2.29 Numerical Fluid Mechanics
Spring 2015

2.29 Numerical Fluid Mechanics
Spring 2015 – Lecture 2

REVIEW Lecture 1

1. Syllabus, Goals and Objectives
2. Introduction to CFD
3. From mathematical models to numerical simulations (1D Sphere in 1D flow)

   Continuum Model – Differential Equations

   => Difference Equations (often uses Taylor expansion and truncation)

   => Linear/Non-linear System of Equations

   => Numerical Solution (matrix inversion, eigenvalue problem, root finding, etc)
4. Error Types
   - Round-off error: due to representation by computers of numbers with a finite number of digits (significant digits) and to arithmetic operations (chopping or rounding)
   - Truncation error: due to approximation/truncation by numerical methods of "exact" mathematical operations/quantities
   - Other errors: model errors, data/parameter input errors, human errors.

2.29 Numerical Fluid Mechanics

REVIEW Lecture 1, Cont'd

- Approximation and round-off errors
  - Significant digits: Numbers that can be used with confidence
  - Absolute and relative errors $E_a = \hat{x} - \hat{x}_a, \quad \varepsilon_a = \dfrac{\hat{x} - \hat{x}_a}{\hat{x}_a}$
    - Iterative schemes and stop criterion: $|\varepsilon_a| = \left|\dfrac{\hat{x}_n - \hat{x}_{n-1}}{\hat{x}_n}\right| \le \varepsilon_s$
  - For n digits correct, base 10: $\varepsilon_s = \dfrac{1}{2} 10^{-n}$
  - Number representations
    - Integer representation
    - Floating-Point representation: $x = m\, b^e \qquad b^{-1} \le m < b^0$
    - Consequence of Floating Point Reals:
      - –Limited range (underflow & overflow)
      - –Limited precision (Quantizing errors)
      - –Relative error constant, absolute error growths with number

For t = significant digits with rounding: $\dfrac{|\Delta x|}{|x|} \le \dfrac{\varepsilon}{2} \qquad \varepsilon = b^{1-t} =$ Machine Epsilon

Numerical Fluid Mechanics – TODAY's Outline

- Approximation and round-off errors
  - Significant digits, true/absolute and relative errors
  - Number representations
  - Arithmetic operations
  - Errors of arithmetic/numerical operations
  - Examples: recursion algorithms (Heron, Horner's scheme) and other examples
    - Order of computations matter
    - Round-off error growth and (in)-stability
- Truncation Errors, Taylor Series and Error Analysis
  - Taylor series
  - Use of Taylor Series to derive finite difference schemes (first-order Euler scheme and forward, backward and centered differences)
  - Error propagation and error estimation:
    - Differential Formula and Standard Error (statistical formula)
  - Error cancellation
  - Condition numbers

Reference: Chapra and Canale, Chaps 3.1-3.4 and 4.1-4.4

Arithmetic Operations

$$r_1 \pm r_2 = m_1 b^{e_1} \pm m_2 b^{e_2}$$

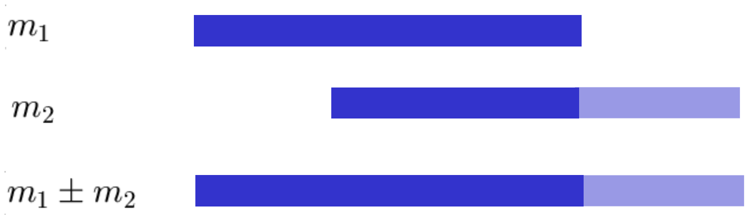
1. Addition and Subtraction

Shift mantissa of smallest number,

assuming $e_1 > e_2$,

Result has exponent of largest number:

$$r_1 \pm r_2 = \left(m_1 \pm m_2 b^{e_2 - e_1}\right) b^{e_1} = m b^{e1}$$



Absolute Error

$$\bar{\epsilon} \leq \bar{\epsilon_1} + \bar{\epsilon_2}$$

Relative Error

$$\bar{\alpha} = \frac{|\bar{m} - m|}{|m|}$$

Unbounded for
$m = m_1 \pm m_2 \rightarrow 0$

2. Multiplication and Division

$$r_1 \times r_2 = m_1 m_2 b^{e1+e2}$$

Multiplication:
Add exp, multiply mantissa, normalize and chop/round

$$m = m1m2 < 1$$

Division:
Subtract exp, divide mantissa, normalize and chop/round

$$0.1_2 \times 0.1_2 = 0.01_2$$

Relative Error

$$\bar{\alpha} \leq \bar{\alpha_1} + \bar{\alpha_2}$$

Bounded

Digital Arithmetics

Finite Mantissa Length

radd.m

Limited precision addition in MATLAB

```
function c = radd(a,b,n)
%
% function c = radd(a,b,n)
%
% Adds two real numbers a and b simulating an arithmetic unit with
% n significant digits, and rounding-off (not chopping-off) of numbers.
% If the inputs a and b provided do not have n digits, they are first
% rounded to n digits before being added.

%--- First determine signs
sa=sign(a);
sb=sign(b);

%--- Determine the largest number (exponent)
if (sa == 0)
    la=-200; %this makes sure that if sa==0, even if b is very small, it will have the largest exponent
else
la=ceil(log10(sa*a*(1+10^(-(n+1))))); %This determines the exponent on the base. Ceiling is used
                                     %since 0<log10(mantissa_base10)<=-1. The 10^etc. term just
                                     %properly increases the exponent estimated by 1 in the case
                                     %of a perfect log: i.e. log10(m b^e) is an integer,
                                     %mantissa is 0.1, hence log10(m)=-1, and
                                     %ceil(log10(m b^e(1+10^-(n+1))) ~< ceil(e +log10(m)+log10(1+10^-(n+1)))=e.
end
if (sb == 0)
    lb=-200;
else
    lb=ceil(log10(sb*b*(1+10^(-(n+1)))));
end
    lm=max(la,lb);
```

radd.m, continued

```
%--- Shift the two numbers magnitude to obtain two integers with n digits
f=10^(n);   %this is used in conjunction with the round function below
at=sa*round(f*sa*a/10^lm);  %sa*a/10^lm shifts the decimal point such that the number starts with 0.something
                            %the f*(*) then raises the number to a power 10^n, to get the desired accuracy
                            %of n digits above the decimal. After rounding to an integer, any figures that
                            %remain below are wiped out.
bt=sb*round(f*sb*b/10^lm);
% Check to see if another digit was added by the round. If yes, increase
% la (lb) and reset lm, at and bt.
ireset=0;
if ((at~=0) & (log10(at)>=n))
    la=la+1; ireset=1;
end
if ((bt~=0) & (log10(bt)>=n))
    lb=lb+1; ireset=1;
end
if (ireset)
    lm=max(la,lb);
    at=sa*round(f*sa*a/10^lm);
    bt=sb*round(f*sb*b/10^lm);
end
ct=at+bt;   %adds the two numbers
sc=sign(ct);

%The following accounts for the case when another digit is added when
%summing two numbers... ie. if the number of digits desired is only 3,
%then 999 +3 = 1002, but to keep only 3 digits, the 2 needs to be wiped out.
if (sc ~= 0)
    if (log10(sc*ct) >= n)
        ct=round(ct/10)*10;
%           'ct'
    end
end

%-----This basically reverses the operation on line 34,38
%  (it brings back the final number to its true magnitude)
c=ct*10^lm/f;
```

Matlab additions and quantizing effect

EXAMPLES

radd (100,4.9,1) = 100

radd (100,4.9,2) = 100

radd (100,4.9,3) = 105

```
>> radd (99.9,4.9,1)= 100
>> radd (99.9,4.9,2)= 100
>> radd (99.9,4.9,3) = 105
```

NOTE: Quantizing effect peculiarities

```
>> radd (0.095,-0.03,1) =0.06
>> radd (0.95,-0.3,1)= 1
```

Difference come from MATLAB round:

```
>> round(10^1*0.095/10^(-1))
     9
>> round(10^1*0.95/10^(0))
    10
```

But note:

```
>> round(10^1*(0.095/10^(-1)))
    10
```

Issues due to Digital Arithmetic

- Large number of additions/subtractions (recursion), e.g.
  - add 1 100,000 times vs.
  - add 0.00001 100,000 times.
- Adding large and small numbers (start from small to large)
- Subtractive cancellation
  - Round-off errors induced when subtracting nearly equal numbers, e.g. roots of polynomials
- Smearing: occurs when terms in sum are larger than the sum
  - e.g. series of mixed/alternating signs
- Inner products: very common computation, but prone to round-off errors
- Some examples of the above provided in following slides

Recursion: Heron's Device

Numerically evaluate square-root

$$\sqrt{s}\,, \quad s > 0$$

Initial guess x_0

$$x_0 \simeq \sqrt{s}$$

Test

$$x_0^2 < s \;\Rightarrow\; x_0 < \sqrt{s} \;\Rightarrow\; \frac{s}{x_0} > \sqrt{s}$$

$$x_0^2 > s \;\Rightarrow\; x_0 > \sqrt{s} \;\Rightarrow\; \frac{s}{x_0} < \sqrt{s}$$

