



8. PRECISION INTEGRATION, NON-LINEAR EQUATIONS

8.1.1 Remember: what we saw during the last lesson

Numerical error propagation, convergence and Richardson extrapolation

Differentiation using finite differences

Integration Newton-Cotes quadrature rules (mid-point, trapezoidal, Simpson)

8.1.2 Overview: what you will learn today

Integration of periodic functions, cubic Hermite polynomials and splines

Gauss & Gauss-Konrod quadrature

Methods for non-linear polynomial equations, bisection, secant and Newton-Raphson

8.2 More on Newton-Cotes quadrature (NAM 5.2, H 8.3.1)

Error estimates are obtained experimentally by comparing successive approximations. Using the highest resolution estimates from the previous example:

```
>> err= abs(1.64207050293635-1.64207049800087)
err = 4.935480024670369e-09
```

with the final result $\int_0^2 \sqrt{\frac{1}{2} + 2 \exp(-x) \sin(2x^2)} dx = 1.642070498 \pm 5 \cdot 10^{-9}$.

The mid-point and trapezoidal rules both integrate exactly linear polynomials.

The Simpson rule integrates exactly cubic polynomials.

In Matlab use recursive Newton-Cotes quadratures with `quad()` or `quadl()`

```
>> I1=quad(inline('sqrt(0.5+2*exp(-x).*sin(2*x.^2))'),0,2,1E-7);
Warning: Recursion level limit in quad reached 148 times. Singularity likely.
>> I2=quad8(@fq,0,2,1E-10); % Function defined in a separate file fq.m
>> I=[I1 I2]
```

```
I = 1.64207049863185 1.64207049860671
```

Periodic functions are best integrated with a trapezoidal rule without extrapolation.

It can be shown that the periodicity cancels out the errors for all powers

$$I = T(h) - \frac{f'(b) - f'(a)}{12} h^2 + \frac{f'''(b) - f'''(a)}{720} h^4 - \dots$$

resulting in an asymptotic convergence that is more rapid than any power of h .

Higher order rules can in principle be defined using Romberg's method.

Runge oscillations however yield negative weights w_i and large cancellation errors.

8.3 Gauss quadrature (NAM 5.3, H 8.3.3)

Choose $2N = 4$ coefficients $(x_1, w_1), (x_2, w_2)$ for a quadrature

$$I = \int_{-1}^1 f(x) dx \approx G_2 \equiv w_1 f(x_1) + w_2 f(x_2)$$

so as to integrate exactly polynomials up to $2N - 1 = 3$ rd order (cubic)

$$\begin{aligned} w_1 + w_2 &= \int_{-1}^1 1 dx = 2, & w_1 x_1 + w_2 x_2 &= \int_{-1}^1 x dx = 0, \\ w_1 x_1^2 + w_2 x_2^2 &= \int_{-1}^1 x^2 dx = \frac{2}{3}, & w_1 x_1^3 + w_2 x_2^3 &= \int_{-1}^1 x^3 dx = 0. \end{aligned}$$

This requires solving a system of $2N = 4$ non-linear equations; the solution for the nodes x_i coincide with the zeros of Legendre polynomials and are generally tabulated: here $x_{1,2} = \mp 1/\sqrt{3}$, $w_{1,2} = 1$ leads to a quadrature

$$I \approx 1 \cdot f(-0.57735) + 1 \cdot f(0.57735)$$

A linear transformation can be used to map any interval $[a; b]$ to $[-1, 1]$

$$\int_a^b f(t) dt = \frac{b-a}{2} \int_{-1}^1 f\left(\frac{b-a}{2}x + \frac{b+a}{2}\right) dx \approx \frac{b-a}{2} \sum_{i=1}^N w_i f\left(\frac{b-a}{2}x_i + \frac{b+a}{2}\right)$$

Gauss quadrature using N nodes to integrate exactly polynomials of order $2N - 1$

$$\int_a^b f(t)dt \approx G_N \equiv \frac{b-a}{2} \sum_{i=1}^N w_i f(t_i), \quad t_i = \left(\frac{b-a}{2}\right) x_i + \left(\frac{b+a}{2}\right)$$

