Lecture 3

Integration (Quadrature)

Summary Lecture 2

- Interpolation
  - Higher-order: stiffer
  - Schemes reproduce the function values on the grid points
  - Only change interpolation scheme at a grid point!
    - Continuous function value, discontinuous derivative
  - Develop coefficients around the target value! – orders of magnitude different in precision
  - Continuous derivatives: spline
    - Cubic spline: O(N) calculations. Higher-order: O(N^3)
  - Rational : can take out poles!
    - Pade approximant – used a lot in physics
  - NR3 code:
    - Polynomial interpolation, rational interpolation, and spline are in interp_1d.h.
    - Interpolation in 2 dimensions: interp_2d.h
  - nr3.h – Doub, VecDoub, etc

Pade approximation

- Pade.h : needs poly.h, ludcm.h, nr3.h. contains Ratfn pade(VecDoub_I &coef) : call pade with as argument a NR vector (VecDoub coef), containing the coefficients of the power series approximation of the function you want to approximate.
- Return value: a Ratfn. This Ratfn is an object with a constructor Ratfn(VecDoub, VecDoub) that makes the interpolation function. The () operator is overloaded. Calling Ratfn(Doub) returns the Pade approximation.
- Usage: VecDoub coef(N); // make vector with N coefs
- Ratfn rat=pade(coef); // make rat = Ratfn function
- Double x = 0.4; double y=rat(x); // calculate y = pade approximation for x=0.4.

```
Ratfn pade(VecDoub_I &cof)
{
        Int j,k,n=(cof.size()-1)/2;
        Doub sum;
        MatDoub q(n,n),qlu(n,n);
        VecDoub x(n),y(n),num(n+1),denom(n+1);
        for (j=0;j<n;j++) {
                y[j]=cof[n+j+1];
                for (k=0;k<n;k++) q[j][k]=cof[j-k+n];
        }
        LUdcmp lu(q);
        lu.solve(y,x);
        for (j=0;j<4;j++) lu.mprove(y,x);
        for (k=0;k<n;k++) {
                for (sum=cof[k+1],j=0;j<=k;j++) sum -= x[j]*cof[k-j];
                y[k]=sum;
        }
        num[0] = cof[0];
        denom[0] = 1.;
        for (j=0;j<n;j++) {
                num[j+1]=y[j];
                denom[j+1] = -x[j];
        }
        return Ratfn(num,denom);
}
```

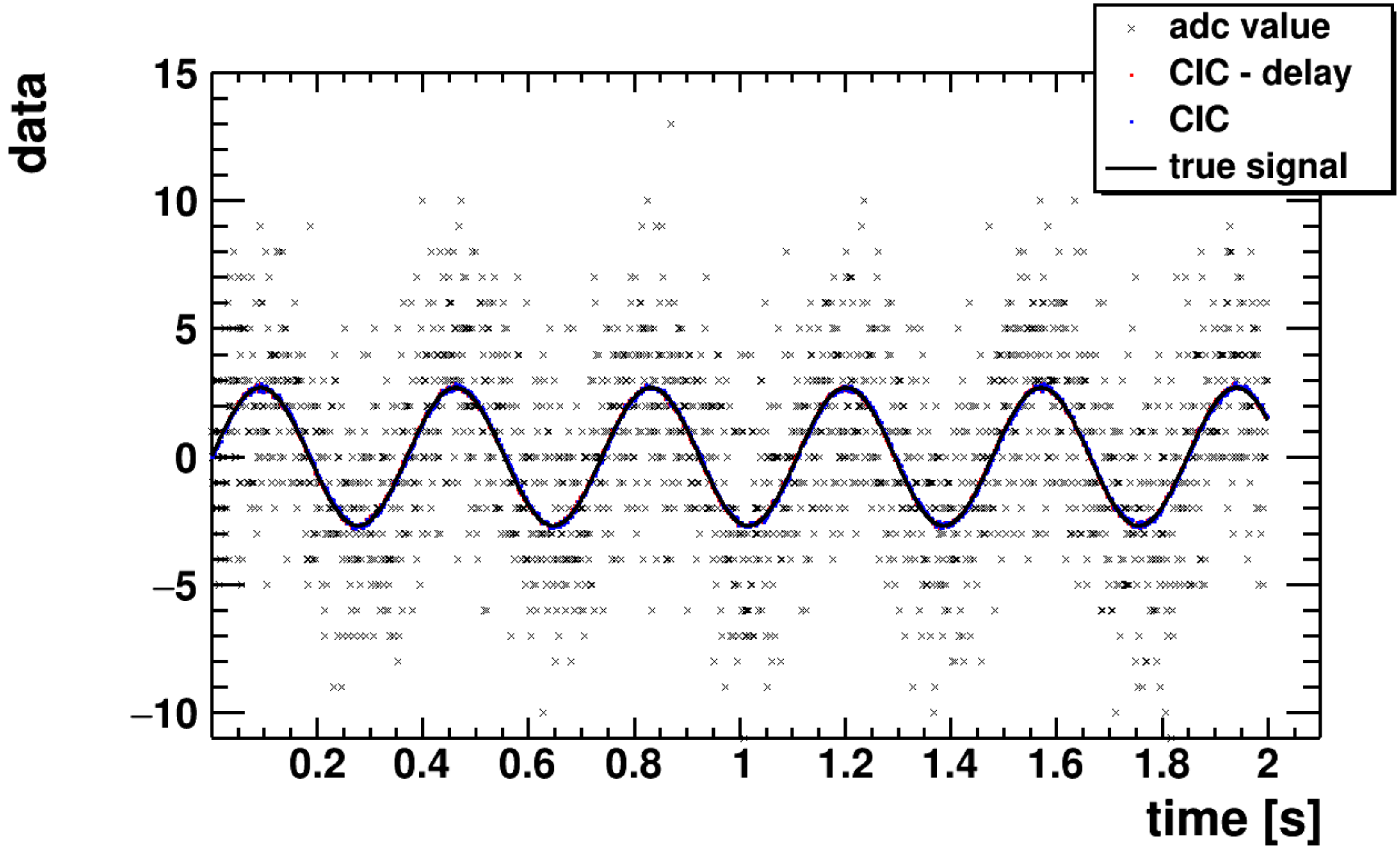
```
struct Ratfn {
        VecDoub cofs;
        Int nn,dd;

        Ratfn(VecDoub_I &num, VecDoub_I &den) : cofs(num.size()
+den.size()-1),
        nn(num.size()), dd(den.size()) {
                Int j;
                for (j=0;j<nn;j++) cofs[j] = num[j]/den[0];
                for (j=1;j<dd;j++) cofs[j+nn-1] = den[j]/den[0];
        }

        Ratfn(VecDoub_I &coffs, const Int n, const Int d) : cofs(coffs),
nn(n),
        dd(d) {}

        Doub operator() (Doub x) const {
                Int j;
                Doub sumn = 0., sumd = 0.;
                for (j=nn-1;j>=0;j--) sumn = sumn*x + cofs[j];
                for (j=nn+dd-2;j>=nn;j--) sumd = sumd*x + cofs[j];
                return sumn/(1.0+x*sumd);
        }

};
```

Exercise 2, CIC filter

- Fast way to calculate powers of running averages
  - In this example, if you would not use a loop it would take 10^12 operations to calculate 1 4-stage average (sums of sums, each sum requiring 1024 additions)
- /home/henkjan/WINDOWS/CompMeth/CompMeth2017/EXERCISES/ex2/cic.cpp
- Output:

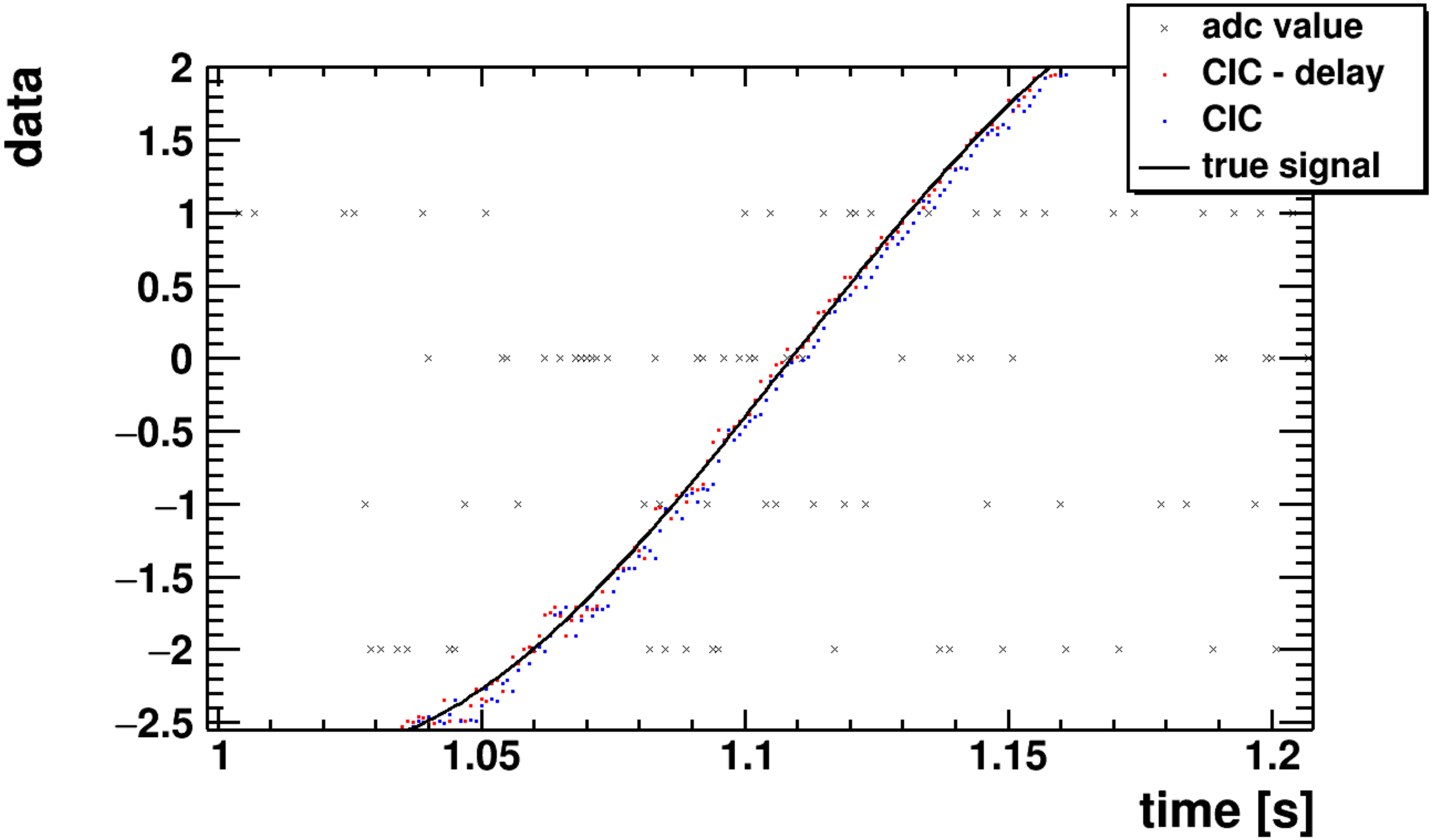
```
RMS of the input minus the signal : 3.06973 RMS of the CIC-filtered result : 0.0654396
```

Exercise 2, CIC filter



Exercise 2, CIC filter

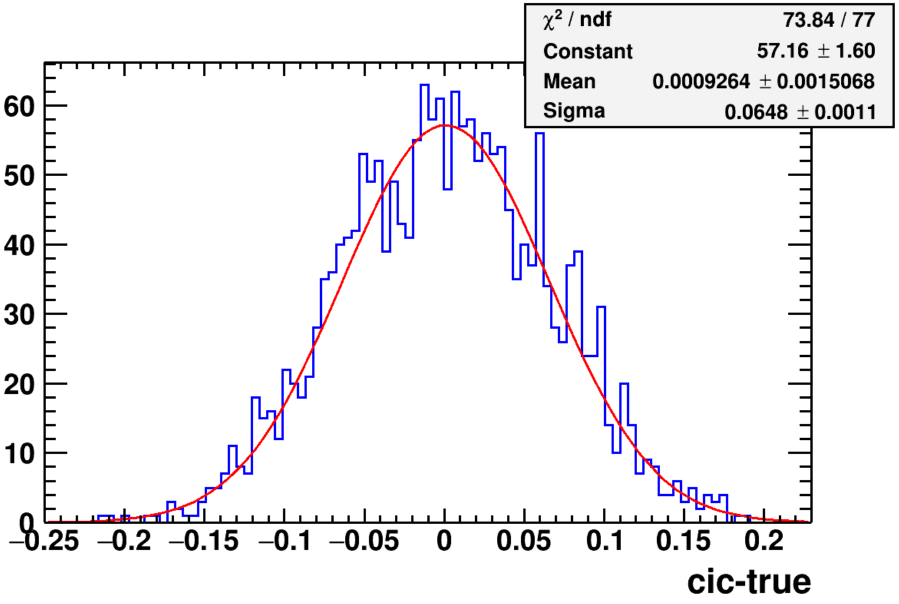
A zoom of the previous picture shows, that the CIC output after N ms has a delay of 2 ms (since it is a weighted average of data in the last 4 ms). This can be easily corrected for.

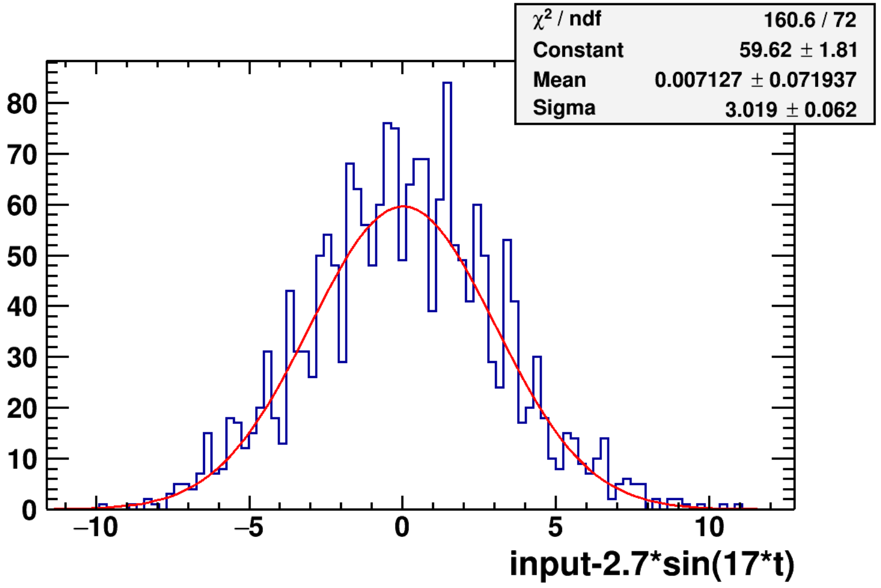


Exercise 2, CIC filter

Residuals of the input minus the true signal.

