Programmierpraktikum

Claudius Gros, SS2012

Institut für theoretische Physik
Goethe-University Frankfurt a.M.

Monte Carlo & Metropolis Sampling

statistical evaluation of integrals

- evaluation of multi-dimensional integrals

$$I = \int f(x_1, x_x, \ldots, x_N)\, dx_1\, dx_2 \cdots dx_N$$

- e.g. 1000 trapezoids per dimension

$$\left(10^3\right)^N = 10^{3N} \qquad \text{function evaluations}$$

- not feasible for large N

Monte Carlo sampling

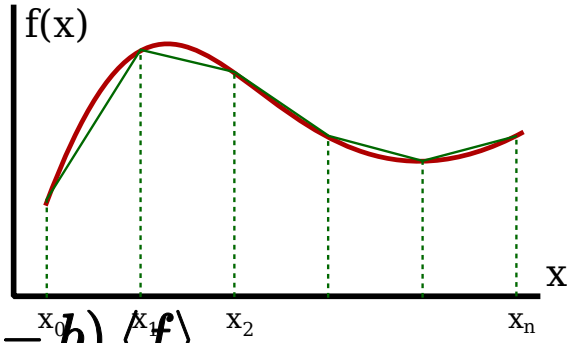
- evaluate the integral by sampling statistically the integrand $f(x_1, x_x, \ldots, x_N)$

Metropolis algorithm

- evaluate the integral by sampling statistically the most important terms of the integrand $f(x_1, x_x, \ldots, x_N)$ (importance sampling)

integration as averaging

$$I = \int_a^b f(x)\,dx = \sum_{i=1}^{N} f(x_i)\,\Delta x$$



- box width $\Delta x = \frac{b-a}{N}$

$$I = (a-b)\left(\frac{1}{N}\sum_i f(x_i)\right) = (a-b)\,\langle f \rangle$$

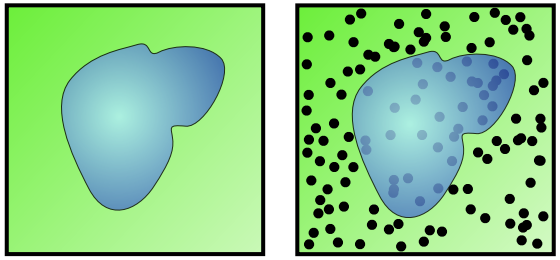
- **expectation value (average)** $\langle f \rangle$
  of integrand

