PHYS-E0412, Computational Physics, Lecture 3, 22 January 2019

Ilja Makkonen

Aalto University
School of Science

Traditional numerical integration

We can use random numbers also for integration.

$$I = \int_a^b f(x)dx$$

But if we have a 1D-problem, everything is very easy and the traditional ways work:

Example: take an uniform grid and evaluate function at these points, and estimate

$$I = \frac{(b-a)}{N} \sum_{i=1}^{N} f(x_i) \quad + \mathcal{O}(h)$$

See Appendix 11A of Gould,Tobochnik and Christian for derivation of the errors.

So the error is proportional to grid spacing $h = (b-a)/N$

You can improve this with the Trapezoidal $\mathcal{O}(N^{-2})$ or Simpson's rule $\mathcal{O}(N^{-4})$

We can also take results from these and extrapolate to $h \to 0$

(This is the Romberg integration.)

Integration with random points

$$I = \int_a^b f(x)dx$$

In *d* dimensions, the number of points in quadrature scales like N_1^d , gets really heavy!

If you have in total N points, the Simpson error scales like $\mathcal{O}\left(N^{-4/d}\right)$

This is not very fast for large dimensions *d* !

But we could do this using random points (***x*** is here d dimensional, V is volume):

$$I = \frac{V_d}{N}\sum_{i=1}^{N} f(x_i) \quad +\mathcal{O}(N^{-1/2})$$

Does not have an obvious dependence on the dimension?

This is the so-called Monte Carlo integration.

Learning objectives for week 3

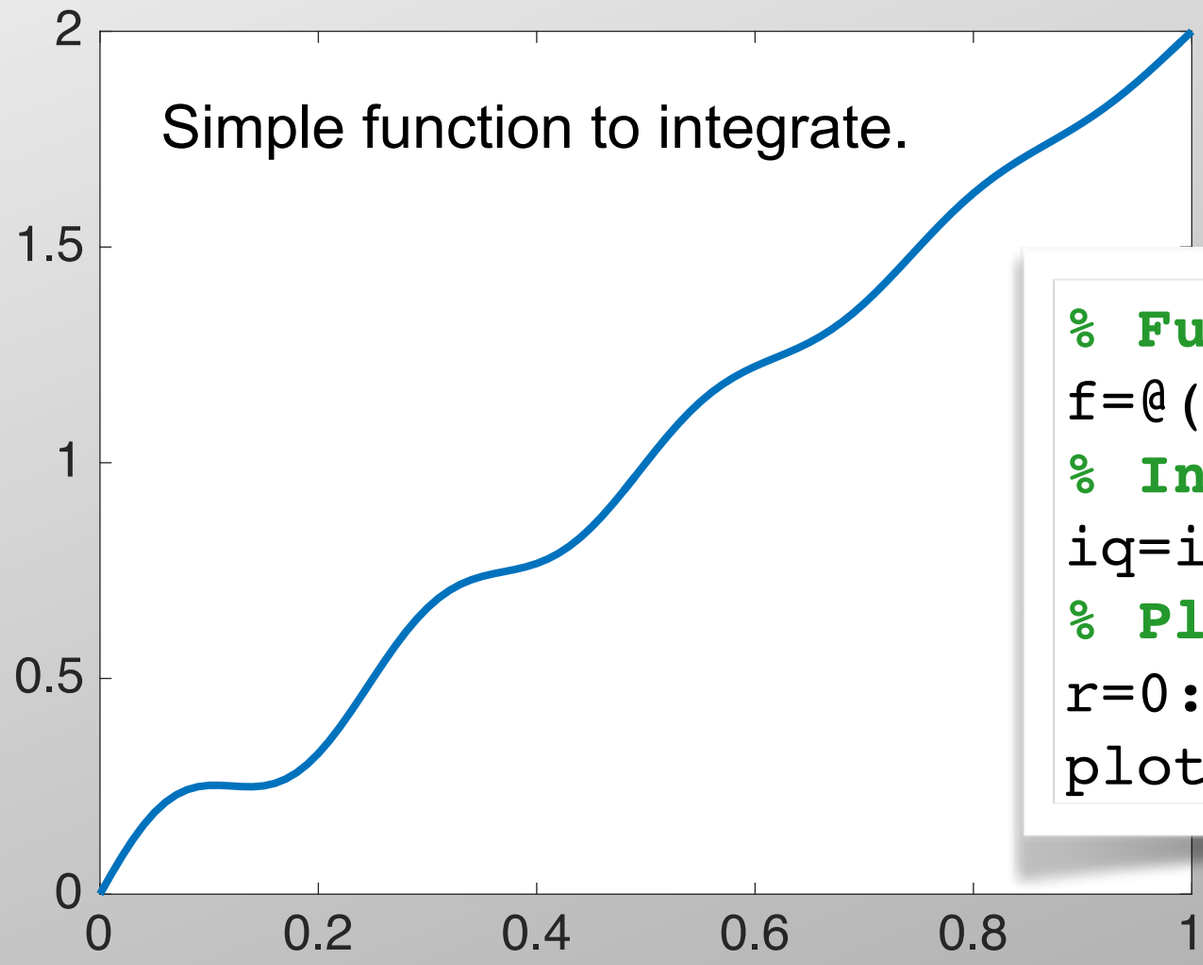
- Monte Carlo integration and error analysis
  - Importance sampling Monte Carlo
    - Inverse transform sampling (1D)
    - The Metropolis Method
      - Error estimation for the Metropolis

Homework 3:

Coulomb interaction energy between Gaussian charges in 3D using Metropolis importance sampling

Traditional integration in Matlab

There are **Matlab** routines for one, two, and three dimensional integrals but nothing beyond that.



