

Applied Engineering Problem Solving

Lesson #4: Numerical Error

Prof. John R. White
Chemical and Nuclear Engineering
UMass-Lowell, Lowell MA

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Lesson #4 Goals

Numerical Error: Round-Off and Truncation Error...

Computer representation of numbers
(just the basics)

Round-off error and machine precision

Implication of round-off error in iterative techniques

Taylor series expansions and the **truncation error** associated with a finite approximation to infinite series (**FD approximation to derivatives**)

Trade-offs associated with **round-off** and **truncation errors**

Gilat:
none

Chapra:
Chapter 4

Lesson # 4 Lecture Notes
and Illustrative Examples

Introduction to
FD Methods
(solution of ODEs)

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Numerical Error



Round-Off Error

Due to the fact that computers can only represent quantities with a finite number of digits

Floating point arithmetic is NOT exact...

related to
finite word
size

Truncation Error

Associated with the approximations that are usually required when attempting to represent an exact mathematical expression or operation

related to
Taylor series

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Interesting Matlab Demo...



Floating point arithmetic is NOT exact...

Case 1: Let's start with a value, say 5.000, and add 0.125 to it several times:

We should get $5.000 + 0.125 = 5.125$

$5.125 + 0.125 = 5.250$

$5.250 + 0.125 = 5.375 \dots$

This is exactly
as observed in
Matlab

Case 2: Let's start with 5.000 again, and add 0.126 to it several times:

We should get $5.000 + 0.126 = 5.126$

$5.126 + 0.125 = 5.252$

$5.252 + 0.126 = 5.378 \dots$

but Matlab gives
 $5.2520000000000001e+000$

Let's do it in Matlab...

Where did the
extra digit
come from?

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Computer Representation of Numbers



Base 10 Arithmetic:

$$x = \sum_{k=0} b_k 10^k \quad \text{and} \quad y = \sum_{k=1} c_k 10^{-k}$$

Thus, x.y could be written as ...b₅b₄b₃b₂b₁b₀ . c₁c₂c₃... using **base 10** notation.

$$5.125 \rightarrow 5 \times 10^0 + 1 \times 10^{-1} + 2 \times 10^{-2} + 5 \times 10^{-3}$$

$$5.126 \rightarrow 5 \times 10^0 + 1 \times 10^{-1} + 2 \times 10^{-2} + 6 \times 10^{-3}$$

Human computers do arithmetic in base 10

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Computer Representation of Numbers



Base 2 (binary) Arithmetic:

$$x = \sum_{k=0} b_k 2^k \quad \text{and} \quad y = \sum_{k=1} c_k 2^{-k}$$

Thus, x.y could be written as ...b₅b₄b₃b₂b₁b₀ . c₁c₂c₃... using **base 2** notation.

$$5.125 \rightarrow 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$

$$4 + 0 + 1 + 0 + 0 + 0.125$$

$$(5.125)_{10} \longleftrightarrow (101.001)_2$$

see next slide

$$(5.126)_{10} \approx (101.0010000001000001)_2$$

$$\approx (5.125991821289063)_{10}$$

16 digits for fractional part

Electronic computers do arithmetic in base 2

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Computer Representation of Numbers



Table 1 Binary representation of some decimal numbers.

x = 5				
k	2 ^k	b _k	cumulative sum	
0	1	1	1	
1	2	0	1	
2	4	1	5	
y = 0.125				
k	2 ^{-k}	c _k	cumulative sum	
1	0.5	0	0	0
2	0.25	0	0	0
3	0.125	1	0.125	1
4	0.0625		0	0.125
5	0.03125		0	0.125
6	0.015625		0	0.125
7	0.0078125		0	0.125
8	0.00390625		0	0.125
9	0.001953125		0	0.125
10	0.0009765625		1	0.1259765625
11	0.00048828125		0	0.1259765625
12	0.000244140625		0	0.1259765625
13	0.0001220703125		0	0.1259765625
14	6.103515625e-005		0	0.1259765625
15	3.0517578125e-005		0	0.1259765625
16	1.52587890625e-005		1	0.125991821289063

Electronic computers do arithmetic in base 2

$$(5.125)_{10} \longleftrightarrow (101.001)_2$$

exact with 3 digits

$$(5.126)_{10} \approx (101.0010000001000001)_2 \approx (5.125991821289063)_{10}$$

approximate with 16 digits

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Computer Representation of Numbers



The important point here is that **all computers have a finite number of digits (bits) to represent a given number (or word)**

For **32 bit machines**:

- each binary digit (0 or 1) → bit
- 8 bits → byte
- single precision word → 4 bytes = 32 bits
- double precision word → 8 bytes = 64 bits

Matlab does all its computations with 64 bit arithmetic to minimize round-off error.

