

TFY4235/FYS8904
Solution problemset 3 Spring 2015



Solutions to the problems are provided in the *Code* subdirectory.

Problem 1.

The problem here is to translate the physical problem — finding the potentials in a chain of 1000 random resistors in series — into a matrix problem on the form $Ax = b$. What is A and b ? v is evidently a vector that contains all potentials. We enumerate the nodes of the chain from 0 to 1000. The resistors are numbered 0 to 999. Node 0 has a potential of $v_0 = 0$ and node 1000 has a potential of $v_{1000} = 1$. The other potentials, v_i , where $i = 1, 999$, are unknown. The Kirchhoff equations give for each of the nodes $i = 1, 999$

$$-c_{i-1}v_{i-1} + (c_{i-1} + c_i)v_i - c_iv_{i+1} = 0. \quad (1)$$

Please note in this expression that we are using *conductance* c rather than *resistance* r as was done in the lectures. The relation between the two quantities is $c = 1/r$.¹ The equations for $i = 1$ og $i = 999$ may be rewritten

$$(c_0 + c_1)v_1 - c_1v_2 = c_0v_0 \quad (2)$$

and

$$-c_{998}v_{998} + (c_{998} + c_{999})v_{999} = c_{999}v_{1000}, \quad (3)$$

where $v_0 = 0$ og $v_{1000} = 1$. From Eqs. (1), (2) and (3) we see that the matrix A is

$$A = \begin{pmatrix} (c_0 + c_1) & -c_1 & 0 & 0 & \dots & 0 & 0 \\ -c_1 & (c_1 + c_2) & -c_2 & 0 & \dots & 0 & 0 \\ 0 & -c_2 & (c_2 + c_3) & -c_3 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & (c_{997} + c_{998}) & -c_{998} \\ 0 & 0 & 0 & 0 & \dots & -c_{998} & (c_{998} + c_{999}) \end{pmatrix} \quad (4)$$

and the vector b is

$$b = \begin{pmatrix} c_0v_0 \\ 0 \\ \vdots \\ 0 \\ c_{999}v_{1000} \end{pmatrix}. \quad (6)$$

¹Why do I do it differently here from the lectures? This is a nice illustration in my mind of the fact that formulating a problem is not a unique process. Sometimes, a reformulation may simplify the problem.

The vector v is

$$v = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{998} \\ v_{999} \end{pmatrix}. \quad (7)$$

The Jacobi iteration is

$$Dv^{n+1} = b - (L + U)v^n \quad (8)$$

where D is the diagonal of A . If we substitute for A and b , we get the matrix equation

$$\begin{pmatrix} (c_0 + c_1)v_1^{n+1} \\ (c_1 + c_2)v_2^{n+1} \\ \vdots \\ (c_{998} + c_{999})v_{999}^{n+1} \end{pmatrix} = \begin{pmatrix} c_0v_0^n + c_1v_2^n \\ c_1v_1^n + c_2v_3^n \\ \vdots \\ c_{998}v_{998}^n + c_{999}v_{1000}^n \end{pmatrix}. \quad (9)$$

Here is a Fortran program that sets up the resistor chain and then finds the potentials by using Eq. (9):

Listing 1: jacobi.f

```

1  program jacobi
2  c Jacobi relaxasjon
3      dimension volt(0:1000),vnew(999),cond(0:999)
4      dimension resi(0:999)
5  c Tilfelding tall initialisering
6      rinv=0.5/2147483647.
7      rold=0.
8      ibm=1927
9      do i=1,1000
10     ibm=ibm*16807
11     enddo
12 c Genererer de 1000 konduktansene
13     do i=0,999
14         ibm=ibm*16807
15         cond(i)=ibm*rinv+0.5
16     enddo
17     do i=1,999
18         resi(i)=1./(cond(i-1)+cond(i))
19     enddo
20 c Presisjonskriterium
21     pres=2.e-4
22 c Initialiserer spenningene
23     do i=0,1000
24         volt(i)=float(i)/1000
25     enddo
26 c Iterasjonen
27     do ite=1,1000000
28         do i=1,999
29             vnew(i)=(cond(i-1)*volt(i-1)+cond(i)*volt(i+1))*resi(i)

```

```

30     enddo
31     eps=0.
32     do i=1,999
33         volt(i)=vnew(i)
34     enddo
35     do i=1,999
36         eps=eps+(cond(i-1)*volt(i-1)+cond(i)*volt(i+1)
37         & -(cond(i-1)+cond(i))*volt(i))**2
38     enddo
39     eps=sqrt(eps)
40     write(*,*) ite,eps
41     if(eps.lt.pres) goto 200
42     enddo
43 c Ferdig
44 200 continue
45 end

