

# Computing Fundamentals

Salvatore Filippone

`salvatore.filippone@uniroma2.it`

2012–2013

Recall arithmetic operators on arrays:

- Unary minus  $-$  (change sign);
- Addition/subtraction by a scalar  $v+1$ ;
- Multiplication/division by a scalar  $\alpha*v$ ,  $v/\beta$ ;
- Element-by-element operators  $+ - .* ./ .^$
- Comparison and logical operators  $< > <= >= \text{any find } | \&$

Look again: *Matrix Operators*

Matrix-matrix: product.

$$C = A*B$$

This is the classical multiplication of matrices as defined in linear algebra

$$C = AB \iff C_{ij} = \sum_k A_{ik} B_{kj}$$

- Map a linear space into another;
- Special cases: rotations, axis scaling, etc.

Rule: number of columns of first matrix must be same as number of rows of second.

The product is *not* commutative, given  $A*B$ ,  $B*A$  will be different or may not even exist at all!

Matrix transpose:  $B_{ij} = A_{ji}$

$$B = A'$$

Matrix exponentiation (by an integer):

$$B = A^n$$

requires  $A$  to be square.

Properties of matrix operators:

- Addition is commutative and associative:

$$A + B = B + A, \quad A + (B + C) = (A + B) + C$$

- Multiplication is associative but *not* commutative:

$$A * B \neq B * A, \quad A * (B * C) = (A * B) * C$$

- Multiplication is distributive on both sides:

$$A * (B + C) = A * B + A * C, \quad (A + B) * C = A * C + B * C$$

- Transpose and inversion of products:

$$(A * B)^T = (B^T) * (A^T), \quad (A * B)^{-1} = (B^{-1}) * (A^{-1})$$

Predefined matrices:

- `eye(m,n)` The identity (neutral element of multiplication);
- `zeros(m,n)` (neutral element of addition);
- `ones(m,n)`
- `rand(m,n)` (uniform distribution)
- `magic(n)`  $N \times N$  magic square

Division: what is division, anyway?

Division: what is division, anyway?

*Division is the inverse of the multiplication operation*



Division: what is division, anyway?

*Division is the inverse of the multiplication operation*

So, if we have

$$AB = C,$$

we can think of division as the operator that combines  $A$  and  $C$  to give back  $B$ . When  $A$  is square and non singular, this is *formally* equivalent to the multiplication by the inverse

$$B = A^{-1}C,$$

and this is in turn equivalent to the Octave/Matlab statement:

$$B = A \setminus C$$

From a purely abstract point of view division is thus equivalent to multiplication by the inverse. But this is not sufficient. Note that:

From a purely abstract point of view division is thus equivalent to multiplication by the inverse. But this is not sufficient. Note that:

- Multiplication by a matrix is non-commutative, therefore we can expect to have different *left* and *right* divisions;
- The inverse is well-defined for (non-singular) square matrices;

From a purely abstract point of view division is thus equivalent to multiplication by the inverse. But this is not sufficient. Note that:

- Multiplication by a matrix is non-commutative, therefore we can expect to have different *left* and *right* divisions;
- The inverse is well-defined for (non-singular) square matrices;
- Inverting a multiplication is equivalent to solving a linear system.

From a purely abstract point of view division is thus equivalent to multiplication by the inverse. But this is not sufficient. Note that:

- Multiplication by a matrix is non-commutative, therefore we can expect to have different *left* and *right* divisions;
- The inverse is well-defined for (non-singular) square matrices;
- Inverting a multiplication is equivalent to solving a linear system.

The last point is key to understanding the behaviour of Octave/Matlab matrix division operator, so we state it again:

- $X=A\backslash B$  is the same as computing the solution to  $AX = B$ ; and therefore
- $X=B/A$  is the same as computing the solution to  $XA = B$ ; but we also have  $B^T = (XA)^T = A^T X^T$ , that is  $X' = A'\backslash B'$ ; therefore  $B/A = (A'\backslash B')$

Let us keep going: from a formal point of view the left division

$$X=A\setminus B$$

is equivalent to the multiplication by the inverse

$$X = \mathbf{inv}(A)*B$$

Let us keep going: from a formal point of view the left division

$$X=A\backslash B$$

is equivalent to the multiplication by the inverse

$$X = \mathbf{inv}(A)*B$$

In practice you should *never* compute the inverse explicitly:

- It is slower, much slower;
- It is less accurate.