However, there is always a finite precision limit!!!

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Machine Epsilon, ϵ_m



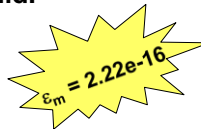
This precision limit is characterized by a number called **machine epsilon**, ϵ_m .

It is defined precisely **as the smallest floating point number, ϵ_m , such that**

$$1 + \epsilon_m > 1$$

We can easily estimate machine epsilon, ϵ_m , on any computer by continually reducing a number (say, by a factor of two) until the above condition is no longer valid.

```
epsilon = 1.0;  
while epsilon + 1.0 > 1.0  
    epsilon = epsilon/2;  
end
```



see meps.m

Matlab has a built-in variable, **eps**, to store this value...

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Ramifications of Round-Off Error...



Although **proper algorithm design** and **64-bit arithmetic** tend to **minimize round off error**, it is always something that you should be aware of when doing numerical computations and code development.

Because of round off error, we almost **never ask the question "Is $x = y$?"**.

Instead, we ask **"Is x close to y ?"** and this is often implemented as follows:

```
rerr = 1; tol = 1e-5;  
while rerr > tol  
    continue calculation that updates x and/or y  
    rerr = abs((x-y)/y);  
end
```

where tol is some user-defined tolerance...

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Taylor Series and Truncation Error



As mentioned previously, **truncation error** is often introduced when we **approximate continuous mathematical functions and operations with a discrete algebraic representation**.

This error is usually associated with the **actual truncation of an infinite series expansion for the quantity of interest to a finite number of terms** -- thus the term, **truncation error**.

Understanding series -- primarily the **Taylor Series** -- is **absolutely essential** for the **study of numerical methods** and for **understanding the concept of truncation error**.

From a **simplistic perspective**, the **Taylor series is a way to evaluate a function at a point $x = x_0 + \Delta x$ in terms of the function and all its derivatives evaluated at point x_0** , or

$$f(x_0 + h) = \frac{f(x_0)h^0}{0!} + \frac{f'(x_0)h^1}{1!} + \frac{f''(x_0)h^2}{2!} + \frac{f'''(x_0)h^3}{3!} + \dots$$

where $h = |\Delta x|$
is called the **step size**

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Taylor Series and Truncation Error



The **forward** Taylor series can be written in many forms:

$$f(x_0 + h) = \frac{f(x_0)h^0}{0!} + \frac{f'(x_0)h^1}{1!} + \frac{f''(x_0)h^2}{2!} + \frac{f'''(x_0)h^3}{3!} + \dots$$

$$f(x_0 + h) = \frac{f(x_0)h^0}{0!} + \frac{f'(x_0)h^1}{1!} + \frac{f''(x_0)h^2}{2!} + \frac{f'''(x_0)h^3}{3!} + O(h^4)$$

$$f(x_{i+1}) = \frac{f(x_i)h^0}{0!} + \frac{f'(x_i)h^1}{1!} + \frac{f''(x_i)h^2}{2!} + \frac{f'''(x_i)h^3}{3!} + O(h^4)$$

$$f_{i+1} = f_i + f'_i h + \frac{f''_i h^2}{2!} + \frac{f'''_i h^3}{3!} + \alpha h^4$$

$$O(h^n) = \alpha h^n$$

where

$O(h^n) = \alpha h^n$ is the error or remainder which, upon truncation, accounts for all the remaining terms in the series. This term is **"proportional to h^n "**

$x_i = x_0$ and $x_{i+1} = x_i + h$ just puts things into discrete form...

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Example of Round-Off & Truncation Error



For $f(x) = e^x$ with the reference point at $x_0 = 0$, the **forward** Taylor series

$$f(x_0 + h) = \frac{f(x_0)h^0}{0!} + \frac{f'(x_0)h^1}{1!} + \frac{f''(x_0)h^2}{2!} + \frac{f'''(x_0)h^3}{3!} + \dots$$

becomes

$$f(h) = e^h = \frac{1}{0!} + \frac{h^1}{1!} + \frac{h^2}{2!} + \frac{h^3}{3!} + \dots$$

since $d^n(e^x)/dx^n = e^x$
and $e^0 = 1$

But, now we can let $h = x$ for convenience of notation.

