LU_decomposition

November 7, 2022

1 LU decomposition

1.0.1 Matrices and Vectors In octave

In octave vectors and matrices are written in a similar way, to separate elements of a row we simply leave a space between them, to get to the next row we simply use a semicolon ;. Knowing this we may define row vectors, column vectors and matrices:

[1]:
```
displayformat matrix latex; ## Command only works in xeus-octave, and it's␣
↪meant for pretty printing, if you are using a different
#version or matlab this will give an error
v=[0 1 2 3 4] # row vector
```

$$v = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \end{bmatrix}$$

[2]:
```
v=[0 ;1 ;2 ;3; 4] # column vector
```

$$v = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

[4]:
```
m=[0 1 2 3;4 5 6 7; 8 9 10 11;12 13 14 15]# matrix
```

$$m = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix}$$

In octave transposition is done with the ' character, one thing to be aware about is that this also the dagger operation meaning that for complex matrices it also performs conjugation

[5]:
```
p=[1 0 0 0; 0 1 0 0; 0 0 0 1; 0 0 1 0]
```

$$p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

```
[6]: p'
```

$$ans = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Finding eigenvalues, eigenvectors and so on in octave is fairly straightforward

The determinant is obtained with the det function, eigevalues and eigenvectors with eig

```
[7]: det(p)
```

```
ans = -1
```

```
[8]: eig(p) #eigenvalues by default
```

$$ans = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

```
[9]: [eivects,eigvals]=eig(p)
```

$$eivects = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ -0.707107 & 0 & 0.707107 & 0 \\ 0.707107 & 0 & 0.707107 & 0 \end{bmatrix}$$

```
eigvals = Diagonal Matrix

  -1    0    0    0
   0    1    0    0
   0    0    1    0
   0    0    0    1
```

Eigenvectors come as a matrix so one needs to do little matrix indexing to recover the actual vectors, In octave one can index elements of a matrix using parenthesis

A(row,column)

in octave : means all elements so for example the element A_{1,1}

may be obtained by A(1,1)

While the first row of A may be obtained by A(:,1)

[10]:
```
eivects(:,1)
```

$$ans = \begin{bmatrix} 0 \\ 0 \\ -0.707107 \\ 0.707107 \end{bmatrix}$$

We check that it is indeed an eigenvector

[11]:
```
p*eivects(:,1)
```

$$ans = \begin{bmatrix} 0 \\ 0 \\ 0.707107 \\ -0.707107 \end{bmatrix}$$

The upper bound on the error of numerical linear algebra calculations is the so called condition number which is defined as the ratio of the maximal and singular values of a matrix, if the condition number is big with respect to 1 (by several orders of magnitude, one order of magnitude roughly translates in loosing a digit of accuracy), then we might expect incorrect results from whatever calculations we do with this matrix

[12]:
```
cond(p)
```

```
ans = 1
```

[13]:
```
cond(m) # m is a singular matrix so a high condition number is expected
```

```
ans = 4.3520e+17
```

1.1 Linear Systems Basic Gaussian elimination

Consider the system of equations

$$\begin{aligned} x + 2y + 3z &= 4 \\ -2y - 4z &= 6 \\ x - y &= 0 \end{aligned}$$

We may write in augmented matrix form in octave simply by:

```
[14]: B=[1 2 3 4;0 -2 -4 6;1 -1 0 0]
```

$$B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & -2 & -4 & 6 \\ 1 & -1 & 0 & 0 \end{bmatrix}$$

```
[15]: A=B(:,1:3); #we extract the system of equations
      b=B(:,4); # and the results from the augmented matrix to use them later
```