The second point would require a long digression into numerical analysis. For the first point, we need to understand how  $A\backslash B$  is actually computed.

Let us keep going: from a formal point of view the left division

$$X=A\setminus B$$

is equivalent to the multiplication by the inverse

$$X = \mathbf{inv}(A)*B$$

In practice you should *never* compute the inverse explicitly:

- It is slower, much slower;
- It is less accurate.

The second point would require a long digression into numerical analysis. For the first point, we need to understand how  $A\setminus B$  is actually computed. First step: if  $A\setminus B$  is equivalent to solving

$$AX = B$$

are there any matrices  $A$  that are easy to handle?



If a coefficient matrix is lower triangular it is easy to solve  $Lx = b$  by forward substitution:

If a coefficient matrix is lower triangular it is easy to solve  $Lx = b$  by forward substitution:

```
n=size(L,1);  
for i=1:n  
    x(i) = b(i) - L(i,1:i-1)*x(1:i-1);  
    x(i) = x(i) / L(i,i);  
end
```

If the diagonal is unitary, the division steps can be skipped. The total number of operations executed is  $\approx n^2$ .

Same reasoning applies to an *upper* triangular matrix  $U$  with back substitution.



Suppose we are able to decompose

$$A = LU$$

where  $L$  is lower triangular and  $U$  is upper triangular; then we have

$$Ax = b \Rightarrow x = U^{-1}L^{-1}b$$

or



Suppose we are able to decompose

$$A = LU$$

where  $L$  is lower triangular and  $U$  is upper triangular; then we have

$$Ax = b \Rightarrow x = U^{-1}L^{-1}b$$

or

$$y = L \backslash b;$$

$$x = U \backslash y;$$

for a cost (after the decomposition) of  $\approx 2n^2$  operations. (Remember: solving a triangular system is easy).



We want to factor  $A = LU$



We want to factor  $A = LU$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} l_{11} & & \\ l_{21} & l_{22} & \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ & u_{22} & u_{23} \\ & & u_{33} \end{pmatrix}$$



We want to factor  $A = LU$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} l_{11} & & \\ l_{21} & l_{22} & \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ & u_{22} & u_{23} \\ & & u_{33} \end{pmatrix}$$

Writing down the products and imposing equality:

$$\begin{pmatrix} a_{11} \\ a_{21} \\ a_{31} \end{pmatrix} = \begin{pmatrix} l_{11} \\ l_{21} \\ l_{31} \end{pmatrix} (u_{11}) \quad ( a_{12} \ a_{13} ) = (l_{11}) ( u_{12} \ u_{13} )$$



We want to factor  $A = LU$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} l_{11} & & \\ l_{21} & l_{22} & \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ & u_{22} & u_{23} \\ & & u_{33} \end{pmatrix}$$

Writing down the products and imposing equality:

$$\begin{pmatrix} a_{11} \\ a_{21} \\ a_{31} \end{pmatrix} = \begin{pmatrix} l_{11} \\ l_{21} \\ l_{31} \end{pmatrix} (u_{11}) \quad (a_{12} \ a_{13}) = (l_{11}) (u_{12} \ u_{13})$$

$$\begin{pmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} l_{22}u_{22} & l_{22}u_{23} \\ l_{32}u_{22} & l_{32}u_{23} + l_{33}u_{33} \end{pmatrix} + \begin{pmatrix} l_{21} \\ l_{31} \end{pmatrix} (u_{12} \ u_{13})$$





We want to factor  $A = LU$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} l_{11} & & \\ l_{21} & l_{22} & \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ & u_{22} & u_{23} \\ & & u_{33} \end{pmatrix}$$

Writing down the products and imposing equality:

$$\begin{pmatrix} a_{11} \\ a_{21} \\ a_{31} \end{pmatrix} = \begin{pmatrix} l_{11} \\ l_{21} \\ l_{31} \end{pmatrix} (u_{11}) \quad (a_{12} \ a_{13}) = (l_{11}) (u_{12} \ u_{13})$$

$$\begin{pmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} l_{22}u_{22} & l_{22}u_{23} \\ l_{32}u_{22} & l_{32}u_{23} + l_{33}u_{33} \end{pmatrix} + \begin{pmatrix} l_{21} \\ l_{31} \end{pmatrix} (u_{12} \ u_{13})$$

$n^2$  equations in  $n^2 + n$  unknowns; need additional constraints.

