Norges teknisk–naturvitenskapelige universitet
Institutt for Matematiske Fag

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

"Teorioppgaver"

Cholesky faktorisering

$\boxed{1}$ Prøv å kjøre Cholesky faktorisering manuelt på så mange små matriser som dere kan; f.eks oppgavene 2.6.3-8.

Solution, 2.6.8 (b)

$$A = \begin{pmatrix} 4 & -2 & 0 \\ -2 & 2 & -1 \\ 0 & -1 & 5 \end{pmatrix}$$

Vi begynner med å finne en Cholesky faktorisering av A. Man kan følge algoritmen på side 121. $R_{11} = (A_{11})^{1/2} = (4)^{1/2} = 2$; $R_{1,2:3} = u^{\mathrm{T}} = 1/R_{1,1}(A_{1,2:3}) = 1/2[-2, 0] = [-1, 0]$.

$$A_{2:3,2:3} := A_{2:3,2:3} - uu^{\mathrm{T}} = \begin{pmatrix} 2 & -1 \\ -1 & 5 \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ -1 & 5 \end{pmatrix}$$

$R_{2,2} = (A_{2,2})^{1/2} = (1)^{1/2} = 1$; $R_{2,3} = u^{\mathrm{T}} = 1/R_{2,2}(A_{2,3}) = -1$; $A_{3,3} = A_{3,3} - uu^{\mathrm{T}} = 5 - 1 = 4$.

Finally $R_{3,3} = (A_{3,3})^{1/2} = 2$.

Thus:

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

One can check the LU factorization using matrix multiuplication:

```
import numpy as np
R = np.array([[2,-1,0],[0,1,-1],[0,0,2]])
print(R.T.dot(R))
[[ 4 -2  0]
 [-2  2 -1]
 [ 0 -1  5]]
```

We can now solve the system $Ax = b$, where $b^{\mathrm{T}} = [0, 3, -7]$.

First, we solve $R^{\mathrm{T}}y = b$: $2y_1 = 0 \implies y_1 = 0$

$-1y_1 + 1y_2 = 3 \implies y_2 = 3$.

$-1y_2 + 2y_3 = -7 \implies 2y_3 = -4 \implies y_3 = -2$.

We now solve the system $Rx = y$.

$2x_3 = -2 \implies x_3 = -1$.

$1x_2 - 1x_3 = 3 \implies x_2 = 2$.

$2x_1 - 1x_2 = 0 \implies x_1 = 1$.

Again, the answer can be checked by sunstituting it into the equation $Ax = b$.

```
import numpy as np
A = np.array([[4,-2,0],[-2,2,-1],[0,-1,5]])
x = np.array([1,2,-1])
print(A.dot(x))
[ 0  3 -7]
```

Sparse matriser

$\boxed{2}$ Consider a tri-diagonal system $Ax = b$, where the matrix A is defined by $A_{i,i} = \alpha_i$, $i = 1, \dots, n$; $A_{i,i+1} = \gamma$, $i = 1, \dots, n-1$; $A_{i,i-1} = \beta_i$, $i = 2, \dots, n$. Rest of the elements are zeros:

$$A = \begin{pmatrix} \alpha_1 & \gamma_1 & 0 & 0 & \dots & 0 \\ \beta_2 & \alpha_2 & \gamma_2 & 0 & \dots & 0 \\ 0 & \beta_3 & \alpha_3 & \gamma_3 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \beta_{n-1} & \alpha_{n-1} & \gamma_{n-1} \\ 0 & \dots & 0 & 0 & \beta_n & \alpha_n \end{pmatrix}$$

Let $LU = A$ be the LU-factorization of A (without pivoting; we assume that such a factorization exists). Describe explicitly the spartity structure (that is, the position of non-zero elements) in matrices L and U. Describe the algorithm for computing the LU-factorization of such a matrix. Further describe an algorithm for solving a linear system $Ax = b$ for the tri-diagonal matrix based on the previously computed LU factorization.

Solution

In LU-factorization algorithm we try to eliminate non-zero elements from under the diagonal by performing row operations. In our case, there is only one non-zero subdiagonal, thus only one element from under the diagonal needs to be eliminated.

For example, to eliminate β_2 we subtract $\beta_2/\alpha_1 \times$ row(1) from row(2), which does not create any new non-zero elements.

Similarly, to eliminate β_3 we subtract $\beta_3/\alpha_2 \times$ row(2) from row(3), which will not create any new non-zeros as β_2 has been eliminated in the previous operation.

Thus the matrix L will contain two diagonals: 1 on the main diagonal (they do not need to be stored) and, say, δ_i, $i = 2, \ldots, n$ on the subdiagonal.

The whole algorithm can be described as follows:

```
for i=2,...,n
  delta[i] = beta[i]/alpha[i-1]
  beta[i] = 0
  alpha[i] = alpha[i] - delta[i]*gamma[i]
end for
```

Note that in principle, to save space, the array `beta` can be used to store the sub-diagonal of L instead of a new array `delta`. Further note that the complexity of LU factorization in this case is $O(n)$ and not $O(n^3)$, which is a huge computational saving.

Having computed the LU factorization, we can solve the linear system using two for-loops ($Ly = b$, $Ux = y$):

```
y[1] = b[1]
for i=2,...,n
  y[i] = b[i] - delta[i]*y[i-1]
end for
x[n] = y[n]/alpha[n]
for i=n-1,...,1
  x[i] = (y[i]-gamma[i]*y[i+1])/alpha[i]
end for
```

Note that the complexity of forward-backward substitutions in this case is $O(n)$ and not $O(n^2)$.