Now we may simply start doing gaussian elimination, the idea is to make all elements but the ones in the diagonal and last row zero, by performing elementary operations on rows (Multiplication, division and addition to other rows), to get some practice on indexing let us do this in octave

```
[16]: B(3,:)=-B(3,:)+B(1,:) #Multiply last row by minus one and add the first row to␣
       ↪it
```

$$B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & -2 & -4 & 6 \\ 0 & 3 & 3 & 4 \end{bmatrix}$$

```
[17]: B(3,:) =( B(3,:) + (3/2) *B(2,:))/(-3) # Change the third row by adding to it 3/
       ↪2 times the second row
      # and multiplying the result by -1/3
```

$$B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & -2 & -4 & 6 \\ -0 & -0 & 1 & -4.33333 \end{bmatrix}$$

```
[18]: B(2,:)=((1/4)*B(2,:)+B(3,:))*-2 # Change the second row by multiplying it by -1/
       ↪4 and add the third row to it
      # multiply everything by -2 in the end
```

$$B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ -0 & 1 & -0 & 5.66667 \\ -0 & -0 & 1 & -4.33333 \end{bmatrix}$$

```
[19]: B(1,:)=B(1,:)-3*B(3,:) # Change the first row by adding the last one times -3
```

$$B = \begin{bmatrix} 1 & 2 & 0 & 17 \\ -0 & 1 & -0 & 5.66667 \\ -0 & -0 & 1 & -4.33333 \end{bmatrix}$$

[20]:
```
B(1,:)=B(1,:)-2*B(2,:) # Change the first row by adding the second one times -2
```

$$B = \begin{bmatrix} 1 & 0 & 0 & 5.66667 \\ -0 & 1 & -0 & 5.66667 \\ -0 & -0 & 1 & -4.33333 \end{bmatrix}$$

By performing Gaussian elimination we found x=5.6667=17/3=y and z=-13/3=-4.33333

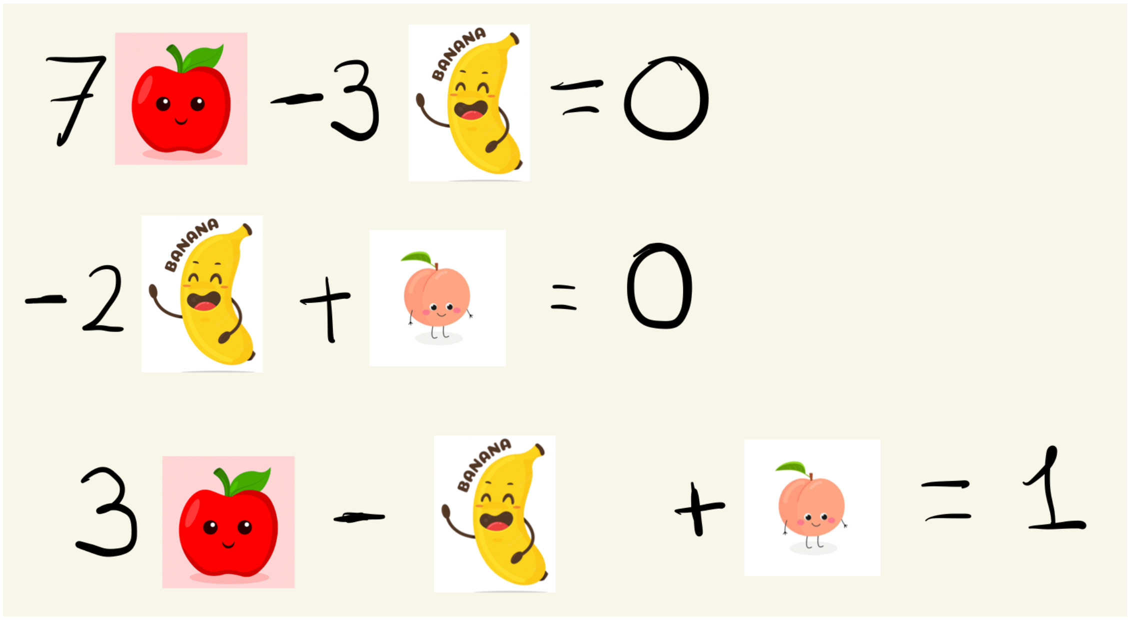
Using Left Division In Octave the built in operation for solving linear systems is called Left division, it works on systems of the form $A\vec{x} = \vec{b}$, What left division is equivalent to is $A^{-1}\vec{b}$, and it is written as A, so we may solve the same system by this simple command

