Source code for project 2
FYS4150

Fredrik E Pettersen

November 12, 2012

.cpp - file

```
/*
 * File:   ising.cpp
 * Author: fredrik
 *
 * Created on 2. november 2012, 10:30

 * Simulate the development of a system of magnetic dipoles by the Ising model.
 */

#include "ising.h"

#define PI 3.14159

int main(int argc, char** argv) {

    int n = atoi(argv[1]);
    int N = 10000000;
    //double g_sigma = 0;
    //int num_cores = 0;
    //double start = clock();

    ofstream outfile;

    double start_temp = 1.0;
    double max_temp = 1.0001;
    double temp_step = 1.0;
    int ntemps = ((max_temp-start_temp)/temp_step)+1;
#pragma omp parallel
    {
        double average_E = 0;
        double average_M = 0;
        double average_E2 = 0;
        double average_M2 = 0;
        double average_Mabs = 0;
        double E,M;
        double variance_M,variance_E;
        variance_E=variance_M = 0;
        long idum = -1*time(0);
        mat spinmatrix = zeros<mat>(n+2,n+2);
        vec averages = zeros<vec>(5);
        int accepted_flips = 0;
        vec w = zeros<vec>(5);
        vec temp = linspace<vec>(start_temp,max_temp,ntemps);
        double prob[n*n+1];
        for(int i = 0; i<=n*n;i++){
```

```
            prob[i] = 0;
        }


#pragma omp for
    for(int t = 0; t < ntemps; t++){
        /*Loop over temperatures*/
        averages(0) = 0; averages(1) = 0; averages(2) = 0;
        averages(3) = 0; averages(4) = 0;
        average_E = 0;
        average_M = 0;
        average_E2 = 0;
        average_M2 = 0;
        average_Mabs = 0;
        for(int p=0;p<5;p++){
            w(p) = exp(-(4*p-8)/temp(t));
        }
        E = M = 0;
        spinmatrix = init(0,n,E,M);
        for(int j = 0; j<N;j++){
            /*Loop over Monte Carlo cycles*/
            accepted_flips = metropolis(n, spinmatrix, E, M, w, &idum);
            if (j>0.1*N){
                averages(0) += E; averages(1) += E*E; averages(2) += fabs(M);
                averages(3) += M; averages(4) += M*M;
                int j = (E+2*n*n)/4;
                prob[j] +=1;
                /*
                outfile<<averages(0)/((double)j)<<"                "<< averages(2)/((double
                        "      "<<accepted_flips<<"   "<<probendl;
                  */
            }
        }
        /*Handling the results*/
        //outfile.open("probability2.txt");

        average_E = averages(0)/(0.9*((double) N));
        average_M = averages(3)/(0.9*((double) N));     //Note the use of abs(M)
        average_Mabs = averages(2)/(0.9*((double) N));
        average_E2 = averages(1)/(0.9*((double) N));
        average_M2 = averages(4)/(0.9*((double) N));
        variance_E = (average_E2 - average_E*average_E)/n/n;
        variance_M = (average_M2 - average_M*average_M)/n/n;
        cout<<"------------------------------"<<endl;
        cout<<"temp = "<<temp(t)<<" out of "<<max_temp<<endl;
        cout<<"average energy "<<average_E/n/n<<" heat capacity "<<variance_E/(temp(t)*tem
        cout<<"average magnetization "<<average_Mabs/n/n<<" magnetic suceptibility "<<\
                variance_M/(temp(t))<<endl;
        /*
        outfile<<average_E/n/n<<"       "<<variance_E/(temp(t)*temp(t))<<\
                "         "<<average_Mabs/n/n <<"            "<<\
                variance_M/(temp(t))<<"       "<<temp(t)<<endl;

        for(int i=0;i<=n*n;i++){
            outfile<<prob[i]/((double)0.9*N)<<endl;
        }
        outfile.close();
        */
    }
#pragma omp critical
    {
```

```
        //crude_mc += l_sum;
