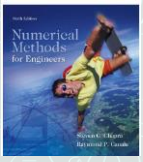
ریاضیات عالی پیشرفته

Numerical Methods for Engineers

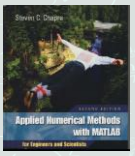
مدرس دکتر پدرام پیوندی

منابع:

Numerical Methods for Engineers, 6th Edition 2009 Steven Chapra



Applied Numerical Methods With MATLAB for Engineers and Scientists Steven Chapra



ارزیابی:

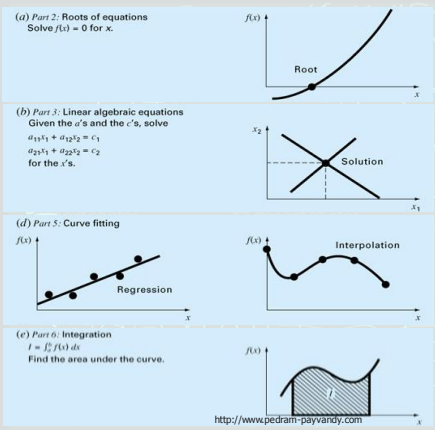
امتحان پایان ترم : ۴۰٪

تمرینات: ۶۰٪

هر جلسه غیبت: ۱۰٪-

- www.pedram-payvandy.com
- peivandi@yazd.ac.ir
- نام ونام خانوادگی شمار دانشجویی
- عنوان درس
- ایمیل

Mathematical Modeling and Engineering Problem solving



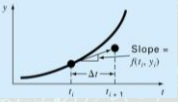
Every part in this book requires some mathematical background

(f) Part 7: Ordinary differential equations

Given $\frac{dy}{dt} \cong \frac{\Delta y}{\Delta t} = f(t, y)$

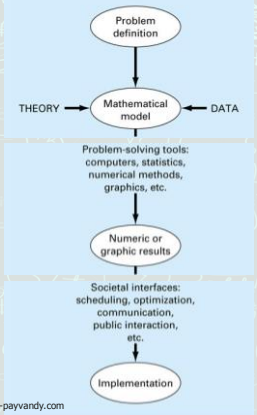
solve for y as a function of t.

$y_{i+1} = y_i + f(t_i, y_i)\,\Delta t$



Computers are great tools,
however, without fundamental understanding of engineering problems, they will be useless.

The Engineering Problem Solving Process



Newton's 2nd law of Motion (**here)

- *"The time rate change of momentum of a body is equal to the resulting force acting on it."*
- Formulated as **F = m.a**
  - **F** = net force acting on the body
  - **m** = mass of the object (kg)
  - **a** = its acceleration (m/s²)
- Some complex models may require more sophisticated mathematical techniques than simple algebra
  - Example, modeling of a falling parachutist:

$$F = F_D + F_U$$

F_U = Force due to air resistance = $-cv$ (**c = drag coefficient**)

F_D = Force due to gravity = mg

$$\frac{dv}{dt} = \frac{F}{m}$$

$$F = F_D + F_U$$

$$F_D = mg$$

$$F_U = -cv$$

$$\frac{dv}{dt} = \frac{mg - cv}{m}$$

$$\frac{dv}{dt} = g - \frac{c}{m}v$$

- This is a first order ordinary **differential equation**. We would like to solve for v (velocity).
- It can **not** be solved using algebraic manipulation
- Analytical Solution:
  If the parachutist is initially at rest (v=0 at t=0), using calculus dv/dt can be solved to give the result:

Dependent variable

Independent variable

$$v(t) = \frac{gm}{c}\left(1 - e^{-(c/m)t}\right)$$

Forcing function

Parameters

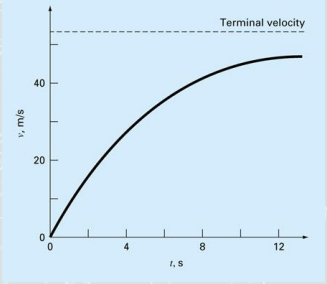
Analytical Solution

$$v(t) = \frac{gm}{c}\left(1 - e^{-(c/m)t}\right)$$

If ***v(t)*** could not be solved **analytically**, then we need to use a numerical method to solve it

g = 9.8 m/s² c =12.5 kg/s
m = 68.1 kg

t (sec.)	V (m/s)
0	0
2	16.40
4	27.77
8	41.10
10	44.87
12	47.49
∞	53.39



Numerical Solution

$$\frac{dv}{dt} \cong \frac{\Delta v}{\Delta t} = \frac{v(t_{i+1}) - v(t_i)}{t_{i+1} - t_i} \quad \ldots\ldots \quad \frac{dv}{dt} = \lim_{\Delta t \to 0} \frac{\Delta v}{\Delta t}$$