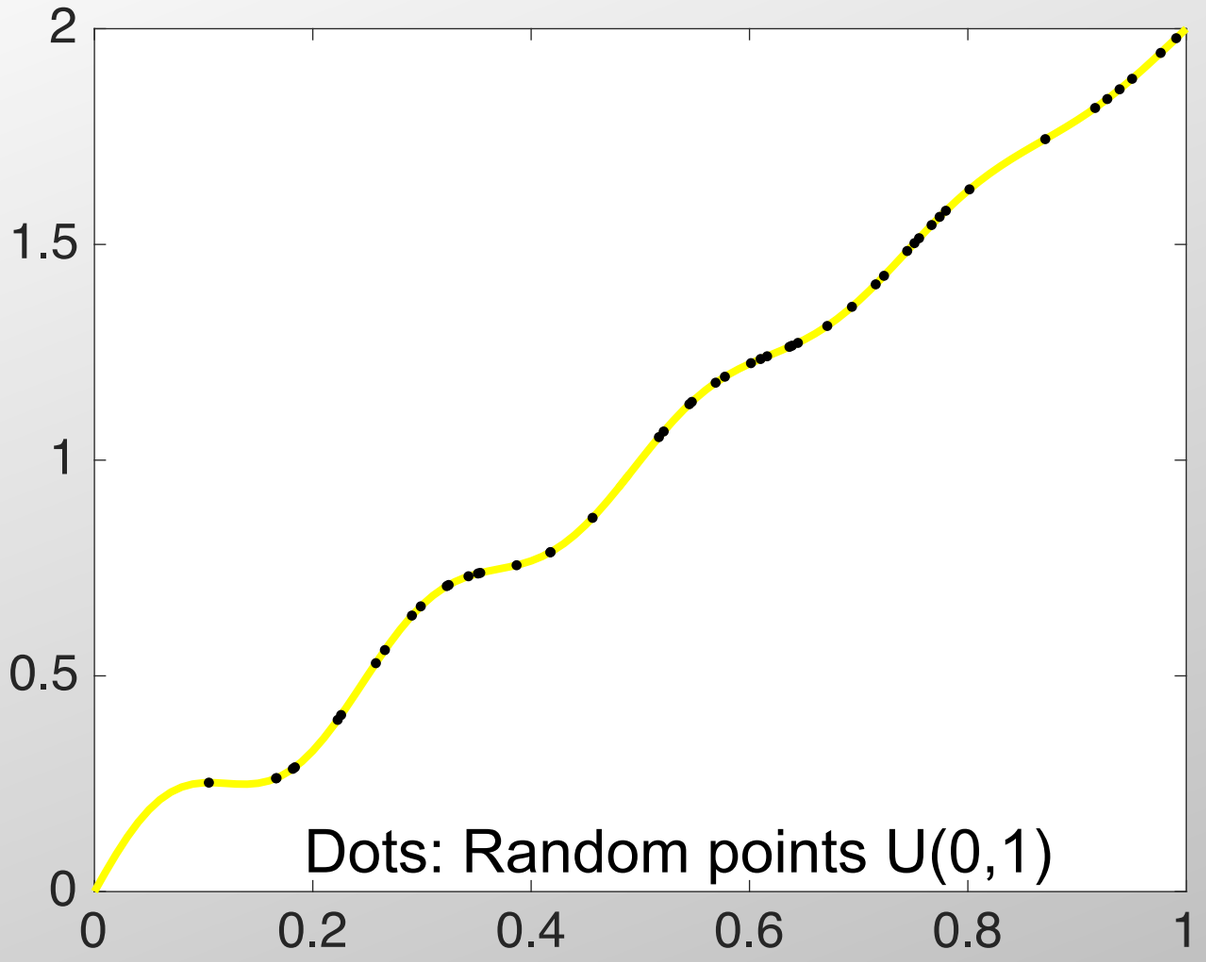
$$f(x) = 2x + \sin(8\pi x)\mathrm{erfc}(x)/10$$

```
% Function handle:
f=@(x) 2*x+.1*sin(8*pi*x).*erfc(x);
% Integration from zero to one:
iq=integral(f,0,1,'AbsTol',1e-10);
% Plot:
r=0:.01:1;
plot(r,feval(f,r),'linewidth',3)
```

Mathematica:

"The Wolfram Language function **NIntegrate** is a general numerical integrator. It can handle a wide range of one-dimensional and multidimensional integrals."

Integration, simple Monte Carlo



N=50 points:

matlab	MC	diff	Error Estimate
1.0034	1.1056	0.1022	0.087882
1.0034	0.88813	0.11523	0.075089
1.0034	1.0211	0.017694	0.080702
1.0034	1.0572	0.053852	0.0796
1.0034	1.1177	0.11439	0.069971
1.0034	1.0385	0.035179	0.074009
1.0034	0.98375	0.019607	0.073693
1.0034	1.1142	0.11079	0.075594
1.0034	0.94731	0.05605	0.07025
1.0034	1.0493	0.045959	0.068691

Mean error is: 0.067095

```
mean_error=0;
Ns=10;
for i=1:Ns,
    f_values=feval(f,rand(N,1));
    im=mean(f_values);
    em=std(f_values)./sqrt(N);
    disp(num2str([iq, im, abs(im-iq), em]))
    mean_error=mean_error+abs(im-iq);
end
mean_error=mean_error/Ns;
```

Average error is 0.07, and is consistent with the error estimate.

$$\Delta I \approx \frac{\sigma_f}{\sqrt{N}}$$

Seems to work, but how to make the error smaller?

Importance sampling to MC

Can we make σ_f smaller?

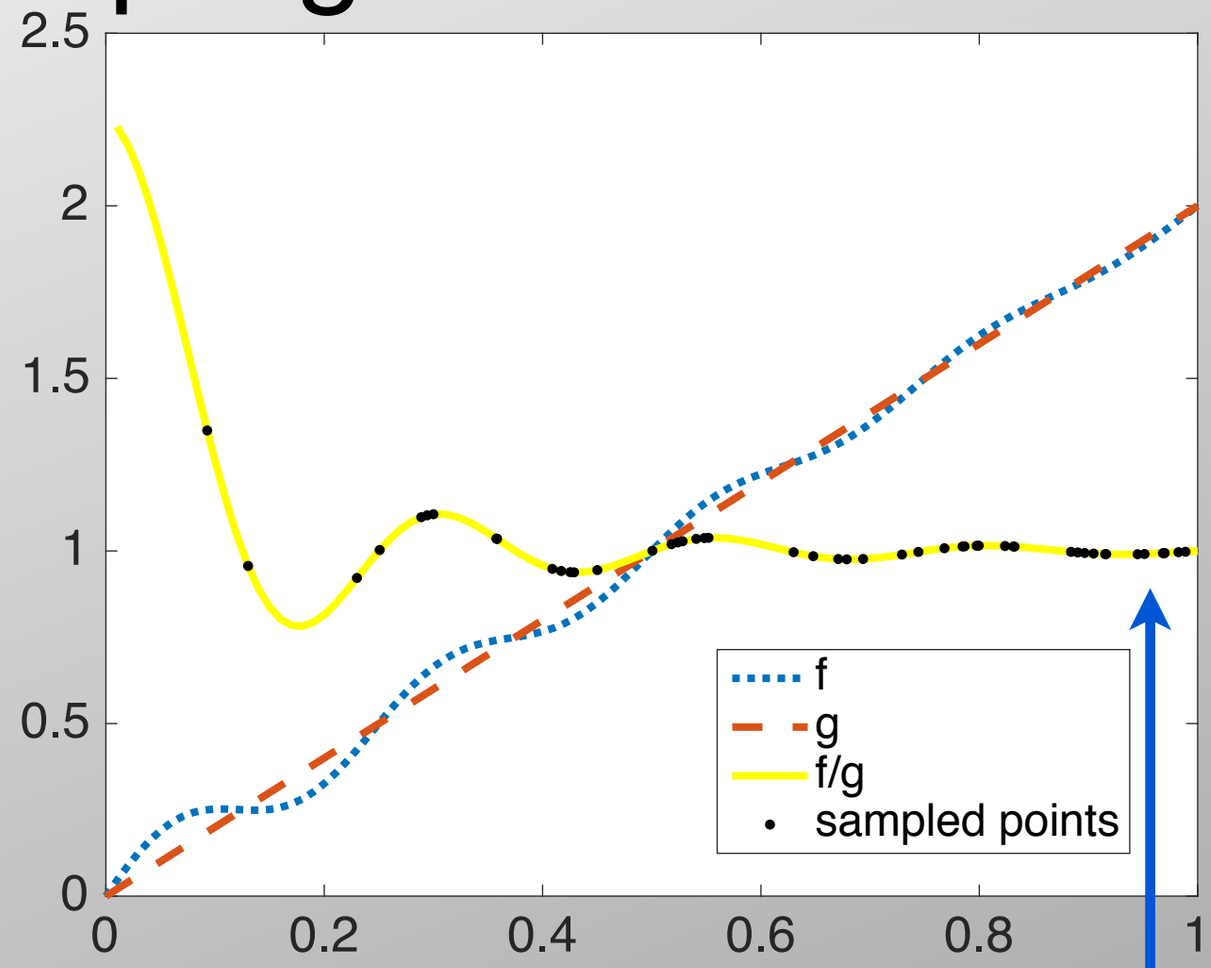
Yes, this is the so-called importance sampling.

