mldivide \, mrdivide /

Left or right matrix division

Syntax

```
mldivide(A,B)      A\B
mrdivide(B,A)      B/A
```

Description

`mldivide(A,B)` and the equivalent `A\B` perform matrix left division (back slash). `A` and `B` must be matrices that have the same number of rows, unless `A` is a scalar, in which case `A\B` performs element-wise division — that is, `A\B = A.\B`.

If `A` is a square matrix, `A\B` is roughly the same as `inv(A)*B`, except it is computed in a different way. If `A` is an `n`-by-`n` matrix and `B` is a column vector with `n` elements, or a matrix with several such columns, then `X = A\B` is the solution to the equation *AX* = *B* (see Algorithm for details). A warning message is displayed if `A` is badly scaled or nearly singular.

If `A` is an `m`-by-`n` matrix with `m ~= n` and `B` is a column vector with `m` components, or a matrix with several such columns, then `X = A\B` is the solution in the least squares sense to the under- or overdetermined system of equations *AX* = *B*. In other words, `X` minimizes `norm(A*X - B)`, the length of the vector *AX* - *B*. The rank `k` of `A` is determined from the QR decomposition with column pivoting (see Algorithm for details). The computed solution `X` has at most `k` nonzero elements per column. If `k < n`, this is usually not the same solution as `x = pinv(A)*B`, which returns a least squares solution.

`mrdivide(B,A)` and the equivalent `B/A` perform matrix right division (forward slash). `B` and `A` must have the same number of columns.

If `A` is a square matrix, `B/A` is roughly the same as `B*inv(A)`. If `A` is an `n`-by-`n` matrix and `B` is a row vector with `n` elements, or a matrix with several such rows, then `X = B/A` is the solution to the equation *XA* = *B* computed by Gaussian elimination with partial pivoting. A warning message is displayed if `A` is badly scaled or nearly singular.

If `B` is an `m`-by-`n` matrix with `m ~= n` and `A` is a column vector with `m` components, or a matrix with several such columns, then `X = B/A` is the solution in the least squares sense to the under- or overdetermined system of equations *XA* = *B*.

> **Note** Matrix right division and matrix left division are related by the equation `B/A = (A'\B')'`.

Least Squares Solutions

If the equation *Ax* = *b* does not have a solution (and *A* is not a square matrix), `x = A\b` returns a *least squares solution* — in other words, a solution that minimizes the length of the vector *Ax* - *b*, which is equal to `norm(A*x - b)`. See Example 3 for an example of this.

Examples

Example 1

Suppose that `A` and `b` are the following.

```
A = magic(3)

A =

     8     1     6
```

```
     3     5     7
     4     9     2

b = [1;2;3]

b =

     1
     2
     3
```

To solve the matrix equation *Ax = b*, enter

```
x=A\b

x =

    0.0500
    0.3000
    0.0500
```

You can verify that `x` is the solution to the equation as follows.

```
A*x

ans =

    1.0000
    2.0000
    3.0000
```

Example 2 — A Singular

If `A` is singular, `A\b` returns the following warning.

```
Warning: Matrix is singular to working precision.
```

In this case, *Ax = b* might not have a solution. For example,

```
A = magic(5);
A(:,1) = zeros(1,5); % Set column 1 of A to zeros
b = [1;2;5;7;7];
x = A\b
Warning: Matrix is singular to working precision.

ans =

   NaN
   NaN
   NaN
   NaN
   NaN
```

If you get this warning, you can still attempt to solve *Ax = b* using the pseudoinverse function `pinv`.

```
x = pinv(A)*b

x =

         0
    0.0209
    0.2717
    0.0808
   -0.0321
```

The result `x` is least squares solution to *Ax = b*. To determine whether `x` is a exact solution — that is, a solution for which *Ax - b = 0* — simply compute

```
A*x-b

ans =

   -0.0603
    0.6246
   -0.4320
    0.0141
    0.0415
```

The answer is not the zero vector, so `x` is not an exact solution.

Pseudoinverses, in the online MATLAB Mathematics documentation, provides more examples of solving linear systems using `pinv`.

Example 3

Suppose that

```
A = [1 0 0;1 0 0];
b = [1; 2];
```

Note that *Ax = b* cannot have a solution, because `A*x` has equal entries for any `x`. Entering

```
x = A\b
```

returns the least squares solution

```
x =

    1.5000
         0
         0
```

along with a warning that `A` is rank deficient. Note that `x` is not an exact solution:

```
A*x-b

ans =

    0.5000
   -0.5000
```

Data Type Support

When computing `X = A\B` or `X = A/B`, the matrices `A` and `B` can have data type `double` or `single`. The following rules determine the data type of the result:

- If both `A` and `B` have type `double`, `X` has type `double`.
- If either `A` or `B` has type single, `X` has type `single`.

Algorithm

The specific algorithm used for solving the simultaneous linear equations denoted by `X = A\B` and `X = B/A` depends upon the structure of the coefficient matrix `A`. To determine the structure of `A` and select the appropriate algorithm, MATLAB software follows this precedence:

1. **If A is sparse and diagonal**, `X` is computed by dividing by the diagonal elements of `A`.
2. **If A is sparse, square, and banded**, then banded solvers are used. Band density is (# nonzeros in the band)/(# nonzeros in a full band). Band density `= 1.0` if there are no zeros on any of the three diagonals.
   - If `A` is real and tridiagonal, i.e., band density `= 1.0`, and `B` is real with only one column, `X` is computed quickly using Gaussian elimination without pivoting.
   - If the tridiagonal solver detects a need for pivoting, or if `A` or `B` is not real, or if `B` has more than one column, but `A` is banded with band density greater than the `spparms` parameter `'bandden'` (default `= 0.5`), then `X` is computed using the Linear Algebra Package (LAPACK) routines in the following table.