With Matlab the nodes and weights can be tabulated as

```
function Ig=gauss(N,F,a,b)
% Gauss quadrature of order N, integrand F, interval [a,b]
switch N
case 2, xg=[.57735026918963]; wg=[
case 3, xg=[
0 .77459666924148]; wg=[.88888888888889 .55555555555556];
case 4, xg=[.33998104358486 .86113631159405]; wg=[.65214515486255 .34785484513745];
case 5, xg=[
0 .53846931010568 .90617984593866];
wg=[.56888888888889 .47862867049937 .23692688505619];
case 6, xg=[.23861918608320 .66120938646627 .93246951420315];
wg=[.46791393457269 .36076157304814 .17132449237917];
case 7, xg=[
0 .40584515137740 .74153118559939 .94910791234276];
wg=[.41795918367347 .38183005050512 .27970539148928 .12948496616887];
otherwise out=NaN; return
end
nh=floor((N+1)/2); n1=nh-floor(N/2)+1; % Complete nodes and weights
xg=[-fliplr(xg(n1:nh)) xg]; wg=[fliplr(wg(n1:nh)) wg];
hh=(b-a)/2; abm=(a+b)/2; tg=hh*xg+abm; % Change interval to [-1;1]
fg=feval(F,tg); Ig=hh*sum(wg.*fg); % Result of Gauss quadrature

>> g=[]; for k=2:5, g=[g gauss(k,inline('exp(-x.^2)'),0,1)]; end; g
g = 0.74659468828286 0.74681458419126 0.74682446813099 0.74682412676625
```

Runge oscillations do not appear for large N because x_i are not evenly spaced: the weights w_i remain all positive, preventing cancellation errors. Unfortunately previous evaluations cannot be re-used when increasing $N \rightarrow N + 1$.

Gauss-Konrod uses 7 Gauss nodes and 8 Konrod nodes for two quadratures

$$G_7 = \frac{b-a}{2} \sum_{i=1}^7 w_i f(t_i), \quad t_i = \left(\frac{b-a}{2}\right) x_i + \left(\frac{b+a}{2}\right)$$

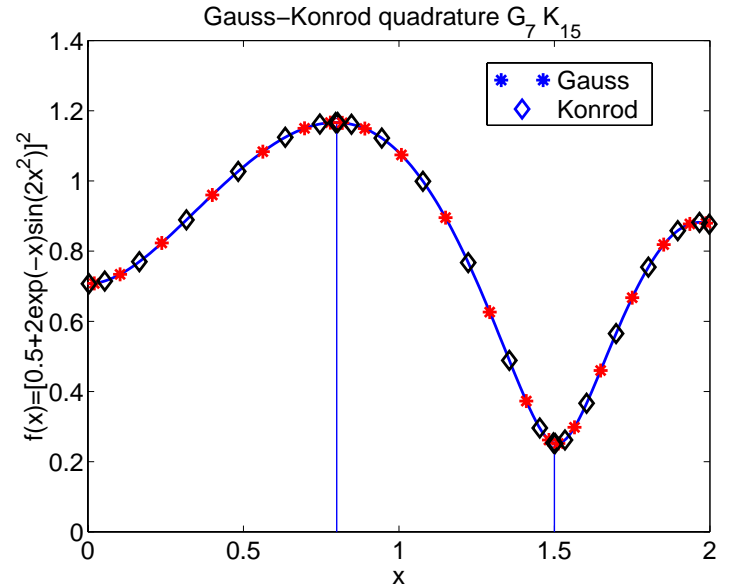
$$K_{15} = \frac{b-a}{2} \left(\sum_{i=1}^7 u_i f(t_i) + \sum_{i=1}^8 v_i f(\tau_i) \right), \quad \tau_i = \left(\frac{b-a}{2}\right) \xi_i + \left(\frac{b+a}{2}\right)$$