[21]:
```
A\b
```

$$ans = \begin{bmatrix} 5.66667 \\ 5.66667 \\ -4.33333 \end{bmatrix}$$

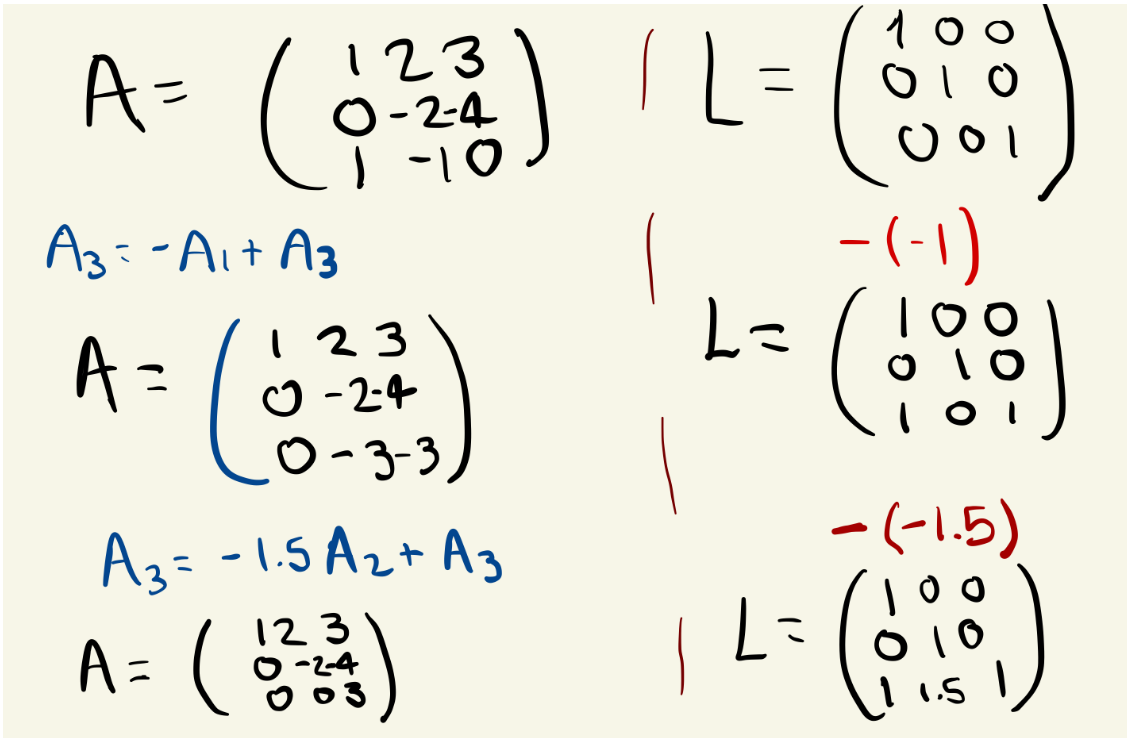
We indeed see the same result

1.1.1 Task 1 Write the next system in augmented matrix form, and perform Gaussian elimination, check your answer using left division . What is the condition number of the system? can you trust the results you obtain from numerical computation?