Thus,

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

Similarly,

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \dots = \sum_{n=0}^{\infty} \frac{(-1)^n x^n}{n!}$$

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

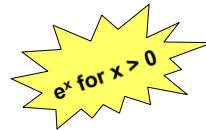
Example of Round-Off & Truncation Error



Consider the computation of e^3 and e^{-3} using the Taylor series **truncated to 8 terms** with only **5 significant digits** in our calculations.

Doing the calculations gives:

$$\begin{aligned} \text{Case 1: } e^3 &\approx 1 + 3 + \frac{9}{2} + \frac{27}{6} + \frac{81}{24} + \frac{243}{120} + \frac{729}{720} + \frac{2187}{5040} \\ &= 1 + 3 + 4.5 + 4.5 + 3.375 + 2.025 + 1.0125 + 0.43393 = 19.846 \end{aligned}$$



and my calculator gives $e^3 = 20.086$ -- thus, our 8-term estimate has an **error of about -1.2%**.

This error is **dominated by truncation error**, with only **a minor loss in accuracy associated with rounding the individual calculations to 5 figures** -- that is, the addition of a few more terms in the series would give very accurate results.

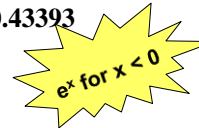
CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Example of Round-Off & Truncation Error



Case 2: $e^{-3} \approx 1 - 3 + 4.5 - 4.5 + 3.375 - 2.025 + 1.0125 - 0.43393$
 $= -7.1430e-2$



and the actual value is $e^{-3} = 4.9787e-2$ -- which shows that our estimate is **terrible with about -243% error** (we didn't even get the correct sign!!!).

This example has a **serious case of both truncation error and round off error** -- in particular, notice that some of the individual terms are nearly a factor of 100 larger than the final result.

Although additional terms in the series would help considerably, **we could never get 5 significant figures of accuracy**, because the subtraction of nearly equal terms leads to the loss of significant digits (this is often referred to as **catastrophic cancellation**).

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Example of Round-Off & Truncation Error



Notice, however, that if we compute the result for e^{-3} using **the inverse of the Case 1 result**, we have

$$e^{-3} = \frac{1}{e^3} = \frac{1}{19.846} = 5.0388e-2$$

which only represents an error of 1.2%.

This is an example of what I mean by **"proper algorithm design"** !

Many times, however, **"proper algorithm design"** is not so simple, so we will **leave much of the hard-core development of various numerical algorithms to the mathematicians and numerical analysis experts**.

In fact, that is why **we will use Matlab to do many of the needed computations**, since many years of experience has shown that **most of the built-in algorithms are quite efficient and robust for a wide range of applications**.

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Derivative Approximations



Forward TS: $f_{i+1} = f_i + f_i' h + \frac{f_i'' h^2}{2!} + \frac{f_i''' h^3}{3!} + \dots + \frac{f_i^{(n)} h^n}{n!} + O(h^{n+1})$

Backward TS: $f_{i-1} = f_i - f_i' h + \frac{f_i'' h^2}{2!} - \frac{f_i''' h^3}{3!} + \dots + (-1)^n \frac{f_i^{(n)} h^n}{n!} + O(h^{n+1})$

Forward Approximation to f_i' :

(from FTS)

$$f_{i+1} = f_i + f_i' h + O(h^2)$$

$$f_i' = \frac{f_{i+1} - f_i}{h} + O(h)$$

1st order forward estimate to f_i'

Backward Approximation to f_i' :

(from BTS)

$$f_{i-1} = f_i - f_i' h + O(h^2)$$

$$f_i' = \frac{f_i - f_{i-1}}{h} + O(h)$$

1st order backward estimate to f_i'

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Derivative Approximations (cont.)



