Noteworthy Numerical Algorithms

Walter Gander

gander@inf.ethz.ch

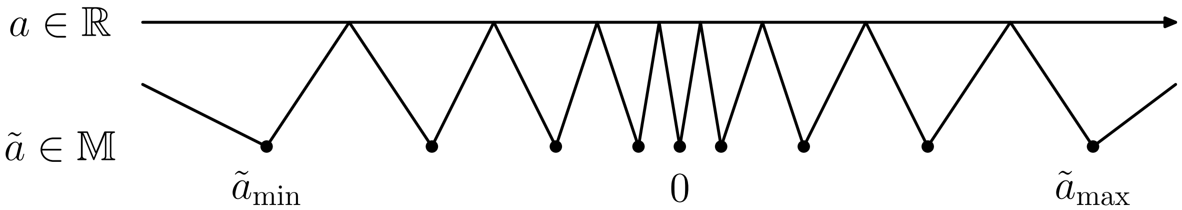
HKBU January 13, 2020

How a Computer Computes

- Mathematics: $\mathbb{R}$ = continuum
- Computer: Machine numbers $\mathbb{M}$ = finite set
  numbers with same leading digits are mapped on the same machine number, e.g. for a computer with 6 decimal digits

$$\begin{array}{ccc} \mathbb{R} & & \mathbb{M} \\ 3.14159265358979\ldots & & \\ 3.14159112233445\ldots & \mapsto & 3.14159 \\ 3.14159005674839\ldots & & \end{array}$$

- $\mathbb{R} \to \mathbb{M}$: an interval $\in \mathbb{R} \mapsto \tilde{a} \in \mathbb{M}$:



Floating Point Numbers

- number $= 8.881784197001252e{-15}$

  mantissa $= \quad 8.881784197001252$

  exponent $= \quad -15$

  $= 8.881784197001252 \times 10^{-15}$

  $= 0\underbrace{.00000000000000 8}_{\leftarrow 15 \text{ places}} 881784197001252$

  Shift decimal point by 15 places

Konrad Zuse

(1910–1995)

Computer Inventor

- $8.881784197001252e{+5} = 8.881784197001252 \times 10^{5}$

  $= 8\underbrace{88178}_{5 \rightarrow}.4197001252$

- Computer: not decimal but binary system

  IEEE Floating Point Standard (since 1985)

IEEE Floating Point Standard (since 1985)

- Representation of a machine number using 64 bits

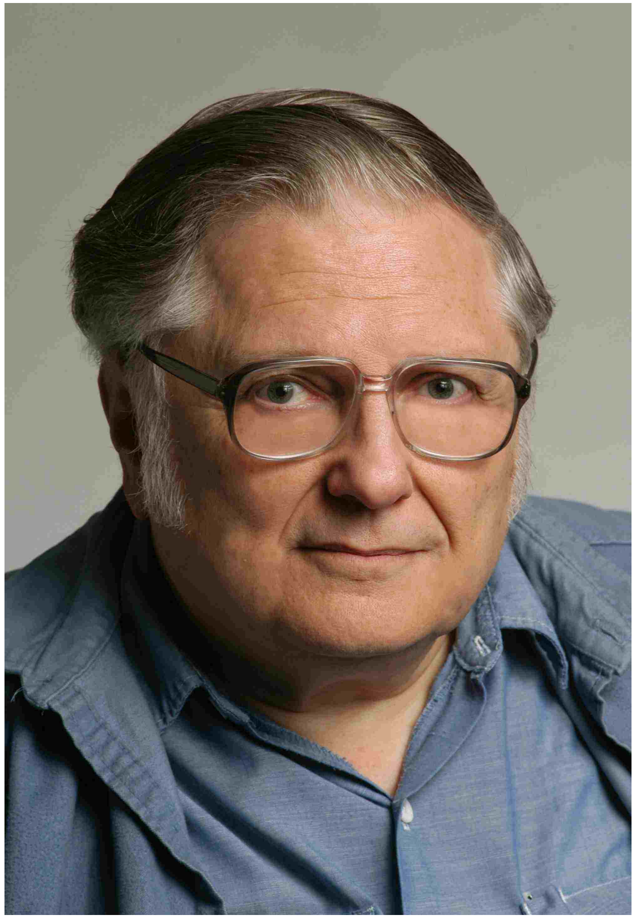
$$\underset{0}{S}\quad \overbrace{\underset{1}{E}EEEEEEEEE\underset{11}{E}}^{e}\quad \overbrace{\underset{12}{F}FFFF\cdots FFFF\underset{63}{F}}^{m}$$

S	1 bit for the sign
e	11 bits for the exponent
m	52 bits for the mantissa

- normal case: $0 < e < 2047, \quad (2^{11} = 2048)$

$$\tilde{a} = (-1)^S \times 2^{e-1023} \times 1.m$$

- spacing of the machine numbers in $[1, 2]$:
  eps=2.2204e-16 (machine precision)
- range of machine numbers
  $-1.7977 \cdot 10^{308} \leq x \leq 1.7977 \cdot 10^{308}$
  (number of hydrogen atoms in universe 10^{82})



William Kahan (*1933)
Father of IEEE Floating Point System

Finite Arithmetic: Rounding after each Operation

runden

Matlab statement			Results		
a	=	10	a	=	10
b	=	a/7	b	=	1.428571428571429
c	=	sqrt(sqrt(sqrt(sqrt(b))))	c	=	1.022542511383932
d	=	exp(16*log(c))	d	=	1.428571428571427
e	=	d*7	e	=	9.999999999999991
a-e			ans	=	8.881784197001252e-15

Rounding errors: For a basic operation $\otimes \in \{+, -, \times, /\}$ we have:

$$a \tilde{\otimes} b = (a \otimes b)(1+\eta), \quad |\eta| < \varepsilon$$

with $\varepsilon = 2.22 \cdot 10^{-16}$ (machine precision).

In principle, computers compute inaccurately!

The Challenge: Nevertheless, achieve correct results!

