2.29 Numerical Fluid Mechanics  
Fall 2011 – 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)
   • 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
  \[ E_a = \hat{x}_a - \hat{x}, \quad \varepsilon_a = \frac{\hat{x}_a - \hat{x}}{\hat{x}_a} \]

– Iterative schemes and stop criterion:
  \[ |\varepsilon_a| = \left| \frac{\hat{x}_n - \hat{x}_{n-1}}{\hat{x}_n} \right| \leq \varepsilon_s \]

– For n digits:
  \[ \varepsilon_s = \frac{1}{2} \times 10^{-n} \]
Numerical Fluid Mechanics – TODAY’s Outline

• Approximation and round-off errors
  – Absolute and relative errors
  – Number representations
  – Arithmetic operations
  – Errors of arithmetic/numerical operations
  – Recursion algorithms (Heron, Horner’s scheme, etc):
    • Order of computations matter

• 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 – numerical stability
  – Error estimation
  – Error cancellation
  – Condition numbers

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

• Number Systems:
  – Base-10: \(1,234_{10} = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0\)
  – Computers (0/1): base-2 \(1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0\) = \(13_{10}\)

• Integer Representation (signed magnitude method):
  – First bit is the sign (0,1), remaining bits used to store the number
  – For a 16-bits computer:
    • Example: \(-13_{10} = 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 1\)
    • Largest range of numbers: \(2^{15}-1\) largest number => \(-32,768\) to \(32,767\) (from 0 to \(1111111111111111\))

• Floating Number Representation

\[x = m \ b^e\]

\(x\) = Number
\(m\) = Mantissa/Significand
\(b\) = Base
\(e\) = Exponent

Sign

Signed
Exponent

Mantissa
Floating Number Representation

**Examples**

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00527 = 0.527_{10} \times 10^{-2_{10}}</td>
<td>10.1_2 = 0.101_2 \times 2^{2_{10}} = 0.101_2 \times 2^{10_2}</td>
</tr>
</tbody>
</table>

**Convention: Normalization of Mantissa m** (so as to have no zeros on the left)

- 0.01234 \Rightarrow 0.1234 \times 10^{-1}
- 12.34 \Rightarrow 0.1234 \times 10^2

**Decimal**

- $0.1 \leq m < 1.0$

**Binary**

- $0.1_2 = 0.5_{10} \leq m < 1.0$

**=> General**

- $b^{-1} \leq m < b^0$
Example

(Chapra and Canale, pg 61)

Consider hypothetical Floating-Point machine in base-2

7-bits word =
• 1 for sign
• 3 for signed exp.
   (1 sign, 2 for exp.)
• 3 for mantissa

Largest and smallest positive number represented are?
Consider hypothetical Floating-Point machine in base-2

7-bits word =  
- 1 for sign  
- 3 for signed exp.  
- 3 for mantissa

Largest number is: \(7 = 2^{(2+1)} (2^{-1} + 2^{-2} + 2^{-3})\)

<table>
<thead>
<tr>
<th>Sign nb</th>
<th>Sign exp</th>
<th>(2^1)</th>
<th>(2^0)</th>
<th>(2^{-1})</th>
<th>(2^{-2})</th>
<th>(2^{-3})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Smallest positive number is: 0.5 \(2^{-3}\)

<table>
<thead>
<tr>
<th>Sign</th>
<th>Sign exp</th>
<th>(2^1)</th>
<th>(2^0)</th>
<th>(2^{-1})</th>
<th>(2^{-2})</th>
<th>(2^{-3})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Consequence of Floating Point Reals

- Limited range of quantities can be represented
  - Min number (Underflow Error) and Max number (Overflow)
- Finite Number of quantities can be represented within the range (limited precision) => “Quantizing errors”
  - Quantizing errors treated either by round-off or chopping.
- Interval $\Delta x$ between numbers increases as numbers grow in magnitude
  - For $t =$ number of significant digits in mantissa and rounding,

\[ \frac{|\Delta x|}{|x|} \leq \frac{E}{2} \]

Relative Error

\[ |\Delta x| \leq \frac{E}{2}|x| \]

Absolute Error

\[ E = b^{1-t} = \text{Machine Epsilon} \]

% Determine machine epsilon in matlab
eps=1
while (eps+1>1)
  eps=eps/2;
end
eps*2

2.29 Numerical Fluid Mechanics
Arithmetic Operations

1. Addition and Subtraction

\[ r_1 \pm r_2 = m_1 b^{e_1} \pm m_2 b^{e_2} \]
Shift mantissa of smallest number, assuming \( e_1 > e_2 \)
Result has exponent of largest number:
\[ r_1 \pm r_2 = \left( m_1 \pm m_2 b^{e_2-e_1} \right) b^{e_1} = mb^{e_1} \]

Absolute Error
\[ \bar{\epsilon} \leq \bar{\epsilon}_1 + \bar{\epsilon}_2 \]
Relative Error
\[ \bar{\alpha} = \frac{|\bar{m} - m|}{|m|} \]
Unbounded for \( m_1 \pm m_2 \to 0 \)

2. Multiplication and Division

Multiplication:
Add exp, multiply mantissa, normalize and chop/round
\[ r_1 \times r_2 = m_1 m_2 b^{e_1+e_2} \]
\[ m = m_1 m_2 < 1 \]

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

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
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

- 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
Recursion: Heron’s Device

Numerically evaluate square-root

\[ \sqrt{s}, \quad s > 0 \]

Initial guess \( x_0 \)

\[ x_0 \approx \sqrt{s} \]

