[S]=T. Sauer, Numerical Analysis, Second International Edition, Pearson, 2014

"Teorioppgaver"

1 Prøv å kjøre LU faktorisering manuelt på så mange små matriser som dere kan - f.eks. oppgavene 2.2.1–2.2.5.

Man kan sjekke svaret vha Numpy og LU-faktorisering kode fra wiki-siden. F.eks., oppgave 2.2.4 (b):

```
>>> import numpy as np
>>> from lu import LU
>>> A=np.array([[4,2,0],[4,4,2],[2,2,3]],dtype='f8')
>>> b=np.array([2,4,6],dtype='f8')
>>> LU(A)
>>> L=np.tril(A,-1)+np.eye(len(b))
>>> U=np.triu(A)
>>> y=np.linalg.solve(L,b)
>>> x=np.linalg.solve(U,y)
>>> print(L)
[[ 1.   0.   0. ]
 [ 1.   1.   0. ]
 [ 0.5  0.5  1. ]]
>>> print(U)
[[ 4.  2.  0.]
 [ 0.  2.  2.]
 [ 0.  0.  2.]]
>>> print(y)
[ 2.  2.  4.]
>>> print(x)
[ 1. -1.  2.]
```

Oppgave 2.2.4 (c):

```
>>> import numpy as np
>>> from lu import LU
>>> A=np.array([[1,-1,1,2],[0,2,1,0],[1,3,4,4],[0,2,1,-1]],dtype='f8')
>>> A1=np.copy(A)
```

```
>>> LU(A1)
>>> L=np.tril(A1,-1)+np.eye(4)
>>> U=np.triu(A1)
>>> print(L)
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 1.  2.  1.  0.]
 [ 0.  1.  0.  1.]]
>>> print(U)
[[ 1. -1.  1.  2.]
 [ 0.  2.  1.  0.]
 [ 0.  0.  1.  2.]
 [ 0.  0.  0. -1.]]
>>> print(np.linalg.norm(A-np.matmul(L,U)))
0.0
```

2 En annen type av oppgaver er å arbeide med vurdering av kompleksiteten i LU - f.eks. oppgavene 2.2.6–2.2.9.

3 Oppgave 2.3.1. Svaret kan sjekkes vha `numpy.linalg.norm(A,numpy.Inf)`

4 Oppgave 2.3.2. Svaret kan sjekkes vha `numpy.linalg.cond(A,numpy.Inf)`

5 Oppgave 2.3.5, 2.3.6

6 Prøv å kjøre LU faktorisering med pivotering manuelt på så mange små matriser som dere kan - f.eks. oppgavene 2.4.1–2.4.4.

Man kan sjekke svaret vha LU-faktorisering kode fra scipy. Husk at scipy beregner factoriseringen $A = \tilde{P}LU$ og ikke $PA = LU$; men $\tilde{P} = P^{-1} = P^{\mathrm{T}}$. F.eks., oppgave 2.4.2 (a):

```
>>> import numpy as np
>>> import scipy.linalg as la
>>> A=np.array([[1,1,0],[2,1,-1],[-1,1,-1]],dtype='f8')
>>> [P,L,U]=la.lu(A)
>>> print(P.T)
[[ 0.  1.  0.]
 [ 0.  0.  1.]
 [ 1.  0.  0.]]
>>> print(L)
[[ 1.          0.          0.        ]
 [-0.5         1.          0.        ]
 [ 0.5         0.33333333  1.        ]]
```

```
>>> print(U)
[[ 2.   1.  -1. ]
 [ 0.   1.5 -1.5]
 [ 0.   0.   1. ]]
>>> print(np.linalg.norm(np.matmul(P.T,A)-np.matmul(L,U),np.Inf))
0.0
```

7 Oppgave 2.4.9, 2.4.10

"Computeroppgaver"

8 Implementer en funksjon `solve` som tar en matris `A` og en vektor `b` som input og produserer en vektor `x` som løser systemet $Ax = b$ vha LU-faktorisering. Du kan bruke $A = LU$ fakrotisering kode `lu.py` fra wiki-siden; da trenger du bare å implementere back-substitusjoner. $Ly = b$ og $Ux = y$.

9 Implementer $PA = LU$ faktorisering i Python. Du kan begynne med $A = LU$ fakrotisering kode `lu.py` fra wiki-siden. Tips: man kan finne posisjonen av den største element som ligger under diagonalen i kolon j som

```
ihat = j+np.argmax(np.fabs(a[j:,j]))
```

To radene i matrisen kan bytes vha

```
a[[ihat,j]]=a[[j,ihat]]
```

Implementasjonen kan testes mot `scipy.linalg.lu`. Husk at scipy beregner factoriseringen $A = \tilde{P}LU$ og ikke $PA = LU$; men $\tilde{P} = P^{-1} = P^{\mathrm{T}}$.

10 Modificer oppgaven 8 slik at den bruker $PA = LU$ faktorisering istedenfor $A = LU$.

11 Oppgave 2.3.3, 2.3.4; brug funksjonen fra oppgaven 10 for å løse systemet.