Computer Arithmetic is Different!

```
>> a=1;
>> b=1+eps;              % b  is the next machine number
>> c=(a+b)/2;            % there is no machine number between a and b
>> c-a                   % thus c has to be rounded
ans =
     0                   % c was rounded down to a
>> b-c
ans =
     2.2204-16
```

Hence we have two machine numbers $a < b$ but on the computer

$$\frac{a+b}{2} = a.$$

Mathematically this is a contradiction, since then $a + b = 2a$, thus $a = b$.

In mathematics,

$$a + x = a \implies x = 0.$$

Not true on a computer! x may be $\neq 0$

$$a \tilde{+} x = a \implies 4.9406e{-}324 \leq |x| < |a| \times eps \quad \text{or } x = 0$$

(eps is the machine precision)

```
>> a=20;
>> x=1e-15;          % x < a*eps = 4.4409e-15
>> b=a+x;
>> b-a
ans =
     0               % b = a but x is not zero
```

In mathematics, the associative law holds

$$(a+b)+c = a+(b+c)$$

not on the computer!

```
>> a=1e-7
a = 1.000000000000000e-07
>> b=5/7
b = 7.142857142857143e-01
>> c=-b+10*eps;
c = -7.142857142857121e-01
>> a+(b+c)
ans = 1.000000002220446e-07
>> (a+b)+c
ans = 1.000000021678105e-07
```

Results are different! Is one better?

To hell with the computer!?

To hell with the computer!?

it is useless for mathematics, since it does not calculate correctly

To hell with the computer!?

it is useless for mathematics, since it does not calculate correctly

No alternative — maybe we have to live with an imperfect machine ?

To hell with the computer!?

it is useless for mathematics, since it does not calculate correctly

No alternative — maybe we have to live with an imperfect machine ?

No!

we have to learn and make good use of computer arithmetic!

Correct Results in Spite of Rounding Errors

- Problem: compute the square root $x = \sqrt{a}$ using only the basic operations $\{+, -, \times, /\}$
- Method: Guess and correct. We want to find x such that $x^2 = a \iff \frac{a}{x} = x$
- Start with some initial value x_1, compute $\frac{a}{x_1}$

$$\text{if} \quad \frac{a}{x_1} \neq x_1 \quad \text{take the mean} \quad x_2 = \frac{1}{2}\left(x_1 + \frac{a}{x_1}\right)$$

- Iterate and obtain Heron's Method

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right), \quad k = 1, 2, \ldots$$

where $\{x_k\} \to \sqrt{a}$ for $k \to \infty$.

Alternative Derivation of Heron's Iteration

Solve $f(x) = x^2 - a = 0$ with Newton's method

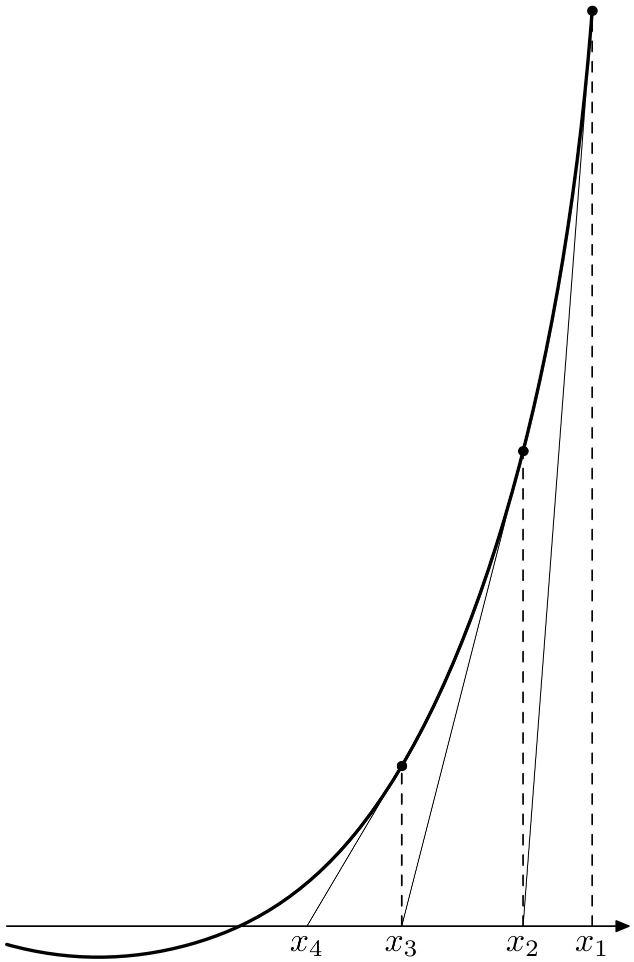
$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

$$f(x) = x^2 - a, \quad f'(x) = 2x$$

$$\Rightarrow x_{k+1} = x_k - \frac{x_k^2 - a}{2x_k}$$

$$= \frac{1}{2}\left(x_k + \frac{a}{x_k}\right)$$

Stopping criterion?



Stopping Criterion

- Traditional approach: stop if $|x_{k+1} - x_k| < \delta$
  not foolproof!
- we can do better: note that if $x_1 > \sqrt{a}$ then monotonous convergence: $\sqrt{a} < \cdots < x_2 < x_1$
  Monotonicity canot hold forever in finite arithmetic! Thus stop if $x_{k+1} \geq x_k$!
- How do we guarantee that $x_1 > \sqrt{a}$?
  Solution: start iteration with $x_0 = 1$
  Claim: $x_1 = (1 + a)/2 > \sqrt{a}$
  Proof: $x_1^2 - a = \frac{(a+1)^2}{4} - a = \frac{(a-1)^2}{4} > 0 \quad \Rightarrow \quad \sqrt{a} < x_1$

Square-Root Program with smart termination: stop iteration when monotonicity is violated!. Does not work in exact arithmetic! testSqrt

```
function x=mysqrt(a);
% computes w=sqrt(a) using Heron's algorithm
%
xold=(1+a)/2;              % xold > sqrt(a)
xnew=(xold+a/xold)/2;      % next iterate
while xnew<xold            % if monotone
  xold=xnew;               % iterate
  xnew=(xold+a/xold)/2;
end
x=xnew;
```

```
>> a= 12345.654321;
>> RelErr=(sqrt(a)-mysqrt(a))/sqrt(a)
RelErr = 1.2790e-16
```