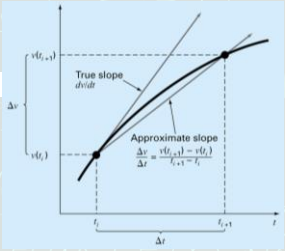
$$\frac{v(t_{i+1}) - v(t_i)}{t_{i+1} - t_i} = g - \frac{c}{m}v(t_i)$$

This equation can be rearranged to yield

$$v(t_{i+1}) = v(t_i) + [g - \frac{c}{m}v(t_i)](t_{i+1} - t_i)$$

t (sec.)	V (m/s)
0	0
2	19.60
4	32.00
8	44.82
10	47.97
12	49.96
∞	53.39

Δt = 2 sec



To minimize the error, use a smaller step size, Δt
No problem, if you use a computer!

Analytical vs. Numerical solution

m=68.1 kg c=12.5 kg/s
g=9.8 m/s

t (sec.)	V (m/s)
0	0
2	16.40
4	27.77
8	41.10
10	44.87
12	47.49
∞	53.39

Δt = 2 sec

t (sec.)	V (m/s)
0	0
2	19.60
4	32.00
8	44.82
10	47.97
12	49.96
∞	53.39

Δt = 0.5 sec

t (sec.)	V (m/s)
0	0
2	17.06
4	28.67
8	41.94
10	45.60
12	48.09
∞	53.39

Δt = 0.01 sec

t (sec.)	V (m/s)
0	0
2	16.41
4	27.83
8	41.13
10	44.90
12	47.51
∞	53.39

$$v(t) = \frac{gm}{c}\left(1 - e^{-(c/m)t}\right)$$

$$v(t_{i+1}) = v(t_i) + [g - \frac{c}{m}v(t_i)]\Delta t$$

CONCLUSION: If you want to minimize the error, use a smaller step size, Δt

Chapter 2

Programming and Software

Chapter 2

- Objective is how to use the computer as a tool to obtain numerical solutions to a given engineering model. There are two ways in using computers:
  - Use available software
  - Or, write computer programs to extend the capabilities of available software, such as Excel and Matlab.
- Engineers should not be tool limited, it is important that they should be able to do both!

- Computer programs are set of instructions that direct the computer to perform a certain task.
- To be able to perform engineering-oriented numerical calculations, you should be familiar with the following programming topics:
  - Simple information representation (constants, variables, and type declaration)
  - Advanced information representation (data structure, arrays, and records)
  - Mathematical formulas (assignment, priority rules, and intrinsic functions)
  - Input/Output
  - Logical representation (sequence, selection, and repetition)
  - Modular programming (functions and subroutines)
- We will focus the last two topics, assuming that you have some prior exposure to programming.

Structured Programming

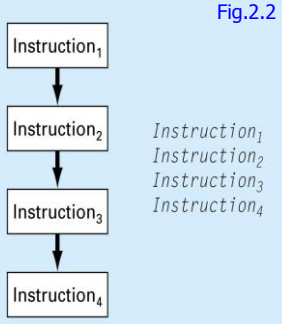
- *Structured programming* is a set of rules that prescribe good style habits for programmer.
  - An organized, well structured code
  - Easily sharable
  - Easy to debug and test
  - Requires shorter time to develop, test, and update
- The key idea is that any numerical algorithm can be composed of using the three fundamental structures:
  - Sequence, selection, and repetition

Fig. 2.1

SYMBOL	NAME	FUNCTION
	Terminal	Represents the beginning or end of a program.
	Flowlines	Represents the flow of logic. The humps on the horizontal arrow indicate that it passes over and does not connect with the vertical flowlines.
	Process	Represents calculations or data manipulations.
	Input/output	Represents inputs or outputs of data and information.
	Decision	Represents a comparison, question, or decision that determines alternative paths to be followed.
	Junction	Represents the confluence of flowlines.
	Off-page connector	Represents a break that is continued on another page.
	Count-controlled loop	Used for loops which repeat a prespecified number of iterations.