Test

\[ x_0^2 < s \quad \Rightarrow \quad x_0 < \sqrt{s} \quad \Rightarrow \quad \frac{s}{x_0} > \sqrt{s} \]
\[ x_0^2 > s \quad \Rightarrow \quad x_0 > \sqrt{s} \quad \Rightarrow \quad \frac{s}{x_0} < \sqrt{s} \]

Mean of guess and its reciprocal

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

Recursion Algorithm

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

MATLAB script

```matlab
% Numerically evaluate square-root

a=26; %Number for which the sqrt is to be computed
n=10; %Number of iteration in recursion
g=2; %Initial guess
dig=5; % Number of Digits

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')
gird on
```
Recursion: Horner’s scheme to evaluate polynomials by recursive additions

Goal: Evaluate polynomial

\[ p(z) = a_0 z^3 + a_1 z^2 + a_2 z + a_3 \]

\[ = ((a_0 z + a_1) z + a_2) z + a_3 \]

Horner’s Scheme

\[ a_0 \quad a_1 \quad a_2 \quad a_3 \]

\[ + \quad z b_0 \quad z b_1 \quad z b_2 \]

\[ (b_0 = a_0) \]

\[ b_0 \quad b_1 \quad b_2 \quad b_3 \]

\[ p(z) = b_3 \]

General order n

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

Recurrence relation

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

\[ p(z) = b_n \]

horner.m

```matlab
% 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
```

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

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

If \( x = 0.5 \), \( b = 0 \), \( c = 0 \) ⇒ \( y = 1.0 \)

Recursion.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

```
res=[(1:1:N) ar' z' a' y']
```

Contd.

```
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']);
```
```matlab
>> recur
b = 1; c = 1; x = 0.5;
dig=2

% Table
% i     delta    Sum      delta(approx)    Sum(approx)
%-------------------
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
```

2.29  Numerical Fluid Mechanics  PFJL Lecture 2, 17
Error Propagation
Spherical Bessel Functions

Differential Equation

\[ x^2 \frac{d^2 y}{dx^2} + 2x \frac{dy}{dx} \left( x^2 - n(n + 1) \right) y = 0 \]

Solutions

\[ j_n(x) y_n(x) \]

<table>
<thead>
<tr>
<th>( n )</th>
<th>( j_n(x) )</th>
<th>( y_n(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \frac{\sin x}{x} )</td>
<td>( -\frac{\cos x}{x} )</td>
</tr>
<tr>
<td>1</td>
<td>( \frac{\sin x}{x^2} - \frac{\cos x}{x} )</td>
<td>( -\frac{\cos x}{x^2} - \frac{\sin x}{x} )</td>
</tr>
</tbody>
</table>

\[ j_n(x) \to 0 \begin{cases} n \to \infty \\ x \to 0 \end{cases} \]

\[ y_n(x) \to -\infty \begin{cases} n \to \infty \\ x \to 0 \end{cases} \]

ps: Bessel functions are only used as example, no need to know everything about them for this class.
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 \( N \approx x + 20 \)
Error Propagation  
Euler’s Method

Differential Equation
\[ \frac{dy}{dx} = f(x, y), \quad y_0 = p \]

Example
\[ f(x, y) = x \left( y = \frac{x^2}{2} + p \right) \]

Discretization
\[ x_n = nh \]

Finite Difference (forward)
\[ \frac{dy}{dx} \bigg|_{x=x_n} \approx \frac{y_{n+1} - y_n}{h} \]

Recurrence
\[ y_{n+1} = y_n + hf(nh, y) \]

Central Finite Difference
\[ \frac{dy}{dx} \bigg|_{x=x_n} \approx \frac{y_{n+1} - y_{n-1}}{2h} \]
Error Analysis
Numerical Instability Example

Evaluate Integral

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

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

Proof:
\[ y_{n+5}y_{n-1} = \int_0^1 \frac{x^n + 5x^{n-1}}{x + 5} dx = \int_0^1 \frac{x^{n-1}(x + 5)}{x + 5} dx = \int_0^1 x^{n-1} dx = \frac{1}{n} \]

3-digit Recurrence:
\[
\begin{align*}
y_0 &= \int_0^1 \frac{dx}{x + 5} = \left[ \log_e(x + 5) \right]_0^1 = \log_e 6 - \log_e 5 = 0.182 \\
y_1 &= 1 - 5y_0 = 1 - 0.910 \simeq 0.090 \\
y_2 &= 0.5 - 5y_1 \simeq 0.049 \\
y_3 &= 0.333 - 5y_2 \simeq 0.083 \quad > y_2 !! \\
y_4 &= 0.25 - 5y_3 \simeq -0.165 \quad < 0 !! 
\end{align*}
\]

Backward Recurrence
\[ y_{n-1} = \frac{1}{5n} - \frac{y_n}{5} \]

\[
\begin{align*}
y_{10} &\simeq y_9 \Rightarrow y_9 + 5y_9 = 0.1 \Rightarrow y_9 = 0.017 \\
y_8 &= \frac{1}{45} - y_9/5 = 0.019 \\
y_7 &= \frac{1}{40} - y_8/5 = 0.021 \\
y_6 &= 0.025 \\
y_5 &= \cdot \\
y_4 &= \cdot \\
y_3 &= \cdot \\
y_2 &= 0.088 \\
y_1 &= 0.182 \quad \text{Correct} \\
y_0 &= 0.182 \quad \text{Correct}
\end{align*}
\]

Exercise: Make MATLAB script