Forward TS: $f_{i+1} = f_i + f_i' h + \frac{f_i'' h^2}{2!} + \frac{f_i''' h^3}{3!} + \dots + \frac{f_i^{(n)} h^n}{n!} + O(h^{n+1})$

Backward TS: $f_{i-1} = f_i - f_i' h + \frac{f_i'' h^2}{2!} - \frac{f_i''' h^3}{3!} + \dots + (-1)^n \frac{f_i^{(n)} h^n}{n!} + O(h^{n+1})$

Central Approximation to f_i' :

(from FTS-BTS)

$$f_{i+1} - f_{i-1} = 2f_i' h + O(h^3)$$

1st derivative

$$f_i' = \frac{f_{i+1} - f_{i-1}}{2h} + O(h^2)$$

2nd order central estimate to f_i'

Central Approximation to f_i'' :

(from FTS+BTS)

$$f_{i+1} + f_{i-1} = 2f_i + f_i'' h^2 + O(h^4)$$

2nd derivative

$$f_i'' = \frac{f_{i-1} - 2f_i + f_{i+1}}{h^2} + O(h^2)$$

2nd order central estimate to f_i''

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Example with Derivative Approximations



Let's estimate, using some FD approximations, the 1st and 2nd derivatives of e^x at $x = 0$

again $d^n(e^x)/dx^n = e^x$ and $e^0 = 1$

Table 3 Approximate derivatives for e^x at $x = 0$ (from deriv_approx.m).

step size, h	$f'_i = \frac{f_i - f_{i-1}}{h}$	$f'_i = \frac{f_{i+1} - f_i}{h}$	$f'_i = \frac{f_{i+1} - f_{i-1}}{2h}$	$f''_i = \frac{f_{i-1} - 2f_i + f_{i+1}}{h^2}$
0.50000	0.78694	1.29744	1.04219	1.02101
0.25000	0.88480	1.13610	1.01045	1.00522
0.12500	0.94002	1.06519	1.00261	1.00130
0.06250	0.96939	1.03191	1.00065	1.00033
0.03125	0.98454	1.01579	1.00016	1.00008

see deriv_approx.m

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Order of Error from Numerical Tests



Truncation error is often “proportional to the step size to some power n ”, or

$$\varepsilon = \alpha h^n$$

The order of error, n , can often be estimated via a set of numerical experiments that use different step sizes.

For example, a plot of ε vs. h on a log-log scale gives a straight line with slope n :

$$\log \varepsilon = \log(\alpha h^n) = \log \alpha + \log h^n = \log \alpha + n \log h$$

Or, with just two separate evaluations, we have

$$\varepsilon_1 = \alpha h_1^n \quad \text{and} \quad \varepsilon_2 = \alpha h_2^n$$

$$\frac{\varepsilon_1}{\varepsilon_2} = \frac{\alpha h_1^n}{\alpha h_2^n} = \left(\frac{h_1}{h_2} \right)^n \quad \text{or} \quad n = \frac{\log(\varepsilon_1/\varepsilon_2)}{\log(h_1/h_2)}$$

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

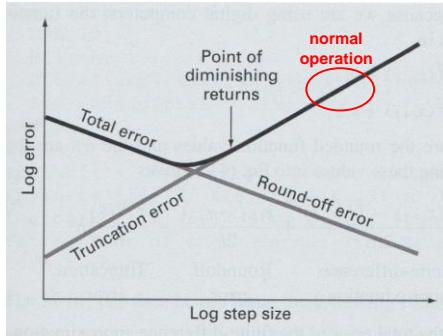
(Oct. 2017)

Truncation vs. Round-off Errors



Reducing the step size, h , is the most common way to reduce truncation error.

However, this often leads to an increased number of computations and, since round-off error is accumulative, more floating point arithmetic leads to more round-off error.



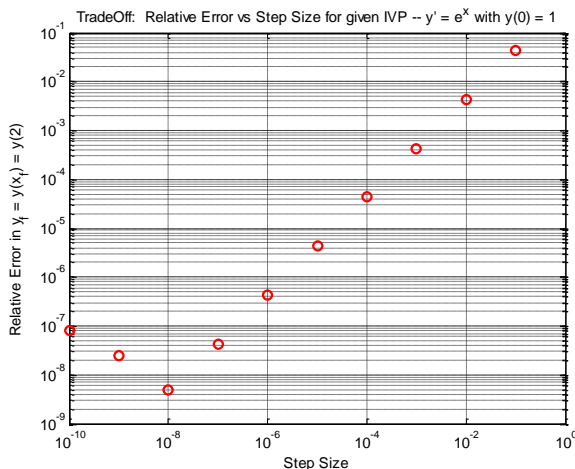
This is illustrated in the sketch, where the total error is simply the sum of the truncation and round-off errors.

However, for most practical engineering problems, the truncation error dominates.