$$I = \int_a^b f(x)dx = \int_a^b \frac{f(x)}{g(x)} g(x)dx$$

$$I = \frac{(b-a)}{N} \sum_{i=1}^{N} \frac{f(x_i)}{g(x_i)}$$

x_i are distributed according to $g(x)$

$$\int_a^b g(x)dx = 1$$



Notice that now more points on the right than left. Points sampled from function “g”.

Again N=50 points:

matlab	MC	diff	Error Estimate
1.0034	1.0091	0.0057527	0.0082505
1.0034	1.0281	0.024693	0.0064036
1.0034	1.0049	0.0015889	0.0039059
1.0034	1.0009	-0.002432	0.0071349
1.0034	0.99782	-0.0055342	0.022068
1.0034	0.99899	-0.0043729	0.0029746
1.0034	1.002	-0.0013426	0.004686
1.0034	0.99267	-0.010689	0.0061726
1.0034	0.99365	-0.0097037	0.0068877
1.0034	1.0208	0.017445	0.017196

Mean error is: 0.0083554

Error estimate is reasonable again

Simulation is now ten times more accurate (with same number of points):

Previous simulation:
Mean error is: 0.067095

Importance sampling to MC

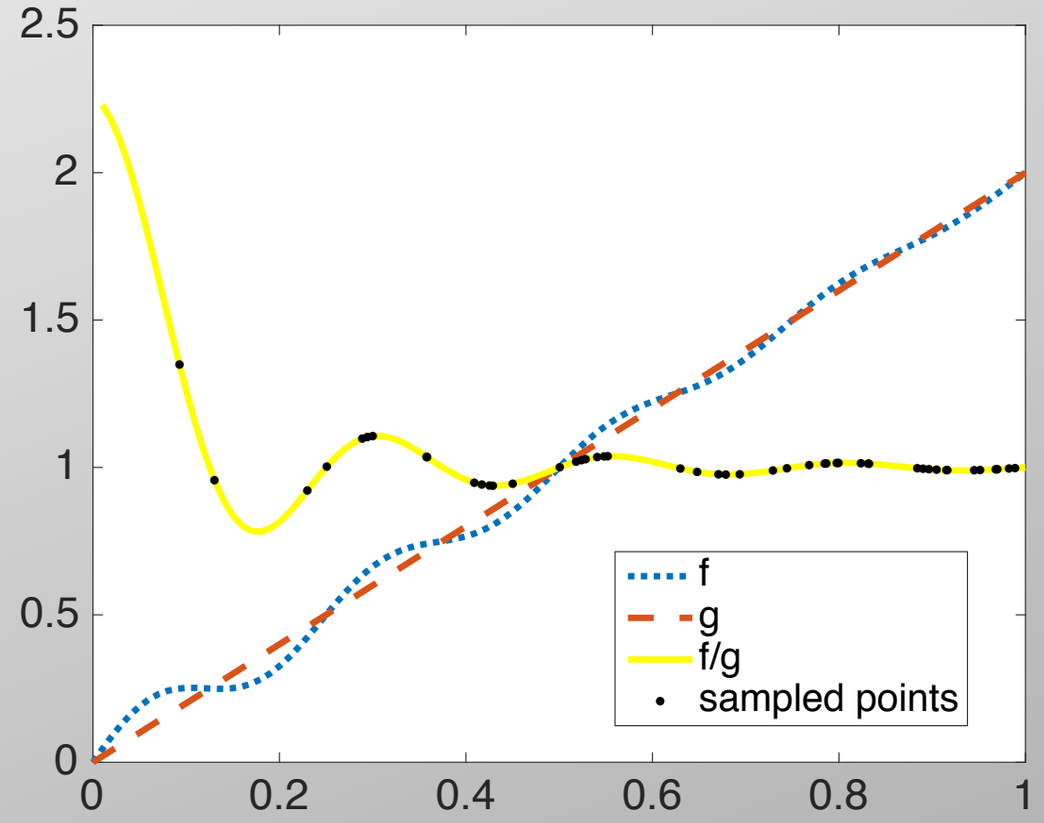
OK, how did that happen in practice?

