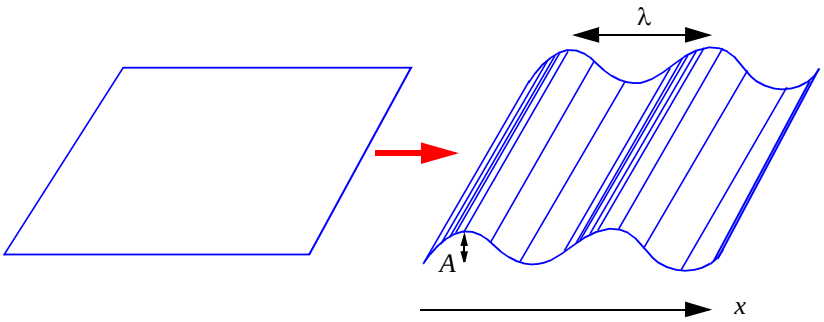
# Integration

• A sheet of corrugated roofing is constructed by a machine that presses a flat metallic sheet into one which has a cross section of a sine wave.

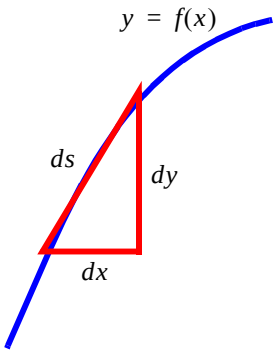



- We want to know how much material is needed to construct say 50 cm of corrugated roofing.

- Let's assume that the profile of the roofing is

$$f(x) = A\sin\frac{2\pi x}{\lambda} \ .$$

- We have to find the path length $s$ of $f(x)$ :




$$s = \int_0^{s_0} ds \, ,$$

- Now we can see that $(ds)^2 = (dx)^2 + (dy)^2 \rightarrow$

$$ds = \sqrt{(dx)^2 + (dy)^2} = dx\sqrt{1 + \left(\frac{dy}{dx}\right)^2} = dx\sqrt{1 + [f'(x)]^2}$$

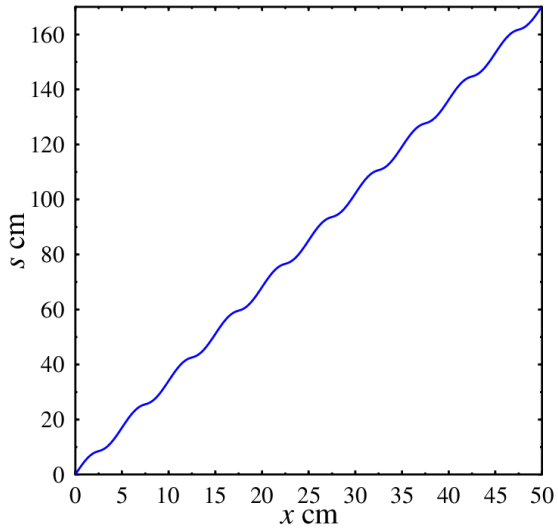
- Finally we get $s = \int_0^x dt\sqrt{1 + [f'(t)]^2} = \int_0^{x_0} dx\sqrt{1 + \left[\cos\frac{2\pi t}{\lambda}\right]^2}$



# Integration

- There is no closed form expression for this integral so we have to resort to numerical methods.

- The results using the Romberg method is shown below with values $A = 5$ cm and $\lambda = 10$ cm:




- Just for curiosity: a good approximation for the integral is

$$s(x) = 3.4035 \cdot x + 0.0703 + \text{oscillations}$$

# Integration

• Note that quite a lot can be done using the symbolic mathematics programs like Maple, Maxima etc:
(example below is made using Maxima from TeXmacs scientific editor;
see http://www.math.u-psud.fr/~anh/TeXmacs/TeXmacs.html)

```
Maxima 5.9.1 http://maxima.sourceforge.net
Using Lisp CMU Common Lisp 19a
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
This is a development version of Maxima. The function bug_report()
provides bug reporting information.
(%i1) f:1/(1+t^3);
```

$$(\%o1)\quad \frac{1}{t^3+1}$$

```
(%i2) assume(x>0,x^2-x+1>0);
```

$$(\%o2)\quad \left[x>0, x^2-x>-1\right]$$

```
(%i3) integrate(f,t,0,x);
```

$$(\%o3)\quad \frac{\sqrt{3}\arctan\left(\frac{\sqrt{3}}{3}\right)}{3}-\frac{\log\left(x^2-x+1\right)-2\sqrt{3}\arctan\left(\frac{2\sqrt{3}\,x-\sqrt{3}}{3}\right)-2\log\left(x+1\right)}{6}$$

```
(%i4)
```



# Integration

• In this chapter we deal mostly with numerical integration of functions with one variable (1D integrals).

• Multidimensional integrals are more complicated to compute.
- A popular method to compute them is the Monte Carlo integration.
- Actually, one can show that when the dimension of the problem is larger than 4 then $\varepsilon(n_{\mathrm{fe}})$ for the MC methods behaves more nicely than the same quantity for more conventional methods.
  - Here $\varepsilon(n_{\mathrm{fe}})$ is the error as a function of the number of function evaluations.

• Numerical integration is sometimes called **numerical quadrature**.

**Quadrature** \Quad"ra*ture, n. [L. quadratura: cf. F. quadrature.]

1. (Math.) The act of squaring; the finding of a square having the same area as some given curvilinear figure; as, the quadrature of a circle; the operation of finding an expression for the area of a figure bounded wholly or in part by a curved line, as by a curve, two ordinates, and the axis of abscissas.
[1913 Webster]

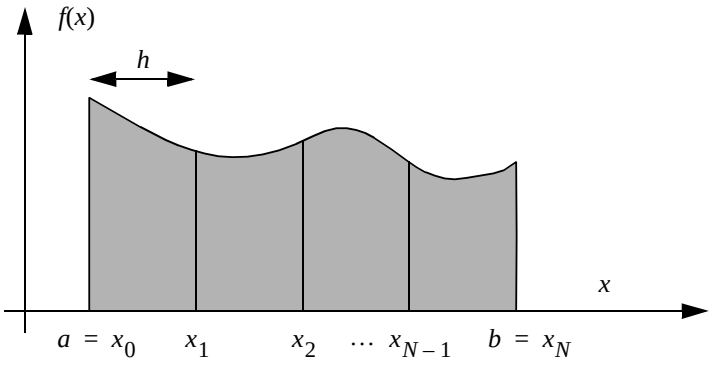
2. A quadrate; a square. --Milton.
[1913 Webster]

3. (Integral Calculus) The integral used in obtaining the area bounded by a curve; hence, the definite integral of the product of any function of one variable into the differential of that variable.
[1913 Webster]

# Integration

• Our purpose is to compute the value of the integral

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




- This the area under the curve $f(x)$ .

- The general philosophy behind all the numerical integration methods is
  1) to approximate the integrand by an easily integrable function $\int_a^b f(x)dx \approx \int_a^b P(x)dx$
  2) divide the interval into smaller pieces (**partition** into **subintervals**) $\int_a^b f(x)dx = \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} f(x)dx$

- The most common approximation is a polynomial.
  - Easy to integrate.
  - We already know how to interpolate functions using polynomials.

- Methods differ in
  1) the degree of approximation
  2) the way of partitioning

- As in the case of numerical differentiation there are two directions to gain accuracy:
  1) refine the partition (decrease step size $h$)
  2) go to higher degree polynomials
- In fact, the same method — Richardson extrapolation — is used in both integration and differentiation



# Integration: trapezoidal rule

• Let us assume we have an equidistant partition of the interval $[a, b]$:

$$x_i = x_0 + ih, \quad i = 0, 1, \ldots, N, \quad x_0 = a, \quad x_N = b$$

- Function values at the nodes are

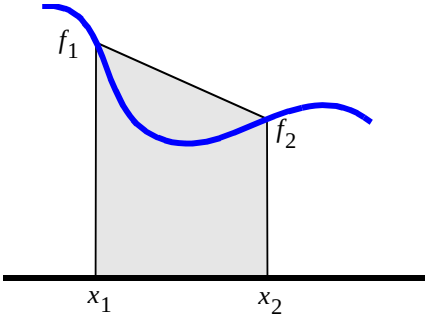
$$f(x_i) \equiv f_i$$

- The integral is computed as a sum of integrals in the subintervals $[x_i, x_{i+1}]$:

$$I = \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} f(x)dx$$

- The most simple (non-trivial) approximation is the linear one:

$$f(x) = f_{i-1} + \left(\frac{f_i - f_{i-1}}{x_i - x_{i-1}}\right)(x - x_{i-1})$$




- In this case the integral of the subinterval is simply the area of the trapezoid

$$\int_{x_1}^{x_2} f(x)dx \approx \frac{1}{2}(x_2 - x_1)(f_2 + f_1)$$

## Integration: trapezoidal rule

- And the total integral is

$$I \approx I_T = \frac{h}{2}\sum_{i=0}^{N}[f_i + f_{i+1}] = h\sum_{i=1}^{N-1} f_i + \frac{h}{2}[f_0 + f_N]$$

- The error in $I_T$ can be estimated based on what we know about the error in interpolation (let's use the same notation as in chapter 6; $n$ is now the polynomial order):

  for each $x \in [a,b]$ $\exists \xi(x) \in (a,b)$ so that

$$f(x) = P(x) + \frac{f^{(n)}(\xi(x))}{n!}\prod_{i=1}^{n}(x - x_i)$$

  - Now we have $n = 2 \rightarrow f(x) - P(x) = \frac{f''(\xi(x))}{2}(x-a)(x-b)$ (and $P(x) = \frac{(b-x)f(a) + (x-a)f(b)}{b-a}$ )

  - The error is $E_T = \int_a^b [f(x) - P(x)] = \int_a^b \frac{f''(\xi(x))}{2}(x-a)(x-b)$ **(1)**



## Integration: trapezoidal rule

- The integral mean-value theorem says that

$$\exists \xi \in [a,b] \text{ for which } \int_a^b w(x)f(x)dx = f(\xi)\int_a^b w(x)dx$$

- This means that we can write **(1)** as (we removed the explicit $x$ dependence from $\xi$):

$$E_T = \frac{f''(\xi)}{2}\int_a^b (x-a)(x-b) = \frac{f''(\xi)}{2}\left[-\frac{(b-a)^3}{6}\right]$$

```
(%i1) integrate((x-a)*(x-b),x,a,b);
(%o1) - (b^3 - 3 a b^2)/6 - (3 a^2 b - a^3)/6
(%i2) factor(%);
(%o2) - (b - a)^3/6
```

- The error term for the integral divided into subintervals is obtained as

$$E_T = \sum_{i=1}^{N}\left(-\frac{h^3}{12}\right)f''(\xi_i) = -\frac{h^3 N}{12}\left[\frac{1}{N}\sum_{i=1}^{N} f''(\xi_i)\right], \quad \xi_i \in [x_i, x_{i-1}]$$