Iterative metoder

$\boxed{3}$ Oppgave 2.5.1

Solution, 2.5.1 (b)

Jacobi:

$$
\begin{aligned}
x^0 &= [0,0,0]^{\mathrm{T}},\\
A &= \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}\\
D &= \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}\\
b &= [0,2,0]^{\mathrm{T}},\\
x^1 &= x^0 + D^{-1}[b - Ax^0] = b/2 = [0,1,0]\\
x^2 &= x^1 + D^{-1}[b - Ax^1] = [0,1,0] + [[0,2,0]^{\mathrm{T}} - [-1,2,-1]^{\mathrm{T}}]/2 = [0,1,0]^{\mathrm{T}} + [1,0,1]^{\mathrm{T}}/2\\
&= [1/2, 1, 1/2]^{\mathrm{T}}.
\end{aligned}
$$

Gauss-Seidel:

$$
\begin{aligned}
x_1^1 &= (b_1 - A_{1,2:3}x_{2:3}^0)/D_{1,1} = 0,\\
x_2^1 &= (b_2 - A_{2,1}x_1^1 - A_{2,3}x_3^0)/D_{2,2} = (2 + 1\cdot 0 + 1\cdot 0)/2 = 1,\\
x_3^1 &= (b_3 - A_{3,1:2}x_{1:2}^1)/D_{3,3} = (0 - 0\cdot 0 + 1\cdot 1)/2 = 1/2\\
x_1^2 &= (b_1 - A_{1,2:3}x_{2:3}^1)/D_{1,1} = (0 + 1\cdot 1 - 0\cdot 0.5)/2 = 1/2,\\
x_2^2 &= (b_2 - A_{2,1}x_1^2 - A_{2,3}x_3^1)/D_{2,2} = (2 + 1\cdot 0.5 + 1\cdot 0.5)/2 = 3/2,\\
x_3^1 &= (b_3 - A_{3,1:2}x_{1:2}^2)/D_{3,3} = (0 - 0\cdot 0.5 + 1\cdot 1.5)/2 = 3/4
\end{aligned}
$$

$\boxed{4}$ Oppgave 2.5.2

Solution, 2.5.2 (b)

The system

$$
\begin{pmatrix} 1 & -8 & -2 \\ 1 & 1 & 5 \\ 3 & -1 & 1 \end{pmatrix}
\begin{pmatrix} u \\ v \\ w \end{pmatrix} =
\begin{pmatrix} 1 \\ 4 \\ -2 \end{pmatrix}
$$

can be rearranged to

$$
\begin{pmatrix} 3 & -1 & 1 \\ 1 & -8 & -2 \\ 1 & 1 & 5 \end{pmatrix}
\begin{pmatrix} u \\ v \\ w \end{pmatrix} =
\begin{pmatrix} -2 \\ 1 \\ 4 \end{pmatrix}
$$

which is diagonally dominant; thus both Jacobi and Gauss–Seidel methods are guaranteed to converge. After that one proceeds as in the previous exercise.

Systemer av ikke-lineære likninger

$\boxed{5}$ Oppgave 2.7.1

Solution, 2.7.1 (d)

$F(u,v,w) = [u^2 + v - w^2, \sin(uvw), uvw^4]$.

The Jacobian matrix is:

$$DF = \begin{pmatrix} 2u & v & -2w \\ vw\cos(uvw) & uw\cos(uvw) & uv\cos(uvw) \\ vw^4 & uw^4 & 4uvw^3 \end{pmatrix}$$

6 Oppgave 2.7.4

Solution, 2.7.4 (b)

The Jacobian matrix is

$$DF = \begin{pmatrix} 2u & 8v \\ 8u & 2v \end{pmatrix}$$

We now take two steps of the Newton's system, starting from $x^0 = (1,1)$:

$$x^1 = x^0 - [DF(1,1)]^{-1}F(1,1) = \begin{pmatrix} 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 2 & 8 \\ 8 & 2 \end{pmatrix}^{-1} \begin{pmatrix} 1+4-4 \\ 4+1-4 \end{pmatrix} = \begin{pmatrix} 0.9 \\ 0.9 \end{pmatrix}$$

$$x^2 = x^2 - [DF(0.9,0.9)]^{-1}F(0.9,0.9) = \begin{pmatrix} 0.9 \\ 0.9 \end{pmatrix} - \begin{pmatrix} 1.8 & 7.2 \\ 7.2 & 1.8 \end{pmatrix}^{-1} \begin{pmatrix} 4.05-4 \\ 4.05-4 \end{pmatrix} \approx \begin{pmatrix} 0.8944 \\ 0.8944 \end{pmatrix}$$

Note that the exact solution is given by $u = v = \pm\sqrt{4/5} \approx \pm 0.89442719099991586$

"Computeroppgaver"

Cholesky faktorisering

—

Sparse matriser

7 Implement a function for solving a tri-diagonal system of equations using LU-factorization in Python (see exercise 2). It should take three numpy-arrays α, β, γ, and b as inputs and produce the solution x as output.

Test your algorithm on some randomly generated tri-diagonal matrices (e.g., generate random arrays β and γ, and then generate a random α such that the resulting matrix is strictly diagonally dominant, hence also non-singular).

Compare the results produced by your algorithm with those produced by the sparse linear solver `scipy.sparse.linalg.spsolve`. (In order to do this you need to create a sparse tri-diagonal matrix. The easiest way to do this is to create it as `scipy.sparse.dia_matrix` and then convert it to `scipy.sparse.csr_matrix` format).

Solution

Se `oppgave7.py`.

Iterative metoder

8 Oppgave 2.5.1-2.5.3

Solution

Se `oppgave_2_5_1.py`, `oppgave_2_5_2.py` og `oppgave_2_5_3.py`.

Systemer av ikke-lineære likninger

9 Oppgave 2.7.5

Solution

Se `oppgave_2_7_5.py`