$$I = \int_a^b f(x)dx = \int_a^b \frac{f(x)}{g(x)} g(x)dx$$

$$I = \frac{(b-a)}{N} \sum_{i=1}^{N} \frac{f(x_i)}{g(x_i)}$$

x_i are distributed according to $g(x)$

$$\int_a^b g(x)dx = 1$$



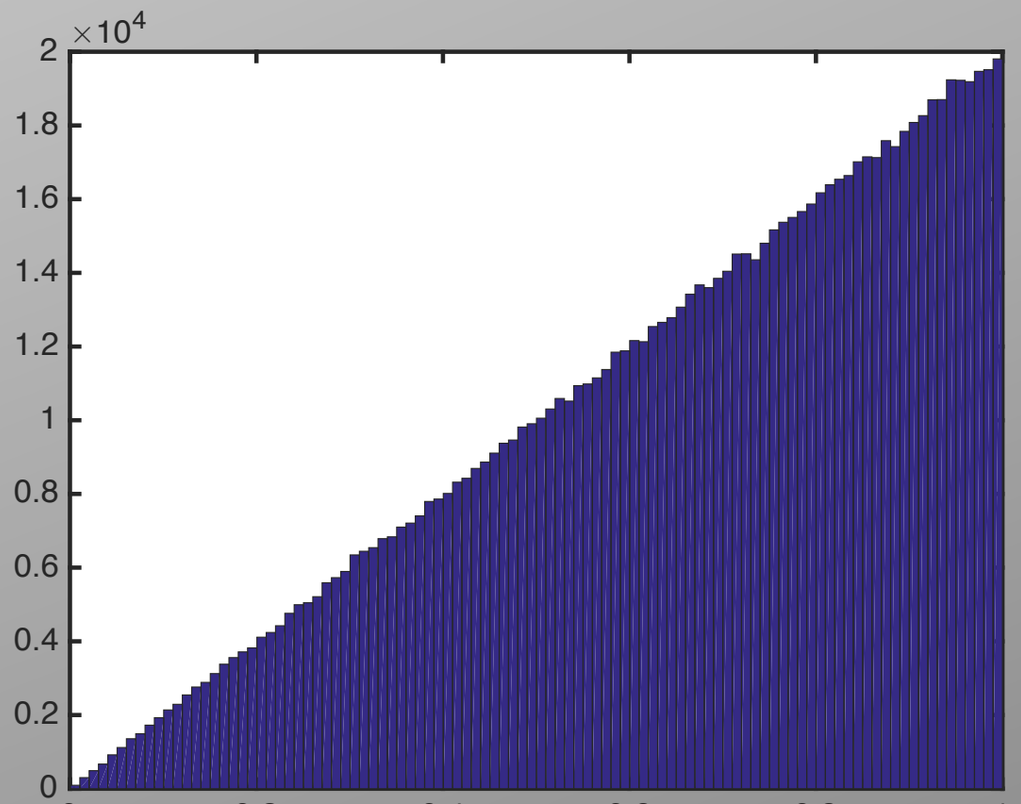

```
g=@(x) 2*x;
fg=@(x) 1+.1*sin(8*pi*x).*erfc(x)./(2*x);
```

Evaluation:

```
fg_values=feval(fg,sqrt(rand(N,1)));
im=mean(fg_values);
em=std(fg_values)./sqrt(N);
```

Why sqrt??

Try plotting: hist(sqrt(rand(10^6,1)),100), gives:



So these points are correctly distributed.

What was that trick?

Inverse distribution function to get x from g(x)

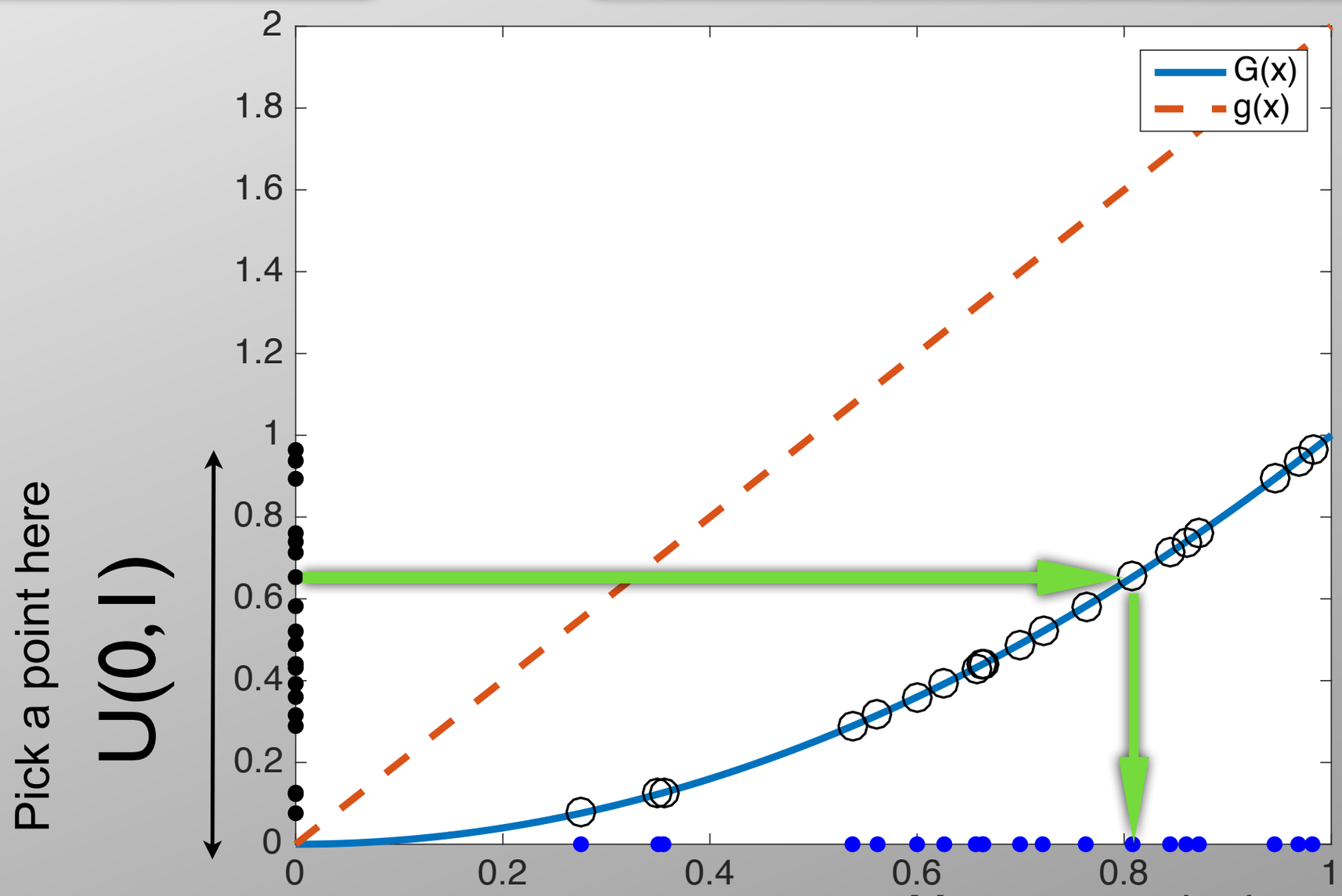
Cumulative distribution function:

$$G(x) = \int_{-\infty}^{x} g(x')dx'$$

If U is uniform, $G^{-1}(U)$ has g distribution.

For $g(x) = 2x$:

$G(x) = x^2$, and $G^{-1}(x) = \sqrt{x}$.



http://demonstrations.wolfram.com/TheMethodOfInverseTransforms/

This is rather hard for a general case (e.g. higher dimensions), we should use something else for sampling!

How to sample x from $w(x)$?