- Factor the diagonal (auxiliary constraint:  $l_{ii} = 1$ )

Compute  $( a_{11} ) \rightarrow ( l_{11} ) ( u_{11} )$

The total cost is  $\approx \frac{2}{3}n^3$ .

- Factor the diagonal (auxiliary constraint:  $l_{ii} = 1$ )

$$\text{Compute } \begin{pmatrix} a_{11} \end{pmatrix} \rightarrow \begin{pmatrix} l_{11} \end{pmatrix} (u_{11})$$

- Update the first column:

$$\begin{pmatrix} l_{21} \\ l_{31} \end{pmatrix} \leftarrow \begin{pmatrix} a_{21} \\ a_{31} \end{pmatrix} (u_{11})^{-1}$$

The total cost is  $\approx \frac{2}{3}n^3$ .

- Factor the diagonal (auxiliary constraint:  $l_{ii} = 1$ )

$$\text{Compute } \begin{pmatrix} a_{11} \end{pmatrix} \rightarrow \begin{pmatrix} l_{11} \end{pmatrix} (u_{11})$$

- Update the first column:

$$\begin{pmatrix} l_{21} \\ l_{31} \end{pmatrix} \leftarrow \begin{pmatrix} a_{21} \\ a_{31} \end{pmatrix} (u_{11})^{-1}$$

- Update the first row:

$$\begin{pmatrix} u_{12} & u_{13} \end{pmatrix} \leftarrow (l_{11})^{-1} \begin{pmatrix} a_{12} & a_{13} \end{pmatrix}$$

The total cost is  $\approx \frac{2}{3}n^3$ .

- Factor the diagonal (auxiliary constraint:  $l_{ii} = 1$ )

$$\text{Compute } \begin{pmatrix} a_{11} \end{pmatrix} \rightarrow \begin{pmatrix} l_{11} \end{pmatrix} (u_{11})$$

- Update the first column:

$$\begin{pmatrix} l_{21} \\ l_{31} \end{pmatrix} \leftarrow \begin{pmatrix} a_{21} \\ a_{31} \end{pmatrix} (u_{11})^{-1}$$

- Update the first row:

$$\begin{pmatrix} u_{12} & u_{13} \end{pmatrix} \leftarrow (l_{11})^{-1} \begin{pmatrix} a_{12} & a_{13} \end{pmatrix}$$

- Update the lower-right submatrix;

$$\begin{pmatrix} \hat{a}_{22} & \hat{a}_{23} \\ \hat{a}_{32} & \hat{a}_{33} \end{pmatrix} \leftarrow \begin{pmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{pmatrix} - \begin{pmatrix} l_{21} \\ l_{31} \end{pmatrix} \begin{pmatrix} u_{12} & u_{13} \end{pmatrix}$$

The total cost is  $\approx \frac{2}{3}n^3$ .

- Factor the diagonal (auxiliary constraint:  $l_{ii} = 1$ )

$$\text{Compute } \begin{pmatrix} a_{11} \end{pmatrix} \rightarrow \begin{pmatrix} l_{11} \end{pmatrix} (u_{11})$$

- Update the first column:

$$\begin{pmatrix} l_{21} \\ l_{31} \end{pmatrix} \leftarrow \begin{pmatrix} a_{21} \\ a_{31} \end{pmatrix} (u_{11})^{-1}$$

- Update the first row:

$$\begin{pmatrix} u_{12} & u_{13} \end{pmatrix} \leftarrow (l_{11})^{-1} \begin{pmatrix} a_{12} & a_{13} \end{pmatrix}$$

- Update the lower-right submatrix;

$$\begin{pmatrix} \hat{a}_{22} & \hat{a}_{23} \\ \hat{a}_{32} & \hat{a}_{33} \end{pmatrix} \leftarrow \begin{pmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{pmatrix} - \begin{pmatrix} l_{21} \\ l_{31} \end{pmatrix} \begin{pmatrix} u_{12} & u_{13} \end{pmatrix}$$

- Apply recursively to lower-right submatrix;