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

Example: Truncation vs. Round-off Errors



Let's show
this in Matlab
see tradeoff.m

Problem:

$$\frac{dy}{dx} = e^x \quad \text{with} \quad y(0) = 1$$

Discrete Form:

$$y_{i+1} = y_i + e^{x_i} h$$

Solution is implemented in a simple recursive loop and repeated with various h values...

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

More Illustrative Examples



On Evaluating Infinite Series – An Example

Example that illustrates how to generate a Taylor series for $f(x) = \sinh(x)$ and on how to efficiently evaluate this **infinite power series** in Matlab.

see
[infinite_series.pdf](#)

Can you derive this? $f(x) = \sinh x = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \dots = \sum_{n=1}^{\infty} \frac{x^{2n-1}}{(2n-1)!}$

two issues

Algorithm to Evaluate Infinite Power Series

Set **maxT** and **tol** for stopping the calculation (also set $\epsilon > \text{tol}$)
 Initialize counter and first term -- set **n = 1** and **T = T₁**
 Initialize the partial sum to the first term -- set **f = T**
while $\epsilon > \text{tol}$ && **n** < **maxT**
 compute **r**, where $r_n = T_{n+1}/T_n$ (specific to function of interest)
 T = **r*****T** (compute next term in series)
 f = **f** + **T** (update partial sum)
 $\epsilon = \max(\text{abs}(T/f))$ (compute maximum relative change)
 n = **n** + 1 (increment counter)
end

How do we compute r_n ?

CHEN.3170 Applied Engineering Problem Solving
 Lesson 4: Numerical Error

(Oct. 2017)

How do we compute r_n ?



For the specific case of $f(x) = \sinh(x)$:

$$f(x) = \sinh x = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \dots = \sum_{n=1}^{\infty} \frac{x^{2n-1}}{(2n-1)!}$$

Thus r_n becomes

$$r_n = \frac{T_{n+1}}{T_n} = \frac{x^{2(n+1)-1}}{(2(n+1)-1)!} \times \frac{(2n-1)!}{x^{2n-1}}$$

$$= \frac{x^2 x^{2n-1}}{(2n+1)(2n)(2n-1)!} \times \frac{(2n-1)!}{x^{2n-1}} = \frac{x^2}{(2n+1)(2n)}$$

where we have used the fact that

$$(2(n+1)-1)! = (2n+2-1)! = (2n+1)! = (2n+1)(2n)(2n-1)!$$

again, see [infinite_series.pdf](#) for the detailed development and implementation within Matlab

CHEN.3170 Applied Engineering Problem Solving
 Lesson 4: Numerical Error

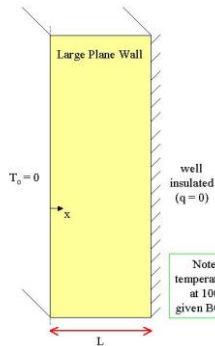
(Oct. 2017)

More Illustrative Examples (cont.)

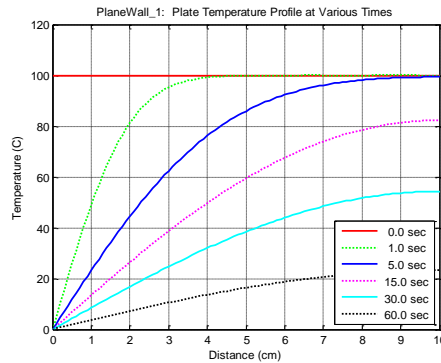


Evaluating and Plotting Space-Time Temperature Distributions

$$T(x,t) = \sum_{n=1}^{\infty} a_n \sin(\lambda_n x) e^{-\alpha \lambda_n^2 t} \quad \text{with} \quad \lambda_n = \frac{(2n-1)\pi}{2L} \quad \text{and} \quad a_n = \frac{4T_i}{(2n-1)\pi}$$



see
planewall_1.pdf



CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)

More Illustrative Examples (cont.)



Introduction to Finite Difference Methods for Solution of ODEs

This is a **BIGGIE** -- we will emphasize this in a separate presentation...

The goal here is to convert **continuous differential equations** into **discrete difference equations**...

IVPs and **BVPs** are treated quite differently

IVPs lead to recursive equations

BVPs lead to simultaneous equations

Let's look at the details...

CHEN.3170 Applied Engineering Problem Solving
Lesson 4: Numerical Error

(Oct. 2017)