        //g_sigma += sum_sigma;
        //num_cores = omp_get_num_threads();
    }
    }
    outfile.close();
    //double stop = clock();
    //double diff = timediff(start,stop)*0.25;

    return 0;
}
```

.h - file

```
/*
 * File:   integrate.h
 * Author: fredrik
 *
 * Created on 2. november 2012, 10:30
 */

#include <cstdlib>
#include <omp.h>
#include <armadillo>
#include <cmath>
#include <iostream>
#include <time.h>
#include <fstream>

/*from lib.h. do'nt know what they do*/
#include <new>
#include <cstdio>
#include <cstdlib>
#include <cstring>


using namespace std;
using namespace arma;


#ifndef INTEGRATE_H
#define INTEGRATE_H

mat init(int high,int n,double &E, double &M);
double timediff(double time1, double time2);
double ran0(long *idum);
int metropolis(int n,mat &spinmatrix,double &E,double &M, vec w, long *idum);
void update_ghosts(mat &spinmatrix, int n, int a, int b);
void update_all(mat &spinmatrix, int n);
char *make_filename(int n, double temp);
double ran3(long *idum);
double ran2(long *idum);
#endif  /* INTEGRATE_H */



mat init(int high, int n,double &E, double &M)
{
        /*
        int a,b;
        a = int (1 + n*ran2(&idum));
```

```
        b = int (1 + n*ran2(&idum));
        //cout<<"a = "<<a<<" b = "<<b<<endl;
        */
        mat spinmatrix;

        if(high){
                //cout<<"hei"<<endl;
                spinmatrix.randu(n+2,n+2);
                for (int i =0;i<n+2;i++){
                        for(int j=0;j<n+2;j++){
                        spinmatrix(i,j) = (int) (2*spinmatrix(i,j));
                        if(spinmatrix(i,j)==0){spinmatrix(i,j)=-1;}
                        }
                }
  }
  else{spinmatrix.ones(n+2,n+2);}
  update_all(spinmatrix,n);
  for(int i = 1; i<=n; i++){
        for(int j = 1; j<= n; j++){
                E -= 0.5*((double) spinmatrix(i,j)*\
                        (spinmatrix(i-1,j)+spinmatrix(i+1,j)+spinmatrix(i,j+1)+spinmatrix(
                M += ((double) spinmatrix(i,j));
        }
  }
  return spinmatrix;
}

void update_all(mat &spinmatrix, int n){
        //Do not updddate all points every time!!!
        for(int i =1; i <= n; i++){
                spinmatrix(0,i) = spinmatrix(n,i);
                spinmatrix(n+1,i) = spinmatrix(1,i);
                spinmatrix(i,0) = spinmatrix(i,n);
                spinmatrix(i,n+1) = spinmatrix(i,1);
        }
        return ;
}

void update_ghosts(mat &spinmatrix, int n, int a, int b){
        /*Do not updddate all points every time!!!*/
        for(int i =1; i <= n; i++){
                spinmatrix(0,i) = spinmatrix(n,i);
                spinmatrix(n+1,i) = spinmatrix(1,i);
                spinmatrix(i,0) = spinmatrix(i,n);
                spinmatrix(i,n+1) = spinmatrix(i,1);
        }
        if(a==n || b==n || a==1 || b==1){
                if(a==1 && b==1){
                        spinmatrix(1,n+1) = spinmatrix(1,1);
                        spinmatrix(n+1,1) = spinmatrix(1,1);
                }
                else if(a==1 && b==n){
                        spinmatrix(1,0) = spinmatrix(1,n);
                        spinmatrix(n+1,n) = spinmatrix(1,n);
                }
                else if(a==n && b==1){
                        spinmatrix(0,1) = spinmatrix(n,1);
                        spinmatrix(n,n+1) = spinmatrix(n,1);
                }
                else if(a==n && b==n){
                        spinmatrix(n,0) = spinmatrix(n,n);
```

```
                        spinmatrix(0,n) = spinmatrix(n,n);
                }
                else if(a==1){
