Introduction to Mathematica

Bruce A. Desmarais

Odum Institute for Research in Social Science

February 25, 2009

Welcome!

Who am I?

Welcome!

Who am I?

- Bruce Desmarais; Red Sox Fan and PhD student in the UNC Poli Sci Dept.

Welcome!

Who am I?

- Bruce Desmarais; Red Sox Fan and PhD student in the UNC Poli Sci Dept.
- Extensive experience with matrix algebra and statistical computing in Matlab and other software/languages

Welcome!

Who am I?

- Bruce Desmarais; Red Sox Fan and PhD student in the UNC Poli Sci Dept.
- Extensive experience with matrix algebra and statistical computing in Matlab and other software/languages
- No formal training in the instruction of Matlab (comments/suggestions will be much appreciated)

What will we cover?

Welcome!

Who am I?

- Bruce Desmarais; Red Sox Fan and PhD student in the UNC Poli Sci Dept.
- Extensive experience with matrix algebra and statistical computing in Matlab and other software/languages
- No formal training in the instruction of Matlab (comments/suggestions will be much appreciated)

What will we cover?

- Basic Structure of Mathematica

Welcome!

Who am I?

- Bruce Desmarais; Red Sox Fan and PhD student in the UNC Poli Sci Dept.
- Extensive experience with matrix algebra and statistical computing in Matlab and other software/languages
- No formal training in the instruction of Matlab (comments/suggestions will be much appreciated)

What will we cover?

- Basic Structure of Mathematica
- Arithmetic

Welcome!

Who am I?

- Bruce Desmarais; Red Sox Fan and PhD student in the UNC Poli Sci Dept.
- Extensive experience with matrix algebra and statistical computing in Matlab and other software/languages
- No formal training in the instruction of Matlab (comments/suggestions will be much appreciated)

What will we cover?

- Basic Structure of Mathematica
- Arithmetic
- Calculus and Linear Algebra

Welcome!

Who am I?

- Bruce Desmarais; Red Sox Fan and PhD student in the UNC Poli Sci Dept.
- Extensive experience with matrix algebra and statistical computing in Matlab and other software/languages
- No formal training in the instruction of Matlab (comments/suggestions will be much appreciated)

What will we cover?

- Basic Structure of Mathematica
- Arithmetic
- Calculus and Linear Algebra
- Intermediate Programming

Welcome!

Who am I?

- Bruce Desmarais; Red Sox Fan and PhD student in the UNC Poli Sci Dept.
- Extensive experience with matrix algebra and statistical computing in Matlab and other software/languages
- No formal training in the instruction of Matlab (comments/suggestions will be much appreciated)

What will we cover?

- Basic Structure of Mathematica
- Arithmetic
- Calculus and Linear Algebra
- Intermediate Programming
- Some Numerical Analysis and Statistics

The Notebook Interface

- Double-click the Mathematica icon to start Mathematica

The Notebook Interface

- Double-click the Mathematica icon to start Mathematica
- When Mathematica starts up, it gives you a blank notebook; save immediately

The Notebook Interface

- Double-click the Mathematica icon to start Mathematica
- When Mathematica starts up, it gives you a blank notebook; save immediately
- You enter Mathematica input into the notebook, then type Shift-Enter to make Mathematica process your input.

The Notebook Interface

- Double-click the Mathematica icon to start Mathematica
- When Mathematica starts up, it gives you a blank notebook; save immediately
- You enter Mathematica input into the notebook, then type Shift-Enter to make Mathematica process your input.
- Mathematica labels your input with In[n]:= . It labels the corresponding output Out[n]=.

The Notebook Interface

- Double-click the Mathematica icon to start Mathematica
- When Mathematica starts up, it gives you a blank notebook; save immediately
- You enter Mathematica input into the notebook, then type Shift-Enter to make Mathematica process your input.
- Mathematica labels your input with In[n]:= . It labels the corresponding output Out[n]=.
- Try typing *2+2*

The Notebook Interface

- Double-click the Mathematica icon to start Mathematica
- When Mathematica starts up, it gives you a blank notebook; save immediately
- You enter Mathematica input into the notebook, then type Shift-Enter to make Mathematica process your input.
- Mathematica labels your input with In[n]:= . It labels the corresponding output Out[n]=.
- Try typing *2+2*
- You may comment the notebook with *(* comment *)* Mathematica ignores everything between the starred parentheses.