- For the term in brackets

$$\min_{x\in[a,b]}[f'(x)] \le \frac{1}{N}\sum_{i=1}^{N} f''(\xi_i) \le \max_{x\in[a,b]}[f'(x)]$$

## Integration: trapezoidal rule

- Since $f$ is a very nice function (as they always are) its second derivative attains its all values between the minimum and maximum at some point in the interval $[a, b]$:

$$f''(\xi) = \frac{1}{N}\sum_{i=0}^{N} f''(\xi_i) \text{ for some value } \xi \in [a, b]$$

- So, we finally get the error term in form (and let's take the absolute value of it)

$$E_T = \frac{h^3 N}{12} f''(\xi) = \frac{h^2 (b-a)}{12} f''(\xi) \quad \text{for some } \xi \in [a, b].$$

- Coding the trapeziodal rule is a piece of cake:

```
double Trapezoid(double (*f)(double x), double a, double b, int n)
     {
          int i;
          double h, result;

          h = (b-a)/(double)n;
          result = 0.5*(f(a)+f(b));

          for (i=1; i<n; i++) result += f(a+i*h);

          return(h*result);
     }
```



## Integration: trapezoidal rule

- For example take the Gaussian $f(x) = e^{-x^2}$ on $[0, 1]$:

| | |
|---|---|
| For $N = 60$ the result is | $0.7468071$ |
| For $N = 500$ the result is | $0.7468238$ |
| The correct answer is | $0.7468241$ with seven decimals. |

- Let's estimate the error using the previous expression:

$$f'(x) = -2xe^{-x^2}, \quad f''(x) = (4x^2 - 2)e^{-x^2}$$

It is easy to see that $|f''(x)| \le 2$ on $[0, 1]$.

$$\rightarrow E_T \le \frac{h^2}{6}$$

- To have an error of at most $0.5\times10^{-4}$ we require that $h = (b-a)/N \le 0.01732$ or $N \ge 58$

- Using the routine with n=58 we obtain 0.7468059 which is incorrect in the 5th digit.

- Thus, with the error expression we can estimate how many points we need in order the get the desired accuracy.

- However, this requires that we can estimate the second derivative, which is not always the case.

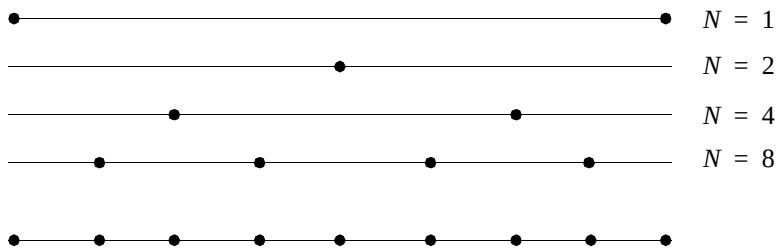
## Integration: trapezoidal rule

- However, as we look at the formula for the trapezoidal rule

$$I_{\mathrm{T}} = h \sum_{i=1}^{N-1} f_i + \frac{h}{2}[f_0 + f_N]$$

we can see that it is easy to add more points to the expression without the need to redo the computation from the beginning:

1) Start with using only the end points $a$ and $b$.
2) Add more points between the old ones and update $I_{\mathrm{T}}$ accordingly until its value does not change more than a predetermined value.






## Integration: trapezoidal rule

- The formula for the iterated trapezoidal rule is easily derived.

  - Let's assume that the interval $[a, b]$ is divided into $2^N$ parts:

$$I_{\mathrm{T}}(N) = h \sum_{i=1}^{2^N-1} f(a + ih) + \frac{h}{2}[f(a) + f(b)]$$

  - We need the method to calculate $I_{\mathrm{T}}(N)$ from $I_{\mathrm{T}}(N-1)$:

$$I_{\mathrm{T}}(N) = \frac{1}{2} I_{\mathrm{T}}(N-1) + \left[ I_{\mathrm{T}}(N) - \frac{1}{2} I_{\mathrm{T}}(N-1) \right]$$

  - The bracketed expression should be calculated with as little extra computation as possible:

$$I_{\mathrm{T}}(N) = h \sum_{i=1}^{2^N-1} f(a + ih) + C$$

$$I_{\mathrm{T}}(N-1) = 2h \sum_{i=1}^{2^{N-1}-1} f(a + 2ih) + 2C$$

$$C = \frac{h}{2}[f(a) + f(b)]$$

## Integration: trapezoidal rule

- By subtraction we get

$$I_{\mathrm{T}}(N) - \frac{1}{2} I_{\mathrm{T}}(N-1) = h \sum_{i=1}^{2^N - 1} f(a + ih) - h \sum_{i=1}^{2^{N-1} - 1} f(a + 2ih) = h \sum_{i=1}^{2^{N-1}} f(a + (2i-1)h)$$

- So, the iterative trapezoidal formula is

$$I_{\mathrm{T}}(N) = \frac{1}{2} I_{\mathrm{T}}(N-1) + h \sum_{i=1}^{2^{N-1}} f(a + (2i-1)h), \; N > 0$$

$$h = \frac{b-a}{2^N}, \quad I_T(0) = \frac{1}{2}(b-a)[f(a) + f(b)]$$



## Integration: trapezoidal rule

- And in C (notation is a bit different; in anticipation of the Romberg method)

```
void RecTrapezoid(double (*f)(double x), double a, double b, int N, double *R)
     {
          int i, j, k, kmax=1;
          double h, sum; h = b-a;
          /* Value of R(0,0) */
          R[0] = (h/2.0)*(f(a)+f(b));
          /* Successive approximations R(n,0) */
          for (i=1;i<=N; i++) {
               h = h/2.0;
               sum = 0;
               kmax = kmax*2;
               for (k=1; k<=kmax-1;k+=2)
                    sum += f(a+k*h);
               R[i] = 0.5*R[i-1]+sum*h;
          }
     }
```

## Integration: trapezoidal rule

- Example of application

The procedure *RecTrapezoid* was used to calculate the value of π by evaluating the integral

$$\pi \approx \int_0^1 \frac{4}{1+x^2} dx$$

The following output is obtained with $N = 9$

```
R(0,0) =    3.000000000000
R(1,0) =    3.100000000000
R(2,0) =    3.131176470588
R(3,0) =    3.138988494491
R(4,0) =    3.140941612041
R(5,0) =    3.141429893175
R(6,0) =    3.141551963486
R(7,0) =    3.141582481064
R(8,0) =    3.141590110458
R(9,0) =    3.141592017807
```

The approximation is correct in the sixth decimal. The correct value is 3.141592654 with nine decimals.



## Integration: higher order methods

• Taking more points into the integral of the subinterval $\int_{x_{i-1}}^{x_i} f(x)dx$ we can develop higher order methods

- Using three points we get the Simpson rule:

$$\int_{x_1}^{x_3} f(x)dx = h\left[\frac{1}{3}f_1 + \frac{4}{3}f_2 + \frac{1}{3}f_3\right] + O(h^5 f^{(4)})$$

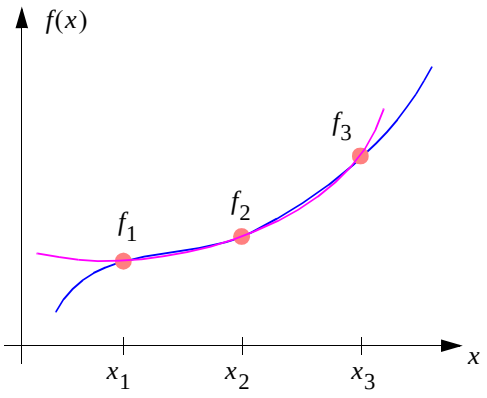
- Here we approximate the integrand with a parabola in $[x_1, x_3]$;

$$f(x) = f_2 + \frac{f_3 - f_1}{2h}(x - x_2) + \frac{f_3 - 2f_2 + f_1}{2h^2}(x - x_2)^2 + O((x_2 - x)^3)$$

- This is easily integrated as

$$\int_{x_1}^{x_3} f(x)dx = \int_{x_1}^{x_3} \left[f_2 + \frac{f_3 - f_1}{2h}(x - x_2) + \frac{f_3 - 2f_2 + f_1}{2h^2}(x - x_2)^2\right] dx$$

$$= \int_{-h}^{h} \left[f_2 + \frac{f_3 - f_1}{2h}x + \frac{f_3 - 2f_2 + f_1}{2h^2}x^2\right] dx$$




$$P_N(x) = \sum_{i=0}^{N} \left\{ f[x_0, x_1, \ldots, x_i] \prod_{j=0}^{i-1} (x - x_j) \right\}$$

## Integration: higher order methods

- And by summing the subintegrals such that they do not overlap we get the approximation for the integral

$$\int_{x_0}^{x_N} f(x)dx = h\left[\frac{1}{3}f_0 + \frac{4}{3}f_1 + \frac{2}{3}f_2 + \frac{4}{3}f_3 + \ldots + \frac{2}{3}f_{N-2} + \frac{4}{3}f_{N-1} + \frac{1}{3}f_N\right]$$

- This formula can not be directly used in iterative computations.

- The error term of the trapezoidal rule can be written in the form (Euler-Maclaurin sum formula; note that here we do not truncate the series after the first term as we did previously):

$$\int_{x_0}^{x_N} f(x)dx = h\left[\frac{1}{2}f_0 + f_1 + f_2 + \ldots + f_{N-1} + \frac{1}{2}f_N\right] - \frac{B_2 h^2}{2!}(f'_N - f'_0) - \ldots - \frac{B_{2k}h^{2k}}{(2k)!}(f_N^{(2k-1)} - f_0^{(2k-1)}) - \ldots$$

- Note that we only have even powers of $h$ here. $B_{2k}$ are Bernoulli numbers ($B_2 = 1/6$, $B_4 = -1/30$, ...)

- Let's assume that we have computed the integral using the trapezoidal rule and with $N$ subintervals: $S_N$
- Similarly let $S_{2N}$ be the calculated with $2N$ subintervals.

- Now if we calculate the weighted average of these two approximations we get

$$S = \frac{4}{3}S_{2N} - \frac{1}{3}S_N$$



## Integration: higher order methods

- Expanding this we get

$$S = \frac{4}{3}\left[h\{\ldots\} - \frac{h^2(f'_N - f'_0)}{4 \times 12} + O(h^4)\right] - \frac{1}{3}\left[\frac{h}{2}\{\ldots\} - \frac{h^2(f'_N - f'_0)}{12} - O(h^4)\right]$$

- It is easy to see that the error terms with $h^2$ cancel and the lowest order error will be $O(h^4)$.

- In fact, expanding the above expression we get the Simpson rule.

- Does this sound familiar? It should. What we have is simply **Richardson exrapolation** (RE).

