Assignment 2 Solutions

1. Estimate the effective hard-sphere radius of the Helium atom knowing only that (i) the solid and gas phases coexist at 297 K and 115 kbar , and (ii) the solid/gas transition in the hard-sphere model occurs at dimensionless pressure $p^* = 11.56$.

   Solution: The unit of pressure in the hard-sphere model is

   $$p_0 = \frac{k_\mathrm{B} T}{a^3}$$

   where a is the hard-sphere diameter (the closest separation of sphere centers). The solid-gas transition in this model occurs at dimensionless pressure

   $$p/p_0 = p^* = 11.56.$$

   Combining these,

   $$r = a/2 = \frac{1}{2}\left(\frac{p}{11.56\, k_\mathrm{B} T}\right)^{1/3},$$

   and using the measured ($T = 297$ K, $p = 115$ kbar) at coexistence, we obtain

   $$r = 0.31 \text{ Å}.$$

   This agrees with what is called the "covalent radius" of Helium. It is nice to see that this microscopic length can be obtained from two macroscopic measurements (coexistence temperature and pressure) and a pure number (11.56).

2. Let u and v be two elements of an $n \times n$ orthogonal matrix U. Find the joint probability distribution $f(u, v)$ for the uniform measure on U for three cases of u and v: (i) they are the same element, (ii) they are distinct but lie on the same row or column, (iii) they lie on distinct rows and columns. Only work out the limiting form for $n \to \infty$ and ignore normalization.

   Solution: Let

   $$X = \begin{bmatrix} w & x \\ y & z \end{bmatrix}$$

   be the 2×2 submatrix of U that includes the elements u and v (either on the same row/column or different). Let $\mathbf{a}$ and $\mathbf{b}$ be $(n-2)$-component vectors corresponding to the remaining elements in the two rows. We are interested in the volume element

$$dw\, dx\, dy\, dz\, d^{n-2}\mathbf{a}\, d^{n-2}\mathbf{b}\ \delta(w^2+x^2+a^2-1)\ \delta(y^2+z^2+b^2-1)\ \delta(wy+xz+\mathbf{a}\cdot\mathbf{b}),$$

where two of the delta-function factors impose the unit norm constraint on the rows and the third keeps them orthogonal. We are going to integrate-out $\mathbf{a}$ and $\mathbf{b}$ to obtain the distribution of the sub-matrix elements.

We first integrate over $\mathbf{a}$ and decompose

$$d^{n-2}\mathbf{a} = d^{n-3}\mathbf{a}_\perp\ da_\parallel,$$

where $a_\parallel$ is the component colinear with $\mathbf{b}$. This integral only involves two of the delta functions:

$$\begin{aligned} f &\propto \int d^{n-3}\mathbf{a}_\perp \int da_\parallel\ \delta(w^2+x^2+a_\parallel^2+a_\perp^2-1)\ \delta(wy+xz+a_\parallel b) \\ &\propto \int d^{n-3}\mathbf{a}_\perp\ \frac{1}{b}\ \delta\left(w^2+x^2+\left(\frac{wy+xz}{b}\right)^2+a_\perp^2-1\right) \\ &\propto a_\perp(b)^{n-5}/b, \end{aligned}$$

where

$$a_\perp(b)^2 = 1-w^2-x^2-\left(\frac{wy+xz}{b}\right)^2.$$

Next, including the third delta function, we integrate over $\mathbf{b}$:

$$\begin{aligned} f &\propto \int d^{n-2}\mathbf{b}\ \frac{a_\perp(b)^{n-5}}{b}\ \delta(y^2+z^2+b^2-1) \\ &\propto b^{n-5}\ a_\perp(b)^{n-5}, \end{aligned}$$

where

$$b^2 = 1-y^2-z^2.$$

Next we discover a very nice fact:

$$\begin{aligned} b^2\ a_\perp(b)^2 &= (1-y^2-z^2)(1-w^2-x^2)-(wy+xz)^2 \\ &= \det\begin{bmatrix} 1-w^2-x^2 & -wy-xz \\ -wy-xz & 1-y^2-z^2 \end{bmatrix} \\ &= \det(1-XX^T). \end{aligned}$$