The total cost is  $\approx \frac{2}{3}n^3$ .



```
function [L, U]=lufact1(A)
    (nargin() == 1) || usage("[L,U] = lufact1(A)");

    m=size(A,1);  n=size(A,2);
    if ((m==0)|| (n==0))
        return
    end

    mn=min(m,n);
    for j=1:mn
%       A(j,j+1:n) = (1.0)\A(j,j+1:n);
        A(j+1:m,j) = A(j+1:m,j)/(A(j,j));
        A(j+1:m,j+1:n) = A(j+1:m,j+1:n) - A(j+1:m,j)*A(j,j+1:n);
    end

    if (nargout() < 1)
        ans = A
    elseif (nargout() == 1)
        L=A;
    elseif (nargout() > 1)
        L=tril(A,-1)+eye(mn);
        U=triu(A);
    end
end

end
```

However we have always assumed we can divide by  $u_{jj}$ ; what if we find a zero value?



However we have always assumed we can divide by  $u_{jj}$ ; what if we find a zero value?

*When an element on the diagonal is zero, search for a nonzero in its column, and swap the relevant rows*

This is equivalent to applying  $P$

$$PA = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1.5 \\ 2 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 1.5 \end{pmatrix}$$

However we have always assumed we can divide by  $u_{jj}$ ; what if we find a zero value?

*When an element on the diagonal is zero, search for a nonzero in its column, and swap the relevant rows*

This is equivalent to applying  $P$

$$PA = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1.5 \\ 2 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 1.5 \end{pmatrix}$$

Thus we are computing  $PA = LU$  to get at the solution

$$Ax = b \Rightarrow x = U^{-1}L^{-1}Pb$$

Extension: what is zero? **Always** search for the coefficient with largest absolute value!

# LU Factorization: Pivoting

```
function [L, U, P]=lupfact1(A)
    (nargin() ==1) || usage(" [L,U,P] = lupfact1(A)");
    m=size(A,1);  n=size(A,2);
    if ((m==0)|| (n==0))
        return
    end

    mn=min(m,n);  lp=eye(m);
    for j=1:mn
        [mx, ix] = max(abs(A(j:m,j)));  ix=ix+j-1;
        tmp(1:n) = A(j,1:n);  A(j,1:n) = A(ix,1:n);  A(ix,1:n) = tmp(1:n);
        tmp(1:m) = lp(j,1:m);  lp(j,1:m) = lp(ix,1:m);  lp(ix,1:m) = tmp(1:m);

%       A(j,j+1:n) = (1.0)\A(j,j+1:n);
        A(j+1:m,j) = A(j+1:m,j)/(A(j,j));
        A(j+1:m,j+1:n) = A(j+1:m,j+1:n) - A(j+1:m,j)*A(j,j+1:n);
    end

    if (nargout() < 1)
        ans = A
    elseif (nargout() == 1)
        L = A;
    elseif (nargout() >= 2)
        L=tril(A,-1);  L(1:mn,1:mn)=L(1:mn,1:mn)+eye(mn);  U=triu(A);
        if (nargout()>2)
            P=lp;
        end
    end
end
end
```

Whenever you apply the division operator

```
x=A\b;
```

this is what Octave/Matlab does internally:

```
[L,U,P] = lu(A);
```

```
z=P*b;
```

```
y=L\z;
```

```
x=U\y;
```

The most expensive part is the invocation of the `lu` function ( $2/3n^3$ ).

If you are solving multiple linear systems with the same matrix, it is more efficient to do:

```
[L,U,P] = lu(A);  
for j=1:k  
    b = rhs(j); % Create the next RHS  
    z=P*b;  
    y=L\z;  
    x=U\y;  
    % do something with x  
end
```

What about  $\text{inv}(A)$ ? Actually this is computed with

$$A^{-1} = U^{-1}L^{-1}P$$

No wonder it costs more: it *starts* with LU factorization, then goes on with more computations!

*Never explicitly form  $\text{inv}(A)*B$ , always use  $A\backslash B$ .*

Cases where you *really* need the inverse exist but are (vanishingly) rare.

# Matrix division (reloaded)

So far we have only ever handled cases where  $A$  is square: since you have to solve a linear system, trying the `\` operator with a rectangular matrix will throw an error. Right?