                        spinmatrix(n+1,b) = spinmatrix(1,b);
                }
                else if(a==n){
                        spinmatrix(0,b) = spinmatrix(n,b);
                }
                else if(b==1){
                        spinmatrix(a,n+1) = spinmatrix(a,1);
                }
                else{
                        spinmatrix(a,0) = spinmatrix(a,n);
                }
        }
        return ;
}
int metropolis(int n,mat &spinmatrix,double &E,double &M, vec w, long *idum){
        int a,b,dE,counter = 0;
        for (int x = 1; x <= n; x++){
                for(int y=1; y <= n; y++){
                        a = (int) (1 + (n)*ran0(idum));
                        b = (int) (1 + (n)*ran0(idum));
                        dE = 2*spinmatrix(a,b)*\
                        (spinmatrix(a+1,b)+spinmatrix(a-1,b)+spinmatrix(a,b+1)+spinmatrix(

                        if(ran0(idum) <= w(dE/4 + 2)){
                                spinmatrix(a,b) *= -1;
                                //cout<<"dE = "<<dE<<endl;
                                E += ((double) dE);
                                M += ((double) 2*spinmatrix(a,b));
                                update_ghosts(spinmatrix, n, a, b);
                                counter++;
                                //spinmatrix.print("asdf");
                        }
                }
        }
        return counter;
}

double timediff(double time1, double time2){
        // This function returns the elapsed time in milliseconds
        return ((time2 - time1)*1000)/CLOCKS_PER_SEC;
}
char *make_filename(int n, double temp){
        //Returns a filename saying something about the particular run.
        char* buffer = new char[60];
        sprintf(buffer,"isingresults_n%d_temp_%.4f.txt",n,temp);
        return buffer;
}
#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define MASK 123459876

double ran0(long *idum)
{
   long     k;
   double   ans;
```

```
    *idum ^= MASK;
    k = (*idum)/IQ;
    *idum = IA*(*idum - k*IQ) - IR*k;
    if(*idum < 0) *idum += IM;
    ans=AM*(*idum);
    *idum ^= MASK;
    return ans;
}
#undef IA
#undef IM
#undef AM
#undef IQ
#undef IR
#undef MASK
#define MBIG 1000000000
#define MSEED 161803398
#define MZ 0
#define FAC (1.0/MBIG)

double ran3(long *idum)
{
        /*DO NOT USE THIS GENERATOR! GIVES VALUES LARGER THAN 1 FOR SOME SEEDS*/
    static int        inext, inextp;
    static long       ma[56];       // value 56 is special, do not modify
    static int        iff = 0;
    long              mj, mk;
    int               i, ii, k;

    if(*idum < 0 || iff == 0) {                 // initialization
       iff    = 1;

       mj     = MSEED - (*idum < 0 ? -*idum : *idum);
       mj    %= MBIG;
       ma[55] = mj;                             // initialize ma[55]

       for(i = 1, mk = 1; i <= 54; i++) {       // initialize rest of table
          ii     = (21*i) % 55;
          ma[ii] = mk;
          mk     = mj - mk;
          if(mk < MZ) mk += MBIG;
          mj = ma[ii];
       }

       for(k = 1; k <= 4; k++) {    // randimize by "warming up" the generator
          for(i = 1; i <= 55; i++) {
             ma[i] -= ma[1 + (i + 30) % 55];
             if(ma[i] < MZ) ma[i] += MBIG;
          }
       }

       inext  =  0;              // prepare indices for first generator number
       inextp = 31;              // 31 is special
       *idum  = 1;
    }

    if(++inext == 56)  inext  = 1;
    if(++inextp == 56) inextp = 1;
    mj = ma[inext] - ma[inextp];
    if(mj < MZ) mj += MBIG;
    ma[inext] = mj;
