation of a shifted limited-memory variable-metric algorithm by Prof. Ladislav Luksan, and graciously posted online under the GNU LGPL at:

http://www.uivt.cas.cz/ luksan/subroutines.html There are two variations of this algorithm: NLOPT_LD_VAR2, using a rank-2 method, and NLOPT_LD_VAR1, using a rank-1 method.

The algorithms are based on the ones described by: Vlcek & Luksan (2006)

I converted Prof. Luksan's code to C with the help of f2c, and made a few minor modifications (mainly to include the NLopt termination criteria).

One of the parameters of this algorithm is the number M of gradients to "remember" from previous optimization steps: increasing M increases the memory requirements but may speed convergence. NLopt sets M to a heuristic value by default, but this can be changed by the set_vector_storage function.

20.8 NLOPT: Augmented Lagrangian algorithm

20.8.1 Implementation

There is one algorithm in NLopt that fits into all of the above categories, depending on what subsidiary optimization algorithm is specified, and that is the augmented Lagrangian method described in: Conn *et al.* (1991) and Birgin & Martínez (2008)
This method combines the objective function and the nonlinear inequality/equality constraints (if any) in to a single function: essentially, the objective plus a "penalty" for any violated constraints. This modified objective function is then passed to another optimization algorithm with no nonlinear constraints. If the constraints are violated by the solution of this sub-problem, then the size of the penalties is increased and the process is repeated; eventually, the process must converge to the desired solution (if it exists).
The subsidiary optimization algorithm is specified by the nlopt_set_local_optimizer function, described in the NLopt Reference. (Don't forget to set a stopping tolerance for this subsidiary optimizer!) Since all of the actual optimization is performed in this subsidiary optimizer, the subsidiary algorithm that you specify determines whether the optimization is gradient-based or derivative-free. In fact, you can even specify a global optimization algorithm for the subsidiary optimizer, in order to perform global nonlinearly constrained optimization (although specifying a good stopping criterion for this subsidiary global optimizer is tricky).
The augmented Lagrangian method is specified in NLopt as NLOPT_AUGLAG. We also provide a variant, NLOPT_AUGLAG_EQ, that only uses penalty functions for equality constraints, while inequality constraints are passed through to the subsidiary algorithm to be handled directly; in this case, the subsidiary algorithm must handle inequality constraints (e.g. MMA or COBYLA). While NLopt uses an independent re-implementation of the Birgin and MartÃŋnez algorithm, those authors provide their own free-software implementation of the method as part of the TANGO project, and implementations can also be found in semi-free packages like LANCELOT.

Part VI

Appendices

Appendix A

Interfaces

A.1 Interfaces to the C family of languages

A.1.1 GNU Compiler Collection

The GNU Compiler Collection (GCC) is a compiler system produced by the GNU Project supporting various programming languages. GCC is a key component of the GNU toolchain. The Free Software Foundation (FSF) distributes GCC under the GNU General Public License (GNU GPL). GCC has played an important role in the growth of free software, as both a tool and an example.

Originally named the GNU C Compiler, because it only handled the C programming language, GCC 1.0 was released in 1987 and the compiler was extended to compile C++ in December of that year.[1] Front ends were later developed for Objective-C, Objective-C++, Fortran, Java, Ada, and Go among others.[3]

As well as being the official compiler of the unfinished GNU operating system, GCC has been adopted as the standard compiler by most other modern Unix-like computer operating systems, including Linux and the BSD family. A port to RISC OS has also been developed extensively in recent years. There is also an old (3.0) port of GCC to Plan9, running under its ANSI/POSIX Environment (APE).[4] GCC is also available for Microsoft Windows operating systems and for the ARM processor used by many portable devices.

For further information on the GNU Compiler Collection, see Wikipedia: GCC (the text above has been copied from this reference), or the GCC Homepage.

A.1.2 MSVC

Microsoft Visual C++ (often abbreviated as MSVC or VC++) is a commercial (free version available), integrated development environment (IDE) product from Microsoft for the C, C++, and C++/CLI programming languages. It features tools for developing and debugging C++ code, especially code written for the Microsoft Windows API, the DirectX API, and the Microsoft .NET Framework.

Although the product originated as an IDE for the C programming language, the compiler's support for that language conforms only to the original edition of the C standard, dating from 1989. The later revisions of the standard, C99 and C11, are not supported.[41] According to Herb Sutter, the C compiler is only included for "historical reasons" and is not planned to be further

developed. Users are advised to either use only the subset of the C language that is also valid C++, and then use the C++ compiler to compile their code, or to just use a different compiler such as Intel C++ Compiler or the GNU Compiler Collection instead.[42]

For further information on Microsoft Visual C++, see Wikipedia: MSVC (the text above has been copied from this reference), or the MSVC Homepage.

A.1.3 C

In computing, C is a general-purpose programming language initially developed by Dennis Ritchie between 1969 and 1973 at AT&T Bell Labs.[5][6] Like most imperative languages in the ALGOL tradition, C has facilities for structured programming and allows lexical variable scope and recursion, while a static type system prevents many unintended operations. Its design provides constructs that map efficiently to typical machine instructions, and therefore it has found lasting use in applications that had formerly been coded in assembly language, most notably system software like the Unix computer operating system.[7]

C is one of the most widely used programming languages of all time,[8][9] and C compilers are available for the majority of available computer architectures and operating systems.

Many later languages have borrowed directly or indirectly from C, including D, Go, Rust, Java, JavaScript, Limbo, LPC, C#, Objective-C, Perl, PHP, Python, Verilog (hardware description language),[4] and Unix's C shell. These languages have drawn many of their control structures and other basic features from C. Most of them (with Python being the most dramatic exception) are also very syntactically similar to C in general, and they tend to combine the recognizable expression and statement syntax of C with underlying type systems, data models, and semantics that can be radically different. C++ and Objective-C started as compilers that generated C code; C++ is currently nearly a superset of C,[10] while Objective-C is a strict superset of C.[11]

Before there was an official standard for C, many users and implementors relied on an informal specification contained in a book by Dennis Ritchie and Brian Kernighan; that version is generally referred to as "K&R" C. In 1989 the American National Standards Institute published a standard for C (generally called "ANSI C" or "C89"). The next year, the same specification was approved by the International Organization for Standardization as an international standard (generally called "C90"). ISO later released an extension to the internationalization support of the standard in 1995, and a revised standard (known as "C99") in 1999. The current version of the standard (now known as "C11") was approved in December 2011.[12]

For further information on C++, see Wikipedia: C (the text above has been copied from this reference), or the Wikipedia: GCC (the text above has been copied from this reference), or the GCC Homepage.

Example in C

```
#include <iostream>
#include "mpreal.h"

using mpfr::mpreal;
using std::cout;
using std::endl;

// double - version
double schwefel(double x)
{
    return 418.9829 - x * sin(sqrt(abs(x)));
}

//MPFR C - version
void mpfr_schwefel(mpfr_t y, mpfr_t x)
```

```
{
   mpfr_t t;
   mpfr_init(t);
   mpfr_abs(t,x,GMP_RNDN);
   mpfr_sqrt(t,t,GMP_RNDN);
   mpfr_sin(t,t,GMP_RNDN);
   mpfr_mul(t,t,x,GMP_RNDN);
   mpfr_set_str(y,"418.9829",10,GMP_RNDN);
   mpfr_sub(y,y,t,GMP_RNDN);
   mpfr_clear(t);
}

// MPFR C++ - version
mpreal mpfr_schwefel(mpreal& x)
{
    return "418.9829" - x*sin(sqrt(abs(x)));
}



int main(int argc, char* argv[])
{
    const int digits = 50;
    mpreal::set_default_prec(mpfr::digits2bits(digits));
    const mpreal pi        =    mpfr::const_pi();
  mpreal x         =   "-343.5";
    mpreal SResult = mpfr_schwefel(x);
    cout.precision(digits); // Show all the digits
    cout << "pi       =    "<<   pi         << endl;
    cout << "SResult  =    "<<   SResult    << endl;
    return 0;
}
```

A.1.4 Objective C

Objective-C is a general-purpose, object-oriented programming language that adds Smalltalk-style messaging to the C programming language. It is the main programming language used by Apple for the OS X and iOS operating systems, and their respective application programming interfaces (APIs), Cocoa and Cocoa Touch.

The programming language Objective-C was originally developed in the early 1980s. It was selected as the main language used by NeXT for its NeXTSTEP operating system, from which OS X and iOS are derived.[2] Generic Objective-C programs that do not use the Cocoa or Cocoa Touch libraries, or using parts that may be ported or reimplemented for other systems can also be compiled for any system supported by GCC or Clang.

Objective-C source code program files usually have .m filename extensions, while Objective-C header files have .h extensions, the same as for C header files. Objective-C++ files are denoted with a .mm file extension.

For further information on C++, see Wikipedia: Objective C (the text above has been copied from this reference), or the GCC Homepage.

Example in Objective C

```
# import "Forwarder.h"
# import "Recipient.h"

int main(void) {
 Forwarder *forwarder = [Forwarder new];
 Recipient *recipient = [Recipient new];

 [forwarder setRecipient:recipient]; //Set the recipient.
 /*
 * Observe forwarder does not respond to a hello message! It will
 * be forwarded. All unrecognized methods will be forwarded to
 * the recipient
 * (if the recipient responds to them, as written in the Forwarder)
 */
 [forwarder hello];

 [recipient release];
 [forwarder release];

 return 0;
}
```

A.1.5 C++

C++ (pronounced ”see plus plus”) is a statically typed, free-form, multi-paradigm, compiled, general-purpose programming language. It is regarded as an intermediate-level language, as it comprises both high-level and low-level language features.[3] Developed by Bjarne Stroustrup starting in 1979 at Bell Labs, C++ was originally named C with Classes, adding object oriented features, such as classes, and other enhancements to the C programming language. The language was renamed C++ in 1983,[4] as a pun involving the increment operator.

C++ is one of the most popular programming languages[5][6] and is implemented on a wide variety of hardware and operating system platforms. As an efficient compiler to native code, its application domains include systems software, application software, device drivers, embedded software, high-performance server and client applications, and entertainment software such as video games.[7] Several groups provide both free and proprietary C++ compiler software, including the GNU Project, LLVM, Microsoft, Intel and Embarcadero Technologies. C++ has greatly influenced many other popular programming languages, most notably C# and Java.

C++ is also used for hardware design, where the design is initially described in C++, then analyzed, architecturally constrained, and scheduled to create a register-transfer level hardware description language via high-level synthesis.[8]

The language began as enhancements to C, first adding classes, then virtual functions, operator overloading, multiple inheritance, templates and exception handling, among other features. After years of development, the C++ programming language standard was ratified in 1998 as ISO/IEC 14882:1998. The standard was amended by the 2003 technical corrigendum, ISO/IEC 14882:2003. The current standard extending C++ with new features was ratified and published by ISO in September 2011 as ISO/IEC 14882:2011 (informally known as C++11).[9]

For further information on C++, see Wikipedia: C++ (the text above has been copied from this reference), or the GCC Homepage.

Example in C++

```
#include <iostream>
#include "mpreal.h"

int main(int argc, char* argv[])
{
    using mpfr::mpreal;
    using std::cout;
    using std::endl;

    // Required precision of computations in decimal digits
    // Play with it to check different precisions
    const int digits = 50;

    // Setup default precision for all subsequent computations
    // MPFR accepts precision in bits - so we do the conversion
    mpreal::set_default_prec(mpfr::digits2bits(digits));

    // Compute all the vital characteristics of mpreal (in current precision)
    // Analogous to lamch from LAPACK
```

```
    const mpreal one       =   1.0;
    const mpreal zero      =   0.0;
    const mpreal eps       =   std::numeric_limits<mpreal>::epsilon();
    const int   base       =   std::numeric_limits<mpreal>::radix;
    const mpreal prec      =   eps * base;
    const int bindigits    =   std::numeric_limits<mpreal>::digits(); // eqv. to
       mpfr::mpreal::get_default_prec();
    const mpreal rnd       =   std::numeric_limits<mpreal>::round_error();
    const mpreal maxval    =   std::numeric_limits<mpreal>::max();
    const mpreal minval    =   std::numeric_limits<mpreal>::min();
    const mpreal small     =   one / maxval;
    const mpreal sfmin     =   (small > minval) ? small * (one + eps) : minval;
    const mpreal round     =   std::numeric_limits<mpreal>::round_style();
    const int   min_exp    =   std::numeric_limits<mpreal>::min_exponent;
    const mpreal underflow =   std::numeric_limits<mpreal>::min();
    const int   max_exp    =   std::numeric_limits<mpreal>::max_exponent;
    const mpreal overflow  =   std::numeric_limits<mpreal>::max();

    // Additionally compute pi with required accuracy - just for fun :)
    const mpreal pi        =   mpfr::const_pi();

    cout.precision(digits); // Show all the digits
    cout << "pi       =   "<<   pi         << endl;
    cout << "eps      =   "<<   eps        << endl;
    cout << "base     =   "<<   base       << endl;
    cout << "prec     =   "<<   prec       << endl;
    cout << "b.digits =   "<<   bindigits  << endl;
    cout << "rnd      =   "<<   rnd        << endl;
    cout << "maxval   =   "<<   maxval     << endl;
    cout << "minval   =   "<<   minval     << endl;
    cout << "small    =   "<<   small      << endl;
    cout << "sfmin    =   "<<   sfmin      << endl;
    cout << "1/sfmin  =   "<<   1 / sfmin  << endl;
    cout << "round    =   "<<   round      << endl;
    cout << "max_exp  =   "<<   max_exp    << endl;
    cout << "min_exp  =   "<<   min_exp    << endl;
    cout << "underflow =  "<<   underflow  << endl;
    cout << "overflow =   "<<   overflow   << endl;

    return 0;
}
```

A.1.6 Objective C++

Objective-C++ is a language variant accepted by the front-end to the GNU Compiler Collection and Clang, which can compile source files that use a combination of C++ and Objective-C syntax. Objective-C++ adds to C++ the extensions that Objective-C adds to C. As nothing is done to unify the semantics behind the various language features, certain restrictions apply:
A C++ class cannot derive from an Objective-C class and vice versa.
C++ namespaces cannot be declared inside an Objective-C declaration.
Objective-C declarations may appear only in global scope, not inside a C++ namespace
Objective-C classes cannot have instance variables of C++ classes that do not have a default constructor or that have one or more virtual methods,[citation needed] but pointers to C++ objects can be used as instance variables without restriction (allocate them with new in the -init method).
C++ "by value" semantics cannot be applied to Objective-C objects, which are only accessible through pointers.
An Objective-C declaration cannot be within a C++ template declaration and vice versa. However, Objective-C types, (e.g., Classname *) can be used as C++ template parameters.
Objective-C and C++ exception handling is distinct; the handlers of each cannot handle exceptions of the other type. This is mitigated in recent runtimes as Objective-C exceptions are either replaced by C++ exceptions completely (Apple runtime), or partly when Objective-C++ library is linked (GNUstep libobjc2).
Care must be taken since the destructor calling conventions of Objective-C and C++âĂŹs exception run-time models do not match (i.e., a C++ destructor will not be called when an Objective-C exception exits the C++ objectâĂŹs scope). The new 64-bit runtime resolves this by introducing interoperability with C++ exceptions in this sense.[15]
Objective-C blocks and C++11 lambdas are distinct entities, however a block is transparently generated on Mac OS X when passing a lambda where a block is expected.[16]
Objective-C++ is Objective-C (probably with COCOA Framework) with the ability to link with C++ code (probable classes).

Yes, you can use this language in XCODE to develop for Mac OS X, iPhone/iPodTouch, iPad. It works very well.
You don't have to do anything weird in your project to use Objective-C++. Just name your Objective-C files with the extension .mm (instead of .m) and you are good to go.

It is my favorite architecture: develop base class library of my game/application in C++ so I can reuse it in other platforms (Windows, Linux) and use COCOA just for the iPhone/iPad UI specific stuff.

For further information on Objective C++, see Wikipedia: C++ (the text above has been copied from this reference), or the GCC Homepage.

Example in C++

```
#include <iostream>
#include "mpreal.h"

int main(int argc, char* argv[])
{
    using mpfr::mpreal;
    using std::cout;
```

```
    using std::endl;

    // Required precision of computations in decimal digits
    // Play with it to check different precisions
    const int digits = 50;

    // Setup default precision for all subsequent computations
    // MPFR accepts precision in bits - so we do the conversion
    mpreal::set_default_prec(mpfr::digits2bits(digits));

    // Compute all the vital characteristics of mpreal (in current precision)
    // Analogous to lamch from LAPACK
    const mpreal one      =   1.0;
    const mpreal zero     =   0.0;
    const mpreal eps      =   std::numeric_limits<mpreal>::epsilon();
    const int   base      =   std::numeric_limits<mpreal>::radix;
    const mpreal prec     =   eps * base;
    const int bindigits   =   std::numeric_limits<mpreal>::digits(); // eqv. to
       mpfr::mpreal::get_default_prec();
    const mpreal rnd      =   std::numeric_limits<mpreal>::round_error();
    const mpreal maxval   =   std::numeric_limits<mpreal>::max();
    const mpreal minval   =   std::numeric_limits<mpreal>::min();
    const mpreal small    =   one / maxval;
    const mpreal sfmin    =   (small > minval) ? small * (one + eps) : minval;


    // Additionally compute pi with required accuracy - just for fun :)
    const mpreal pi       =   mpfr::const_pi();

    cout.precision(digits); // Show all the digits
    cout << "pi       =   "<<   pi         << endl;
    cout << "eps      =   "<<   eps        << endl;
    cout << "base     =   "<<   base       << endl;
    cout << "prec     =   "<<   prec       << endl;
    cout << "b.digits =   "<<   bindigits  << endl;
    cout << "rnd      =   "<<   rnd        << endl;
    cout << "maxval   =   "<<   maxval     << endl;
    cout << "minval   =   "<<   minval     << endl;
    cout << "small    =   "<<   small      << endl;
    cout << "sfmin    =   "<<   sfmin      << endl;
    cout << "1/sfmin  =   "<<   1 / sfmin  << endl;

    return 0;
}
```

A.2 Component Object Model (COM) Interface

Component Object Model (COM) is a binary-interface standard for software components introduced by Microsoft in 1993. It is used to enable interprocess communication and dynamic object creation in a large range of programming languages. COM is the basis for several other Microsoft

technologies and frameworks, including OLE, OLE Automation, ActiveX, COM+, DCOM, the Windows shell, DirectX, and Windows Runtime

Overview[edit source — edit]The essence of COM is a language-neutral way of implementing objects that can be used in environments different from the one in which they were created, even across machine boundaries. For well-authored components, COM allows reuse of objects with no knowledge of their internal implementation, as it forces component implementers to provide well-defined interfaces that are separated from the implementation. The different allocation semantics of languages are accommodated by making objects responsible for their own creation and destruction through reference-counting. Casting between different interfaces of an object is achieved through the QueryInterface method. The preferred method of inheritance within COM is the creation of sub-objects to which method calls are delegated.

COM is an interface technology defined and implemented as standard only on Microsoft Windows and Apple's Core Foundation 1.3 and later plug-in API,[1] that in any case implement only a subset of the whole COM interface.[2] For some applications, COM has been replaced at least to some extent by the Microsoft .NET framework, and support for Web Services through the Windows Communication Foundation (WCF). However, COM objects can be used with all .NET languages through .NET COM Interop. Networked DCOM uses binary proprietary formats, while WCF encourages the use of XML-based SOAP messaging. COM is very similar to other component software interface technologies, such as CORBA and Java Beans, although each has its own strengths and weaknesses.

Unlike C++, COM provides a stable ABI that does not change between compiler releases.[3] This makes COM interfaces attractive for object-oriented C++ libraries that are to be used by clients compiled using different compiler versions.

For further information, see Wikipedia: Component Object Model (the text above has been copied from this reference).

Missing:
A description of how to build the relevant projects.

A.2.1 VBScript (Windows Script Host)

VBScript (Visual Basic Scripting Edition) is an Active Scripting language developed by Microsoft that is modeled on Visual Basic. It is designed as a âĂIJlightweightâĂİ language with a fast interpreter for use in a wide variety of Microsoft environments. VBScript uses the Component Object Model to access elements of the environment within which it is running; for example, the FileSystemObject (FSO) is used to create, read, update and delete files.

VBScript has been installed by default in every desktop release of Microsoft Windows since Windows 98;[1] in Windows Server since Windows NT 4.0 Option Pack;[2] and optionally with Windows CE (depending on the device it is installed on).

A VBScript script must be executed within a host environment, of which there are several provided with Microsoft Windows, including: Windows Script Host (WSH), Internet Explorer (IE), and Internet Information Services (IIS).[3] Additionally, the VBScript hosting environment is embeddable in other programs, through technologies such as the Microsoft Script Control (msscript.ocx).

VBScript can also be used to create applications that run directly on a workstation running Microsoft Windows. The simplest example is a script that makes use of the Windows Script Host (WSH) environment. Such a script is usually in a stand-alone file with the file extension .vbs. The script can be invoked in two ways. Wscript.exe is used to display output and receive input through a GUI, such as dialog and input boxes. Cscript.exe is used in a command line environment.

VBScript can be included in two other types of scripting files: Windows Script Files, and HTML Applications.

A Windows Script File (WSF) is styled after XML. A WSF file can include multiple VBS files. As a result WSF files provide a means for code reuse: one can write a library of classes or functions in one or more .vbs files, and include those files in one or more WSF files to use and reuse that functionality in a modular way. The files have extension .wsf and can be executed using wscript.exe or cscript.exe, just like a .vbe file.

An HTML Application (HTA) is styled after HTML. The HTML in the file is used to generate the user interface, and a scripting language such as VBScript is used for the program logic. The files have extension .hta and can be executed using mshta.exe.

VBScript (and JScript) can also be used in a Windows Script Component - an ActiveX-enabled script class that can be invoked by other COM-enabled applications.[13] These files have extension .wsc.
For further information on VBScript, see Wikipedia: VBScript (the text above has been copied from this reference).

Example for using the library

```
Option Explicit

Sub Print(s)
WScript.Echo s
End Sub
```

```
Sub DemoBetadist()
Dim mp, x, df1 , df2, px1
Set mp = CreateObject("mpNumerics.mp_Lib")
With mp
'FloatingPointType: Single = 1, Double = 2, Multi = 3, Interval = 4, Decimal = 5,
   Rational = 6
.FloatingPointType() = 3
.Prec10() = 36
Set x = .Real(0.1)
Set df1 = .Real(13.0)
Set df2 = .Real(23.0)
Set px1 = .Real(0.0005)
' pdf = 1, LeftTail = 2, RightTail = 3, LeftQuantile = 4, RightQuantile = 5
Print ("#TableStart#")
Print ("Item" & "|" & "Value")
Print ("Density:" & "|" & .BetaDist(1, x, df1, df2).Str())
Print ("LeftTail:" & "|" & .BetaDist(2, x, df1, df2).Str())
Print ("RightTail:" & "|" & .BetaDist(3, x, df1, df2).Str())
Print ("LeftQuantile:" & "|" & .BetaDist(4, px1, df1, df2).Str())
Print ("RightQuantile:" & "|" & .BetaDist(5, px1, df1, df2).Str())
Print("#TableEnd#")
Print("")
End With
Set mp = Nothing
End Sub

Call DemoBetadist()
```

Example for using Excel

```
Option Explicit

Sub Print(s)
WScript.Echo s
End Sub

Sub DemoExcel()
Dim objExcel 'As Excel.Application
Set objExcel = CreateObject("Excel.Application")
'objExcel.Workbooks.Open("C:\Extra\mpNumerics\Output\mpTemp00.html")
objExcel.Visible = True
objExcel.Workbooks.Add
objExcel.Cells(1, 1).Value = "Test value"
Set objExcel = Nothing
End Sub

Call DemoExcel()
```

A.2.2 JScript (Windows Script Host)

JScript is Microsoft's dialect of the ECMAScript standard[1] that is used in Microsoft's Internet Explorer.

JScript is implemented as a Windows Script engine.[2] This means that it can be "plugged in" to any application that supports Windows Script,[3] such as Internet Explorer, Active Server Pages, and Windows Script Host. It also means that any application supporting Windows Script can use multiple languages (JScript, VBScript, Perl, and others).

JScript was first supported in the Internet Explorer 3.0 browser released in August 1996. Its most recent version is JScript 9.0, included in Internet Explorer 9.

JScript supports conditional compilation, which allows a programmer to selectively execute code within block comments. This is an extension to the ECMAScript standard that is not supported in other JavaScript implementations.

The original JScript is an Active Scripting engine. Like other Active Scripting languages, it is built on the COM/OLE Automation platform and provides scripting capabilities to host applications.

This is the version used when hosting JScript inside a Web page displayed by Internet Explorer, in an HTML application, in classic ASP, in Windows Script Host scripts and several other Automation environments.

JScript is sometimes referred to as "classic JScript" or "Active Scripting JScript" to differentiate it from newer .NET-based versions.

Some versions of JScript are available for multiple versions of Internet Explorer and Windows. For example, JScript 5.7 was introduced with Internet Explorer 7.0 and is also installed for Internet Explorer 6.0 with Windows XP Service Pack 3, while JScript 5.8 was introduced with Internet Explorer 8.0 and is also installed with Internet Explorer 6.0 on Windows Mobile 6.5.

Microsoft's implementation of ECMAScript 5th Edition in Windows 8 Consumer Preview is called JavaScript and the corresponding Visual Studio 11 Express Beta includes a âĂIJcompletely newâĂİ, full-featured JavaScript editor with IntelliSense enhancements for HTML5 and ECMAScript 5 syntax, âĂIJVSDOCâĂİ annotations for multiple overloads, simplified DOM configuration, brace matching, collapsible outlining and âĂIJgo to definitionâĂİ.[6]

For further information on JScript, see Wikipedia: JScript (the text above has been copied from this reference). For further information on JavaScript, see Wikipedia: JavaScript.

Example for using the library

```
var stdin = WScript.StdIn;
var stdout = WScript.StdOut;

var mp = new ActiveXObject ("mpNumerics.mp_Lib");
mp.Prec10 = 60;
mp.FloatingPointType = 3;
var x = mp.Real(2);
var y = mp.Sqrt(x);
var s = y.Str();

stdout.WriteLine("Sqrt(2):");
```

```
stdout.WriteLine(s);
```

Example for using Excel

```
var xls = new ActiveXObject ( "Excel.Application" );
xls.visible = true;
var newBook = xls.Workbooks.Add;
newBook.Worksheets.Add;
newBook.Worksheets(1).Activate;
newBook.Worksheets(1).Cells(1,1).value="First Column, First Cell";
newBook.Worksheets(1).Cells(2,1).value="First Column, Second Cell";
newBook.Worksheets(1).Cells(1,2).value="Second Column, First Cell";
newBook.Worksheets(1).Cells(2,2).value="Second Column, Second Cell";
newBook.Worksheets(1).Name="WorkSheet from Javascript";
// newBook.Worksheets(1).SaveAs("C:\\temp\\TEST2.XLS");
```

A.2.3 Visual Basic for Applications, Visual Basic 6.0

Visual Basic for Applications (VBA) is an implementation of Microsoft's event-driven programming language Visual Basic 6 and its associated integrated development environment (IDE).