This gives us the most compact way of writing the distribution of the sub-matrix elements:

$$f(X) \propto \det(1 - XX^T)^{\frac{n-5}{2}}.$$

As a counterpart of Archimedes' hat-box theorem, now for 2×2 sub-matrices within a uniformly sampled orthogonal matrix U, we see that the elements of the sub-matrix are uniformly distributed when U is 5×5.

To arrive at the two pair-distributions you were given on the course website, first expand the exact sub-matrix distribution for $XX^T = O(1/n)$, including all terms up to $1/n^2$ corrections. The result will be the Gaussian

$$e^{-(n/2)(w^2+x^2+y^2+z^2)}$$

times a polynomial factor. Next, replace two of the elements by u and v and integrate-out the other two.

3. Analytically calculate the $\alpha = 1$ Hadamard-model partition function Z to order β^2 (leading terms of the "high temperature expansion"). You will need to use the three results above, for the joint distribution of matrix elements. By taking derivatives of $\log Z$ obtain the mean energy to order β and heat capacity to order β^2. As a check, these should both scale as n^2 (the naive "volume" of the system). With n^2 divided out we will denote these $e(\beta)$ and $c(\beta)$. In a future assignment you will use these results to check your Hadamard simulation at high temperatures.

Solution:

First expand Z to order β^2:

$$\begin{aligned} Z &= \int dU \left(1 + \beta\sqrt{n}\sum_{ij}|U_{ij}| + \frac{1}{2}\beta^2 n \sum_{ij}\sum_{kl}|U_{ij}||U_{kl}| + \cdots\right) \\ &= 1 + \beta\sqrt{n}n^2\langle|u|\rangle + \\ &\quad \frac{1}{2}\beta^2 n\left(n^2\langle|u|^2\rangle + 2n^2(n-1)\langle|u||v|\rangle_1 + n^2(n-1)^2\langle|u||v|\rangle_2\right) + \cdots \end{aligned}$$

The angle brackets mean "uniform average over orthogonal matrices", and the subscripts 1 and 2 denote, respectively, elements on the same or different row/column. We compute these averages using the distributions f_1 and f_2 from the website, noting that these are even functions of u and v so the integrals over u and v may be restricted to the range $(0, \infty)$ and the absolute values above can be removed. For the averages over a single element we may use either f_1 or f_2. Here are the results:

$$\begin{aligned}
\sqrt{n}n^2\langle|u|\rangle &= n^2\left(\sqrt{\frac{2}{\pi}}+\frac{1}{2\sqrt{2\pi}\,n}+\frac{1}{16\sqrt{2\pi}\,n^2}+\cdots\right)\\
n^3\langle|u|^2\rangle &= n^2\\
2n^3(n-1)\langle|u||v|\rangle_1 &= n^3\left(\frac{4}{\pi}-\frac{4}{\pi\,n}+\cdots\right)\\
n^3(n-1)^2\langle|u||v|\rangle_2 &= n^4\left(\frac{2}{\pi}-\frac{3}{\pi\,n}+\frac{5}{4\pi\,n^2}+\cdots\right).
\end{aligned}$$

Only the second of these is a finite expression (the defining property of orthogonal matrices) — for the rest only as many terms are included as we need to calculate $\log Z$ to $O(\beta^2)$ and the leading order in “volume”, n^2:

$$\log Z = n^2\left(\beta\sqrt{\frac{2}{\pi}}+\frac{\beta^2}{2}\left(1-\frac{3}{\pi}\right)+O(\beta^3)\right)+O(n).$$