Basic Arithmetic

The following are the basic arithmetic operations:

Basic Arithmetic

The following are the basic arithmetic operations:

`x^y`	`power`
`-x`	`minus`
`x/y`	`divide`
`x y z or x*y*z`	`multiply`
`x+y+z`	`add`

Mathematica obeys the usual order of operations as determined by paranthetic grouping.

Basic Arithmetic

The following are the basic arithmetic operations:

`x^y`	`power`
`-x`	`minus`
`x/y`	`divide`
`x y z or x*y*z`	`multiply`
`x+y+z`	`add`

Mathematica obeys the usual order of operations as determined by paranthetic grouping.
Try some out

Exact vs. Approximate Arithmetic

- Enter *2^100*

Exact vs. Approximate Arithmetic

- Enter *2^100*
- This result is given exactly..won't happen in R, SAS, STATA or Excel

Exact vs. Approximate Arithmetic

- Enter *2^100*
- This result is given exactly..won't happen in R, SAS, STATA or Excel
- If you don't want exact results *2^100 //N*

Exact vs. Approximate Arithmetic

- Enter *2^100*
- This result is given exactly..won't happen in R, SAS, STATA or Excel
- If you don't want exact results *2^100 //N*
- //N tells Mathematica you want approximate results

Exact vs. Approximate Arithmetic

- Enter *2^100*
- This result is given exactly..won't happen in R, SAS, STATA or Excel
- If you don't want exact results *2^100 //N*
- //N tells Mathematica you want approximate results
- Decimals imply approximations

Exact vs. Approximate Arithmetic

- Enter *2^100*
- This result is given exactly..won't happen in R, SAS, STATA or Excel
- If you don't want exact results *2^100 //N*
- //N tells Mathematica you want approximate results
- Decimals imply approximations
- *N[exp,n]* returns *exp* to n significant digits

Exact vs. Approximate Arithmetic

- Enter *2^100*
- This result is given exactly..won't happen in R, SAS, STATA or Excel
- If you don't want exact results *2^100 //N*
- //N tells Mathematica you want approximate results
- Decimals imply approximations
- *N[exp,n]* returns *exp* to n significant digits
- Any approximate number in a calculation makes the result approximate

Some Functions

`Sqrt[x]`	square root ()
`Exp[x]`	exponential ()
`Log[x]`	natural logarithm ()
`Log[b, x]`	logarithm to base ()
`n!`	factorial (product of integers)
`Abs[x]`	absolute value
`Round[x]`	closest integer to
`Mod[n, m]`	modulo (remainder on division of by)
`Random[ ]`	pseudorandom number between 0 and 1
`Max[x, y, ... ]`, `Min[x, y, ... ]`	maximum, minimum of , ,
`FactorInteger[n]`	prime factors of n (see Section 3.2.5)

Some Functions

`Sqrt[x]`	`square root ()`
`Exp[x]`	`exponential ()`
`Log[x]`	`natural logarithm ()`
`Log[b, x]`	`logarithm to base ()`
`n!`	`factorial (product of integers )`
`Abs[x]`	`absolute value`
`Round[x]`	`closest integer to`
`Mod[n, m]`	`modulo (remainder on division of by )`
`Random[ ]`	`pseudorandom number between 0 and 1`
`Max[x, y, ... ], Min[x, y, ... ]`	`maximum, minimum of , ,`
`FactorInteger[n]`	`prime factors of n (see Section 3.2.5)`

Note Caps.

Arithmetic Summary

- Arguments of functions are given in square brackets.

Arithmetic Summary

- Arguments of functions are given in square brackets.
- Names of built-in functions have their first letters capitalized

Arithmetic Summary

- Arguments of functions are given in square brackets.
- Names of built-in functions have their first letters capitalized
- Multiplication can be represented by a space.

Arithmetic Summary

- Arguments of functions are given in square brackets.
- Names of built-in functions have their first letters capitalized
- Multiplication can be represented by a space.
- Powers are denoted by ^.

Arithmetic Summary

- Arguments of functions are given in square brackets.
- Names of built-in functions have their first letters capitalized
- Multiplication can be represented by a space.
- Powers are denoted by ^.
- Numbers in scientific notation are entered, for example, as *2.5*^-4 or 2.5 10^-4*.