where K_{15} reuses the values $f(t_i)$ from G_7 to integrate exactly polynomials only up to order 20 (instead of 30 with 15 Gauss points), but provides an error estimate

$$\text{Error} < \frac{10d}{\sqrt{d/|K_{15}|}}, \quad d = |K_{15} - G_7|$$

With Matlab divide the quadrature into 3 intervals

```
>> x=[0 0.8 1.5 2]; Ig=[]; Ik=[]; Ek=[];
>> for i=1:length(x)-1,
    Igi =gauss(7,@fq,x(i),x(i+1)); Ig=[Ig Igi];
    [Iki,e]=g7k15(@fq,x(i),x(i+1)); Ik=[Ik Iki];
    Ek=[Ek e];
end
>> G7=sum(Ig), K15= sum([Ik;Ek],2)
G7 = 1.64207309184149
K15 = 1.64207049806980
      0.00000009898750
```



In Matlab the Gauss-Konrod nodes and weights can be tabulated as

```
function [Igk,err]=g7k15(F,a,b);
% Gauss-Kronrod with 15-points, integrand F, interval [a,b]
% 7 Gauss points and weights
xg=[.405845151377397 .741531185599394 .949107912342758];
wg=[.381830050505119 .279705391489277 .129484966168870];
xg=[-fliplr(xg) 0 xg]; wg=[fliplr(wg) 0.417959183673469 wg];
% 8 Kronrod points and new weights
xk=[.207784955007898 .586087235467691 .864864423359769 .991455371120813];
u=[.190350578064785 .140653259715525 .063092092629979];
v=[.204432940075298 .169004726639267 .104790010322250 .022935322010529];
xk=[-fliplr(xk) xk]; u=[fliplr(u) .209482141084728 u]; v=[fliplr(v) v];
% Quadrature in standard interval [-1,1]
hh=(b-a)/2; abm=(a+b)/2;
tg=hh*xg+abm; fg=feval(F,tg); G7 = hh*sum(wg.*fg);
tk=hh*xk+abm; fk=feval(F,tk); K15= hh*(sum(u.*fg)+sum(v.*fk));
% Error estimate
d=abs(K15-G7); err=10*d*sqrt(d/abs(K15));
% Return approximation and error estimate
Igk=K15; err=max([eps*abs(K15) err]);
```

Peer Teaching (2 × 1 minutes to think, explain to your neighbour and vote)

Choosing a quadrature. Which method / rule should you use to integrate numerically with the highest precision possible

a) A cubic (Hermite or Bézier) polynomial

$$c) \int_0^{2\pi} (1 + \sin x) e^{\cos x} dx$$

b) A square root

$$d) \int_0^{2\pi} (1 + \sin x) e^{\sqrt{3}x} dx$$

↙ trapezoidal

↗ Simpson (Romberg)

↘ G_2

↙ $G_7 K_{15}$

8.4 Tricks and other quadratures (NAM 5.5-5.6, H 8.4)

Singularities: change variable, integrate analytically or use the lowest order possible

$$\int_0^{\pi/2} \frac{\cos(x)}{\sqrt{x}} \overset{\boxed{\sqrt{x}=t}}{=} \int_0^{\sqrt{\pi/2}} \frac{\cos(t^2)}{t} 2t dt = 2 \int_0^{\sqrt{\pi/2}} \cos(t^2) dt$$

$$\int_0^{\pi/2} \frac{1}{\sqrt{\sin x}} \overset{\boxed{-\frac{1}{\sqrt{x}} + \frac{1}{\sqrt{x}}}}{=} \underbrace{\int_0^{\pi/2} \left(\frac{1}{\sqrt{\sin x}} - \frac{1}{\sqrt{x}} \right) dx}_{\text{well behaved}} + \underbrace{\int_0^{\pi/2} \frac{1}{\sqrt{x}} dx}_{\text{analytically}=\sqrt{2\pi}}$$

Improper integrals: transform with $x=-\log t$, $x=\frac{t}{1-t}$ or find an upper bound for the tail