Mean of guess and its reciprocal

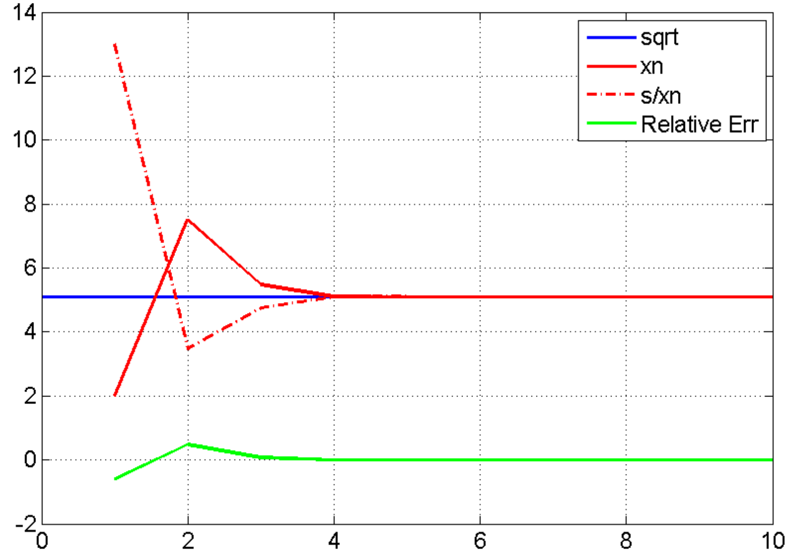
$$x_1 = \frac{1}{2}\left(x_0 + \frac{s}{x_0}\right)$$

Recursion Algorithm

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{s}{x_n}\right)$$

```
a=26;    %Number for which the sqrt is to be computed
n=10;    %Number of iteration in recursion
g=2;     %Initial guess
% Number of Digits
dig=5;
     sq(1)=g;
     for i=2:n
      sq(i)= 0.5*radd(sq(i-1),a/sq(i-1),dig);
     end
     '      i       value    '
     [ [1:n]' sq']
     hold off
     plot([0 n],[sqrt(a) sqrt(a)],'b')
     hold on
     plot(sq,'r')
     plot(a./sq,'r-.')
     plot((sq-sqrt(a))/sqrt(a),'g')
     legend('sqrt','xn','s/xn','Relative Err')
     grid on
```

MATLAB script
heron.m



Recursion: Horner's scheme to evaluate polynomials by recursive additions

Goal: Evaluate polynomial

$$\begin{aligned} p(z) &= a_0 z^3 + a_1 z^2 + a_2 z + a_3 \\ &= ((a_0 z + a_1) z + a_2)\, z + a_3 \end{aligned}$$

Horner's Scheme

$$\begin{array}{rcccc} & a_0 & a_1 & a_2 & a_3 \\ + & & z b_0 & z b_1 & z b_2 \\ \hline & b_0 & b_1 & b_2 & b_3 \end{array}$$

$(b_0 = a_0)$

$$p(z) = b_3$$

General order n

$$p(z) = a_0 z^n + a_1 z^{n-1} + \cdots + a_{n-1} z + a_n$$

Recurrence relation

$$b_0 = a_0 \ , \quad b_i = a_i + z b_{i-1} \ , \quad i = 1, \ldots n$$

$$\boxed{p(z) \ = \ b_n}$$

horner.m

```
% Horner's scheme
% for evaluating polynomials
a=[ 1 2 3 4 5 6 7 8 9 10 ];
n=length(a) -1 ;
z=1;
b=a(1);
% Note index shift for a
for i=1:n
    b=a(i+1)+ z*b;
end
p=b
```

```
>> horner

p =

    55
```

For home suggestion: utilize radd.m for all additions above and compare the error of Horner's scheme to that of a brute force summation, for both z negative/positive

Recursion: Order of Operations Matter

Tends to: 0 1

$$y = f(x) = \sum_{n=1}^{\infty} [x^n + b\sin[\pi/2 - \pi/10n] - c\cos[pi/(10(n+1))]]$$