Visual Basic for Applications enables building user defined functions, automating processes and accessing Windows API and other low-level functionality through dynamic-link libraries (DLLs). It supersedes and expands on the abilities of earlier application-specific macro programming languages such as Word's WordBasic. It can be used to control many aspects of the host application, including manipulating user interface features, such as menus and toolbars, and working with custom user forms or dialog boxes.

As its name suggests, VBA is closely related to Visual Basic and uses the Visual Basic Runtime Library, but it can normally only run code within a host application rather than as a standalone program. It can, however, be used to control one application from another via OLE Automation. For example, it is used to automatically create a Word report from Excel data, which are automatically collected by Excel from polled observation sensors. VBA has the ability to use (but not create) (ActiveX/COM) DLLs, and later versions add support for class modules.

VBA is built into most Microsoft Office applications, including Office for Mac OS X (apart from version 2008) and other Microsoft applications such as Microsoft MapPoint and Microsoft Visio
For further information, see Wikipedia: VBA (the text above has been copied from this reference).

Visual Basic is a third-generation event-driven programming language and integrated development environment (IDE) from Microsoft for its COM programming model first released in 1991. Microsoft intends Visual Basic to be relatively easy to learn and use.[1][2] Visual Basic was derived from BASIC and enables the rapid application development (RAD) of graphical user interface (GUI) applications, access to databases using Data Access Objects, Remote Data Objects, or ActiveX Data Objects, and creation of ActiveX controls and objects. The scripting language VBScript is a subset of Visual Basic.

A programmer can create an application using the components provided by the Visual Basic program itself. Programs written in Visual Basic can also use the Windows API, but doing so requires external function declarations. Though the program has received criticism for its perceived faults, version 3 of Visual Basic was a runaway commercial success, and many companies offered third party controls greatly extending its functionality.

The final release was version 6 in 1998. Microsoft's extended support ended in March 2008 and the designated successor was Visual Basic .NET (now known simply as Visual Basic).
For further information, see Wikipedia: VB6 (the text above has been copied from this reference).

```
'Imports System
'Imports System.Console
'Imports Microsoft.VisualBasic.Strings
'Imports mpNumericsLib

'Module Module1

Sub Print(s As String)
WriteLine(s)
End Sub
```

```
Sub DemoBetadist()
Dim mp As New mp_Lib
Dim x, df1 , df2, px1 As New mp_Real
With mp
'FloatingPointType: Single = 1, Double = 2, Multi = 3, Interval = 4, Rational = 5
.FloatingPointType() = 3
.Prec10() = 40
x = .Real(0.1)
df1 = .Real(13.0)
df2 = .Real(23.0)
px1 = .Real(0.0005)
' pdf = 1, LeftTail = 2, RightTail = 3, LeftQuantile = 4, RightQuantile = 5
Print ("Density:" & .BetaDist(1, x, df1, df2).Str())
Print ("LeftTail:"& .BetaDist(2, x, df1, df2).Str())
Print ("RightTail:" & .BetaDist(3, x, df1, df2).Str())
Print ("LeftQuantile:" & .BetaDist(4, px1, df1, df2).Str())
Print ("RightQuantile:" & .BetaDist(5, px1, df1, df2).Str())
End With
mp = Nothing
End Sub

Sub Main()
Call DemoBetadist()
End Sub

'End Module
```

A.2.4 OpenOffice Basic

OpenOffice Basic (formerly known as StarOffice Basic or StarBasic or OOoBasic) is a dialect of the programming language BASIC that is included with the OpenOffice, StarOffice and LibreOffice office suites.

Although Openoffice Basic itself is similar to other dialects of Basic, such as Microsoft's VBA, the application programming interface (API) is very different, as the example below of a macro illustrates. While there is a much easier way to obtain the "paragraph count" document property, the example shows the fundamental methods for accessing each paragraph in a text document, sequentially.

For further information, see Wikipedia: OpenOffice Basic (the text above has been copied from this reference).

For help regarding the language, see the OpenOffice.org BASIC Programming Guide. Information on the OpenOffice API is available from OpenOffice API.

```
Sub ParaCount
'
' Count number of paragraphs in a text document
'
Dim Doc As Object, Enum As Object, TextEl As Object, Count As Long
Doc = ThisComponent
' Is this a text document?
If Not Doc.SupportsService("com.sun.star.text.TextDocument") Then
MsgBox "This macro must be run from a text document", 64, "Error"
Exit Sub
End If
Count = 0
' Examine each component - paragraph or table?
Enum = Doc.Text.CreateEnumeration
While Enum.HasMoreElements
TextEl = Enum.NextElement
' Is the component a paragraph?
If TextEl.SupportsService("com.sun.star.text.Paragraph") Then
Count = Count + 1
End If
Wend
'Display result
MsgBox Count, 0, "Paragraph Count"
End Sub
```

Example for using the library

```
Sub DemoBetadist()
Dim mp, x, df1 , df2, px1
Set mp = CreateObject("mpNumerics.mp_Lib")
With mp
'FloatingPointType: Single = 1, Double = 2, Multi = 3, Interval = 4, Decimal = 5,
   Rational = 6
.FloatingPointType() = 3
.Prec10() = 36
```

```
Set x = .Real(0.1)
Set df1 = .Real(13.0)
Set df2 = .Real(23.0)
Set px1 = .Real(0.0005)
' pdf = 1, LeftTail = 2, RightTail = 3, LeftQuantile = 4, RightQuantile = 5
Print ("#TableStart#")
Print ("Item" & "|" & "Value")
Print ("Density:" & "|" & .BetaDist(1, x, df1, df2).Str())
Print ("LeftTail:" & "|" & .BetaDist(2, x, df1, df2).Str())
Print ("RightTail:" & "|" & .BetaDist(3, x, df1, df2).Str())
Print ("LeftQuantile:" & "|" & .BetaDist(4, px1, df1, df2).Str())
Print ("RightQuantile:" & "|" & .BetaDist(5, px1, df1, df2).Str())
Print("#TableEnd#")
Print("")
End With
Set mp = Nothing
End Sub
```

A.2.5 Lua

Lua is a lightweight multi-paradigm programming language designed as a scripting language with ”extensible semantics” as a primary goal. Lua is cross-platform since it is written in ISO C. Lua has a relatively simple C API, thus ”Lua is especially useful for providing end users with an easy way to program the behavior of a software product without getting too far into its innards.”

Lua is commonly described as a âĂIJmulti-paradigmâĂİ language, providing a small set of general features that can be extended to fit different problem types, rather than providing a more complex and rigid specification to match a single paradigm. Lua, for instance, does not contain explicit support for inheritance, but allows it to be implemented relatively easily with metatables. Similarly, Lua allows programmers to implement namespaces, classes, and other related features using its single table implementation; first-class functions allow the employment of many powerful techniques from functional programming; and full lexical scoping allows fine-grained information hiding to enforce the principle of least privilege.

In general, Lua strives to provide flexible meta-features that can be extended as needed, rather than supply a feature-set specific to one programming paradigm. As a result, the base language is light âĂŞ the full reference interpreter is only about 180 kB compiled[1] âĂŞ and easily adaptable to a broad range of applications.

Lua is a dynamically typed language intended for use as an extension or scripting language, and is compact enough to fit on a variety of host platforms. It supports only a small number of atomic data structures such as boolean values, numbers (double-precision floating point by default), and strings. Typical data structures such as arrays, sets, lists, and records can be represented using LuaâĂŹs single native data structure, the table, which is essentially a heterogeneous associative array.

Lua implements a small set of advanced features such as first-class functions, garbage collection, closures, proper tail calls, coercion (automatic conversion between string and number values at run time), coroutines (cooperative multitasking) and dynamic module loading.

By including only a minimum set of data types, Lua attempts to strike a balance between power and size.
For further information on Lua, see Wikipedia: Lua (the text above has been copied from this reference), or the Lua for Windows Homepage.

Example for using the library

```
--Enable COM support
require("luacom")

--Load the mpNumerics library
mp = luacom.CreateObject("mpNumerics.mp_Lib")

--Set Floating point type to MPFR with 60 decimal digits precision
mp.FloatingPointType = 3
mp.Prec10 = 60

--Assign values to x1 and x2
x1 = mp:Real(4.5)
x2 = mp:Real("1.1")
```

```
--Calculate x3 = x1 / x2
x3 = x1:Div(x2)

--Print the value of x3
print ("Result: ", x3:Str())
```

Example for using Excel

```
require('luacom')
excel = luacom.CreateObject("Excel.Application")
excel.Visible = true
wb = excel.Workbooks:Add()
ws = wb.Worksheets(1)

for i=1, 20 do
ws.Cells(i,1).Value2 = i
end

-- excel.DisplayAlerts = false
-- excel:Quit()
-- excel = nil
```

A.2.6 Ruby

Ruby is a dynamic, reflective, general-purpose object-oriented programming language that combines syntax inspired by Perl with Smalltalk-like features. It was also influenced by Eiffel and Lisp.[8] Ruby was first designed and developed in the mid-1990s by Yukihiro "Matz" Matsumoto in Japan.

Ruby supports multiple programming paradigms, including functional, object oriented and imperative. It also has a dynamic type system and automatic memory management; it is therefore similar in varying respects to Smalltalk, Python, Perl, Lisp, Dylan, and CLU.

The syntax of Ruby is broadly similar to that of Perl and Python. Class and method definitions are signaled by keywords. In contrast to Perl, variables are not obligatorily prefixed with a sigil. When used, the sigil changes the semantics of scope of the variable. One difference from C and Perl is that keywords are typically used to define logical code blocks, without braces (i.e., pair of and). For practical purposes there is no distinction between expressions and statements.[39] Line breaks are significant and taken as the end of a statement; a semicolon may be equivalently used. Unlike Python, indentation is not significant.

One of the differences of Ruby compared to Python and Perl is that Ruby keeps all of its instance variables completely private to the class and only exposes them through accessor methods (attr_writer, attr_reader, etc.). Unlike the "getter" and "setter" methods of other languages like C++ or Java, accessor methods in Ruby can be created with a single line of code via metaprogramming; however, accessor methods can also be created in the traditional fashion of C++ and Java. As invocation of these methods does not require the use of parentheses, it is trivial to change an instance variable into a full function, without modifying a single line of code or having to do any refactoring achieving similar functionality to C# and VB.NET property members.

For further information on Ruby, see Wikipedia: Ruby (the text above has been copied from this reference), or the Ruby Homepage. An easy-to-install package for Windows can be found at RubyForge.

Example for using the library

```
#Enable COM support
require 'win32ole'

#Load the mpNumerics library
mp = WIN32OLE.new("mpNumerics.mp_Lib")

#Set Floating point type to MPFR with 60 decimal digits precision
mp.FloatingPointType = 3
mp.Prec10 = 60

#Assign values to x1 and x2
x1 = mp.Real(4.5)
x2 = mp.Real("1.1")

#Calculate x3 = x1 / x2
x3 = x1.Div(x2)

#Print the value of x3
puts x3.Str
```

```
obj = WIN32OLE_VARIANT.new([[1.345345,2,3],[4,5,6]])
p obj[0,0]
p obj[1,0]
obj[0,0] = 7
p obj.value
```

Example for using Excel

```
require 'win32ole'
xl = WIN32OLE.new("Excel.Application")

puts "Excel failed to start" unless xl

xl.Visible = true

workbook = xl.Workbooks.Add
sheet = workbook.Worksheets(1)

#create some fake data
data_a = []
(1..10).each{|i| data_a.push i }

data_b = []
(1..10).each{|i| data_b.push((rand * 100).to_i) }

#fill the worksheet with the fake data
#showing 3 ways to populate cells with values
(1..10).each do |i|
sheet.Range("A#{i}").Select
xl.ActiveCell.Formula = data_a[i-1]


sheet.Range("B#{i}").Formula = data_b[i-1]


cell = sheet.Range("C#{i}")
cell.Formula = "=A#{i} - B#{i}"

end

#chart type constants (via http://support.microsoft.com/kb/147803)
xlArea = 1
xlBar = 2

xlColumn = 3
xlLine = 4
xlPie = 5
xlRadar = -4151

xlXYScatter = -4169
```

```
xlCombination = -4111
xl3DArea = -4098

xl3DBar = -4099
xl3DColumn = -4100
xl3DLine = -4101

xl3DPie = -4102
xl3DSurface = -4103
xlDoughnut = -4120

#creating a chart
chart_object = sheet.ChartObjects.Add(10, 80, 500, 250)

chart = chart_object.Chart
chart_range = sheet.Range("A1", "B10")

chart.SetSourceData(chart_range, nil)
chart.ChartType = xlXYScatter

#get the value from a cell

val = sheet.Range("C1").Value
puts val

#saving as pre-2007 format
excel97_2003_format = -4143

pwd = Dir.pwd.gsub('/','\\') << '\\'

#otherwise, it sticks it in default save directory- C:\Users\Sam\Documents on my
    system
#workbook.SaveAs("#{pwd}whatever.xls", excel97_2003_format)

#xl.Quit
```

A.2.7 PHP CLI

PHP is a server-side scripting language designed for web development but also used as a general-purpose programming language. PHP is now installed on more than 244 million websites and 2.1 million web servers.[2] Originally created by Rasmus Lerdorf in 1995, the reference implementation of PHP is now produced by The PHP Group.[3] While PHP originally stood for Personal Home Page,[4] it now stands for PHP: Hypertext Preprocessor, a recursive acronym.[5]

PHP code is interpreted by a web server with a PHP processor module which generates the resulting web page: PHP commands can be embedded directly into an HTML source document rather than calling an external file to process data. It has also evolved to include a command-line interface capability and can be used in standalone graphical applications.[6]

PHP is free software released under the PHP License, which is incompatible with the GNU General Public License (GPL) due to restrictions on the usage of the term PHP.[7] PHP can be deployed on most web servers and also as a standalone shell on almost every operating system and platform, free of charge.[8]

The PHP interpreter only executes PHP code within its delimiters. Anything outside its delimiters is not processed by PHP (although non-PHP text is still subject to control structures described in PHP code) . The most common delimiters are ¡?php to open and ?¿ to close PHP sections. ¡script language=”php”¿ and ¡/script¿ delimiters are also available, as are the shortened forms ¡? or ¡?= (which is used to echo back a string or variable) and ?¿ as well as ASP-style short forms ¡% or ¡%= and %¿. While short delimiters are used, they make script files less portable as support for them can be disabled in the PHP configuration, and so they are discouraged.[37] The purpose of all these delimiters is to separate PHP code from non-PHP code, including HTML.[38]

The first form of delimiters, ¡?php and ?¿, in XHTML and other XML documents, creates correctly formed XML ’processing instructions’.[39] This means that the resulting mixture of PHP code and other markup in the server-side file is itself well-formed XML.

Variables are prefixed with a dollar symbol, and a type does not need to be specified in advance. Unlike function and class names, variable names are case sensitive. Both double-quoted (””) and heredoc strings provide the ability to interpolate a variable’s value into the string.[40] PHP treats newlines as whitespace in the manner of a free-form language (except when inside string quotes), and statements are terminated by a semicolon.[41] PHP has three types of comment syntax: /* */ marks block and inline comments; // as well as # are used for one-line comments.[42] The echo statement is one of several facilities PHP provides to output text, e.g., to a web browser.

In terms of keywords and language syntax, PHP is similar to most high level languages that follow the C style syntax. if conditions, for and while loops, and function returns are similar in syntax to languages such as C, C++, C#, Java and Perl.

PHP CLI is a short for PHP Command Line Interface. As the name implies, this is a way of using PHP in the system command line. Or by other words it is a way of running PHP Scripts that aren’t on a web server (such as Apache web server or Microsoft IIS). People usually treat PHP as web development, server side tool. However, PHP CLI applies all advantages of PHP to shell scripting allowing to create either service side supporting scripts or system application even with GUI.

For further information on PHP, see Wikipedia: PHP (the text above has been copied from this reference), or the PHP for Windows Homepage, or the PHP CLI Homepage.

Example for using the library

```
#PHP Command line example
<?php
#Load the mpNumerics library
$mp = new COM("mpNumerics.mp_Lib") or die("Cannot open library");

#Set Floating point type to MPFR with 60 decimal digits precision
$mp->FloatingPointType = 3;
$mp->Prec10 = 60;

#Assign values to x1 and x2
$x1 = $mp->Real(4.5);
$x2 = $mp->Real('1.1');

#Calculate x3 = x1 / x2
$x3 = $x1->Div($x2);

echo "Hello world of PHP CLI! \n";

#Print the value of x3
echo $x3->Str();
?>
```

Example for using Excel

```
#PHP.ini has to be stored in the PDP application directory (derived from the sample
    inis)
#Need to enable the win extension_dir directive
#Need to add:

#[COM_DOT_NET]
#extension=php_com_dotnet.dll

#as explained in
#http://www.php.net/manual/en/com.installation.php

<?php
$xlApp = new COM("Excel.Application");
$xlApp->Workbooks->Add();
$xlApp->Range("A1:C6")->Select();
$xlApp->ActiveCell->Formula = "Hello World!";
$xlApp->Visible = 1;
?>
```

A.2.8 Perl

Perl is a family of high-level, general-purpose, interpreted, dynamic programming languages. The languages in this family include Perl 5 and Perl 6.[4]

Though Perl is not officially an acronym,[5] there are various backronyms in use, such as: Practical Extraction and Reporting Language.[6] Perl was originally developed by Larry Wall in 1987 as a general-purpose Unix scripting language to make report processing easier.[7] Since then, it has undergone many changes and revisions. The latest major stable revision of Perl 5 is 5.18, released in May 2013. Perl 6, which began as a redesign of Perl 5 in 2000, eventually evolved into a separate language. Both languages continue to be developed independently by different development teams and liberally borrow ideas from one another.

The Perl languages borrow features from other programming languages including C, shell scripting (sh), AWK, and sed.[8] They provide powerful text processing facilities without the arbitrary data-length limits of many contemporary Unix tools,[9] facilitating easy manipulation of text files. Perl 5 gained widespread popularity in the late 1990s as a CGI scripting language, in part due to its parsing abilities.[10]

In addition to CGI, Perl 5 is used for graphics programming, system administration, network programming, finance, bioinformatics, and other applications. It's nicknamed "the Swiss Army chainsaw of scripting languages" because of its flexibility and power,[11] and possibly also because of its perceived "ugliness".[12] In 1998, it was also referred to as the "duct tape that holds the Internet together", in reference to its ubiquity and perceived inelegance.[13]

The overall structure of Perl derives broadly from C. Perl is procedural in nature, with variables, expressions, assignment statements, brace-delimited blocks, control structures, and subroutines.

Perl also takes features from shell programming. All variables are marked with leading sigils, which unambiguously identify the data type (for example, scalar, array, hash) of the variable in context. Importantly, sigils allow variables to be interpolated directly into strings. Perl has many built-in functions that provide tools often used in shell programming (although many of these tools are implemented by programs external to the shell) such as sorting, and calling on operating system facilities.

Perl takes lists from Lisp, hashes ("associative arrays") from AWK, and regular expressions from sed. These simplify and facilitate many parsing, text-handling, and data-management tasks. Also shared with Lisp are the implicit return of the last value in a block, and the fact that all statements have a value, and thus are also expressions and can be used in larger expressions themselves.

Perl 5 added features that support complex data structures, first-class functions (that is, closures as values), and an object-oriented programming model. These include references, packages, class-based method dispatch, and lexically scoped variables, along with compiler directives (for example, the strict pragma). A major additional feature introduced with Perl 5 was the ability to package code as reusable modules.

All versions of Perl do automatic data-typing and automatic memory management. The interpreter knows the type and storage requirements of every data object in the program; it allocates and frees storage for them as necessary using reference counting (so it cannot deallocate circular data structures without manual intervention). Legal type conversions âĂŤ for example, conversions from number to string âĂŤ are done automatically at run time; illegal type conversions are fatal errors.

ActivePerl is a closed source distribution from ActiveState that has regular releases that track the core Perl releases.[65] The distribution also includes the Perl package manager (PPM),[66] a popular tool for installing, removing, upgrading, and managing the use of common Perl modules.

Strawberry Perl is an open source distribution for Windows. It has had regular, quarterly releases since January 2008, including new modules as feedback and requests come in. Strawberry Perl aims to be able to install modules like standard Perl distributions on other platforms, including compiling XS modules.

For further information on Perl, see Wikipedia: Perl (the text above has been copied from this reference), or the Perl Homepage.

ActivePerl is available from ActivePerl Homepage. This distribution includes support for COM. See here for an example.

Example for using the library

```
#Enable COM support
use Win32::OLE;

#Load the mpNumerics library
$mp = Win32::OLE->new('mpNumerics.mp_Lib');

#Set Floating point type to MPFR with 60 decimal digits precision
$mp->{FloatingPointType} = 3;
$mp->{Prec10} = 60;

#Assign values to x1 and x2
$x1 = $mp->Real(4.5);
$x2 = $mp->Real('1.1');

#Calculate x3 = x1 / x2
$x3 = $x1->Div($x2);

#Print the value of x3
print $x3->Str();

# Wait for user input...
# print "Press <return> to continue...";
# $x = <STDIN>;
```

Example for using the Excel

```
use Win32::OLE;

# Start Excel and make it visible
$xlApp = Win32::OLE->new('Excel.Application');
$xlApp->{Visible} = 1;

# Create a new workbook
$xlBook = $xlApp->Workbooks->Add;

# Our data that we will add to the workbook...
```

```
$mydata = [["Item",   "Category", "Price"],
["Nails",   "Hardware", "5.25"],
["Shirt",   "Clothing", "23.00"],
["Hammer",  "Hardware", "16.25"],
["Sandwich", "Food",    "5.00"],
["Pants",   "Clothing", "31.00"],
["Drinks",  "Food",     "2.25"]];

# Write all the data at once...
$rng = $xlBook->ActiveSheet->Range("A1:C7");
$rng->{Value} = $mydata;

# Create a PivotTable for the data...
$tbl = $xlBook->ActiveSheet->PivotTableWizard(1, $rng, "", "MyPivotTable");

# Set pivot fields...
$tbl->AddFields("Category", "Item");
$tbl->PivotFields("Price")->{Orientation} = 4; # 4=xlDataField

# Create a chart too...
$chart = $xlBook->Charts->Add;
$chart->SetSourceData($rng, 2);
$chart->{ChartType} = 70; # 3D-pie chart
$chart->Location(2, "Sheet4");

# Wait for user input...
# print "Press <return> to continue...";
# $x = <STDIN>;

# Clean up
# $xlBook->{Saved} = 1;
# $xlApp->Quit;
# $xlBook = 0;
# $xlApp = 0;
# $xlApp = 0;
print "All done.";
```

A.2.9 Python

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C. The language provides constructs intended to enable clear programs on both a small and large scale.

Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.

Like other dynamic languages, Python is often used as a scripting language, but is also used in a wide range of non-scripting contexts. Using third-party tools, Python code can be packaged into standalone executable programs. Python interpreters are available for many operating systems.

CPython, the reference implementation of Python, is free and open source software and has a community-based development model, as do nearly all of its alternative implementations. CPython is managed by the non-profit Python Software Foundation.

For further information on Python, see Wikipedia: Python (the text above has been copied from this reference), or the Python Homepage. Support for COM is included in the distribution of the ActivePython Community Edition.

Python can use GMP und MPFR thanks to GMPY2, with documentation here.

IPython is an integrations platform for various scientific libraries (NumPy, SciPy, matlibplot, pandas etc.) http://ipython.org/. Popular distributions are the
Community Edition of Anaconda: http://docs.continuum.io/anaconda/index.html,

Book recommendation: McKinney (2012).

Example for using the library

```
#Enable COM support
from win32com.client import Dispatch

#Load the mpNumerics library
mp = Dispatch("mpNumerics.mp_Lib")

#Set Floating point type to MPFR with 60 decimal digits precision
mp.FloatingPointType = 3
mp.Prec10 = 60

#Assign values to x1 and x2
x1 = mp.Real(4.5)
x2 = mp.Real(1.21)

#Calculate x3 = x1 / x2
x3 = x1.Div(x2)

#Print the value of x3
print (x3.Str())
```

Example for using Excel

```
#Enable COM support
from win32com.client import Dispatch

#Load the Excel library
xl = Dispatch("Excel.Application")
xl.Visible = 1
xl.Workbooks.Add()
xl.Cells(1,1).Value = "Hello442"
print("From Python")
```

To compile the mpmath library libraries, Python 2.7 is required.

A.2.9.1 Downloading and installing Python 2.7

ActivePython is ActiveState's complete and ready-to-install distribution of Python. It provides a one-step installation of all essential Python modules, as well as extensive documentation. The Windows distribution ships with PyWin32 – a suite of Windows tools developed by Mark Hammond, including bindings to the Win32 API and Windows COM. ActivePython can be downloaded from

http://www.activestate.com/activepython/downloads.

The latest release version of the 2.7x series is 2.7.6.9. You need to download 2 separate files to support compilation of both 32 bit and 64 bit dlls.

A.2.10 R (Statistical System)

R is a free software programming language and a software environment for statistical computing and graphics. The R language is widely used among statisticians and data miners for developing statistical software[2][3] and data analysis.[3] Polls and surveys of data miners are showing R's popularity has increased substantially in recent years.[4][5][6]

R is an implementation of the S programming language combined with lexical scoping semantics inspired by Scheme. S was created by John Chambers while at Bell Labs. R was created by Ross Ihaka and Robert Gentleman[7] at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team, of which Chambers is a member. R is named partly after the first names of the first two R authors and partly as a play on the name of S.[8]

R is a GNU project.[9][10] The source code for the R software environment is written primarily in C, Fortran, and R.[11] R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems. R uses a command line interface; however, several graphical user interfaces are available for use with R.

R provides a wide variety of statistical and graphical techniques, including linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, clustering, and others. R is easily extensible through functions and extensions, and the R community is noted for its active contributions in terms of packages. There are some important differences, but much code written for S runs unaltered. Many of R's standard functions are written in R itself, which makes it easy for users to follow the algorithmic choices made. For computationally intensive tasks, C, C++, and Fortran code can be linked and called at run time. Advanced users can write C, C++[12] or Java[13] code to manipulate R objects directly.

R is highly extensible through the use of user-submitted packages for specific functions or specific areas of study. Due to its S heritage, R has stronger object-oriented programming facilities than most statistical computing languages. Extending R is also eased by its lexical scoping rules.[14]

Another strength of R is static graphics, which can produce publication-quality graphs, including mathematical symbols. Dynamic and interactive graphics are available through additional packages.[15]

R has its own LaTeX-like documentation format, which is used to supply comprehensive documentation, both on-line in a number of formats and in hard copy.

R is an interpreted language; users typically access it through a command-line interpreter. If a user types "2+2" at the R command prompt and presses enter, the computer replies with "4", as shown below:

```
¿ 2+2
[1] 4
```

Like many other languages, R supports matrix arithmetic. R's data structures include scalars, vectors, matrices, data frames (similar to tables in a relational database) and lists.[16] R's extensible object-system includes objects for (among others): regression models, time-series and geo-spatial coordinates.

R supports procedural programming with functions and, for some functions, object-oriented programming with generic functions. A generic function acts differently depending on the type of arguments passed to it. In other words, the generic function dispatches the function (method) specific to that type of object. For example, R has a generic print() function that can print almost

every type of object in R with a simple "print(objectname)" syntax.

