LU Decomposition

In [1]:
```
import pyJvsip as pv
f='%.5f'
```

Solve using LU Class

Create some data A x = b
Note we create an x and calculate a b directly

In [2]:
```
n=5
A=pv.create('mview_d',n,n).fill(0.0)
A.block.vector.randn(5)
x=pv.create('vview_d',n).randn(9)
print('Matrix A');A.mprint(f)
print('Known x vector');x.mprint(f)
b=A.prod(x)
print('Calculated b=Ax vector');b.mprint(f)
```

```
Matrix A
[ 0.50802  0.53515  0.69864 -0.96027  0.23142;
  0.04042 -0.47661  0.20765  0.50621 -0.38285;
  0.15746  0.78115 -0.96815 -0.32034  0.79250;
  0.79172 -0.25782  0.12663  1.35454  0.25523;
 -0.19459  0.34111 -0.49602  0.17191  1.62412]

Known x vector
[ 0.39248 -1.35556 -0.24268  1.22453 -0.65029]

Calculated b=Ax vector
[-2.02196  1.48038 -1.66977  2.12221 -1.26404]
```

Note LU overwrites the input matrix; so to preserve our original matrix we use a copy. The LU object will keep a reference to the copy (which means python wont garbage collect it).

In [3]:
```
u,s,v=A.copy.svd
s.mprint(f)
xe=v.prod(u.transview.prod(b)/s)
x.mprint(f)
xe.mprint(f)
print('%.5e'%(xe-x).normFro)
```

```
[ 3.58566  2.94871  2.15187  2.04869  1.54947  0.99577  0.65505
0.07359]

[ 0.39248 -1.35556 -0.24268  1.22453 -0.65029  0.30630  2.11901
0.81714]

[ 0.39248 -1.35556 -0.24268  1.22453 -0.65029  0.30630  2.11901
0.81714]

4.34053e-15
```

First we solve using the LU class directly.

Note LU.luSel is a dictionary which lets you select the LU decomposition type using the matrix type

In [14]:
```
print('Example of LU.luSel: %s'%pv.LU.luSel[A.type])
luObj = pv.LU(pv.LU.luSel[A.type],n)
_=luObj.decompose(A.copy)
print('Solve for x using b. Done in place. Here we make a copy of b
xb = b.copy
luObj.solve(pv.VSIP_MAT_NTRANS,xb).mprint(f)
print('Calculate an error using (x-xb).normFro %.5e:'%(x-xb).normFro
```

```
Example of LU.luSel: lu_d
Solve for x using b. Done in place. Here we make a copy of b first.
[ 0.39248 -1.35556 -0.24268  1.22453 -0.65029]

Calculate an error using (x-xb).normFro 1.31158e-15:
```

In pyJvsip a method is defined on matrix views which will create the LU object for you. We do the same problem.

In [15]:
```
xb=b.copy
luObj=A.copy.lu
luObj.solve(pv.VSIP_MAT_NTRANS,xb).mprint(f)
print('Calculate an error using (x-xb).normFro %.5e:'%(x-xb).normFro
```

```
[ 0.39248 -1.35556 -0.24268  1.22453 -0.65029]

Calculate an error using (x-xb).normFro 1.31158e-15:
```

For a simple solver we can also just solve directly. If we wanted to solve using matrix operator 'HERM' or 'TRANS' then we would need the more complicated version.

In [16]:
```
xb=b.copy
A.copy.luSolve(xb).mprint(f)
print('Calculate an error using (x-xb).normFro %.5e:'%(x-xb).normFro
```

```
[ 0.39248 -1.35556 -0.24268  1.22453 -0.65029]

Calculate an error using (x-xb).normFro 1.31158e-15:
```

We also have a pyJvsip method to calculate an inverse using the LU methods.

In [17]:
```
Ainv=A.copy.luInv
Ainv.mprint(f)
A.mprint(f)
A.prod(Ainv).mprint(f)
```

```
[ 1.71577  6.37607  2.44288 -0.60934  0.16226;
 -2.21403 -12.03181 -3.18940  2.34171 -1.33243;
 -0.43436 -4.59340 -2.15392  0.91370 -0.11345;
 -1.51525 -6.06420 -1.92946  1.53927 -0.51398;
  0.69832  2.53000  0.50896 -0.44871  0.93476]

[ 0.50802  0.53515  0.69864 -0.96027  0.23142;
  0.04042 -0.47661  0.20765  0.50621 -0.38285;
  0.15746  0.78115 -0.96815 -0.32034  0.79250;
  0.79172 -0.25782  0.12663  1.35454  0.25523;
 -0.19459  0.34111 -0.49602  0.17191  1.62412]

[ 1.00000 -0.00000 -0.00000  0.00000  0.00000;
 -0.00000  1.00000 -0.00000 -0.00000 -0.00000;
 -0.00000  0.00000  1.00000  0.00000  0.00000;
 -0.00000  0.00000 -0.00000  1.00000  0.00000;
  0.00000  0.00000  0.00000  0.00000  1.00000]
```