TFY4235/FYS8904

Solution problemset 1 Spring 2015

NTNU

Institutt for fysikk

Problem 1.
Two solutions to the problem are provided in the *code* subdirectory. One in Fortran (listing 1) and one in C (listing 2).

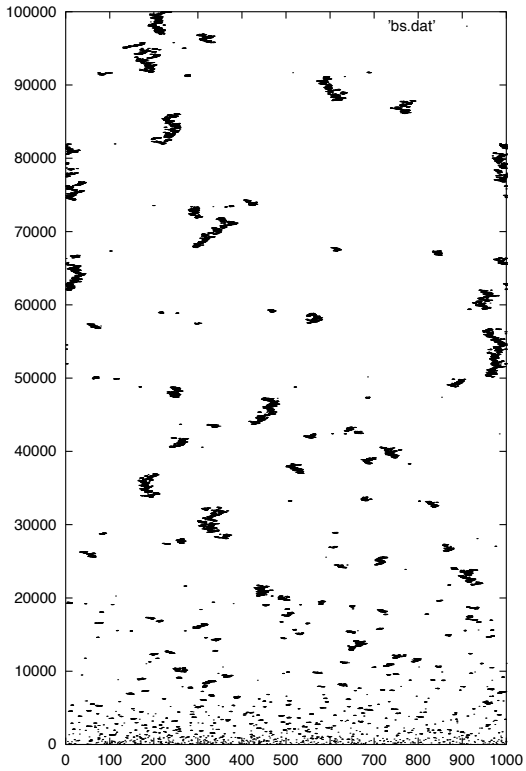

Figure 1: Results from the Bak-Sneppen model.

There are two aspects of this program that is not straight forward: 1) The implementation of the random number generator. This is a so-called portable generator that goes under the name "16807" and which works by overflow. We will describe this in detail later on in the course. The second aspect is the implementation of the periodic boundary conditions. The way I have implemented them is the most efficient: There are no logical tests to perform.
In the plot in figure 1 I have reproduced Fig. 3a in the Bak-Sneppen paper. It shows clearly how the activity in the model is "clumped" together. This clumping corresponds to the avalanches of change (think about this!). Here the x-axis shows position and the y-axis shows time.

Listing 1: Implementation of the Bak-Sneppen model in Fortran

```
      program bs
c
c Program implementing Bak-Sneppen model
c
      parameter(l=1000,mtime=100000,idum=4711)
c
      dimension fitnes(l),ir(l),il(l)
c
c Heating up random number generator
c
      ibm=idum
      rinv=0.5/(2.**31-1.)
      do i=1,1000
      ibm=ibm*16807
      enddo
c
c Implementing periodic boundary conditions
c
      do i=1,l
      ir(i)=i+1
      il(i)=i-1
      enddo
      ir(l)=1
      il(1)=l
c
c Initializing fitness vector
c
      do i=1,l
      ibm=ibm*16807
      fitnes(i)=(ibm*rinv+0.5)
      enddo
c
c Loop over time
c
      do itime=1,mtime
c
c finding smallest fitness factor
c
      is=0
      fs=2.0
      do i=1,l
      if(fitnes(i).le.fs) then
      is=i
      fs=fitnes(i)
      endif
      enddo
c
c Updating element with smallest fitness and neighbors
c
      ibm=ibm*16807
      fitnes(   is )=(ibm*rinv+0.5)
```

```
52        ibm=ibm*16807
53        fitnes(ir(is))=(ibm*rinv+0.5)
54        ibm=ibm*16807
55        fitnes(il(is))=(ibm*rinv+0.5)
56  c
57  c Writing position of least fit element
58  c
59        write(*,*) is,itime
60  c
61        enddo
62  c
63        end
```

Listing 2: Implementation of the Bak-Sneppen model in C

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<sys/time.h>
4  #include<math.h>
5
6  // RNG Variables and constants
7  const double RINV = 0.5/(1 << 31);      // 0.5/(2^31)
8  int IBM;
9
10 // Variables and constants
11 int N;
12 int dtmax;
13
14 int mini;
15 double minr;
16 double t_tmp;
17 double time_init, time_sim, time_out;
18
19 // Arrays
20 double * r;     // Array for storing fitness values
21 int * mins;     // Array for storing which species was least fit
22
23
24 double walltime ( void ) {  // Returns current time, with microsecond
       precision
25     static struct timeval t;
26     gettimeofday ( &t, NULL );
27     return ( t.tv_sec + 1e-6 * t.tv_usec );
28 }
29
30 int randIBM() {      // Random integer between -2^31 and 2^31 - 1
31     IBM*=16807;     // google: LCG - Linear Congruential Generator
32     return IBM;
33 }
34
35 double randIBMf() {  // Random number between 0 and 1, [0,1)
36     return randIBM()*RINV + 0.5;
37 }
```

```
38
39 int main (int argc, char ** argv) {
40     int i,j;
41
42     // INITIALIZATIONS
43   t_tmp = walltime();
44     for (i = 0; i < argc; i++) {
45         printf("%s ",argv[i]);
46     }
47     printf("\n");
48
49     if (argc == 1) {
50         printf("No arguments from command line. Using default values\n
               ");
51         N = 64;
52         dtmax = 10000;
53     } else if (argc == 3) {
54         N = atoi(argv[1]);       // First argument, number of species
55         dtmax = atoi(argv[2]);  // Second argument, number of
               iterations
56     } else {
57         printf("Invalid number of arguments from command line,
               aborting.\n");
58         exit(-1);
59     }
60     printf(" N = %d\n",N);
61     printf(" dtmax = %d\n\n",dtmax);
62
63     FILE * f1 = fopen("out.d","w");
64
65     // Initializing RNG
66     IBM = 2*time(0) + 1;
67     for (i = 0; i < 1000; i++)
68         randIBM();
69
70     r = (double*)malloc(sizeof(double)*N);
71     mins = (int*)malloc(sizeof(int)*dtmax);
72
73     // Initialize fitness array
74     for(i = 0; i < N; i++){
75         r[i] = randIBMf();
76     }
77     time_init = walltime() - t_tmp;
78
79     // MAIN LOOPS
80     t_tmp = walltime();
81     for(j = 0;j < dtmax; j++){
82         minr = 1;
83         for(i=0;i<N;i++){
84             if (r[i] < minr){
85                 minr = r[i];
86                 mini = i;
87             }
```

```
88          }
89
90          mins[j] = mini;
91          int right = mini + 1;
92          int left = mini - 1;
93
94          if(mini == N-1)
95              right = 0;
96          if(mini == 0)
97              left = N-1;
98
99          r[mini] = randIBMf();
100         r[left] = randIBMf();
101         r[right] = randIBMf();
102     }
103     time_sim = walltime() - t_tmp;
104
105     // PRINT TO FILE
106     t_tmp = walltime();
107     for (i = 0; i < dtmax; i++){
108         fprintf(f1,"%d    %d\n",mins[i], i);
109     }
110     time_out = walltime() - t_tmp;
111
112     // TIMINGS
113     printf("time init: %f\n",time_init);
114     printf("time sim: %f\n",time_sim);
115     printf("time out: %f\n",time_out);
116
117     // FREE MEMORY
118     free(r);
119     free(mins);
120     fclose(f1);
121     return 0;
122 }
```