- **Sequence.** Computer code must be implemented one instruction at a time, unless you instruct otherwise. The structure can be expressed as a flowchart or pseudocode.

Fig.2.2



19

Fig. 2.3

- **Selection.** Splits the program's flow into branches based on outcome of a logical condition.

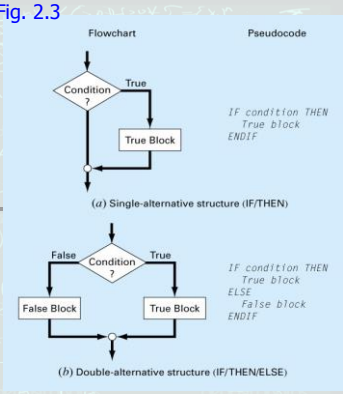
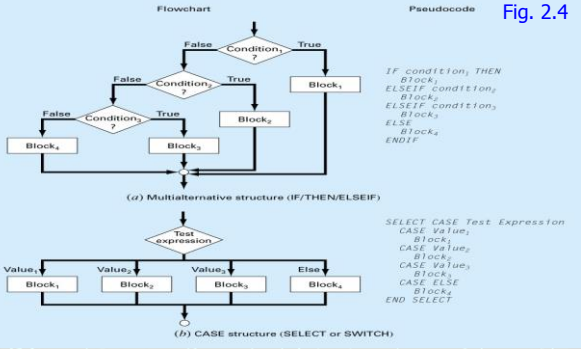


Fig. 2.4



- **Repetition.** A means to implement instructions repeatedly.

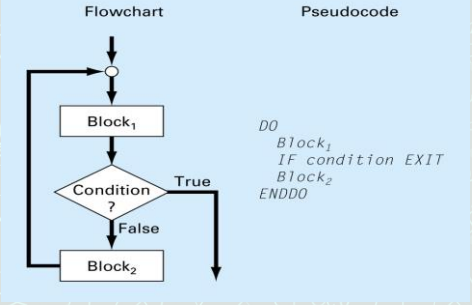
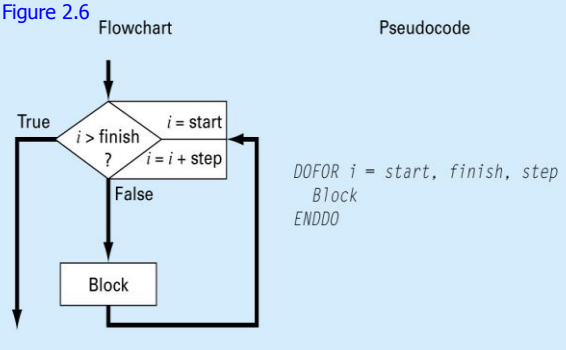


Figure 2.6



Modular Programming

- The computer programs can be divided into subprograms, or modules, that can be developed and tested separately.
- Modules should be as independent and self contained as possible.
- Advantages to modular design are:
  - It is easier to understand the underlying logic of smaller modules
  - They are easier to debug and test
  - Facilitate program maintenance and modification
  - Allow you to maintain your own library of modules for later use

```
FUNCTION Euler(dt, ti, tf, yi)
t = ti
y = yi
h = dt
DO
  IF t + dt > tf THEN
    h = tf - t
  ENDIF
  dydt = dy(t, y)
  y = y + dydt * h
  t = t + h
  IF t ≥ tf EXIT
ENDDO
Euler = y
END
```

Fig. 2.7

EXCEL

- Is a spreadsheet that allow the user to enter and perform calculations on rows and columns of data.
- When any value on the sheet is changed, entire calculation is updated, therefore, spreadsheets are ideal for "what if?" sorts of analysis.
- Excel has some built in numerical capabilities including equation solving, curve fitting and optimization.
- It also includes VBA as a macro language that can be used to implement numerical calculations.
- It has several visualization tools, such as graphs and three dimensional plots.