Relative error is smaller than machine precision $\varepsilon = 2.22 \cdot 10^{-16}$

Same Algorithm in Maple

xmaple Wurzel.mw

works perfectly for different calculation precision

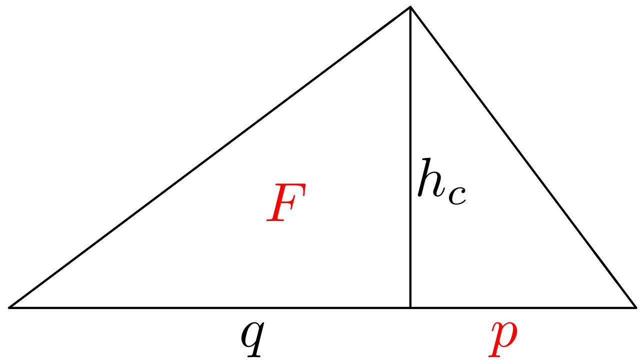
```
# Digits:=50;

MySqrt:= proc(a)
local xold, xnew;
xold:=(1.0+a)/2.0; xnew:=(xold+a/xold)/2.0;
while xnew<xold do
   xold:=xnew;
   xnew:=(xold+a/xold)/2.0;
end do;
xnew;
end proc:

MySqrt(Pi^2);
```

Solving a Nonlinear Equation

- Given a rectangular triangle Δ with area $F = 12$, segment $p = 2$. Compute the sides of the triangle.



- solution: $c = p + q$, $h_c^2 = pq$, replacing c and h_c in $F = \frac{c}{2}\, h_c$
- we get a nice equation

$$F = \frac{p+q}{2}\sqrt{pq}$$

- insert $F = 12$ and $p = 2 \implies$ equation for $x = q$

$$f(x) = \frac{2+x}{2}\sqrt{2x} - 12 = 0$$

- $f(2) = -8, \quad f(8) = 8 \implies 2 < x < 8$

Bisection Algorithm

- $f(a) < 0, \quad f(b) > 0 \implies a < x < b$
- try with $x = (a+b)/2$
  if $f(x) < 0$ move $a = x$ otherwise $b = x$
- iterate untill interval is small: $b - a < tol$
-
```
function x=bisectnaive(f,a,b,tol)
while b-a > tol
  x=(a+b)/2;
  if f(x) < 0, a=x; else b=x; end
end
```
- does not work for high precision ?? `dreieck`
  How about if $f(a) > 0$ and $f(b) < 0$?

Foolproof Bisection

dreieck2

```
function x=Bisection(f,a,b)
fa=f(a); v=1; if fa>0, v=-1; end;  % Determining the course of f
if fa*f(b)>0
  error('f(a) and f(b) have the same sign')
end
x=(a+b)/2;
while (a<x) & (x<b)                % as long as x in (a,b)
  if v*f(x)<0, a=x; else b=x; end; % continue to iterate
  x=(a+b)/2
end
```

we make use of the finite set of machine numbers!

Foolproof program.

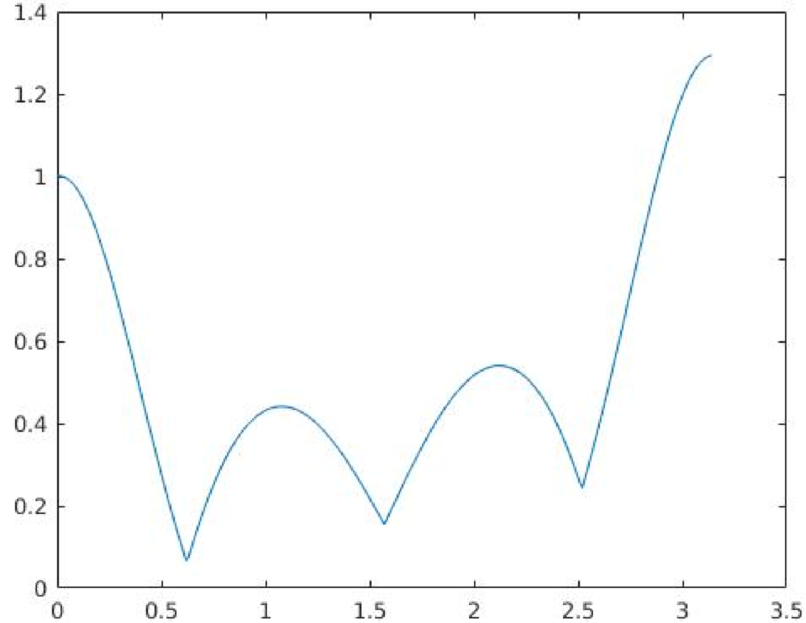
Does not work in exact arithmetic!