If $x = 0.5\ ,\quad b = 0\ ,\quad c = 0 \Rightarrow y = 1.0$

recur.m

```
N=20; sum=0; sumr=0;
b=1; c=1; x=0.5;
xn=1;
% Number of significant digits in computations
dig=2;
ndiv=10;
for i=1:N
  a1=sin(pi/2-pi/(ndiv*i));
  a2=-cos(pi/(ndiv*(i+1)));
% Full matlab precision
  xn=xn*x;
  addr=xn+b*a1;
  addr=addr+c*a2;
  ar(i)=addr;
  sumr=sumr+addr;
  z(i)=sumr;
% additions with dig significant digits
  add=radd(xn,b*a1,dig);
  add=radd(add,c*a2,dig);
% add=radd(b*a1,c*a2,dig);
% add=radd(add,xn,dig);
  a(i)=add;
  sum=radd(sum,add,dig);
  y(i)=sum;
end
sumr
```

Result of small, but significant term ‘destroyed’ by subsequent addition and subtraction of almost equal, large numbers.

Remedy:
Change order of additions

recur.m Contd.

```
'      i        delta      Sum     delta(approx)  Sum(approx)'
res=[[1:1:N]' ar' z' a' y']

hold off
a=plot(y,'b'); set(a,'LineWidth',2);
hold on
a=plot(z,'r'); set(a,'LineWidth',2);
a=plot(abs(z-y)./z,'g'); set(a,'LineWidth',2);
legend([ num2str(dig) ' digits'],'Exact','Error');
```

recur.m

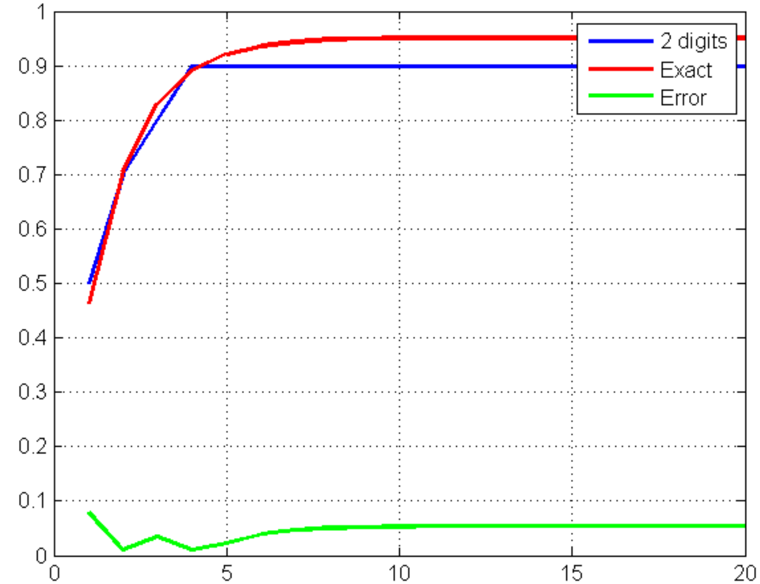
```
>> recur

b = 1; c = 1; x = 0.5;
dig=2

      i        delta       Sum     delta(approx) Sum(approx)

..\codes_2\recur.png
res =

    1.0000    0.4634    0.4634    0.5000    0.5000
    2.0000    0.2432    0.7065    0.2000    0.7000
    3.0000    0.1226    0.8291    0.1000    0.8000
    4.0000    0.0614    0.8905    0.1000    0.9000
    5.0000    0.0306    0.9212         0    0.9000
    6.0000    0.0153    0.9364         0    0.9000
    7.0000    0.0076    0.9440         0    0.9000
    8.0000    0.0037    0.9478         0    0.9000
    9.0000    0.0018    0.9496         0    0.9000
   10.0000    0.0009    0.9505         0    0.9000
   11.0000    0.0004    0.9509         0    0.9000
   12.0000    0.0002    0.9511         0    0.9000
   13.0000    0.0001    0.9512         0    0.9000
   14.0000    0.0000    0.9512         0    0.9000
   15.0000    0.0000    0.9512         0    0.9000
   16.0000   -0.0000    0.9512         0    0.9000
   17.0000   -0.0000    0.9512         0    0.9000
   18.0000   -0.0000    0.9512         0    0.9000
   19.0000   -0.0000    0.9512         0    0.9000
   20.0000   -0.0000    0.9512         0    0.9000
```



Order of Recurrence - Error Propagation
Numerical Instability Example

Evaluate Integral

$$y_n = \int_0^1 \frac{x^n}{x+5} dx \ , n = 0, 2 \ldots \infty$$

Recurrence Relation: $y_n = \frac{1}{n} - 5y_{n-1}$

Proof :

$$y_n + 5y_{n-1} = \int_0^1 \frac{x^n + 5x^{n-1}}{x+5} dx = \int_0^1 \frac{x^{n-1}(x+5)}{x+5} dx = \int_0^1 x^{n-1} dx = \frac{1}{n}$$

3-digit Recurrence:

$$\begin{aligned}
y_0 &= \int_0^1 \frac{dx}{x+5} = [\log_e(x+5)]_0^1 = \log_e 6 - \log_e 5 = 0.182 \\
y1 &= 1 - 5y_0 = 1 - 0.910 \simeq 0.0090 \\
y2 &= 0.5 - 5y_1 \simeq 0.050 \\
y3 &= 0.333 - 5y_2 \simeq 0.083 \\
y4 &= 0.25 - 5y_3 \simeq -0.165
\end{aligned}$$

> y2 !!