From this we obtain the high temperature limits of the energy and heat capacity (per matrix element):

$$\begin{aligned}
e(\beta) &= \left(-\frac{\partial}{\partial\beta}\log Z\right)/n^2 = -\sqrt{\frac{2}{\pi}}-\beta\left(1-\frac{3}{\pi}\right)+\cdots\\
c(\beta) &= \left(\beta^2\frac{\partial^2}{\partial\beta^2}\log Z\right)/n^2 = \beta^2\left(1-\frac{3}{\pi}\right)+\cdots
\end{aligned}$$

4. Learn how to wrap the efficient Givens-rotation function you worked out in the last assignment in a working piece of executable code in the language of you choice. Estimate the energy $e(\beta)$ and heat capacity $c(\beta)$ at $\beta = 5.5$ for $n = 12$. Demonstrate that your sampling time T for these averages is sufficient (greater than the mixing time) by checking that the statistical error for sampling time NT decreases as $1/\sqrt{N}$.

   Solution:

   I ended up writing everything in C, but look forward to seeing more interesting solutions! My C code is appended and will also be made available on the course Github site. It has a modular design that should make life easy in forthcoming assignments. The `main()` in this version is set up to check convergence with respect to the number of sweeps for a fixed $n = 12$ and $\beta = 5.5$. Here is the output:

```
sweeps  mean energy                      heat capacity

    16  -0.932093926 +/- 0.001156354    0.162263063 +/- 0.012572
    64  -0.941703509 +/- 0.002371654    0.394436733 +/- 0.072660
   256  -0.942983670 +/- 0.001784216    0.716368853 +/- 0.058590
  1024  -0.943288847 +/- 0.000799536    0.926331417 +/- 0.021191
  4096  -0.943328850 +/- 0.000389784    0.978457616 +/- 0.008574
 16384  -0.943297388 +/- 0.000280087    0.992133992 +/- 0.004568
 65536  -0.943147251 +/- 0.000103103    1.001298635 +/- 0.001949
262144  -0.943058885 +/- 0.000066475    1.001615650 +/- 0.000595
```

   When the number of sweeps exceeds 10^3 the error estimate decreases by about a factor of 2 whenever the number of samples (sweeps) is increased by a factor of 4 — as it should when the 20 blocks being used to estimate the error are statistically independent. This number of sweeps is therefore a rough estimate of the mixing time (for this n and β). Since heat capacity is a measure of the fluctuations in the energy, it is not surprising that the heat capacity has a larger error than the energy.

   There are two deficiencies in my code that have to be addressed when seeking higher precision results. First, a higher quality pseudo-random number generator should replace C's `rand()`. Second, after very many Givens rotations the orthogonality of U is compromised as a result of the inexact floating point arithmetic. One remedy for that is to periodically re-orthogonalize

U. An easier strategy, and the one in place now, is to periodically (at the start of each averaging block) initialize U to the identity matrix. This only works when the accumulation of floating point errors is not significant within the mixing time (the smallest averaging time).

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

#define NMAX 100
#define B 20

int n;
double h[NMAX][NMAX],u[NMAX],v[NMAX];
double energy,angle,beta,twopi,accept;


double urand()
  {
  return ((double)rand())/RAND_MAX;
  }


void rowrot(int i,int j,double a)
  {
  int k;
  double eold,enew,c,s;

  eold=0.;
  for(k=0;k<n;++k)
    {
    u[k]=h[i][k];
    v[k]=h[j][k];

    eold+=-fabs(u[k])-fabs(v[k]);
    }

  c=cos(a);
  s=sin(a);

  enew=0.;
  for(k=0;k<n;++k)
    {
    h[i][k]=c*u[k]+s*v[k];
    h[j][k]=-s*u[k]+c*v[k];

    enew+=-fabs(h[i][k])-fabs(h[j][k]);
    }

  if(exp(-beta*(enew-eold))>urand())
    {
    energy+=enew-eold;
    accept++;
    }
  else
    for(k=0;k<n;++k)
      {
      h[i][k]=u[k];
      h[j][k]=v[k];
      }
  }