Minimization

Problem: compute the minima of

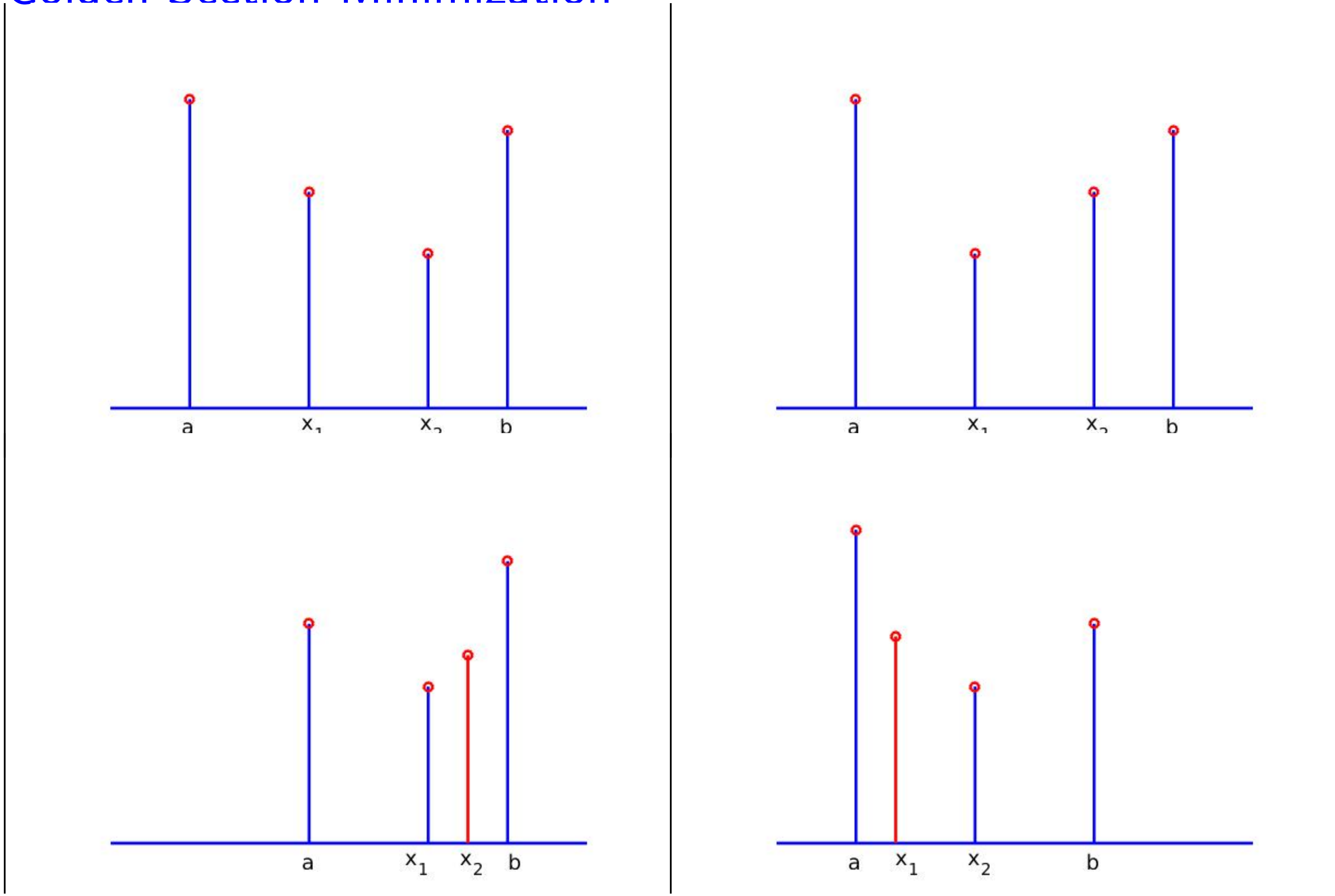
$$f(x) = 0.5\sin(x/5) + |\cos(\sin(2x) + x)|$$

in $(0, 3)$



We cannot simply solve $f'(x) = 0$

Golden Section Minimization



Choosing x_1 and x_2 so to Recycle Function Values

- Consider $x_1 = \lambda a + (1-\lambda)b$. Then x_1 is closer to a if $\lambda > 0.5$. Similarly $x_2 = (1-\lambda)a + \lambda b$ is closer to b
- If we shift b: $\implies x_2^{\text{new}} = x_1$ and $b^{\text{new}} = x_2$

$$
\begin{aligned}
x_1 &= \lambda a + (1-\lambda)b = x_2^{\text{new}} = (1-\lambda)a + \lambda b^{\text{new}} \\
\iff \lambda a + (1-\lambda)b &= (1-\lambda)a + \lambda((1-\lambda)a + \lambda b) \\
\iff 0 &= a - b - \lambda(a-b) - \lambda^2(a-b) \\
\iff \lambda^2 + \lambda - 1 = 0 &\iff \lambda_{1,2} = -\frac{1}{2} \pm \frac{\sqrt{5}}{2}
\end{aligned}
$$

- Choose $\lambda > 0 \implies \lambda = -\frac{1}{2} + \frac{\sqrt{5}}{2} = 0.618\ldots$
- Golden ratio!

$$
\frac{a}{a+b} = \frac{b}{a} \implies \lambda = \frac{b}{a} = -\frac{1}{2} + \frac{\sqrt{5}}{2}
$$

Foolproof Minimization notice the termination criterium

testMinimize

```
function x=Minimize(f,a,b)
fa=f(a); fb=f(b);
la=(-1+sqrt(5))/2;
x1=la*a+(1-la)*b; x2=(1-la)*a+la*b;
fx1=f(x1); fx2=f(x2);
while a<x1 & x1<x2 & x2<b
  if fx1>fx2
    a=x1;
    x1=x2; fx1=fx2;
    x2=(1-la)*a+la*b; fx2=f(x2);
  else
    b=x2;
    x2=x1; fx2=fx1;
    x1=la*a+(1-la)*b; fx1=f(x1);
  end;
end;
x=x1;
```

```
>> f=@(x) 0.5*sin(x/5) + abs(cos(sin(2*x)+x))
f =
  function_handle with value:
    @(x)0.5*sin(x/5)+abs(cos(sin(2*x)+x))

>> Minimize(f,0,1)
ans =
   0.623049193277906
>> Minimize(f,1,2)
ans =
   1.570796326794897
>> Minimize(f,2,3)
ans =
   2.518543460311887
```

Does not work in exact arithmetic!

Correct Program – Wrong Results!