- We can go further to higher orders by using RE: Romberg algorithm

## Integration: Romberg algorithm

- Richardson extrapolation in short:

1) $\phi(h) = L - \sum_{k=1}^{\infty} a_{2k} h^{2k}$

2) Goal is to find the limit $L = \lim_{h \to 0} \phi(h)$

3) Choose a value for $h$ and calculate the numbers $D(n, 1) = \phi\left(\frac{h}{2^n}\right), \quad n = 1, 2, \ldots$

4) Now $D(n, 1) = L + \sum A(k, 1)\left(\frac{h}{2^n}\right)^{2k}$

5) We can get more accurate estimates by **RE** using the recursion relation:

$$D(n, m+1) = \frac{4^m}{4^m - 1} D(n, m) - \frac{1}{4^m - 1} D(n-1, m)$$

6) It can be shown that the quantities $D(n, m)$ have the form

$$D(n, m) = L + \sum_{k=m}^{\infty} A(k, m)\left(\frac{h}{2^n}\right)^{2k}$$

$$\begin{array}{lllll} D(1,1) & & & & \\ D(2,1) & D(2,2) & & & \\ D(3,1) & D(3,2) & D(3,3) & & \\ \ldots & \ldots & \ldots & & \\ D(N,1) & D(N,2) & D(N,3) & \ldots & D(N,N) \end{array}$$



## Integration: Romberg algorithm

- We have already seen that the trapezoidal formula for the integral has the form similar to $\phi(h)$
  → we can use RE to improve the estimate of the trapezoidal rule.

- It goes as follows

1) First compute integrals using the trapezoidal rule for different step sizes $h = \frac{(b-a)}{2^n}$.

Denote these as $R(n, 0)$.

$$R(n, 0) = \frac{1}{2} R(n-1, 0) + h \sum_{i=1}^{2^{n-1}} f(a + (2i-1)h)$$

$$R(0, 0) = \frac{1}{2}(b-a)[f(a) + f(b)]$$

2) Refine the trapezoidal rule results using RE:

$$R(n+1, m+1) = R(n+1, m) + \frac{1}{4^{m+1} - 1}[R(n+1, m) - R(n, m)]$$

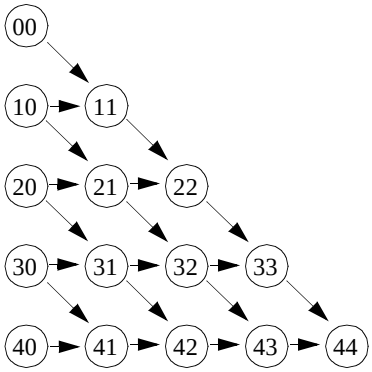
## Integration: Romberg algorithm

- As a result we get the familiar triangle

$$
\begin{array}{lllll}
R(0,0) & & & & \\
R(1,0) & R(1,1) & & & \\
R(2,0) & R(2,1) & R(2,2) & & \\
\dots & \dots & \dots & & \\
R(n,0) & R(n,1) & R(n,2) & \dots & R(n,n)
\end{array}
$$

- The first column is computed using the trapezoidal rule and others are higher order refinements.

- The recursion formula can be expressed graphically as






## Integration: Romberg algorithm

- This is quite simple to code:

```
void Romberg(double (*f)(double x), double a, double b, int n, double R[][MAXN])
     {
          int i, j, k, kmax=1;
          double h, sum;

          h = b-a; R[0][0] = (h/2.0)*(f(a)+f(b));

          for (i=1; i<=n; i++) {

               h = h/2.0; sum = 0;
               kmax = kmax*2;

               /* Trapezoidal estimate R(i,0) */
               for (k=1; k<=kmax-1; k+=2)
                    sum += f(a+k*h);
               R[i][0] = 0.5*R[i-1][0]+sum*h;

               /* Successive R(i,j) */
               for(j=1; j<=i; j++)
                    R[i][j] = R[i][j-1] +(R[i][j-1]-R[i-1][j-1])/(pow(4.0,(double)j)-1.0);
               }
     }
```

- Here the elements of the array $R(i,j)$ are computed row by row up to the specified number of rows, $n$ (this number need not to be very large because the method converges very quickly, e.g., $n = 4,5$ is often suitable).

## Integration: Romberg algorithm

- Example:

Using same example as before, we calculate the value of π by evaluating the integral

$$\pi \approx \int_0^1 \frac{4}{1+x^2} dx$$

(Correct value is 3.141592654.)

The output using double-precision:

```
3.0000000000
3.1000000000 3.1333333333
3.1311764706 3.1415686275 3.1421176471
3.1389884945 3.1415925025 3.1415940941 3.1415857838
3.1409416120 3.1415926512 3.1415926611 3.1415926384 3.1415926653
3.1414298932 3.1415926536 3.1415926537 3.1415926536 3.1415926536 3.1415926536
```

And using single-precision:

```
3.0000000000
3.0999999046 3.1333332062
3.1311764717 3.1415686607 3.1421177387
3.1389884949 3.1415925026 3.1415941715 3.1415858269
3.1409416199 3.1415927410 3.1415927410 3.1415927410 3.1415927410
3.1414299011 3.1415927410 3.1415927410 3.1415927410 3.1415927410 3.1415927410
```

We notice that the algorithm converges very quickly (with double precision, $n = 5$ is enough to obtain nine decimals of precision). With single precision, the accuracy of the calculation is limited to six decimals.



## Integration: adaptive algorithms

• It is not always the best strategy to select the subintervals to be equidistant.
- We need an adaptive algorithm to select automatically the partitioning of the interval.

- Here we use the Simpson's rule **(SR)** presented previously:

$$\int_a^b f(x)dx \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

- It's error can be estimated by the formula[1]

$$E = -\frac{1}{90}\left(\frac{b-a}{2}\right)^5 f^{(4)}(\xi)\ ,\ \xi \in [a,b]$$

- The principle of the adaptive algorithm is the following:

  - First we divide the interval $[a, b]$ into two subintervals.
  - Then we decide whether each of the subintervals is further divided.
  - The process continues until some specific accuracy is achieved.

- We have to develop a test for deciding whether to do the division
  - SR for the integral can be written as

$$I = S(a,b) + E(a,b)$$

where $$S(a,b) = \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

1. Atkinson, Eq. (5.1.15)

## Integration: adaptive algorithms

- Letting $h = b - a$ we have

$$I = S^{(1)} + E^{(1)}, \text{ where } S^{(1)} = S(a,b) \text{ and } E^{(1)} = -\frac{1}{90}\left(\frac{h}{2}\right)^5 f^{(4)}$$

- Two applications of the SR give $[c = (a+b)/2]$

$$I = S^{(2)} + E^{(2)} \text{ where } S^{(2)} = S(a,c) + S(c,b) \text{ and } E^{(2)} = -\frac{1}{90}\left(\frac{h/2}{2}\right)^5 f^{(4)} - \frac{1}{90}\left(\frac{h/2}{2}\right)^5 f^{(4)} = \frac{E^{(1)}}{16}$$

- Now we subtract the first evaluation from the second:

$$\begin{aligned} I - I = 0 &= S^{(2)} + E^{(2)} - S^{(1)} - E^{(1)} \\ \rightarrow S^{(2)} - S^{(1)} &= E^{(2)} - E^{(1)} = 15E^{(2)} \\ \rightarrow E^{(2)} &= \frac{S^{(2)} - S^{(1)}}{15} \end{aligned}$$

- So the second application of SR can be written as

$$I = S^{(2)} + E^{(2)} = S^{(2)} + \frac{S^{(2)} - S^{(1)}}{15}$$

Well, it is a bit misleading to write the integral in this way. To be precise the errors $E$ are all *maximum* errors.

- This can be used to evaluate the accuracy of our approximation of $I$



## Integration: adaptive algorithms

- We can use the following test to decide whether to continue the splitting process:

$$\frac{1}{15}\left|S^{(2)} - S^{(1)}\right| < \varepsilon$$

- If the test is not satisfied the interval $[a,b]$ is split into two subintervals $[a,c]$ and $[c,b]$.

- Performing the test for these subintervals we have to use the tolerance $\varepsilon/2$.

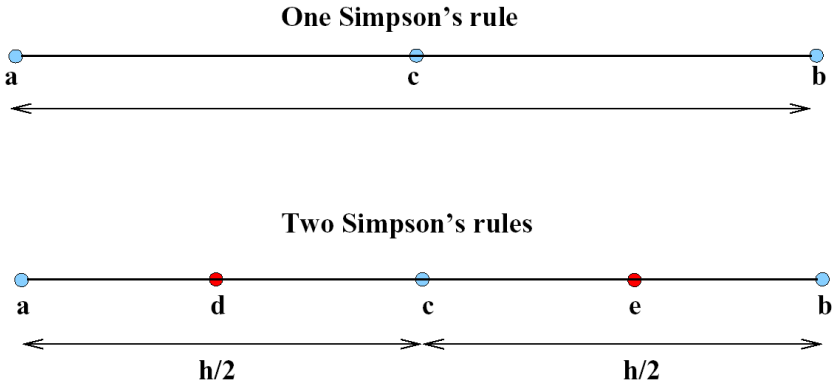
## Integration: adaptive algorithms

- Adaptive algorithm can be programmed as a recursive procedure.

  - Two variables are used to calculate the approximation using SR:

    one_simpson: $S^{(1)} = S(a,b) = \frac{h}{6}[f(a) + 4f(c) + f(b)]$

    two_simpson: $S^{(2)} = S(a,c) + S(c,b) = \frac{h}{12}[f(a) + 4f(d) + 2f(c) + 4f(e) + f(b)]$




  - After evaluating $S^{(1)}$ and $S^{(2)}$ we use the division test $|S^{(2)} - S^{(1)}|/15 < \varepsilon$ to check whether to go on with subdivisions.



## Integration: adaptive algorithms

  - If the criterion is not fulfilled the procedure calls itself with `left_simpson` corresponding to the left side $[a, c]$ and the `right_simpson` corresponding to $[c, b]$. At recursive calls $\varepsilon \leftarrow \varepsilon/2$.

  - Below is the C implementation:

```
double Simpson(double a, double b, double eps, int level, int level_max)
{
     int i, j, k, kmax=1;
     double c, d, e, h, result;
     double one_simpson, two_simpson;
     double left_simpson, right_simpson;
     h = b-a;
     c = 0.5*(a+b);
     one_simpson = h*(f(a)+4.0*f(c)+f(b))/6.0;
     d = 0.5*(a+c); e = 0.5*(c+b);
     two_simpson = h*(f(a)+4.0*f(d)+2.0*f(c)+4.0*f(e)+f(b))/12.0;
     /* Check for level */
     if (level+1 >= level_max) {
          result = two_simpson;
          printf("Maximum level reached\n");
     } else {
          /* Check for desired accuracy */
          if (fabs(two_simpson-one_simpson) < 15.0*eps)
               result = two_simpson + (two_simpson-one_simpson)/15.0;
          /* Divide further */
          else {
               left_simpson = Simpson(a,c,eps/2.0,level+1,level_max);
               right_simpson = Simpson(c,b,eps/2.0,level+1,level_max);
               result = left_simpson + right_simpson;
          }
     }
     return(result);
}
```

## Integration: adaptive algorithms

**Application example**

As an example, the program is used to calculate the integral

$$\int_0^{2\pi} \left[\frac{cos(2x)}{e^x}\right] dx$$

The desired accuracy is set to $\varepsilon = \frac{1}{2} \times 10^{-4}$. An external function procedure is written for $f$ and its name is given as the first argument to *Simpson*.