Fig. 2.8

(a) Pseudocode	(b) Excel VBA
IF/THEN:	

IF condition THEN
  True block
ENDIF
	If b <> 0 Then
	  rl = -c / b
	End If

IF/THEN/ELSE:	

IF condition THEN
  True block
ELSE
  False block
ENDIF
	If a < 0 Then
	  b = Sqr(Abs(a))
	Else
	  b = Sqr(a)
	End If

IF/THEN/ELSEIF:	

IF condition1 THEN
  Block1
ELSEIF condition2
  Block2
ELSEIF condition3
  Block3
ELSE
  Block4
ENDIF
	If class = 1 Then
	  x = x + 8
	ElseIf class < 1 Then
	  x = x - 8
	ElseIf class < 10 Then
	  x = x - 32
	Else
	  x = x - 64
	End If

CASE:	

SELECT CASE Test Expression
  CASE Value1
  Block1
  CASE Value2
  Block2
  CASE Value3
  Block3
  CASE ELSE
  Block4
END SELECT
	Select Case a + b
	  Case Is < -50
	  x = -5
	  Case Is < 0
	  x = -5 - (a + b) / 10
	  Case Is < 50
	  x = (a + b) / 10
	  Case Else
	  x = 5
	End Select

DOEXIT:	

DO
  Block1
  IF condition EXIT
  Block2
ENDIF
	Do
	  i = i + 1
	  If i >= 10 Then Exit Do
	  j = i*x
	Loop

MATLAB

- Is a flagship software which was originally developed as a *matrix laboratory*. A variety of numerical functions, symbolic computations, and visualization tools have been added to the matrix manipulations.
- MATLAB is closely related to programming.

Fig. 2.9

(a) Pseudocode	(b) MATLAB
IF/THEN:	

IF condition THEN
  True block
ENDIF
	if b ~= 0
	  rl = -c / b;
	end

IF/THEN/ELSE:	

IF condition THEN
  True block
ELSE
  False block
ENDIF
	if a < 0
	  b = sqrt(abs(a));
	else
	  b = sqrt(a);
	end

IF/THEN/ELSEIF:	

IF condition1 THEN
  Block1
ELSEIF condition2
  Block2
ELSEIF condition3
  Block3
ELSE
  Block4
ENDIF
	if class == 1
	  x = x + 8;
	elseif class < 1
	  x = x - 8;
	elseif class < 10
	  x = x - 32;
	else
	  x = x - 64;
	end

CASE:	

SELECT CASE Test Expression
  CASE Value1
  Block1
  CASE Value2
  Block2
  CASE Value3
  Block3
  CASE ELSE
  Block4
END SELECT
	switch a + b
	  case 1
	  x = -5;
	  case 2
	  x = -5 - (a + b) / 10;
	  case 3
	  x = (a + b) / 10;
	  otherwise
	  x = 5;
	end

DOEXIT:	

DO
  Block1
  IF condition EXIT
  Block2
ENDIF
	while (1)
	  i = i + 1;
	  if i >= 10, break, end
	  j = i*x;
	end

COUNT-CONTROLLED LOOP:	

DOFOR i = start, Finish, step
  Block
ENDDO
	for i = 1:10:2
	  x = x + i;
	end

Other Languages and Libraries

- Fortran 90 (IMSL)
- C++

Approximations and Round-Off Errors

Chapter 3

- Numerical methods yield *approximate* results that are close to the exact analytical solution.
- How confident we are in our approximate result ? In other words,

"how much error is present in our calculation and is it tolerable?"

Significant Figures

- Number of *significant figures* indicates precision. Significant digits of a number are those that can be used with *confidence*, e.g., the number of certain digits plus one estimated digit.

53,800 How many significant figures?

5.38×10^4 3
5.3800×10^4 5

Zeros are sometimes used to locate the decimal point not significant figures.
0.00001753 4
0.001753 4

Identifying Significant Digits

- All non-zero digits are considered significant. For example, 91 has two significant figures, while 123.45 has five significant figures
- Zeros appearing anywhere between two non-zero digits are significant. Ex: 101.1002 has seven significant figures.
- Leading zeros are not significant. Ex: 0.00052 has two significant figures.
- Trailing zeros in a number containing a decimal point are significant. Ex: 12.2300 has six significant figures: 1, 2, 2, 3, 0 and 0. The number 0.000122300 still has only six significant figures (the zeros before the 1 are not significant). In addition, 120.00 has five significant figures.
- The significance of trailing zeros in a number not containing a decimal point can be ambiguous. For example, it may not always be clear if a number like 1300 is accurate to the nearest unit. Various conventions exist to address this issue.