$$I = V\,\langle f \rangle$$

- volume V
- expectation values may be evaluated statistically

area integration

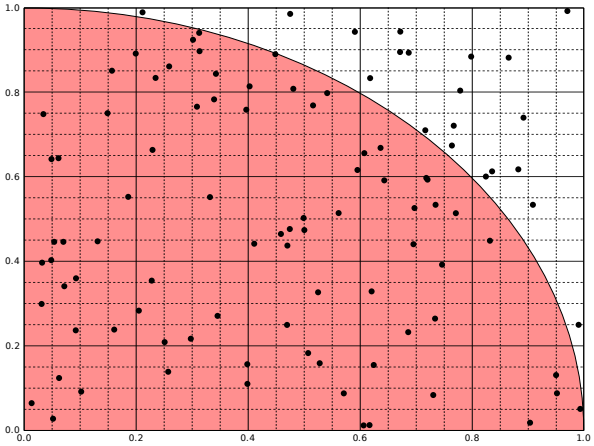
$$I = \int_{-a}^{a} f(x,y)\, dx\, dy$$



with

$$f(x,y) = \begin{cases} 1 & \text{inside} \\ 0 & \text{outside} \end{cases}$$

- evaluation of π



```
1. /** Evaluating \pi with simple Monte Carlo evaluation.
2.   */
3. public class MonteCarlo {
4.   static final double x0 = 0.0, x1 = 1.0;     // x-bounds
5.   static final double y0 = 0.0, y1 = 1.0;     // y-bounds
6.
7. public static void main(String[] args) {
8.   int N = 1000000;                            // number of draws
9.   int deltaN = N/10;                          // printout intervals
10.  double x, y, sum = 0;
11.
12. // --- loop over all random evaluations
13.  for (int i=0; i<N; i++)
14.    {
15.    x = Math.random();      // random draws
16.    y = Math.random();
17.    sum += MonteCarlo.myFunction(x,y);
18.    if ((i+1)%deltaN==0)
19.      System.out.printf("%8d %8.4f\n",i+1,4.0*sum/(i+1.0));
20.    }  // end of loop over all Monte Carlo draws
21. }  // end of MonteCarlo.main()
22.
23. /** The function to be evaluated.
```

```
24.   * Here the circle.
25.   */
26. public static double myFunction(double x, double y) {
27.   if ( (x<MonteCarlo.x0)||(x>MonteCarlo.x1)||
28.        (y<MonteCarlo.y0)||(y>MonteCarlo.y1) )
29.     {
30.     System.out.println("x or y value out of range");
31.     return 0.0;
32.     }
33.   if (y*y<=1.0-x*x)        // x*x+y*y=1 is the circle equation
34.      return 1.0;
35.   else
36.      return 0.0;
37.   } // end of MonteCarlo.myFunction()
38.
39. } // end of class MonteCarlo
```

basic statistics

statistics

- mean (expecatation value)

$$\langle f \rangle \; = \; \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

- variance σ^2
  standard deviation σ

$$\sigma^2 \; = \; \frac{1}{N} \sum_{i=1}^{N} \Big(f(x_i) - \langle f \rangle \Big)^2$$

- note: σ is not the standard deviation for the process,
  not the an estimate for the accuracy of $\langle f \rangle$
- question: how do we estimate the accuracy of the measurement: $\langle f \rangle$

Monte Carlo - convergence

data binning

- divide the N measurements into $k = 1, \ldots, K$ bins containing each $N_K = N/K$ data points

$$\langle f \rangle_k \; = \; \frac{1}{N_K} \sum_{i=(k-1)N_k}^{kN_k} f(x_i), \qquad\qquad k = 1, \ldots, K$$

- example: $N_k = 3$

$$\underbrace{f(x_1) \;\; f(x_2) \;\; f(x_3)}_{\langle f \rangle_1} \;\; \underbrace{f(x_4) \;\; f(x_5) \;\; f(x_6)}_{\langle f \rangle_2} \;\; \underbrace{f(x_7) \;\; f(x_8) \;\; f(x_9)}_{\langle f \rangle_3} \;\; \ldots$$

- the accuracy of $\langle f \rangle$ is given by the standard deviation of $\langle f \rangle_k$

$$\Delta\langle f\rangle \;=\; \sqrt{\frac{1}{K}\sum_{k=1}^{K}\Big(\langle f\rangle_k - \langle f\rangle\Big)^2}$$