Running the program with double precision floating-point numbers, we obtain the result 0.1996271. It is correct within the tolerance we set beforehand.

We can use Matlab or Maple to determine the correct answer. The following Matlab commands

```
syms t
res = int(cos(2*t)./exp(t),0,2*pi)
eval(res)
```

```
(%i1) integrate(cos(2*x)/%e^x,x,0,2*%pi);
(%o1) 1/5 - e^(-2*%pi)/5
(%i2)
```

first return the exact value $\frac{1}{5}(1 - e^{-2\pi})$ and then evaluate the integral giving (with ten digits) 0.19962 65115.



## Integration: adaptive algorithms

- One can explicitely derive higher order integration formulas (so called Newton-Cotes formulas) by increasing the order of the interpolating polynomial.
  - The first four formulas (including trapezoidal and Simpson's rules) are as below:

$n = 1$: $\int_a^b f(x)dx = \frac{h}{2}[f(a) + f(b)] - \frac{h^2}{12}f^{(2)}(\xi)$ trapezoidal

$n = 2$: $\int_a^b f(x)dx = \frac{h}{3}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right] - \frac{h^5}{90}f^{(4)}(\xi)$ Simpson

$n = 3$: $\int_a^b f(x)dx = \frac{3h}{8}[f(a) + 3f(a+h) + 3f(b-h) + f(b)] - \frac{3h^5}{80}f^{(4)}(\xi)$ 3/8

$n = 4$: $\int_a^b f(x)dx = \frac{2h}{45}\left[7f(a) + 32f(a+h) + 12f\left(\frac{a+b}{2}\right) + 32f(b-h) + 7f(b)\right] - \frac{8h^7}{945}f^{(6)}(\xi)$ Milne

  - Here, as usual, $h = \frac{b-a}{n}$, $\xi \in [a, b]$.
  - These are for one subinterval. The composite formulas can be easily derived.
  - For more information, see e.g. Atkinson, section 5.2.

## Integration: Gaussian quadrature

• In the previous algorithms the integrals were computed by evaluating the function at more or less evenly spaced points and by weighting them appropriately.

• However, the more we have free parameters the higher is the order of the method.

• The basic idea of Gaussian quadrature (**GQ**) is to choose the points where the function is evaluated in an optimal way.

- This means that the 'number of degrees of freedom' is doubled.

- In principle the degree of the method is also doubled: With the same number of function evaluations we get more accurate results.

- But remember: The function has to be smooth and nice behaving.

- GQ algorithms are often derived for weighted integrals:

$$\int_a^b W(x)f(x)dx \approx \sum_{j=1}^{N} w_j f(x_j)$$

- Here $W(x)$ is the weight function.
- It is useful for e.g. removing singularities from the integrand.

- The node positions $\{x_i\}$ and weights $\{w_i\}$ are chosen such that the quadrature is exact for polynomial $f(x)$.



## Integration: Gaussian quadrature

- For example in computing the integral

$$\int_{-1}^{1} \frac{\exp(-\cos^2 x)}{\sqrt{1-x^2}} dx$$

it is wise to choose the weight function as

$$W(x) = \frac{1}{\sqrt{1-x^2}}$$

- The weight function need not be written explicitely: by writing $g(x) = W(x)f(x)$ and $v_j = w_j / W(x_j)$ the integral is

$$\int_a^b g(x)dx = \sum_{j=1}^{N} v_j g(x_j)$$

- How do we determine the nodes $\{x_i\}$ and weights $\{w_i\}$?

- A straightforward way is to write down equations that require that the GQ is exact for polynomials upto some degree.

## Integration: Gaussian quadrature

- Let's take a simple example:

  - Integration in the interval $[-1, 1]$ with the weight function $W(x) = 1$.

$$\int_{-1}^{1} f(x)dx \approx \sum_{j=1}^{N} w_j f(x_j)$$

  - We have to determine $\{x_i\}$ and $\{w_i\}$ such that the error

$$E_N(f) = \int_{-1}^{1} f(x)dx - \sum_{j=1}^{N} w_j f(x_j)$$

    is zero for polynomials with as high degree as possible.

  - Note that the nodes and weights also depend on $N$: $x_{j,N}$, $w_{j,N}$. However, for simplicity, we drop the second subscript.

  - We know that

$$E_N(a_0 + a_1 x + \ldots + a_m x^m) = a_0 E_N(1) + a_1 E_N(x) + \ldots + a_m E_N(x^m)$$



## Integration: Gaussian quadrature

- This means that $E_N(x) = 0$ for all polynomials with degree below $m + 1$ only if

$$E_N(x^i) = 0, \quad i = 0, 1, \ldots, m$$

- Let $N = 1$. We now have two parameters $x_1$ and $w_1$, which should satisfy

$$\begin{cases} E_N(1) = 0 \\ E_N(x) = 0 \end{cases}$$

  or

$$\begin{cases} \int_{-1}^{1} 1dx - w_1 = 0 \\ \int_{-1}^{1} xdx - w_1 x_1 = 0 \end{cases} \rightarrow \begin{cases} w_1 = 2 \\ x_1 = 0 \end{cases}$$

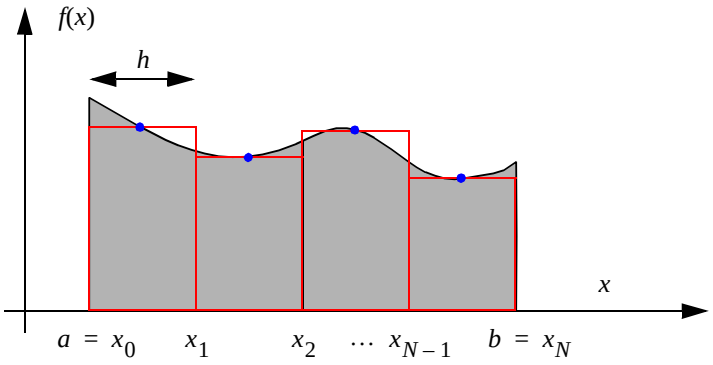
- So, the integral takes the form

$$\int_{-1}^{1} f(x)dx \approx 2f(0)$$

## Integration: Gaussian quadrature

- A sidenote: This is actually so called *midpoint rule* for evaluating the integral:

$$\int_a^b f(x)dx = h\sum_{i=1}^{N} f\left(\frac{x_{i-1}+x_i}{2}\right)$$




- Let $N = 2$. Now we have four parameters: $w_1, w_2, x_1, x_2$

- These are obtained by solving:

$$E_N(x^i) = \int_{-1}^{1} x^i dx - [w_1 x_1^i + w_2 x_2^i] = 0, \quad i = 0, 1, 2, 3$$

or

$$\begin{cases} w_1 + w_2 = 2 \\ w_1 x_1 + w_2 x_2 = 0 \\ w_1 x_1^2 + w_2 x_2^2 = 2/3 \\ w_1 x_1^3 + w_2 x_2^3 = 0 \end{cases}$$



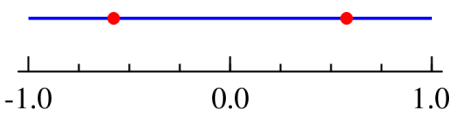
## Integration: Gaussian quadrature

- Solution for this is

$$\begin{cases} w_1 = 1 \\ w_2 = 1 \\ x_1 = -\frac{\sqrt{3}}{3} \\ x_2 = \frac{\sqrt{3}}{3} \end{cases}$$

and the approximation for the integral is

$$\int_{-1}^{1} f(x)dx \approx f(-\frac{\sqrt{3}}{3}) + f(\frac{\sqrt{3}}{3}) \ .$$




- This rule has a degree of three. Using Simpson's rule the same accuracy is obtained by using three points.

## Integration: Gaussian quadrature

- For an arbitrary value of $N$ we have $2N$ parameters and they can be obtained requiring that

$$\int_{-1}^{1} f(x)dx \approx \sum_{j=1}^{N} w_j f(x_j)$$

is accurate for integrals of polynomials with degree $2N-1$ or lower.

- The equations are

$$E_N(x^i) = 0, \quad i = 0, 1, 2, \ldots, 2N-1$$

or

$$\sum_{j=1}^{N} w_j x_j^i = \begin{cases} 0, i = 1, 3, 5, \ldots, 2N-1 \\ \frac{2}{i+1}, i = 0, 2, 4, \ldots, 2N-2 \end{cases}$$

- However, this is not the way to go. These are nonlinear equations and not so easy to solve.

- Better methods are based on orthogonal polynomials.



## Integration: Gaussian quadrature

- Let the integration interval be $[a, b]$.

- Inner product of two polynomials (or functions in general) with weight $W(x)$ is defined as

$$\langle f|g\rangle = \int_a^b W(x)f(x)g(x)dx.$$

- Two polynomials are orthogonal if

$$\langle f|g\rangle = 0.$$

- A polynomial is normalized if

$$\langle f|f\rangle = 1.$$

- With the following algorithm one can construct a set of mutually orthogonal polynomials with exactly one member of degree $j$, $p_j(x)$, $j = 0, 1, 2, \ldots$ :

1. $p_{-1}(x) \equiv 0$, $p_0(x) \equiv 1$
2. $p_{j+1}(x) = (x - a_j)p_j(x) - b_j p_{j-1}(x)$ , $j = 0, 1, 2, \ldots$ (The last term omitted when $j = 0$!)

   where $a_j = \frac{\langle xp_j|p_j\rangle}{\langle p_j|p_j\rangle}$, $j = 0, 1, 2, \ldots$ and $b_j = \frac{\langle p_j|p_j\rangle}{\langle p_{j-1}|p_{j-1}\rangle}$, $j = 1, 2, 3, \ldots$.

## Integration: Gaussian quadrature

- These polynomials are of the form $p_j(x) = x^j + c_{j-1}x^{j-1} + c_{j-2}x^{j-2} + \ldots$

- They can always be normalized by dividing by $\sqrt{\langle p_j | p_j \rangle}$.

- Every polynomial $p_j(x)$ has exactly $j$ distinct roots in the interval $[a, b]$.
- Roots of $p_j(x)$ interleave the $j-1$ roots of $p_{j-1}(x)$: use $p_{j-1}(x)$ to locate roots of $p_j(x)$.

