TFY4235/FYS8904

Solution problemset 4 Spring 2015

NTNU

Institutt for fysikk

Problem 1.

A conjugate gradient program is included in listing 1 and 2. It uses 199 iterations to reach the solution with a presicion of 2×10^{-4}. Rather than using the conjugate gradient version taught in class and listed in *Numerical Recipes*. Rather, I have used a version that is described in Batrouni and Hansen, Journal of Statistical Physics, **52**, 747 (1988) — see equations (32)–(38) there. This version of the conjugate gradient algorithm is the most efficient for this class of problem.

Listing 1: cg.f

```
      program cg
c konjugert gradient
      dimension volt(0:1000),cond(0:999)
      dimension p(0:1000),r(999),ap(999)
c Tilfelding tall initialisering
      rinv=0.5/2147483647.
      ibm=1927
      do i=1,1000
      ibm=ibm*16807
      enddo
c Genererer de 1000 konduktansene
      do i=0,999
      ibm=ibm*16807
      cond(i)=ibm*rinv+0.5
      enddo
c Presisjonskriterium
      pres=2.e-4
c Initialiserer spenningene
      do i=0,1000
      volt(i)=float(i)/1000
      enddo
c Initialiserer hjelpevektorer
      do i=1,999
      p(i)=-cond(i-1)*(volt(i-1)-volt(i))-cond(i)*(volt(i+1)-volt(i))
      r(i)=p(i)
      enddo
      p(   0)=0.
      p(1000)=0.
c Iterasjonen
      do ite=1,2*1000
      rps=0.
      do i=1,999
      rps=rps+r(i)*r(i)
      enddo
      if(sqrt(rps).le.pres) goto 200
```

```
      do i=1,999
      ap(i)=cond(i-1)*(p(i-1)-p(i))+cond(i)*(p(i+1)-p(i))
      enddo
      am=0.
      do i=1,999
      am=am+p(i)*ap(i)
      enddo
      am=rps/am
      do i=1,999
      volt(i)=volt(i)+am*p(i)
      enddo
      do i=1,999
      r(i)=r(i)-am*ap(i)
      enddo
      rpn=0.
      do i=1,999
      rpn=rpn+r(i)*r(i)
      enddo
      bm=rpn/rps
      do i=1,999
      p(i)=r(i)+bm*p(i)
      enddo
      enddo
c Ferdig
200   continue
      write(*,*) ite
      end
```

Listing 2: main.c

```
#include<stdio.h>
#include<stdlib.h>

int IBM = 1927;  //SEED
const int N = 1024;
double rinv = 0.5/((long int)1 << 31);
double pres = 2.0e-6;
double rps, rpn, am, bm;

double * cond, * resi, * volt, * vnew, * p, * r, * ap, *gs, *sor, *cg;

double randIBM(){          //returns random numbrs on the unit
   interval
    IBM*=16807;
    return IBM*rinv + 0.5;
}

int main(){

    /* INITIALIZATION */

    int i;
  int n = 0;
```

```
23
24      cond = (double*)malloc(sizeof(double)*N);
25      volt = (double*)malloc(sizeof(double)*(N+1));
26      p = (double*)malloc(sizeof(double)*(N+1));
27      r = (double*)malloc(sizeof(double)*(N-1));
28      ap = (double*)malloc(sizeof(double)*(N-1));
29
30      // Burn-in
31    for(i=0;i<1000;i++) //throwaway random numbers
32          randIBM();
33
34      // Randomizing conductances.
35      for (i = 0; i < N; i++)
36          cond[i] = randIBM();
37
38      // Initialize voltages to a linear profile.
39      for(i=0;i<N+1;i++)
40          volt[i] = (double)i/N;
41
42    // Initialization of work vectors.
43      rps = 0;
44      for (i=1;i<N;i++){
45          p[i] = -cond[i-1]*(volt[i-1] - volt[i]) - cond[i]*(volt[i+1]-
                volt[i]);
46          r[i-1] = p[i];
47          rps += r[i-1]*r[i-1];
48      }
49      p[0] = 0;
50      p[N] = 0;
51
52      while( rps > pres*pres){
53          am = 0;
54          rpn = 0;
55
56          for (i=1;i<N;i++){
57              ap[i-1] = cond[i-1]*(p[i-1] - p[i]) + cond[i]*(p[i+1] - p[
                    i]);
58              am += p[i]*ap[i-1];
59          }
60
61          am = rps/am;
62
63          for (i=1;i<N;i++){
64              volt[i] += am*p[i];
65          }
66
67          for (i=1;i<N;i++){
68              r[i-1] -= am*ap[i-1];
69          }
70
71          for (i=0;i<N-1;i++){
72              rpn += r[i]*r[i];
73          }
```

```
74
75          bm = rpn/rps;
76          for (i=1;i<N;i++){
77              p[i] = r[i-1] + bm*p[i];
78          }
79
80      rps = rpn;
81          n++;
82      }
83
84      printf("Conjugate gradient: %d iterations\n",n);
85      printf("v[%d]: conj=%.9f\n",N/2,volt[N/2]);
86
87      /* De-allocate. */
88      free(cond);
89      free(volt);
90      free(p);
91      free(r);
92      free(ap);
93
94      return 0;
95  }
```

It is worth noticing that in the conjugate gradient algorithm the vector b is only used during initialization. See how it is done in the program. (It is not obvious, so look very carefully!)

You should note that the conjugate gradient algorithm has the peculiarity that even if there are bugs in the program, it may still find the solution. However, the convergence will be terrible. Hence, if it seemingly works but it is slow, there are bugs in the program.