Metropolis algorithm is a very elegant solution for this.

Science 4 February 2000:
Vol. 287 no. 5454 p. 799
DOI: 10.1126/science.287.5454.799d

RANDOM SAMPLES

Algorithms for the Ages

"Great algorithms are the poetry of computation," says Francis Sullivan of the Institute for Defense Analyses' Center for Computing Sciences in Bowie, Maryland. He and Jack Dongarra of the University of Tennessee and Oak Ridge National Laboratory have put together a sampling that might have made Robert Frost beam with pride—had the poet been a computer jock. Their list of 10 algorithms having "the greatest influence on the development and practice of science and engineering in the 20th century" appears in the January/February issue of *Computing in Science & Engineering*. If you use a computer, some of these algorithms are no doubt crunching your data as you read this. The drum roll, please:

1946: The Metropolis Algorithm for Monte Carlo. Through the use of random processes, this algorithm offers an efficient way to stumble toward answers to problems that are too complicated to solve exactly.

1947: Simplex Method for Linear Programming. An elegant solution to a common problem in planning and decision-making.

1950: Krylov Subspace Iteration Method. A technique for rapidly solving the linear equations that abound in scientific computation.

How to sample x from $w(x)$?

Metropolis solution:

0. Start from point x_0.

1. Move by random δ: $\tilde{x}_i = x_i + \delta$

2. Accept with probability $r = \min\left(1, \frac{w(\tilde{x}_i)}{w(x_i)}\right)$.

3. If accepted: $x_{i+1} = \tilde{x}_i$, if not: $x_{i+1} = x_i$.

4. Return to 1.

δ can be, e.g, uniformly distributed.

δ should be adjusted to have $\approx 50\%$ acceptance.

Mathematical foundation of this is stochastic processes, Markov chains and all that.

Stationary probability distribution

Probability flow between any two points is in balance (so-called ”detailed balance”):

$$p(x)T(x \to x') = p(x')T(x' \to x)$$

***T*=transition probability**

Insert the rules of the Metropolis algorithm (transition = accepted times moved):

$$p(x)\min\left(1, \frac{w(x')}{w(x)}\right) \times M(x \to x') = p(x')\min\left(1, \frac{w(x)}{w(x')}\right) \times M(x' \to x)$$

***M*=moving probability**

Moves are symmetric in Metropolis: $M(x \to x') = M(x' \to x)$

$$\frac{p(x)}{p(x')} = \min\left(1, \frac{w(x)}{w(x')}\right) / \min\left(1, \frac{w(x')}{w(x)}\right) = \frac{w(x)}{w(x')}$$

So in balance, we sample the correct distribution.

The number of steps needed to reach balance is unknown and depends on the problem.

The moves should be such that all possible points can be reached in the simulation, counter example: function with two peaks separated by a huge distance.

Simple Metropolis example

$$I = \frac{\int_{-\infty}^{\infty} x^p \exp(-x^2) dx}{\int_{-\infty}^{\infty} \exp(-x^2) dx}$$

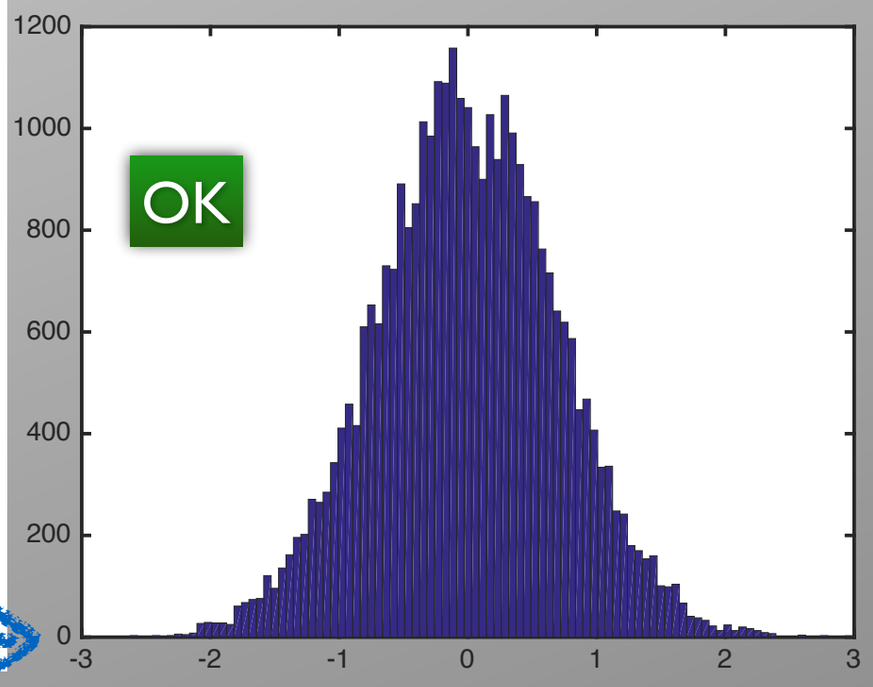
"Importance sampling" both with $g(x) = C \exp(-x^2)$

We get for the integral ratio $I \approx \frac{\langle x^p \rangle}{\langle 1 \rangle}$

(Normalization C does not matter.)

```
g=@(x) exp(-x.^2);
x=0;
delta=5;
g_now=feval(g,x);
mean_fs=zeros(1,10);
N=2^15;
xx=zeros(N,1);
accepted=0;
for i=1:N,
    xt=x+(rand-.5)*delta;
    g_trial=feval(g,xt);
    if g_trial/g_now>rand,
        x=xt;
        g_now=g_trial;
        accepted=accepted+1;
    else
        % rejected, nothing done
    end
    xx(i)=x; % we store coordinates to show correct sampling
    mean_fs=mean_fs+(x.^(1:10)-mean_fs)./i;
end
disp(strcat('acceptance=',num2str(accepted/N)))
hist(xx,100)
```

		Exact
p=1	0.0001	0
p=2	0.5078	0.5000
p=3	-0.0090	0
…	0.7850	0.7500
…	-0.0485	0
	2.0753	1.8750
	-0.3308	0
	7.8236	6.5625
	-2.6278	0
	37.7911	29.5313



Simple Metropolis example

$$I = \frac{\int_{-\infty}^{\infty} x^p \exp(-x^2) dx}{\int_{-\infty}^{\infty} \exp(-x^2) dx}$$