Error Definitions

True error: E_t = True value – Approximation (+/-)

True percent relative error: $\varepsilon_t = \left|\frac{\text{True value} - \text{Approximation}}{\text{True value}}\right| \times 100\%$

Approximate Error

- For numerical methods, the true value will be known only when we deal with functions that can be solved *analytically*.
- In real world applications, we usually do not know the answer a priori.

Approximate Relative Error: $\varepsilon_a = \left|\frac{\text{Approximate error}}{\text{Approximation}}\right| \times 100\%$

Iterative approaches (e.g. Newton's method)

Approx. Relative Error: $\varepsilon_a = \left|\frac{\text{(Current Approx.) - (Previous Approx.)}}{\text{CurrentApprox.}}\right| \times 100\%$

Computations are repeated until stopping criterion is satisfied

$|\varepsilon_a| < \varepsilon_s$ ← Pre-specified % tolerance based on your knowledge of the solution. (Use absolute value)

If ε_s is chosen as: $\varepsilon_s = (0.5 \times 10^{(2-n)})\%$

Then the result is correct to *at least* n significant figures *(Scarborough 1966)*

EXAMPLE 3.2: *Maclaurin series expansion*

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!}$$

Calculate $e^{0.5}$ (= 1.648721…) up to 3 significant figures. During the calculation process, compute the *true* and *approximate* percent relative errors at each step

Error tolerance → $\varepsilon_s = (0.5 \times 10^{(2-3)})\% = 0.05\%$

Terms
1
1+(0.5)
1+(.5)+(.5)²/2
1+(.5)+(.5)²/2+(.5)³/6

Count	Result	ε_t (%) True	ε_a (%) Approx.
1	1	39.3	
2	1.5	9.02	33.3
3	1.625	1.44	7.69
4	1.6458333	0.175	1.27
5	1.6484375	0.0172	0.158
6	1.648697917	0.00142	0.0158

Round-off and Chopping Errors

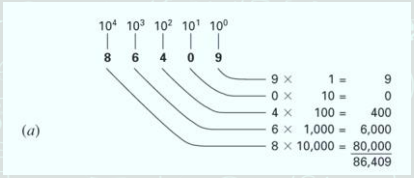
- Numbers such as **π**, **e**, or **√7** cannot be expressed by a fixed number of significant figures. Therefore, they can not be represented exactly by a computer which has a **fixed word-length**

$$\pi = 3.1415926535\ldots$$

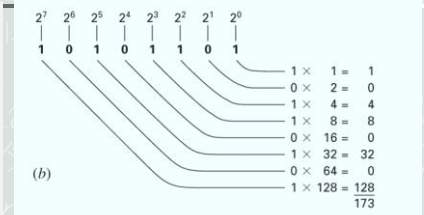
- Discrepancy introduced by this omission of significant figures is called ***round-off*** **or** ***chopping*** **errors.**
- If π is to be stored on a base-10 system carrying 7 significant digits,
  chopping: π=3.141592 error: ε_t=0.00000065
  round-off: π=3.141593 error: ε_t=0.00000035
- Some machines use ***chopping***, because ***rounding*** has additional computational overhead.

Number Representation

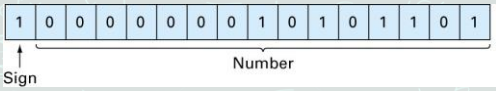
86409 in Base-10



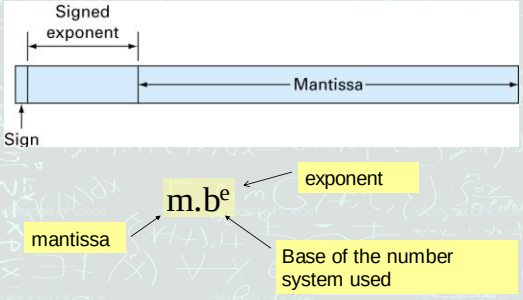
173 in Base-2



The representation of -173 on a 16-bit computer using the *signed magnitude method*



Computer representation of a floating-point number



156.78 ➧➧➧ 0.15678×10^3
(in a floating point base-10 system)

$\frac{1}{34} = 0.029411765$ Suppose only 4 decimal places to be stored

0.0294×10^0

- ***Normalize*** **➔ remove the leading zeroes.**
- Multiply the mantissa by 10 and lower the exponent by 1