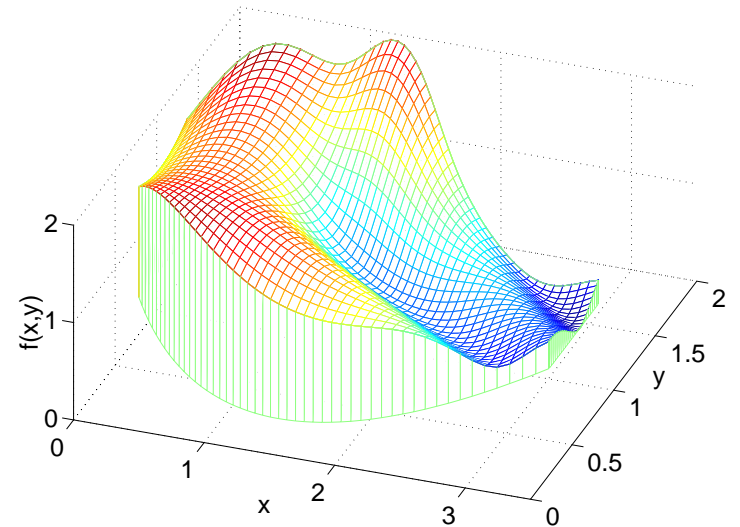
$$\begin{aligned} \int_0^{\infty} \frac{dx}{x^4 + \sqrt{1+x^6}} &= \int_0^X \frac{dx}{x^4 + \sqrt{1+x^6}} + \underbrace{\int_X^{\infty} \frac{dx}{x^4 + \sqrt{1+x^6}}}_{\text{Error} < \int_X^{\infty} \frac{dx}{x^4} = X^{-3}/3} \\ &= \int_0^{15} \frac{dx}{x^4 + \sqrt{1+x^6}}, \quad \text{Error} < 10^{-4} \end{aligned}$$

Double integrals are treated one dimension after the other

$$\int \int_D f(x, y) dx dy = \int_a^b \underbrace{\left[\int_{c(x)}^{d(x)} f(x, y) dy \right]}_{g(x)} dx$$

```
function f=fq2d(x,y)
    f=sqrt(3+sin(3*x*y)+cos(3*y)).*exp(-(x/pi).^2);

function I=q2d(Q);
%Quadrature in 2D, Q the refinement factor
nx=100*Q; ny=50*Q; a=0; b=pi; % Domain a,b
dx=(b-a)/nx; x=(a:dx:b)';
c=x.^2/pi-x+pi/4; % Domain c(x)
d=x/pi+exp(-x/pi)+exp(-(x-1).^2); % d(x)
Y=[]; F=[]; G=[];
for i=1:nx+1
    dy=(d(i)-c(i))/ny; y=c(i):dy:d(i);
    f=fq2d(x(i),y);
    g=dy*(sum(f)-0.5*(f(1)+f(ny+1)));%Trapezoidal-y
    Y=[Y; y]; F=[F; f]; G=[G; g];
end
I=dx*(sum(G)-0.5*(G(1)+G(nx+1))); %Trapezoidal-x
return
>>
>> I1=q2d(1); I2=q2d(2); I=[I1 I2 I2+(I2-I1)/3]
I = 5.76381892297752 Simple resolution
    5.76411082260474 Double resolution
    5.76420812248047 Romberg / Simpson 2D
```



For rectangular integration domains D , you can also use `dblquad()`.