"Importance sampling" both with $g(x) = C \exp(-x^2)$

How about step size and error?

```
g=@(x) exp(-x.^2);
x=0;
delta=5;
g_now=feval(g,x);
mean_fs=zeros(1,10);
N=2^15;
xx=zeros(N,1);
accepted=0;
for i=1:N,
    xt=x+(rand-.5)*delta;
    g_trial=feval(g,xt);
    if g_trial/g_now>rand,
        x=xt;
        g_now=g_trial;
        accepted=accepted+1;
    else
        % rejected, nothing done
    end
    xx(i)=x; % we store coordinates to show correct sampling
    mean_fs=mean_fs+(x.^(1:10)-mean_fs)./i;
end
disp(strcat('acceptance=',num2str(accepted/N)))
hist(xx,100)
```

```
delta=5
acceptance=0.43576
          Error
   Estimate   Correct
     0.0039     0.0001
     0.0040     0.0078
     0.0080     0.0090
     0.0148     0.0350
     0.0340     0.0485
```

```
delta=0.5
acceptance=0.92795
          Error
   Estimate   Correct
     0.0042     0.0093
     0.0045     0.0670
     0.0094     0.0192
     0.0176     0.2481
     0.0404     0.1147
```

```
delta=50
acceptance=0.044128
          Error
   Estimate   Correct
     0.0039     0.0200
     0.0037     0.0022
     0.0069     0.0568
     0.0118     0.0489
     0.0254     0.1858
```

Small delta, large acceptance rate.

Large delta, small acceptance rate.

Error estimates are not realistic for those two! Data points are not independent!

Autocorrelation

The sequence of Metropolis in not independent, correlation between steps.

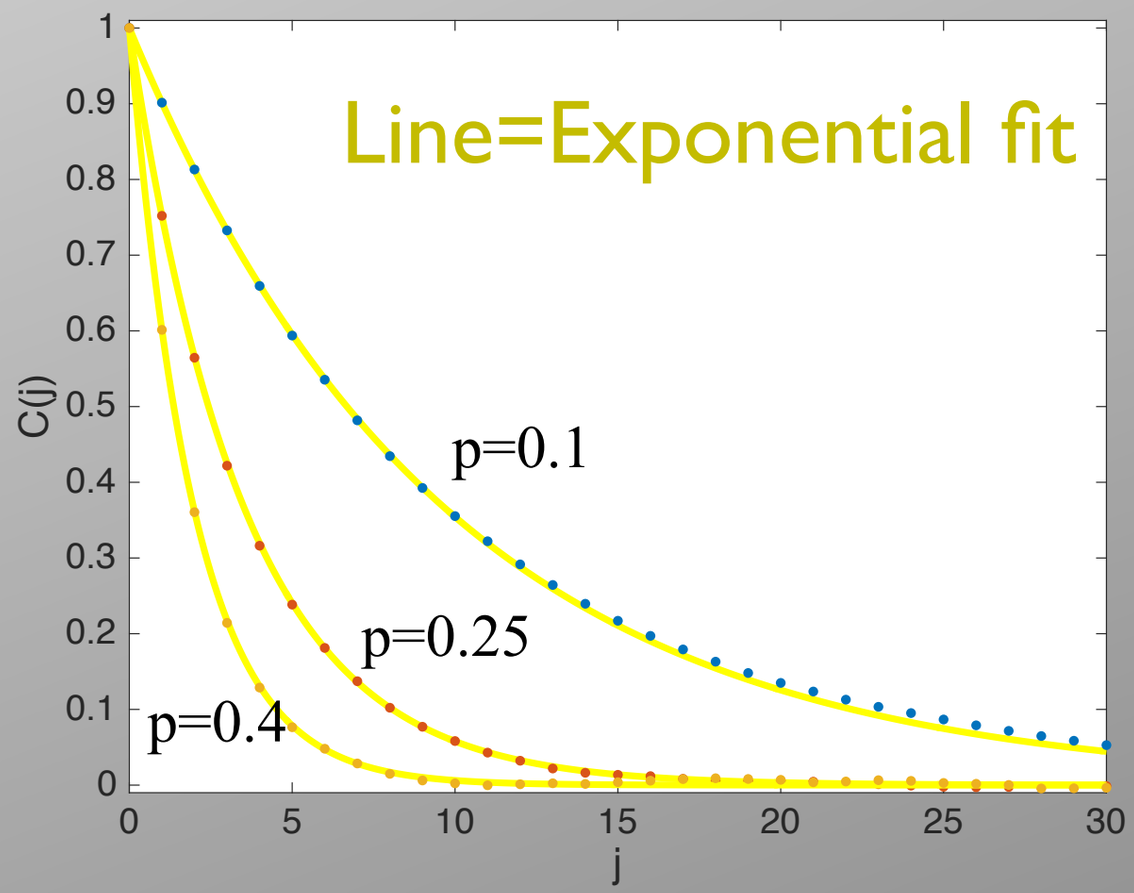
This can be measured using the autocorrelation function for a sequence of values f_i

$$C(j) = \frac{\langle f_i f_{i+j} \rangle - \langle f_i \rangle^2}{\langle f_i^2 \rangle - \langle f_i \rangle^2}$$

Simple process:

```
for i=2:N,
 f(i)=(1-p)*f(i-1)+p*(rand-.5);
end
```

gives:



C(0)=1

C(i)=0, for *i>0* if a random sequence.

Autocorrelation in practice

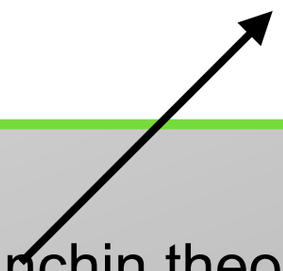
Algorithms for the Ages

1965: Fast Fourier Transform. Perhaps the most ubiquitous algorithm in use today, it breaks down waveforms (like sound) into periodic components.

```
function a = acc(x)
% This is a matlab/octave function for calculating autocorrelation.
% We use here fft, as the autocorrelation is easier in Fourier space.
% This, however, assumes periodicity for the data.
%
N=max(size(x));
Ex=mean(x);

a=real(ifft(abs(fft(x-Ex)).^2))'./N;

a=a./a(1);
```



“Wiener-Khinchin theorem”
(or simply the ”convolution theorem”)

N log(N) vs N^2 !

The bad method makes sense if only
a few values of j are needed from C(j).

```
function a = acc_bad(x)

N=max(size(x));
a=zeros(N,1);

Ex=mean(x);
Ex2=mean(x.^2);

for j=0:N-1,
 a(j+1)=mean(x(1:N-j).*x(1+j:N));
end

a=(a-Ex.^2)./(Ex2-Ex.^2);
```

Error estimate for the Metropolis

The correlation length in the Metropolis sequence can be estimated from:

$$C(\mathbf{1}) = \frac{\langle f_i f_{i+\mathbf{1}} \rangle - \langle f_i \rangle^2}{\langle f_i^2 \rangle - \langle f_i \rangle^2}$$

Notice that we need only one extra expectation value compared with the previous error estimates (no need for FFT). Number of steps needed to get an uncorrelated data point:

$$N_C \approx \frac{1 + C(1)}{1 - C(1)} \qquad \text{or} \qquad N_C \approx -1/\log(C(1))$$

This scales the number of data points we have, and we get an error estimate as:

$$\sigma / \sqrt{N/N_C} \longleftarrow$$ "Effective sample size"

Even better option:

- Perform M independent Metropolis simulations to get $\{I_i\}_{i=1}^{M}$.
- Calculate expectation value and variance from the result set.
- Use central limit theorem for error: $\Delta I \approx \sigma/\sqrt{M}$.