1.2 The LU decomposition

The idea of this matrix decomposition is to write the matrix A as $A = LU$ where U is an upper triangular matrix and L is a lower triangular matrix, How to get one is similar to Gaussian elimination, let us break down the procedure, which is quite similat to the gaussian elimination we performed before, we begin by the initial matrix A, and try to make it upper diagonal by gaussian elimnation, that will be our matrix U, we however need to keep track of the factors we multiply our rows by so that we obtain zeros below the diagonal, we place minus those factors on the same position of the zeros they caused on an identity matrix, that will be our matrix L perhaps the next example will make it clearer



$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & -2 & -4 \\ 1 & -1 & 0 \end{pmatrix}$
$L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
$A_3 = -A_1 + A_3$
$-(-1)$
$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & -2 & -4 \\ 0 & -3 & -3 \end{pmatrix}$
$L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$
$A_3 = -1.5 A_2 + A_3$
$-(-1.5)$
$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & -2 & -4 \\ 0 & 0 & 3 \end{pmatrix}$
$L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1.5 & 1 \end{pmatrix}$

[22]:
```
U=[1 2 3;0 -2 -4;0 0 3]
```

$$U = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & -4 \\ 0 & 0 & 3 \end{bmatrix}$$

[23]:
```
L=[1 0 0;0 1 0;1 1.5 1]
```

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1.5 & 1 \end{bmatrix}$$

[24]:
```
L*U #Checking the product gives A
```

$$ans = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & -4 \\ 1 & -1 & 0 \end{bmatrix}$$

The Lu decompostion is useful because it allows to solve determinants quite easily (as oposed to laplace expansion as we discussed in the lecture). Let us briefly remember why:

We may rewrite the determinant of A as

$det(A) = det(LU) = det(L)det(U)$

In general the determinant of a matrix $n \times n$ requires us to compute n determinants of $(n-1) \times (n-1)$ matrices so you end up with something proportional to n! operations (if this is not clear refresh laplace's expansion)

On the other hand for the Lu decomposition you need to perform $2n$ operations (n multiplications and n aditions) on n-1 rows to get the elements on the first column to be zeros, then $(2n-1)$ operations on $n-2$ columns to get the zeros on the second column and so on,the total of operations is then

$\sum_{i=0}^{n} 2(n-i)(n-1+i) \approx \int 2n^2 dn \approx \frac{2}{3}n^3$

and the determinant of a triangular matrix is just the product of the diagonal elements, so then we need to 2 n multiplications and one addition the number of steps then scales roughly as

$\frac{4}{3}n^4$

To get the determinant of a $n ×n $, let us see how different the number of operations would be for n=30?

[25]:
```
4*(30^4)/3
```

```
ans = 1080000
```

[26]:
```
factorial(30)
```

```
ans = 2.6525e+32
```

Task Produce a plot of the time it would take to compute the determinant for each approach against n (up to 100), suppose each operation takes 10^{-9} seconds, use days as your measure of time

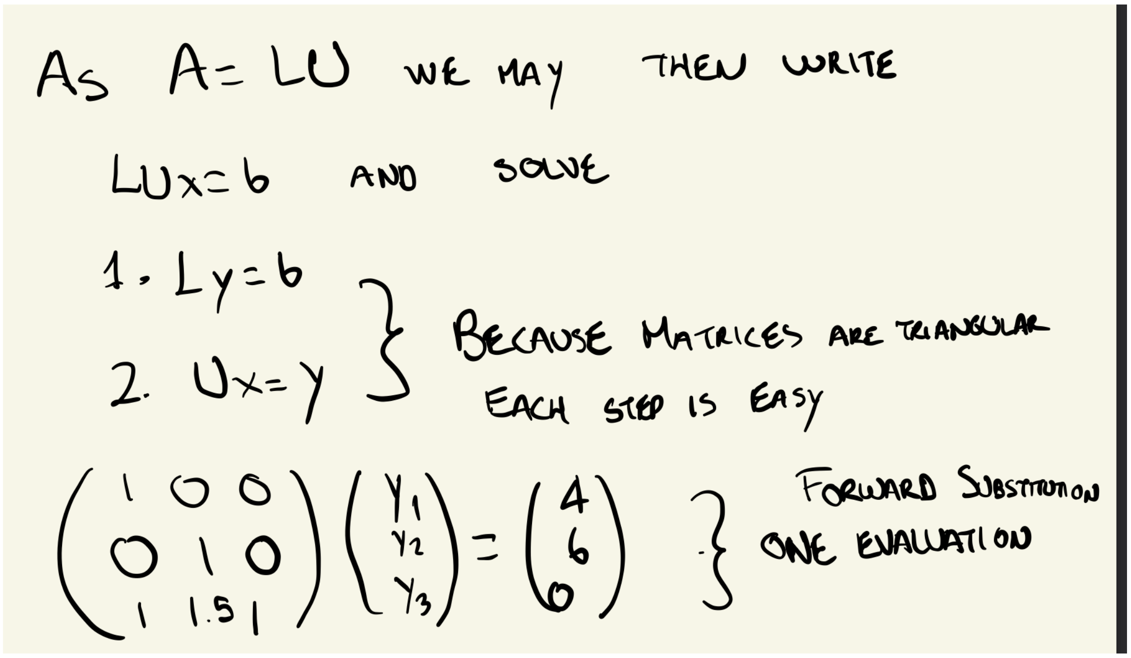
We may find the product of the diagonal elements via a for loop