Referencing Previous Calculations

There are Many ways to reference past outputs

`%`	`the last result generated`
`%%`	`the next-to-last result`
`%% ... % (k times)`	`the kth previous result`
`%n`	`the result on output line Out[n] (to be used with care)`

Variable Definition

`x = value`	assign a value to the variable x
`x = y = value`	assign a value to both x and y
`x =.` or `Clear[x]`	remove any value assigned to x

Variable Definition

```
x = value          assign a value
                   to the variable
                   x

x = y = value      assign a value to
                   both x and y

x =.orClear[x]     remove any value
                   assigned to x
```

Conventionally, use only lower cases

Lists

```
x = {x1,x2...xn}      Make x a list
                      containing n elements
```

Lists

`x = {x1,x2...xn}`	`Make x a list containing n elements`

Referencing list elements

`{a, b, c}`	`a list`
`Part[list, i] or list[[i]]`	`the ith element of list`
`Part[list, {i, j, ... }] or ''`	`a list of the i, j, ... elements of list`

Bracketing

We have now seen the four bracketing forms in Mathematica. Here's a summary.

```
(term)        parentheses for grouping

f[x]          square brackets for functions

{a, b, c}     curly braces for lists

v[[i]]        double brackets for indexing (Part[v, i])
```

Semicolons

Semicolons can be used to control output related to the operations done in the Mathematica notebook.

`expr ;`	`do an operation, but display no output`
`expr1; expr2; expr3`	`do several operations, and give the result of the last one`
`expr1; expr2 ;`	`do the operations, but print no output`

Managing Errors

The square root function should have only one argument. Mathematica prints a message to warn you that you have given two arguments here

```
Sqrt[4, 5]
```

Each message has a name. You can switch off messages using Off.

```
Off[Sqrt::argx]
```

This switches Sqrt::argx back on again.

```
On[Sqrt::argx]
```

Symbolic Computation

- Transformation rules applied to expressions

Symbolic Computation

- Transformation rules applied to expressions
- Try *3x-2x*

Symbolic Computation

- Transformation rules applied to expressions
- Try *3x-2x*
- Values can be (locally) assigned to variables

Symbolic Computation

- Transformation rules applied to expressions
- Try *3x-2x*
- Values can be (locally) assigned to variables
- Try *1 + 2x /. x -> 3*

Symbolic Computation

- Transformation rules applied to expressions
- Try *3x-2x*
- Values can be (locally) assigned to variables
- Try *1 + 2x /. x -> 3*
- Now try *1 + 2x /. x -> 3*

Symbolic Computation

- Transformation rules applied to expressions
- Try *3x-2x*
- Values can be (locally) assigned to variables
- Try *1 + 2x /. x -> 3*
- Now try *1 + 2x /. x -> 3*
- This time *(x + y) (x - y)*^{2}*/.*$\{x->3, y->1-a\}$

Putting Expressions into Different Forms

`Expand[expr]`	multiply out products and powers
`ExpandAll[expr]`	apply Expand everywhere
`Factor[expr]`	reduce to a product of factors
`Together[expr]`	put all terms over a common denominator
`Apart[expr]`	separate into terms with simple denominators
`Cancel[expr]`	cancel common factors between numerators and denominators

Simplify Expression

Simplify is used as either *Simplify[expr]* or *Simplify[expr,assum]*. Try these:

```
Simplify[Sqrt[x^2], 0 < x]
```

```
Simplify[ArcSin[Sin[x]], -Pi/2 < x $<$ Pi/2]
```

```
Simplify[Sqrt[x^2], Element[x, Reals]]
```

```
Simplify[Sin[x + 2 n Pi], Element[n, Integers]]
```

Derivatives

`D[f, x]`	partial derivative
`D[f, x1, x2, ... ]`	multiple derivative
`D[f, {x, n}]`	repeated derivative
`Dt[f]`	total differential

Derivatives

```
D[f, x]                 partial derivative

D[f, x1, x2, ... ]      multiple derivative

D[f, {x, n}]            repeated derivative

Dt[f]                   total differential

D[ ArcTan[x], x ]

D[ x^n, {x, 3} ]

D[ 2 x f[x^2], x ]
```

Integrals

```
Integrate[f, x]

Integrate[f, x, y]

Integrate[f, {x,a ,b}]

Integrate[f, {x,ax ,bx }, {y,ay ,by }]
```