- The connection between orthogonal polynomials and GQ is the following theorem:

- For each $N$ there is a unique numerical integration formula

$$\int_a^b W(x)f(x)dx \approx \sum_{j=1}^{N} w_j f(x_j)$$

which is exact for polynomials of degree $\leq 2N-1$.
The nodes are the roots of the polynomial $p_N(x)$
which is a member of the set of orthogonal polynomials
defined in the interval $[a, b]$ with respect to weight $W(x)$.
The weights are

$$w_j = \frac{\langle p_{N-1} | p_{N-1} \rangle}{p_{N-1}(x_j)p'_N(x_j)}.$$

For proof see e.g. Atkinson chapter 5.3.



## Integration: Gaussian quadrature

- Note that in principle the weights could be obtained from equations

$$\begin{cases} w_1 p_0(x_1) + w_2 p_0(x_2) + \ldots + w_N p_0(x_N) = \int_a^b W(x)p_0(x)dx \\ w_1 p_1(x_1) + w_2 p_1(x_2) + \ldots + w_N p_1(x_N) = \int_a^b W(x)p_1(x)dx \\ \ldots \\ w_1 p_{N-1}(x_1) + w_2 p_{N-1}(x_2) + \ldots + w_N p_{N-1}(x_N) = \int_a^b W(x)p_{N-1}(x)dx \end{cases}$$

- Note that $\int_a^b W(x)p_j(x)dx = 0$ when $j > 0$ because $p_0(x)$ is a constant and polynomials are orthogonal.

- One can show that with these nodes and weights also polynomials $p_k(x)$, $k = N, N+1, \ldots, 2N-1$ are integrated exactly.

- Preparing a GQ scheme has three stages

1. Generate the orthogonal polynomials by the recursion formula.
2. Determine the roots of the polynomials.
3. Compute the weights.

- List of nodes and weights for the abovementioned 'classical' cases can be found in the literature:
  - See for example *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, edited by: Abramowitz, M.; Stegun, I.A
  (Online version of the book at http://www.knovel.com/knovel2/Toc.jsp?BookID=528 doesn't seem to include the tables!)

## Integration: Gaussian quadrature

- In the literature there a quite a few ‘classical’ GQ schemes for different weight functions:

Gauss-Legendre: $\begin{cases} W(x) = 1 \\ -1 < x < 1 \\ (j+1)P_{j+1} = (2j+1)xP_j - jP_{j-1} \end{cases}$

Gauss-Chebyshev: $\begin{cases} W(x) = \dfrac{1}{\sqrt{1-x^2}} \\ -1 < x < 1 \\ T_{j+1} = 2xT_j - T_{j-1} \end{cases}$

Gauss-Laguerre: $\begin{cases} W(x) = x^{\alpha}e^{-x} \\ 0 < x < \infty \\ (j+1)L_{j+1}^{\alpha} = (-x+2j+\alpha+1)L_j^{\alpha} - (j+\alpha)L_{j-1}^{\alpha} \end{cases}$

Gauss-Hermite: $\begin{cases} W(x) = e^{-x^2} \\ -\infty < x < \infty \\ H_{j+1} = 2xH_j - 2jH_{j-1} \end{cases}$

Gauss-Jacobi: $\begin{cases} W(x) = (1-x)^{\alpha}(1+x)^{\beta} \\ -1 < x < 1 \\ c_j P_{j+1}^{(\alpha,\beta)} = (d_j + e_j x)P_j^{(\alpha,\beta)} - f_j P_{j-1}^{(\alpha,\beta)} \end{cases}$ , where $\begin{cases} c_j = 2(j+1)(j+\alpha+\beta+1)(2j+\alpha+\beta) \\ d_j = (2j+\alpha+\beta+1)(\alpha^2-\beta^2) \\ e_j = (2j+\alpha+\beta)(2j+\alpha+\beta+1)(2j+\alpha+\beta+2) \\ f_j = 2(j+\alpha)(j+\beta)(2j+\alpha+\beta+2) \end{cases}$



## Integration: Gaussian quadrature

- For the Chebyshev GQ the nodes and weights are readily calculated as

$$\begin{cases} x_j = \cos\left[\dfrac{\pi\left(j-\frac{1}{2}\right)}{N}\right] \\ w_j = \dfrac{\pi}{N} \end{cases}$$

- When the nodes and weights are known the integral is computed simply as

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

- Limits of integration can easily be changed if desired: e.g. $[-1, 1] \to [a, b]$

## Integration: Gaussian quadrature

- Error estimation of Gaussian quadrature is more complicated than in the case of e.g. trapezoidal or Simpson's rule.
  - For derivation of a general error expression see e.g. Atkinson, chapter 5.3.
  - A common practical method to estimate error is the comparison of consecutive iterations (i.e. results with increasing number of nodes):

$$I - I_n \approx I_m - I_n ,$$

where $m > n$.

  - The fact that in calculating $I_m$ we can not utilize $I_n$ brings some inefficiency here.
  - However, there are Gaussian quadrature rules that a designed such that they use the previous nodes.
  - One example of this is the **Gauss-Kronrod** quadrature rule.



## Integration: Gaussian quadrature

- In Gauss-Kronrod quadrature we start from Gaussian quadrature $G_N$

$$G_N = \sum_{i=1}^{N} w_i f(x_i) .$$

- To get a more accurate estimate we calculate

$$K_{2N+1} = \sum_{i=1}^{N} a_i f(x_i) + \sum_{j=1}^{N+1} b_j f(y_j) ,$$

where the nodes of $G_N$ are reused but all weigths $a_i$ and $b_i$ are new.
  - In addition there are $N+1$ new nodes $y_j$.
  - Kronrod showed that $K_{2N+1}$ is accurate for polynomials with degrees up to $3N+1$.

- The error estimate could be calculated as

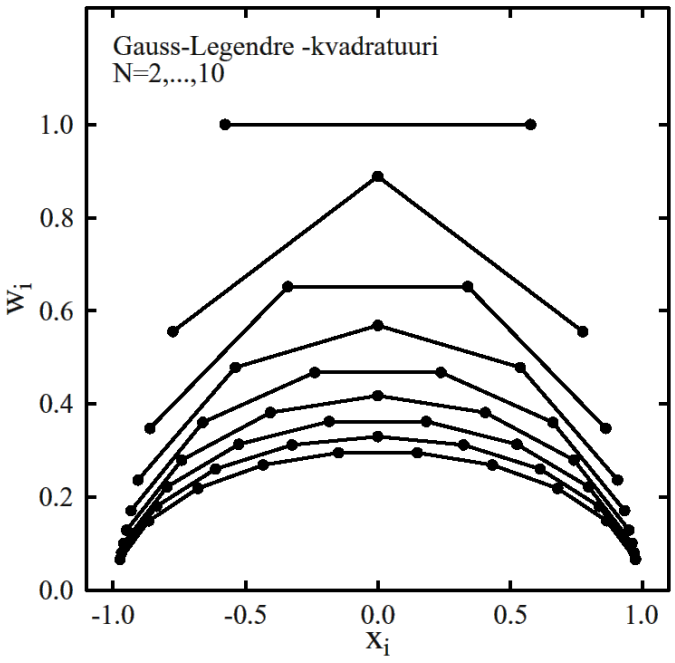
$$\varepsilon \approx \left| G_N - K_{2N+1} \right| .$$

- However, experience says that this is usually too pessimistic; a more realistic is

$$\varepsilon \approx (200 \left| G_N - K_{2N+1} \right|)^{1.5} .$$

# Integration: Gaussian quadrature

- For example the nodes and weights of Gauss-Legendre quadrature:






# Integration: Gaussian quadrature

- Comparison of the methods (source Atkinson: An Introduction to Numerical Analysis)

$$I = \int_0^{\pi} e^x \cos x \, dx$$

**Table 5.1** Trapezoidal rule for evaluating (5.19)

| $n$ | $I_n$ | $E_n$ | Ratio | $\tilde{E}_n$ |
|---|---|---|---|---|
| 2 | −17.389259 | 5.32 | | 4.96 |
| | | | 4.20 | |
| 4 | −13.336023 | 1.27 | | 1.24 |
| | | | 4.06 | |
| 8 | −12.382162 | 3.12E−1 | | 3.10E−1 |
| | | | 4.02 | |
| 16 | −12.148004 | 7.77E−2 | | 7.76E−2 |
| | | | 4.00 | |
| 32 | −12.089742 | 1.94E−2 | | 1.94E−2 |
| | | | 4.00 | |
| 64 | −12.075194 | 4.85E−3 | | 4.85E−3 |
| | | | 4.00 | |
| 128 | −12.071558 | 1.21E−3 | | 1.21E−3 |
| | | | 4.00 | |
| 256 | −12.070649 | 3.03E−4 | | 3.03E−4 |
| | | | 4.00 | |
| 512 | −12.070422 | 7.57E−5 | | 7.57E−5 |

**Table 5.2** Simpson rule for evaluating (5.19)

| $n$ | $I_n$ | $E_n$ | Ratio | $\tilde{E}_n$ |
|---|---|---|---|---|
| 2 | −11.5928395534 | −4.78E−1 | | −1.63 |
| | | | 5.59 | |
| 4 | −11.9849440198 | −8.54E−2 | | −1.02E−1 |
| | | | 14.9 | |
| 8 | −12.0642089572 | −6.14E−3 | | −6.38E−3 |
| | | | 15.5 | |
| 16 | −12.0699513233 | −3.95E−4 | | −3.99E−4 |
| | | | 15.9 | |
| 32 | −12.0703214561 | −2.49E−5 | | −2.49E−5 |
| | | | 16.0 | |
| 64 | −12.0703447599 | −1.56E−6 | | −1.56E−6 |
| | | | 16.0 | |
| 128 | −12.0703462191 | −9.73E−8 | | −9.73E−8 |
| | | | 16.0 | |
| 256 | −12.0703463103 | −6.08E−9 | | −6.08E−9 |

**Table 5.16** Example of Romberg integration for (5.19)

| $k$ | Nodes | $J_k(f)$ | Error |
|---|---|---|---|
| 1 | 3 | −11.59283955342 | −4.8E−1 |
| 2 | 5 | −12.01108431754 | −5.9E−2 |
| 3 | 9 | −12.07042041287 | 7.4E−5 |
| 4 | 17 | −12.07034720873 | 8.9E−7 |
| 5 | 33 | −12.07034631632 | −7.0E−11 |
| 6 | 65 | −12.07034631639 | <5.0E−12 |

**Table 5.9** Gaussian quadrature for (5.19)

| $n$ | $I_n$ | $I-I_n$ |
|---|---|---|
| 2 | −12.33621046570 | 2.66E−1 |
| 3 | −12.12742045017 | 5.71E−2 |
| 4 | −12.07018949029 | −1.57E−4 |
| 5 | −12.07032853589 | −1.78E−5 |
| 6 | −12.07034633110 | 1.47E−8 |
| 7 | −12.07034631753 | 1.14E−9 |
| 8 | −12.07034631639 | <5.0E−12 |

# Integration: improper integrals

• Improper integrals have one of the following properties:

1) Upper limit is $\infty$ or lower limit is $-\infty$.
2) Integrand goes to finite values at the endpoints but can not be evaluated. E.g. $\sin x/x$ at $x = 0$.
3) There is an integrable singularity at either endpoint. E.g. $x^{-1/2}$ at $x = 0$.
4) There is an integrable singularity at a known point in the interval.