Although mostly used by statisticians and other practitioners requiring an environment for statistical computation and software development, R can also operate as a general matrix calculation toolbox - with performance benchmarks comparable to GNU Octave or MATLAB.[17]

For further information on R, see Wikipedia: R (the text above has been copied from this reference), or the R Homepage.

COM support can be obtained by installing the R RDCOMClient
Installation of the binary should be as straightforward as any other R package for Windows, e.g. use the command

```
install.packages("RDCOMClient", repos = "http://www.omegahat.org/R")
```

or use the Packages menu and make certain to include the Omegahat repository in the list of repositories to search.
There exists also a commercial R for Excel distribution.

A popular GUI for R is Rstudio.
Within RStudio:
Tools - Install Packages.
Type RD in the dialogue box.
RDCOMClient should appear in the drop down box.
Select RDCOMClient and click on Install.
Needs to be done separately for 32 bit and 64 bit.

R contains packages which provide interfaces to
GMP (http://mulcyber.toulouse.inra.fr/projects/gmp) and
MPFR (http://rmpfr.r-forge.r-project.org/).

Book recommendation: Adler (2012).
Book recommendation: Verzani (2011).
Book recommendation: Chang (2012).

Example for using the library

```
#Enable COM support
require("RDCOMClient")

#Load the mpNumerics library
mp = COMCreate("mpNumerics.mp_Lib")

#Set Floating point type to MPFR with 60 decimal digits precision
mp[["Prec10"]] = 160
mp[["FloatingPointType"]] = 3

#Assign values to x1 and x2
x1 = mp$Real("4.5")
x2 = mp$Real("1.1")

#Perform arithmetic operations
x3 = x1$Plus(x2)
```

```
x4 = x1$Div(x2)

#Display output
mp[["Prec10"]]
x3$Str()
x4$Str()
```

Example for using Excel

```
#Load Library
require("rcom")

#Create instance of Excel
xlApp = comCreateObject("Excel.Application")

#Add 1 workbook and make it visible
wb = xlApp[["Workbooks"]]$Add()
xlApp[["Visible"]] = TRUE

#Display the name of the 1st worksheet
ws = wb[["Worksheets", 1]]
wsname = ws[["Name"]]
wsname

#Assign values to a range
mrange = ws[["Range", "A1:B10"]]
mrange[["Value"]] = 10.3

#Display the values of the range
d = mrange[["Value"]]
d
$
```

A.2.11 MatLab (COM interface)

MATLAB has already been introduced in section A.3.8 Apart from the the .NET interface described in this section, MATLAB has also a COM interface. Its use is illustrated by the examples below.

Example for using the library via COM

```
%Open a COM server on Matlab

mp = actxserver('mpNumerics.mp_Lib');
mp.Prec10 = 60;
mp.FloatingPointType = 3;

x = mp.Real(2);
y = mp.Sqrt(x);
s = y.Str();

s2 = char(s);
fprintf('s is equal to %s.\n',s2);

quit;
```

Example for using Excel

```
%Open a COM server on Matlab

x = 4.3;
fprintf('x is equal to %6.2f.\n',x);

Excel = actxserver('Excel.Application');

Excel.Workbooks.Add();
Excel.Visible = true;
quit;
```

A.3 Languages with CLR Support

A.3.1 Visual Basic .NET

Visual Basic .NET (VB.NET) is an object-oriented computer programming language that can be viewed as an evolution of the classic Visual Basic (VB), implemented on the .NET Framework. Microsoft currently supplies two main editions of IDEs for developing in Visual Basic: Microsoft Visual Studio 2012, which is commercial software and Visual Basic Express Edition 2012, which is free of charge. The command-line compiler, VBC.EXE, is installed as part of the freeware .NET Framework SDK. Mono also includes a command-line VB.NET compiler. The most recent version is VB 2012, which was released on August 15, 2012.

A.3.1.1 Visual Basic 2005

Visual Basic 2005 was the name used to refer to Visual Basic .NET, as Microsoft decided to drop the .NET portion of the title.
For this release, Microsoft added many features, including:
Edit and Continue
Design-time expression evaluation.
The My pseudo-namespace (overview, details), which provides easy access to certain areas of the .NET Framework that otherwise require significant code to access dynamically generated classes (notably My.Forms)
The Using keyword, simplifying the use of objects that require the Dispose pattern to free resources
Just My Code, which when debugging hides (steps over) boilerplate code written by the Visual Studio .NET IDE and system library code
Data Source binding, easing database client/server development
Generics
Partial classes, a method of defining some parts of a class in one file and then adding more definitions later; particularly useful for integrating user code with auto-generated code
Operator overloading and nullable Types
Support for unsigned integer data types commonly used in other languages

A.3.1.2 Visual Basic 2008

Visual Basic 2008 was released together with the Microsoft .NET Framework 3.5 on 19 November 2007.
For this release, Microsoft added many features, including:
A true conditional operator, "If(condition as boolean, truepart, falsepart)", to replace the "IIf" function. Anonymous types
Support for LINQ
Lambda expressions
XML Literals
Type Inference
Extension methods

A.3.1.3 Visual Basic 2010

In April 2010, Microsoft released Visual Basic 2010. Microsoft had planned to use the Dynamic Language Runtime (DLR) for that release[8] but shifted to a co-evolution strategy between Visual

Basic and sister language C# to bring both languages into closer parity with one another. Visual Basic's innate ability to interact dynamically with CLR and COM objects has been enhanced to work with dynamic languages built on the DLR such as IronPython and IronRuby.[9] The Visual Basic compiler was improved to infer line continuation in a set of common contexts, in many cases removing the need for the "_" line continuation character. Also, existing support of inline Functions was complemented with support for inline Subs as well as multi-line versions of both Sub and Function lambdas.[10]

A.3.1.4 Visual Basic 2012

The latest version of Visual Basic .NET, which uses .NET framework 4.5.
Async Feature,
Iterators,
Call Hierarchy,
Caller Information and
Global Keyword in Namespace Statements
are some of the major features introduced in this version of VB.

A.3.1.5 Relation to older versions of Visual Basic

Whether Visual Basic .NET should be considered as just another version of Visual Basic or a completely different language is a topic of debate. This is not obvious, as once the methods that have been moved around and that can be automatically converted are accounted for, the basic syntax of the language has not seen many "breaking" changes, just additions to support new features like structured exception handling and short-circuited expressions. Two important data type changes occurred with the move to VB.NET. Compared to VB6, the Integer data type has been doubled in length from 16 bits to 32 bits, and the Long data type has been doubled in length from 32 bits to 64 bits. This is true for all versions of VB.NET. A 16-bit integer in all versions of VB.NET is now known as a Short. Similarly, the Windows Forms GUI editor is very similar in style and function to the Visual Basic form editor.

The version numbers used for the new Visual Basic (7, 7.1, 8, 9, ...) clearly imply that it is viewed by Microsoft as still essentially the same product as the old Visual Basic.

The things that have changed significantly are the semanticsâĂŤfrom those of an object-based programming language running on a deterministic, reference-counted engine based on COM to a fully object-oriented language backed by the .NET Framework, which consists of a combination of the Common Language Runtime (a virtual machine using generational garbage collection and a just-in-time compilation engine) and a far larger class library. The increased breadth of the latter is also a problem that VB developers have to deal with when coming to the language, although this is somewhat addressed by the My feature in Visual Studio 2005.

The changes have altered many underlying assumptions about the "right" thing to do with respect to performance and maintainability. Some functions and libraries no longer exist; others are available, but not as efficient as the "native" .NET alternatives. Even if they compile, most converted VB6 applications will require some level of refactoring to take full advantage of the new language. Documentation is available to cover changes in the syntax, debugging applications, deployment and terminology.[11]

For further information on Visual Basic .NET, see Wikipedia: Visual Basic .NET (the text above has been copied from this reference).

Example for using the library

```
Imports System
Imports System.Console
Imports Microsoft.VisualBasic
Imports Microsoft.VisualBasic.Strings
Imports MatrixClass2

Module Module1
Sub Main()
mp.Prec10() = 100 : mp.FloatingPointType() = 3
Dim Y1, Y2, Y3, Y4 As New mpNum
Y1 = mp.Sqrt(2)
Writeline("#Sqrt(12): ")
Writeline("@" & Y1)
Y2 = Sqrt(2)
Writeline("#Sqrt(12): ")
Writeline("@" & Y2)
Y3 = Y1 - Y2
Y4 = Y3 + CNum("1.4")
Writeline("#Diff:")
Writeline("@" & Y4)
End Sub
End Module
```

Example for using Excel

```
Imports System
Imports System.Console
Imports Microsoft.VisualBasic
Imports Microsoft.VisualBasic.Strings

Module Module1
Sub DemoExcel()
Dim objExcel As Object
objExcel = CreateObject("Excel.Application")
'objExcel.Workbooks.Open("C:\Extra\mpNumerics\Output\mpTemp00.html")
objExcel.Visible = True
objExcel.Workbooks.Add
objExcel.Cells(1, 1).Value = "Test value"
objExcel = Nothing
End Sub

Sub Main()
Call DemoExcel()
End Sub
End Module
```

Example for using Forms

```
Imports System.Windows.Forms

Partial Class MyForm : Inherits Form
'Component's Declaration
Friend WithEvents lblFirstName As Label = New Label
Friend WithEvents lblLastName As Label = New Label
Friend WithEvents txtFirstName As TextBox = New TextBox
Friend WithEvents txtLastName As TextBox = New TextBox
Friend WithEvents btnShow As Button = New Button

Private Sub InitializeComponent()
Me.Text = "My Second Example Form"

'lblFirstName Setting
lblFirstName.Text = "First Name : "
'Set the label into AutoSize
lblFirstName.AutoSize = True
'Set the location/position of the lblFirstName Object relative to the form
   System.Drawing.Point(x, y)
lblFirstName.Location = New System.Drawing.Point(10,10)

'lblLastName Setting
lblLastName.Text = "Last Name : "
lblLastName.AutoSize = True
lblLastName.Location = New System.Drawing.Point(10, 60)

'txtFirstName Setting
txtFirstName.MaxLength = 50
txtFirstName.Size = New System.Drawing.Size(150, 40)
txtFirstName.Location = New System.Drawing.Point(100, 10)

'txtLastName Setting
txtLastName.MaxLength = 50
txtLastName.Size = New System.Drawing.Size(150, 40)
txtLastName.Location = New System.Drawing.Point(100, 60)

'btnShow Setting
btnShow.Text = "&Show"
btnShow.Size = New System.Drawing.Size(50, 30)
btnShow.Location = New System.Drawing.Point(10, 100)

'Adding the control/component into the Form
Me.Controls.Add(lblLastName)
Me.Controls.Add(lblFirstName)
Me.Controls.Add(txtLastName)
Me.Controls.Add(txtFirstName)
Me.Controls.Add(btnShow)
Me.Size = New System.Drawing.Size(txtLastname.Right + 20, btnShow.Top + 70)
Me.StartPosition = FormStartPosition.CenterScreen
```

```
End Sub

Private Sub btnShow_Clicked(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles btnShow.Click
MessageBox.Show("Welcome " & txtFirstName.Text & " " & txtLastName.Text,"Welcome")
End Sub

Public Sub New()
InitializeComponent()
End Sub

End Class


Module Module1

Function Main(ByVal cmdArgs() As String) As Integer
Application.EnableVisualStyles()
Dim theForm As New MyForm
theForm.ShowDialog()
Return 0
End Function

End Module
```

Example for using .NET Charts

```
Imports System.Windows.Forms
Imports System.Windows.Forms.DataVisualization.Charting

Module Module1

Function Main(ByVal cmdArgs() As String) As Integer
Dim Chart1 As System.Windows.Forms.DataVisualization.Charting.Chart
Chart1 = New Chart()
Dim chartArea1 As New ChartArea()
Chart1.ChartAreas.Add("Default")
Chart1.Series.Add("Default")

' Populate series data
Dim yValues As Double() = {65.62, 75.54, 60.45, 34.73, 85.42}
Dim xValues As String() = {"France", "Canada", "Germany", "USA", "Italy"}
Chart1.Series("Default").Points.DataBindXY(xValues, yValues)

' Set Doughnut chart type
Chart1.Series("Default").ChartType = SeriesChartType.Doughnut

' Set labels style
Chart1.Series("Default")("PieLabelStyle") = "Outside"

' Set Doughnut radius percentage
Chart1.Series("Default")("DoughnutRadius") = "60"

' Explode data point with label "Italy"
Chart1.Series("Default").Points(4)("Exploded") = "true"

' Enable 3D
Chart1.ChartAreas("Default").Area3DStyle.Enable3D = false

' Set drawing style
chart1.Series("Default")("PieDrawingStyle") = "SoftEdge"


' Set Chart control size
Chart1.Size = New System.Drawing.Size(360, 260)

Dim FileName As String
FileName = cmdArgs(0)
'FileName = "I:\mpNew\mpNumerics\VBNET.emf"
'Chart1.SaveImage(FileName, ChartImageFormat.EmfDual)
Chart1.Serializer.Save(FileName)
Return 0
End Function

End Module
```

Example for using the speech synthesizer

```
Imports System.Windows.Forms
Imports System.Speech.Synthesis

Module Module1

Function Main(ByVal cmdArgs() As String) As Integer
Dim speaker as New SpeechSynthesizer()
speaker.Rate = 1
speaker.Volume = 100
speaker.Speak("Hello world".)
speaker.SetOutputToWaveFile("c:\soundfile.wav")
speaker.Speak("Hellow world.")
speaker.SetOutputToDefaultAudioDevice()
'Must remember to reset out device or the next call to speak
'will try to write to a file
End Function

End Module
```

Example for using Matlab as a COM Server from Visual Basic
This example calls a user-defined MATLAB function named solve_bvp from a Microsoft Visual Basic client application through a COM interface. It also plots a graph in a new MATLAB window and performs a simple computation:

```
Dim MatLab As Object
Dim Result As String
Dim MReal(1, 3) As Double
Dim MImag(1, 3) As Double

MatLab = CreateObject("Matlab.Application")

'Calling MATLAB function from VB
'Assuming solve_bvp exists at specified location
Result = MatLab.Execute("cd d:\matlab\work\bvp")
Result = MatLab.Execute("solve_bvp")

'Executing other MATLAB commands
Result = MatLab.Execute("surf(peaks)")
Result = MatLab.Execute("a = [1 2 3 4; 5 6 7 8]")
Result = MatLab.Execute("b = a + a ")
'Bring matrix b into VB program
MatLab.GetFullMatrix("b", "base", MReal, MImag)
```

The following examples require NetOffice to be installed.
Example for calling Excel using NetOffice

```
Imports NetOffice
'Imports Office = NetOffice.OfficeApi
Imports Excel = NetOffice.ExcelApi
'Imports NetOffice.ExcelApi.Enums

Module Program

Private Sub GetActiveExcel()
Dim xlProxy As Object =
    System.Runtime.InteropServices.Marshal.GetActiveObject("Excel.Application")
Dim xlApp As Excel._Application = New Excel._Application(Nothing, xlProxy)
Dim workBook As Excel.Workbook = xlApp.ActiveWorkbook
Dim workSheet As Excel.Worksheet = xlApp.ActiveSheet
Dim wbName As String = workBook.Name
System.Console.WriteLine(wbName)
'VERY IMPORTANT! OTHERWISE LATER CALLS WILL FAIL!
xlApp.Dispose()
End Sub

Sub Main()
GetActiveExcel()
End Sub

End Module
```

Example for calling Word using NetOffice

```
Imports NetOffice
Imports Office = NetOffice.OfficeApi
Imports Word = NetOffice.WordApi
Imports NetOffice.WordApi.Enums

Module Program

Private Sub GetActiveWord()
Dim wdProxy As Object =
    System.Runtime.InteropServices.Marshal.GetActiveObject("Word.Application")
Dim wdApp As Word._Application = New Word._Application(Nothing, wdProxy)
'VERY IMPORTANT! OTHERWISE LATER CALLS WILL FAIL!
wdApp.Dispose()
End Sub

Sub Main()
GetActiveWord()
End Sub

End Module
```

Example for calling PowerPoint using NetOffice

```
Imports NetOffice
Imports Office = NetOffice.OfficeApi
Imports PowerPoint = NetOffice.PowerPointApi
Imports NetOffice.PowerPointApi.Enums

Module Program

Private Sub GetActivePowerpoint()
Dim ppProxy As Object =
    System.Runtime.InteropServices.Marshal.GetActiveObject("Powerpoint.Application")
Dim ppApp As Powerpoint._Application= New Powerpoint._Application(Nothing, ppProxy)

'VERY IMPORTANT! OTHERWISE LATER CALLS WILL FAIL!
ppApp.Dispose()
End Sub

Sub Main()
GetActivePowerpoint()
End Sub

End Module
```

A.3.2 C# 4.0

C# C#[note 1] (pronounced see sharp) is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, procedural, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoft within its .NET initiative and later approved as a standard by Ecma (ECMA-334) and ISO (ISO/IEC 23270:2006). C# is one of the programming languages designed for the Common Language Infrastructure.

C# is intended to be a simple, modern, general-purpose, object-oriented programming language.[6] Its development team is led by Anders Hejlsberg. The most recent version is C# 5.0, which was released on August 15, 2012.
C# has the following syntax:
Semicolons are used to denote the end of a statement. Curly braces are used to group statements. Statements are commonly grouped into methods (functions), methods into classes, and classes into namespaces. Variables are assigned using an equals sign, but compared using two consecutive equals signs. Square brackets are used with arrays, both to declare them and to get a value at a given index in one of them

By design, C# is the programming language that most directly reflects the underlying Common Language Infrastructure (CLI).[30] Most of its intrinsic types correspond to value-types implemented by the CLI framework. However, the language specification does not state the code generation requirements of the compiler: that is, it does not state that a C# compiler must target a Common Language Runtime, or generate Common Intermediate Language (CIL), or generate any other specific format. Theoretically, a C# compiler could generate machine code like traditional compilers of C++ or Fortran. Some notable features of C# that distinguish it from C and C++ (and Java, where noted) are:

C# supports strongly typed implicit variable declarations with the keyword var, and implicitly typed arrays with the keyword new[] followed by a collection initializer. Meta programming via C# attributes is part of the language. Many of these attributes duplicate the functionality of GCC's and VisualC++'s platform-dependent preprocessor directives.

Like C++, and unlike Java, C# programmers must use the keyword virtual to allow methods to be overridden by subclasses. Extension methods in C# allow programmers to use static methods as if they were methods from a class's method table, allowing programmers to add methods to an object that they feel should exist on that object and its derivatives.

The type dynamic allows for run-time method binding, allowing for JavaScript like method calls and run-time object composition. C# has strongly typed and verbose function pointer support via the keyword delegate.

Like the Qt framework's pseudo-C++ signal and slot, C# has semantics specifically surrounding publish-subscribe style events, though C# uses delegates to do so. C# offers Java-like synchronized method calls, via the attribute [MethodImpl(MethodImplOptions.Synchronized)], and has support for mutually-exclusive locks via the keyword lock. The C# languages does not allow for global variables or functions. All methods and members must be declared within classes. Static members of public classes can substitute for global variables and functions.

Local variables cannot shadow variables of the enclosing block, unlike C and C++.

A C# namespace provides the same level of code isolation as a Java package or a C++ namespace,

with very similar rules and features to a package. C# supports a strict Boolean data type, bool. Statements that take conditions, such as while and if, require an expression of a type that implements the true operator, such as the boolean type. While C++ also has a boolean type, it can be freely converted to and from integers, and expressions such as if(a) require only that a is convertible to bool, allowing a to be an int, or a pointer. C# disallows this "integer meaning true or false" approach, on the grounds that forcing programmers to use expressions that return exactly bool can prevent certain types of programming mistakes common in C or C++ such as if (a = b) (use of assignment = instead of equality ==).

In C#, memory address pointers can only be used within blocks specifically marked as unsafe, and programs with unsafe code need appropriate permissions to run. Most object access is done through safe object references, which always either point to a "live" object or have the well-defined null value; it is impossible to obtain a reference to a "dead" object (one that has been garbage collected), or to a random block of memory. An unsafe pointer can point to an instance of a value-type, array, string, or a block of memory allocated on a stack. Code that is not marked as unsafe can still store and manipulate pointers through the System.IntPtr type, but it cannot dereference them. Managed memory cannot be explicitly freed; instead, it is automatically garbage collected. Garbage collection addresses the problem of memory leaks by freeing the programmer of responsibility for releasing memory that is no longer needed.

In addition to the try...catch construct to handle exceptions, C# has a try...finally construct to guarantee execution of the code in the finally block, whether an exception occurs or not.

Multiple inheritance is not supported, although a class can implement any number of interfaces. This was a design decision by the language's lead architect to avoid complication and simplify architectural requirements throughout CLI. When implementing multiple interfaces that contain a method with the same signature, C# allows the programmer to implement each method depending on which interface that method is being called through, or, like Java, allows the programmer to implement the method once and have that be the single invocation on a call through any of the class's interfaces.

C#, unlike Java, supports operator overloading. Only the most commonly overloaded operators in C++ may be overloaded in C#. C# is more type safe than C++. The only implicit conversions by default are those that are considered safe, such as widening of integers. This is enforced at compile-time, during JIT, and, in some cases, at runtime. No implicit conversions occur between booleans and integers, nor between enumeration members and integers (except for literal 0, which can be implicitly converted to any enumerated type). Any user-defined conversion must be explicitly marked as explicit or implicit, unlike C++ copy constructors and conversion operators, which are both implicit by default.

C# has explicit support for covariance and contravariance in generic types, unlike C++ which has some degree of support for contravariance simply through the semantics of return types on virtual methods.

Enumeration members are placed in their own scope. C# provides properties as syntactic sugar for a common pattern in which a pair of methods, accessor (getter) and mutator (setter) encapsulate operations on a single attribute of a class. No redundant method signatures for the getter/setter implementations need be written, and the property may be accessed using attribute syntax rather than more verbose method calls.

Checked exceptions are not present in C# (in contrast to Java). This has been a conscious

decision based on the issues of scalability and versionability. For further information on C#, see Wikipedia: C# (the text above has been copied from this reference)

Example for using the library

```
using System;
using System.Collections.Generic;
using System.Text;
using MatrixClass2;


namespace ConsoleSimple
{
class Program
{
static void Main(string[] args)
{
mp.Prec10 = 339;
mp.FloatingPointType = 3;
double x1 = 15.0;
mpNum Y1 = "5.12";
mpNum Y2 = Y1 * x1;
mpNum Y3 = mp.Sqrt(Y1);
Console.WriteLine(" x1: " + x1 + "; Y1: ");
Console.WriteLine("@" + Y1.Str());
Console.WriteLine(" Y2: " + Y2.Str() + "; Y3: " );
Console.WriteLine("@" + Y3.Str());
}
}
}
```

Example for using Excel

```
using System;

namespace DemoExcel
{
class Program
{
static void Main(string[] args)
{
dynamic xlApp = Activator.CreateInstance(Type.GetTypeFromProgID("Excel.Application"));
xlApp.Visible = true;
xlApp.Workbooks.Add();
xlApp.Cells(1, 1).Value = "Test value";
}
}
}
```

A.3.3 JScript 10.0

JScript .NET is a .NET programming language developed by Microsoft.
The primary differences between JScript and JScript .NET can be summarized as follows:
Firstly, JScript is a scripting language, and as such programs (or more suggestively, scripts) can be executed without the need to compile the code first. This is not the case with the JScript .NET command-line compiler, since this next-generation version relies on the .NET Common Language Runtime (CLR) for execution, which requires that the code be compiled to Common Intermediate Language (CIL), formerly called Microsoft Intermediate Language (MSIL), code before it can be run. Nevertheless, JScript .NET still provides full support for interpreting code at runtime (e.g., via the Function constructor or the eval function) and indeed the interpreter can be exposed by custom applications hosting the JScript .NET engine via the VSA[jargon] interfaces.
Secondly, JScript has a strong foundation in Microsoft's ActiveX/COM technologies, and relies primarily on ActiveX components to provide much of its functionality (including database access via ADO, file handling, etc.), whereas JScript .NET uses the .NET Framework to provide equivalent functionality. For backwards-compatibility (or for where no .NET equivalent library exists), JScript .NET still provides full access to ActiveX objects via .NET / COM interop using both the ActiveXObject constructor and the standard methods of the .NET Type class.
Although the .NET Framework and .NET languages such as C# and Visual Basic .NET have seen widespread adoption, JScript .NET has never received much attention, by the media or by developers. It is not supported in Microsoft's premier development tool, Visual Studio .NET. However, ASP.NET supports JScript .NET.
For further details, see Wikipedia: JScript.NET (the text above has been copied from this reference).
Example for using the library:

```
//Load the mpNumerics library
import MatrixClass2;

//Set Floating point type to MPFR with 60 decimal digits precision
mp.FloatingPointType = 3;
mp.Prec10 = 60;

//Assign values to x1 and x2
var x1 = mp.CNum("32.47");
var x2 = mp.CNum("12.41");

//Calculate x3 = x1 / x2
var x3 = x1 / x2;

//Print the value of x3
print("Result: ", x3.Str());
```

Example for using Excel:

```
// Declare the variables
var Excel, Book;

// Create the Excel application object.
Excel = new ActiveXObject("Excel.Application");
```

```
// Make Excel visible.
Excel.Visible = true;

// Create a new work book.
Book = Excel.Workbooks.Add()

// Place some text in the first cell of the sheet.
Book.ActiveSheet.Cells(1,1).Value = "This is column A, row 1";

// Save the sheet.
Book.SaveAs("C:\\TEST.XLS");

// Close Excel with the Quit method on the Application object.
Excel.Application.Quit();
```

A.3.4 C++ 10.0, Visual Studio

C++/CLI (Common Language Infrastructure) is a language specification created by Microsoft and intended to supersede Managed Extensions for C++. It is a complete revision that aims to simplify the older Managed C++ syntax, which is now deprecated.[1] C++/CLI was standardized by Ecma as ECMA-372. It is currently available in Visual Studio 2005, 2008, 2010 and 2012, including the Express editions.
Syntax changes[edit]C++/CLI should be thought of as a language of its own (with a new set of keywords, for example), instead of the C++ superset-oriented Managed C++. Because of this, there are some major syntactic changes, especially related to the elimination of ambiguous identifiers and the addition of .NET-specific features.
Many conflicting syntaxes, such as the multiple versions of operator new() in MC++ have been split: in C++/CLI, .NET reference types are created with the new keyword gcnew. Also, C++/CLI has introduced the concept of generics (conceptually similar to standard C++ templates, but quite different in their implementation).
In C++/CLI the only type of pointer is the normal C++ pointer, and the .NET reference types are accessed through a handle, with the new syntax ClassNameînstead of ClassName. This new construct is especially helpful when managed and standard C++ code is mixed; it clarifies which objects are under .NET automatic garbage collection and which objects the programmer must remember to explicitly destroy.
Operator overloading works analogously to standard C++. Every * becomes a ^, every & becomes an %, but the rest of the syntax is unchanged, except for an important addition: Operator overloading is possible not only for classes themselves, but also for references to those classes. This feature is necessary to give a ref class the semantics for operator overloading expected from .NET ref classes. In reverse, this also means that for .Net framework ref classes, reference operator overloading often is implicitly implemented in C++/CLI.
For further information, see Wikipedia: C++/CLI (the text above has been copied from this reference).
Example for using the library

```
// compile with: /clr
using namespace System;
using namespace MatrixClass2;


int main()
{
mp^ MP = gcnew mp;
MP->Prec10 = 30;
MP->FloatingPointType = 3;
mpNum^ x1 = gcnew mpNum;
x1 = "3.4";
mpNum^ x2 = gcnew mpNum;
x2 = "13.4";
mpNum^ x3 = gcnew mpNum;
x3 = x1 / x2;
String^ Result = x1->Str();
Console::WriteLine("Result: {0} ", Result);
return 0;
}
```

Example for mixing managed and unmanaged code

```
// compile with: /clr
using namespace System;
using namespace MatrixClass2;


int main()
{
// pragma_directives_managed_unmanaged.cpp
// compile with: /clr
#include <stdio.h>
#include <iostream>

// func1 is managed
void func1() {
System::Console::WriteLine("In managed C++ function (func1).");
}

// #pragma unmanaged
// push managed state on to stack and set unmanaged state
#pragma managed(push, off)

// func2 is unmanaged
void func2() {
printf("In unmanaged C function (func2).\n");
}

// func3 is unmanaged
void func3() {
std::cout << "In unmanaged C++ function (func3)." << std::endl;
}


// #pragma managed
#pragma managed(pop)

// main is managed
int main() {
func1();
func2();
func3();
}
```

A.3.5 F# 3.0

F# (pronounced F Sharp) is a strongly typed, multi-paradigm programming language encompassing functional, imperative and object-oriented programming techniques. F# is most often used as a cross-platform CLI language, but can also be used to generate JavaScript[3] and GPU[4] code.
F# is developed by the F# Software Foundation,[5] Microsoft and open contributors. An open source, cross-platform compiler for F# is available from the F# Software Foundation.[6] F# is also a fully supported language in Visual Studio.[7] Other tools supporting F# development[clarification needed] include Mono, MonoDevelop, SharpDevelop and WebSharper
F# originated as a variant of ML and has been influenced by OCaml, C#, Python, Haskell,[2] Scala and Erlang.
For further information, see the F# Homepage or Wikipedia: F# (the text above has been copied from this reference).