- error estimate: $\Delta\langle f\rangle$
- usually $K \sim 10, 20$
  N_k large

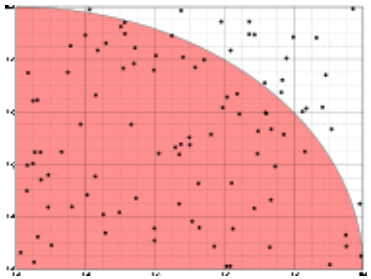
law of large numbers

$$\Delta\langle f\rangle \;\sim\; \frac{1}{\sqrt{N}}$$

- convergence very slow

Monte Carlo with error estimation

- evaluation of π



```
 1. /** Evaluating \pi with simple Monte Carlo evaluation,
 2.   * calculating variance and standard deviation (error).
 3.   * On input: number of measurements N.
 4.   */
 5. public class MonteCarloVariance {
 6.   static final double x0 = 0.0, x1 = 1.0;          // x-bounds
 7.   static final double y0 = 0.0, y1 = 1.0;          // y-bounds
 8.
 9. public static void main(String[] args) {
10.
11. // --- determine the number of draws from input
12.   int N = 0;
```

```
13.   if (args.length==1)
14.     N = Integer.parseInt(args[0]);
15.   else
16.     {
17.     System.out.println("Wrong number of input parameters");
18.     return;
19.     }
20.
21. // --- other parameters and variables
22.   int K = 20;                                   // number of bins
23.   int nK = N/K;                                 // number of data per bin
24.   double[] data = new double[N];                // storing all measurements
25.   double[] meanBin = new double[K];             // means of bins
26.   double mean = 0.0, sigma = 0.0;               // result: mean and error
27.   double x, y;                                  // variables
28.
29. // --- loop over all random evaluations
30.   for (int i=0; i<N; i++)
31.     {
32.     x = Math.random();                          // random draws
33.     y = Math.random();
34.     data[i] = MonteCarloVariance.myFunction(x,y);  // storing measurement
35.     }  // end of loop over all Monte Carlo draws
36.
```

```
37. // --- evaluate the overall mean and the mean of the bins
38.     for (int i=0; i<N; i++)
39.       mean += data[i];
40.     mean = mean/N;
41.
42.     for (int k=0; k<K; k++)
43.       {
44.       meanBin[k] = 0.0;
45.       for (int n=0; n<nK; n++)                // nK data points in each bin
46.         meanBin[k] += data[k*nK+n];
47.       meanBin[k] = meanBin[k]/nK;
48.       }  // end of loop over all bins
49.
50. // --- evaluate the error: variance of means of bins
51.     for (int k=0; k<K; k++)
52.       sigma += (meanBin[k]-mean)*(meanBin[k]-mean);
53.     sigma = Math.sqrt(sigma/K);
54.
55. // --- printout of result
56.     System.out.printf(" number of measurements : %d\n",N);
57.     System.out.printf(" number of bins         : %d\n",K);
58.     System.out.printf(" data points per bin    : %d\n",nK);
59.     System.out.printf(" estimate for pi        : %8.4f\n",4.0*mean);
60.     System.out.printf(" estimate for accuracy  : %8.4f\n",4.0*sigma);
```

```
61.
62. }  // end of MonteCarloVariance.main()
63.
64. /** The function to be evaluated.
65.   * Here the circle.
66.   */
67. public static double myFunction(double x, double y) {
68.   if ( (x<MonteCarloVariance.x0)||(x>MonteCarloVariance.x1)||
69.        (y<MonteCarloVariance.y0)||(y>MonteCarloVariance.y1) )
70.     {
71.     System.out.println("x or y value out of range");
72.     return 0.0;
73.     }
74.   if (y*y<=1.0-x*x)
75.      return 1.0;
76.   else
77.      return 0.0;
78.   } // end of MonteCarloVariance.myFunction()
79.
80. } // end of class MonteCarloVariance
```

integrals over distribution functions

- probability distribution function $\rho(x)$

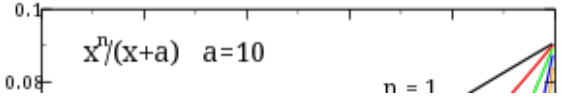
$$I = \int g(x)\,\rho(x)\,dx, \qquad \rho(x) \geq 0, \qquad \int \rho(x)\,dx = 1$$

- example: Boltzman factor $e^{\beta E_\lambda}$
  probability that a state with energy E_λ
  is thermally occupied

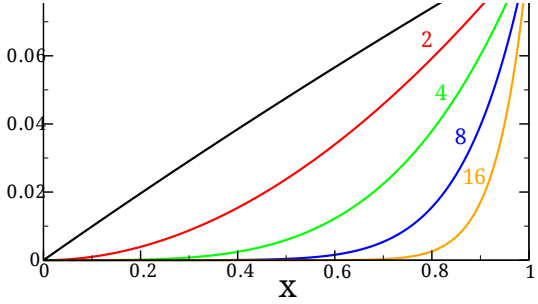
$$\rho = \frac{e^{\beta E_\lambda}}{Z}, \qquad Z = \sum_\alpha e^{\beta E_\alpha}$$