- Note that some integrals are simply impossible to evalutate: e.g. $\int_1^\infty \frac{dx}{x}$, $\int_{-\infty}^{\infty} \cos x dx$.

- In case (1) changing the integration may variable help.

- For the interval $[0, \infty]$ suitable substitutions are

$$t = \frac{x}{1+x} : [0, \infty] \to [0, 1]$$

$$t = e^{-x}: \quad [0, \infty] \to [1, 0]$$

- For interval $[-\infty, \infty]$ one can try for example

$$t = \frac{e^x - 1}{e^x + 1}: [-\infty, \infty] \to [-1, 1]$$



# Integration: improper integrals

- Gauss-Laguerre and Gauss-Hermite quadratures are defined for infinite intervals:

Gauss-Laguerre: $\begin{cases} W(x) = x^\alpha e^{-x} \\ 0 < x < \infty \end{cases}$

Gauss-Hermite: $\begin{cases} W(x) = e^{-x^2} \\ -\infty < x < \infty \end{cases}$

- For example $I = \int_0^\infty e^{-x} \sin x dx$ can be computed by using Gauss-Laguerre quadrature ($\alpha = 0$):

$$I \approx \sum_{i=1}^{N} w_i \sin x_i$$

- Results for various values of $N$ (exact value $I = \frac{1}{2}$):

```
  2  0.432459454679844
  4  0.504879279460199
  8  0.499987753735300
 16  0.499999999985333
 32  0.500000000000000
 64  0.500000000000000
128  0.500000000000000
256  0.500000000000000
```

```
(%i9) f:exp(-x)*sin(x);

(%o9) e^-x sin x
(%i10) integrate(f,x);

(%o10) e^-x (- sin x - cos x) / 2
(%i11) factor(%);

(%o11) - e^-x (sin x + cos x) / 2
```

## Integration: improper integrals

- Case (2) can often be computed by open methods (i.e. methods that do not evaluate the function at the end points).
  - Note that GQs are examples of these kind of rules.

- Case (3) can often be computed by changing the variable.
  - A simple case: $I = \int_0^1 \frac{dx}{\sqrt{x}}$. Its exact value is $I = 2$.
  - Applying Gauss-Laguerre quadrature (with the proper interval scaling $[0, 1] \to [-1, 1]$ enables us to compute $I$:

```
  2  1.65068012388578
  4  1.80634254040352
  8  1.89754094923051
 16  1.94722751142287
 32  1.97320909141769
 64  1.98650087152911
128  1.99322419413980
256  1.99660549565866
512  1.99830109231141
```

  - However, convergence is slow even though GLQ removed the difficulty of eveluating the function in origin.
  - In general for inverse-square-root singularities in the integral $\int_a^b f(x)dx$ the following substitution may help:

$$x - a = t^2$$



## Integration: improper integrals

  - This gives the integral in the form

$$\int_a^b f(x)dx = \int_0^{\sqrt{b-a}} 2tf(a + t^2)dt$$

  - Now our example integral reduces to ($a = 0$, $b = 1$, $f(x) = x^{-1/2}$)

$$\int_0^1 \frac{dx}{\sqrt{x}} = \int_0^1 2tf(t^2)dt = 2\int_0^1 dt = 2.$$

- Case (4) can be integrated in two parts so that we end up with singularities at the endpoints.

## Integration: summary of 1D integrals

• If your function is one of those nice-behaving ones Gaussian quadrature is probably the best choice.
  - You need to know the nodes and weights.
  - They can be found in the literature (in form of tables and subroutines) for 'classical' quadrature weights.
  - Also, when going to higher number of nodes the previous results can not be utlilized.

• Romberg's method is almost as good as GQ.
  - For nice functions if you don't know suitable GQ rules.

• If your function is not nice-behaving trapezoidal or Simpson's rule are robust though slow.
  - Adaptive algorithms may save some CPU time.



## Integration: multidimensional integrals

• Multidimensional integrals are problematic for two reasons:

1) The number of function evaluations increases quickly when the desired accuracy becomes better.

2) Integration volume may have a complicated shape.

• Certain integrals can be reduced to 1D.
• For example so called iterated integrals can be expressed as 1D integrals:

$$\int_0^x dt_n \int_0^{t_n} dt_{n-1} \cdots \int_0^{t_3} dt_2 \int_0^{t_2} f(t_1) dt_1 = \frac{1}{(n-1)!} \int_0^x (x-t)^{n-1} f(t) dt$$

• If the integration volume is complicated but the function smooth one could use the Monte Carlo method.

• If the volume is simple and the function smooth we can break up the integral into repeated 1D integrals:

$$I = \iiint dx dy dx f(x, y, z)$$

$$= \int_{x_1}^{x_2} dx \int_{y_1(x)}^{y_2(x)} dy \int_{z_1(x,y)}^{z_2(x,y)} dz f(x, y, z)$$

## Integration: multidimensional integrals

- E.g. integration of a function over unit circle:

$$\int_{-1}^{1} dx \int_{-\sqrt{1-x^2}}^{\sqrt{1-x^2}} dy f(x,y)$$

- Let's call the innermost integral in the above equation as $G(x,y)$

$$G(x,y) = \int_{z_1(x,y)}^{z_2(x,y)} dz f(x,y,z)$$

and its integral as $H(x)$

$$H(x) = \int_{y_1(x)}^{y_2(x)} G(x,y)dy$$

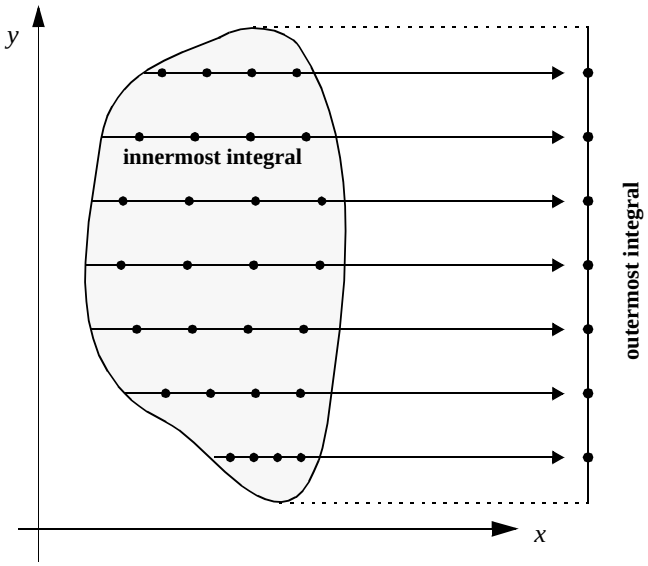
- The final results is the integral of $H$ over $[x_1, x_2]$

$$I = \int_{x_1}^{x_2} H(x)dx$$



## Integration: multidimensional integrals

- Graphically:



## Integration: multidimensional integrals

- In C we could say it as below

```
static double xsav,ysav;
static double (*nrfunc)(double,double,double);

double quad3d(double (*func)(double, double, double),
double x1, double x2)
{
     double qgaus(double (*func)(double), double a,
             double b);
     double f1(double x);

     nrfunc=func;
     return qgaus(f1,x1,x2);
}

double f1(double x)
{
     double qgaus(double (*func)(double), double a,
double b);
     double f2(double y);
     double yy1(double),yy2(double);

     xsav=x;
     return qgaus(f2,yy1(x),yy2(x));
}

double f2(double y)
{
     double qgaus(double (*func)(double), double a, double b);
     double f3(double z);
     double z1(double,double),z2(double,double);
     ysav=y;
     return qgaus(f3,z1(xsav,y),z2(xsav,y));
}
```

```
double qgaus(double (*func)(double), double a, double
b)
{
  int j;
  double xr,xm,dx,s;
  static double x[]={0.0,0.1488743389,0.4333953941,
       0.6794095682,0.8650633666,0.9739065285};
  static double w[]={0.0,0.2955242247,0.2692667193,
       0.2190863625,0.1494513491,0.0666713443};
  xm=0.5*(b+a);
  xr=0.5*(b-a);
  s=0;
  for (j=1;j<=5;j++) {
       dx=xr*x[j];
       s += w[j]*((*func)(xm+dx)+(*func)(xm-dx));
  }
  return s *= xr;
}
```

```
double f3(double z)
{
  return (*nrfunc)(xsav,ysav,z);
}
```



## Integration: multidimensional integrals

- And below the main program using this routine:

```
/* Driver for routine quad3d */
#include <stdio.h>
#include <math.h>
#define PI 3.1415927
#define NVAL 10
static double xmax;
double quad3d(double (*func)(double,
double, double), double x1, double x2);
double func(double x,double y,double z)
{
     return x*x+y*y+z*z;
}
double z1(double x,double y)
{
     return (double)
         -sqrt(xmax*xmax-x*x-y*y);
}
double z2(double x,double y)
{
    return (double) sqrt(xmax*xmax-x*x-y*y);
}
double yy1(double x)
{
     return (double) -sqrt(xmax*xmax-x*x);
}
double yy2(double x)
{
     return (double) sqrt(xmax*xmax-x*x);
}
```

```
int main(void)
{
 int i;
 double xmin,s;

 printf("Integral of r^2 over a spherical
volume\n\n");
 printf("%13s %10s
%11s\n","radius","QUAD3D","Actual");
 for (i=1;i<=NVAL;i++) {
     xmax=0.1*i;
     xmin = -xmax;
     s=quad3d(func,xmin,xmax);
     printf("%12.2f %12.6f %11.6f\n",
     xmax,s,4.0*PI*pow(xmax,5.0)/5.0);
 }
 return 0;
}
```

## Integration: multidimensional integrals

- This program integrates $f(x, y, z) = r^2$ ($r = \sqrt{x^2 + y^2 + z^2}$) over a sphere with radius $R$.
  - The exact value is $\frac{4}{5}\pi R^5$.

  - And the results:

```
3dint> a.out
Integral of r^2 over a spherical volume

       radius     QUAD3D      Actual
        0.10     0.000025    0.000025
        0.20     0.000805    0.000804
        0.30     0.006112    0.006107
        0.40     0.025756    0.025736
        0.50     0.078601    0.078540
        0.60     0.195583    0.195432
        0.70     0.422733    0.422406
        0.80     0.824187    0.823550
        0.90     1.485211    1.484063
        1.00     2.515219    2.513274
```

  - Note the large error though we use a 10-point GQ in 1D integration.



## Integration: multidimensional integrals

• Another way to compute multidimensional integrals is so called **product rule**

- Let's assume that we have a 2D integral

$$I = \int\int_D f(x, y)dA$$

- As in the 1D case we could try to find an approximation of the form

$$I = \sum_{i=1}^{N} w_i f(x_i, y_i) + R_N$$

which integrates exactly 2D polynomials with degree no more than $d$.
  - $R_N$ is the error.