Computing π as limit of surfaces of regular polygons in unit circle

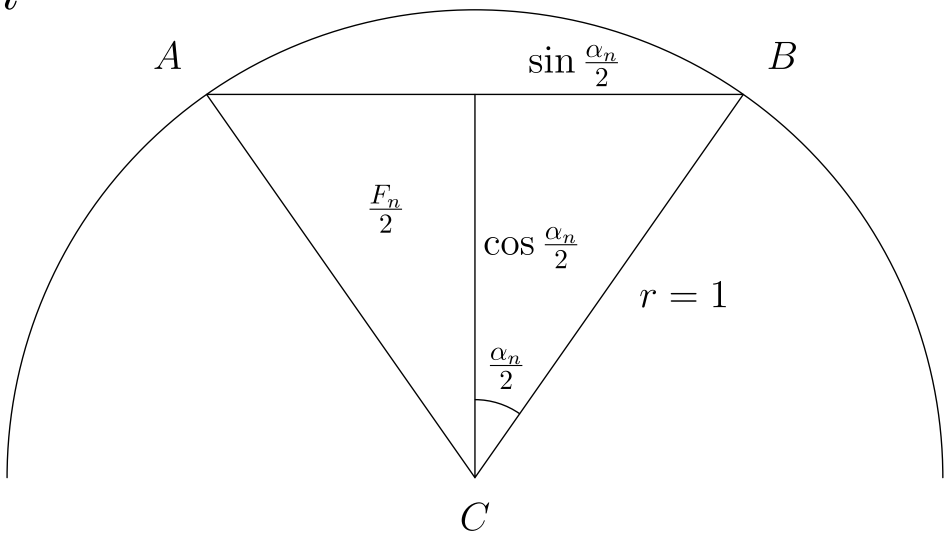
- A_n: area n-polygon, F_n: area ΔABC

$$A_n = n\, F_n = n\,\frac{\sin\alpha_n}{2}, \qquad \alpha_n = \frac{2\pi}{n}$$

- $\lim_{n\to\infty} A_n = \pi$
- $A_6 = \frac{3}{2}\sqrt{3} = 2.5981$
  $A_{12} = 3$
- Recursion $A_n \to A_{2n}$

$$\sin\left(\frac{\alpha}{2}\right) = \sqrt{\frac{1-\sqrt{1-\sin^2\alpha}}{2}}$$

Only two square-roots
and rational operations



Program

pinaive

```
s=sqrt(3)/2; A=3*s; n=6;               % initialization
z=[A-pi n A s];                        % store the results
while s>1e-10                          % termination if s=sin(alpha) small
  s=sqrt((1-sqrt(1-s*s))/2);           % new sin(alpha/2) value
  n=2*n; A=n*s/2;                      % A = new polygon area
  z=[z; A-pi n A s];
end
m=length(z);
for i=1:m
  fprintf('%10d %20.15f %20.15f %20.15f\n',z(i,2),z(i,3),z(i,1),z(i,4))
end
```

n	A_n	error	sin(alpha_n)
6	2.598076211353316	-0.543516442236477	0.866025403784439
12	3.000000000000000	-0.141592653589794	0.500000000000000
24	3.105828541230250	-0.035764112359543	0.258819045102521
48	3.132628613281237	-0.008964040308556	0.130526192220052
96	3.139350203046872	-0.002242450542921	0.065403129230143
192	3.141031950890530	-0.000560702699263	0.032719082821776
384	3.141452472285344	-0.000140181304449	0.016361731626486
768	3.141557607911622	-0.000035045678171	0.008181139603937
1536	3.141583892148936	-0.000008761440857	0.004090604026236
3072	3.141590463236762	-0.000002190353031	0.002045306291170
6144	3.141592106043048	-0.000000547546745	0.001022653680353
12288	3.141592516588155	-0.000000137001638	0.000511326906997
24576	3.141592618640789	-0.000000034949004	0.000255663461803
49152	3.141592645321216	-0.000000008268577	0.000127831731987
98304	3.141592645321216	-0.000000008268577	0.000063915865994
196608	3.141592645321216	-0.000000008268577	0.000031957932997
393216	3.141592645321216	-0.000000008268577	0.000015978966498
786432	3.141593669849427	0.000001016259634	0.000007989485855
1572864	3.141592303811738	-0.000000349778055	0.000003994741190
3145728	3.141608696224804	0.000016042635011	0.000001997381017
6291456	3.141586839655041	-0.000005813934752	0.000000998683561
12582912	3.141674265021758	0.000081611431964	0.000000499355676
25165824	3.141674265021758	0.000081611431964	0.000000249677838
50331648	3.143072740170040	0.001480086580246	0.000000124894489
100663296	3.159806164941135	0.018213511351342	0.000000062779708
201326592	3.181980515339464	0.040387861749671	0.000000031610136
402653184	3.354101966249685	0.212509312659892	0.000000016660005
805306368	4.242640687119286	1.101048033529493	0.000000010536712
1610612736	6.000000000000000	2.858407346410207	0.000000007450581
3221225472	0.000000000000000	-3.141592653589793	0.000000000000000

Cancellation

$$\begin{array}{r}
1.2345e0 \\
-1.2344e0 \\
\hline
0.0001e0 \quad = \quad 1.0000e{-4}
\end{array}$$

- if both numbers exact $\Longrightarrow$ result $1.0000e{-4}$ exact
- if both numbers affected by rounding errors from earlier calculations, then

$$\begin{array}{r}
1.2345e0 \\
-1.2344e0 \\
\hline
0.0001e0 \quad = \quad 1.xxxxe{-4} \quad \text{wrong!}
\end{array}$$

Rearrange the computation and avoid cancellation

$$\sin\frac{\alpha_n}{2} = \sqrt{\frac{1-\sqrt{1-\sin^2\alpha_n}}{2}} \quad \text{unstable recursion}$$

$$= \sqrt{\frac{1-\sqrt{1-\sin^2\alpha_n}}{2}\,\frac{1+\sqrt{1-\sin^2\alpha_n}}{1+\sqrt{1-\sin^2\alpha_n}}}$$

$$= \sqrt{\frac{1-(1-\sin^2\alpha_n)}{2(1+\sqrt{1-\sin^2\alpha_n})}}$$

$$\sin\frac{\alpha_n}{2} = \frac{\sin\alpha_n}{\sqrt{2(1+\sqrt{1-\sin^2\alpha_n})}} \quad \text{stable recursion}$$

Stable Computation of π

pistabil

```
oldA=0;s=sqrt(3)/2; newA=3*s; n=6;     % initialization
z=[newA-pi n newA s];                  % store the results
while newA>oldA                        % quit if area does not increase
   oldA=newA;
   s=s/sqrt(2*(1+sqrt((1+s)*(1-s)))); % new sin-value
   n=2*n; newA=n/2*s;
   z=[z; newA-pi n newA s];
end
m=length(z);
for i=1:m
  fprintf('%10d %20.15f %20.15f\n',z(i,2),z(i,3),z(i,1))
end
```

Note the elegant termination criterion!