The CIC filter performs 50 times better than the random input (sampling at 1ms rate). CIC filters are often applied in electronics, signal processing and data analysis.





Integration of functions (Quadrature)

- intuitive expectation: integral is much harder to calculate than differential
- basic task: calculate

$$I = \int_a^b f(x)\,dx \;\Leftrightarrow\; I = y(b) \text{ with } \frac{dy}{dx} = f(x) \text{ and } y(a) = 0$$

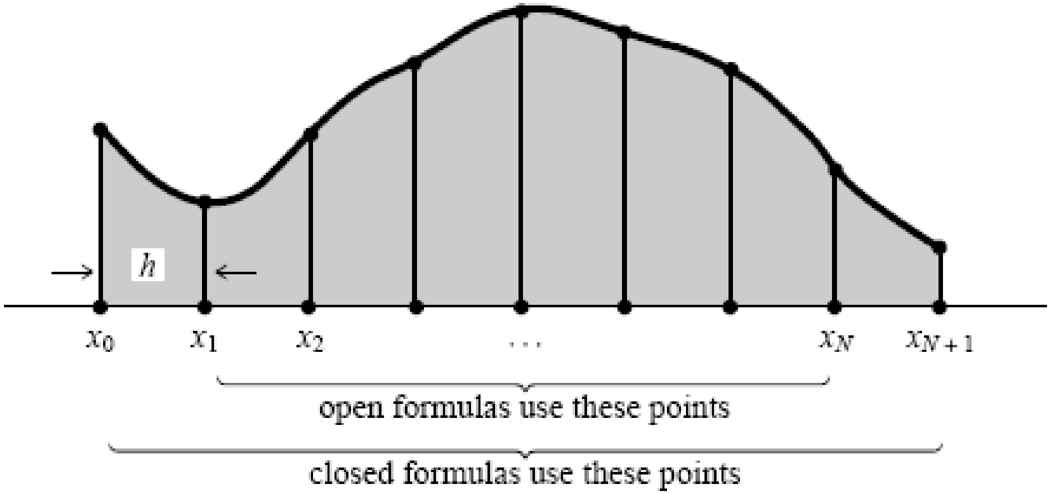
- chapter 17,18: integration of differential equations.
  - if possible, use this differential form when the function is concentrated in sharp peaks or when the shape is strongly scale-dependent. Also handy for functions of many variables.
- Also possible:
  - Chebyshev integration (next lecture)
  - Fourier integration – for periodic functions

Classical formulas

Functions at fixed stepsize h :

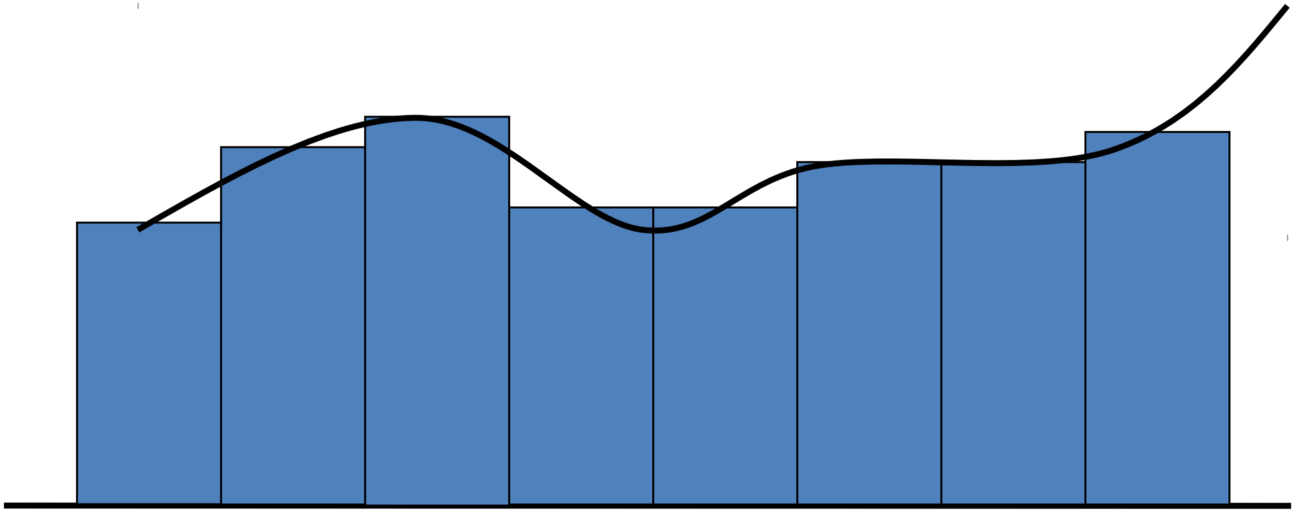
$x_i = x_0 + ih \qquad i = 0, \ \ldots, n$

$f(x_i) \rightarrow f_i$


h
x_0
x_1
x_2
...
x_N
x_{N+1}
open formulas use these points
closed formulas use these points

Trapezoidal Rule

- zeroth- order: function replaced by sum:

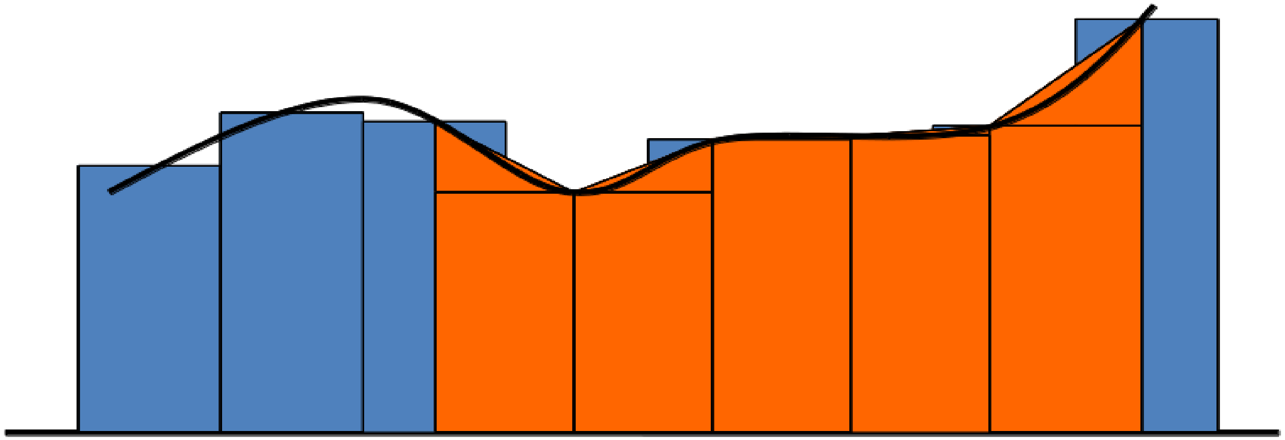


$$Zeroth - order: \quad \int f(x)dx \rightarrow h\sum f_i$$

limit of h → 0 : many sums (lots of computing time) and many additions of small numbers to the total → loss off accuracy