- inverse temperature $\beta = 1/(k_B T)$
  partition function Z

importance sampling



- the integral can be dominate by small regions of
  phase space with large values for $\rho(x)$



$$\rho(x) = \frac{1}{y_n} \frac{x^n}{x+a}, \qquad y_n = \int_0^1 \frac{x^n}{x+a}\, dx$$

- **importance sampling**
  Monte Carlo sampling of phase space with probability $\rho(x)$ to sample x

random walk through configuration space

- random walk

$$x_1, \; x_2, \; x_3, \; \ldots, \; x_N$$

- probability to visit a given phase space x

$$P_N(x) \; = \; \frac{1}{N} \sum_{n=1}^{N} \delta(x - x_n), \qquad \int P_N(x) dx = 1$$

- construct walk with correct limiting behaviour

$$\lim_{N \to \infty} P_N(x) \; = \; \rho(x)$$

- we then have

$$I = \int g(x)\rho(x)dx \; = \; \lim_{N\to\infty} \int g(x) P_N(x) dx$$

$$= \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \int g(x)\delta(x-x_n)dx$$

- and

$$I \; = \; \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} g(x_n)$$

- this is the basis of importance sampling:
  the important parts of phase space are visited (sampled) often
  the important parts of phase space seldomly
  by the random walk $\{x_n\}$

Metropolis algorithm

- method to generate a random walk $\{x_n\}$
  with the correct limiting behviour

$$P_N(x) = \frac{1}{N}\sum_{n=1}^{N}\delta(x-x_n), \qquad \lim_{N\to\infty} P_N(x) = \rho(x)$$

- **Markov chain**
  random walk where the probility to select x_{n+1}
  depends only on the x_n
  (and not on x_{n-1}, x_{n-2}, ... : no memory)

transition rules

- select candidate for $\tilde{x}_{n+1}$ randomly
- draw a random number $r \in [0,1]$

$$\text{if} \quad \left\{ \begin{array}{l} \rho(\tilde{x}_{n+1})/\rho(x_n) \geq r \\ \rho(\tilde{x}_{n+1})/\rho(x_n) \leq r \end{array} \right\} \quad \text{then} \quad \left\{ \begin{array}{rrcl} \textbf{accept candidate:} & x_{n+1} & = & \tilde{x}_{n+1} \\ \textbf{reject candidate:} & x_{n+1} & = & x_n \end{array} \right.$$

- repeat
- transition probability from state x to state y

$$P(x \to y) \; = \; \frac{\rho(y)}{\rho(x)}$$

detailed balance

- Markov chain $\{x_n\}$ with transition probability

$$P(x_n \to x_{n+1}) \;=\; \frac{\rho(x_{n+1})}{\rho(x_n)}$$

- two states x and y phase space
  with $\rho(y) > \rho(x)$
- (relative) number of transtions $x \to y$

$$T(x \to y) \;=\; P_N(x)\, P(x \to y) \;=\; P_N(x)$$

  since $\rho(y)/\rho(x) > 1$

- (relative) number of transtions $y \to x$

$$T(y \to x) \;=\; P_N(y)\, P(y \to x) \;=\; P_N(y)\rho(x)/\rho(y)$$

  since $\rho(x)/\rho(y) < 1$

detailed balance

- for a large number of walkers (random walks)
  of for $N \to \infty$
  the probability $P_N(x)$ will be determined
  by the steady-state condition (detailed balance)

$$T(y \to x) \;=\; T(x \to y)$$

  there would be otherwise an infinite accumulation
  leading to

$$P_N(x) \;=\; P_N(y)\rho(x)/\rho(y), \qquad \frac{P_N(x)}{P_N(y)} \;=\; \frac{\rho(x)}{\rho(y)}$$