8.5 Non-linear equations (NAM 6.1-6.5, H 5.5)

Problem: find x a solution of the non-linear equation $f(x) = 0$

- $f(x)$ changes sign unless x is a double root, e.g. $(x - 1)^2 = 0$
- $f'(x)$ changes sign if x is a double root, e.g. $2x - 2 = 0$

Polynomial equations are solved using Matlab's command `root()`

The zeros of $P_7(x) = (-35x + 315x^3 - 693x^5 + 429x^7)/16$, the Legendre polynomial used in the Gauss-Konrod G_7K_{15} quadrature, are

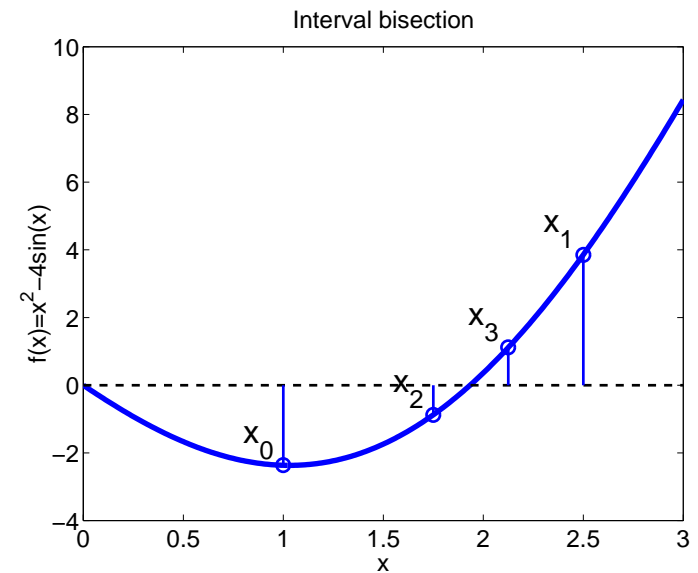
```
>> xg7=roots([429 0 -693 0 315 0 -35 0])'  
xg7 = 0 0.9491 -0.9491 0.7415 -0.7415 0.4058 -0.4058
```

Bisection method: divide the interval succesively by two

```
function x=bisection(F,a,b,tol)  
fa=feval(F,a);  
while abs(b-a)>tol  
x=a+(b-a)/2; fx=feval(F,x);  
if sign(fa)==sign(fx)  
a=m; fa=fx;  
else  
b=m; fb=fx;  
end  
end  
return
```

```
>> bisection(inline('x.^2-4*sin(x)'),1.,2.5,1E-6)  
ans = 1.93375420570374 (21 iterations)
```

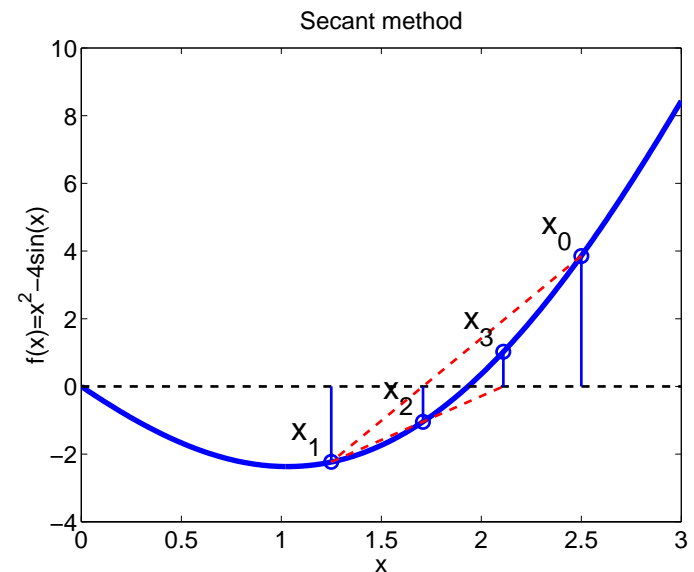
Robust, but very slow with $\text{Error} < \log_2 \left(\frac{b-a}{\text{tol}} \right)$



Secant method: $x_{n+1} = x_n - \frac{(x_n - x_{n-1})f(x_n)}{f(x_n) - f(x_{n-1})}$

```
function x=secant(F,a,b,tol)
    fa=feval(F,a); fb=feval(F,b);
    while abs(b-a)>tol
        x=b-fb*(b-a)/(fb-fa); fx=feval(F,x);
        a=b; fa=fb;
        b=x; fb=fx;
    end
    return
>> secant(inline('x.^2-4*sin(x)'),1.5,2.5,1E-6)
ans = 1.93375376282779 (6 iterations)
```