Trapezoidal Rule

- zeroth- order: function replaced by sum
- First order: linear interpolation between x_i and x_{i+1}



$$Zeroth-order: \quad \int f(x)dx \rightarrow h\sum f_i$$

$$First-order: \quad \int f(x)dx \rightarrow h\sum \frac{f_i + f_{i+1}}{2}$$

Trapezoidal rule, Simpsons rule

- The trapezoidal rule is correct for first-order polynomials. It considers 2 points, ½(f_0+f_1), so it is called a two-point function:

$$\text{Trapezoidial rule: } \int_{x_0}^{x_1} f(x)\,dx = h\frac{1}{2}[f_0+f_1]+O(h^3 f^{(2)}) \quad \text{with } f^{(n)}(x)=\frac{d^n f(x)}{dx^n}$$

- With more than 2 points, one can calculate an integral between these points to higher order. The three-point formula is correct to order 2 (and due to a cancellation, also to order 3): Simpsons rule

$$\text{Simpsons rule: } \int_{x_0}^{x_2} f(x)\,dx = h\frac{1}{3}[f_0+4f_1+f_2]+O(h^5 f^{(4)})$$

- The four-point formula is also correct up to order three.

Classical formulas

The five-point formula is correct to fifth order (again due to a cancellation) : Bode’s formula:

$$\text{Bode's rule: } \int_{x_0}^{x_4} f(x)\,dx = h\frac{1}{45}[14f_0+64f_1+24f_2+64f_3+14f_4]+O\left(h^7 f^{(6)}\right)$$

- factors: derive from writing down the equations with arbitrary factors p,q,r,s,...; calculate the integral for the functions 1, x, x^2, x^3, ... and solve the coupled equations. In that manner, all polynomials up to that order are represented correctly

Classical formulas

- f(x) = 1, x, x^2 :
  - Integrals with 3 points, 2 steps, running from 0 to 2 give: $pf_0+qf_1+rf_2$

$$\int_0^2 1\,dx = p+q+r = 2$$

$$\int_0^2 x dx = q+2r = 2$$

$$\int_0^2 x^2\,dx = q+4r = 8/3$$

  - p=1/3, q=4/3, r=1/3 (Simpsons rule)
  - (happens to be correct to third order too)

$$\int_0^2 x^3 dx = q+8r = 4$$

Extended formulas (Closed)

- calculate N times for intervals 0-1, 1-2,...

$$\int_{a=x_0}^{b=x_{N-1}} f(x)\,dx = h\left[\frac{1}{2}f_0 + f_1 + f_2 + \ldots + f_{N-2} + \frac{1}{2}f_{N-1}\right] + O\left(\frac{(b-a)^3}{N^2} f^{(2)}\right) \qquad \text{Trapezoidal}$$

$$\int_{x_0}^{x_{N-1}} f(x)\,dx = h\left[\frac{1}{3}f_0 + \frac{4}{3}f_1 + \frac{2}{3}f_2 + \frac{4}{3}f_3 + \ldots + \frac{2}{3}f_{N-3} + \frac{4}{3}f_{N-2} + \frac{1}{3}f_{N-1}\right] + O\left(\frac{1}{N^4} f^{(4)}\right) \qquad \text{Simpson's}$$

- Example : 5 points calculation of sin(x) between [0,pi] and e^x and sqrt(x) between [0,4], with trapezoidal, Simpson's, and Bode's rule:

Example, sin(x)

$$\int_{a=x_0}^{b=x_{N-1}} f(x)\,dx = h\left[\frac{1}{2}f_0+f_1+f_2+\ldots+f_{N-2}+\frac{1}{2}f_{N-1}\right]+O\left(\frac{(b-a)^3}{N^2}f^{(2)}\right) \qquad \text{Trapezoidal}$$

$$\int_{x_0}^{x_{N-1}} f(x)\,dx = h\left[\frac{1}{3}f_0+\frac{4}{3}f_1+\frac{2}{3}f_2+\frac{4}{3}f_3+\ldots+\frac{2}{3}f_{N-3}+\frac{4}{3}f_{N-2}+\frac{1}{3}f_{N-1}\right]+O\left(\frac{1}{N^4}f^{(4)}\right) \qquad \text{Simpson's}$$

Bode's rule: $\int_{x_0}^{x_4} f(x)\,dx = h\frac{1}{45}[14f_0+64f_1+24f_2+64f_3+14f_4]+O(h^7 f^{(6)})$

$$\sin(x): x_i = i\pi/4;\quad f_0=f_4=0,\quad f_1=f_3=1/2\sqrt{2}, f_2=1$$

$$\textit{trapezoidal}: \int_0^{\pi}\sin(x)\,dx = \frac{\pi}{4}\left[0+1/2\sqrt{2}+1+1/2\sqrt{2}+0\right]+O(1/N^2)=1.8961+O(1/N^2)$$

$$\textit{Simpson}: \int_0^{\pi}\sin(x)\,dx = \frac{\pi}{12}\left[0+2\sqrt{2}+2+2\sqrt{2}+0\right]+O(1/N^4)=2.0046+O(1/N^4)$$

$$\textit{Bode}: \int_0^{\pi}\sin(x)\,dx = \frac{\pi}{180}\left[0+32\sqrt{2}+24+32\sqrt{2}+0\right]+O(1/N^4)=1.9986+O(1/N^6)$$

Example, e^x

$$\int_{a=x_0}^{b=x_{N-1}} f(x)\,dx = h\left[\frac{1}{2}f_0+f_1+f_2+\ldots+f_{N-2}+\frac{1}{2}f_{N-1}\right]+O\left(\frac{(b-a)^3}{N^2}f^{(2)}\right) \qquad \text{Trapezoidal}$$

$$\int_{x_0}^{x_{N-1}} f(x)\,dx = h\left[\frac{1}{3}f_0+\frac{4}{3}f_1+\frac{2}{3}f_2+\frac{4}{3}f_3+\ldots+\frac{2}{3}f_{N-3}+\frac{4}{3}f_{N-2}+\frac{1}{3}f_{N-1}\right]+O\left(\frac{1}{N^4}f^{(4)}\right) \qquad \text{Simpson's}$$

Bode's rule: $\displaystyle\int_{x_0}^{x_4} f(x)\,dx = h\frac{1}{45}[14f_0+64f_1+24f_2+64f_3+14f_4]+O\left(h^7 f^{(6)}\right)$

$$e^x \quad : x_i = i; \quad f_i = e^i$$

$$trapezoidal: \int_0^4 e^x\,dx = \left[1/2+e+e^2+e^3+e^4/2\right] = 1.0819\left(e^4-1\right)$$

$$Simpson: \int_0^4 e^x\,dx = \frac{1}{3}\left[1+4e+2e^2+4e^3+e^4\right]+O\left(1/N^4\right) = 1.0049\left(e^4-1\right)$$

$$Bode: \int_0^4 e^x\,dx = \frac{1}{45}\left[14+64e+24e^2+64e^3+14e^4\right] = 1.0013\left(e^4-1\right)$$