< 0 !!

Backward Recurrence

$$y_{n-1} = \frac{1}{5n} - \frac{y_n}{5}$$

$$\begin{aligned}
y_{10} &\simeq y_9 \Rightarrow y_9 + 5y_9 = 0.1 \Rightarrow y_9 = 0.017 \\
y_8 &= 1/45 - y_9/5 = 0.019 \\
y_7 &= 1/40 - y_8/5 = 0.021 \\
y_6 &= 0.025 \\
&\cdot \\
&\cdot \\
&\cdot \\
y_1 &= 0.088 \\
y_0 &= 0.182
\end{aligned}$$

Correct

Exercise: Make MATLAB script

Order of Recurrence - Error Propagation

ps: Bessel functions are only used as example, no need to know everything about them for this class.

Spherical Bessel Functions

Differential Equation

$$x^2\frac{d^2y}{dx^2}+2x\frac{dy}{dx}(x^2-n(n+1))y=0$$

Solutions

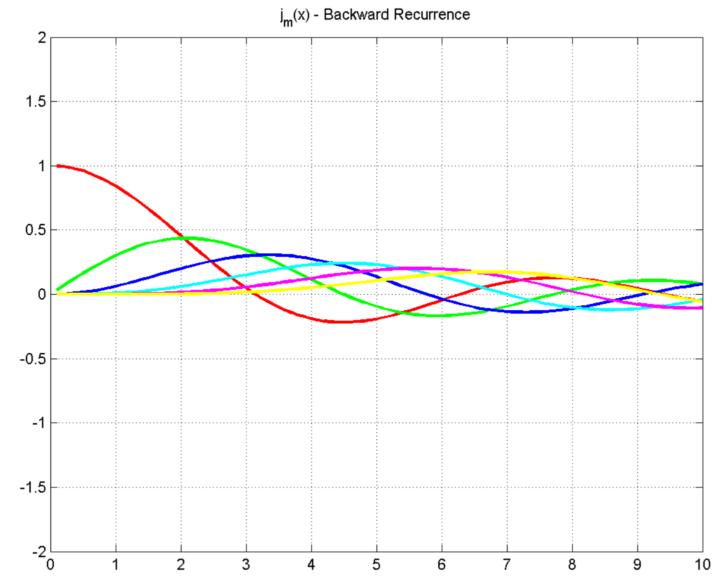
$$j_n(x)y_n(x)$$

n	$j_n(x)$	$y_n(x)$
0	$\frac{\sin x}{x}$	$-\frac{\cos x}{x}$
1	$\frac{\sin x}{x^2}-\frac{\cos x}{x}$	$-\frac{\cos x}{x^2}-\frac{\sin x}{x}$

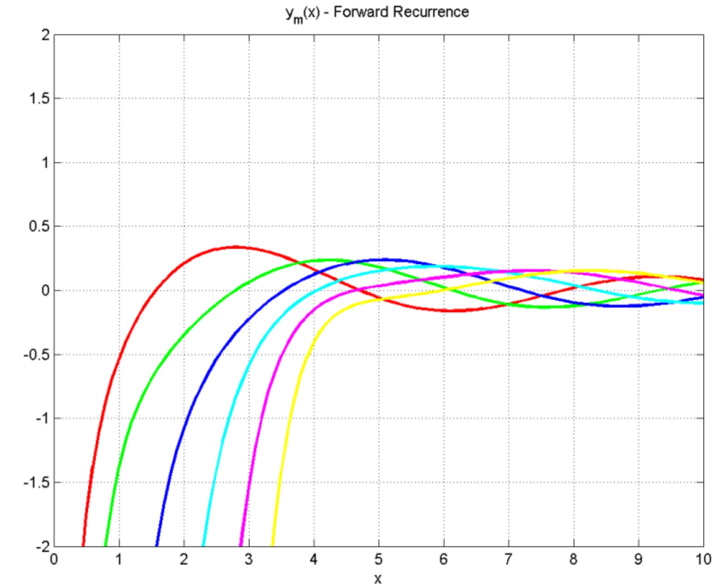
Bessel fct. of 1st kind
$$j_n(x)\rightarrow 0\begin{cases} n\rightarrow\infty \\ x\rightarrow 0\end{cases}$$

