MathWorks®
Accelerating the pace of engineering and science

MATLAB® Numerical Representations and Numerical Methods

Numerical Representations

`single(A)`	Convert to single precision (8 digits).
`double(A)`	(Default) Convert to double precision (16 digits).
`vpa(A)`	Variable precision arithmetic (Default 32 digits).
`digits(n)`	Change variable precision used to n.
`eps(X)`	Spacing of floating-point numbers. eps(X) is the positive distance from ABS(X) to the next larger in magnitude floating point number of the same precision as X. X may be either double precision or single precision. For all X, eps(X) is equal to eps(ABS(X)).
`X = intmin` `intmin(CLASSNAME)`	Is the smallest value in the integer class CLASSNAME. Valid values of CLASSNAME are 'int8', 'uint8', 'int16', 'uint16', 'int32', 'uint32', 'int64' and 'uint64'. The default is 'int32'.
`X = intmax` `intmax(CLASSNAME)`	Is the largest value in the integer class CLASSNAME. (like intmin)
`realmin`	Smallest positive normalized floating-point number. Default is 'double' which is analogous to IEEE double precision.
`realmin(CLASSNAME)`	Returns the smallest positive normalized floating-point number in CLASSNAME.
`realmax` `realmax(CLASSNAME)`	Largest positive normalized floating-point number. (like realmin)
`binStr = dec2bin(D)`	Convert decimal integer to its binary representation.
`D = bin2dec(binStr)`	Convert text representation of binary integer to double value.
`baseStr = dec2base(D,n)`	Convert decimal integer to its base-n representation.
`D = base2dec(baseStr,n)`	Convert text representation of base-n integer to double value.
`binStr = dec2bin(D)`	Convert decimal integer to its binary representation.
`hexStr = dec2hex(D)`	Convert decimal integer to its hexadecimal representation.

Common Operations

`a:dx:b`	Defines a vector with dx spacing.
`linspace(a,b,n)`	A vector with n equally spaced values.
`logspace(a,b,n)`	A vector of n logarithmically spaced values.
`length(A)`	Length of largest array dimension.
`size(A)`	Array dimensions.
`numel(A)`	Number of elements in array.
`zeros(m,n);`	Create m x n matrix of zeros.
`ones(m,n)`	Creates m x n matrix of ones.
`eye(n)`	Creates a n x n identity matrix.
`[X, Y] = meshgrid(x,y)`	Creates 2D and 3D grids.
`rand(m,n), randn, randi`	Uniformly distributed random numbers (DRN) or normally DRN or integers.

Condition Number of Matrices

`cond(A)`	Default 2-norm.
`cond(A,P)`	P = 1, returns the 1-norm condition number. P = 2, returns the 2-norm condition number. P = inf, returns the infinity condition number. P = 'fro', returns the Frobenius norm condition number.
`C = rcond(A)`	C is an estimate for the reciprocal condition of A in 1-norm. If A is well conditioned, rcond(A) is near 1.0. If A is badly conditioned, rcond(A) is near 0.
`c = condest(A)`	Computes a lower bound c for the 1-norm condition number of a square matrix A. Useful for sparse matrices.
`n = norm(X, p)`	Returns the p-norm of vector or matrix. Use p = 1, 2 or Inf. P =2 is the default.
`n = normest(S)`	Returns an estimate of the 2-norm of the matrix S using the relative error tolerance tol.
`n = normest(S,tol)`	The default tolerance is 1.0e-6.
`c = condeig(A)`	Returns a vector of condition numbers for the eigenvalues of A. These condition numbers are the reciprocals of the cosines of the angles between the left and right eigenvectors.

Vector and Matrix Representation

`D = diag(v)`	Returns a square diagonal matrix with the elements of vector v on the main diagonal.
`v = diag(D)`	Returns the diagonal elements of a matrix in vector form.
`R = rref(A)`	Produces the reduced row echelon form of A
`rank(A)`	provides an estimate of the number of linearly independent rows or columns of a matrix A.
`rank(A,TOL)`	The number of singular values of A that are larger than TOL. By default, TOL = max(size(A)) * eps(norm(A)).
`S = sparse(A)`	Converts a full matrix into sparse form by squeezing out any zero elements. If a matrix contains many zeros, converting the matrix to sparse storage saves memory.
`S = sparse(m,n)`	Generates an m-by-n all zero sparse matrix.
`TF = issparse(S)`	Returns logical 1 (true) if the storage class of S is sparse and logical 0 (false) otherwise.
`A = full(S)`	Converts sparse matrix S to full storage organization, such that issparse(A) returns logical 0 (false).
`spy(S)`	Plots the sparsity pattern of matrix S. Nonzero values are colored while zero values are white. The plot displays the number of nonzeros in the matrix, nz = nnz(S). For additional operations see sprandsym, sprandn, spones, spalloc, and speye.