Example for using the library

```
open System.Windows.Forms
open MatrixClass2

// Create a window and set a few properties
let form = new Form(Visible=true, TopMost=true, Text="Welcome to F#")

// mp.FloatingPointType = 3 // Does not work, need mp.SetFloatingPointType(3)

let x1 = mp.CNum("32.47")
let x2 = mp.CNum("32.47")
let x3 = x1 + x2
let s = x3.Str()

let label = new Label(Text = s)

// Add the label to the form
form.Controls.Add(label)

// Finally, run the form
[<System.STAThread>]
Application.Run(form)
```

Example for using functions

```
/// Iteration using a 'for' loop
let printList lst =
for x in lst do
printfn "%d" x

/// Iteration using a higher-order function
let printList2 lst =
List.iter (printfn "%d") lst

/// Iteration using a recursive function and pattern matching
let rec printList3 lst =
```

```
match lst with
| [] -> ()
| h :: t ->
printfn "%d" h
printList3 t
```

A.3.6 IronPython 2.7

IronPython is an implementation of the Python programming language targeting the .NET Framework and Mono. Jim Hugunin created the project and actively contributed to it up until Version 1.0 which was released on September 5, 2006.[2] Thereafter, it was maintained by a small team at Microsoft until the 2.7 Beta 1 release; Microsoft abandoned IronPython (and its sister project IronRuby) in late 2010, after which Hugunin left to work at Google.[3] IronPython 2.0 was released on December 10, 2008.[4] The project is currently maintained by a group of volunteers at Microsoft's CodePlex open-source repository. It is free and open-source software, and can be implemented with Python Tools for Visual Studio, which is a free and open-source extension for free, isolated, and commercial versions of Microsoft's Visual Studio IDE.[5] [6]
IronPython is written entirely in C#, although some of its code is automatically generated by a code generator written in Python.
IronPython is implemented on top of the Dynamic Language Runtime (DLR), a library running on top of the Common Language Infrastructure that provides dynamic typing and dynamic method dispatch, among other things, for dynamic languages.[7] The DLR is part of the .NET Framework 4.0 and is also a part of trunk builds of Mono. The DLR can also be used as a library on older CLI implementations.
For further information, see Wikipedia: IronPython (the text above has been copied from this reference).
The original distribution of SharpDevelop includes IronPython (it is not included in the mpFormula IDE). Ironpython can be downloaded from the IronPython Homepage. Visual Studio integration is available through Python Tools for Visual Studio.

```
#Load CLR
import clr

#Load the mpNumerics library
clr.AddReference('MatrixClass2')
from MatrixClass2 import mp, mpNum

#Set Floating point type to MPFR with 60 decimal digits precision
mp.FloatingPointType = 3;
mp.Prec10 = 60;

#Assign values to x1 and x2
x1 = mp.CNum("32.47");
x2 = mp.CNum("12.41");

#Calculate x3 = x1 / x2
x3 = x1 / x2;

#Print the value of x3
print "Result: ", x3.Str();
```

A.3.7 ILNumerics

ILNumerics is a mathematical class library for Common Language Infrastructure (CLI) developers. It simplifies the implementation of an array of numerical algorithms. ILNumerics was designed to help developers create distribution-ready applications. Interfaces of existing algebra systems were often found to be less effective, when it comes to distribution/integration into existing projects; therefore, ILNumerics does not come with an interpreter but directly utilizes features of modern development environments and programming languages like C#.

N-dimensional arrays, complex numbers, linear algebra, FFT and plotting controls (2D and 3D) help developing algorithms on every platform the CLI runs on. Developers formulate computational algorithms directly in their favorite CLI language - avoiding the need for interfacing 3rd party mathematical frameworks. The syntax is vastly compatible to well known and established mathematical programs like MATLAB and GNU Octave. Due to its strong type safety algorithms developed that way are more stable and robust at run time. The library is the only math library so far, which takes the characteristics of the CLI into account and therefore achievers better execution performance than its competitors

Since ILNumerics comes as a CLI assembly, it targets Common Language Infrastructure (CLI) applications. Just like Java - those frameworks are often criticized for not being suitable for numerical computations. Reasons are the memory management by a garbage collector and the intermediate language execution. Nevertheless, due to efficient memory management (pooling), the performance of ILNumerics algorithms beat the speed of many competing frameworks by factors.[2] Linear algebra routines rely on processor specific optimized versions of LAPACK and BLAS, which further increases performance and reliability of computational results. All internal functions are parallelized. The efficiency does even allow the use for 'numbercrunching' applications, which would otherwise only be suitable for Fortran - yet providing much higher implementational convenience.
The software can be downloaded from the ILNumerics Homepage, and is available in a Professional Edition and a Community Edition.
The current Community Edition expects that the Visual C++ Redistributable for Visual Studio 2012 is installed. If this is not the case, it can be downloaded from Microsoft: Visual C++ Redistributable for Visual Studio 2012.
For further information, see Wikipedia: ILNumerics (the text above has been copied from this reference).

Example for using the library

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using ILNumerics;

namespace ConsoleApplication1 {

// it is recommended to derive from ILMath
class Program : ILNumerics.ILMath {

static void Main(string[] args) {
// create a matrix A, give values explicitely
```

```
ILArray<double> A = array<double>(
new double[]{1,1,1,1,1,2,3,4,1,3,6,10,1,4,10,20},4,4);
// use a creation function for B
ILArray<double> B = counter(4,2);

// use a function of the base class: ILMath.linsolve
ILArray<double> Result = linsolve(A,B);

// A.ToString() gives formated output
Console.Out.WriteLine("A: " + Environment.NewLine + A.ToString());
Console.Out.WriteLine("B: " + Environment.NewLine + B.ToString());
Console.Out.WriteLine("A * [Result] = B: " + Environment.NewLine
+ Result.ToString());

// check result:
// uses norm, multiply, eps and binary operators
if (norm(multiply(A, Result) - B) <= eps) {
Console.Out.WriteLine("Result ok");
} else {
Console.Out.WriteLine("Result false");
}
Console.ReadKey();
}
}
}
```

This section describes the 2d and 3d visualization capabilities of ILNumerics. The rendering engine has been redesigned to better adapt the development of interactive technical applications and scientific visualizations with C# and .NET. It now provides many features of modern game engines, abstracting away platform specifics and the low level details of computer graphics.

```
// add a new plot cube
var scene = new ILScene {
new ILPlotCube(twoDMode: false) {
// add a surface
new ILSurface(ILSpecialData.sincf(40, 60, 2.5f)) {
// make thin transparent wireframes
Wireframe = { Color = Color.FromArgb(50, Color.LightGray) },
// choose a different colormap
Colormap = Colormaps.Summer,
}
}
};
// rotate the plot cube
scene.First<ILPlotCube>().Rotation = Matrix4.Rotation(
new Vector3(1f,0.23f,1), 0.7f);
scene;
```

A.3.8 MatLab (.NET interface)

MATLAB (matrix laboratory) is a numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran.

Although MATLAB is intended primarily for numerical computing, an optional toolbox uses the MuPAD symbolic engine, allowing access to symbolic computing capabilities. An additional package, Simulink, adds graphical multi-domain simulation and Model-Based Design for dynamic and embedded systems.

The MATLAB application is built around the MATLAB language, and most use of MATLAB involves typing MATLAB code into the Command Window (as an interactive mathematical shell), or executing text files containing MATLAB codes, including scripts and/or functions.[6]

Variables are defined using the assignment operator, =. MATLAB is a weakly typed programming language because types are implicitly converted.[7] It is a dynamically typed language because variables can be assigned without declaring their type, except if they are to be treated as symbolic objects,[8] and that their type can change. Values can come from constants, from computation involving values of other variables, or from the output of a function.

As suggested by its name (a contraction of "Matrix Laboratory"), MATLAB can create and manipulate arrays of 1 (vectors), 2 (matrices), or more dimensions. In the MATLAB vernacular, a vector refers to a one dimensional (1ÃŮN or NÃŮ1) matrix, commonly referred to as an array in other programming languages. A matrix generally refers to a 2-dimensional array, i.e. an mÃŮn array where m and n are greater than 1. Arrays with more than two dimensions are referred to as multidimensional arrays. Arrays are a fundamental type and many standard functions natively support array operations allowing work on arrays without explicit loops.

MATLAB can call functions and subroutines written in the C programming language or Fortran. A wrapper function is created allowing MATLAB data types to be passed and returned. The dynamically loadable object files created by compiling such functions are termed "MEX-files" (for MATLAB executable).[13][14]

Libraries written in Java, ActiveX or .NET can be directly called from MATLAB and many MATLAB libraries (for example XML or SQL support) are implemented as wrappers around Java or ActiveX libraries. Calling MATLAB from Java is more complicated, but can be done with a MATLAB extension,[15] which is sold separately by MathWorks, or using an undocumented mechanism called JMI (Java-to-MATLAB Interface),[16] which should not be confused with the unrelated Java Metadata Interface that is also called JMI.

As alternatives to the MuPAD based Symbolic Math Toolbox available from MathWorks, MATLAB can be connected to Maple or Mathematica.[17]

Libraries also exist to import and export MathML. MATLAB has a COM interface which is described in section A.2.11.

For further information on MatLab, see Wikipedia: MatLab (the text above has been copied from this reference), or the MatLab Homepage. See also MatLab External Interfaces.

Example for using the library

```
x = 4.3;
fprintf('Start: x is equal to %6.2f.\n',x);

%Load mpFormula .NET assembly
NET.addAssembly('MatrixClass2');

%Instantiate local reference to mpFormula
mp = MatrixClass2.mp;

%Set Floating point type to MPFR with 60 decimal digits precision
NET.setStaticProperty('MatrixClass2.mp.FloatingPointType', 3);
NET.setStaticProperty('MatrixClass2.mp.Prec10', 50);
Myprec = mp.Prec10;

fprintf('End: Myprec is equal to %6.2f.\n',Myprec);

%Assign values to x1 and x2
x1 = mp.CNum(4.5);
x2 = mp.CNum('1.1');
x3 = x1 / x2;
x4 = MatrixClass2.mpNum;

x4 = mp.CNum(4.5356346345);
s = x3.Str();
s2 = char(s);
fprintf('s is equal to %s.\n',s2);
fprintf('End: x is equal to %6.2f.\n',x);
quit;
```

A.4 Java (via jni4net)

Java is a general-purpose, concurrent, class-based, object-oriented computer programming language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. Java applications are typically compiled to bytecode (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture. Java is, as of 2012, one of the most popular programming languages in use, particularly for client-server web applications, with a reported 10 million users.[10][11] Java was originally developed by James Gosling at Sun Microsystems (which has since merged into Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

The original and reference implementation Java compilers, virtual machines, and class libraries were developed by Sun from 1991 and first released in 1995. As of May 2007, in compliance with the specifications of the Java Community Process, Sun relicensed most of its Java technologies under the GNU General Public License. Others have also developed alternative implementations of these Sun technologies, such as the GNU Compiler for Java and GNU Classpath.

The syntax of Java is largely derived from C++. Unlike C++, which combines the syntax for structured, generic, and object-oriented programming, Java was built almost exclusively as an object-oriented language. All code is written inside a class, and everything is an object, with the exception of the primitive data types (e.g. integers, floating-point numbers, boolean values, and characters), which are not classes for performance reasons. Unlike C++, Java does not support operator overloading or multiple inheritance for classes.

Oracle Corporation is the current owner of the official implementation of the Java SE platform, following their acquisition of Sun Microsystems on January 27, 2010. This implementation is based on the original implementation of Java by Sun. The Oracle implementation is available for Mac OS X, Windows and Solaris. Because Java lacks any formal standardization recognized by Ecma International, ISO/IEC, ANSI, or other third-party standards organization, the Oracle implementation is the de facto standard.

The Oracle implementation is packaged into two different distributions: The Java Runtime Environment (JRE) which contains the parts of the Java SE platform required to run Java programs and is intended for end-users, and the Java Development Kit (JDK), which is intended for software developers and includes development tools such as the Java compiler, Javadoc, Jar, and a debugger. OpenJDK is another notable Java SE implementation that is licensed under the GPL. The implementation started when Sun began releasing the Java source code under the GPL. As of Java SE 7, OpenJDK is the official Java reference implementation.

Programs written in Java have a reputation for being slower and requiring more memory than those written in C++. However, Java programs' execution speed improved significantly with the introduction of Just-in-time compilation in 1997/1998 for Java 1.1, the addition of language features supporting better code analysis (such as inner classes, the StringBuffer class, optional assertions, etc.), and optimizations in the Java virtual machine itself, such as HotSpot becoming the default for Sun's JVM in 2000. As of December 2012, microbenchmarks show Java 7 is approximately 44% slower than C++.

For further information on Java, see Wikipedia: Java (the text above has been copied from this

reference), or the Java Homepage.

The Java SDK can be downloaded from Java SDK Download Homepage. Currently versions 1.7 and 1.8 are supported.
Java programs can access the functions provided by the mpFormula Library using the `jni4net` bridge between Java and .NET (see the jni4net Homepage for more information).
For this reason, a Java program using the mpFormula Library needs to include a number of import statements for the mpLib and the mpNum types. It is also necessary to intialize the bridge between Java and .NET and to load and register the relevant assembly, as shown in the program below.

The supporting .dll and .jar files are located in
`..\mpFormula40\Toolbox\mpFormula4java\Win32\work` for 32 bit, and
`..\mpFormula40\Toolbox\mpFormula4java\Win64\work` for 64 bit.

A copy of the example program below can be found in
`..\mpFormula40\Toolbox\mpFormula4java\Win32` for 32 bit and
`..\mpFormula40\Toolbox\mpFormula4java\Win64` for 64 bit.

The batch files in these directories assume that the Java SDK is installed and that the directory containing `javac.exe` and `java.exe` is in the path. It is also assumed that on Windows 64 bit only the binaries of the 64 bit (but not of the 32 bit) version the Java SDK are in the path. To compile and/or run a 32 bit Java program on Windows 64 bit, use the batch files `CompileAndRun32bitOnW64.cmd` and `RunOnly32bitOnW64.cmd`, modifying the absolute paths in these batch files as needed.

Example for using the library

```
import java.io.IOException;

//Import Java2Net Bridge and mpFormula classes
import net.sf.jni4net.Bridge;
import mpformula4java.mpLib;
import mpformula4java.mpLibT;
import mpformula4java.mpNum;
import mpformula4java.mpNumT;

public class MyCalcUsageInJava {
  public static void main(String arsg[]) throws IOException {

    //Initialize Java2Net Bridge and load relevant assemblies
    System.out.printf("Opening Java2Net Bridge ... \n");
    Bridge.init();
    Bridge.LoadAndRegisterAssemblyFrom(new java.io.File("mpFormula4java.j4n.dll"));

    //Initialize the mpNumerics library
    mpLib mp = new mpLibT();

    //Set Floating point type to MPFR with 60 decimal digits precision
    mp.SetFloatingPointType(3);
    mp.SetPrec10(50);

    //Assign values to x1 and x2
```

```
    mpNum x1 = mp.Num("12.0");
    System.out.printf("Value of x1 is : " + x1.Str() + "\n");

    mpNum x2 = mp.Sqrt(x1);
    System.out.printf("Value of x2 is : " + x2.Str() + "\n");

    mpNum x3 = x1.Plus(x2);
    System.out.printf("Value of x1 + x2 is : " + x3.Str() + "\n");

    x3 = x1.Minus(x2);
    System.out.printf("Value of x1 - x2 is : " + x3.Str() + "\n");

    x3 = x1.Times(x2);
    System.out.printf("Value of x1 * x2 is : " + x3.Str() + "\n");

    //Calculate x3 = x1 / x2
    x3 = x1.Div(x2);

    //Print the value of x3
    System.out.printf("Value of x1 / x2 is : " + x3.Str() + "\n");

    System.out.printf("Closing Java2Net Bridge ... \n");

  }
}
```

A.4.0.1 Downloading and installing the Java SDK

The Java SDK can be downloaded from the Java SDK Download Homepage. Currently releases 1.7 and 1.8 are available for download. The prebuild binaries of mpFormula use the 1.7 release. To support both 32 bit and 64 bit builds, you need to install the 32 and 64 bit editions of the SDK separately.

The batch files used for building Java support assume that the Java SDK is installed and that the directory containing `javac.exe` and `java.exe` is in the path. It is also assumed that you build on Windows 64 bit and that only the binaries of the 64 bit (but not of the 32 bit) version the Java SDK are in the path.

A.4.0.2 Downloading and installing jni4net

jni4net: jni4net.

Copyright (c) Pavel Ŝavara.

The license is the GNU Lesser General Public License, Version 3 (see appendix D.1.3)

The required binary files are contained in the file `jni4net-0.8.6.0-bin.zip`, which can be downloaded from

http://sourceforge.net/projects/jni4net/files/.

Unzip this file and copy the content of the resulting folder to `\mpFormula40\Java2Net`.

A.4.0.3 Applying jni4net to build the Java interface

The source code which is specific to providing Java support for mpFormula is automatically generated when building the documentation and is contained in the file `\Toolbox\mpFormula4java\Source\CSharp\Calc.cs`.

In the folder `\Toolbox\mpFormula4java\Source\CSharp`, open the solution `mpFormula4java.sln`, and compile the assembly `mpFormula4java.dll` for 32 and 64 bit (Release).

From `\Toolbox\mpFormula4java\Source`, copy the subfolder `\MakeJavaBridge` into the folder `\mpFormula40\Java2Net`.

Within the folder `\mpFormula40\Java2Net\MakeJavaBridge` run the command file `Make.cmd`. This will do the following:
- generateProxies will copy all dependencies from jni4net lib to work directory
- generateProxies will run the proxygen tool to wrap `mpFormula4java.dll`
– proxygen: generate java proxies
– proxygen: generate C# proxies
– proxygen: generate build.cmd
- generateProxies will run `work\build2.cmd` to compile the output generated by proxygen.
– build2.cmd : run javac to compile java classes
– build2.cmd : run jar to package classes into .JAR file
– build2.cmd : run csc to compile C# classes and produce .DLL

Once this has been completed without error messages, run the command file `Make Install.cmd`.
This will install the appropriate .cmd, .jar and .dll files into `\Toolbox\mpFormula4java\Win32` for 32 bit and `\Toolbox\mpFormula4java\Win64`, for 64 bit.

A.4.0.4 Testing the Java-.NET Bridge

A copy of the example program below can be found in `..\mpFormula40\Toolbox\mpFormula4java\Win32` for 32 bit and `..\mpFormula40\Toolbox\mpFormula4java\Win64` for 64 bit.

The batch files in these directories assume that the Java SDK is installed and that the directory containing `javac.exe` and `java.exe` is in the path. It is also assumed that on Windows 64 bit only the binaries of the 64 bit (but not of the 32 bit) version the Java SDK are in the path. To compile and/or run a 32 bit Java program on Windows 64 bit, use the batch files `CompileAndRun32bitOnW64.cmd` and `RunOnly32bitOnW64.cmd`, modifying the absolute paths in these batch files as needed.

To confirm that the installation was succesful, run the command files `CompileAndRun32bit.cmd` and `CompileAndRun64bit.cmd` in their respective folders.

A.5 SQLite and System.Data.SQLite

SQLite is a relational database management system contained in a small (350 KB) C programming library. In contrast to other database management systems, SQLite is not a separate process that is accessed from the client application, but an integral part of it.

SQLite is ACID-compliant and implements most of the SQL standard, using a dynamically and weakly typed SQL syntax that does not guarantee the domain integrity.

SQLite is a popular choice as embedded database for local/client storage in application software such as web browsers. It is arguably the most widely deployed database engine, as it is used today by several widespread browsers, operating systems, and embedded systems, among others. SQLite has many bindings to programming languages.

The source code for SQLite is in the public domain.

SQLite implements most of the SQL-92 standard for SQL but it lacks some features. For example it has partial support for triggers, and it can't write to views (however it supports INSTEAD OF triggers that provide this functionality). While it supports complex queries, it still has limited ALTER TABLE support, as it can't modify or delete columns.

SQLite uses an unusual type system for an SQL-compatible DBMS. Instead of assigning a type to a column as in most SQL database systems, types are assigned to individual values; in language terms it is dynamically typed. Moreover, it is weakly typed in some of the same ways that Perl is: one can insert a string into an integer column (although SQLite will try to convert the string to an integer first, if the column's preferred type is integer). This adds flexibility to columns, especially when bound to a dynamically typed scripting language. However, the technique is not portable to other SQL products. A common criticism is that SQLite's type system lacks the data integrity mechanism provided by statically typed columns in other products. The SQLite web site describes a "strict affinity" mode, but this feature has not yet been added.[12] However, it can be implemented with constraints like CHECK(typeof(x)='integer').

Several computer processes or threads may access the same database concurrently. Several read accesses can be satisfied in parallel. A write access can only be satisfied if no other accesses are currently being serviced. Otherwise, the write access fails with an error code (or can automatically be retried until a configurable timeout expires). This concurrent access situation would change when dealing with temporary tables. This restriction is relaxed in version 3.7 when WAL is turned on enabling concurrent reads and writes.

A standalone program called sqlite3 is provided that can be used to create a database, define tables within it, insert and change rows, run queries and manage an SQLite database file. This program is a single executable file on the host machine. It also serves as an example for writing applications that use the SQLite library.

SQLite is a popular choice for local/client SQL storage within a web browser and within a rich internet application framework;[14] most notably the leaders in this area (Google Gears,[15] Adobe AIR,[16] and Firefox[17]) embed SQLite.

SQLite full Unicode support is optional.

SQLite also has bindings for a large number of programming languages. An ADO.NET adapter, initially developed by Robert Simpson, is maintained jointly with the SQLite developers since April 2010.[23] An ODBC driver has been developed and is maintained separately by Christian

Werner.[24] Werner's ODBC driver is the recommend connection method for accessing SQLite from OpenOffice.[25] There is also a COM (ActiveX) wrapper making SQLite accessible on Windows to scripted languages such as JScript and VBScript. This adds database capabilities to HTML Applications (HTA).

For further information on SQLite, see Wikipedia: SQLite (the text above has been copied from this reference), or the SQLite Homepage, or the System.Data.SQLite Homepage.

A good introduction to SQLIte is Kreibich (2010)

The SQLite OBDC driver is available from http://www.ch-werner.de/sqliteodbc/. This driver can be used to access SQLite files from MSOffice, LibreOffice, and the .NET programming languages.

A.5.1 SQLite Graphical User Interfaces

SQLite itself does not provide a Graphical User Interface.
There are however a number of programs available, such as
SQLite2009 Pro Enterprise Manager: SQLite2009 Pro Enterprise Manager, or
SQLIte Studio: SQLIte Studio.

A.5.2 Testing the SQLite Interface to mpFromula

To test the interface, start the application `sqlite3.exe` in the folder `\mpFormula40\Toolbox\SQLiteExtensions`

At the command prompt, type

```
.load sql_trig.sqlite3ext sql_trig_init
SELECT cosd(60);
```

The result should be `0.5`.
Alternatively, at the command prompt, type

```
 select load_extension('sql_trig.sqlite3ext.dll')
SELECT cosd(60);
```

The result should be `0.5`.

A.5.3 Testing the System.Data.SQLite Interface to mpFromula

To test the System.Data.SQLite interface, start the CSharp solution `SQLiteDemo.sln` in the folder `\mpFormula40\Toolbox\SQLiteDemo32`

In the source code, the section

```
public static SQLiteConnection GetConnection()
{
  string MyFName = RootDir32() + @"Workbooks\Scripts.db";
  SQLiteConnection connection = new SQLiteConnection("Data Source=" + MyFName +
      ";Version=3;", true);
  try
  {
    connection.Open();
```

```
        String ExtDir = RootDir32() + @"SQLiteExtensions\sql_trig.sqlite3ext";
        connection.LoadExtension(ExtDir, "sql_trig_init");
      }
      catch
      {
        throw;
      }
      return connection;
    }
```

illustrates how the extension library is loaded. Run the program to see some additional sample output.