Potential of a Gaussian charge cloud

Charge distribution (3d): $\rho(r) = \exp(-r^2)/\sqrt{\pi^3} \quad \Rightarrow \quad \phi(r) = \int \frac{\rho(r')}{|\mathbf{r}-\mathbf{r}'|} d^3\mathbf{r}'.$

Cartesian integral by Mathematica does not work out:

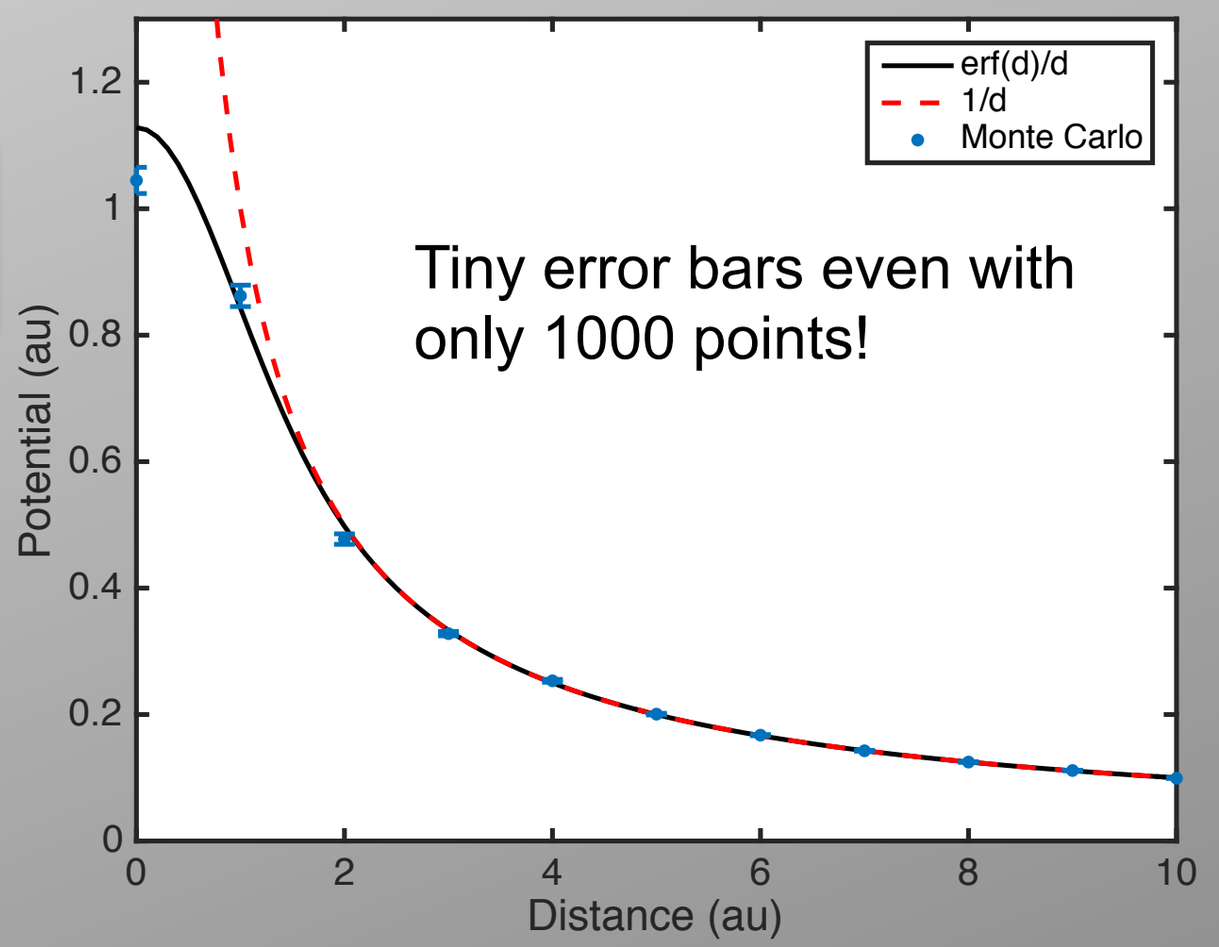
$$\int \mathrm{d}x\mathrm{d}y\mathrm{d}z \exp(-x^2-y^2-z^2)/\sqrt{\pi^3}/\sqrt{(x-d)^2+y^2+z^2} = \frac{\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{\frac{1}{2}\left(d^2-2dx-x^2-y^2\right)} K_0\left(\frac{1}{2}\left((d-x)^2+y^2\right)\right) dydx}{\pi^{3/2}}$$

Monte Carlo, sampling point charge positions r=(x,y,z) from Gaussian, evaluating Coulomb potential at point (d,0,0).

Moving one coordinate at time:

$$\frac{e^{-(x+\delta x)^2}}{e^{-x^2}} = e^{-\delta x(\delta x+2x)}$$

```
for i=1:N,
 for j=1:3,
  dx=(rand-.5)*delta;
  if exp(-dx*(dx+2*r(j)))>rand,
    r(j)=r(j)+dx;
    accepted=accepted+1;
  else
  % rejected, nothing done
 end
end
% Coulomb potential at point "at"
Vee=1./norm(r-at);
```



Homework 3

PHYS-E0412 Computational Physics :: Homework 3

Due date 29.1.2019 at 10 am

Metropolis importance sampling

Let us consider two Gaussian charge distributions in three dimensions, i.e. each with density

$$\rho(\mathbf{r}) \propto e^{-x^2-y^2-z^2}. \tag{1}$$

The total charge of each cloud is assumed to be one. Furthermore, to make things more interesting, we separate the charge clouds by a distance d. We are then asking what is the Coulomb interaction energy between the two charge clouds. This is effectively to evaluate the six-dimensional integral

$$U(d) := \frac{\int dx_1 dy_1 dz_1 \int dx_2 dy_2 dz_2 \; e^{-x_1^2-y_1^2-z_1^2} \frac{1}{\|\mathbf{r}_1-\mathbf{r}_2\|} e^{-(x_2-d)^2-y_2^2-z_2^2}}{\int dx_1 dy_1 dz_1 \int dx_2 dy_2 dz_2 \; e^{-x_1^2-y_1^2-z_1^2} \; e^{-(x_2-d)^2-y_2^2-z_2^2}} \;, \tag{2}$$

which is to be done by the Metropolis Monte Carlo integration.