- the Metropolis algorithm hence correctly yields

$$\lim_{N\to\infty} P_N(x) \;=\; \rho(x)$$

evaluation of relative properties

$$y_n = \int_0^1 \frac{x^n}{x+a}\, dx, \qquad y_{n+1} = \int_0^1 \frac{x^{n+1}}{x+a}\, dx = \int_0^1 x\, \frac{x^n}{x+a}\, dx$$

- with the Metropolis algorithm only relativ quantities can be evaluated

$$\frac{y_{n+1}}{y_n} = \int x\, \rho(x)\, dx, \qquad \rho(x) = \frac{1}{y_n}\, \frac{x^n}{x+a}$$

- use statistics (error estimate) from Monte Carlo program

```
1. /** Evaluating y_{n+1}/y_n, with y_n = \int_0^1 x^n/(a+x) dx
2.    * with Monte Carlo (Metropolis) algorithm.
3.    * On input: number of Monte Carlo steps N.
4.    */
5. public class Metropolis {
6.    static final double aPar = 10.0;                // parameter
7.    static final double nPow = 3.0;                 // power
8.
9. public static void main(String[] args) {
```

```
10.
11. // --- determine the number of draws from input
12.   int N = 0;
13.   if (args.length==1)
14.     N = Integer.parseInt(args[0]);
15.   else
16.     {
17.     System.out.println("Wrong number of input parameters");
18.     return;
19.     }
20.
21. // --- other parameters and variables
22.   int K = 20;                                   // number of bins
23.   int nK = N/K;                                 // number of data per bin
24.   double[] data = new double[N];                // storing all measurements
25.   double[] meanBin = new double[nK];            // means of bins
26.   double mean = 0.0, sigma = 0.0;               // result: mean and error
27.   double x = Math.random();                     // random starting value
28.   double y, ratio;                              // variables
29.   data[0] = x;                                  // first measurment
30.
31. // --- loop over all Monte Carlo steps
32.   for (int i=1; i<N; i++)
33.     {
```

```
34.     y = Math.random();                                  // random candiate
35.     ratio = Metropolis.myDensity(y)/Metropolis.myDensity(x);
36.     if (ratio>Math.random())
37.       x = y;                                            // accept candidate
38.     data[i] = x;                                        // storing measurement
39.     }  // end of loop over all Monte Carlo draws
40.
41. // --- evaluate the overall mean and the mean of the bins
42.     for (int i=0; i<N; i++)
43.       mean += data[i];
44.     mean = mean/N;
45.
46.     for (int k=0; k<K; k++)
47.       {
48.       meanBin[k] = 0.0;
49.       for (int n=0; n<nK; n++)                  // nK data points in each bin
50.         meanBin[k] += data[k*nK+n];
51.       meanBin[k] = meanBin[k]/nK;
52.       }  // end of loop over all bins
53.
54. // --- evaluate the error: variance of means of bins
55.     for (int k=0; k<K; k++)
56.       sigma += (meanBin[k]-mean)*(meanBin[k]-mean);
57.     sigma = Math.sqrt(sigma/K);
```

```
58.
59. // --- printout of result
60.     System.out.printf(" a  in  y^n/(x+a)       : %8.4f\n",aPar);
61.     System.out.printf(" n  in  y^n/(x+a)       : %8.4f\n",nPow);
62.     System.out.printf(" number of measurements : %d\n",N);
63.     System.out.printf(" number of bins         : %d\n",K);
64.     System.out.printf(" data points per bin    : %d\n",nK);
65.     System.out.printf(" y_{n+1}/y_n  estimate  : %8.4f\n",mean);
66.     System.out.printf(" estimate for accuracy  : %8.4f\n",sigma);
67.     System.out.printf("\n");
68.     System.out.printf(" ratio y_4/y_3 (exact)  : %8.4f\n",0.797490089);
69.
70. }  // end of Metropolis.main()
71.
72. /** The proability density.
73.   */
74. public static double myDensity(double x) {
75.  return Math.pow(x,nPow)/(x+aPar);
76.   } // end of Metropolis.myDensity()
77.
78. } // end of class Metropolis
```