```
-- Test1.db
SELECT * FROM Cardio WHERE RR > 250;
SELECT * FROM Cardio WHERE (RR - 25000.0) / 2 > 2500.0;
SELECT * FROM Cardio WHERE (RR > 25000);
SELECT Cardio.RR, Cardio.HR FROM Cardio WHERE RR > 30000;

SELECT 1873.0 / 45.0 ;
SELECT Total(RR) FROM Cardio;
```

To create a table. use

```
-- Test1.db
CREATE TABLE "Code_VBNET" (
"Index1" INTEGER PRIMARY KEY NOT NULL ,
"Category" INTEGER NOT NULL ,
"Description" TEXT NOT NULL ,
"CategoryName" TEXT NOT NULL ,
"Code" TEXT
);
```

SQLite: SQLite Homepage.
System.Data.SQLite: System.Data.SQLite Homepage.
These libraries are in the public domain.

A.5.3.1 Downloading the SQLite source code

The SQLite source code can be downloaded from
http://www.sqlite.org/download.html.
You need to download the file

`sqlite-amalgamation-3080500.zip` (1.44 MiB) :

This ZIP archive contains all C source code for SQLite 3.8.5 combined into a single source file (the amalgamation).

Unzip this file and from the resulting folder copy all files into the folder
`\mpFormula40\Toolbox\SQLiteExtensions`.

A.5.3.2 Downloading and installing System.Data.SQLite

The following precompiled binaries of System.Data.SQLite can be downloaded from http://system.data.sqlite.org/index.html/doc/trunk/www/downloads.wiki.

Precompiled Binaries for 32-bit Windows (.NET Framework 4.0):

`sqlite-netFx40-binary-bundle-Win32-2010-1.0.93.0.zip` (2.09 MiB):

This binary package features the mixed-mode assembly and contains all the binaries for the x86 version of the System.Data.SQLite 1.0.93.0 (3.8.5) package. The Visual C++ 2010 SP1 runtime for x86 and the .NET Framework 4.0 are required.

Unzip this file and from the resulting folder copy the file `System.Data.SQLite.dll` into the folder `\mpFormula40\Toolbox\mpNum\Win32\Bin`.

Precompiled Binaries for 64-bit Windows (.NET Framework 4.0):

`sqlite-netFx40-binary-bundle-x64-2010-1.0.93.0.zip` (2.11 MiB) :

This binary package features the mixed-mode assembly and contains all the binaries for the x64 version of the System.Data.SQLite 1.0.93.0 (3.8.5) package. The Visual C++ 2010 SP1 runtime for x64 and the .NET Framework 4.0 are required.

Unzip this file and from the resulting folder copy the file `System.Data.SQLite.dll` into the folder `\mpFormula40\Toolbox\mpNum\Win64\Bin`.

A.5.3.3 Compiling the SQLite Interface

The mpFormula SQLite interface to SQLite is provided in form of a SQLite Extension DLL. The source code is contained in the file `\mpFormula40\Toolbox\SQLiteExtensions\sql_trig.c`, which is automatically generated when the documentation is build.
You need to have Microsoft Visual C++ 2010 installed to compile the SQLite Interface. Run the batch file `compile.bat` to build the 32 bit version, and the file `compile64.bat` to build the 64 bit version.

A.5.3.4 Testing the SQLite Interface

To test the interface, start the application `sqlite3.exe` in the folder `\mpFormula40\Toolbox\SQLiteExtensions`

At the command prompt, type

```
.load sql_trig.sqlite3ext sql_trig_init
SELECT cosd(60);
```

The result should be `0.5`.

A.5.3.5 Testing the System.Data.SQLite Interface

To test the System.Data.SQLite interface, start the CSharp solution `SQLiteDemo.sln` in the folder `\mpFormula40\Toolbox\SQLiteDemo32`

In the source code, the section

```
public static SQLiteConnection GetConnection()
{
  string MyFName = RootDir32() + @"Workbooks\Scripts.db";
  SQLiteConnection connection = new SQLiteConnection("Data Source=" + MyFName +
      ";Version=3;", true);
  try
  {
    connection.Open();
    String ExtDir = RootDir32() + @"SQLiteExtensions\sql_trig.sqlite3ext";
    connection.LoadExtension(ExtDir, "sql_trig_init");
  }
  catch
  {
    throw;
  }
  return connection;
}
```

illustrates how the extension library is loaded. Run the program to see some additional sample output.

A.6 gnuplot

gnuplot is a command-line program that can generate two- and three-dimensional plots of functions, data, and data fits. It is frequently used for publication-quality graphics as well as education. The program runs on all major computers and operating systems (GNU/Linux, Unix, Microsoft Windows, Mac OS X, and others). It is a program with a fairly long history, dating back to 1986. Despite its name, this software is not distributed under the GNU General Public License (GPL), but its own more restrictive open source license.[1]

gnuplot can produce output directly on screen, or in many formats of graphics files, including Portable Network Graphics (PNG), Encapsulated PostScript (EPS), Scalable Vector Graphics (SVG), JPEG and many others. It is also capable of producing LaTeX code that can be included directly in LaTeX documents, making use of LaTeX's fonts and powerful formula notation abilities. The program can be used both interactively and in batch mode using scripts.

The program is well supported and documented. Extensive help can also be found on the Internet.[2][3]

gnuplot is used as the plotting engine of Maxima, and gretl, and it can be used from various languages, including Perl (via CPAN), Python (via Gnuplot-py and SAGE), Java (via jgnuplot), Ruby (via Ruby Gnuplot), Ch (via Ch Gnuplot), and Smalltalk (Squeak and GNU Smalltalk). gnuplot also supports piping.[4]

For further information on Gnuplot, see Wikipedia: Gnuplot (the text above has been copied from this reference), or the gnuplot Homepage.

An introductory text for gnuplot is Janert (2010)

For examples see
http://soukoreff.com/gnuplot/,
http://ayapin-film.sakura.ne.jp/Gnuplot/pm3d.html,
http://gnuplot-tricks.blogspot.co.uk/,
http://www.phyast.pitt.edu/ zov1/gnuplot/html/intro.html

A connection between gnuplot and the mpFormula library can be made using the import statement, which makes the mpFormula functions available in double precision.

```
unset title
set label 1 "Kuen's Surface" at screen 0.57, 0.9
    #set label 1 font "frscript,20"
set style line 3 linetype -1 linewidth 0.5
set pm3d depthorder hidden3d 3
set style fill transparent solid 0.65 border
set palette
set hidden3d

set ticslevel 0
unset xtics ; unset ytics ; unset ztics
unset border ; unset colorbox ; unset key
set lmargin at screen 0.1
set bmargin at screen 0.1
```

```
set rmargin at screen 0.9
set tmargin at screen 0.9

set parametric
set dummy u,v
set urange [-4.5:4.5]
set vrange [0.05:pi-0.05]
set isosamples 51,51
set view 122, 357, 1.35, 1.08

a = 1.0

x(u,v) = 2.*a*(cos(u)+u*sin(u))*sin(v) / (1+u**2.*(sin(v))**2)
y(u,v) = 2.*a*(sin(u)-u*cos(u))*sin(v) / (1+u**2.*(sin(v))**2)
z(u,v) = a*log(tan(v/2.))+2.*cos(v)/(1+u**2.*(sin(v))**2)

splot x(u,v), y(u,v), z(u,v) with pm3d
```

Appendix B

Building the library

Building the toolbox and the library from scratch is a much more involved process than just using them.
Conceptually, it could be described as a top-down process which starts at the level of the modification of the source files for the documentation, the following (automated) generation of various *.xml, *.cs, *.h files and their processing with appropriate tools, which create the .NET, COM, native DLL and spreadsheet interfaces, ultimately leading to the connecting point of the mpNumC.h header file.
It could also be described as a bottom-up process, starting with the compilation of the *.c, *.h amd *.asm of the GMP, MPFR and FLINT library. followed by the compilation of the Eigen and Boost template libraries with the various supported data types, again leading to the connecting point of the mpNumC.h header file.
In practice, it is easiest to start any rebuilding of the toolbox or the library with an already working installation, with the following steps in mind:

- When changing a function, or introducing a new one, always start at the documentation, and provide all information which is required for automated generation of dependent files.
- Compile the documentation in latex, and process the output with makemenu etc.
- Run the routines which are necessary to update the .NET, COM, native DLL and spreadsheet interfaces.
- Decide whether you need to update the mpNumC.h header file.

Alternatively, you could start with a breaking change in one of the underlying libraries (e.g. GMP), recompile them first, then recompile all of the dependent libraries.

B.1 Building the Library, Part 1

To compile the basic underlying libraries, MSYS2 and a recent version of GCC with pthread support are required.

B.1.1 Downloading GCC and the MinGW-w64 toolchain

From http://tdm-gcc.tdragon.net/download download

```
tdm64-gcc-4.9.2-3.exe
```

and run the program.
In the start dialogue, select "Create", and in the following dialogue, select "MinGW-w64/TDM64 (32-bit and 64-bit)". Click "Next" to proceed to the licence terms and accept them by clicking "Next" again.
Choose an installation directory, in this example

```
C:\TDM-GCC-64
```

and confirm by clicking "Next". Select a download mirror that is geographically close to you, and click "Next" again. In the following panel, select the type of install (for our purposes, "TDM-GCC Recommended. C/C++" is fine), and click "Install".

B.1.2 Downloading, installing and configuring MSYS2

MSYS2 is an updated, modern version of MSYS, both of which are Cygwin (POSIX compatibility layer) forks with the aim of better interoperability with native Windows software.
The name is a contraction of Minimal SYStem 2, and aims to provide support to facilitate using the bash shell, Autotools, revision control systems and the like for building native Windows applications using MinGW-w64 toolchains.
For additional infirmation, see the project's website http://msys2.github.io/.

B.1.2.1 Downloading, installing and configuring MSYS2 32 bit

Download the latest version of MSYS2 32 bit, in this example the file

```
msys2-i686-20150202.exe
```

from http://msys2.github.io/.
or alternatively from http://sourceforge.net/projects/msys2/files/Base/i686/.

Start the program and wait until the installation has finished. Then confirm to start MSYS2. In the MSYS2 command prompt, type:

```
pacman-key --init     #Download keys
```

Restart MSYS2, then type

```
pacman -Syu           #Update package database and full system upgrade
```

Restart MSYS2, then type

```
pacman -S diffutils git m4 make patch tar msys/openssh
```

Copy the files from

```
C:\TDM-GCC-64
```

into

```
C:\msys32\mingw32
```

Restart MSYS2, then type

```
export PATH=/usr/local/bin:/usr/bin:/opt/bin:/mingw32/bin
```

Finally, in the directory

```
C:\Mingw32
```

execute (as administrator) the command file

```
autorebase.bat
```

B.1.3 Downloading installing and configuring Code::Blocks

Code::Blocks is a free C, C++ and Fortran IDE. It is designed to be very extensible and fully configurable. It can be downloaded from
Codeblocks: Codeblocks.
Unpack the file ExternalTools.exe in the mpFormula40 directory. Start the mpFormula Shell and select Compilers - Codeblocks. This will start Codeblocks 13.
In Codeblocks, choose the menu item Settings - Compiler... In the dialogue box, open the drop-down list Selected Compiler and navigate to GNU GCC Compiler 32 bit (at the very bottom of the list, you may need to scroll down).
Select the tab "Toolchain executables". In the text box "Compiler's installation directory", enter the absolute path to the directory containing the compiler, e.g.

```
C:\msys32\mingw32
```

Select the sub-tab "Program Files" and make sure that the entries are as follows:

```
C Compiler: x86_64-w64-mingw32-gcc.exe
C++ compiler: x86_64-w64-mingw32-g++.exe
Linker for dynamic libs: x86_64-w64-mingw32-g++.exe
Linker for static libs: ar.exe
Debugger: GDB/CDB debugger: Default
Resource compiler: windres.exe
Make program: mingw32-make.exe
```

B.1.4 Downloading, compiling and installing GMP

The source code of GMP can be downloaded from
GMP: http://gmplib.org/ .
The license is the GNU Lesser General Public License, Version 3 (see appendix D.1.3)
The contributors are listed in section C.1.1
The mpformula library uses GMP version 6.0.0.

Compiling and installing needs to be done separately for 32 bit and 64 bit. The following detailed instructions are for 32 bit. For 64 bit, replace `MP32` by `MP64` and `Win32` by `Win64` in all file names and directory names, e.g. instead of `gmp_configure_Win32_Release.sh` for 32 bit use `gmp_configure_Win64_Release.sh` for 64 bit.

Within Windows Explorer, copy the directory `gmp-6.0.0` from the `ExternalLibraries` directory to the `ExternalTools\msys32\home\MP32` directory.

Within Windows Explorer, copy the file `gmp_configure_Win32_Release.sh` from the `mpFormulaC\Source\Configure\gmp` directory to the `ExternalTools\msys32\home\MP32\gmp-6.0.0` directory.

Start MSYS2. Within MSYS2, type

```
cd ..
```

to change to the home directory, and then type

```
cd MP32/gmp-6.0.0
```

to change to the

```
/home/MP32/gmp-6.0.0
```

directory. Within MSYS2, type

```
ls *.sh
```

to list only the shell scripts. Within MSYS2, type

```
bash gmp_configure_Win32_Release.sh
```

to start the shell script which will configure the GMP source files for compilation of a 32 bit static library. This will take some time to complete. Once this has been completed (i.e. the command prompt has returned), type

```
make
```

to start the actual compilation. Again, this will take some time to complete. Once this has been done (i.e. the command prompt has returned), type

```
make check
```

to start the compilation and execution of a number of test programs. All tests should pass. Only if all of these steps have been completed successfully, you should finally type

```
make install
```

to install the compiled files into the folder

```
C:\Extra\mpFormula40\ExternalTools1\msys\local\Win32
```

B.1.5 Downloading, compiling and installing MPFR

The source code of MPFR can be downloaded from
MPFR: http://www.mpfr.org/ .
The license is the GNU Lesser General Public License, Version 3 (see appendix D.1.3)
The contributors are listed in section C.1.2
The mpformula library uses MPFR version 3.1.2.

Compiling and installing needs to be done separately for 32 bit and 64 bit. The following detailed instructions are for 32 bit. For 64 bit, replace `MP32` by `MP64` and `Win32` by `Win64` in all file names and directory names, e.g. instead of `mpfr_configure_Win32_Release.sh` for 32 bit use `mpfr_configure_Win64_Release.sh` for 64 bit.

Within Windows Explorer, copy the directory `mpfr-3.1.2` from the `ExternalLibraries` directory to the `ExternalTools\msys32\home\MP32` directory.

Within Windows Explorer, copy the file `mpfr_configure_Win32_Release.sh` from the `mpFormulaC\Source\Configure\mpfr` directory to the `ExternalTools\msys32\home\MP32\mpfr-3.1.2` directory.

Start MSYS2. Within MSYS2, type

```
cd ..
```

to change to the home directory, and then type

```
cd MP32/mpfr-3.1.2
```

to change to the

```
/home/MP32/mpfr-3.1.2
```

directory. Within MSYS2, type

```
ls *.sh
```

to list only the shell scripts. Within MSYS2, type

```
bash mpfr_configure_Win32_Release.sh
```

to start the shell script which will configure the MPFR source files for compilation of a 32 bit static library. This will take some time to complete. Once this has been completed (i.e. the command prompt has returned), type

```
make
```

to start the actual compilation. Again, this will take some time to complete. Once this has been done (i.e. the command prompt has returned), type

```
make check
```

to start the compilation and execution of a number of test programs. All tests should pass. Only if all of these steps have been completed successfully, you should finally type

```
make install
```

to install the compiled files into the folder

```
C:\Extra\mpFormula40\ExternalTools1\msys\local\Win32
```

B.1.6 Downloading, compiling and installing MPC

The source code of MPC can be downloaded from
MPC: http://www.multiprecision.org/index.php?prog=mpc .
The license is the GNU Lesser General Public License, Version 3 (see appendix D.1.3)
The contributors are listed in section C.1.3
The mpformula library uses MPC version 1.0.3.

Compiling and installing needs to be done separately for 32 bit and 64 bit. The following detailed instructions are for 32 bit. For 64 bit, replace `MP32` by `MP64` and `Win32` by `Win64` in all file names and directory names, e.g. instead of `mpc_configure_Win32_Release.sh` for 32 bit use `mpc_configure_Win64_Release.sh` for 64 bit.

Within Windows Explorer, copy the directory `mpc-1.0.3` from the `ExternalLibraries` directory to the `ExternalTools\msys32\home\MP32` directory.

Within Windows Explorer, copy the file `mpc_configure_Win32_Release.sh` from the `mpFormulaC\Source\Configure\mpc` directory to the `ExternalTools\msys32\home\MP32\mpc-1.0.3` directory.

Start MSYS2. Within MSYS2, type

```
cd ..
```

to change to the home directory, and then type

```
cd MP32/mpc-1.0.3
```

to change to the

```
/home/MP32/mpc-1.0.3
```

directory. Within MSYS2, type

```
ls *.sh
```

to list only the shell scripts. Within MSYS2, type

```
bash mpc_configure_Win32_Release.sh
```

to start the shell script which will configure the MPC source files for compilation of a 32 bit static library. This will take some time to complete. Once this has been completed (i.e. the command prompt has returned), type

```
make
```

to start the actual compilation. Again, this will take some time to complete. Once this has been done (i.e. the command prompt has returned), type

```
make check
```

to start the compilation and execution of a number of test programs. All tests should pass. Only if all of these steps have been completed successfully, you should finally type

```
make install
```

to install the compiled files into the folder

```
C:\Extra\mpFormula40\ExternalTools1\msys\local\Win32
```

B.1.7 Downloading, compiling and installing MPFI

The source code of MPFI can be downloaded from
MPFI: https://gforge.inria.fr/projects/mpfi/ .
The license is the GNU Lesser General Public License, Version 3 (see appendix D.1.3)
The contributors are listed in section C.1.4
The mpformula library uses MPFI version 1.5.2., which is the same as 1.5.1 with a patch to enable DLLs.

Compiling and installing needs to be done separately for 32 bit and 64 bit. The following detailed instructions are for 32 bit. For 64 bit, replace `MP32` by `MP64` and `Win32` by `Win64` in all file names and directory names, e.g. instead of `mpfi_configure_Win32_Release.sh` for 32 bit use `mpfi_configure_Win64_Release.sh` for 64 bit.

Within Windows Explorer, copy the directory `mpfi-1.5.2` from the `ExternalLibraries` directory to the `ExternalTools\msys32\home\MP32` directory.

Within Windows Explorer, copy the file `mpfi_configure_Win32_Release.sh` from the `mpFormulaC\Source\Configure\mpfi` directory to the `ExternalTools\msys32\home\MP32\mpfi-1.5.2` directory.

Start MSYS2. Within MSYS2, type

```
cd ..
```

to change to the home directory, and then type

```
cd MP32/mpfi-1.5.2
```

to change to the

```
/home/MP32/mpfi-1.5.2
```

directory. Within MSYS2, type

```
ls *.sh
```

to list only the shell scripts. Within MSYS2, type

```
bash mpfi_configure_Win32_Release.sh
```

to start the shell script which will configure the MPFI source files for compilation of a 32 bit static library. This will take some time to complete. Once this has been completed (i.e. the command prompt has returned), type

```
make
```

to start the actual compilation. Again, this will take some time to complete. Once this has been done (i.e. the command prompt has returned), type

```
make check
```

to start the compilation and execution of a number of test programs. All tests should pass. Only if all of these steps have been completed successfully, you should finally type

```
make install
```

to install the compiled files into the folder

```
C:\Extra\mpFormula40\ExternalTools1\msys\local\Win32
```

B.1.8 Downloading, compiling and installing FLINT

FLINT: http://www.flintlib.org/ .

The license is the GNU General Public License, Version 2 or later (see appendix D.1.1 and D.1.4)

The contributors are listed in section C.1.5

The mpFormulaC library uses FLINT version 2.4.5.

Compiling and installing needs to be done separately for 32 bit and 64 bit. The following detailed instructions are for 32 bit. For 64 bit, replace `MP32` by `MP64` and `Win32` by `Win64` in all file names and directory names, e.g. instead of `flint_configure_Win32_Release.sh` for 32 bit use `flint_configure_Win64_Release.sh` for 64 bit.

Within Windows Explorer, copy the directory `flint-2.4.5` from the `ExternalLibraries` directory to the `ExternalTools\msys32\home\MP32` directory.

Within Windows Explorer, copy the file `flint_configure_Win32_Release.sh` from the `..\mpFormulaC\Source\Configure\flint` directory to the `..\ExternalTools\msys32\home\MP32\flint-2.4.5` directory.

Start MSYS2. Within MSYS2, type

```
cd ..
```

to change to the home directory, and then type

```
cd MP32/flint-2.4.5
```

to change to the

```
/home/MP32/flint-2.4.5
```

directory. Within MSYS2, type

```
ls *.sh
```

to list only the shell scripts. Within MSYS2, type

```
bash flint_configure_Win32_Release.sh
```

to start the shell script which will configure the FLINT source files for compilation of a 32 bit static library. This will take some time to complete. Once this has been completed (i.e. the command prompt has returned), type

```
make
```

to start the actual compilation. Again, this will take some time to complete. Once this has been done (i.e. the command prompt has returned), type

```
make check
```

to start the compilation and execution of a number of test programs. All tests should pass. Only if all of these steps have been completed successfully, you should finally type

```
make install
```

to install the compiled files into the folder

```
..\ExternalTools\msys\local\Win32
```

B.1.8.1 Remaining issues with the 64 bit build

For FLINT 32 bit compilation works without errors, and all tests pass.

For FLINT 64 bit, compilation works without errors or warnings; however, in the original distribution, five test programs crash. To run all test programs, copy the files `t-clog.c`, `t-clog_ui.c`, `t-flog.c`, `t-flog_ui.c` from the folder

`..\mpFormulaC\Source\Configure\flint\FailedTests64bit_Dummy\fmpz\test`

into the folder

`..\mpFormulaC\ExternalTools\msys\home\MP\flint-2.4.5\fmpz\test`

Also, copy the file `t-factor_zassenhaus.c` from the folder

`..\mpFormulaC\Source\Configure\flint\FailedTests64bit_Dummy\fmpz_poly_factor\test`

into the folder

`..\mpFormulaC\ExternalToolsC\msys\home\MP\flint-2.4.5\fmpz_poly_factor\test`

and only then run make check.

B.1.9 Downloading, compiling and installing ARB

ARB: http://fredrikj.net/arb/# .

The actual download site is
https://github.com/fredrik-johansson/arb/releases .

Additional information is available from the blog of the author of Arb:
http://fredrikj.net/blog/ .

From the same author a FLINT/ARB Wrapper in Python is available:
http://fredrikj.net/python-flint/ .

Also from the same author a related Python library is available:
http://mpmath.org/ .

The license is the GNU General Public License, Version 2 or later (see appendix D.1.1 and D.1.4)
The contributors are listed in section C.1.6
The mpformula library uses ARB version 2.5.0.

Compiling and installing needs to be done separately for 32 bit and 64 bit. The following detailed instructions are for 32 bit. For 64 bit, replace `MP32` by `MP64` and `Win32` by `Win64` in all file names and directory names, e.g. instead of `arb_configure_Win32_Release.sh` for 32 bit
use `arb_configure_Win64_Release.sh` for 64 bit.

Within Windows Explorer, copy the directory `arb-2.5.0` from the `ExternalLibraries` directory to the `ExternalTools\msys32\home\MP32` directory.

Within Windows Explorer, copy the file `arb_configure_Win32_Release.sh` from the `mpFormulaC\Source\Configure\arb` directory to the `ExternalTools\msys32\home\MP32\arb-2.5.0` directory.

Start MSYS2. Within MSYS2, type

```
cd ..
```

to change to the home directory, and then type

```
cd MP32/arb-2.5.0
```

to change to the

```
/home/MP32/arb-2.5.0
```

directory. Within MSYS2, type

```
ls *.sh
```

to list only the shell scripts. Within MSYS2, type

```
bash arb_configure_Win32_Release.sh
```

to start the shell script which will configure the ARB source files for compilation of a 32 bit static library. This will take some time to complete. Once this has been completed (i.e. the command prompt has returned), type

```
make
```

to start the actual compilation. Again, this will take some time to complete. Once this has been done (i.e. the command prompt has returned), type

```
make check
```

to start the compilation and execution of a number of test programs. All tests should pass. Only if all of these steps have been completed successfully, you should finally type

```
make install
```

to install the compiled files into the folder

```
..\ExternalTools1\msys\local\Win32
```

B.1.9.1 Remaining issues with the 32 bit build

For ARB 32 bit, compilation works without errors or warnings, and all tests but two pass.
To run all test programs, copy the file `t-epsilon_arg.c` from the folder

```
..\mpFormulaC\Source\Configure\arb\FailedTests32bit_Dummy\acb_modular\test
```

into the folder

```
..\ExternalTools\msys\home\MP32\arb-2.5.0\acb_modular\test
```

and also copy the file `t-set_str.c` from the folder

```
..\mpFormulaC\Source\Configure\arb\FailedTests32bit_Dummy\arb\test
```

into the folder

```
..\ExternalTools\msys\home\MP32\arb-2.5.0\arb\test
```

and only then run make check.

For ARB 64 bit, compilation works without errors but a number of warnings, and currently all tests crash (reason unknown).

B.2 Building the Library, Part 2

B.2.1 Downloading, compiling and installing XSC-MPFI

XSC-MPFI: http://www2.math.uni-wuppertal.de/ xsc/xsc/cxscsoftware.htmlmpfr-mpfi .
The license is the GNU Library General Public License, Version 2 (see appendix D.1.2)
The contributors are listed in section C.1.8
The mpformula library uses XSC-MPFI version 0.10.0.

B.2.2 Downloading, compiling and installing libmpdec

libmpdec is a complete implementation of the General Decimal Arithmetic Specification, which defines a general purpose arbitrary precision data type together with rigorously specified functions and rounding behavior.

The license is the Simplified BSD License (see appendix D.1.3)
The contributors are listed in section C.1.4
The mpformula library uses libmpdec version 2.4.1.

The libmpdec library could historically be downloaded from

libmpdec: http://www.bytereef.org/mpdecimal/ .
unfortunately, this website seems to be no longer reachable. However, a copy of the source code is still available from a numver of sites, including

libmpdec: https://launchpad.net/ubuntu/+source/mpdecimal/2.4.1-1/ .

The file to download is mpdecimal-2.4.1.orig.tar.gz. which should unpacked into the ExternalLibraries folder.

Requirements ————
- Visual Studio 2008 or later.
- For the scripted build vcvarsall.bat must be in the PATH.
download and unpack the source file into the (working) folder

```
I:\Data\mpFormula40\ExternalTools2\mpdecimal-2.4.1
```

Copy the file Makefile.vc from the folder

```
I:\Data\mpFormula40\Toolbox\mpNum\Configure\libmpdec
```

to the folder

```
I:\Data\mpFormula40\ExternalTools2\mpdecimal-2.4.1\libmpdec
```

This will make sure that the vc runtime is included statically in the dll.
open a Visual Studio Command Prompt 32 bit as administrator.
Within the Visual Studio Command Prompt, navigate to

```
I:\Data\mpFormula40\ExternalTools2\mpdecimal-2.4.1\vcbuild
```

type
vcbuild32.bat
to start building the 32 bit dll. If successful, the static library, the dynamic library, the common header file and an executable for running the unit tests should be in the dist32 directory.
open a Visual Studio Command Prompt 64 bit as administrator.
Within the Visual Studio Command Prompt, navigate to

```
I:\Data\mpFormula40\ExternalTools2\mpdecimal-2.4.1\vcbuild
```

type
vcbuild64.bat
to start building the 64 bit dll. If successful, the static library, the dynamic library, the common header file and an executable for running the unit tests should be in the dist32 directory.
Get the unit tests ——————
Run gettests.bat. This creates a directory 'testdata' and copies additional tests into the directory. If wget is installed (Cygwin), the script tries to download IBM's official test cases and copy them to 'testdata'.
Otherwise, download:
http://speleotrove.com/decimal/dectest.zip
Unzip the archive such that all .decTest files from the archive are in the testdata directory. The directory structure should look like this:

```
vcbuild\official.decTest
vcbuild\additional.decTest
vcbuild\testdata\*.decTest
```

Run the unit tests ——————
Depending on the build, run runtest64.bat or runtest32.bat.

B.3 Building the Library, Part 3

B.3.1 Boost Math

Boost Math: http://www.boost.org/doc/libs/1_57_0/libs/math/doc/html/index.html .
The roadmap for the development version is available at
http://www.boost.org/doc/libs/master/libs/math/doc/html/math_toolkit/history2.html .
http://www.boost.org/doc/libs/master/libs/math/doc/html/index.html .
The license is the Boost Software License, Version 1.0(see appendix D.2.2)
The contributors are listed in section C.1.12
The mpformula library uses Boost Math version 1.57.0.

B.3.2 Boost Random

Boost Random: http://www.boost.org/doc/libs/1_57_0/doc/html/boostrandom.html .
The license is the Boost Software License, Version 1.0(see appendix D.2.2)
The contributors are listed in section C.1.13
The mpformula library uses Boost Random version 1.57.0.

B.3.3 Eigen

Eigen: http://eigen.tuxfamily.org/index.php?title=MainPage .
The license is the Mozilla Public License, Version 2 (see appendix D.2.1)
The contributors are listed in section C.1.10
The mpformula library uses Eigen version 3.2.2.

B.3.4 Source Code derived from other libraries

B.3.4.1 MPFRC++

MPFRC++: http://www.holoborodko.com/pavel/mpfr/ . LGPL version 2.1 ?
The current license on the MPFRC++ web site is the GNU General Public License, Version 3 (see appendix D.1.4)
The MPFRC++ has not used an official versioning system from early on. The version used by the mpformula library is a version optimized for use with the Eigen library, which is part of the Eigen distribution (in the folder \unsupported\test\mpreal, dated 26Feb2014), and which does not have a version number. The license of this version is the GNU Library General Public License, Version 2.1 (see appendix D.1.2)
The contributors are listed in section C.1.9

B.4 Building the documentation

The following software was used to build the documentation:
miktex 2.9.4813 : miktex.
texniccenter 2.02: texniccenter.
After installing these programs one additional step is needed:
Build → Select Output Profile: Latex ⇒ PDF
Build → Define Output Profile: Latex ⇒ PDF. In this dialogue box, select the tab ”Postprocessor”
In the Listbox ”Processors”, add an item and name it ”Nomenclature”
In the Textfield ”Executable:”, enter the full path to ”miktex-makeindex.exe” ,like

```
C:\Program Files\MiKTeX 2.9\miktex\bin\x64\miktex-makeindex.exe
```

In the Textfield ”Arguments:”, enter the following:

```
-s nomencl.ist "%tm.nlo" -o "%tm.nls"
```

B.5 Additional libraries

The source code of AMath and DAMath can be downloaded from:

http://www.wolfgang-ehrhardt.de .
You need to download 2 separate files to support compilation of both 32 bit and 64 bit dlls:

- amath_2014-05-11.zip contains the Amath package
- damath_2014-05-11.zip contains the DAMath package

B.5.1 MPIR

MPIR: http://www.mpir.org/ .
The license is the GNU Lesser General Public License, Version 3 (see appendix D.1.3)
The contributors are listed in section C.1.1
The mpformula library uses MPIR version 2.6.0.

B.5.2 gmpfrxx

gmpfrxx: http://math.berkeley.edu/ wilken/code/gmpfrxx/ .
The license is the GNU Lesser General Public License, Version 3 (see appendix D.1.3)

B.6 Working Notes

B.7 Where to find VB Code

Most new code is loaded when starting the current version of Stats32.xls
The code itself is then dynamically loaded into Source.xls.
Within Source.xls, most new noncentral code is in Approx.bas.
This includes Doubly noncentral t and F.
New Code for Pearson Rho is in ALL_DistribN (at the end).
LambdaDemo is in ALL_DistribX.
The finite series for cdis, tdis and fdis are in tdisn_Owen.

B.8 How to run Permutation Code

Within Stats32.xls, goto Interface.bas, and within this module to DemoTabularProcs
For additional fine tuning: Within Source.xls, goto XLInterface.bas, and within this module to TabularProcs.

Appendix C

Acknowledgements

C.1 Contributors to libraries used in the numerical routines

C.1.1 Contributors to GMP

The following text has been copied from the GMP manual (5.1.2):

"Torbjörn Granlund wrote the original GMP library and is still the main developer. Code not explicitly attributed to others, was contributed by Torbjörn. Several other individuals and organizations have contributed GMP. Here is a list in chronological order on first contribution:

Gunnar Sjödin and Hans Riesel helped with mathematical problems in early versions of the library.

Richard Stallman helped with the interface design and revised the first version of this manual.

Brian Beuning and Doug Lea helped with testing of early versions of the library and made creative suggestions.

John Amanatides of York University in Canada contributed the function `mpz_probab_prime_p`.

Paul Zimmermann wrote the REDC-based mpz_powm code, the Schönhage-Strassen FFT multiply code, and the Karatsuba square root code. He also improved the Toom3 code for GMP 4.2. Paul sparked the development of GMP 2, with his comparisons between bignum packages. The ECMNET project Paul is organizing was a driving force behind many of the optimizations in GMP 3. Paul also wrote the new GMP 4.3 nth root code (with Torbjörn).

Ken Weber (Kent State University, Universidade Federal do Rio Grande do Sul) contributed now defunct versions of `mpz_gcd`, `mpz_divexact`, `mpn_gcd`, and `mpn_bdivmod`, partially supported by CNPq (Brazil) grant 301314194-2.

Per Bothner of Cygnus Support helped to set up GMP to use Cygnus' configure. He has also made valuable suggestions and tested numerous intermediary releases.

Joachim Hollman was involved in the design of the `mpf` interface, and in the `mpz` design revisions for version 2.

Bennet Yee contributed the initial versions of `mpz_jacobi` and `mpz_legendre`.

Andreas Schwab contributed the files `mpn/m68k/lshift.S` and `mpn/m68k/rshift.S` (now in `.asm`

form).

Robert Harley of Inria, France and David Seal of ARM, England, suggested clever improvements for population count. Robert also wrote highly optimized Karatsuba and 3-way Toom multiplication functions for GMP 3, and contributed the ARM assembly code.

Torsten Ekedahl of the Mathematical department of Stockholm University provided significant inspiration during several phases of the GMP development. His mathematical expertise helped improve several algorithms.

Linus Nordberg wrote the new configure system based on autoconf and implemented the new random functions.

Kevin Ryde worked on a large number of things: optimized x86 code, m4 asm macros, parameter tuning, speed measuring, the configure system, function inlining, divisibility tests, bit scanning, Jacobi symbols, Fibonacci and Lucas number functions, printf and scanf functions, perl interface, demo expression parser, the algorithms chapter in the manual, `gmpasm-mode.el`, and various miscellaneous improvements elsewhere.

Kent Boortz made the Mac OS 9 port.

Steve Root helped write the optimized alpha 21264 assembly code.

Gerardo Ballabio wrote the `gmpxx.h` C++ class interface and the C++ `istream` input routines.

Jason Moxham rewrote `mpz_fac_ui`.

Pedro Gimeno implemented the Mersenne Twister and made other random number improvements.

Niels Möller wrote the sub-quadratic GCD, extended GCD and jacobi code, the quadratic Hensel division code, and (with Torbjörn) the new divide and conquer division code for GMP 4.3. Niels also helped implement the new Toom multiply code for GMP 4.3 and implemented helper functions to simplify Toom evaluations for GMP 5.0. He wrote the original version of `mpn_mulmod_bnm1`, and he is the main author of the mini-gmp package used for gmp bootstrapping.

Alberto Zanoni and Marco Bodrato suggested the unbalanced multiply strategy, and found the optimal strategies for evaluation and interpolation in Toom multiplication.

Marco Bodrato helped implement the new Toom multiply code for GMP 4.3 and implemented most of the new Toom multiply and squaring code for 5.0. He is the main author of the current mpn_mulmod_bnm1 and mpn_mullo_n. Marco also wrote the functions mpn_invert and mpn_invertappr. He is the author of the current combinatorial functions: binomial, factorial, multifactorial, primorial.

David Harvey suggested the internal function `mpn_bdiv_dbm1`, implementing division relevant to Toom multiplication. He also worked on fast assembly sequences, in particular on a fast AMD64 `mpn_mul_basecase`. He wrote the internal middle product functions `mpn_mulmid_basecase`, `mpn_toom42_mulmid`, `mpn_mulmid_n` and related helper routines.

Martin Boij wrote `mpn_perfect_power_p`.

Marc Glisse improved `gmpxx.h`: use fewer temporaries (faster), specializations of `numeric_limits` and `common_type`, C++11 features (move constructors, explicit bool conversion, UDL), make the conversion from `mpq_class` to `mpz_class` explicit, optimize operations where one argument is a small compile-time constant, replace some heap allocations by stack allocations. He also fixed

the eofbit handling of C++ streams, and removed one division from `mpq/aors.c`.

(This list is chronological, not ordered after significance. If you have contributed to GMP but are not listed above, please tell `gmp − devel@gmplib.org` about the omission!)

The development of floating point functions of GNU MP 2, were supported in part by the ESPRIT-BRA (Basic Research Activities) 6846 project POSSO (POlynomial System SOlving).

The development of GMP 2, 3, and 4 was supported in part by the IDA Center for Computing Sciences.

Thanks go to Hans Thorsen for donating an SGI system for the GMP test system environment."

C.1.2 Contributors to MPFR

The following text has been copied from the MPFR manual (3.1.2):

"The main developers of MPFR are Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, Philippe Théveny and Paul Zimmermann.

Sylvie Boldo from ENS-Lyon, France, contributed the functions `mpfr_agm` and `mpfr_log`. Sylvain Chevillard contributed the `mpfr_ai` function.

David Daney contributed the hyperbolic and inverse hyperbolic functions, the base-2 exponential, and the factorial function.

Alain Delplanque contributed the new version of the `mpfr_get_str` function.

Mathieu Dutour contributed the functions `mpfr_acos`, `mpfr_asin` and `mpfr_atan`, and a previous version of `mpfr_gamma`.

Laurent Fousse contributed the `mpfr_sum` function.

Emmanuel Jeandel, from ENS-Lyon too, contributed the generic hypergeometric code, as well as the internal function `mpfr_exp3`, a first implementation of the sine and cosine, and improved versions of `mpfr_const_log2` and `mpfr_const_pi`.

Ludovic Meunier helped in the design of the `mpfr_erf` code.

Jean-Luc Rémy contributed the `mpfr_zeta` code.

Fabrice Rouillier contributed the `mpfr_xxx_z` and `mpfr_xxx_q` functions, and helped to the Microsoft Windows porting.

Damien Stehlé contributed the `mpfr_get_ld_2exp` function.

We would like to thank Jean-Michel Muller and Joris van der Hoeven for very fruitful discussions at the beginning of that project, Torbjörn Granlund and Kevin Ryde for their help about design issues, and Nathalie Revol for her careful reading of a previous version of this documentation. In particular Kevin Ryde did a tremendous job for the portability of MPFR in 2002-2004.

The development of the MPFR library would not have been possible without the continuous support of INRIA, and of the LORIA (Nancy, France) and LIP (Lyon, France) laboratories. In particular the main authors were or are members of the PolKA, Spaces, Cacao and Caramel project-teams at LORIA and of the Arénaire and AriC project-teams at LIP.

This project was started during the Fiable (reliable in French) action supported by INRIA, and continued during the AOC action. The development of MPFR was also supported by a grant (202F0659 00 MPN 121) from the Conseil Régional de Lorraine in 2002, from INRIA by an "associate engineer" grant (2003-2005), an "opération de développement logiciel" grant (2007-2009), and the post-doctoral grant of Sylvain Chevillard in 2009-2010. The MPFR-MPC workshop in June 2012 was partly supported by the ERC grant ANTICS of Andreas Enge."

C.1.3 Contributors to MPC

The main developers of MPC are Andreas Enge, Philippe Théveny and Paul Zimmermann.

C.1.4 Contributors to MPFI

The following text has been copied from the MPFI manual (1.5.1):

"MPFI has been written by Fabrice Rouillier, Nathalie Revol, Sylvain Chevillard, Hong Diep Nguyen, Christoph Lauter and Philippe Théveny. Its development has greatly benefited from the patient and supportive help of the MPFR team."

C.1.5 Contributors to FLINT

The following text has been copied from the FLINT manual (2.4.3):

xxxx

C.1.6 Contributors to ARB

The following text has been copied from the FLINT manual (2.4.3):

xxxx

C.1.7 Contributors to XSC

The main developers of XSC are Frithjof Blomquist, Werner Hofschuster, Walter Krämer.

The Credits sction from the C-XSC - A C++ Class Library for Extended Scientific Computing Documentation 2.5.3 main page http://www2.math.uni-wuppertal.de/ xsc/xsc/cxsc/apidoc/html/main.html contains the following statement:

"The work on C-XSC started in 1990 at the Institute for Applied Mathematics (Prof. Kulisch), University of Karlsruhe. Many colleagues and scientists have directly and indirectly contributed to the realization of C-XSC. The authors would like to thank each of them for his or her cooperation. Special thanks go to U. AllendÃűrfer, C. Baumhof, H. Berlejung, H. Bleher, H. BÃűhm, B. Bohl, G. Bohlender, F. Blomquist, K. Braune, H.H. Chen, D. Cordes, A. Davidenkoff, H.-C. Fischer, M. Grimmer, K. GrÃijner, R. Hammer, M. Hinz, M. Hocks, B. HÃűffgen, W. Hofschuster, P. Januscke, E. Kaucher, R. Kelch, R. Kirchner, R. Klatte, W. Klein, W. KrÃďmer, U. Kulisch, C. Lawo, M. Metzger, W.L. Miranker, M. Neaga, M. Neher, D. Ratz, M. Rauch, S. Ritterbusch, S.M. Rump, R. Saier, D. Schiriaev, L. Schmidt, G. Schumacher, U. Storck, J. SuckfÃijll, F. Toussaint, C. Ullrich, W. Walter, S. Wedner, G. Werheit, A. Wiethoff, H.W. Wippermann, J. Wolff von Gudenberg and M. Zimmer.

C-XSC is an outcome of an ongoing collaboration of the Institute for Applied Mathematics (Prof. Kulisch), University of Karlsruhe and the Institute for Scientific Computing/Software Engineering (Prof. KrÃďmer), University of Wuppertal. For the latest news and up to date software contact http://www.math.uni-wuppertal.de/ xsc/ .

Thanks to the referees for valuable comments and suggestions."

C.1.8 Contributors to XSC-MPFI

The main developers of XSC-MPFI are Frithjof Blomquist, Werner Hofschuster, Walter Krämer. The library is in part based on work by Hans-Stephan Brand.

C.1.9 Contributors to MPFRC++

The main developer of MPFRC++ is Pavel Holoborodko.
Contributors: Dmitriy Gubanov, Konstantin Holoborodko, Brian Gladman, Helmut Jarausch, Fokko Beekhof, Ulrich Mutze, Heinz van Saanen, Pere Constans, Peter van Hoof, Gael Guennebaud, Tsai Chia Cheng, Alexei Zubanov, Jauhien Piatlicki, Victor Berger, John Westwood.

C.1.10 Contributors to Eigen

The following statement is copied from the Eigen Homepage:

"The Eigen project was started by Benoît Jacob (founder) and Gaël Guennebaud (guru). Many other people have since contributed their talents to help make Eigen successful. Here's an alphabetical list: (note to contributors: do add yourself!)

Philip Avery: Fix bug and add functionality to AutoDiff module
Abraham Bachrach: Added functions for cwise min/max with a scalar
Sebastien Barthelemy: Fix EIGEN_INITIALIZE_MATRICES_BY_NAN
Carlos Becker: Wrote some of the pages of the tutorial
David Benjamin: Artwork: the owls
Cyrille Berger: Fix error in logic of installation script
Armin Berres: Lots of fixes (compilation warnings and errors)
Jose Luis Blanco: Build fixes for MSVC and AMD64, correction in docs
Mark Borgerding: FFT module
Romain Bossart: Updates to Sparse solvers
Kolja Brix: Added documentation to Householder module, fixes for ARPACK wrapper and KroneckerProduct
Gauthier Brun: Making a start with a divide-and-conquer SVD implementation
Thomas Capricelli: Migration to mercurial, Non-linear optimization and numerical differentiation, cron-job to update the online dox
Nicolas Carre: Making a start with a divide-and-conquer SVD implementation
Jean Ceccato: Making a start with a divide-and-conquer SVD implementation
Andrew Coles: Fixes (including a compilation error)r
Marton Danoczy: MSVC compilation fix, support for ARM NEON with Clang 3.0 and LLVM-GCC
Jeff Dean: Fix in vectorized square root for small arguments
Christian Ehrlicher: MSVC compilation fix
Daniel Gomez Ferro: Improvements in Sparse and in matrix product

Rohit Garg: Vectorized quaternion and cross products, improved integer product
Mathieu Gautier: QuaternionMap and related improvements
Anton Gladky: Visual Studio 2008 and GCC 4.6 compilation fixes
Stuart Glaser: Prevent allocations in LU decomposition
Marc Glisse: C++11 compilation issues (suffices for literals)
Frederic Gosselin: Improve filter for hidden files in CMake
GaÃńl Guennebaud: Core developer
Philippe Hamelin: Allow CMake project to be included in another project
Marcus D. Hanwell: CMake improvements. Marcus is a developer at Kitware!
David Harmon: Arpack support module
Chen-Pang He: Many improvements to MatrixFunctions and KroneckerProduct modules
Hauke Heibel: Extended matrix functions, STL compatibility, Splines, CMake improvements, and more ...
Christoph Hertzberg: Quaternions, shifts for Cholmod, bug fixes, lots of user support on forums and IRC
Pavel Holoborodko: Multi-precision support with MPFR C++
Tim Holy: Improvements to tutorial, LDLT update and downdate
Intel: Back-end to Intel Math Kernel Library (MKL)
Trevor Irons: Square root for complex numbers, fix compile errors and mistake in docs
BenoÃőt Jacob : Core developer
Bram de Jong: Improvement to benchmark suite
Kibeom Kim: Implement *= /= * / operations for VectorwiseOp
Claas KÃűhler: Improvements to Fortran and FFTW in CMake
Alexey Korepanov: Add RealQZ class
Igor Krivenko: Properly cast constants when using non-standard scalars
Marijn Kruisselbrink: CMake fixes
Moritz Lenz: Allow solving transposed problem with SuperLU
Sebastian Lipponer: MSVC compilation support
Daniel Lowenberg: Add SparseView class
David J. Luitz: Bug fix for sparse * dense matrix product
Angelos Mantzaflaris: Fix to allow IncompleteLUT to be used with MPFR
D J Marcin: Fix operator & precedence bug
Konstantinos A. Margaritis: AltiVec and ARM NEON vectorization
Ricard Marxer: Reverse, redux improvements, the count() method, some dox
Vincenzo Di Massa: CMake fix
Christian Mayer: Early code review and input in technical/design discussions
Frank Meier-DÃűrnberg: MSVC compatibility fixes
Keir Mierle: LDLT decomposition and other improvements, help with MPL relicensing
Laurent Montel: CMake improvements. Laurent is (with Alexander) one of the CMake gurus at KDE!
Eamon Nerbonne: Compilation fixes for win32
Alexander Neundorf: CMake improvements. Alexander is (with Laurent) one of the CMake gurus at KDE!
Jason Newton: Componentwise tangent functions
Jitse Niesen: Matrix functions, large improvements in the Eigenvalues module and in the docs, and more ...
Desire Nuentsa: Many improvements to Sparse module: SparseLU, SparseQR, ILUT, PaStiX-Support, âĂę
Jan OberlÃďnder: Compatibility with termios.h

Jos van den Oever: Compilation fix
Michael Olbrich: Early patches, including the initial loop meta-unroller
Simon Pilgrim: Optimizations for NEON
Bjorn Piltz: Visual C compilation fix
Benjamin Piwowarski: Add conservativeResize() for sparse matrices
Zach Ploskey: Copy-editing of tutorial
Giacomo Po: MINRES iterative solver
Sergey Popov: Fix bug in SelfAdjointEigenSolver
Manoj Rajagopalan: Introduce middleRows() / middleCols(), bug fix for nonstandard numeric types
Stjepan Rajko: MSVC compatibility fix
Jure Repinc: CMake fixes
Kenneth Frank Riddile: Lots of Windows/MSVC compatibility fixes, handling of alignment issues
Adolfo Rodriguez: Prevent allocations in matrix decompositions
Peter RomÃąn : Support for SuperLU's ILU factorization
Oliver Ruepp: Bug fix in sparse matrix product with row-major matrices
Radu Bogdan Rusu: Fix compilation warning
Guillaume Saupin: Skyline matrices
Michael Schmidt: Fix in assembly when identifying CPU
Jakob Schwendner: Test for unaligned quaternions
Martin Senst: Bug fix for empty matrices
Benjamin Schindler: gdb pretty printers
Michael Schmidt: Compilation fix connected to min/max
Dennis Schridde: New typedefs like AlignedBox3f
Jakob Schwendner: Benchmark for Geometry module
Sameer Sheorey: Fix gdb pretty printer for variable-size matrices
Andy Somerville: Functions to get intersection between two ParametrizedLines
Alex Stapleton: Help with tough C++ questions
Adam Szalkowski: Bug fix in MatrixBase::makeHouseholder()
Adolfo Rodriguez: Tsourouksdissian Version of JacobiSVD that pre-allocates its resources
Piotr Trojanek: QCC compilation fixes
Anthony Truchet: Bugfix in QTransform and QMatrix support
James Richard Tyrer: CMake fix
Rhys Ulerich: Pkg-config support, improved GDB pretty-printer
Ingmar Vanhassel: CMake fix
Scott Wheeler: Documentation improvements
Urs Wolfer: Fixed a serious warning
Manuel Yguel: Bug fixes, work on inverse-with-check, the Polynomial module
Pierre Zoppitelli: Making a start with a divide-and-conquer SVD implementation

Eigen is also using code that we copied from other sources. They are acknowledged in our sources and in the Mercurial history, but let's also mention them here:

Intel Corporation SSE code for 4x4 matrix inversion taken from here. Tim Davis AMD reordering simplicial sparse Cholesky factorization adapted from SuiteSparse Julien Pommier SSE implementation of exp,log,cos,sin math functions from GMM++ Yousef Saad IncompleteLUT preconditioner coming from ITSOL Minpack authors Algorithms for non linear optimization.

Special thanks to Tuxfamily for the wonderful quality of their services, and the GCC Compile Farm Project that gives us access to many various systems including ARM NEON. ”

C.1.11 Contributors to Boost Multiprecision

The main authors of Boost Multiprecision are John Maddock and Christopher Kormanyos.

The Acknowledgements section states:

"This library would not have happened without:
Christopher Kormanyos' C++ decimal number code.
Paul Bristow for patiently testing, and commenting on the library.
All the folks at GMP, MPFR and libtommath, for providing the "guts" that makes this library work.
"The Art Of Computer Programming", Donald E. Knuth, Volume 2: Seminumerical Algorithms, Third Edition (Reading, Massachusetts: Addison-Wesley, 1997), xiv+762pp.
ISBN 0-201-89684-2"

C.1.12 Contributors to Boost Math

The main authors of the Boost Math Toolkit are Paul A. Bristow, Hubert Holin, Christopher Kormanyos, Bruno Lalande, John Maddock, Johan RÃěde, Benjamin Sobotta, Gautam Sewani, Thijs van den Berg, Daryle Walker, and Xiaogang Zhang.

The Credits and Acknowledgements section states:

"Hubert Holin started the Boost.Math library. The Quaternions, Octonions, inverse hyperbolic functions, and the sinus cardinal functions are his.

Daryle Walker wrote the integer gcd and lcm functions.

John Maddock started the special functions, the beta, gamma, erf, polynomial, and factorial functions are his, as is the "Toolkit" section, and many of the statistical distributions.

Paul A. Bristow threw down the challenge in A Proposal to add Mathematical Functions for Statistics to the C++ Standard Library to add the key math functions, especially those essential for statistics. After JM accepted and solved the diffi cult problems, not only numerically, but in full C++ template style, PAB implemented a few of the statistical distributions. PAB also tirelessly proof-read everything that JM threw at him (so that all remaining editorial mistakes are his fault).

Xiaogang Zhang worked on the Bessel functions and elliptic integrals for his Google Summer of Code project 2006.

Bruno Lalande submitted the "compile time power of a runtime base" code.

Johan RÃěde wrote the optimised floating-point classifi cation and manipulation code, and nonfi nite f acets to permit C99 output of infinities and NaNs. (nonfi nite facets were not added until Boost 1.47 but had been in use with Boost.Spirit). This library was based on a suggestion from Robert Ramey, author of Boost.Serialization. Paul A. Bristow expressed the need for better handling of Input & Output of NaN and infi nity for the C++ Standard Library and suggested following the C99 format.

Antony Polukhin improved lexical cast avoiding stringstream so that it was no longer necessary to use a globale C99 facet to handle nonfinites.

HÃěkan ArdÃű, Boris Gubenko, John Maddock, Markus SchÃűpfl in and Olivier Verdier tested

the floating-point library and Martin Bonner, Peter Dimov and John Maddock provided valuable advice.

Gautam Sewani coded the logistic distribution as part of a Google Summer of Code project 2008.

M. A. (Thijs) van den Berg coded the Laplace distribution. (Thijs has also threatened to implement some multivariate distributions).

Thomas Mang requested the inverse gamma in chi squared distributions for Bayesian applications and helped in their implementation, and provided a nice example of their use.

Professor Nico Temme for advice on the inverse incomplete beta function.

Victor Shoup for NTL, without which it would have much more diffi cult to produce high accurac y constants, and especially the tables of accurate values for testing.

We are grateful to Joel Guzman for helping us stress-test his Boost.Quickbook program used to generate the html and pdf versions of this document, adding several new features en route.

Plots of the functions and distributions were prepared in W3C standard Scalable Vector Graphic (SVG) format using a program created by Jacob Voytko during a Google Summer of Code (2007). From 2012, the latest versions of all Internet Browsers have support for rendering SVG (with varying quality). Older versions, especially (Microsoft Internet Explorer (before IE 9) lack native SVG support but can be made to work with Adobe's free SVG viewer plugin). The SVG fi les can be con verted to JPEG or PNG using Inkscape.

We are also indebted to Matthias Schabel for managing the formal Boost-review of this library, and to all the reviewers - including Guillaume Melquiond, Arnaldur Gylfason, John Phillips, Stephan Tolksdorf and Jeff Garland - for their many helpful comments.

Thanks to Mark Coleman and Georgi Boshnakov for spot test values from Wolfram Mathematica, and of course, to Eric Weisstein for nurturing Wolfram MathWorld, an invaluable resource.

The Skew-normal distribution and Owen's t function were written by Benjamin Sobotta."'

C.1.13 Contributors to Boost Random

The main authors of the Boost Random are Jens Maurer and Steven Watanabe.

The History and Acknowledgements section states:

"In November 1999, Jeet Sukumaran proposed a framework based on virtual functions, and later sketched a template-based approach. Ed Brey pointed out that Microsoft Visual C++ does not support in-class member initializations and suggested the enum workaround. Dave Abrahams highlighted quantization issues.

The first public release of this random number library materialized in March 2000 after extensive discussions on the boost mailing list. Many thanks to Beman Dawes for his original min_rand class, portability fixes, documentation suggestions, and general guidance. Harry Erwin sent a header file which provided additional insight into the requirements. Ed Brey and Beman Dawes wanted an iterator-like interface.

Beman Dawes managed the formal review, during which Matthias Troyer, Csaba Szepesvari, and Thomas Holenstein gave detailed comments. The reviewed version became an official part of

boost on 17 June 2000.

Gary Powell contributed suggestions for code cleanliness. Dave Abrahams and Howard Hinnant suggested to move the basic generator templates from namespace boost::detail to boost::random.

Ed Brey asked to remove superfluous warnings and helped with uint64_t handling. Andreas Scherer tested with MSVC. Matthias Troyer contributed a lagged Fibonacci generator. Michael Stevens found a bug in the copy semantics of normal_distribution and suggested documentation improvements."

C.1.14 Contributors to Boost Odeint

The main authors of the Boost Odeint are Karsten Ahnert and Mario Mulansky.

The History and Acknowledgements section states:

Acknowledgments
Steven Watanabe for managing the Boost review process.
All people who participated in the odeint review process on the Boost mailing list.
Paul Bristow for helping with the documentation.
The Google Summer Of Code (GSOC) program for funding and Andrew Sutton for supervising us during the GSOC and for lots of useful discussions and feedback about many implementation details..
Joachim Faulhaber for motivating us to participate in the Boost review process and many detailed comments about the library.
All users of odeint. They are the main motivation for our efforts.

Contributers
Andreas Angelopoulos implemented the sparse matrix implicit Euler stepper using the MTL4 library.
Rajeev Singh implemented the stiff Van der Pol oscillator example.
Sylwester Arabas improved the documentation.
Denis Demidov provided the adaption to the VexCL and Viennacl libraries.
Christoph Koke provided improved binders.
Lee Hodgkinson provided the black hole example.
Michael Morin fixed several typos in the documentation and the the source code comments.

C.1.15 Contributors to NLOpt

The main author of NLOpt is Steven G. Johnson.

The Acknowledgements section states:

"We are grateful to the many authors who have published useful optimization algorithms implemented in NLopt, especially those who have provided free/open-source implementations of their algorithms.

Please cite these authors if you use their code or the implementation of their algorithm in NLopt. See the documentation for the appropriate citation for each of the algorithms in NLopt âĂŤ please see the Citing NLopt information."

Appendix D

Licenses

D.1 GNU Licenses

D.1.1 GNU General Public License, Version 2

GNU LIBRARY GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1991 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.
[This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary GPL.] Preamble The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.
This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.
When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.
To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.
For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.
Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.
Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.
Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming

the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
âĂća) The modified work must itself be a software library. âĂćb) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change. âĂćc) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License. âĂćd) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful. (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)
These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.
Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.
In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.
Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.
This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.
If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a ”work that uses the Library”. Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.
However, linking a ”work that uses the Library” with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a ”work that uses the library”.

The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

âĂća) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.) âĂćb) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution. âĂćc) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place. âĂćd) Verify that the user has already received a copy of these materials or that you have already sent this user a copy. For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

âĂća) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above. âĂćb) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.
If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.
It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.
This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions

are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).
To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.
one line to give the library's name and an idea of what it does. Copyright (C) year name of author
This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details.
You should have received a copy of the GNU Library General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA. Also add information on how to contact you by electronic and paper mail.
You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:
Yoyodyne, Inc., hereby disclaims all copyright interest in the library 'Frob' (a library for tweaking knobs) written by James Random Hacker.
signature of Ty Coon, 1 April 1990 Ty Coon, President of Vice That's all there is to it!

D.1.2 GNU Library General Public License, Version 2

GNU LIBRARY GENERAL PUBLIC LICENSE
Version 2, June 1991

[This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary GPL.] Preamble The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended

to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.
The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.
Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".
A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.
The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)
"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.
Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.
You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
âĂća) The modified work must itself be a software library. âĂćb) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change. âĂćc) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License. âĂćd) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful. (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)
These requirements apply to the modified work as a whole. If identifiable sections of that work are

not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.
Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.
In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.
Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.
This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.
If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a ”work that uses the Library”. Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.
However, linking a ”work that uses the Library” with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a ”work that uses the library”. The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.
When a ”work that uses the Library” uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.
If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)
Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a ”work that uses the Library” with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer’s own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:
âĂća) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.) âĂćb) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution. âĂćc) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place. âĂćd) Verify that the user has already received a copy of these materials or that you have already sent this user a copy. For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.
It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:
âĂća) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above. âĂćb) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason

(not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.
If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.
It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.
This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and ”any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY ”AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR RE-

DISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).
To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the ”copyright” line and a pointer to where the full notice is found.
one line to give the library’s name and an idea of what it does. Copyright (C) year name of author
This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details.
You should have received a copy of the GNU Library General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA. Also add information on how to contact you by electronic and paper mail.
You should also get your employer (if you work as a programmer) or your school, if any, to sign a ”copyright disclaimer” for the library, if necessary. Here is a sample; alter the names:
Yoyodyne, Inc., hereby disclaims all copyright interest in the library ‘Frob’ (a library for tweaking knobs) written by James Random Hacker.
signature of Ty Coon, 1 April 1990 Ty Coon, President of Vice That’s all there is to it!

D.1.3 GNU Lesser General Public License, Version 3

GNU LESSER GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.
As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.
"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.
An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.
A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".
The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.
The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.
You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.
If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:
a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.
The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:
a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.
You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:
a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
d) Do one of the following:
0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.
e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.
You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:
a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.
b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.
The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.
If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

D.1.4 GNU General Public License, Version 3

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.
To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.
A "covered work" means either the unmodified Program or a work based on the Program.
To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.
To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.
An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.
The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.
A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.
The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.
The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.
The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.
The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.
All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.
You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose

of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.
Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.
No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.
When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.
You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.
You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.
You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:
a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.
A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.
You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:
a) Convey the object code in, or embodied in, a physical product (including a physical distribution

medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.
A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.
A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.
"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.
If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).
The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.
Corresponding Source conveyed, and Installation Information provided, in accord with this section must

be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.
"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.
When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.
Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:
a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.
All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.
If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.
Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.
You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).
However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.
Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.
You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.
Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.
An ”entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.
You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.
A ”contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's ”contributor version”.
A contributor's ”essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, ”control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.
Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.
In the following three paragraphs, a ”patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To ”grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.
If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. ”Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.
A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.
Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.
If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.
Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.
The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.
If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.
Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.
THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY

KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.
IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.
If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.
To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the ”copyright” line and a pointer to where the full notice is found.

<one line to give the program’s name and a brief idea of what it does.> Copyright (C) <year> <name of author>
This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Also add information on how to contact you by electronic and paper mail.
If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:
<program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’. This is free software, and you are welcome to redistribute it under certain conditions; type ‘show c’ for details.
The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, your program’s commands might be different; for a GUI interface, you would

use an ”about box”.
You should also get your employer (if you work as a programmer) or school, if any, to sign a ”copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>
The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>

D.1.5 GNU Free Documentation License, Version 1.3

GNU Free Documentation License
1.3, 3 November 2008

0. PREAMBLE
The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.
This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.
We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS
This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.
A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.
A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.
The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.
The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.
A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise

Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".
Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.
The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.
The "publisher" means any person or entity that distributes copies of the Document to the public.
A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.
The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING
You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.
You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY
If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.
If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.
If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy

(directly or through your agents or retailers) of that edition to the public.
It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS
You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:
A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.
If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.
You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties-for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one. The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS
You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.
The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.
In the combination, you must combine any sections Entitled ”History” in the various original documents, forming one section Entitled ”History”; likewise combine any sections Entitled ”Acknowledgements”, and any sections Entitled ”Dedications”. You must delete all sections Entitled ”Endorsements”.

6. COLLECTIONS OF DOCUMENTS
You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.
You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS
A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an ”aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.
If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION
Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or

disclaimer, the original version will prevail.
If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION
You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.
However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.
Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.
Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE
The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.
Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING
"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.
"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.
"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.
An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.
The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:
Copyright (C) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".
If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ? Texts." line with this:
with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.
If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

D.2 Other Licenses

D.2.1 Mozilla Public License, Version 2.0

Mozilla Public License Version 2.0

1. Definitions
1.1. ”Contributor”
means each individual or legal entity that creates, contributes to the creation of, or owns Covered Software.
1.2. ”Contributor Version”
means the combination of the Contributions of others (if any) used by a Contributor and that particular ContributorâĂŹs Contribution.
1.3. ”Contribution”
means Covered Software of a particular Contributor.
1.4. ”Covered Software”
means Source Code Form to which the initial Contributor has attached the notice in Exhibit A, the Executable Form of such Source Code Form, and Modifications of such Source Code Form, in each case including portions thereof.
1.5. ”Incompatible With Secondary Licenses”
means
