LU with Partial Pivoting

In [131]:

```
#keep
import numpy as np
import numpy.linalg as la

np.set_printoptions(precision=3, suppress=True)
```

Set-up

Let's grab a (admittedly well-chosen) sample matrix `A`:

In [132]:

```
#keep
n = 4

np.random.seed(235)
A = np.round(5*np.random.randn(n, n))
A[0,0] = 0
A[2,1] = 17
A[0,2] = 19
A
```

Out[132]:

```
array([[  0.,   4.,  19.,  -7.],
       [ -1.,  -2., -10.,  -0.],
       [  1.,  17.,   1.,  -4.],
       [ -5.,  -8.,  -6.,  -2.]])
```

Permutation matrices

Now define a function `row_swap_mat(i, j)` that returns a permutation matrix that swaps row i and j:

In [133]:

```
def row_swap_mat(i, j):
    P = np.eye(n)
    P[i] = 0
    P[j] = 0
    P[i, j] = 1
    P[j, i] = 1
    return P
```

What do these matrices look like?

In [134]:

```
#keep
row_swap_mat(0,1)
```

Out[134]:

```
array([[ 0.,  1.,  0.,  0.],
       [ 1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
```

Do they work?

In [135]:

```
row_swap_mat(0,1).dot(A)
```

Out[135]:

```
array([[ -1.,  -2., -10.,   0.],
       [  0.,   4.,  19.,  -7.],
       [  1.,  17.,   1.,  -4.],
       [ -5.,  -8.,  -6.,  -2.]])
```

Part I

`U` is the copy of `A` that we'll modify:

In [136]:

```
#keep
U = A.copy()
```

First column

Create P1 to swap up the right row:

In [137]:

```
P1 = row_swap_mat(0, 3)
U = P1.dot(U)
U
```

Out[137]:

```
array([[ -5.,  -8.,  -6.,  -2.],
       [ -1.,  -2., -10.,   0.],
       [  1.,  17.,   1.,  -4.],
       [  0.,   4.,  19.,  -7.]])
```

In [138]:

```
M1 = np.eye(n)
M1[1,0] = -U[1,0]/U[0,0]
M1[2,0] = -U[2,0]/U[0,0]
M1
```

Out[138]:

```
array([[ 1. ,  0. ,  0. ,  0. ],
       [-0.2,  1. ,  0. ,  0. ],
       [ 0.2,  0. ,  1. ,  0. ],
       [ 0. ,  0. ,  0. ,  1. ]])
```

In [139]:

```
#keep
U = M1.dot(U)
U
```

Out[139]:

```
array([[ -5. ,  -8. ,  -6. ,  -2. ],
       [  0. ,  -0.4,  -8.8,   0.4],
       [  0. ,  15.4,  -0.2,  -4.4],
       [  0. ,   4. ,  19. ,  -7. ]])
```

Second column

Create `P2` to swap up the right row:

In [140]:

```
P2 = row_swap_mat(2,1)
U = P2.dot(U)
U
```

Out[140]:

```
array([[ -5. ,   -8. ,   -6. ,   -2. ],
       [  0. ,  15.4,  -0.2,  -4.4],
       [  0. ,  -0.4,  -8.8,   0.4],
       [  0. ,   4. ,  19. ,  -7. ]])
```

Make the second-column elimination matrix `M2`:

In [141]:

```
#keep
M2 = np.eye(n)
M2[2,1] = -U[2,1]/U[1,1]
M2[3,1] = -U[3,1]/U[1,1]
M2
```

Out[141]:

```
array([[ 1.   ,  0.   ,  0.   ,  0.   ],
       [ 0.   ,  1.   ,  0.   ,  0.   ],
       [ 0.   ,  0.026,  1.   ,  0.   ],
       [ 0.   , -0.26 ,  0.   ,  1.   ]])
```

In [142]:

```
#keep
U = M2.dot(U)
U
```

Out[142]:

```
array([[ -5.   ,  -8.   ,  -6.   ,  -2.   ],
       [  0.   ,  15.4  ,  -0.2  ,  -4.4  ],
       [  0.   ,   0.   ,  -8.805,   0.286],
       [  0.   ,   0.   ,  19.052,  -5.857]])
```

Third column

Create `P3` to swap up the right entry:

In [143]:

```
#keep
P3 = row_swap_mat(3, 2)
U = P3.dot(U)
U
```

Out[143]:

```
array([[ -5.   ,  -8.   ,  -6.   ,  -2.   ],
       [  0.   ,  15.4  ,  -0.2  ,  -4.4  ],
       [  0.   ,   0.   ,  19.052,  -5.857],
       [  0.   ,   0.   ,  -8.805,   0.286]])
```

Make the third-column elimination matrix `M3`:

In [144]:

```
#keep
M3 = np.eye(n)
M3[3,2] = -U[3,2]/U[2,2]
M3
```

Out[144]:

```
array([[ 1.   ,  0.   ,  0.   ,  0.   ],
       [ 0.   ,  1.   ,  0.   ,  0.   ],
       [ 0.   ,  0.   ,  1.   ,  0.   ],
       [ 0.   ,  0.   ,  0.462,  1.   ]])
```

In [145]:

```
#keep
U = M3.dot(U)
U
```

Out[145]:

```
array([[ -5.   ,  -8.   ,  -6.   ,  -2.   ],
       [  0.   ,  15.4  ,  -0.2  ,  -4.4  ],
       [  0.   ,   0.   ,  19.052,  -5.857],
       [  0.   ,   0.   ,   0.   ,  -2.421]])
```

Wrap-up

So we've built $M3P_3M_2P_2M_1P_1A = U$.

In [150]:

```
#keep
M3.dot(P3).dot(M2).dot(P2).dot(M1).dot(P1).dot(A)
```

Out[150]:

```
array([[ -5.   ,  -8.   ,  -6.   ,  -2.   ],
       [  0.   ,  15.4  ,  -0.2  ,  -4.4  ],
       [  0.   ,   0.   ,  19.052,  -5.857],
       [  0.   ,   0.   ,   0.   ,  -2.421]])
```

That left factor is anything but lower triangular:

In [151]:

```
#keep
M3.dot(P3).dot(M2).dot(P2).dot(M1).dot(P1)
```

Out[151]:

```
array([[ 0.   ,  0.   ,  0.   ,  1.   ],
       [ 0.   ,  0.   ,  1.   ,  0.2  ],
       [ 1.   ,  0.   , -0.26 , -0.052],
       [ 0.462,  1.   , -0.094, -0.219]])
```

Part II

Now try the reordering trick:

In [160]:

```
#keep
L3 = M3
L2 = P3.dot(M2).dot(la.inv(P3))
L1 = P3.dot(P2).dot(M1).dot(la.inv(P2)).dot(la.inv(P3))
```

In [155]:

```
L3.dot(L2).dot(L1).dot(P3).dot(P2).dot(P1)
```

Out[155]:

```
array([[ 0.   ,  0.   ,  0.   ,  1.   ],
       [ 0.   ,  0.026,  1.   ,  0.   ],
       [ 1.   , -0.266, -0.26 ,  0.   ],
       [ 0.462,  0.878, -0.094,  0.   ]])
```

We were promised that all of the `L`*n* are still lower-triangular:

In [168]:

```
#keep
print(L1)
print(L2)
print(L3)
```

```
[[ 1.   0.   0.   0. ]
 [ 0.2  1.   0.   0. ]
 [ 0.   0.   1.   0. ]
 [-0.2  0.   0.   1. ]]
[[ 1.     0.     0.     0.   ]
 [ 0.     1.     0.     0.   ]
 [ 0.    -0.26   1.     0.   ]
 [ 0.     0.026  0.     1.   ]]
[[ 1.     0.     0.     0.   ]
 [ 0.     1.     0.     0.   ]
 [ 0.     0.     1.     0.   ]
 [ 0.     0.     0.462  1.   ]]
```

So their product is, too:

In [172]:

```
#keep
Ltemp = L3.dot(L2).dot(L1)
Ltemp
```

Out[172]:

```
array([[ 1.   ,  0.   ,  0.   ,  0.   ],
       [ 0.2  ,  1.   ,  0.   ,  0.   ],
       [-0.052, -0.26 ,  1.   ,  0.   ],
       [-0.219, -0.094,  0.462,  1.   ]])
```

`P` is still a permutation matrix (but a more complicated one):

In [174]:

```
P = P3.dot(P2).dot(P1)
P
```

Out[174]:

```
array([[ 0.,  0.,  0.,  1.],
       [ 0.,  0.,  1.,  0.],
       [ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.]])
```

So to sum up, we've made:

In [175]:

```
Ltemp.dot(P).dot(A) - U
```

Out[175]:

```
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
```

Multiply from the left by $\texttt{Ltemp}^{-1}$, which is *also* lower triangular:

In [179]:

```
#keep
L = la.inv(Ltemp)
L
```

Out[179]:

```
array([[ 1.   ,  0.   ,  0.   ,  0.   ],
       [-0.2  ,  1.   ,  0.   ,  0.   ],
       [ 0.   ,  0.26 ,  1.   ,  0.   ],
       [ 0.2  , -0.026, -0.462,  1.   ]])
```

In [180]:

```
#keep
P.dot(A) - L.dot(U)
```

Out[180]:

```
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0., -0.,  0.,  0.]])
```