Hint. The integral can be cast to the average

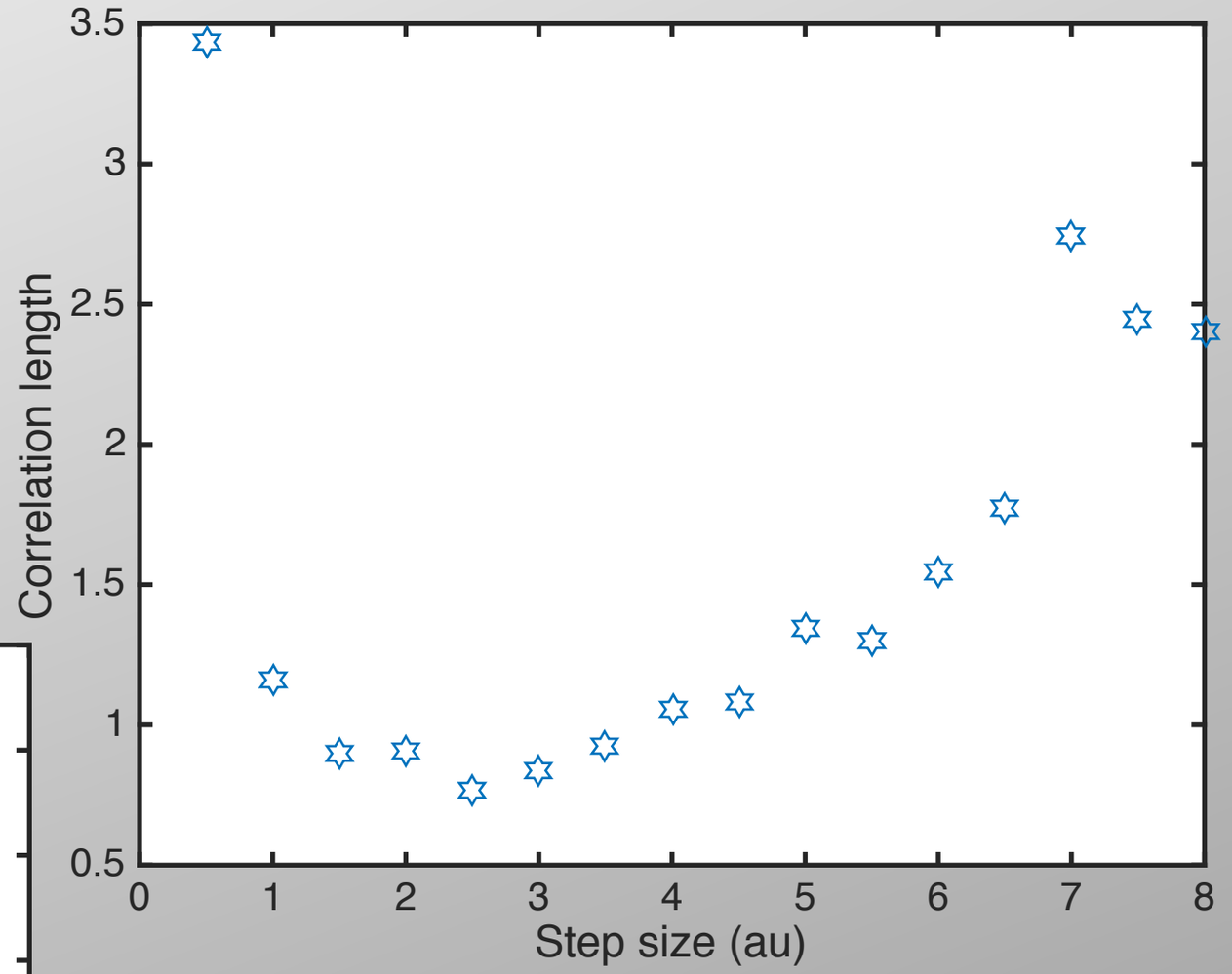
$$U(d) \approx \left\langle \frac{1}{\|\mathbf{r}_1 - \mathbf{r}_2\|} \right\rangle, \tag{3}$$

Step size, autocorrelation and acceptance:

Short steps, nearly all trials accepted.

Large steps give small acceptance.

Correlation length has the smallest value at around 60% acceptance for this case.



This is for a fixed distance d.

A simple rule of thumb: around 50% of the trial steps should be accepted for decent performance.

Second example

$$\frac{\int \sum_i x_i^2 \exp(-\sum_i x_i^2)}{\int \exp(-\sum_i x_i^2)}$$

N coordinates x

Running many times to get independent results

Error estimate from standard deviation of the results

Step size adjusted to get 50% acceptance rate

```
N=100;
[result, coordinates, delta]=metroN(zeros(N,1),50,5);

Ns=100;
res=zeros(Ns,1);
for i=1:Ns,
 [result, coordinates, delta]=metroN(coordinates,1000,delta);
 res(i)=result;
 [result, coordinates, delta]=metroN(coordinates,100,delta);
end

mean(res)
std(res)/sqrt(Ns)
disp(strcat('integral= ',num2str(mean(res)),' +/- ',num2str(std(res)/sqrt(Ns))))
```

```
function [ res,x_out,delta_out,acceptance ] = metroN( x,Ns,delta )
%metropolis integration of high-dimensional gaussian integral ratio:
% int sum x^2  exp(-sum x^2) / int exp(-sum x^2)
Nc=size(x,1);% Number of coordinates
accepted=0;
mean_fs=0;
for i=1:Ns, % Ns steps done
   for j=1:Nc, % Coordinates changed one by one
      xt=x(j)+(rand-.5)*delta;
      if exp(-xt^2+x(j)^2)>rand,
         x(j)=xt;
         accepted=accepted+1;
      else
         % rejected, nothing done
      end
   end
   mean_fs=mean_fs+(sum(x.^2)-mean_fs)./i;
end
disp(strcat('acceptance=',num2str(accepted/Ns/Nc)))
x_out=x;
res=mean_fs;
acceptance=accepted/Ns/Nc;
delta_out=delta*log(.5)/log(acceptance); % Step size scaled
end
```

How to sample x from $w(x)$?

Metropolis solution:

0. Start from point x_0.

1. Move by random δ: $\tilde{x}_i = x_i + \delta$

2. Accept with probability $r = \min\left(1, \frac{w(\tilde{x}_i)}{w(x_i)}\right)$.

3. If accepted: $x_{i+1} = \tilde{x}_i$, if not: $x_{i+1} = x_i$.

4. Return to 1.

δ can be, e.g, uniformly distributed.

δ should be adjusted to have $\approx 50\%$ acceptance.

Mathematical foundation of this is stochastic processes, Markov chains and all that.