0.2941×10^{-1}

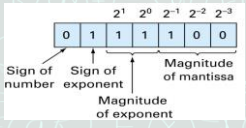
Additional significant figure is retained

- Due to ***Normalization***, absolute value of m is limited: $\frac{1}{b} \le m < 1$

for ***base-10*** system: **0.1 ≤ *m* < 1**
for ***base-2*** system: **0.5 ≤ *m* < 1**

- Floating point representation allows both fractions and very large numbers to be expressed on the computer. However,
  - Floating point numbers take up more room
  - Take longer to process than integer numbers.

Q: What is the smallest positive floating point number that can be represented using a 7-bit word (3-bits reserved for mantissa).
What is the number?
(* Solve Example 3.4 page 61 *)



Another Exercise: What is the largest positive floating point number that can be represented using a 7-bit word (3-bits reserved for mantissa).

Additional Notes on floating point numbers:

- Addition of two floating point numbers (normalization is needed)
- Multiplication
- Overflow / Underflow
  very small and very large numbers can not be represented using a fixed-length mantissa/exponent representation, therefore overflow and underflow can occur while doing arithmetic with these numbers.
- Double precision arithmetic is always recommended
- The interval between Numbers increases as the numbers grow in magnitude

Chapter 4

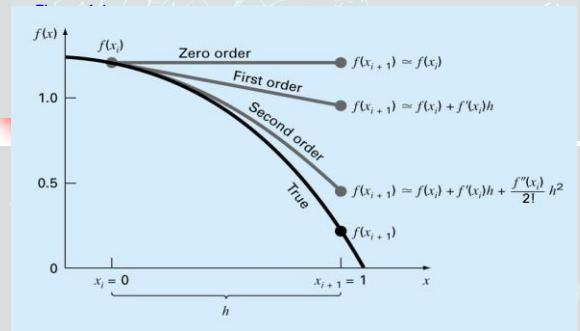
Truncation Errors and the Taylor Series

Chapter 4

- Non-elementary functions such as trigonometric, exponential, and others are expressed in an approximate fashion using Taylor series when their values, derivatives, and integrals are computed.
- Any smooth function can be approximated as a polynomial. Taylor series provides a means to predict the value of a function at one point in terms of the function value and its derivatives at another point.

$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + \frac{f''(x_i)}{2!}(x_{i+1} - x_i)^2 + \frac{f^{(3)}(x_i)}{3!}(x_{i+1} - x_i)^3 + \cdots + \frac{f^{(n)}(x_i)}{n!}(x_{i+1} - x_i)^n + R_n$$

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



Example:

To get the *cos*(x) for small x:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

If x=0.5

cos(0.5) =1-0.125+0.0026041-0.0000127+ …
=0.877582

From the supporting theory, for this series, the error is no greater than the first omitted term.

$$\therefore \quad \frac{x^8}{8!} \quad for \quad x = 0.5 \quad = 0.0000001$$

- Any smooth function can be approximated as a polynomial.

$f(x_{i+1}) \approx f(x_i)$ *zero order* approximation, only true if x_{i+1} and x_i are very close to each other.

$f(x_{i+1}) \approx f(x_i) + f'(x_i)\,(x_{i+1}-x_i)$ *first order* approximation, in form of a straight line

http://www.pedram-payvandy.com

n^{th} order approximation

$$f(x_{i+1}) \cong f(x_i) + f'(x_i)(x_{i+1}-x_i) + \frac{f''}{2!}(x_{i+1}-x_i)^2 + \ldots + \frac{f^{(n)}}{n!}(x_{i+1}-x_i)^n + R_n$$

$(x_{i+1}-x_i)= h$ *step size* (define first)

$$R_n = \frac{f^{(n+1)}(\varepsilon)}{(n+1)!}h^{(n+1)}$$

- Reminder term, R_n, accounts for all terms from (n+1) to infinity.