[27]:
```
detL=1;
for i=1:size(L)(1);
    detL*=L(i,i);
end
detL
```

```
detL = 1
```

Task Find the determinant of det(A) using the LU decomposition (without using the built in function det()) later compare with the result from det

The other main application of the LU decomposition is solving linear systems. To see how let us follow our example



AS $A = LU$ WE MAY THEN WRITE

$LUx = b$ AND SOLVE

$$\left.\begin{array}{l} 1.\ Ly = b \\ 2.\ Ux = y \end{array}\right\}$$ BECAUSE MATRICES ARE TRIANGULAR EACH STEP IS EASY

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1.5 & 1 \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \\ 0 \end{pmatrix} \Bigg\}$$ FORWARD SUBSTITUTION ONE EVALUATION

As we can see once we have LU decomposition everything reduces to forward and backward substitution, below we can find an implementation for forward substitution

[28]:
```
function x = forward(L,x)
% FORWARD. Forward elimination.
% For lower triangular L, x = forward(L,b) solves L*x = b
n= size(L)(1);
for k = 1:n
j = 1:k-1;
x(k) = (x(k) - L(k,j)*x(j))/L(k,k); # We always divide by the diagonal on the kth row-> so we have y=a and not by=a
end
end
```

[29]:
```
forward(L,[4;6;0])
```

$$ans = \begin{bmatrix} 4 \\ 6 \\ -13 \end{bmatrix}$$

Task Implement an algorithm that does the backward substitution, use it to finish solving the linear system (step 2)

[30]:
```
function x = backward(U,b)
```

```
end
```

[31]:
```
backward(U,[4;6;-13])
```

$$ans = \begin{bmatrix}5.66667 & 5.66667 & -4.33333\end{bmatrix}$$

Task: dolittle's algorithm is a way of implementing the LU decomposition and it's often the algoritm used by libraries the basic idea is that the elements of the U matrix are given by:

$$\forall j$$
$$for\ i = 0 \rightarrow U_{ij} = A_{ij}$$
$$for\ i > 0 \rightarrow U_{ij} = A_{ij} - \sum_{k=0}^{i-1} L_{ik}U_{kj}$$
$$\forall i$$
$$for\ j = 0 \rightarrow L_{ij} = \frac{A_{ij}}{U_{ij}}$$
$$for\ j > 0 \rightarrow L_{ij} = \frac{A_{ij}}{U_{ij}} - \sum_{k=0}^{j-1} L_{ik}U_{kj}$$

Complete the template below to obtain a function that gives the LU decomposition, or implement your own

[32]:
```
function [lower,upper]=dolittle(A)
n=size(A)(1);
lower=zeros(n,n);
upper=eye(n,n); # Identity matrix
for i=1:n
    for k=i:n
    suma1=0;
    suma2=0;
        for j=1:i
        suma1= #sum for i>0
        suma2= # sum for j>0
        end
    upper(i,k)=;#U_{ij} for i>0
    lower(k,i)=; #L_{ij} for j>0
    end
end
end
```