Integrals

```
Integrate[f, x]

Integrate[f, x, y]

Integrate[f, {x,a ,b}]

Integrate[f, {x,ax ,bx }, {y,ay ,by }]

Integrate[Sin[x]^2, {x, a, b} ]

Integrate[ x^2 + y^2, {x, 0, 1}, {y, 0, x} ]

Integrate[ x^x, x ]
```

Sums and Products

```
Sum[f, {i, a, b}]

Sum[f, {i, a, b, di}]

Sum[f, {i, ai, bi}, {j, aj, bj}]

Product[f, {i, a, b}]
```

Sums and Products

```
Sum[f, {i, a, b}]

Sum[f, {i, a, b, di}]

Sum[f, {i, ai, bi}, {j, aj, bj}]

Product[f, {i, a, b}]

Sum[x^i/i, {i, 1, 7}]

Sum[x^i/i, {i, 1, 5, 2}]

Product[x + i, {i, 1, 4}]

Sum[x^i y^j, {i, 1, 3}, {j, 1, i}]
```

Logicals and Compounds

`x == y`	equal
`x != y`	unequal
`x > y`	greater than
`x >= y`	greater than or equal to
`x < y`	less than
`x <= y`	less than or equal to
`x == y == z`	all equal
`x != y != z`	all unequal (distinct)
`x > y > z, etc.`	strictly decreasing, etc.
`!p`	not
`p && q && ...`	and
`p || q || ...`	or

Equation Solving

```
Solve[lhs==rhs, x]

Solve[{lhs1==rhs1,lhs2==rhs2 , ... }, {x, y, ...}]

Eliminate[{lhs1==rhs1,lhs2==rhs2 , ... }, {x, y,...}]
```

Equation Solving

```
Solve[lhs==rhs, x]

Solve[{lhs1==rhs1,lhs2==rhs2 , ... }, {x, y, ...}]

Eliminate[{lhs1==rhs1,lhs2==rhs2 , ... }, {x, y,...}]

Solve[x^2 + 2x - 7  0, x]

Solve[{a x + y  0, 2 x + (1-a) y  1}, {x, y}]

Eliminate[{a x + y  0, 2 x + (1-a) y  1}, y]
```

Inequalities

`Reduce[ineqs, {x, y, ... }]`	reduce a collection of inequalities
`FindInstance[ineqs, {x, y, ... }]`	find an instance satisfies the ineqs
`Maximize[{expr, ineq}, {x, y, ... }]`	maximize expr while satisfying ineqs

Inequalities

`Reduce[ineqs, {x, y, ... }]`	reduce a collection of inequalities
`FindInstance[ineqs, {x, y, ... }]`	find an instance satisfies the ineqs
`Maximize[{expr, ineq}, {x, y, ... }]`	maximize expr while satisfying ineqs

```
Reduce[x + y < 1 && y > x > 0, {x, y}]

FindInstance[x + y < 1 && y^2 > x > 0, {x, y}]

Maximize[{x^2 + y, x^2 + y^2  1}, {x, y}]
```

Differential Eqns

`DSolve[eqns, y[x], x]`	`solve a differential equation for y[x], taking x as the independent variable`
`DSolve[eqns, y, x]`	`give a solution for y in pure function form`

Differential Eqns

```
DSolve[eqns, y[x], x]    solve a differential equation
                         for y[x], taking x as the
                         independent variable

DSolve[eqns, y, x]       give a solution for y in pure
                         function form
```

```
DSolve[ y'[x]==  a y[x] + 1, y[x], x ]

DSolve[ {y'[x] == a y[x] + 1, y[0] == 0}, y[x], x ]

DSolve[ y'[x]==  x + y[x], y, x ]
```

Power Series

```
Series[(1 + x)^n, {x, 0, 3}]

Normal[%]

Series[Exp[-a t] (1 + Sin[2 t]), {t, 0, 4}]

Normal[%]

Series[Exp[x], {x, 0, 5}]

Normal[%]

Series[1 + f[t], {t, 0, 3}]
```

Limits

```
Limit[expr, x->x0]  the limit of expr as x approaches x0

t = Sin[x]/x

t /. x->0

t /. x->0.01

Limit[t, x->0]
```

Function Definition

```
f[x_] := x^2          define the function f

f[x_,y_] := x^2*y     define the function f

?f                    show the definition of f

Clear[f]              clear all definitions for f
```