Bessel fct. of 2nd kind
$$y_n(x)\rightarrow -\infty\begin{cases} n\rightarrow\infty \\ x\rightarrow 0\end{cases}$$

$j_n(x)$



$y_n(x)$



Order of Recurrence - Error Propagation

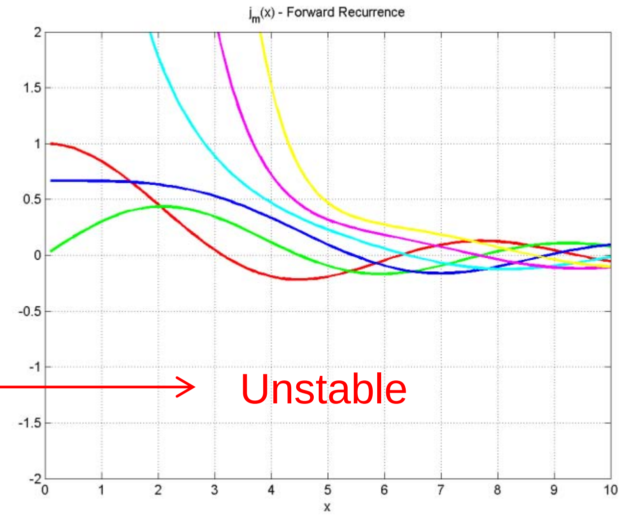
Spherical Bessel Functions

Forward Recurrence

$$j_{n+1}(x) = \frac{2n+1}{x} j_n(x) - j_{n-1}(x)$$

Forward Recurrence

$$\frac{2n+1}{x} j_n(x) \simeq j_{n-1}(x)$$



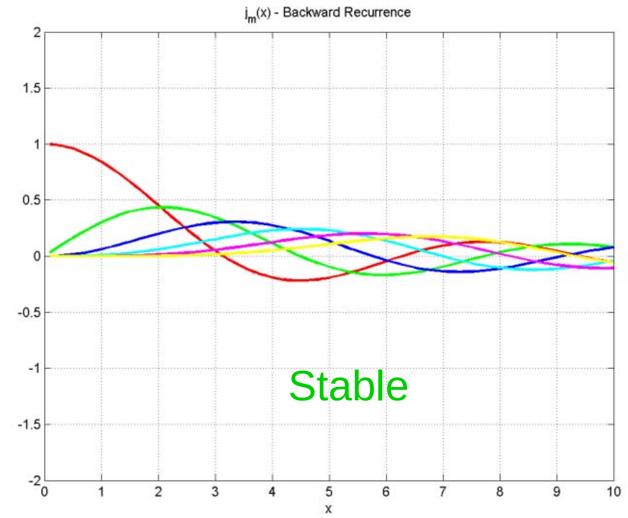
Backward Recurrence

$$j_{n-1}(x) = \frac{2n+1}{x} j_n(x) - j_{n+1}(x)$$

Miller's algorithm

$$j_N(x) = 1 \; , \; j_{N+1}(x) = 0 \; , \; j_0(x) = \frac{\sin x}{x}$$

with $N \sim x+20$



Error Propagation: Round-off and Truncation Errors

Differential Equation

$$\frac{dy}{dx} = f(x,y)\ ,\ y_0 = p$$

Example

$$f(x,y) = x\ (y = x^2/2 + p)$$

Discretization

$$x_n = nh$$

Finite Difference (forward)