Does not work in exact arithmetic, it makes use of finite arithmetic.

n	A_n	error
6	2.598076211353316	-0.543516442236477
12	3.000000000000000	-0.141592653589793
24	3.105828541230249	-0.035764112359544
48	3.132628613281238	-0.008964040308555
96	3.139350203046867	-0.002242450542926
192	3.141031950890509	-0.000560702699284
384	3.141452472285462	-0.000140181304332
768	3.141557607911857	-0.000035045677936
1536	3.141583892148318	-0.000008761441475
3072	3.141590463228050	-0.000002190361744
6144	3.141592105999271	-0.000000547590522
12288	3.141592516692156	-0.000000136897637
24576	3.141592619365383	-0.000000034224410
49152	3.141592645033690	-0.000000008556103
98304	3.141592651450766	-0.000000002139027
196608	3.141592653055036	-0.000000000534757
393216	3.141592653456104	-0.000000000133690
786432	3.141592653556371	-0.000000000033422
1572864	3.141592653581438	-0.000000000008355
3145728	3.141592653587705	-0.000000000002089
6291456	3.141592653589271	-0.000000000000522
12582912	3.141592653589663	-0.000000000000130
25165824	3.141592653589761	-0.000000000000032
50331648	3.141592653589786	-0.000000000000008
100663296	3.141592653589791	-0.000000000000002
201326592	3.141592653589794	0.000000000000000
402653184	3.141592653589794	0.000000000000001
805306368	3.141592653589794	0.000000000000001

Quadratic Equations

- Given $x^2 + px + q = 0$, compute solutions x_1 and x_2
- Should always work if p, q, x_1 and x_2 are machine numbers
- Standard formula

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}.$$

- Factorizing $x^2 + px + q = (x - x_1)(x - x_2)$
- naive program

```
function [x1,x2]=QuadEquation(p,q)
discriminant=(p/2)^2-q;
if discriminant<0
    error('Solutions are complex')
end
d=sqrt(discriminant);
x1=-p/2+d; x2=-p/2-d;
```

Test of QuadEquation

qnaive

- $(x-2)(x+3) = x^2 + x - 6 = 0$

```
>> [x1,x2]=QuadEquation(1,-6)
x1=2, x2=-3
```
correct

- $(x-10^9)(x+2\cdot 10^{-9}) = x^2 + (2\cdot 10^{-9} - 10^9)x + 2$

```
>> [x1,x2]=QuadEquation(2e-9-1e9,2)
 x1=1.0000e+09, x2=0
```
wrong

- $(x+10^{200})(x-1) = x^2 + (10^{200}-1)x - 10^{200}$

```
>> [x1,x2]=QuadEquation(1e200-1,-1e200)
x1=Inf, x2=-Inf
```
wrong

Better Algorithm for Computer

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}.$$

If $|p| > 1$ factor out (avoid overflow)

$$x_{1,2} = -\frac{p}{2} \pm |p|\sqrt{\frac{1}{4} - q/p/p}$$

Avoid cancellation by using Theorem of Vieta ($|x_1| \geq |x_2|$)

$$x_1 = -\text{sign}(p)\left(|p|/2 + |p|\sqrt{\frac{1}{4} - q/p/p}\right)$$

$$x_2 = q/x_1 \qquad \text{Vieta}$$

Foolproof Program for Quadratic Equations

qprofi

```
function [x1,x2]=quadeq(p,q)
if abs(p/2)>1                      % avoid overflow
  fak=abs(p); disc=0.25-q/p/p; % by factoring out
else
  fak=1; disc=(p/2)^2-q;
end
if disc<0
  error('solutions are complex')
else
  x1=abs(p/2)+fak*sqrt(disc);   % compute the larger solution (in modulus)
  if p>0, x1=-x1; end           % adjust sign
  if x1==0, x2=0;
  else
     x2=q/x1;                   % avoid cancellation using Vieta
  end                           % for second solution
end
```

Test of quadeq

- $(x-2)(x+3) = x^2 + x - 6 = 0$

  `>> [x1,x2]=quadeq(1,-6)      x1 =  2     x2 = -3`

  correct

- $(x-10^9)(x+2\cdot 10^{-9}) = x^2 + (2\cdot 10^{-9} - 10^9)x + 2$

  `>> [x1,x2]=quadeq(2e-9-1e9,2)  x1=1.0000e+09  x2=2.0000e-09`

  correct!

- $(x+10^{200})(x-1) = x^2 + (10^{200}-1)x - 10^{200}$

  `>> [x1,x2]=quadeq(1e200-1,-1e200)    x1=-1.0000e+200    x2=1`

  correct!

Exponential Function Problem: compute e^x using the 4 basic operations $\{+, -, \times, /\}$

Solution: use Taylor series (converges for every x):

$$e^x = \sum_{j=0}^{\infty} \frac{x^j}{j!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

```
function s=ExpUnstable(x,tol);
%
s=1; term=1; k=1;              % s partial sum
while abs(term)>tol*abs(s)   % iterate while next term is large
  so=s; term=term*x/k;         % new term
  s=so+term; k=k+1;            % new partial sum
end
```

Testresults: works fine for $x > -1$

testExpUnstable

```
>> x=20; [ExpUnstable(x,1e-14) exp(x)]
ans =
   1.0e+08 *
   4.851651954097881   4.851651954097902
>> x=1; [ExpUnstable(x,1e-14) exp(x)]
ans =
   2.718281828459046   2.718281828459046
>> x=-1; [ExpUnstable(x,1e-14) exp(x)]
ans =
   0.367879441171442   0.367879441171442
>> x=50; [ExpUnstable(x,1e-14) exp(x)]
ans =
   1.0e+21 *
   5.184705528587043   5.184705528587072
```

For negative x we get

```
>> x=-10; [ExpUnstable(x,1e-14) exp(x)]
ans =
   1.0e-04 *
   0.45399296230313   0.45399297624848
>> x=-20; [ExpUnstable(x,1e-14) exp(x)]
ans =
   1.0e-08 *
   0.56218844721304 2   0.20611536224385 6
>> x=-50; [ExpUnstable(x,1e-14) exp(x)]
ans =
   1.0e+04 *
   1.107293338289196   0.000000000000000
```

Why?

- For $x = -20$, the terms in the series

$$1 - \frac{20}{1!} + \frac{20^2}{2!} - \cdots + \frac{20^{20}}{20!} - \frac{20^{21}}{21!} + \cdots$$

  become large and have alternating signs.