Function Definition

```
f[x_] := x^2          define the function f

f[x_,y_] := x^2*y     define the function f

?f                    show the definition of f

Clear[f]              clear all definitions for f

f[x_] := x^2

h[x_, xmax_] := (x - xmax)^2 / xmax

?h
```

Constructing Matrices

```
A = MatrixForm[{{a11,a12,..a1n},..,{am1,am2..,amn}}]

Array[f, {m, n}]          build an mxn matrix f[i, j]

DiagonalMatrix[list]  generate a diagonal matrix

IdentityMatrix[n]  generate an  identity matrix
```

Constructing Matrices

```
A = MatrixForm[{{a11,a12,..a1n},..,{am1,am2..,amn}}]

Array[f, {m, n}]          build an mxn matrix f[i, j]

DiagonalMatrix[list]  generate a diagonal matrix

IdentityMatrix[n]  generate an  identity matrix

{{1, 1, 1}, {2, 2, 2}}

MatrixForm[%]

DiagonalMatrix[{a, b, c}]

Array[a, {2, 2}]
```

Working with Matrix Elements

`m[[i, j]]`	the [i,j]th entry
`m[[i]]`	the ith row
`m[[All, i]]`	the column i
`Take[m, {i0,i1}, {j0, j1}]`	submatrix
`m[[{i1, ... ,ir},{j1, ... ,jr}]]`	submatrix
`Tr[m, List]`	elements on the diagonal
`ArrayRules[m]`	positions of non-zero elements

Matrix Multiplication

`c v, c m, etc.`	`multiply each element by scalar`
`v.v, v.m, m.v, m.m, etc.`	`vector and matrix multiplication`
`Cross[v, v]`	`vector cross product`
`Outer[Times, t, u]`	`outer product`

Matrix Multiplication

```
c v, c m, etc.                multiply each element by scalar

v.v, v.m, m.v, m.m, etc.      vector and matrix multiplication

Cross[v, v]                   vector cross product

Outer[Times, t, u]            outer product

k {a, b, c}

{{a, b}, {c, d}} . {x, y}

Outer[Times, {{1, 2}, {3, 4}}, {x, y, z}]

DiagonalMatrix[{a, b, c}]
```

Basic Matrix Operations

`Transpose[m]`	transpose
`ConjugateTranspose[m]`	conjugate transpose
`Inverse[m]`	matrix inverse
`Det[m]`	determinant
`Minors[m]`	matrix of minors
`Minors[m, k]`	minors
`Tr[m]`	trace

Solving Linear Systems

`LinearSolve[m, b]`	a vector x which solves the matrix equation m.x == b
`NullSpace[m]`	a list of basis vectors whose linear combinations satisfy the matrix equation m.x == 0
`RowReduce[m]`	a simplified form of m obtained by making linear combinations of rows

Solving Linear Systems

`LinearSolve[m, b]`	a vector x which solves the matrix equation m.x == b
`NullSpace[m]`	a list of basis vectors whose linear combinations satisfy the matrix equation m.x == 0
`RowReduce[m]`	a simplified form of m obtained by making linear combinations of rows

```
m = {{1, 5}, {2, 1}}
m . {x, y}  {a, b}
Solve[ %, {x, y} ]
```

Eigen Vals. and Vects.

`Eigenvalues[m]`	`a list of the eigenvalues`
`Eigenvectors[m]`	`a list of the eigenvectors`
`Eigensystem[m]`	`{eigenvalues,eigenvectors}`

Eigen Vals. and Vects.

```
Eigenvalues[m]                    a list of the eigenvalues

Eigenvectors[m]                   a list of the eigenvectors

Eigensystem[m]                    {eigenvalues,eigenvectors}

Eigenvalues[ {{a, b}, {-b, 2a}} ]

m = {{2.3, 4.5}, {6.7, -1.2}}

Eigenvectors[ m ]

{vals, vecs} = Eigensystem[m]
```

Numeric Types

In monitoring precision of numerical operations, it is important to understand the different numeric storage types.

Numeric Types

In monitoring precision of numerical operations, it is important to understand the different numeric storage types.

```
Integer    arbitrary-length exact integer

Rational   integer/integer in lowest terms

Real       approximate real number, with any specified prec

Complex    complex number of the form number + number I
```

Numeric Types

In monitoring precision of numerical operations, it is important to understand the different numeric storage types.

```
Integer    arbitrary-length exact integer

Rational   integer/integer in lowest terms

Real       approximate real number, with any specified prec

Complex    complex number of the form number + number I
```

Precise arithmetic is performed if all numbers are rational and integer types.