Example, sqrt(x)

$$\int_{a=x_0}^{b=x_{N-1}} f(x)\,dx = h\left[\frac{1}{2}f_0+f_1+f_2+\ldots+f_{N-2}+\frac{1}{2}f_{N-1}\right]+O\left(\frac{(b-a)^3}{N^2}f^{(2)}\right) \qquad \text{Trapezoidal}$$

$$\int_{x_0}^{x_{N-1}} f(x)\,dx = h\left[\frac{1}{3}f_0+\frac{4}{3}f_1+\frac{2}{3}f_2+\frac{4}{3}f_3+\ldots+\frac{2}{3}f_{N-3}+\frac{4}{3}f_{N-2}+\frac{1}{3}f_{N-1}\right]+O\left(\frac{1}{N^4}f^{(4)}\right) \qquad \text{Simpson's}$$

$$\text{Bode's rule: } \int_{x_0}^{x_4} f(x)\,dx = h\frac{1}{45}[14f_0+64f_1+24f_2+64f_3+14f_4]+O\left(h^7 f^{(6)}\right)$$

$$\sqrt{x} \quad : x_i = i;\ \ f_i = \sqrt{i} \qquad\qquad \text{exact}:\ 5\frac{1}{3}$$

$$\textit{trapezoidal}: \int_0^4 \sqrt{x}\,dx = \left[0+1+\sqrt{2}+\sqrt{3}+1\right] = 5.1463$$

$$\textit{Simpson}: \int_0^4 \sqrt{x}\,dx = \frac{1}{3}\left[0+4+2\sqrt{2}+4\sqrt{3}+2\right] = 5.2522$$

$$\textit{Bode}: \int_0^4 \sqrt{x}\,dx = \frac{1}{45}\left[0+64+24\sqrt{2}+64\sqrt{3}+28\right] = 5.2621$$

Extended formulas

- Precision is gained, just by weighing the function values differently!
  - 1 order of precision obtained, just from relative weight of end point (both for Simpson and for trapezoidal rule)
  - The alternation of weights (4/3 – 2/3 -4/3) in the interior for Simpson's rule is nothing special; an extended formula with equal weight interior points and the same order of precision can be obtained from using the 4-point function.
  - The increase in precision is just due to the special treatment of end points

Extended formulas (open)

- When integrating from a pole, or towards infinity, one cannot enclose the endpoint. One can use an open formula to calculate towards the endpoint:
  - Lowest order :

$$\int_{x_0}^{x_1} (f_x) = 1/2\,h[f_0 + f_1] + O\left(h^3 f^{(2)}\right) \rightarrow = h f_1 + O\left(h^2 f\,'\right)$$

- higher orders:

$$\int_{x_0}^{x_1} (f_x) \rightarrow h[3/2 f_1 - 1/2 f_2] + O\left(h^3 f^{(2)}\right)$$

$$\int_{x_0}^{x_1} (f_x) \rightarrow h[23/12 f_1 - 16/12 f_2 + 5/12 f_3] + O\left(h^4 f^{(3)}\right)$$

Extended formulas (open)

- construct them by using the closed formula for the interior part and adding the extrapolated open step for the ends.
  - end steps: done only once. **Use 1 order lower extrapolation** than for interior steps.

Extended open Trapezoidal

$$\int_{x_0}^{x_{N-1}} f(x) = h\left[\frac{3}{2} f_1 + f_2 + f_3 + \ldots + f_{N-3} + \frac{3}{2} f_{N-2}\right] + O(1/N^2)$$

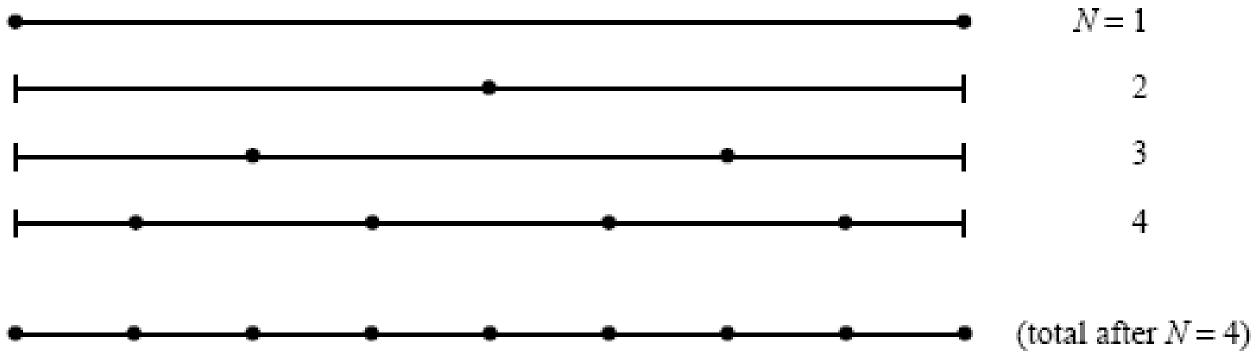
Extended open Simpsons

$$\int_{x_0}^{x_{N-1}} f(x) = h\left[\frac{27}{12} f_1 + 0 + \frac{13}{12} f_3 + \frac{4}{3} f_4 + \ldots + \frac{4}{3} f_{N-5} + \frac{13}{12} f_{N-4} + 0 + \frac{27}{12} f_{N-2}\right] + O(1/N^4)$$

- open formulas: 1 order lower precision (in stepsize) overall.

Elementary algorithms

- Starting point: extended trapezoidal rule



- one can add new points in between without losing the previous work.
- error extrapolation is **entirely even** in powers of 1/N

Euler-MacLaurin Summation formula

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

$$-\frac{B_{2k}h^{2k}}{(2k)!}\left(f_{N-1}^{(2k-1)} - f_0^{(2k-1)}\right)$$

Bernouille number B: $\frac{t}{e^t - 1} = \sum_{n=0}^{n} B_n \frac{t^n}{n!}$

- The exact solution of an integral can be expanded in the Euler-MacLaurin summation series.
- Only even terms in the stepsize enter.
- Coeffients given by the Bernouille numbers
  - They grow large for large for large n

Romberg integration

- The leading error in the second step will be ¼ of the previous step. It can be cancelled by:

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

- The next order error is fourth-order in 1/N.
- This procedure, applied once, is exactly equivalent to Simpsons rule.
- Romberg integration: use k successive refinements to eliminate errors up to O(1/N^{2k}).
  - Converges much more rapidly than trapezoid rule or Simpsons rule.

Improper integrals

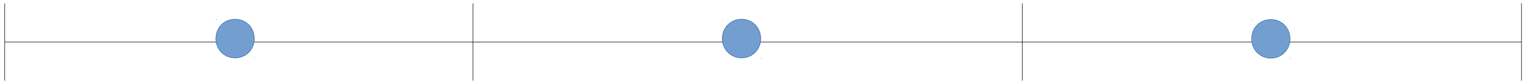
- Improper:
  - finite value, but cannot be calculated at boundary (e.g. sinx/x at 0)
  - one of the limits is infinity
  - integrable singularity (e.g. 1/sqrt(x) at 0)
    - at boundary
    - at known place in interval [a,b]
    - at unknown place
- How to solve? we need an algorithm like the Romberg integration, but for **open** formulas. Extended midpoint rule also only has even errors:

$$\int_{x_0}^{x_{N-1}} f(x)dx = h\left[\frac{1}{2}f_{1/2} + f_{3/2} + \ldots + f_{N-3/2}\right] - B_2\frac{h^2}{4}\left(f'_{N-1} - f'_0\right) - \ldots$$

$$-\frac{B_{2k}h^{2k}}{(2k)!}\left(1-2^{-2k+1}\right)\left(f_{N-1}^{(2k-1)} - f_0^{(2k-1)}\right) + \ldots..$$