- Largest terms (before they decline):

$$\frac{20^{19}}{19!} = \frac{20^{20}}{20!} = 4.3e7.$$

- Because of the growth of the terms, partial sums grow to about same size as largest term $\approx 10^7$.

- Partial sums should converge to $e^{-20} = 2.06e{-}9$

  only possible by cancellation! $\Longrightarrow$ errors

The Smart Program for e^x Does not work in exact arithmetic!

- $\frac{x^n}{n!} \to 0$ very fast as $n \to \infty$

  Termination criterion: let $s = \sum_{k=0}^{n-1} \frac{x^k}{k!}, \quad t = \frac{x^n}{n!}$

  if $s + t = s$ then stop summation.

- Avoid cancellation: use $e^{-x} = \frac{1}{e^x}$.

```
function s=Exp(x);
if x<0, v=-1; x=abs(x); else v=1; end  % make x>0
so=0; s=1; term=1; k=1;
while s~=so                             % sum till s+term=s
  so=s; term=term*x/k;
  s=so+term; k=k+1;
end
if v<0, s=1/s; end;                     % modify for x<0
```

Test of `Exp.m`

testExp

```
xx=[-50,-20,-10,-1,1,20]
for x=xx
 (Exp(x)-exp(x))/exp(x)
end
ans =
  -4.8757e-16
ans =
   2.0066e-16
ans =
   2.9851e-16
ans =
  -1.5089e-16
ans =
     0
ans =
  -1.2285e-16
```

Relative error is smaller than eps!

Adaptive Quadrature

Problem: compute numerically

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

Popular idea: I_1, I_2 2 approximations for I

If $|I_1 - I_2| < \epsilon |I_2| \quad \Rightarrow I = I_2$

else recursion $m = (a+b)/2$ (divide-and-conquer algorithm)

$$I = \int_a^m f(x)\,dx + \int_m^b f(x)\,dx.$$

compute both integrals independently.

Naive implementation

adapt3

- Test-Problem: $\int_0^1 \sqrt{x}\, dx$
- Methods: Simpson's rule $S(h)$, $I_1 = S(h)$, $I_2 = S(h/2)$
- Termination: $|I_1 - I_2| < \epsilon |I_2| \quad \Rightarrow I = I_2$ with $\epsilon = 10^{-4}$
- Results (depending on MATLAB Version):
  - Segmentation fault
  - ??? Maximum recursion limit of 500 reached
  - Out of memory. The likely cause is an infinite recursion within the program.
- Reason: for this example I_1 and I_2 never match to 4 digits!

$\Rightarrow$ **bad termination criterion**

Better Termination Criteria

adapt3 modified

- Idea: if $|I_1|$ is small, stop recursion: $|I_1| < \eta \left|\int_a^b f(x)dx\right|$
- Need therefore approximation `is` for the unknown integral
- For $\int_0^1 \sqrt{x}\, dx$ using the paramaters $\epsilon = \eta = 10^{-4}$, `is` $= 1$ and the criterion

```
if (abs(i1-i2) < epsilon*abs(i2)) | (abs(i1)<eta*is),
```

$\Longrightarrow I = 0.666617217$ (41 function evaluations)

- Problems: selection of
  - ϵ for $|I_1 - I_2| < \epsilon |I_2|$
  - η for $|I_1| < \eta$ `is`
  - `is` $\approx \left|\int_a^b f(x)dx\right|$

problem- and machine-dependent, wrong selection easily possible

Eliminate Parameters

- Estimate of integral with Simpson:

$$\texttt{is} = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

- `is` must be non-zero, therefore replace

$$\texttt{is} = |\texttt{is}| + b - a$$

- Criterion 1: $|I_2| < \eta \, \texttt{is}$

  better use: `if is + i2 == is` $\Longrightarrow$ η eliminated.

- Criterion 2: $|I_1 - I_2| < \epsilon |I_2|$ is too stringent.

  Better consider $|I_1 - I_2| < \epsilon$ `is`.

  Even better: `if (is + (i1-i2) == is)` $\Longrightarrow$ ϵ also eliminated

Termination Criterions

- Generally Criterion 2 is met before Criterion 1, thus terminate

```
if is + (i1-i2) == is
```

- For lower tolerance `tol`, replace

```
is = is*tol/eps
```

  where `eps` is the machine precision

Programs

Main program calls recursive function adapt

```
function I=Adapt(f,a,b,tol)
%
tol=tol/10;
if tol<eps,tol=eps; end;                % change unrealistic tol
fa=f(a); fb=f(b); fm=f((a+b)/2);
is=(b-a)/6*(fa+4*fm+fb);                % compute Simpson approximation
is=(abs(is)+b-a)*tol/eps;               % compute rough estimate for integral
I=adapt(f,a,b,fa,fm,fb,is);             % call recursive function
```

Recursive Function

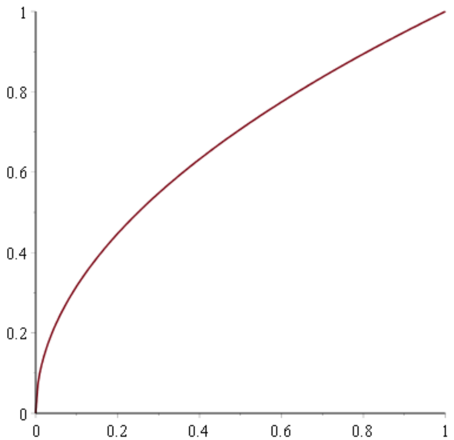
```
function I=adapt(f,a,b,fa,fm,fb,is)
%
m=(a+b)/2;h=(b-a)/4;
fml=f(a+h);fmr=f(b-h);
i1=h/1.5*(fa+4*fm+fb);                  % Simpson for h
i2=h/3*(fa+4*(fml+fmr)+2*fm+fb);        % Simpson for h/2
i1=(16*i2-i1)/15;                       % Romberg extrapolation
if is+(i1-i2)==is                       % Termination criterion
  I=i1; disp([a b-a I])
else
  I=adapt(f,a,m,fa,fml,fm,is) + adapt(f,m,b,fm,fmr,fb,is);
end
```

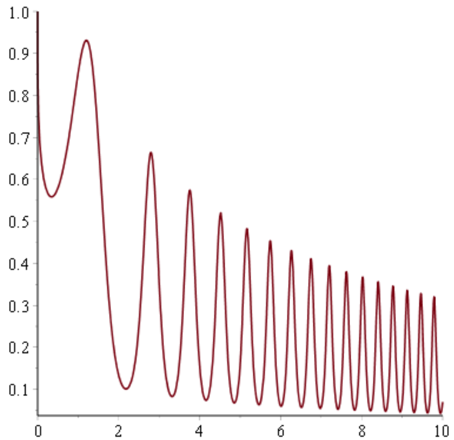
Does not work in exact arithmetic! Makes use of finite arithmetic.