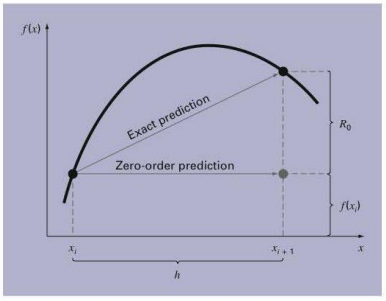
http://www.pedram-payvandy.com

- ε is not known exactly, lies somewhere between $x_{i+1} > \varepsilon > x_i$.
- Need to determine $f^{n+1}(x)$, to do this you need f'(x).
- If we knew f(x), there wouldn't be any need to perform the Taylor series expansion.
- However, $R=O(h^{n+1})$, $(n+1)^{th}$ order, the order of truncation error is h^{n+1}.
- O(h), halving the step size will halve the error.
- $O(h^2)$, halving the step size will quarter the error.

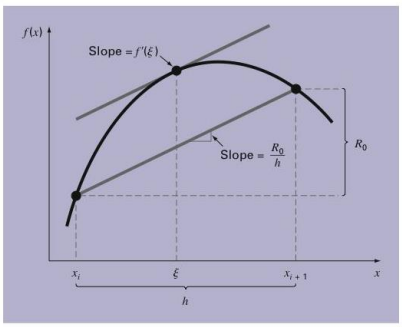
http://www.pedram-payvandy.com

$f(x_{i+1}) \cong f(x_i)$

$$R_0 = f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \frac{f^{(3)}(x_i)}{3!}h^3 + \cdots \qquad R_0 \cong f'(x_i)h$$



52



$$f'(\xi) = \frac{R_0}{h}$$

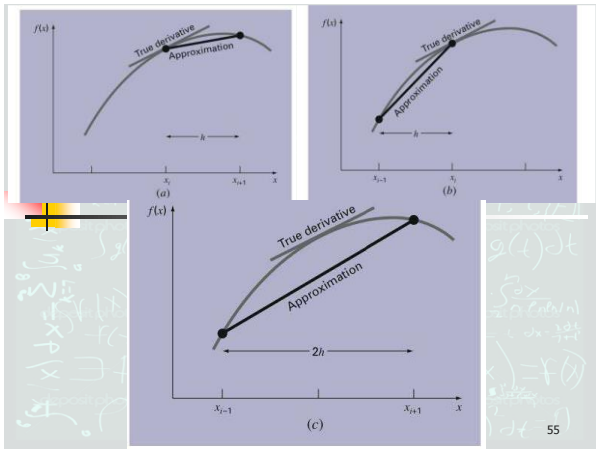
53

- Truncation error is decreased by addition of terms to the Taylor series.
- If h is sufficiently small, only a few terms may be required to obtain an approximation close enough to the actual value for practical purposes.

$$f(x_{i-1}) = f(x_i) - f'(x_i)h + \frac{f''(x_i)}{2!}h^2 - \cdots$$

$$f'(x_i) \cong \frac{f(x_i) - f(x_{i-1})}{h} = \frac{\nabla f_1}{h}$$

http://www.pedram-payvandy.com



55

Error Propagation

- fl(x) refers to the floating point (or computer) representation of the real number x. Because a computer can hold a finite number of significant figures for a given number, there may be an error (round-off error) associated with the floating point representation. The error is determined by the precision of the computer (ε).

http://www.pedram-payvandy.com

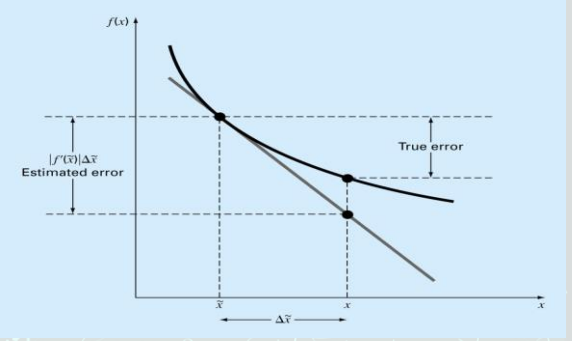
- Suppose that we have a function f(x) that is dependent on a single independent variable x. x_{fl} is an approximation of x and we would like to estimate the effect of discrepancy between x and x_{fl} on the value of the function:

$\Delta f(x_{fl}) = |f(x) - f(x_{fl})|$ both f(x) and $f(x_{fl})$ are unknown

Employ Taylor series to compute f(x) near $f(x_{fl})$, dropping the second and higher order terms