Second Euler-MacLaurin summation formula, using midpoints

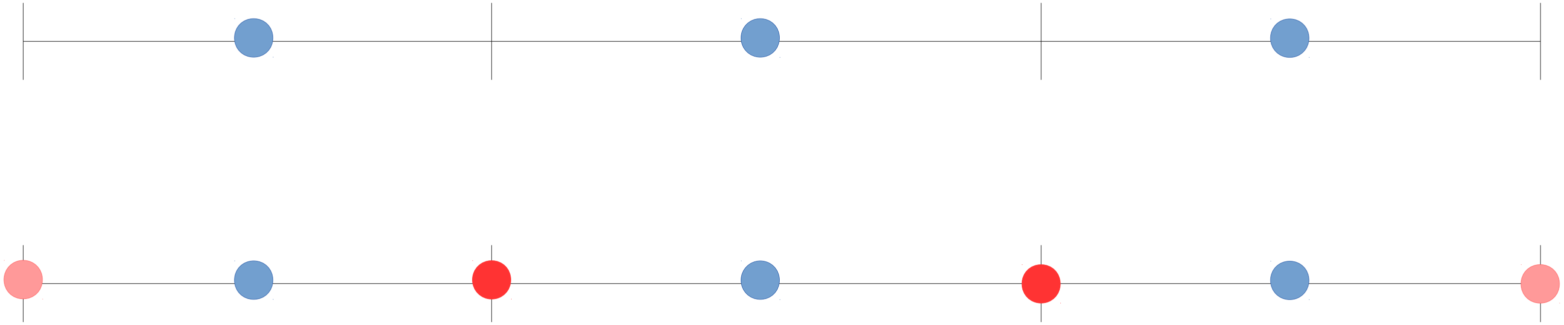
Open integration



Using the midpoints, one gets the same sum as the trapezoidal rule. (see page 8).

- One cannot double the points however:

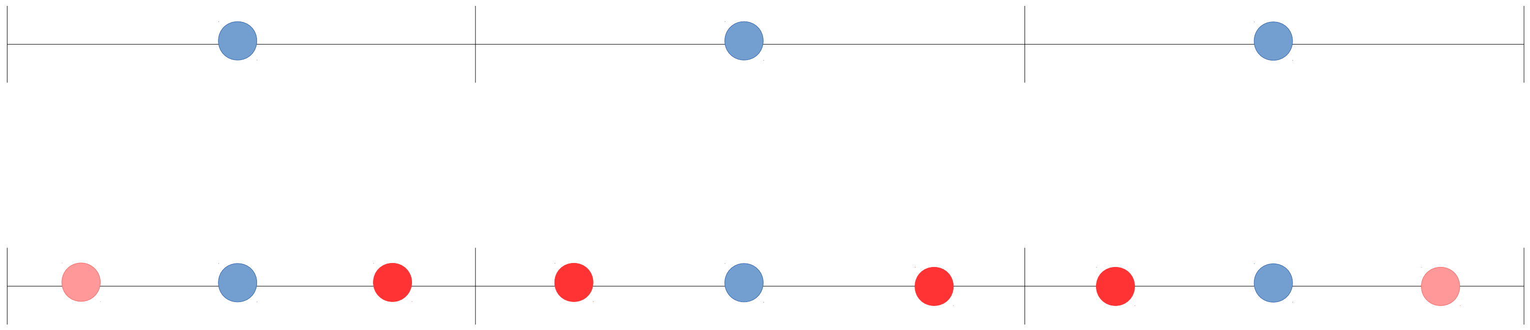
Open integration



Using the midpoints, one gets the same sum as the trapezoidal rule. (see page 8).

- One cannot double the points.
- But one could TRIPLE the points:

Open integration



Using the midpoints, one gets the same sum as the trapezoidal rule. (see page 8).

- One cannot double the points.
- But one could TRIPLE the points! (1/6, ½, 5/6, 7/6, 3/2, 11/6, 13/6, 5/2 ...

Open integration

- It is not possible to double the number of points by adding intermediate points, but you can **triple** them.
- the new sum is $S=(9S_{3N}-S_N)/8$.
- routine qromo does Romberg-like integration for open intervals.
- infinite boundary → change of variables. E.g.

$$\int_a^b f(x)dx \Leftrightarrow \int_a^b f(1/t)d(1/t) = \int_{1/b}^{1/a} \frac{f(1/t)}{t^2} dt$$

$$\int_{-\infty}^{\infty} f(x)dx = \int_{\frac{-\pi}{2}}^{\frac{\pi}{2}} f(\arctan(\phi)) \frac{1}{1+\phi^2} d\phi$$

Variable transformations

- Variable change needed, for integrable singularities at endpoints. Can be coded by yourself, but NR version 3 has methods for it:

$$\int_a^b f(x)\,dx \Leftrightarrow \int_{1/b}^{1/a} \frac{1}{t^2} f(t)\,dt \quad \text{with} \quad t = 1/x \quad \text{and } (ab>0) \qquad \text{when } \lim_{x\to\infty} f(x) < 1/x^2$$

$$\int_a^\infty f(x)\,dx \Leftrightarrow \int_0^{e^{-a}} \frac{1}{t} f(-\log(t))\,dt \quad with \quad t = e^{-x} \quad \text{when function decays exponentially}$$

- Routine Midpoint that is used (in quadrature.h) with the Romberg integration can be replaced for these coordinate changes with midinf and midexp, respectively
- Also implemented: TANH rule and double exponential rule – to get an integrand that goes to zero rapidly at the end points (Derule in quadrature.h)

$$I = \int_a^b f(x)dx = \int_c^d f(t)\frac{dx}{dt}dt \qquad x = \frac{1}{2}(b+a) + \frac{1}{2}(b-a)\tanh(t) \quad x \in [a,b], t \in [-\infty, \infty]$$

$$\frac{dx}{dt} = \frac{1}{2}(b-a)\frac{1}{\cosh^2 t} = \frac{2}{b-a}(b-x)(x-a)$$

Gaussian quadrature

- Freedom to choose coefficients (Romberg integration): leads to much faster convergence.
- Freedom to choose abscissas: extra degree of freedom
  - you can arrive at higher-order convergence faster, when integrand is smooth (**exponential convergence**!)
  - you can make the integral exact for a class of functions of polynomials*weight function W

