Calcolo numerico

Integer is the head used for integers.

Rational is the head used for rational numbers.

Real is the head used for real (floating-point) numbers.

Complex is the head used for complex numbers.

Numeri interi

IntegerQ[expr] gives True if expr is an integer, and False otherwise.

EvenQ[expr] gives True if expr is an even integer, and False otherwise.

OddQ[expr] gives True if expr is an odd integer, and False otherwise.

PrimeQ[expr] yields True if expr is a prime number, and yields False otherwise. In the current version of Mathematica, the algorithm used for large integers is probabilistic, but very reliable (pseudoprime test and Lucas test).

IntegerDigits[n] gives a list of the decimal digits in the integer n.
IntegerDigits[n, b] gives a list of the base-b digits in the integer n.
IntegerDigits[n, b, k] gives a list of length k containing the least significant digits of n.

Quotient[n, m] gives the integer quotient of n and m, defined as Floor[n/m].

Mod[m, n] gives the remainder on division of m by n. The result has the same sign as n.

GCD[n1, n2, ...] gives the greatest common divisor of the integers ni.

LCM[n1, n2, ...] gives the least common multiple of the integers ni.

FactorInteger[n] gives a list of the prime factors of the integer n, together with their exponents.

Divisors[n] gives a list of the integers that divide n.

EulerPhi[n] gives the Euler totient function phi(n). It is the number of positive integers less than n which are relatively prime to n.

Prime[n] gives the nth prime number.

PrimePi[n] gives the number of primes less than or equal to n.

n! gives the factorial of n.

Binomial[n, m] gives the binomial coefficient.

Multinomial[n1, n2, ...] gives the multinomial coefficient (n1+n2+...)!/(n1! n2! ...).

Numeri "reali"

NumberQ[expr] gives True if expr is a number, and False otherwise.

N[expr] gives the numerical value of expr.
N[expr, n] does computations to n-digit precision.

Positive[x] gives True if x is a positive number.

Negative[x] gives True if x is a negative number.

Sign[x] gives -1, 0 or 1 depending on whether x is negative, zero, or positive.
Sign[z] gives the sign of the complex number z.

RealDigits[x] gives a list of the digits in the approximate real number x, together with the number of digits that appear to the left of the decimal point in scientific notation.
RealDigits[x, b] gives a list of base-b digits in x.

Floor[x] gives the greatest integer less than or equal to x.

Ceiling[x] gives the smallest integer greater than or equal to x.

Round[x] gives the integer closest to x.

Rationalize[x] takes Real numbers in x that are close to rationals, and converts them to exact Rational numbers.
Rationalize[x, dx] performs the conversion whenever the error made is smaller in magnitude than dx.

Chop[expr] replaces approximate real numbers in expr that are close to zero by the exact integer 0.
Chop[expr, tol] replaces approximate real numbers in expr that differ from zero by less than tol with 0.

Random[ ] gives a uniformly distributed pseudorandom Real in the range 0 to 1.
Random[type, range] gives a pseudorandom number of the specified type, lying in the specified range. Possible types are: Integer, Real and Complex. The default range is 0 to 1. You can give the range {min, max} explicitly; a range specification of max is equivalent to {0, max}.

SeedRandom[n] resets the pseudorandom number generator, using the integer n as a seed.
SeedRandom[ ] resets the generator, using as a seed the time of day.

Intervalli

Interval[{min, max}] represents the range of values between min and max.
Interval[{a,b}, {c,d}, ...] represents the union of the ranges a to b, c to d, ...

IntervalMemberQ[int, x] gives True if the number x in in the interval int, False otherwise.
IntervalMemberQ[int1, int2] gives True if the interval int2 is contained within the interval int1.

IntervalUnion[int1, int2, ...] gives an interval representing the union of the intervals int1, int2, ...

IntervalIntersection[int1, int2, ...] gives an interval representing the intersection of the intervals int1, int2, ...

Fuzioni numeriche

NSum[f, {i, imin, imax}] gives a numerical approximation to the sum of f with i running from imin to imax.
NSum[f, {i, imin, imax, di}] uses a step di in the sum.
NSum[f, {i, imin, imax}, {j, jmin, jmax}, ...] gives a multi-dimensional summation.

NProduct[f, {i, imin, imax}] gives a numerical approximation to the product of f with i running from imin to imax.
NProduct[f, {i, imin, imax, di}] uses a step di in the product.

FindMinimum[f, {x, x0}] searches for a local minimum in f, starting from the point x=x0.

ConstrainedMax[f, {inequalities}, {x, y, ...}] finds the global maximum of f in the domain specified by the inequalities. The variables x, y, ... are all assumed to be non-negative.

ConstrainedMin[f, {inequalities}, {x, y, ...}] finds the global minimum of f in the domain specified by the inequalities. The variables x, y, ... are all assumed to be non-negative.

FindRoot[lhs == rhs, {x, x0}] searches for a numerical solution to the equation lhs == rhs, starting with x == x0.

NSolve[eqns, vars] attempts to solve numerically an equation or set of equations for the variables vars. Any variable in eqns but not vars is regarded as a parameter.
NSolve[eqns] treats all variables encountered as vars above.
NSolve[eqns, vars, prec] attempts to solve numerically the equations for vars using prec digits precision.

NDSolve[eqns, y, {x, xmin, xmax}] finds a numerical solution to the differential equations eqns for the function y with the independent variable x in the range xmin to xmax.
NDSolve[eqns, {y1, y2, ...}, {x, xmin, xmax}] finds numerical solutions for the functions yi.
NDSolve[eqns, y, {x, x1, x2, ...}] forces a function evaluation at each of x1, x2, ... The range of numerical integration is from Min[x1, x2, ...] to Max[x1, x2, ...].

NIntegrate[f, {x, xmin, xmax}] gives a numerical approximation to the integral of f with respect to x over the interval xmin to xmax.

Interpolation[data] constructs an InterpolatingFunction object which represents an approximate function that interpolates the data. The data can have the forms {{x1, f1}, {x2, f2}, ...} or {f1, f2, ...}, where in the second case, the xi are taken to have values 1, 2, ...

Fit[data, funs, vars] finds a least-squares fit to a list of data as a linear combination of the functions funs of variables vars. The data can have the form {{x1, y1, ..., f1}, {x2, y2, ..., f2}, ...}, where the number of coordinates x, y, ... is equal to the number of variables in the list vars. The data can also be of the form {f1, f2, ...}, with a single coordinate assumed to take values 1, 2, ... The argument funs can be any list of functions that depend only on the objects vars.