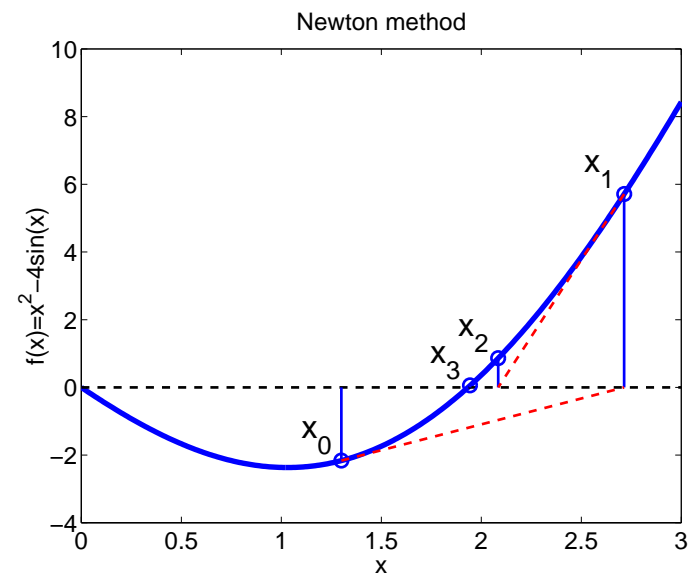
Fast superlinear convergence, but requires a good initial guess. Doesn't need $f'(x)$.



Newton Raphson method: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

```
function x=newton(F,Fp,x,tol)
    a=x+2*tol;
    while abs(x-a)>tol
        a=x;
        fx=feval(F,x); fxp=feval(Fp,x);
        x=x-fx/fxp;
    end
    return
>> newton(inline('x.^2-4*sin(x)'),...
           inline('2*x-4*cos(x)'),1.3,1E-6)
ans = 1.93375376282702
```

Superfast quadratic convergence, but requires a good initial guess and $f'(x)$.



Convergence rate p defined with $|x_{n+1} - \alpha| \approx C|x_n - \alpha|^p$

Denote the solution $x = \alpha$ and expand with Taylor around x_n

$$f(\alpha) = f(x_n) + (\alpha - x_n)f'(x_n) + \frac{(\alpha - x_n)^2}{2}f''(x_n) + \dots$$

Divide by $f'(x_n) \neq 0$ and substitute the solution $f(\alpha) = 0$

$$0 = \frac{f(x_n)}{f'(x_n)} + \alpha - x_n + (\alpha - x_n)^2 \frac{f''(x_n)}{2f'(x_n)} + \dots$$

$$\underbrace{x_n - \frac{f(x_n)}{f'(x_n)}}_{\text{Newton: } x_{n+1}} - \alpha = (\alpha - x_n)^2 \frac{f''(x_n)}{2f'(x_n)} + \dots$$

which yields a quadratic convergence near a single root

$$|x_{n+1} - \alpha| \approx C|x_n - \alpha|^2, \quad C = \lim_{n \rightarrow \infty} \left| \frac{f''(x_n)}{2f'(x_n)} \right| = \left| \frac{f''(\alpha)}{2f'(\alpha)} \right|$$

The convergence gets down to linear near a double root $f'(\alpha) = 0$.

Fixed point iteration method: rewrite $f(x) = 0$ in a form $x_{n+1} = G(x_n)$

$$f(x) = x^2 - 4 \sin x = 0 \Rightarrow x_{n+1} = 2\sqrt{\sin x_n}$$

```
function x=fixiter(F,x,tol)
    a=x+2*tol;
    while abs(x-a)>tol
        a=x;
        x=feval(F,x);
    end
    return
>> fixiter(inline('2*sqrt(sin(x))'),2.5,1E-6)
ans = 1.93375360809067 (15 iterations)
```

Converges linearly if $|G'(x)| < 1$, but is of little practical interest.