$$\frac{dy}{dx}\Big|_{x=x_n} \simeq \frac{y_{n+1} - y_n}{h}$$

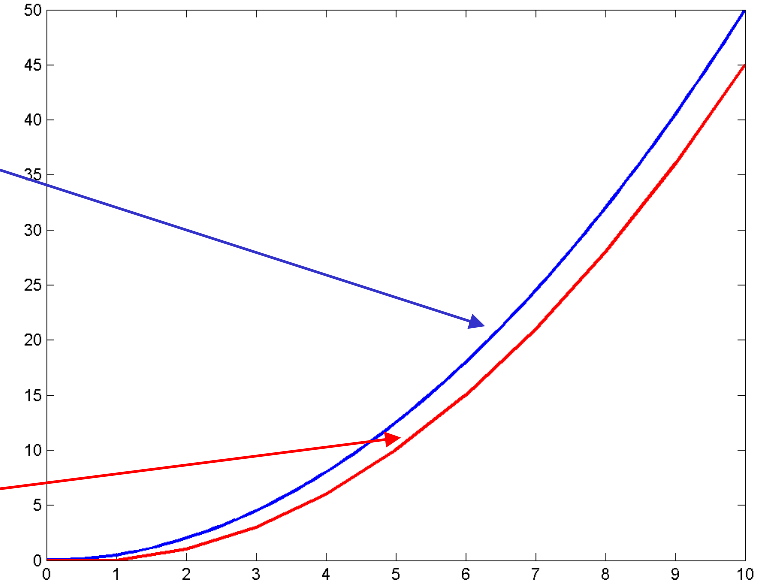
Recurrence

$$y_{n+1} = y_n + hf(nh, y)$$

Central Finite Difference

$$\frac{dy}{dx}\Big|_{x=x_n} \simeq \frac{y_{n+1} - y_{n-1}}{2h}$$

Euler's Method



euler.m

Truncation Errors, Taylor Series and Error Analysis

Taylor Series:

- Provides a mean to predict a function at one point in terms of its values and derivatives at another point (in the form of a polynomial)
- Hence, any smooth functions can be approximated by a polynomial
- Taylor Series (Mean for integrals theorems):

$$f(x_{i+1}) = f(x_i) + \Delta x\, f'(x_i) + \frac{\Delta x^2}{2!} f''(x_i) + \frac{\Delta x^3}{3!} f'''(x_i) + \ldots + \frac{\Delta x^n}{n!} f^n(x_i) + R_n$$

$$R_n = \frac{\Delta x^{n+1}}{n+1!} f^{(n+1)}(\xi) = \int_{x_i}^{x_{i+1}} \frac{(x_{i+1}-t)^n}{n!} f^{n+1}(t)dt$$

- = constant + line + parabola + etc

Taylor Series to Derive Finite Difference Schemes

Taylor series, Δx constant:

$$f(x_{i+1}) = f(x_i) + \Delta x\, f'(x_i) + \frac{\Delta x^2}{2!} f''(x_i) + \frac{\Delta x^3}{3!} f'''(x_i) + \ldots + \frac{\Delta x^n}{n!} f^n(x_i) + R_n$$

$$R_n = \frac{\Delta x^{n+1}}{n+1!} f^{(n+1)}(\xi)$$

- Forward finite-difference estimate of $f'(x_i)$ with 1st order accuracy

$$\left. f(x_{i+1}) = f(x_i) + \Delta x\, f'(x_i) + \frac{\Delta x^2}{2!} f''(x_i) + O(\Delta x^3) \right\} \longrightarrow f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{\Delta x} - \underline{\underline{\frac{\Delta x}{2!} f''(x_i) - O(\Delta x^2)}}$$

- Centered finite-difference estimate of $f'(x_i)$ with 2nd order accuracy

$$\left.\begin{aligned} &\text{Forward} \quad f(x_{i+1}) = f(x_i) + \Delta x\, f'(x_i) + \frac{\Delta x^2}{2!} f''(x_i) + \frac{\Delta x^3}{3!} f'''(x_i) + O(\Delta x^4) + O(\Delta x^5) \\ &\text{Backward} \quad f(x_{i-1}) = f(x_i) - \Delta x\, f'(x_i) + \frac{\Delta x^2}{2!} f''(x_i) - \frac{\Delta x^3}{3!} f'''(x_i) + O(\Delta x^4) - O(\Delta x^5) \end{aligned}\right\}$$

$$\longrightarrow \quad f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2\Delta x} - \underline{\underline{\frac{\Delta x^2}{3!} f'''(x_i) - O(\Delta x^4)}}$$

- Order p of accuracy indicates how fast the error is reduced when the grid is refined (not the magnitude of the error)

Derivation of General "Differential" Error Propagation Formula

Univariate Case $y = f(x)$

Recall: $$f(x_{i+1}) = f(x_i) + \Delta x\, f'(x_i) + \frac{\Delta x^2}{2!} f''(x_i) + \frac{\Delta x^3}{3!} f'''(x_i) + \ldots + \frac{\Delta x^n}{n!} f^n(x_i) + R_n$$

$$R_n = \frac{\Delta x^{n+1}}{n+1!} f^{(n+1)}(\xi) = O(\Delta x^{n+1})$$

Hence, $$\Delta y \doteq \Delta f = f(x_{i+1}) - f(x_i)$$

$$= \Delta x\, f'(x_i) + \frac{\Delta x^2}{2!} f''(x_i) + \frac{\Delta x^3}{3!} f'''(x_i) + \ldots + \frac{\Delta x^n}{n!} f^n(x_i) + R_n$$

$$\text{For } \Delta x \ll 1, \quad \Delta f = \Delta x\, f'(x_i) + O(\Delta x^2) \simeq \Delta x\, f'(x_i)$$