[33]:
```
[l,u]=dolittle(A)
```

$$l = \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\1 & 1.5 & 1\end{bmatrix}$$

$$u = \begin{bmatrix}1 & 2 & 3\\0 & -2 & -4\\0 & 0 & 3\end{bmatrix}$$

[34]:
```
l*u
```

$$ans = \begin{bmatrix}1 & 2 & 3\\0 & -2 & -4\\1 & -1 & 0\end{bmatrix}$$

PD: The LU decomposition is not unique and octave's built in function uses permutation matrices which we did not cover, in this case LU=PA (If you just ask for two outputs it will multiply $P^{-1}L$)

[35]:
```
[l,u,p]=lu(A)
```

$$l = \begin{bmatrix}1 & 0 & 0\\1 & 1 & 0\\0 & 0.666667 & 1\end{bmatrix}$$

$$u = \begin{bmatrix}1 & 2 & 3\\0 & -3 & -3\\0 & 0 & -2\end{bmatrix}$$

```
p = Permutation Matrix

   1   0   0
   0   0   1
   0   1   0
```

[36]:
```
l*u
```

$$ans = \begin{bmatrix}1 & 2 & 3\\1 & -1 & 0\\0 & -2 & -4\end{bmatrix}$$

[37]:
```
p*A
```

$$ans = \begin{bmatrix}1 & 2 & 3\\1 & -1 & 0\\0 & -2 & -4\end{bmatrix}$$

1.3 Inverse using LU

To find the inverse of a matrix using the LU decomposition is not different from solving a linear system using it, to see why let us do a little rewriting:

$$A = LU \to LUA^{-1} = \mathcal{I}$$

Now we may break this problem into three linear systems by noting that

$$\mathcal{I} = \begin{pmatrix} e_1 & e_2 & e_3 & \dots & e_n \end{pmatrix}$$

Where $e_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, e_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \dots, e_n = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$

So that if A_i^{-1} are each of the columns of the inverse matrix then we need to solve the systems:

$$\begin{aligned} LUA_1^{-1} &= e_1 \\ LUA_2^{-1} &= e_2 \\ &\vdots \\ LUA_n^{-n} &= e_n \end{aligned}$$

We finally join the A_{i}^{-1} and obtain the inverse matrix, let us illustrate that with our example

We start with $LUA_1^{-1} = e_1$

[38]:
```
forward(L,[1;0;0])
```

$$ans = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

[39]:
```
a1=backward(U,[1;0;-1])
```

$$a1 = \begin{bmatrix} 0.666667 & 0.666667 & -0.333333 \end{bmatrix}$$

We repeat the process with the next column

[40]:
```
forward(L,[0;1;0])
```

$$ans = \begin{bmatrix} 0 \\ 1 \\ -1.5 \end{bmatrix}$$

[41]:
```
a2=backward(U,[0;1;-1.5])
```

$$a2 = \begin{bmatrix} 0.5 & 0.5 & -0.5 \end{bmatrix}$$

and finally we do this for the last column

[42]:
```
forward(L,[0;0;1])
```

$$ans = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

[43]:
```
a3=backward(U,[0;0;1])
```

$$a3 = \begin{bmatrix} 0.333333 & -0.666667 & 0.333333 \end{bmatrix}$$

Finally we join the results

[44]:
```
inverse=[a1;a2;a3]'
```

$$inverse = \begin{bmatrix} 0.666667 & 0.5 & 0.333333 \\ 0.666667 & 0.5 & -0.666667 \\ -0.333333 & -0.5 & 0.333333 \end{bmatrix}$$

[45]:
```
A*inverse
```

$$ans = \begin{bmatrix} 1 & 0 & -5.55112\text{ -17} \\ 0 & 1 & 0 \\ 1.11022\text{ -16} & 0 & 1 \end{bmatrix}$$

Task Using the functions in this notebook (no built in libraries) create a function that finds the inverse of a matrix using the LU decomposition, test it on the little system with fruits from one of the previous tasks