Solving Ax = b (x = A^-1*b) and Matrix Decomposition

`x = A^-1*b = inv(A)*b`	Not recommended because it is costly and prone to errors. A^-1 might also not exist. If you do use it is recommended that you use inv(A) for solving systems of linear equations.
`x = A\B = mldivide(A,B)`	Solves A*x = B. The matrices A and B must have the same number of rows. MATLAB® displays a warning message if A is badly scaled or nearly singular but performs the calculation regardless. mldivide is rarely used. It enables operator overloading for classes.
`X = pinv(A)`	Produces a matrix X of the same dimensions as A' so that A*X*A = A, X*A*X = X and A*X and X*A are Hermitian. The computation is based on SVD(A) and any singular values less than a tolerance are treated as zero.
`UT_X = triu(X)`	Returns the upper triangular part of X.
`LT_X = tril(X)`	Returns the lower triangular part of X. Note: X ≠ LT_X*UT_X
`[L,U] = lu(A)`	LU matrix factorization. factorizes the full or sparse matrix A into an upper triangular matrix U and a permuted lower triangular matrix L such that A = L*U.
`[L,U,P] = lu(A)` `A_2 = P'*L*U;` `A_2 == round(A, 2);`	also returns a permutation matrix P such that A = P'*L*U. With this syntax, L is unit lower triangular, and U is upper triangular.
`[L,U] = ilu(A)`	Incomplete LU factorization. Performs the incomplete LU factorization of a sparse matrix A with zero-fill, and returns the lower triangular matrix L and the upper triangular matrix U.
`[L,U,P] = ilu(A)`	Also returns the permutation matrix P such that L and U are incomplete factors of P*A or A*P. By default, P is an identity matrix for the incomplete LU factorization without pivoting.
`W = ilu(A)`	Returns the nonzeros of the LU factors. The output W is equal to L + U - speye(size(A)).
`R = chol(A)`	Cholesky factorization Factorizes symmetric positive definite matrix A into an upper triangular R that satisfies A = R'*R. If A is nonsymmetric, then chol treats the matrix as symmetric and uses only the diagonal and upper triangle of A.
`R = chol(A,triangle)`	Specifies which triangular factor of A to use in computing the factorization. For example, if triangle is 'lower', then chol uses only the diagonal and lower triangular portion of A to produce a lower triangular matrix R that satisfies A = R*R'. The default value of triangle is 'upper'.
`L = ichol(A)`	Incomplete Cholesky factorization. Performs the incomplete Cholesky factorization of A with zero-fill. A must be a sparse square matrix. By default, ichol references the lower triangle of A and produces lower triangular factors.

`R = qr(A)`	QR decomposition. Returns the upper-triangular R factor of the QR decomposition A = Q*R.
`[Q,R] = qr(A)`	Performs a QR decomposition on m-by-n matrix A such that A = Q*R. The factor R is an m-by-n upper-triangular matrix, and the factor Q is an m-by-m orthogonal matrix.
`[Q,R,P] = qr(A)`	Additionally returns a permutation matrix P such that A*P = Q*R. If A is full, the permutation matrix is chosen so that abs(diag(R)) is decreasing.
`S = svd(A)`	Singular value decomposition Returns the singular values of matrix A in descending order.
`[U,S,V] = svd(A)`	Performs a singular value decomposition of matrix A, such that A = U*S*V'.
`s = svds(A)`	Subset of singular values and vectors Returns a vector of the six largest singular values of matrix A. This is useful when computing all of the singular values with svd is computationally expensive, such as with large sparse matrices.
`s = svds(A,k)`	Returns the k largest singular values.
`s = svds(A,k,sigma)`	Returns k singular values based on the value of sigma. For example, svds(A,k,'smallest') returns the k smallest singular values.
`[U,V,X,C,S] = gsvd(A,B)`	Generalized SVD. A and B must have the same number of columns, but may have different numbers of rows.
`linsolve(A,B)`	Solve linear system of the form AX = B.
`lsqminnorm(A,B)`	Least-squares solution to linear equation.
	For additional methods see pcg, minres, lsqr, symmlq, qmr, and bicg.

Linear Algebra and Statistics | Note: MATLAB is Column Major

`trace(A)`	Sum of diagonal elements of matrix.
`det(A)`	Determinant of matrix.
`poly(A)`	Characteristic polynomial of matrix.
`eig(A), eigs`	Eigenvalues and vectors of matrix (subset).
`null(A)`	Null space of matrix.
`orth(A)`	Orthonormal basis for matrix range.
`sum(A), prod`	Sum or product (along columns).
`max(A), min(A),`	Largest and smallest element(s).
`[minA, maxA]= bounds(A)`	Both largest and smallest element(s).
`mean(A), median, mode`	Statistical operations.
`std(A), var`	Standard deviation and variance.
`movsum(A,n), movprod, movmax, movmin, movmean, movmedian, movstd, movvar`	Moving statistical functions. n = length of moving window.
`cumsum(A), cumprod, cummax, cummin`	Cumulative statistical functions.
`normalize(A)`	Normalize data.