$f(x) - f(x_{fl}) \cong f'(x_{fl})(x - x_{fl})$

http://www.pedram-payvandy.com



http://www.pedram-payvandy.com

Also, let ε_t, the fractional relative error, be the error associated with fl(x). Then

$$\frac{|xfl - x|}{|x|} = \varepsilon_t \quad where \quad \varepsilon_t \le \varepsilon$$

Machine epsilon, upper boundary

Rearranging, we get

$$|xfl - x| = \varepsilon_t |x|$$
$$xfl = x(\varepsilon_t + 1)$$

http://www.pedram-payvandy.com

- Case 1: Addition of x_1 and x_2 with associated errors ε_{t1} and ε_{t2} yields the following result:

$x_{fl1} = x_1(1+\varepsilon_{t1})$

$x_{fl2} = x_2(1+\varepsilon_{t2})$

$x_{fl1} + x_{fl2} = \varepsilon_{t1} x_1 + \varepsilon_{t2} x_2 + x_1 + x_2$

$$\frac{\varepsilon_t}{100\%} = \frac{(x_{fl_1}) + (x_{fl_2}) - (x_1 + x_2)}{x_1 + x_2} = \frac{\varepsilon_{t1} x_1 + \varepsilon_{t2} x_2}{x_1 + x_2}$$

•A large error could result from addition if x_1 and x_2 are almost equal magnitude but opposite sign, therefore one should avoid subtracting nearly equal numbers.

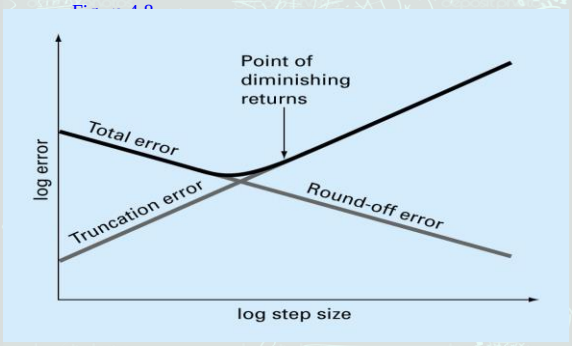
http://www.pedram-payvandy.com

- Since ε_{t1}, ε_{t2} are both small, the term $\varepsilon_{t1}\varepsilon_{t2}$ should be small relative to $\varepsilon_{t1}+\varepsilon_{t2}$. Thus the magnitude of the error associated with one multiplication or division step should be $\varepsilon_{t1}+\varepsilon_{t2}$.

  $\varepsilon_{t1} \leq \varepsilon$ (upper bound)

- Although error of one calculation may not be significant, if 100 calculations were done, the error is then approximately 100ε. The magnitude of error associated with a calculation is directly proportional to the number of multiplication steps.

- Overflow: Any number larger than the largest number that can be expressed on a computer will result in an overflow.
- Underflow (Hole) : Any positive number smaller than the smallest number that can be represented on a computer will result an underflow.
- Stable Algorithm: In extended calculations, it is likely that many round-offs will be made. Each of these plays the role of an input error for the remainder of the computation, impacting the eventual output. Algorithms for which the cumulative effect of all such errors are limited, so that a useful result is generated, are called "stable" algorithms. When accumulation is devastating and the solution is overwhelmed by the error, such algorithms are called unstable.



Solution Methods:
Analytical Solutions

Analytical solutions are available for special equations only.

Analytical solution of $a x^2 + b x + c = 0$

$$roots = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

No analytical solution is available for $x - e^{-x} = 0$

Graphical Illustration

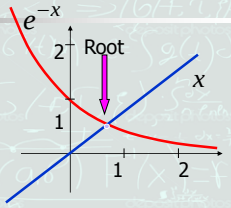
- Graphical illustration are useful to provide an initial guess to be used by other methods

Solve

$x = e^{-x}$

The root $\in [0,1]$

$root \approx 0.6$



Bracketing/Open Methods

- In bracketing methods, the method starts with an interval that contains the root and a procedure is used to obtain a smaller interval containing the root.
  - Examples of bracketing methods : Bisection method
- In the open methods, the method starts with one or more initial guess points. In each iteration a new guess of the root is obtained.

Solution Methods

Many methods are available to solve nonlinear equations

- Bisection Method
- Newton's Method
- Secant Method
- Fixed point iterations
- False position Method

→ These will be covered.

- Muller's Method
- Bairstow's Method
- .....

http://www.pedram-payvandy.com