Norges teknisk–naturvitenskapelige universitet
Institutt for Matematiske Fag

[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.

Solution, 2.2.2 (a)

$$A = \begin{pmatrix} 4 & 2 & 0 \\ 4 & 4 & 2 \\ 2 & 2 & 3 \end{pmatrix}$$

I will perform LU in place and denote the elements of L using green color, finalized elements of U using blue color.

First we go through column(1) and eliminate non-zero subdiagonal elements:

Subtract $1\times$ row(1) from row(2):

$$A = \begin{pmatrix} 4 & 2 & 0 \\ 1 & 2 & 2 \\ 2 & 2 & 3 \end{pmatrix}$$

Subtract $1/2\times$ row(1) from row(3):

$$A = \begin{pmatrix} 4 & 2 & 0 \\ 1 & 2 & 2 \\ 1/2 & 1 & 3 \end{pmatrix}$$

We know move on to column(2). Subtract $1/2\times$ row(2) from row(3) [only black elements need to be modified]

$$A = \begin{pmatrix} 4 & 2 & 0 \\ 1 & 2 & 2 \\ 1/2 & 1/2 & 2 \end{pmatrix}$$

or

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1/2 & 1/2 & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} 4 & 2 & 0 \\ 0 & 2 & 2 \\ 0 & 0 & 2 \end{pmatrix}.$$

We can check the factorization numerically:

```
>>> A = np.array([[4,2,0],[4,4,2],[2,2,3]],dtype='f8')
>>> L = np.array([[1,0,0],[1,1,0],[1/2.,1/2.,1]],dtype='f8')
>>> U = np.array([[4,2,0],[0,2,2],[0,0,2]],dtype='f8')
>>> np.matmul(L,U)-A
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  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.

Solution, 2.2.9: antakelsen her at å løse $k = 100$ systemer *etter å have beregnet LU-faktoriseringen* "koster" samme tid som å beregne LU faktorisering, og målet er å finne n.

Da kan vi skrive

$$\frac{2}{3}n^3 = 2kn^2,$$

(se seksjon 2.2.3 i boken) eller $n = 3k = 300$.

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)`

Solution, 2.3.2 (b)

$$A = \begin{pmatrix} 1 & 2.01 \\ 3 & 6 \end{pmatrix}$$

$$\|A\|_\infty = \max\{1 + 2.01, 3 + 6\} = 9.$$

$$A^{-1} = \frac{1}{\det(A)} \begin{pmatrix} 6 & -2.01 \\ -3 & 1 \end{pmatrix} = \frac{1}{1 \cdot 6 - (-3) \cdot (-2.01)} \begin{pmatrix} 6 & -2.01 \\ -3 & 1 \end{pmatrix} = -\frac{1}{0.03} \begin{pmatrix} 6 & -2.01 \\ -3 & 1 \end{pmatrix}$$
$$= \begin{pmatrix} -200 & 67 \\ 100 & -33\frac{1}{3} \end{pmatrix}$$

$$\|A^{-1}\|_\infty = \max\{200 + 67, 100 + 33\tfrac{1}{3}\} = 267.$$

Therefore $\kappa_\infty(A) = 9 \cdot 267 = 2403$. Note that because the rows of A are almost linearly dependent, even though A itself is small, the norm of A^{-1} is quite sizeable resulting in a relatively large condition number.

5 Oppgave 2.3.5, 2.3.6

Solution, 2.3.6 (a)

Backward error is:

$$\|Ax - b\|_\infty = \|[-10 + 2 \cdot 6 - 3, 2 \cdot (-10) + 4.01 \cdot 6 - 6.01]\|_\infty = \|[-1, -1.95]\|_\infty = 1.95$$

and therefore the relative backward error is

$$\frac{\|Ax-b\|_\infty}{\|b\|_\infty} = \frac{1.95}{6.01} \approx 0.32446.$$

In order to compute the forward error we need to know the solution to our system, which is $x_a = [1, 1]$. the relative forward error is:

$$\frac{\|x-x_a\|_\infty}{\|x_a\|_\infty} = \frac{\|[-11, 5]\|_\infty}{\|[1, 1]\|_\infty} = \frac{11}{1} = 11.$$

(e):

We compute the inverse of A first:

$$A^{-1} = \frac{1}{1 \cdot 4.01 - 2 \cdot 2}\begin{pmatrix} 4.01 & -2 \\ -2 & 1 \end{pmatrix} = \begin{pmatrix} 401 & -200 \\ -200 & 100 \end{pmatrix}$$

Therefore

$$\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty = 6.01 \cdot 601 = 3612.01.$$

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.

Solution, 2.4.2 (c)

$$A = \begin{pmatrix} 1 & 2 & -3 \\ 2 & 4 & 2 \\ -1 & 0 & 3 \end{pmatrix} \qquad P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

I will perform LU in place and denote the elements of L using green color, finalized elements of U using blue color.

First we go through column(1). The largest element is in row (2). We exchange rows 1 and 2 in A and P:

$$A = \begin{pmatrix} 2 & 4 & 2 \\ 1 & 2 & -3 \\ -1 & 0 & 3 \end{pmatrix} \qquad P = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Subtract $1/2\times$ row(1) from row(2):

$$A = \begin{pmatrix} 2 & 4 & 2 \\ 1/2 & 0 & -4 \\ -1 & 0 & 3 \end{pmatrix} \qquad P = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Subtract $-1/2\times$ row(1) from row(3):

$$A = \begin{pmatrix} 2 & 4 & 2 \\ 1/2 & 0 & -4 \\ -1/2 & 2 & 4 \end{pmatrix} \qquad P = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

We move on to column (2). Since $|2| > |0|$, we exchange rows (2) and (3):

$$A = \begin{pmatrix} 2 & 4 & 4 \\ -1/2 & 2 & 4 \\ 1/2 & 0 & -4 \end{pmatrix} \qquad P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Because element in position (3,2) is 0, there is nothing left to eliminate, and the final answer is:

$$A = \begin{pmatrix} 2 & 4 & 4 \\ -1/2 & 2 & 4 \\ 1/2 & 0 & -4 \end{pmatrix} \qquad P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

We verify our computations:

```
>>> import numpy as np
>>> import scipy.linalg as la
>>> A=np.array([[1,2,-3],[2,4,2],[-1,0,3]],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.   1. ]]
>>> print(U)
[[ 2.  4.  2.]
 [ 0.  2.  4.]
 [ 0.  0. -4.]]
```

7 Oppgave 2.4.9, 2.4.10

Solution, 2.4.9

(a):

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 2 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 2 \\ -1 & -1 & 1 & 2 \\ -1 & -1 & -1 & 1 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 2 \\ -1 & -1 & 1 & 2 \\ -1 & -1 & -1 & 2 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 2 \\ -1 & -1 & 1 & 4 \\ -1 & -1 & -1 & 2 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 2 \\ -1 & -1 & 1 & 4 \\ -1 & -1 & -1 & 4 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 2 \\ -1 & -1 & 1 & 4 \\ -1 & -1 & -1 & 8 \end{pmatrix}$$

No row exchanges are necessary, so $P = I$, the identity matrix.

(b): We can see that all subdiagonal elements of L are -1, all diagonal elements of U are 1 (except the last one), and elements of the last column are $U_{in} = 2^{i-1}$, $i = 1, \ldots, n$.

Solution, 2.4.10

(a): Since P is just a permultation matrix, all elements of PA satisfy the inequality $|(PA)_{ij}| \leq 1$, $i, j = 1, \ldots, n$.

Because of partial pivoting, each element of L satisfies $|L_{i,j}| \leq 1$, see a comment on p. 95 in the book. (This is clear from the algorithm, as we select the largest entry in the column (absolute value) to divide the rest of the column with.)

We now look more carefully at LU-algorithm. First we possibly exchange the rows. Then, when we traverse column j, $j = 1, \ldots, n-1$, each element A_{ik}, $i > j$, $k \geq j$ is updated once as $A_{ik} := A_{ik} - L_{ij}A_{jk}$. Note that after each update we have $|A_{ik}| \leq |A_{ik}| + |L_{ij}||A_{jk}|$.

Thus after eliminating the subdiagonal elements in column 1 the largest element is at most $|A_{ik}| \leq 1 + 1 \cdot 1 = 2$. After eliminating the subdiagonal elements in column 2 it is $|A_{ik}| \leq 2 + 1 \cdot 2 = 4$ etc. Finally, after eliminating the subdiagonal element in column $n-1$ the largest element is at most 2^{n-1}.

(b): The question is formulated very vaguely. One can for example scale the matrix A by dividing it with its largest element, say $A_{\hat{i}\hat{j}}$. Then the elements of the new matrix $A_{\hat{i}\hat{j}}^{-1}A$ will be bounded by 1 in absolute magnitude. We then get

$$PA_{\hat{i}\hat{j}}^{-1}A = LU$$

with $|U_{ij}| \leq 2^{n-1}$. We can rescale the equations back to obtain the LU factorization of the original matrix:

$$PA = L(A_{\hat{i}\hat{j}}U).$$

Thus the elements in the upper triangular matrix will be bounded by $A_{\hat{i}\hat{j}}2^{n-1}$. (Note that we must include the factor $A_{\hat{i}\hat{j}}$ into U and not L, since L must have 1 on the main diagonal by convention.)

"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.