Numeric Types

In monitoring precision of numerical operations, it is important to understand the different numeric storage types.

```
Integer    arbitrary-length exact integer

Rational   integer/integer in lowest terms

Real       approximate real number, with any specified prec

Complex    complex number of the form number + number I
```

Precise arithmetic is performed if all numbers are rational and integer types.

```
Head[x]      Returns Numeric Type
```

Interval Math

Especially in probability applications the following built-in interval arithmetic functions can be helpful.

Interval Math

Especially in probability applications the following built-in interval arithmetic functions can be helpful.

```
Interval[{min, max}]  the interval from min to max

IntervalUnion[{min1,max1},{min2,max2} , ... ]

IntervalIntersection[{min1,max1},{min2,max2}, ... ]

IntervalMemberQ[interval, x]

IntervalMemberQ[interval1, interval2]
```

Some Useful Numerical Functions

The following functions are very helpful in many settings

Some Useful Numerical Functions

The following functions are very helpful in many settings

`IntegerPart[x]`	integer part of x
`FractionalPart[x]`	fractional part of x
`Round[x]`	integer closest to x
`Floor[x]`	greatest integer not larger than x
`Ceiling[x]`	least integer not smaller than x
`Rationalize[x]`	rational number approximation to x
`Rationalize[x, dx]`	within tolerance dx

Some Boolean Tools

A Mathematica exclusive.....I think

Some Boolean Tools

A Mathematica exclusive.....I think

- The function *Boole[expr]* returns 1 if the expression is true 0 otherwise

Some Boolean Tools

A Mathematica exclusive.....I think

- The function *Boole[expr]* returns 1 if the expression is true 0 otherwise
- Try *Integrate[Boole[*$x^2 + y^2 1], \{x, -1, 1\}, \{y, -1, 1\}$*]*

Some Boolean Tools

A Mathematica exclusive.....I think

- The function *Boole[expr]* returns 1 if the expression is true 0 otherwise
- Try *Integrate[Boole[*$x^2 + y^2 1], \{x, -1, 1\}, \{y, -1, 1\}$*]*
- Another cool function is *Piecewise[*$\{\{$*val1,cond1* $\}$, $\{$*val2,cond2* $\}$, ... $\}$*,val]*

Some Boolean Tools

A Mathematica exclusive.....I think

- The function *Boole[expr]* returns 1 if the expression is true 0 otherwise
- Try *Integrate[Boole[*$x^2 + y^2 1], \{x, -1, 1\}, \{y, -1, 1\}$*]*
- Another cool function is *Piecewise[{{val1,cond1 }, {val2,cond2 }, ... },val]*
- You dont need to include *val* in the above call

Some Boolean Tools

A Mathematica exclusive.....I think

- The function *Boole[expr]* returns 1 if the expression is true 0 otherwise
- Try *Integrate[Boole[*$x^2 + y^2 1], \{x, -1, 1\}, \{y, -1, 1\}$*]*
- Another cool function is *Piecewise[{{val1,cond1 }, {val2,cond2 }, ... },val]*
- You dont need to include *val* in the above call
- Now try *Plot[Piecewise[*$\{\{x^2, x < 0\}, \{1 - x, x > 0\}\}], \{x, -1, 1\}$*]*

Random Numbers in Mathematica

Random Numbers in Mathematica

`Random[ ]`	real between 0 and 1
`Random[Real, max]`	real between 0 and max
`Random[Real, {min, max}]`	real between min and max
`SeedRandom[ ]`	reseed the pseudorandom generator with the time of day
`SeedRandom[s]`	reseed with the integer s

Functions, Functions....

These are self-explanatory but essential transcendentals

Functions, Functions....

These are self-explanatory but essential transcendentals

```
Exp[z]
Log[z]
Sin[z], Cos[z], Tan[z], Csc[z], Sec[z], Cot[z]
ArcSin[z], ArcCos[z], ArcTan[z], ArcCsc[z],
ArcSec[z], ArcCot[z]
Sinh[z], Cosh[z], Tanh[z], Csch[z], Sech[z],
Coth[z]
ArcSinh[z], ArcCosh[z], ArcTanh[z], ArcCsch[z],
ArcSech[z], ArcCoth[z]
```