MathWorks®
Accelerating the pace of engineering and science

Numerical Methods

Error types in numerical solutions

1) Round off errors due to having a finite number of bits.

Rounding: You have 0.25678 and you round it to 0.26 and then perform calculations.

Chopping: You have 1/3 and then you chop digits until you have 0.33 then preform calculations that require precision.

2) Truncation error: Due to approximating a process or function. Talyor series approximations are finite.

3) Swamping: When significant digits get "washed out" when you have a huge number and a tiny number.

4) (Catastrophic) Cancellation: Floating point arithmetic is not associate, order matters. Sometimes when you subtract two numbers that are close together, you will lose significant digits.

Note: When using a package or toolbox see how roundup is handled.

ODE Solvers

`[t,y] = ode45(odefun,tspan,y0)`	Solves nonstiff differential equations. Medium order method.
`ode23`	— low order method
`ode78`	— high order method
`ode89`	— high order method
`ode113`	— variable order method.
`[t,y] = ode15s(odefun,tspan,y0)`	Solve stiff differential equations and DAEs — variable order method.
`ode23s`	Solve stiff differential equations — low order method.
`ode23t`	Solve moderately stiff ODEs and DAEs — trapezoidal rule.
`ode23tb`	Solve stiff differential equations — trapezoidal rule + backward differentiation formula.
`[t,y]= ode15i(odefun,tspan,y0,yp0)`	Solve fully implicit differential equations. Variable order method.
`[y0_new,yp0_new] = decic(odefun,t0,y0,fixed_y0,yp0,fixed_yp0)`	Compute consistent initial conditions for ode15i.

Numerical Integration and Differentiation

`integral(fun,xmin,xmax)`	Numerically integrates function fun from xmin to xmax using global adaptive quadrature and default error tolerances.
`integral2(fun,xmin,xmax,ymin,ymax)`	Approximates the integral of the function z = fun(x,y) over the planar region xmin ≤ x ≤ xmax and ymin(x) ≤ y ≤ ymax(x).
`integral3(fun,xmin,xmax,ymin,ymax,zmin,zmax)`	Approximates the integral of the function z = fun(x,y,z) over the region xmin ≤ x ≤ xmax, ymin(x) ≤ y ≤ ymax(x), and zmin(x,y) ≤ z ≤ zmax(x,y).
`q = quadgk(fun,a,b)`	Integrates the function handle fun from a to b using high-order global adaptive quadrature and default error tolerances.
`q = quad2d(fun,a,b,c,d)`	Approximates the integral of fun(x,y) over the planar region a≤x≤b and c(x)≤y≤d(x). The bounds c and d can each be scalars or function handles.
`curl(X,Y,Z,U,V,W)`	Curl and angular velocity.
`divergence(X,..,W)`	Compute divergence of vector field.
`deval(sol,x)`	Evaluate solution of differential equation.
`pdepe(m,pde,ic,..., bc,xm,ts)`	Solve 1D partial differential equation.
`pdeval(m,xmesh,..., usol,xq)`	Interpolate numeric PDE solution.

Integrate Numeric Data

`Y = [1 4 9 16 25]; % This corresponds to f(x) = x^2 in the domain [1, 5]`
`Q = trapz(Y) % 42`
	Trapezoidal numerical integration.
	Computes the approximate integral of Y via the trapezoidal method with unit spacing. The size of Y determines the dimension to integrate along.

`Q2 = cumtrapz(Y)`
`% Q2 = 0 2.5 9 21.5 42`
`% Q2(end) == trapz(Y)`
	Cumulative trapezoidal numerical integration.
	Computes the approximate cumulative integral of Y via the trapezoidal method with unit spacing. The size of Y determines the dimension to integrate along.

`X = 0:pi/5:pi;`
`Y = sin(X');`
`Q = trapz(X,Y)`
`Q = cumtrapz(X,Y)`
	Integrates Y with respect to the coordinates or scalar spacing specified by X.

Finite Difference Derivatives

`L = del2(U)`	Discrete Laplacian. Returns a discrete approximation of Laplace's differential operator applied to U using the default spacing, h = 1, between all points.
`L = del2(U,h)`	Specifies a uniform, scalar spacing, h, between points in all dimensions of U.
`Y = diff(X)`	Differences and approximate derivatives. Calculates differences between adjacent elements of X along the first array dimension whose size does not equal 1.
`Y = diff(X,n)`	Calculates the nth difference by applying the diff(X) operator recursively n times. In practice, this means diff(X,2) is the same as diff(diff(X)).
`F_prime = diff(f)/h;`	Approximates derivatives using a f – function and h – step size.
`FX = gradient(F)`	Returns the one-dimensional numerical gradient of vector F. The output FX corresponds to ∂F/∂x, which are the differences in the x (horizontal) direction. The spacing between points is assumed to be 1.
`[FX,FY,FZ,...,FN] = gradient(F)`	Returns the N components of the numerical gradient of F, where F is an array with N dimensions.