## Matrix division (reloaded)

So far we have only ever handled cases where  $A$  is square: since you have to solve a linear system, trying the `\` operator with a rectangular matrix will throw an error. Right? Well, let's try:

```
octave:20> a = [1 , 2 ; 3 , 4 ; 5 , 6 ;]
```

```
a =
```

```
1    2
```

```
3    4
```

```
5    6
```

```
octave:21> b = [3 , 4 , 5] ';
```

```
octave:22> x = a \ b
```



## Matrix division (reloaded)

So far we have only ever handled cases where  $A$  is square: since you have to solve a linear system, trying the `\` operator with a rectangular matrix will throw an error. Right? Well, let's try:

```
octave:20> a=[1,2;3,4;5,6;]
```

```
a =
```

```
  1   2
```

```
  3   4
```

```
  5   6
```

```
octave:21> b=[3,4,5]';
```

```
octave:22> x=a\b
```

```
x =
```

```
 -2.0000
```

```
  2.5000
```

What's going on here?

Go back to the beginning and say it again:

*Applying the matrix division operator  $x=A\backslash b$  is equivalent to solving the linear system  $Ax = b$*

Go back to the beginning and say it again:

*Applying the matrix division operator  $x=A\backslash b$  is equivalent to solving the linear system  $Ax = b$*

So, to define division for a *rectangular* matrix we need to know what to do with a linear system:

$$Ax = b$$

when  $A$  is  $m \times n$  and  $m \neq n$  (i.e.: rectangular).

Linear algebra comes to the rescue:

*When  $A$  is not square, we can minimize the mismatch between RHS and LHS, i.e. we search for a least-squares solution*

$$Ax = b \Rightarrow \min_x \|Ax - b\|_2$$

When  $A$  is square and non-singular this reduces to a “normal” system solution.

The least squares solution is computed through the  $QR$  factorization

$$A = QR$$

where  $Q$  is orthogonal (i.e.  $QQ^T = Q^TQ = I$ ) and  $R$  is upper triangular (trapezoidal).

$$[Q, R] = \text{qr}(A);$$

# Matrix division (reloaded)

Here are the full rules for

$$X = A \setminus B;$$

When  $A$  is a scalar, this is simply a term-by-term division;

# Matrix division (reloaded)

Here are the full rules for

$$X = A \setminus B;$$

When  $A$  is a scalar, this is simply a term-by-term division; otherwise:

- The matrices  $A$  and  $B$  must have the same number of rows;
- The number of rows of  $X$  equals the number of columns of  $A$ ;
- The number of columns of  $X$  equals the number of columns of  $B$ ;
- The solution always exists in the least squares sense, not necessarily in the usual matrix inversion sense; moreover, it is not necessarily unique;

# Matrix division (reloaded)

Here are the full rules for

$$X = A \setminus B;$$

When  $A$  is a scalar, this is simply a term-by-term division; otherwise:

- The matrices  $A$  and  $B$  must have the same number of rows;
- The number of rows of  $X$  equals the number of columns of  $A$ ;
- The number of columns of  $X$  equals the number of columns of  $B$ ;
- The solution always exists in the least squares sense, not necessarily in the usual matrix inversion sense; moreover, it is not necessarily unique;

The *right* division  $X=B/A$  is equivalent to

$$X = (A' \setminus B')';$$

from which we can derive the row/columns constraints.

## An example: curve fitting

Suppose you are measuring a physical phenomenon: you get a set of points  $(x_i, y_i)$  and you make a guess: they lie on a parabola. You should have

$$y_i = ax_i^2 + bx_i + c, \quad i = 1, \dots, n$$

for some (unknown) coefficients  $a, b, c$ . However measurements have *noise*, so the equations will *not* be satisfied exactly. What do you do?



## An example: curve fitting

Suppose you are measuring a physical phenomenon: you get a set of points  $(x_i, y_i)$  and you make a guess: they lie on a parabola. You should have

$$y_i = ax_i^2 + bx_i + c, \quad i = 1, \dots, n$$

for some (unknown) coefficients  $a, b, c$ . However measurements have *noise*, so the equations will *not* be satisfied exactly. What do you do?

$$C = XX \setminus Y$$

where

$$XX = \begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ \vdots & \vdots & \vdots \\ x_n^2 & x_n & 1 \end{pmatrix}$$

The coefficients will give the *best fit* parabola. BTW: use the Vandermonde matrix function

$$XX = \text{vander}(x, 3)$$