- A 2D polynomial with degree $d$ has terms $x^p y^q$ and $p + q \leq d$.
- For example when $d = 2$ the polynomial is

$$a_{00} + a_{10}x + a_{01}y + a_{20}x^2 + a_{11}xy + a_{02}y^2$$

- One (though not necessarily the optimal) way is so called product rule.

## Integration: multidimensional integrals

- Let's write

$$\int_{\alpha}^{\beta} f(x)dx = \sum_{i=1}^{N} w_i f(x_i) + R_1 \qquad \textbf{(1)}$$

$$\int_{a}^{b} g(y)dy = \sum_{i=1}^{M} p_i g(y_i) + R_2 \qquad \textbf{(2)}$$

- This 2D integral can be written in the form

$$\int_{a}^{b}\int_{\alpha}^{\beta} f(x,y)dxdy = \int_{a}^{b}\left[\sum_{i=1}^{N} w_i f(x_i,y) + R_1\right]dy$$

$$= \sum_{i=1}^{N}\left[w_i \int_{a}^{b} f(x_i,y)dy\right] + \int_{a}^{b} R_1 dy$$

$$= \sum_{i=1}^{N} w_i\left[\sum_{j=1}^{M} p_j f(x_i,y_j) + R_2\right] + \int_{a}^{b} R_1 dy$$

$$= \sum_{i=1}^{N}\sum_{j=1}^{M} w_i p_j f(x_i,y_j) + \sum_{i=1}^{N} w_i R_2 + \int_{a}^{b} R_1 dy$$

$$= \sum_{i=1}^{N}\sum_{j=1}^{M} w_i p_j f(x_i,y_j) + R$$

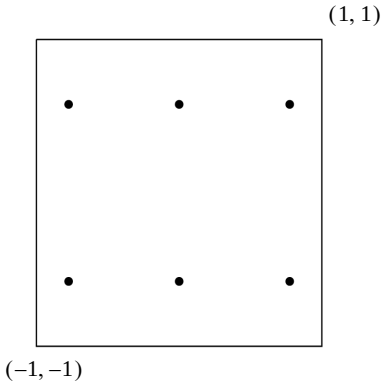


## Integration: multidimensional integrals

- This the **product rule**.

- We use $NM$ points to compute the integral.

- Below is a simple example of a $3 \times 2$ product rule:




- One can show that if equations **(1)** and **(2)** are exact for polynomials with degrees no more than $d_1$ and $d_2$ then product rule integrates exactly polynomials $x^p y^q$ where $p \le d_1$ and $q \le d_2$.

## Integration: multidimensional integrals

- Let's take an example:

$$f(x, y) = xye^{-x^2y}$$
$$x, y \in [0, 1]$$

- Exact value of the integral

$$\int_0^1\int_0^1 f(x, y)dxdy = \int_0^1\int_0^1 xye^{-x^2y}dxdy = \frac{1}{2e} \approx 0.18393972$$

- The code on the following page computes this integral with number of nodes as an input.

- Note: This code uses NR routines; e.g. **`gauleg`** to compute Gauss-Legendre nodes and weights.



## Integration: multidimensional integrals

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define NRANSI
#include "nr.h"
#include "nrutil.h"

float f(float x, float y)
{
   return x*y*exp(-x*x*y);
   /* return sin(5.0*x*x)*cos(y*x); */
}

int main(int argc, char **argv)
{
   int nx,ny,i,j;
   float x1,x2,y1,y2;
   float s;
   float *xx,*wx,*xy,*wy;

   if (argc!=7 ) {
        fprintf(stderr,
       "Komentorivi: %s nx ny x1 x2 y1 y2\n",argv[0]);
        return (1);
   }
   nx=atoi(*++argv); ny=atoi(*++argv);
   x1=atof(*++argv); x2=atof(*++argv);
   y1=atof(*++argv); y2=atof(*++argv);
   xx=vector(1,nx);
   wx=vector(1,nx);
   xy=vector(1,ny);
   wy=vector(1,ny);