```

```
    return mj*FAC;
}
#undef MBIG
#undef MSEED
#undef MZ
#undef FAC
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)

double ran2(long *idum)
{
    int             j;
    long            k;
    static long     idum2 = 123456789;
    static long     iy=0;
    static long     iv[NTAB];
    double          temp;

    if(*idum <= 0) {
        if(-(*idum) < 1) *idum = 1;
        else             *idum = -(*idum);
        idum2 = (*idum);
        for(j = NTAB + 7; j >= 0; j--) {
            k     = (*idum)/IQ1;
            *idum = IA1*(*idum - k*IQ1) - k*IR1;
            if(*idum < 0) *idum +=  IM1;
            if(j < NTAB)  iv[j]  = *idum;
        }
        iy=iv[0];
    }
    k     = (*idum)/IQ1;
    *idum = IA1*(*idum - k*IQ1) - k*IR1;
    if(*idum < 0) *idum += IM1;
    k     = idum2/IQ2;
    idum2 = IA2*(idum2 - k*IQ2) - k*IR2;
    if(idum2 < 0) idum2 += IM2;
    j     = iy/NDIV;
    iy    = iv[j] - idum2;
    iv[j] = *idum;
    if(iy < 1) iy += IMM1;
    if((temp = AM*iy) > RNMX) return RNMX;
    else return temp;
}
// End: function ran3()
```

script

```
"""
A simple script which runs an executable file in the same directory, collects the output-f
```

```
program into one file with a reasonable filename, and plots the results in the end.
"""

import os, glob, numpy as np
import matplotlib.pyplot as mpl

N = [20,40,60,80]
counter = [0]*len(N)
'''
j=0
for i in N:
        print "-------HER!----------"
        os.system('./kalle %d'%i)
        outfile = open('collected_results_ising_n_%d.txt'%i,'w')
        counter[j] = i
        kake = []
        for dings in sorted(glob.glob('isingresults_n%d*.txt'%i)):
                noe = open(dings,'r')
                kake.append(noe.read())
                outfile.write(kake[-1])
                os.remove(dings)
        outfile.close()
        j+=1
'''
mpl.figure(1)
n=0
for somefile in sorted(glob.glob('collected_results_ising*.txt')):
        print somefile
        infile = np.loadtxt(somefile)
        mpl.plot(infile[:,-1],infile[:,0],label='%d by %d' %(counter[n],counter[n]))
        mpl.hold('on')
        mpl.xlabel('temperature in units of kT/J')
        mpl.ylabel('average energy per particle')
        #mpl.figlegend((line1),'%d by %d' %(counter[n],counter[n]),'upper left')
        n +=1
mpl.legend(loc=2)

#mpl.savefig('filename.extension')

mpl.figure(2)
n=0
for somefile in sorted(glob.glob('collected_results_ising*.txt')):
        infile = np.loadtxt(somefile)
        mpl.plot(infile[:,-1],infile[:,1],label='%d by %d' %(N[n],N[n]))
        mpl.hold('on')
        mpl.xlabel('temperature in units of kT/J')
        mpl.ylabel('heat capacity per particle')
        n +=1

mpl.legend(loc=2)
#mpl.savefig('filename.extension')
mpl.figure(3)
n=0

for somefile in sorted(glob.glob('collected_results_ising*.txt')):
        infile = np.loadtxt(somefile)
        mpl.plot(infile[:,-1],infile[:,2],label='%d by %d' %(N[n],N[n]))
        mpl.hold('on')
        mpl.xlabel('temperature in units of kT/J')
        mpl.ylabel('average magnetization per particle')
        n +=1
```

```
mpl.legend(loc=1)
#mpl.savefig('filename.extension')
mpl.figure(4)
n=0

for somefile in sorted(glob.glob('collected_results_ising*.txt')):
        infile = np.loadtxt(somefile)
        mpl.plot(infile[:,-1],infile[:,1],label='%d by %d' %(N[n],N[n]))
        mpl.hold('on')
        mpl.xlabel('temperature in units of kT/J')
        mpl.ylabel('magnetic suceptibility per particle')
        n +=1

mpl.legend(loc=2)
#mpl.savefig('filename.extension')
mpl.show()
```