void colrot(int i,int j,double a)
  {
  int k;
  double eold,enew,c,s;

  eold=0.;
  for(k=0;k<n;++k)
    {
    u[k]=h[k][i];
    v[k]=h[k][j];

    eold+=-fabs(u[k])-fabs(v[k]);
    }

  c=cos(a);
  s=sin(a);

  enew=0.;
  for(k=0;k<n;++k)
    {
    h[k][i]=c*u[k]+s*v[k];
    h[k][j]=-s*u[k]+c*v[k];

    enew+=-fabs(h[k][i])-fabs(h[k][j]);
    }

  if(exp(-beta*(enew-eold))>urand())
    {
    energy+=enew-eold;
    accept++;
    }
  else
    for(k=0;k<n;++k)
      {
      h[k][i]=u[k];
      h[k][j]=v[k];
      }
  }


double sweep()
  {
  int i,j;
  double a;

  accept=0;

  for(i=0;i<n-1;++i)
  for(j=i+1;j<n;++j)
    {
    a=angle*(urand()-.5);
    rowrot(i,j,a);

    a=angle*(urand()-.5);
    colrot(i,j,a);
    }

  return ((double)accept)/(n*(n-1));
  }


void init(int sweeps)
  {
  int i,j,s;

  for(i=0;i<n;++i)
  for(j=0;j<n;++j)
    h[i][j]=0.;

  for(i=0;i<n;++i)
    h[i][i]=sqrt((double)n);

  energy=-sqrt((double)n)*n;

  twopi=4.*acos(0.);
  angle=twopi;

  for(s=0;s<sweeps;++s)
        {
    if(sweep()<.5)
      angle*=.99;
    else
      {
      angle*=1.01;
      if(angle>twopi)
        angle=twopi;
      }
    }
  }


void average(int sweeps,double *aenergy,double *heatcap)
  {
  int s;
  double e,e2;

  init(sweeps);

  e=0.;
  e2=0.;

  for(s=0;s<sweeps;++s)
    {
    sweep();

    e+=energy;
    e2+=energy*energy;
    }

  e/=sweeps;
  e2/=sweeps;

  *aenergy=e/(n*n);
  *heatcap=beta*beta*(e2-e*e)/(n*n);
  }


void measure(int sweeps,double *eave,double *eerr,double *cave,double *cerr)
  {
  int b;
  double e,c,eave2,cave2;

  *eave=0.;
  eave2=0.;

  *cave=0.;
  cave2=0.;

  for(b=0;b<B;++b)
    {
    average(sweeps,&e,&c);

    *eave+=e;
    eave2+=e*e;

    *cave+=c;
    cave2+=c*c;
    }

  *eave/=B;
  eave2/=B;
  *eerr=sqrt((eave2-(*eave)*(*eave))/B);

  *cave/=B;
  cave2/=B;
  *cerr=sqrt((cave2-(*cave)*(*cave))/B);
  }


int main(int argc,char* argv[])
  {
  int p,sweeps;
  char *outfile;
  double eave,eerr,cave,cerr;
  FILE *fptr;

  if(argc==4)
    {
    n=atoi(argv[1]);
    beta=atof(argv[2]);
    outfile=argv[3];
    }
  else
    {
    printf("expected three arguments: n, beta, outfile\n");
    return 1;
    }

  fptr=fopen(outfile,"w");
  fprintf(fptr,"      sweeps  mean energy                     heat capacity\n\n");
  fclose(fptr);

  srand(time(NULL));

  sweeps=16;

  for(p=0;p<8;++p)
    {
    measure(sweeps,&eave,&eerr,&cave,&cerr);

    fptr=fopen(outfile,"a");
    fprintf(fptr,"%12d%14.9lf +/-%12.9lf %14.9lf +/-%12.9lf\n",sweeps,eave,eerr,cave,cerr);
    fclose(fptr);

    sweeps*=4;
    }

  return 0;
  }
```