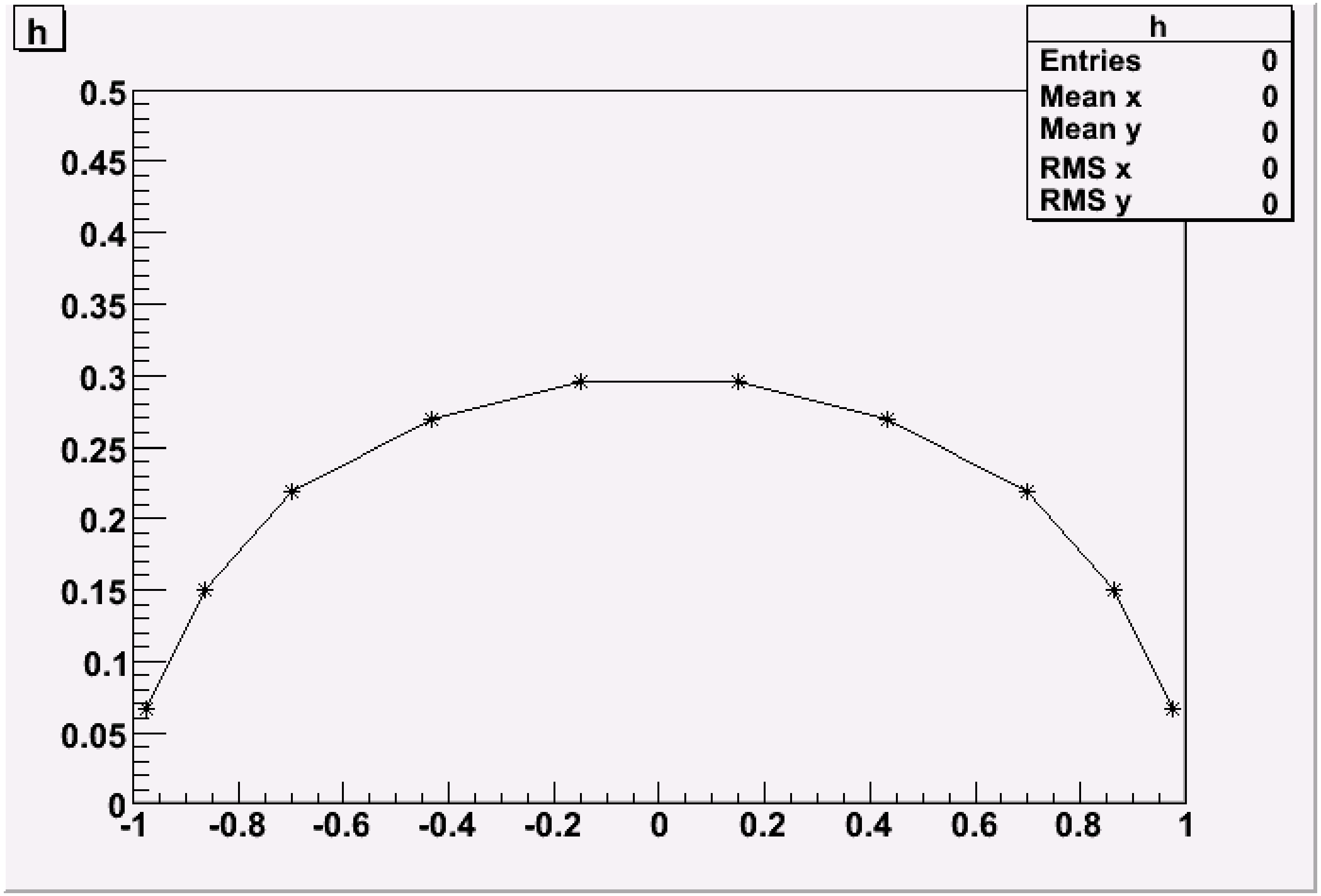
$$\int_a^b W(x) f(x)\,dx \approx \sum_{j=0}^{N} w_j f(x_j) \text{ Exact for f(x) polynomial}$$

e.g. $\displaystyle\int_{-1}^{1} \frac{e^{-\cos^2 x}}{\sqrt{(1-x^2)}}\,dx$

choose weight function $\quad W(x) = \dfrac{1}{\sqrt{(1-x^2)}} \quad$ (Gauss-Chebyshev)

Gauss-Legendre

- Gauss-Legendre: legendre polynomials, W=1. 10-point example of abscissa and weights



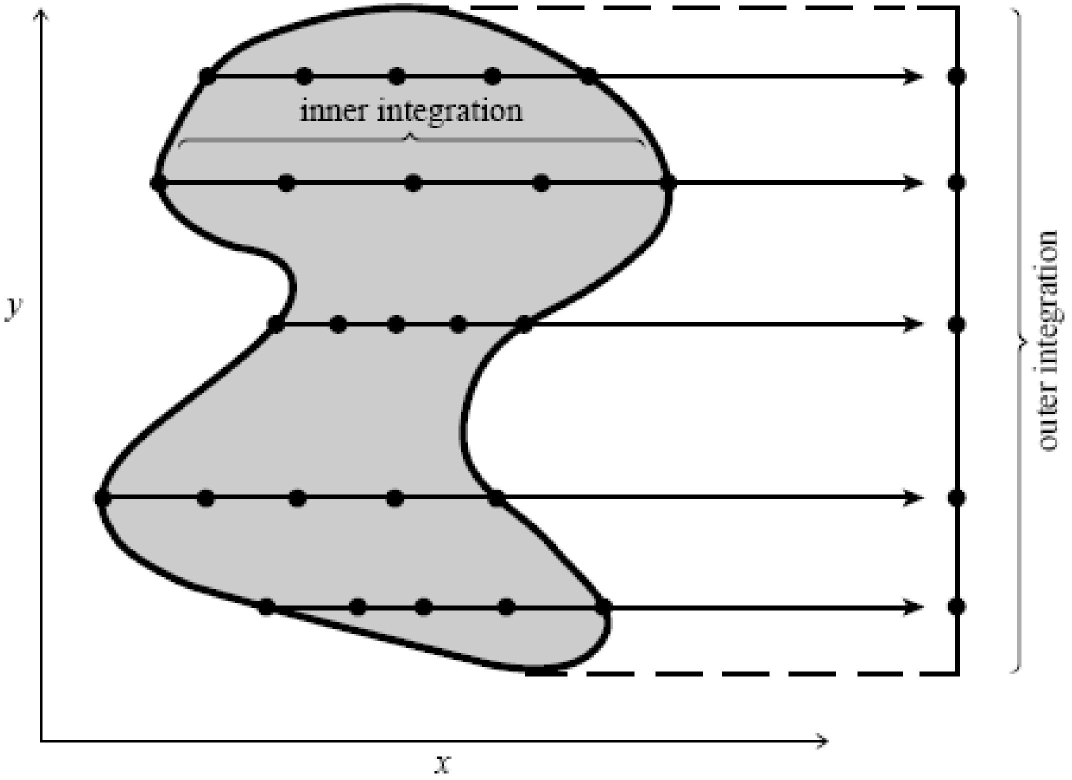
Gaussian quadrature

- Fine for smooth functions
  - For *periodic* functions, the order of the quadrature should be higher than the number of roots, in general
    - sin(x) f(x) → Fourier techniques
  - Needs a bit of experimentation to see how well a quadrature scheme does.

Multi-dimensional integrals

- number of function evaluations grows like the power of the dimension (e.g. typically 30 for 1-dim, 30,000 for 3 dim)
- boundary may be complicated
- Try always to reduce it to lower dimension integrals.
  - (e.g. spherical symmetry: polar coordinates)
- complicated boundary: resort to Monte Carlo techniques.
  - Take care when function is strongly peaked.
- if boundary is simple, Gaussian quadratures may give a relatively fast answer.

Multidimensional integration



Numerical Recipes v3 routines:

- Simpson rule, and trapezoid : functions returning a double.