	Real	Complex
`A` and `B` `double`	DGBTRF, DGBTRS	ZGBTRF, ZGBTRS
`A` or `B` `single`	SGBTRF, SGBTRS	CGBTRF, CGBTRS

3. **If A is an upper or lower triangular matrix**, then `X` is computed quickly with a backsubstitution algorithm for upper triangular matrices, or a forward substitution algorithm for lower triangular matrices. The check for triangularity is done for full matrices by testing for zero elements and for sparse matrices by accessing the sparse data structure.

   If A is a full matrix, computations are performed using the Basic Linear Algebra Subprograms (BLAS) routines in the following table.

	Real	Complex
`A` and `B` `double`	DTRSV, DTRSM	ZTRSV, ZTRSM
`A` or `B` `single`	STRSV, STRSM	CTRSV, CTRSM

4. **If A is a permutation of a triangular matrix**, then `X` is computed with a permuted backsubstitution algorithm.

5. **If A is symmetric, or Hermitian, and has real positive diagonal elements**, then a Cholesky factorization is attempted (see `chol`). If `A` is found to be positive definite, the Cholesky factorization attempt is successful and requires less than half the time of a general factorization. Nonpositive definite matrices are usually detected almost immediately, so this check also requires little time.

   If successful, the Cholesky factorization for full `A` is

   ```
   A = R'*R
   ```

   where `R` is upper triangular. The solution `X` is computed by solving two triangular systems,

   ```
   X = R\(R'\B)
   ```

   Computations are performed using the LAPACK routines in the following table.

	Real	Complex

A and B double	DLANSY, DPOTRF, DPOTRS, DPOCON	ZLANHE, ZPOTRF, ZPOTRS, ZPOCON
A or B single	SLANSY, SPOTRF, SPOTRS, SDPOCON	CLANHE, CPOTRF, CPOTRS, CPOCON

6. **If A is sparse**, then MATLAB software uses CHOLMOD to compute X. The computations result in
   ```
   P'*A*P = R'*R
   ```
   where P is a permutation matrix generated by amd, and R is an upper triangular matrix. In this case,
   ```
   X = P*(R\(R'\(P'*B)))
   ```
7. **If A is not sparse but is symmetric**, and the Cholesky factorization failed, then MATLAB solves the system using a symmetric, indefinite factorization. That is, MATLAB computes the factorization `P'*A*P=L*D*L'`, and computes the solution X by `X=P*(L'\(D\(L\(P*B))))`. Computations are performed using the LAPACK routines in the following table:

	Real	Complex
A and B double	DLANSY, DSYTRF, DSYTRS, DSYCON	ZLANHE, ZHETRF, ZHETRS, ZHECON
A or B single	SLANSY, SSYTRF, SSYTRS, SSYCON	CLANHE, CHETRF, CHETRS, CHECON

8. **If A is Hessenberg**, but not sparse, it is reduced to an upper triangular matrix and that system is solved via substitution.

9. **If A is square** and does not satisfy criteria 1 through 6, then a general triangular factorization is computed by Gaussian elimination with partial pivoting (see lu). This results in
   ```
   A = L*U
   ```
   where L is a permutation of a lower triangular matrix and U is an upper triangular matrix. Then X is computed by solving two permuted triangular systems.
   ```
   X = U\(L\B)
   ```
   If A is not sparse, computations are performed using the LAPACK routines in the following table.

	Real	Complex
A and B double	DLANGE, DGESV, DGECON	ZLANGE, ZGESV, ZGECON
A or B single	SLANGE, SGESV, SGECON	CLANGE, CGESV, CGECON

   If A is sparse, then UMFPACK is used to compute X. The computations result in
   ```
   P*(R\A)*Q = L*U
   ```
   where
   - P is a row permutation matrix
   - R is a diagonal matrix that scales the rows of A
   - Q is a column reordering matrix.

   Then `X = Q*(U\L\(P*(R\B)))`.

   > **Note** The factorization P*(R\A)*Q = L*U differs from the factorization used by the function lu, which does not scale the rows of A.

10. **If A is not square**, then Householder reflections are used to compute an orthogonal-triangular factorization.
    ```
    A*P = Q*R
    ```
    where P is a permutation, Q is orthogonal and R is upper triangular (see qr). The least squares solution X is computed with
    ```
    X = P*(R\(Q'*B))
    ```
    If A is sparse, MATLAB computes a least squares solution using the sparse qr factorization of A.

    If A is full, MATLAB uses the LAPACK routines listed in the following table to compute these matrix factorizations.

	Real	Complex
`A` and `B` `double`	DGEQP3, DORMQR, DTRTRS	ZGEQP3, ZORMQR, ZTRTRS
`A` or `B` `single`	SGEQP3, SORMQR, STRTRS	CGEQP3, CORMQR, CTRTRS

Note To see information about choice of algorithm and storage allocation for sparse matrices, set the `spparms` parameter `'spumoni' = 1`.

Note `mldivide` and `mrdivide` are not implemented for sparse matrices `A` that are complex but not square.

See Also

Arithmetic Operators, `linsolve`, `ldivide`, `rdivide`

Provide feedback about this page

mkpp mlint