a.that the initial Contributor has attached the notice described in Exhibit B to the Covered Software; or
b.that the Covered Software was made available under the terms of version 1.1 or earlier of the License, but not also under the terms of a Secondary License.
1.6. ”Executable Form”
means any form of the work other than Source Code Form.
1.7. ”Larger Work”
means a work that combines Covered Software with other material, in a separate file or files, that is not Covered Software.
1.8. ”License”
means this document.
1.9. ”Licensable”
means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently, any and all of the rights conveyed by this License.
1.10. ”Modifications”
means any of the following:
a.any file in Source Code Form that results from an addition to, deletion from, or modification of the contents of Covered Software; or
b.any new file in Source Code Form that contains any Covered Software.
1.11. ”Patent Claims” of a Contributor
means any patent claim(s), including without limitation, method, process, and apparatus claims, in any patent Licensable by such Contributor that would be infringed, but for the grant of the License, by the making, using, selling, offering for sale, having made, import, or transfer of either its Contributions or its Contributor Version.
1.12. ”Secondary License”
means either the GNU General Public License, Version 2.0, the GNU Lesser General Public License, Version 2.1, the GNU Affero General Public License, Version 3.0, or any later versions of those licenses.
1.13. ”Source Code Form”
means the form of the work preferred for making modifications.
1.14. ”You” (or ”Your”)

means an individual or a legal entity exercising rights under this License. For legal entities, "You" includes any entity that controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants and Conditions
2.1. Grants
Each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:
a.under intellectual property rights (other than patent or trademark) Licensable by such Contributor to use, reproduce, make available, modify, display, perform, distribute, and otherwise exploit its Contributions, either on an unmodified basis, with Modifications, or as part of a Larger Work; and
b.under Patent Claims of such Contributor to make, use, sell, offer for sale, have made, import, and otherwise transfer either its Contributions or its Contributor Version.
2.2. Effective Date
The licenses granted in Section 2.1 with respect to any Contribution become effective for each Contribution on the date the Contributor first distributes such Contribution.
2.3. Limitations on Grant Scope
The licenses granted in this Section 2 are the only rights granted under this License. No additional rights or licenses will be implied from the distribution or licensing of Covered Software under this License. Notwithstanding Section 2.1(b) above, no patent license is granted by a Contributor:
a.for any code that a Contributor has removed from Covered Software; or
b.for infringements caused by: (i) Your and any other third partyâĂŹs modifications of Covered Software, or (ii) the combination of its Contributions with other software (except as part of its Contributor Version); or
c.under Patent Claims infringed by Covered Software in the absence of its Contributions.
This License does not grant any rights in the trademarks, service marks, or logos of any Contributor (except as may be necessary to comply with the notice requirements in Section 3.4).
2.4. Subsequent Licenses
No Contributor makes additional grants as a result of Your choice to distribute the Covered Software under a subsequent version of this License (see Section 10.2) or under the terms of a Secondary License (if permitted under the terms of Section 3.3).
2.5. Representation
Each Contributor represents that the Contributor believes its Contributions are its original creation(s) or it has sufficient rights to grant the rights to its Contributions conveyed by this License.
2.6. Fair Use
This License is not intended to limit any rights You have under applicable copyright doctrines of fair use, fair dealing, or other equivalents.
2.7. Conditions
Sections 3.1, 3.2, 3.3, and 3.4 are conditions of the licenses granted in Section 2.1.

3. Responsibilities
3.1. Distribution of Source Form
All distribution of Covered Software in Source Code Form, including any Modifications that You create or to which You contribute, must be under the terms of this License. You must inform recipients that the Source Code Form of the Covered Software is governed by the terms of this License, and how they can obtain a copy of this License. You may not attempt to alter or restrict the recipientsâĂŹ rights in the Source Code Form.
3.2. Distribution of Executable Form
If You distribute Covered Software in Executable Form then:
a.such Covered Software must also be made available in Source Code Form, as described in Section 3.1, and You must inform recipients of the Executable Form how they can obtain a copy of such Source Code

Form by reasonable means in a timely manner, at a charge no more than the cost of distribution to the recipient; and

b.You may distribute such Executable Form under the terms of this License, or sublicense it under different terms, provided that the license for the Executable Form does not attempt to limit or alter the recipientsâĂŹ rights in the Source Code Form under this License.

3.3. Distribution of a Larger Work

You may create and distribute a Larger Work under terms of Your choice, provided that You also comply with the requirements of this License for the Covered Software. If the Larger Work is a combination of Covered Software with a work governed by one or more Secondary Licenses, and the Covered Software is not Incompatible With Secondary Licenses, this License permits You to additionally distribute such Covered Software under the terms of such Secondary License(s), so that the recipient of the Larger Work may, at their option, further distribute the Covered Software under the terms of either this License or such Secondary License(s).

3.4. Notices

You may not remove or alter the substance of any license notices (including copyright notices, patent notices, disclaimers of warranty, or limitations of liability) contained within the Source Code Form of the Covered Software, except that You may alter any license notices to the extent required to remedy known factual inaccuracies.

3.5. Application of Additional Terms

You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, You may do so only on Your own behalf, and not on behalf of any Contributor. You must make it absolutely clear that any such warranty, support, indemnity, or liability obligation is offered by You alone, and You hereby agree to indemnify every Contributor for any liability incurred by such Contributor as a result of warranty, support, indemnity or liability terms You offer. You may include additional disclaimers of warranty and limitations of liability specific to any jurisdiction.

4. Inability to Comply Due to Statute or Regulation

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Software due to statute, judicial order, or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be placed in a text file included with all distributions of the Covered Software under this License. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

5. Termination

5.1. The rights granted under this License will terminate automatically if You fail to comply with any of its terms. However, if You become compliant, then the rights granted under this License from a particular Contributor are reinstated (a) provisionally, unless and until such Contributor explicitly and finally terminates Your grants, and (b) on an ongoing basis, if such Contributor fails to notify You of the non-compliance by some reasonable means prior to 60 days after You have come back into compliance. Moreover, Your grants from a particular Contributor are reinstated on an ongoing basis if such Contributor notifies You of the non-compliance by some reasonable means, this is the first time You have received notice of non-compliance with this License from such Contributor, and You become compliant prior to 30 days after Your receipt of the notice.

5.2. If You initiate litigation against any entity by asserting a patent infringement claim (excluding declaratory judgment actions, counter-claims, and cross-claims) alleging that a Contributor Version directly or indirectly infringes any patent, then the rights granted to You by any and all Contributors for the Covered Software under Section 2.1 of this License shall terminate.

5.3. In the event of termination under Sections 5.1 or 5.2 above, all end user license agreements (excluding distributors and resellers) which have been validly granted by You or Your distributors under this License prior to termination shall survive termination.

6. Disclaimer of Warranty
Covered Software is provided under this License on an ”as is” basis, without warranty of any kind, either expressed, implied, or statutory, including, without limitation, warranties that the Covered Software is free of defects, merchantable, fit for a particular purpose or non-infringing. The entire risk as to the quality and performance of the Covered Software is with You. Should any Covered Software prove defective in any respect, You (not any Contributor) assume the cost of any necessary servicing, repair, or correction. This disclaimer of warranty constitutes an essential part of this License. No use of any Covered Software is authorized under this License except under this disclaimer.

7. Limitation of Liability
Under no circumstances and under no legal theory, whether tort (including negligence), contract, or otherwise, shall any Contributor, or anyone who distributes Covered Software as permitted above, be liable to You for any direct, indirect, special, incidental, or consequential damages of any character including, without limitation, damages for lost profits, loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses, even if such party shall have been informed of the possibility of such damages. This limitation of liability shall not apply to liability for death or personal injury resulting from such partyâĂŹs negligence to the extent applicable law prohibits such limitation. Some jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, so this exclusion and limitation may not apply to You.

8. Litigation
Any litigation relating to this License may be brought only in the courts of a jurisdiction where the defendant maintains its principal place of business and such litigation shall be governed by laws of that jurisdiction, without reference to its conflict-of-law provisions. Nothing in this Section shall prevent a partyâĂŹs ability to bring cross-claims or counter-claims.

9. Miscellaneous
This License represents the complete agreement concerning the subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not be used to construe this License against a Contributor.

10. Versions of the License
10.1. New Versions
Mozilla Foundation is the license steward. Except as provided in Section 10.3, no one other than the license steward has the right to modify or publish new versions of this License. Each version will be given a distinguishing version number.
10.2. Effect of New Versions
You may distribute the Covered Software under the terms of the version of the License under which You originally received the Covered Software, or under the terms of any subsequent version published by the license steward.
10.3. Modified Versions
If you create software not governed by this License, and you want to create a new license for such software, you may create and use a modified version of this License if you rename the license and remove any references to the name of the license steward (except to note that such modified license differs from this License).
10.4. Distributing Source Code Form that is Incompatible With Secondary Licenses
If You choose to distribute Source Code Form that is Incompatible With Secondary Licenses under the terms of this version of the License, the notice described in Exhibit B of this License must be attached.

Exhibit A - Source Code Form License Notice
This Source Code Form is subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at http://mozilla.org/MPL/2.0/.

If it is not possible or desirable to put the notice in a particular file, then You may include the notice in a location (such as a LICENSE file in a relevant directory) where a recipient would be likely to look for such a notice.
You may add additional accurate notices of copyright ownership.

Exhibit B - ”Incompatible With Secondary Licenses” Notice
This Source Code Form is ”Incompatible With Secondary Licenses”, as defined by the Mozilla Public License, v. 2.0.

D.2.2 Boost Software License, Version 1.0

Boost Software License Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

D.2.3 MIT License

MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the ”Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED ”AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Part VII

Back Matter

Bibliography

Abramowitz, M., & Stegun., I.A. 1970. *Handbook of Mathematical Functions*. New York: Dekker. Available as http://www.math.sfu.ca/~cbm/aands/.

Adler, Joseph. 2012. *R in a Nutshell*. 2nd edn. O'Reilly Media. Online resource: http://shop.oreilly.com/product/0636920022008.do.

Ahnert, Karsten, & Mulansky, Mario. 2013 (June). *Boost Odeint Library*. http://www.boost.org/libs/odeint/. Documentation available as http://www.boost.org/doc/libs/1_54_0/libs/numeric/odeint/doc/html/index.html.

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Croz, J. Du, Greenbaum, A., Hammarling, S., McKenney, A., & Sorensen, D. 1999. *LAPACK Users' Guide*. 3rd edn. Society for Industrial and Applied Mathematics (SIAM). LAPACK Homepage: http://www.netlib.org/lapack.

Arndt, J. 2011. *Matters Computational - Ideas, Algorithms, Source Code*. Springer. Available from http://www.jjj.de/fxt/#fxtbook.

Barker, V.A., Blackford, S., Dongarra, J., Croz, J. Du, Hammarling, S., Marinova, M., Wasniewski, J., & Yalamov, P. 2001. *LAPACK95 Users' Guide*. 1st edn. Society for Industrial and Applied Mathematics (SIAM). LAPACK Homepage: http://www.netlib.org/lapack.

Benton, Denise, & Krishnamoorthy, K. 2003. Computing discrete mixtures of continuous distributions: noncentral chisquare, noncentral t and the distribution of the square of the sample multiple correlation coefficient. *Computational Statistics & Data Analysis*, **14**, 249–267. Available as http://www.ucs.louisiana.edu/~kxk4695/CSDA-03.pdf.

Bernstein, Dennis S. 2009. *Matrix mathematics: theory, facts, and formulas*. 2nd edn. Princeton University Press.

Birgin, E. G., & Martínez, J. M. 2008. Improving ultimate convergence of an augmented Lagrangian method. *Optimization Methods and Software*, **23**(2), 177–195.

Björck, Å., & Hammarling, S. 1983. A Schur method for the square root of a matrix. *Linear Algebra Appl.*, **52/53**, 127–140.

Blomquist, F. 2005. *Automatische a priori Fehlerabschätzungen zur Entwicklung optimaler Algorithmen und Intervallfunktionen in C-XSC (in German)*. Available from http://www.math.uni-wuppertal.de/~xsc/literatur/a_priori.pdf.

Blomquist, F., Hofschuster, W., & Krämer, W. 2008a. A Modified Staggered Correction Arithmetic with Enhanced Accuracy and Very Wide Exponent Range. *In:* Cuyt, Annie, Krämer, Walter, Luther, Wolfram, & Markstein, Peter (eds), *Numerical Validation in Current Hardware*

Architectures. Dagstuhl Seminar Proceedings, no. 08021. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. Available from http://drops.dagstuhl.de/opus/volltexte/2008/1445.

Blomquist, F., Hofschuster, W., & Krämer, W. 2008b. *Real and Complex Staggered (Interval) Arithmetics with Wide Exponent Range (in German)*. Preprint BUGHW-WRSWT 2008/1, Universität Wuppertal, 2008, Available at http://www2.math.uni-wuppertal.de/~xsc/preprints/prep_08_1.pdf.

Blomquist, F., Hofschuster, W., & Krämer, W. 2012. *Umfangreiche C-XSC-Langzahlpakete für beliebig genaue reelle und komplexe Intervallrechnung*. Preprint 2012/2, Universität Wuppertal, 2012 (in german). Online resource: http://www2.math.uni-wuppertal.de/~xsc/xsc/cxsc_software.html#mpfr-mpfi. The manual is available from http://www2.math.uni-wuppertal.de/~xsc/preprints/prep_12_2.pdf.

Box, M. J. 1965. A new method of constrained optimization and a comparison with other methods. *Computer J*, **8**(1), 42–52.

Brent, R. 1972. *Algorithms for Minimization without Derivatives*. 1st edn. Prentice-Hall (Reprinted by Dover, 2002.).

Brent, R., & Zimmermann, P. 2010. *Modern Computer Arithmetic*. 1st edn. Cambridge University Press. A preliminary version (0.5.9) of the book is available from http://www.loria.fr/~zimmerma/mca/mca-cup-0.5.9.pdf.

Brent, R. P. 1992a. On the Periods of Generalized Fibonacci Recurrences. *Computer Sciences Laboratory Australian National University*.

Brent, R. P. 1992b. Uniform random number generators for supercomputers. *Proc. of Fifth Australian Supercomputer Conference, Melbourne*, 704–706.

Bristow, Paul A., Holin, Hubert, Kormanyos, Christopher, Lalande, Bruno, Maddock, John, RÃěde, Johan, Sobotta, Benjamin, Sewani, Gautam, van den Berg, Thijs, Walker, Daryle, , & Zhang, Xiaogang. 2013 (June). *Boost Math Toolkit Library*. http://www.boost.org/libs/math/. Documentation available as http://www.boost.org/doc/libs/1_54_0/libs/math/doc/html/index.html.

Carlson, B. C., & Gustafson, J. L. 1994. Asymptotic Approximations for Symmetric Elliptic Integrals. *Siam Journal on Mathematical Analysis*, **25**.

Carlson, B.C. 1995. Numerical computation of real or complex elliptic integrals. *Numerical Algorithms*, **10**(1), 13–26.

Casagrande, J. T., Pike, M. C., & Smith, P. G. 1978. The Power Function of the "Exact" Test for Comparing Two Binomial Distributions. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, **27**(2), 176–180. Article Stable URL: http://www.jstor.org/stable/2346945.

Chang, Winston. 2012. *R Graphics Cookbook - Practical Recipes for Visualizing Data*. 1st edn. O'Reilly Media. Online resource: http://shop.oreilly.com/product/0636920023135.do.

Cheney, Ward, & Kincaid, David. 2008. *Numerical Mathematics and Computing*. 6th edn. Thomson Brooks/Cole.

Chernick, Michael R. 2008. *Bootstrap methods : a guide for practitioners and researchers*. 2nd edn. John Wiley & Sons.

Conlon, M., & Thomas, R. G. 1993. Algorithm AS 280: The Power Function for Fisher's Exact Test. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, **42**(1), 258–260. Article Stable URL: http://www.jstor.org/stable/2347431.

Conn, A. R., Gould, N. I. M., & Toint, Ph. L. 1991. A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM J. Numer. Anal.*, **28**(2), 545–572.

Corless, R. M., Gonnet, G. H., Hare, D. E. G., Jeffrey, D. J., & Knuth, D. E. 1996. On the Lambert W Function. *Pages 329–359 of: Advances in Computational Mathematics*.

Cuyt, A.A.M., Petersen, V., Verdonk, B., Waadeland, H., & Jones, W.B. 2008. *Handbook of Continued Fractions for Special Functions*. Springer.

Davies, Ph., & Higham, N. J. 2003. A Schur-Parlett algorithm for computing matrix functions. *SIAM J. Matrix Anal. Applic.*, **25**, 464–485.

Dembo, R. S., & Steihaug, T. 1982. Truncated Newton algorithms for large-scale optimization. *Math. Programming*, **26**, 190–212.

DiDonato, A.R., & Morris, A.H. 1986. Computation of the Incomplete Gamma Function Ratios and their Inverse. *ACM TOMS*, **12**, 377–393. Fortran source: ACM TOMS 13 (1987) pp. 318-319; available from http://netlib.org/toms/654.

DiDonato, Armido R., & Morris, Jr., Alfred H. 1992. Algorithm 708: Significant digit computation of the incomplete beta function ratios. *ACM Trans. Math. Softw.*, **18**(3), 360–373.

Ding, C. G. 1992. Algorithm AS 275: Computing the non-central χ^2 distribution function. *Applied Statistics*, **41**, 478âĂŞ–482.

Dongarra, J.J., DuCroz, J., Hammarling, S., & Hanson, R. 1988. An Extended Set of Fortran Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, **14**, 1–32. BLAS Homepage: http://www.netlib.org/blas/.

Dongarra, J.J., Duff, I., DuCroz, J., & Hammarling, S. 1990. A Set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, **16**, 1–28. BLAS Homepage: http://www.netlib.org/blas/.

Dormand, J.R., & Prince, P.J. 1980. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, **6**(1), 19 – 26.

Eastlake, D., Crocker, S., & Schiller, J. 1994. Randomness Recommendations for Security. *Network Working Group, RFC 1750*.

Edelman, Alan, & Murakami, H. 1995. Polynomial Roots from Companion Matrix Eigenvalues. *Math. Comput.*, **64**(210), 763–776. Available as http://www.ams.org/journals/mcom/1995-64-210/S0025-5718-1995-1262279-2/S0025-5718-1995-1262279-2.pdf.

Enge, Andreas, Gastineau, Mickaël, Théveny, Philippe, & Zimmermann, Paul. 2012 (July). *mpc — A library for multiprecision complex arithmetic with exact rounding*. 1.0 edn. INRIA. http://mpc.multiprecision.org/. The MPC manual is available from http://www.multiprecision.org/mpc/download/mpc-1.0.1.pdf.

Fousse, Laurent, Hanrot, Guillaume, Lefèvre, Vincent, Pélissier, Patrick, & Zimmermann, Paul. 2007. MPFR: A Multiple-Precision Binary Floating-Point Library with Correct Rounding. *ACM Transactions on Mathematical Software*, **33**(2), 13:1–13:15. Online resource: http://www.mpfr.org/. The MPFR manual is available from http://www.mpfr.org/mpfr-current/mpfr.pdf, and a description of the algorithms from http://www.mpfr.org/algorithms.pdf.

Gablonsky, J. M., & Kelley, C. T. 2001. A locally-biased form of the DIRECT algorithm. *J. Global Optimization*, **21**(1), 27–37.

Gil, Amparo, Segura, Javier, & Temme, Nico M. 2007. *Numerical methods for special functions*. SIAM.

Gil, Amparo, Segura, Javier, & Temme, NicoM. 2011. Basic Methods for Computing Special Functions. *Pages 67–121 of:* Simos, Theodore E. (ed), *Recent Advances in Computational and Applied Mathematics*. Springer Netherlands. Preprint available from http://www.researchgate.net/publication/229032140_Basic_Methods_for_Computing_Special_Functions/file/d912f5093e6adedc38.pdf.

Gleser, L. J. 1976. A canonical representation for the noncentral Wishart distribution useful for simulation. *Journal of the American Statistical Association*, 690–695.

Golhar, M. B. 1972. The errors of first and second kinds in Welch-Aspin's solution of the Behrens-Fisher problem. *Journal of Statistical Computation and Simulation*, **1**(3), 209–224.

Golub, Gene H., & Van Loan, Charles F. 1996. *Matrix Computations*. 3rd edn. The John Hopkins University Press.

Granlund, Torbjörn, & the GMP development team. 2013. *GNU MP: The GNU Multiple Precision Arithmetic Library*. 5.1.2 edn. Online resource: http://gmplib.org/. The GMP manual is available from http://gmplib.org/gmp-man-5.1.2.pdf.

Grantham, Jon. 2001. Frobenius pseudoprimes. *Mathematics of Computation*, **70**, 873–891. Available at http://www.ams.org/journals/mcom/2001-70-234/S0025-5718-00-01197-2/home.html.

Gudmundsson, S. 1998. *Parallel Global Optimization*. M.Phil. thesis, Technical University of Denmark.

Guennebaud, Gaël, Jacob, Benoît, *et al.* 2010. *Eigen v3, a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.* Online resource: http://eigen.tuxfamily.org.

Guenther, William C. 1974. Sample Size Formulas for Some Binomial Type Problems. *Technometrics*, **16**, 465–467.

Hairer, E., & Wanner, G. 2010. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. 2nd edn. Springer, Berlin.

Hairer, E., Wanner, G., & Lubich, C. 2006. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. 2nd edn. Springer-Verlag Gmbh.

Hairer, E., Nørsett, S. P., & Wanner, G. 2009. *Solving Ordinary Differential Equations I: Nonstiff Problems*. 2nd edn. Springer, Berlin.

Hammer, R., Hocks, M., Ratz, D., & Kulisch, U. 1995. *C++ Toolbox for Verified Scientific Computing I: Basic Numerical Problems Theory, Algorithms, and Programs.* Secaucus, NJ, USA: Springer-Verlag New York, Inc. Online resource: http://link.springer.com/book/10.1007/978-3-642-79651-7/page/1.

Hardy D.W., Richman F., Walker C.L. 2009. *Applied Algebra: Codes, Ciphers and Discrete Algorithms (Discrete Mathematics and Its Applications).* 2nd edn. Chapman and Hall/CRC.

Harkness, W. L., & Katz, L. 1964. Comparison of the Power Functions for the Test of Independence in 2 ÃŮ 2 Contingency Tables. *The Annals of Mathematical Statistics*, **35**(3), 1115–1127. Article Stable URL: http://www.jstor.org/stable/2238241.

Hayes, Brian. 2003. A Lucid Interval. *American Scientist*, **91**(6), 484–488. Available at: http://www.cs.utep.edu/interval-comp/hayes.pdf.

Hellekalek, P. 1995. Inversive pseudorandom number generators: concepts, results and links. *Pages 255–262 of:* Alexopoulos, C., Kang, K., Lilegdon, W.R., , & Goldsman, D. (eds), *Proceedings of the 1995 Winter Simulation Conference.* Available from ftp://random.mat.sbg.ac.at/pub/data/wsc95.ps.

Hendrix, E. M. T., Ortigosa, P. M., , & García, I. 2001. On success rates for controlled random search. *J. Global Optim. 21*, **21**, 239–263.

Higham, N. J. 1987. Computing real square roots of a real matrix. *Linear Algebra Appl.*, **88/89**, 405–430.

Higham, N. J. 2005. The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Applic.*, **26**, 1179–1193.

Higham, N. J. 2008. *Functions of Matrices: Theory and Computation.* 1st edn. Society for Industrial and Applied Mathematics (SIAM). ISBN 978-0-898716-46-7.

Higham, N. J., & Lin, L. 2011. A Schur-Padé algorithm for fractional powers of a matrix. *SIAM J. Matrix Anal. Applic.*, **32**(3), 1056–1078.

Hofschuster, W. 2000. *Zur Berechnung von Funktionswerteinschließungen bei speziellen Funktionen der mathematischen Physik.* Ph.D. thesis, Universität Karlsruhe. Available online at: http://digbib.ubka.uni-karlsruhe.de/volltexte/documents/1493.

Hofschuster, W., & Krämer, W. 2004. C-XSC 2.0: A C++ Library for Extended Scientific Computing. *Pages 15–35 of:* Alt, R., Frommer, A., Kearfott, R.B., & Luther, W. (eds), *Numerical Software with Result Verification, Lecture Notes in Computer Science.* Springer-Verlag, Heidelberg. Online resource: http://www2.math.uni-wuppertal.de/~xsc/xsc-sprachen.html. A preprint is available from http://www2.math.uni-wuppertal.de/~xsc/preprints/prep_03_5.pdf.

Holoborodko, P. 2008-2012. *MPFR C++: A high-performance C++ interface for the MPFR library.* Online resource: http://www.holoborodko.com/pavel/mpfr/.

James, F. 1994. RANLUX: A Fortran implementation of the high-quality pseudorandom number generator of Luescher. *Computer Physics Communications*, **79**, 111–114.

Janert, P.K. 2010. *Gnuplot in Action - Understanding Data with Graphs.* 1st edn. Manning Publications.

Johnson, N. L., Kotz, S., & Balakrishnan, N. 1994.. *Continuous Univariate Distributions, Volume 1*. 2nd edn. Wiley & Sons, Inc., New York-London-Sydney-Toronto.

Johnson, N. L., Kotz, S., & Balakrishnan, N. 1995.. *Continuous Univariate Distributions, Volume 2*. 2nd edn. Wiley & Sons, Inc., New York-London-Sydney-Toronto.

Johnson, Steven G. 2012. *NLOPT v2.3: a free/open-source library for nonlinear optimization.* Online resource: http://ab-initio.mit.edu/wiki/index.php/NLopt.

Jones, D. R., Perttunen, C. D., & Stuckmann, B. E. 1993. Lipschitzian optimization without the lipschitz constant. *J. Optimization Theory and Applications*, **79**, 157.

Joye, M., & Quisquater, J.-J. 1996. Efficient computation of full Lucas sequences. *Electronics Letters*, **32(6)**, 537–538. Available at http://joye.site88.net/papers/JQ96lucas.pdf.

Kaelo, P., & Ali, M. M. 2006. Some variants of the controlled random search algorithm for global optimization. *J. Optim. Theory Appl.*, **130**(2), 27–37.

Kahan, W. 1997. *Lecture Notes on the Status of IEEE Standard 754 for Binary Floating-Point Arithmetic*. Tech. rept. Elect. Eng. & Computer Science, University of California, Berkeley, CA. Available as http://www.cs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF.

Kammler, D. W. 2008. *A First Course in Fourier Analysis*. 1st edn. Cambridge University Press.

Klatte, R., Kulisch, U., Neaga, M., Ratz, D., & Ullrich, Ch. 1991. *PASCAL-XSC Language Reference with Examples*. Springer Berlin Heidelberg. Available at: http://www2.math.uni-wuppertal.de/~xsc/literatur/PXSCENGL.pdf.

Knüsel, L., & Michalk, J. 1987. Asymptotic expansion of the power function of the two-sample binomial test with and without randomization. *Metrika*, **34**(1), 31–44. Available at: http://rd.springer.com/article/10.1007%2FBF02613128#.

Knuth, Donald E. 1981. *Art of Computer Programming, Volume 2: Seminumerical Algorithms*. 2nd edn. Addison-Wesley Professional. Online resource: http://www-cs-faculty.stanford.edu/~knuth/taocp.html.

Knuth, Donald E. 1997. *Art of Computer Programming, Volume 1: Fundamental Algorithms, Volume 2: Seminumerical Algorithms*. 3rd edn. Addison-Wesley Professional. Online resource: http://www-cs-faculty.stanford.edu/~knuth/taocp.html.

Kraft, D. 1988. *A software package for sequential quadratic programming*. Technical Report DFVLR-FB 88-28. Institut für Dynamik der Flugsysteme, Oberpfaffenhofen.

Kraft, D. 1994. Algorithm 733: TOMP-Fortran modules for optimal control calculations. *ACM Transactions on Mathematical Software*, **20**(3), 262–281.

Krah, Stefan. 2012. *mpdecimal: a package for correctly-rounded arbitrary precision decimal floating point arithmetic*. 2.3 edn. Online resource: http://www.bytereef.org/mpdecimal/index.html.

Krämer, W., Kulisch, U., & Lohner, R. 1994. *Numerical Toolbox for Verified Computing II: Advanced Numerical Problems*. Draft Manuscript. Preprint (1994) available from http://www2.math.uni-wuppertal.de/~xsc/literatur/tb2.pdf.

Krämer, W., Kulisch, U., & Lohner, R. 2006. *Numerical Toolbox for Verified Computing II: Advanced Numerical Problems*. SpringerVerlag. Pascal Version Preprint (1994) available from http://www2.math.uni-wuppertal.de/~xsc/literatur/tb2.pdf.

Krämer, W., Zimmer, M., & Hofschuster, W. 2012. Using C-XSC for High Performance Verified Computing. *Pages 168–178 of: Applied Parallel and Scientific Computing*. Lecture Notes in Computer Science, vol. 7134. Springer Berlin Heidelberg.

Kreibich, Jay A. 2010. *Using SQLite*. O'Reilly Media. Online resource: http://shop.oreilly.com/product/9780596521196.do.

Kreutzer, W. 1986. *System Simulation: Programming Styles and Languages (International Computer Science Series)*. 1st edn. Addison-Wesley.

Kucherenko, S., & Sytsko, Y. 2005. Application of deterministic low-discrepancy sequences in global optimization. *Computational Optimization and Applications*, **30**, 297–318.

Lange, Kenneth. 2010. *Numerical Analysis for Statisticians*. 2nd edn. Springer Science+Business Media.

Lawson, C., Hanson, R., Kincaid, D., & Krogh, F. 1979. Linear Algebra Subprograms for Fortran Usage. *ACM Transactions on Mathematical Software*, **5**, 308–325. BLAS Homepage: http://www.netlib.org/blas/.

L'Ecuyer, P. 1988. Efficient and Portable Combined Random Number Generators. *Communications of the ACM*, **31**(6), 742–749.

L'Ecuyer, P. 1996. Maximally Equidistributed Combined Tausworthe Generators. *Mathematics of Computation*, **65**(213), 203–213.

Leimkuhler, B., & Reich, S. 2005. *Simulating Hamiltonian Dynamics*. 1st edn. Cambridge University Press.

Lewis, P.A., Goodman, A.S., & Miller, J.M. 1969. A pseudo-random number generator for the System/360. *IBM Systems Journal*, **8**(2), 136–146.

Ling, Robert F., & Pratt, John W. 1984. The accuracy of Peizer approximations to the hypergeometric distribution, with comparisons to some other approximations. *JASA. Journal of the American Statistical Association*, **79**, 49–60.

Liu, D. C., & Nocedal, J. 1989. On the limited memory BFGS method for large scale optimization. *Math. Programming*, **45**, 503–528.

Luescher, M. 1994. A portable high-quality random number generator for lattice field theory calculations. *Computer Physics Communications*, **79**, 100–110.

Luschny, Peter. 2012. *Approximation Formulas for the Factorial Function*. Web. Accessed June 3, 2012. Available at http://www.luschny.de/math/factorial/approx/SimpleCases.html.

Maddock, John, & Kormanyos, Christopher. 2013 (June). *Boost Multiprecision Library*. http://www.boost.org/libs/multiprecision/. Documentation available as http://www.boost.org/doc/libs/1_54_0/libs/multiprecision/doc/html/index.html.

Madsen, K., Zertchaninov, S., & Zilinskas, A. 1998. *Global Optimization using Branch-and-Bound*. Unpublished Manuscript.

Matsumoto, M., & Nishimura, T. 1998. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation: Special Issue on Uniform Random Number Generation*, **8**(1), 3–30. Available as http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/ARTICLES/mt.pdf.

Maurer, Jens, & Watanabe, Steven. 2013 (June). *Boost Random Number Library*. http://www.boost.org/libs/random/. Documentation available as http://www.boost.org/doc/libs/1_54_0/doc/html/boost_random.html.

McKinney, Wes. 2012. *Python for Data Analysis*. O'Reilly Media. Online resource: http://shop.oreilly.com/product/0636920023784.do.

McLachlan, R. I. 1995. On the numerical integration of ordinary differential equations by symmetric composition methods. *SIAM J. Sci. Comput.*, **16**(1), 151–168.

Moler, C., & Stewart, G. 1973. An Algorithm for Generalized Matrix Eigenvalue Problems. *SIAM J. Numer. Anal.*, **10**(2), 241–256.

Monahan, John F. 2011. *Numerical methods of statistics*. 2nd edn. Cambridge University Press.

Moore R. E., Cloud M. J., Kearfott R. B. 2009. *Introduction to Interval Analysis*. Philadelphia: Society for Industrial and Applied Mathematics.

Moré, J. J., Garbow, B. S., & Hillstrom, K. E. 1980. *User Guide for MINPACK-1, Argonne National Laboratory Report ANL-80-74, Argonne, Ill.* Available as http://www.mcs.anl.gov/~more/ANL8074a.pdf (Chapter 1-3) and http://www.mcs.anl.gov/~more/ANL8074b.pdf (Chapter 4).

Nelder, J. A., & Mead, R. 1965. A simplex method for function minimization. *The Computer Journal*, **7**, 308–313.

Nocedal, J. 1980. Updating quasi-Newton matrices with limited storage. *Math. Comput.*, **35**, 773–782.

Olver, F.W.J., Lozier, D.W., Boisvert, R.F., & Clark, C.W. 2010. *NIST Handbook of Mathematical Functions*. 1 edn. Cambridge. Online resource: NIST Digital Library of Mathematical Functions http://dlmf.nist.gov/.

Ong, S. H., & Lee, P. A. 1979. The Non-central Negative Binomial Distribution. *Biometrical Journal. Journal of Mathematical Methods in Biosciences. [Continues: Biometrische Zeitschrift. Zeitschrift fÃijr mathematische Methoden in den Biowissenschaften]*, **21**, 611–628.

Owen, D. B. 1956. Tables for computing bivariate normal probabilities. *Ann. Math. Statist.*, **27**, 1075–1090. Available as http://projecteuclid.org/DPubS/Repository/1.0/Disseminate?view=body&id=pdf_1&handle=euclid.aoms/1177728074.

Park, S. K., & Miller, K. W. 1988. Random Number Generators: Good ones are hard to find. *Communications of the ACM*, **31**(10), 1192–1201.

Patefield, M., & Tand, D. 2000. Fast and accurate Calculation of OwenâĂŹs T-Function. *Journal of Statistical Software*. Available as http://www.jstatsoft.org/v05/i05/paper.

Peizer, David B., & Pratt, John W. 1968. A Normal Approximation for Binomial, F, Beta, and Other Common, Related Tail Probabilities. I (Ref: P1457-1483). *Journal of the American Statistical Association*, **63**, 1416–1456.

Pikovsky, A., Rosemblum, M., & Kurths, J. 2001. *Synchronization: A Universal Concept in Nonlinear Sciences*. 1st edn. Cambridge University Press.

Powell, M. J. D. 1994. A direct search optimization method that models the objective and constraint functions by linear interpolation. *Pages 51–67 of:* Gomez, S., & Hennart, J.-P. (eds), *Advances in Optimization and Numerical Analysis*. Kluwer Academic: Dordrecht.

Powell, M. J. D. 1998. Direct search algorithms for optimization calculations. *Acta Numerica*, **7**, 287–336.

Powell, M. J. D. 2004. The NEWUOA software for unconstrained optimization without derivatives. *Pages 51–67 of:* Pardalos, P., & Pillo, G. (eds), *Proc. 40th Workshop on Large Scale Nonlinear Optimization (Erice, Italy*.

Powell, M. J. D. 2009. *The BOBYQA algorithm for bound constrained optimization without derivatives*. technical report NA2009/06. Department of Applied Mathematics and Theoretical Physics, Cambridge England.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., & Flannery, B.P. 2007. *Numerical Recipes: The Art of Scientific Computing*. 3rd edn. Cambridge University Press. Available from http://www.nr.com/.

Price, W. L. 1978. A controlled random search procedure for global optimization. *Pages 71–84 of:* Dixon, L. C. W., & Szego, G. P. (eds), *Towards Global Optimization 2*. North-Holland Press, Amsterdam.

Price, W. L. 1983. Global optimization by controlled random search. *J. Optim. Theory Appl.*, **40**(3), 333–348.

Pugh, G.R. 2004. An Analysis of the Lanczos Gamma Approximation. *PhD thesis, The University of British Columbia*. Available as http://laplace.physics.ubc.ca/ThesesOthers/Phd/pugh.pdf.

Revol, Nathalie, & Rouillier, Fabrice. 2005. Motivations for an arbitrary precision interval arithmetic and the MPFI library. *Reliable Computing*, **11**(4), 275–290. Online resource: https://gforge.inria.fr/projects/mpfi/.

Richardson, J. A., & Kuester, J. L. 1973. The complex method for constrained optimization. *Commun. ACM*, **16**(8), 487–489.

Rinne, H. 2008. *Taschenbuch der Statistik*. 4 edn. Frankfurt, M. : Deutsch.

Rinnooy Kan, A. H. G., & Timmer, G. T. 1987a. Stochastic global optimization methods, part I: clustering methods. *Mathematical Programming*, **39**, 27–56.

Rinnooy Kan, A. H. G., & Timmer, G. T. 1987b. Stochastic global optimization methods, part II: multilevel methods. *Mathematical Programming*, **39**, 57–78.

Rowan, T. 1990. *Functional Stability Analysis of Numerical Algorithms*. Ph.D. thesis, Department of Computer Sciences, University of Texas at Austin.

Rump, S.M. 1999. INTLAB - INTerval LABoratory. *In Tibor Csendes, editor, Developments in Reliable Computing. Kluwer Academic Publishers, Dordrecht*, 77–104. Online resource: http://www.ti3.tu-harburg.de/~rump/intlab/.

Runarsson, T. Ph., & Yao, X. 2000. Stochastic ranking for constrained evolutionary optimization. *IEEE Trans. Evolutionary Computation*, **4**(3), 284–294.

Runarsson, T. Ph., & Yao, X. 2005. Search biases in constrained evolutionary optimization. *IEEE Trans. on Systems, Man, and Cybernetics Part C: Applications and Reviews*, **35**(2), 233–243.

Seber, G.A.F. 2008. *A Matrix Handbok for for Statisticians*. 1st edn. John Wiley & Sons, Inc.

Shoup, Victor. 2009. *A Computational Introduction to Number Theory and Algebra*. 2nd edn. Cambridge University Press. Available as http://shoup.net/ntb/ntb-v2.pdf.

Svanberg, K. 2002. A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM J. Optim.*, **12**(2), 487–489.

Temme, N. M. 1979. The asymptotic expansion of the incomplete gamma functions. *SIAM J. Math. Anal.*, **10**, 757–âĂŞ766.

Temme, N. M. 1994. A Set of Algorithms for the Incomplete Gamma Functions. *Probability in the Engineering and Informational Sciences*, **8**(4), 291–307.

Temme, Nico M. 1996. *Special Functions: An Introduction to the Classical Functions of Mathematical Physics*. John Wiley & Sons, Inc. Online resource: http://onlinelibrary.wiley.com/book/10.1002/9781118032572.

Tiwari, Ram C., & Yang, Jie. 1997. Algorithm AS 318: An Efficient Recursive Algorithm for Computing the Distribution Function and Non-centrality Parameter of the Non-central F-distribution. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, **46**(3), 408–413.

Tretter, M. J., & Walster, G. W. 1979. Continued Fractions for the Incomplete Beta Function: Additions and Corrections. *Ann. Stat.*, **7**(2), 462–465. Available at http://projecteuclid.org/euclid.aos/1176344629.

Upton, Graham J. G. 1982. A Comparison of Alternative Tests for the 2 ÃŮ 2 Comparative Trial. *Journal of the Royal Statistical Society. Series A (General)*, **145**(1), 86–105. Article Stable URL: http://www.jstor.org/stable/2981423.

Van Hauwermeiren, M., & Vose, D. 2009. *A Compendium of Distributions. [ebook]*. Vose Software, Ghent, Belgium. Available from http://www.vosesoftware.com.

Verzani, John. 2011. *Getting Started with RStudio - An Integrated Development Environment for R*. 1st edn. O'Reilly Media. Online resource: http://shop.oreilly.com/product/0636920021278.do.

Vlcek, J., & Luksan, L. 2006. Shifted limited-memory variable metric methods for large-scale unconstrained minimization. *J. Computational Appl. Math.*, **186**, 365–390.

Walck, C. 2007. *Handbook on Statistical Distributions for experimentalists*. University of Stockholm, Internal Report SUF-PFY/96–01. Available as http://www.stat.rice.edu/~dobelman/textfiles/DistributionsHandbook.pdf.

Wang, S., & Gray, H.L. 1993. Approximating tail probabilities of noncentral distributions. *Computational Statistics & Data Analysis*, **15**(3), 343 – 352. Available at: http://www.sciencedirect.com/science/article/pii/016794739390261Q.

Wedner, Stefan. 2000. *Verifizierte Bestimmung singulärer Integrale - Quadratur und Kubatur*. Ph.D. thesis, Universität Karlsruhe. Available online at: http://digbib.ubka.uni-karlsruhe.de/volltexte/documents/3175.

Wikipedia contributors. 2013. *Pareto distribution — Wikipedia, The Free Encyclopedia*. Available at http://en.wikipedia.org/w/index.php?title=Pareto_distribution&oldid=559561732.

Wilkening, J. 2008. *gmpfrxx : A C++ interface for GMP and MPFR*. Online resource: http://math.berkeley.edu/~wilken/code/gmpfrxx/.

Yoshida, H. 1990. Construction of higher order symplectic integrators. *Physics Letters A*, **150**(5), 262–268.

Zeng, Zhonggang. 2004. Algorithm 835: MultRoot—a Matlab Package for Computing Polynomial Roots and Multiplicities. *ACM Trans. Math. Softw.*, **30**(2), 218–236. Preprint available at http://www.neiu.edu/~zzeng/Papers/zrootpak.pdf, Software avaialbe from http://www.neiu.edu/~zzeng/multroot.htm.

Zeng, Zhonggang. 2005. Computing multiple roots of inexact polynomials. *Math. Comput.*, **74**(250). Preprint available at http://www.neiu.edu/~zzeng/mathcomp/zroot.pdf.

Zertchaninov, S., & Madsen, K. 1998. *A C++ Programme for Global Optimization*. IMM-REP-1998-04. Department of Mathematical Modelling, Technical University of Denmark, DK-2800 Lyngby, Denmark.

Nomenclature

$(a)_n$	Pochhammer symbol (page 56)
$\chi^2_{\nu,\alpha}$	α quantile of the central χ^2-distribution with ν degrees of freedom (page 112)
$\Gamma(x)$	Gamma Function (page 56)
$\Phi(x)$	CDF of the standardized normal distribution (page 126)
$\phi(x)$	pdf of the standardized normal distribution (page 126)
$\Phi^{-1}(\alpha)$	Inverse CDF of the standardized normal distribution (page 127)
$F_F(m,n,x)$	CDF of the central F-distribution (page 115)
$f_F(m,n,x)$	pdf of the central F-distribution (page 115)
$F_N(x;\mu,\sigma^2)$	CDF of the normal distribution with mean μ and variance σ^2 (page 126)
$F_N(x;\mu,\sigma^2)$	pdf of the normal distribution with mean μ and variance σ^2 (page 126)
$F_N^{-1}(\alpha;\mu,\sigma^2)$	Inverse CDF of the normal distribution with mean μ and variance σ^2 (page 127)
$F_t(n,x)$	CDF of the central t-distribution (page 130)
$f_t(n,x)$	pdf of the central t-distribution (page 130)
$F_{\chi^2}(n,x)$	CDF of the central chi-square distribution (page 111)
$f_{\chi^2}(n,x)$	pdf of the central chi-square distribution (page 111)
$F_{\chi^2}(n,x;\lambda)$	CDF of the noncentral chi-square distribution (page 347)
$f_{\chi^2}(n,x;\lambda)$	pdf of the noncentral chi-square distribution (page 347)
$F_{\nu_1,\nu_2,\alpha}$	α quantile of the central F-distribution with ν_1 and ν_2 degrees of freedom (page 115)
$F_{\text{Beta'}}(x;a,b,\lambda)$	CDF of the (singly) noncentral Beta-distribution (page 345)
$f_{\text{Beta'}}(x;a,b,\lambda)$	pdf of the (singly) noncentral Beta-distribution (page 345)
$F_{\text{Beta}}(a,b,x)$	CDF of the central Beta-distribution (page 106)
$f_{\text{Beta}}(a,b,x)$	pdf of the central Beta-distribution (page 106)
$F_{\text{Bin}}(n,k;p)$	CDF of the binomial distribution (page 109)
$f_{\text{Bin}}(n,k;p)$	pmf of the binomial distribution (page 109)
$F_{\text{NegBin}}(n,k;p)$	CDF of the negative binomial distribution (page 124)
$f_{\text{NegBin}}(n,k;p)$	pmf of the negative binomial distribution (page 124)
$f_{F'}(x;m,n)$	CDF of the (singly) noncentral F-distribution (page 350)
$f_{F'}(x;m,n)$	pdf of the (singly) noncentral F-distribution (page 350)
$F_{t'}(n,x,\delta)$	CDF of the (singly) noncentral t-distribution (page 352)
$f_{t'}(n,x,\delta)$	pdf of the (singly) noncentral t-distribution (page 352)
$I'_x(a,b)$	Derivative of the normalised incomplete beta function (page 60)
$I_x(a,b)$	Normalised incomplete beta function (page 57)
$t_{\nu,\alpha}$	α quantile of the central t-distribution with ν degrees of freedom (page 131)
$T_{\text{Owen}}(a,b)$	Owen's T-Function (page 356)
$t_{n,\delta;\alpha}$	α quantile of the noncentral t-distribution with ν degrees of freedom and noncentrality parameter δ (page 352)
z_α	α quantile of the standardized normal distribution (page 127)

CDF	cumulative distribution function (page 99)
pdf	probability density function (page 99)
pmf	probability mass function (page 99)