  Doub qsimp(T &func, const Doub a, const Doub b, const Doub eps=1.0e-10) in quadrature.h
- T is template for the type the function &func returns. a and b are the integration limits. eps is the relative target accuracy.
  - func(x) should however take a Doub or double as input variable
  - You may want to set JMAX higher.
- Romberg: quadrature.h, romberg.h
- Gaussian quadrature: calculate weights and abscissa with qgaus.h
  - You can check for needed include files from the nr website (see first lecture)
  - You can check the value of an integral with Wolfram (see link on www.nikhef.nl/~henkjan/ computational methods)

Example Romberg

```
struct Quadrature{                                    // in  (quadrature.h)
        Int n;
        virtual Doub next() = 0;
};
template<class T>          //(to make machine independent. T can be double T or float or int or real*8 etc)
struct Trapzd : Quadrature {
        Doub a,b,s;
        T &func;
        Trapzd() {};
        Trapzd(T &funcc, const Doub aa, const Doub bb) :
                func(funcc), a(aa), b(bb) {n=0;}
        Doub next() {
                Doub x,tnm,sum,del;
                Int it,j;
                n++;
                if (n == 1) {
                        return (s=0.5*(b-a)*(func(a)+func(b)));    //trapezoid rule
                } else {
                        for (it=1,j=1;j<n-1;j++) it <<= 1;
                        tnm=it;
                        del=(b-a)/tnm;
                        x=a+0.5*del;
                        for (sum=0.0,j=0;j<it;j++,x+=del) sum += func(x);
                        s=0.5*(s+(b-a)*sum/tnm);
                        return s;
                }
        }
};
```

Example Romberg

```
template<class T>
Doub qtrap(T &func, const Doub a, const Doub b, const Doub eps=1.0e-10) {
	const Int JMAX=20;
	Doub s,olds=0.0;
	Trapzd<T> t(func,a,b);
	for (Int j=0;j<JMAX;j++) {
		s=t.next();
		if (j > 5)
			if (abs(s-olds) < eps*abs(olds) ||
				(s == 0.0 && olds == 0.0)) return s;
		olds=s;
	}
	throw("Too many steps in routine qtrap");
}
template<class T>
Doub qsimp(T &func, const Doub a, const Doub b, const Doub eps=1.0e-10) {
	const Int JMAX=20;
	Doub s,st,ost=0.0,os=0.0;
	Trapzd<T> t(func,a,b);
	for (Int j=0;j<JMAX;j++) {
		st=t.next();
		s=(4.0*st-ost)/3.0;
		if (j > 5)
			if (abs(s-os) < eps*abs(os) ||
				(s == 0.0 && os == 0.0)) return s;
		os=s;
		ost=st;
	}
	throw("Too many steps in routine qsimp");
}
```

Trapezoid interpolation can be called directly via
Result=qtrap(func,xlow,xhig, precision)

Simpson interpolation can be called directly via
Result=qsimp(func,xlow,xhig, precision)

Example Romberg

```
template <class T>
Doub qromb(T &func, Doub a, Doub b, const Doub eps=1.0e-14) {
        const Int JMAX=30, JMAXP=JMAX+1, K=9;
        VecDoub s(JMAX),h(JMAXP);
        Poly_interp polint(h,s,K);
        h[0]=1.0;
        Trapzd<T> t(func,a,b);
        for (Int j=1;j<=JMAX;j++) {
                s[j-1]=t.next();
                if (j >= K) {
                        Doub ss=polint.rawinterp(j-K,0.0);
                        if (abs(polint.dy) <= eps*abs(ss)) return ss;
                }
                h[j]=0.25*h[j-1];
        }
        throw("Too many steps in routine qromb");
}
```

Modified JMAX, K to be able to make more steps. If not, the routine throws already after K=6 iterations.

Poly_interp: interp_1d needed

Trapzd: quadrature needed

For qromo (open : also Midpoint used, in quadrature include file)

Example Romberg

```
#include "nr3.h"
#include "interp_1d.h"
#include "quadrature.h"
#include "romberg.h"
#include <iostream>

using namespace std;

Doub f1(Doub x) {
        if (fabs(x)>1e-30) return (sin(x)/x);
        return 1;
}

Doub f2(Doub x) {
        if (x<-1000) return 0;
        if (x>1000) {
                cerr << " overflow, exp(1000) asked), function is infinite " << endl;
                return 1e200;
        }
        return exp(x);
}

int main() {
        Doub a,b;
        cout << "Give Boundaries " << endl;
        cin >> a >> b;
        cout << " Romberg sinx/x between " << a << " and " << b << " = " <<qromb(f1,a,b,1e-12) << endl;
        cout << " Romberg e^x between " << a << " and " << b << " = " <<qromb(f2,a,b,1e-12) << endl;
        return 0;
}
```

Example Gaussian Quadrature

- Using weight W(x) = 1, Legendre polynomials

```
#include "nr3.h"
#include "gamma.h"
#include "gauss_wgts.h"
#include <iostream>

using namespace std;

Doub f1(Doub x) {
        if (fabs(x)>1e-30) return (sin(x)/x);
        return 1;
}

int main() {
        Doub a,b;
        VecDoub xval(100),weight(100);
        cout << "Give Boundaries " << endl;
        cin >> a >> b;
        gauleg(a,b,xval,weight);
        Doub integral=0;
        for (int i=0;i<100;i++) integral+=f1(xval[i])*weight[i];
        cout << " Gaussian quadrature sinx/x between "  << a << " and " << b << " = " << integral << endl;
        return 0;
}
```

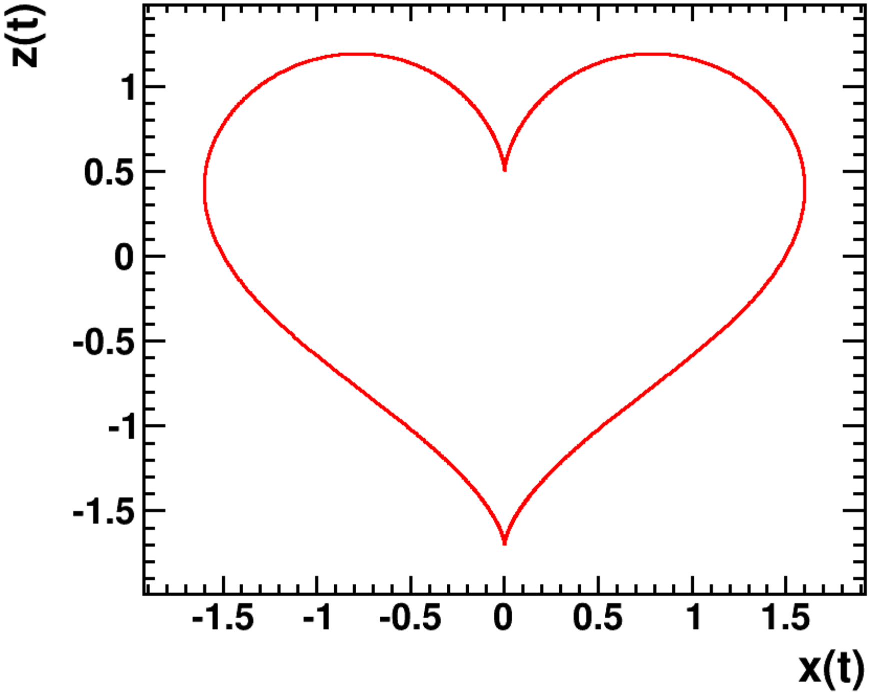
Valentine’s Exercise

- The graph shown is a heart-shaped curve, parametrized as

$$x = 1.6\sin^3(t)$$
$$z = 1.3\cos(t) - 0.5\cos(2t) - 0.2\cos(3t) - 0.1\cos(4t)$$
$$0 \le t \le 2\pi$$

  the surface of this curve equals 1.8π.

- A three-dimensional extension is easily obtained in cylindrical coordinates, by equating positive x (t E [0,pi]) as radius. A function enclosed in the interior of this surface can be integrated.
- In exercise 4, the volume integral over the heart shape of the function r^z is calculated.



/home/henkjan/WINDOWS/CompMeth/CompMeth2017/Exercises/Exercise4.pdf