Thus, for an error on x equal to Δx such that $|\Delta x| \doteq \varepsilon \ll 1$, we have an error on y equal to :

$$\boxed{\varepsilon_y = |\Delta y| = |\Delta f| \simeq |\Delta x\, f'(x_i)| = |\Delta x|\, |f'(x_i)| = \varepsilon |f'(x_i)|}$$

Multivariate case

$$y = f(x_1, x_2, x_3, \ldots, x_n)$$

Derivation done in class on the board

$$\boxed{\text{For } |\Delta x_i| \ll 1, \quad \varepsilon_y \leq \sum_{i=1}^{n} \left| \frac{\partial f(x_1, \ldots, x_n)}{\partial x_i} \right| \varepsilon_i}$$

General Error Propagation Formula (The Differential Formula)

Note: not to scale. For this large plotted Δx, second derivative not negligible

$$y = f(x_1, x_2, \ldots x_n)$$

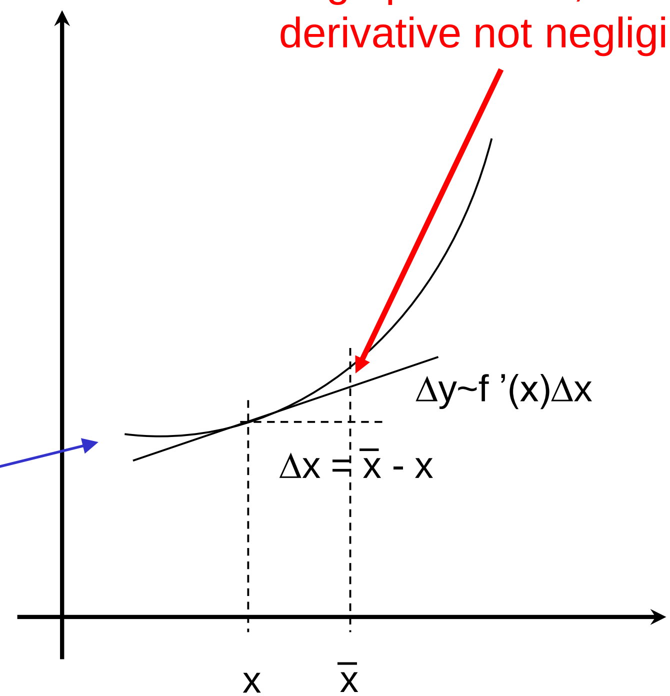
Absolute Errors

$$\epsilon_1, \epsilon_2, \ldots \epsilon_n$$

$$\epsilon_y\,?$$

Function of one variable

$$y = f(x) \quad \bar{y} = f(\bar{x})$$ => Δy~f ’(x)Δx



Fct. of *n* var., General Error Propagation Formula

$$\Delta y \;\simeq\; \sum_{i=1}^{n} \frac{\partial f(x_1, \ldots, x_n)}{\partial x_i} \Delta x_i$$

$$\boxed{\epsilon_y \;\leq\; \sum_{i=1}^{n} \left| \frac{\partial f(x_1, \ldots, x_n)}{\partial x_i} \right| |\epsilon_i|}$$

Error Propagation Example with Differential Approach: Multiplications

Multiplication

$$y = x_1 x_2$$

$$\Rightarrow \quad \log y = \log x_1 + \log x_2$$

$$\Rightarrow \quad \frac{1}{y}\frac{\partial y}{\partial x_i} = \frac{1}{x_i}$$

$$\Rightarrow \quad \frac{\partial y}{\partial x_i} = \frac{y}{x_i}$$

=>

Error Propagation Formula

$$\left|\frac{\Delta y}{y}\right| \le \sum_{i=1}^{2} \left|\frac{\Delta x_i}{x_i}\right|$$

$$\varepsilon^r{}_y \le \sum_{i=1}^{2} \varepsilon^r{}_i$$

Relative Errors Add for Multiplication

Another example, more general case: $y = x_1^{m_1} x_2^{m_2} \cdots x_n^{m_n}$

$$\boxed{\varepsilon^r{}_y \le \sum_{i=1}^{n} |m_i| \, \varepsilon^r{}_i}$$