   gauleg(x1,x2,xx,wx,nx);
   gauleg(y1,y2,xy,wy,ny);
   s=0.0;
   for (i=1;i<=nx;i++)
        for (j=1;j<=ny;j++)
             s+=wx[i]*wy[j]*f(xx[i],xy[j]);
   printf("GL-kvadratuuri %20.12g\n",s);
   return (0);
}
#undef NRANSI
```

## Integration: multidimensional integrals

- Results:

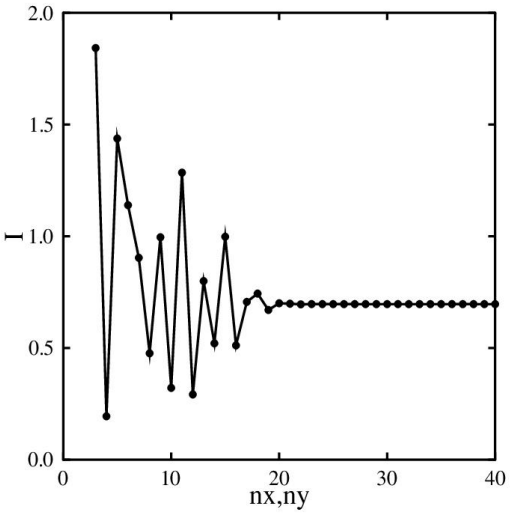
```
2d> xgauleg 3 3 0 1 0 1
GL-kvadratuuri       0.183959037066
Tarkka arvo          0.183939725161
2d> xgauleg 5 5 0 1 0 1
GL-kvadratuuri       0.183939725161
Tarkka arvo          0.183939725161
```

- This is not bad. Though note that the function is smooth and the integration volume regular.
- Not also that the function is evaluated $nx \times ny$ times.

- A more difficult case is

$$I = \int_0^3 \int_0^3 \sin(5x^2)\cos(xy)\,dy\,dx$$

- Because the integral oscillates we need many nodes to get reasonable results:






## Integration: multidimensional integrals

- Sometimes the integration area is not rectangular.

- By suitably changing variables the are can be transformed into rectangular shape:
- E.g. integral over a triangle:

$$I = \iint_\Delta f(x,y)\,dA\,, \quad \Delta = \{(x,y)\colon 0 \le x \le 1,\ 0 \le y \le x\}$$

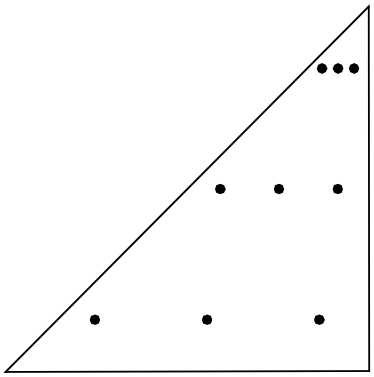
- Let's change variables: $u = x,\ v = \frac{y}{x} \rightarrow du = dx,\ dv = \frac{dy}{x}$

- Now the integral is written as

$$I = \int_0^1 dx \int_0^x dy f(x,y) = \int_0^1 du \int_0^1 dv f(x,y)u$$

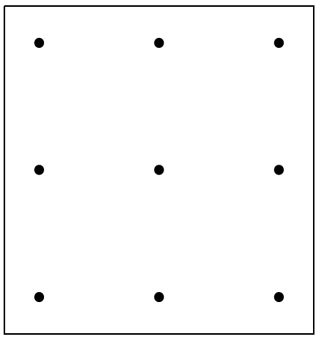
and we can use the product rule.

- However, the transformation changes the sampling density of the integration area:



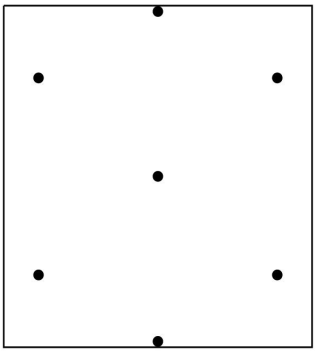
## Integration: multidimensional integrals

- There are methods where the nodes are chosen in a more optimal way than in the product rule.



  - For example the 3x3 Gauss-Legendre product rule has degree of 5 and need 9 function evaluations:

  - **Radon formula** gives more optimal nodes: in the 7 point method



$$\int_{-1}^{1}\int_{-1}^{1} f(x,y)\,dx\,dy \approx \frac{5}{9}[f(r,s)+f(r,-s)+f(-r,s)+f(-r,-s)]$$

$$+\frac{20}{63}[f(0,t)+f(0,-t)]+\frac{8}{7}f(0,0)$$

where $r = \sqrt{3/5}$, $s = \sqrt{1/3}$, $t = \sqrt{14/15}$.

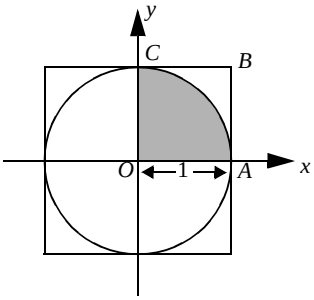
  - This method is of degree 5 and needs only 7 function evaluations.



## Integration: multidimensional integrals

• Monte Carlo (**MC**) method

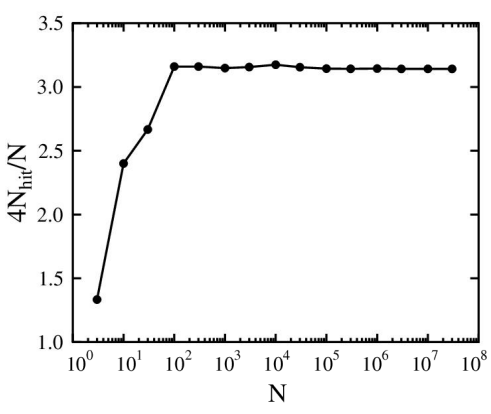
  - Computing the value of $\pi$ by MC:

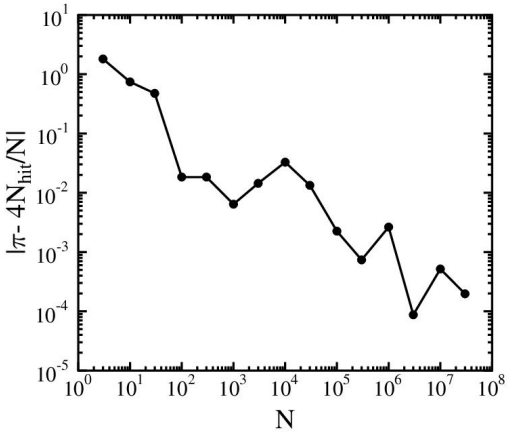



    - Generate $N_{\text{tot}}$ random numbers in the interval $[0, 1)$ and calculate the number of hits to the gray area $N_{\text{hit}}$:

$$\pi \approx \frac{4N_{\text{hit}}}{N_{\text{tot}}}$$

    - The accuracy of the result depends on $N_{\text{tot}}$ and behaves as $O(N_{\text{tot}}^{-1/2})$:



## Integration: multidimensional integrals

- In 1D integration other methods are superior compared to MC.

- But let's take an example from statistical physics:

  - We have to compute the average potential energy of the system

$$\langle U \rangle = \int U e^{-\beta U} d\mathbf{r}$$

    where $\beta = 1/k_B T$ and $U$ is the system potential energy.

  - Our system consists of 100 atoms in a box $x, y, z \in [-L/2, L/2]$.

  - A rough Simpson's method would need $10^{300}$ function evaluations.

  - MC integration using unbiased sampling:

    1) Generate configuration $i$ by randomly placing atoms to the box.
    2) Calculate system potential energy $U(i)$ and Boltzmann factor $e^{-\beta U}$

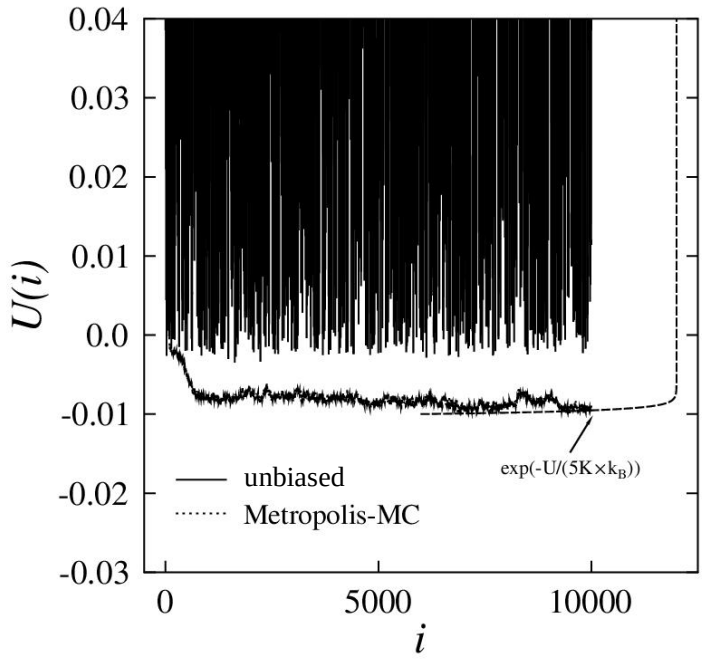
  - The integral becomes now

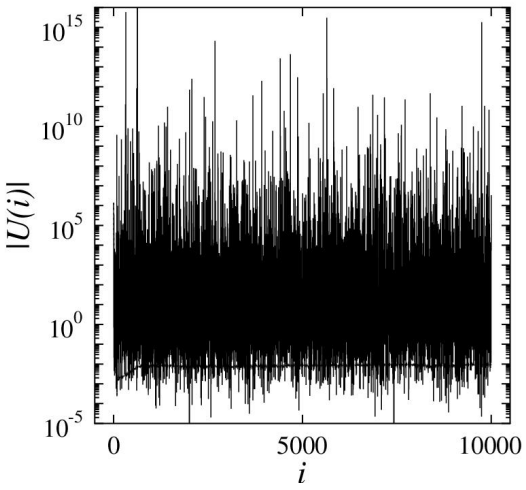
$$\langle U \rangle \propto \sum_{i=1}^{n} U(i) e^{-\beta U(i)}$$



## Integration: multidimensional integrals

- This is very inefficient as can be seen from the figures below:




- Here the Metropolis sampling was started from a random gonfiguration; $T = 5$ K.

- The interaction was Lennard-Jones: $U(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$

## Integration: multidimensional integrals

- So, when generating the points in MC integration it is useful to do that in such a way that we sample the volume in those places where the integrand is not negligible: **importance sampling**.

- Assume we must compute the integral

$$F = \int_{x_1}^{x_2} dx f(x)$$

- Let's write it in the form

$$F = \int_{x_1}^{x_2} dx \left[\frac{f(x)}{\rho(x)}\right] \rho(x)$$

where $\rho(x)$ is an arbitrary probability density (i.e. $\rho(x) \geq 0$, $\int_{x_1}^{x_2} dx \rho(x) = 1$).

- Now we generate a sample of size $N$ by choosing random numbers $\zeta$ that have the distribution $\rho(x)$ in the interval $[x_1, x_2]$.

- In this case the integral can be written as

$$F = \left\langle \frac{f(\zeta)}{\rho(\zeta)} \right\rangle$$



## Integration: multidimensional integrals

- The most simple density would be

$$\rho(x) = \frac{1}{x_2 - x_1}, \; x_1 \leq x \leq x_2$$

when the integral would be

$$F \approx \frac{x_2 - x_1}{N} \sum_{i=1}^{N} f(\zeta_i)$$

- What would be the optimal density function?

- The error of the integral can be written as

$$(\Delta F)^2 = \langle f^2/\rho^2 \rangle - \langle f/\rho \rangle^2$$

- As an integral this is

$$(\Delta F)^2 = \int_{x_1}^{x_2} \frac{f^2}{\rho^2} \rho \, dx - \left[\int_{x_1}^{x_2} \frac{f}{\rho} \rho \, dx\right]^2 = \int_{x_1}^{x_2} \frac{f^2}{\rho^2} \rho \, dx - \left[\int_{x_1}^{x_2} f \, dx\right]^2$$

## Integration: multidimensional integrals

- Now we apply variational calculus to find the optimal $\rho$:

$$\frac{\delta}{\delta\rho}\left\{\int_{x_1}^{x_2}\frac{f^2}{\rho^2}\rho dx-\left[\int_{x_1}^{x_2}f dx\right]^2+\lambda\int_{x_1}^{x_2}\rho dx\right\}=0$$

(Note the additional condition of normalization of $\rho$.)

- By dropping the middle term (it doesn't depend on $\rho$) and moving the variation inside the integral we get

$$-\frac{f^2}{\rho^2}+\lambda=0 \quad\Rightarrow\quad \rho=\frac{|f|}{\int_{x_1}^{x_2}|f|dx}$$

- Really nice: in order to get $\rho$ we should know the integral of the function (well, of its absolute value)!

- There are workarouns to this problem:

1) $\rho$ need not necessarily be exactly the optimum but being near to it may suffice.
2) Generating samples from a distribution can be done without knowing its normalization.

→ **Markov chain Monte Carlo**



## Integration: multidimensional integrals

- In Metropolis MC we generate a series of points (or system states or configurations) in such a way that they obey the desired probability density. (Markov chain)

- The algorithm consists of the probability density of the states $m$: $\rho_m$, and the transition probabilites between states: $\pi_{mn}$.

- The series of states generated by these obey the probability density $\rho_m$ if

(i) $\pi_{mn}\geq 0$

(ii) $\sum_n \pi_{mn}=1$

(iii) $\rho_m\pi_{mn}=\rho_n\pi_{nm}$

- The system also has to be ergodic: it is possible to go from a state to any other state (not in one step necessarily).

- What is essential here is that only the ratio of probability densities is needed → no need to know the normalization.

- Conditions (i)-(iii) are not very strict: They leave a lot for your imagination.

## Integration: multidimensional integrals

- **Metropolis Monte Carlo** is one (and the first[1]) realization of this

$$\pi_{mn} = \begin{cases} \alpha_{mn}; & \rho_n \geq \rho_m \quad m \neq n \\ \alpha_{mn}\left(\dfrac{\rho_n}{\rho_m}\right); & \rho_n < \rho_m, \quad m \neq n \\ 1 - \displaystyle\sum_{m \neq n} \pi_{mn}, & m = n \end{cases}$$

- Here $\alpha_{mn}$ is a symmetric stochastic matrix: $\alpha_{mn} = \alpha_{nm}$. In a way it describes the 'trial' probability of the.

- Based on the symmetry of $\alpha$ one can easily show that the above algorithm obeys the conditions (i),(ii),(iii).

- If you want to know more about the MC method attend the course(s)

**Basics of Monte Carlo simulations**
**Monte Carlo simulations in physics**
http://beam.helsinki.fi/~eholmstr/mc/

1. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller, Equation of State Calculations by Fast Computing Machines, *J. Chem. Phys.*, **21** (1953) 1087.



## Integration: multidimensional integrals

• A final note: A list of top ten algorithms (one of them) of the 20th century can be found at
http://www.siam.org/siamnews/05-00/current.htm

- Below is the list (it seems to be in the cronological order)

1. **Metropolis Monte Carlo**. von Neuman et al., 1946
2. **Simplex linear programming method** (optimization). Dantzig, 1947
3. **Krylov subspace iteration method** (solving $\mathbf{Ax} = \mathbf{b}$ for large matrices). Hestenes et al., 1950
4. **Decompositional approach** to matrix computations. Householder, 1951
5. **Fortran optimizing compiler**. Backus, 1957
6. **QR algorithm** for eigenvalue problems. Francis, 1959-61
7. **Quicksort**. Hoare, 1962
8. **Fast Fourier transform**. Cooley and Tukey, 1965
9. **Integer relation detection algorithm**. Ferguson and Forcade, 1977
10. **Fast multipole algorithm**. Greengard and Rokhlin, 1987

## Integration: practical tools

• Matlab

| | |
|---|---|
| `quad` | adaptive Simpson |
| `quadl` | adaptive Lobatto (GQ on $[-1, 1]$ with $W(x) = 1$ and end points included) |

• Octave

| | |
|---|---|
| `quad` | integration using **Quadpack** routines |

• Quadpack

Package of integration routines written in Fortran 77.
Atkinson: '*The package is well tested and appears to be an excellent collection of programs.*'
Can be downloaded from http://www.netlib.org/quadpack
See also
http://www.scs.fsu.edu/~burkardt/f_src/quadpack/quadpack.html

• SLATEC

Both adaptive and non-adaptive routines
`(d)gaus8`:adaptive 8-point Gauss-Legendre
`(d)qnc79`:adaptive 7-point Newton-Cotes

• GSL

Reimplements the algorithms used in Quadpack
Both adaptive and non-adaptive routines

**Adaptive Gauss**: use previous nodes (but new weights) (Kronrod extension; Gauss-Kronrod quadrature):

$$\int_a^b W(x)f(x)dx = \sum_{i=0}^{N} w_i f(x_i)$$

$$\Downarrow$$

$$\int_a^b W(x)f(x)dx = \sum_{i=0}^{2N+1} v_i f(x_i)$$

Patterson: $N \to N + M$ points.

Note:
$N$-point **Newton-Cotes** =
approximating the function
by a polynomial of degree $N - 1$

Trapezoidal rule = 2-point Newton-Cotes
Simpson = 3-point Newton-Cotes



## Integration: practical tools

• In many applications there are certain special integrals for which efficient methods have been developed.

- Double-folded integrals using Fourier-Bessel expansion.
- Fermi-Dirac integrals.
- Fourier integrals.