```

The Gauss-Seidel iteration,

$$(L + D)v^{n+1} = b - Uv^n, \quad (10)$$

is very easy to implement. Rewrite (10) to

$$Dv^{n+1} = b - Uv^n - Lv^{n+1} \quad (11)$$

and compare with the Jacobi relaxation (8). Since we read the indices from $i = 1$ to $i = 999$, we implement the Gauss-Seidel iteration in the previous program by simply removing the work vector `vnew(i)`. Here is the Gauss-Seidel program:

Listing 2: gauss.f

```

1     program gauss
2 c Gauss-Seidel relaksasjon
3     dimension volt(0:1000),cond(0:999)
4     dimension resi(0:999)
5 c Tilfelding tall initialisering
6     rinv=0.5/2147483647.
7     rold=0.
8     ibm=1927
9     do i=1,1000
10        ibm=ibm*16807
11    enddo
12 c Genererer de 1000 konduktansene
13    do i=0,999
14        ibm=ibm*16807
15        cond(i)=ibm*rinv+0.5
16    enddo
17    do i=1,999
18        resi(i)=1./(cond(i-1)+cond(i))
19    enddo
20 c Presisjonskriterium
21    pres=2.e-4
22 c Initialiserer spenningene

```

```

23     do i=0,1000
24     volt(i)=float(i)/1000
25     enddo
26 c Iterasjonen
27     do ite=1,1000000
28     do i=1,999
29     volt(i)=(cond(i-1)*volt(i-1)+cond(i)*volt(i+1))*resi(i)
30     enddo
31     eps=0.
32     do i=1,999
33     eps=eps+(cond(i-1)*volt(i-1)+cond(i)*volt(i+1)
34     & -(cond(i-1)+cond(i))*volt(i))**2
35     enddo
36     eps=sqrt(eps)
37     write(*,*) ite,eps
38     if(eps.lt.pres) goto 200
39     enddo
40 c Ferdig
41 200 continue
42 end

```

Now follows a direct implementation of the SOR algorithm,

$$(D + L)v^{n+1} = \omega v_{GS}^{n+1} + (1 - \omega)v^n \quad (12)$$

where v_{GS}^{n+1} is the result of the Gauss-Seidel iteration (10). The actual Fortran program looks like this:

Listing 3: sor.f

```

1     program sor
2 c sor relaksasjon
3     dimension volt(0:1000),vnew(999),cond(0:999)
4     dimension resi(0:999)
5 c Leser inn parameteren omega
6     write(*,1)
7 1     format(1x,' Omega = ',%)
8     read(*,*) om
9 c Tilfelding tall initialisering
10    rinv=0.5/2147483647.
11    rold=0.
12    ibm=1927
13    do i=1,1000
14    ibm=ibm*16807
15    enddo
16 c Genererer de 1000 konduktansene
17    do i=0,999
18    ibm=ibm*16807
19    cond(i)=ibm*rinv+0.5
20    enddo
21    do i=1,999
22    resi(i)=1./(cond(i-1)+cond(i))

```

```

23     enddo
24 c Presisjonskriterium
25     pres=2.e-4
26 c Initialiserer spenningene
27     do i=0,1000
28         volt(i)=float(i)/1000
29     enddo
30 c Iterasjonen
31     do ite=1,1000000
32         do i=1,999
33             vnew(i)=volt(i)
34         enddo
35         do i=1,999
36             volt(i)=(cond(i-1)*volt(i-1)+cond(i)*volt(i+1))*resi(i)
37         enddo
38         do i=1,999
39             volt(i)=om*volt(i)+(1.-om)*vnew(i)
40         enddo
41         eps=0.
42         do i=1,999
43             eps=eps+(cond(i-1)*volt(i-1)+cond(i)*volt(i+1)
44             & -(cond(i-1)+cond(i))*volt(i))**2
45         enddo
46         eps=sqrt(eps)
47         write(*,*) ite,eps
48         if(eps.lt.pres) goto 200
49     enddo
50 c Ferdig
51 200 continue
52 end

```

With a stop criterion set to 2×10^{-4} , the Jacobi program used 115973 iterations, the Gauss-Seidel program used 112 iterations. the SOR program used 75 iterations when $\omega = 2.5$. The Jacobi method performs a lot worse than theory would suggest.

The number of iterations goes *down* when increasing the number of resistors. One would naively have expected the opposite. The reason is that the current flowing through the chain is inversely proportional to the number of resistors. The stopping criterion is proportional to the current flowing. Hence, we need to adjust it down when adjusting the number of resistors up.