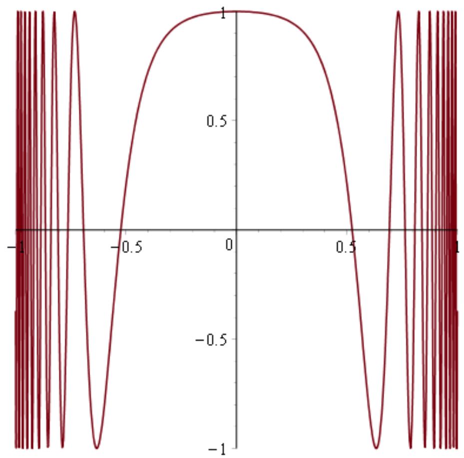
Test Results

testadapt

function	a	b	tol	# fct-eval	Adapt	exact
$\sqrt{x}$	0	1	10^{-4}	25	0.666616928507943	0.666666666666666
			10^{-12}	985	0.666666666666278	
$e^{\sin x^2} - \sqrt[3]{x}$	0	10	10^{-6}	629	2.966183232341963	2.966181555903316
			10^{-15}	35441	2.966181555903321	
$\cos\left(xe^{4x^2}\right)$	-1	1	10^{-6}	641	0.708263705656116	0.708263775050469
			10^{-15}	35065	0.708263775050469	







BIT
2000, Vol. 40, No. 1, pp. 084–101

0006-3835/00/4001-0084 $15.00

ADAPTIVE QUADRATURE—REVISITED *

WALTER GANDER and WALTER GAUTSCHI

Institut für Wissenschaftliches Rechnen, ETH, CH-8092 Zürich, Switzerland
email: gander@inf.ethz.ch, wxg@cs.purdue.edu

Dedicated to Cleve B. Moler on his 60th birthday

Abstract.

First, the basic principles of adaptive quadrature are reviewed. Adaptive quadrature programs being recursive by nature, the choice of a good termination criterion is given particular attention. Two Matlab quadrature programs are presented. The first is an implementation of the well-known adaptive recursive Simpson rule; the second is new and is based on a four-point Gauss–Lobatto formula and two successive Kronrod extensions. Comparative test results are described and attention is drawn to serious deficiencies in the adaptive routines `quad` and `quad8` provided by Matlab.

```
function [Q,fcnt] = quad(funfcn,a,b,tol,trace,varargin)
%QUAD   Numerically evaluate integral, adaptive Simpson quadrature.
%   Q = QUAD(FUN,A,B) tries to approximate the integral of scalar-valued
%   function FUN from A to B to within an error of 1.e-6 using recursive
%   adaptive Simpson quadrature. FUN is a function handle. The function
%   Y=FUN(X) should accept a vector argument X and return a vector result
%   Y, the integrand evaluated at each element of X.
%
%   Q = QUAD(FUN,A,B,TOL) uses an absolute error tolerance of TOL
%   instead of the default, which is 1.e-6.  Larger values of TOL
%   result in fewer function evaluations and faster computation,
%   but less accurate results.  The QUAD function in MATLAB 5.3 used
%   a less reliable algorithm and a default tolerance of 1.e-3.
%
%   Q = QUAD(FUN,A,B,TOL,TRACE) with non-zero TRACE shows the values
%   of [fcnt a b-a Q] during the recursion. Use [] as a placeholder to
%   obtain the default value of TOL.
%
%   [Q,FCNT] = QUAD(...) returns the number of function evaluations.
%
%   Use array operators .*, ./ and .^ in the definition of FUN
%   so that it can be evaluated with a vector argument.
%
%   QUAD will be removed in a future release. Use INTEGRAL instead.
%
%   Example:
%      Q = quad(@myfun,0,2);
%   where the file myfun.m defines the function:
%      %-------------------%
%      function y = myfun(x)
%      y = 1./(x.^3-2*x-5);
```

```
%     %-------------------%
%
%   or, use a parameter for the constant:
%     Q = quad(@(x)myfun2(x,5),0,2);
%   where the file myfun2.m defines the function:
%     %----------------------%
%     function y = myfun2(x,c)
%     y = 1./(x.^3-2*x-c);
%     %----------------------%
%
%   Class support for inputs A, B, and the output of FUN:
%     float: double, single
%
%   See also INTEGRAL, INTEGRAL2, INTEGRAL3, QUADGK, QUAD2D, TRAPZ,
%   FUNCTION_HANDLE.

%   Based on "adaptsim" by Walter Gander.
%   http://www.inf.ethz.ch/personal/gander

%   Reference:
%   [1] W. Gander and W. Gautschi, Adaptive Quadrature - Revisited,
%       BIT Vol. 40, No. 1, March 2000, pp. 84-101.

%   Copyright 1984-2013 The MathWorks, Inc.
```

Final Remarks

- Mathematical algorithms cannot be indiscriminately transferred to the computer. They need to be analysed and adapted to finite arithmetic.
- A great example is the Algol program `jacobi` by Heinz Rutishauser for computing the eigenvalues of a symmetric matrix, published in the Handbook and thorougly explained in Walter Gander, Martin J. Gander, Felix Kwok, *Scientific Computing, an Introduction Using* MAPLE *and* MATLAB. Springer Verlag, 2014.
- Even the popular algorithm for solving quadratic equations has to be modified as already George Forsythe noticed in 1966[a]

---

[a] George E. Forsythe, *How Do You Solve a Quadratic Equation?*. Stanford Technical Report No. CS40, June 16, 1966