**Numerical Analysis using Matlab**

Ahmed Aboemeira

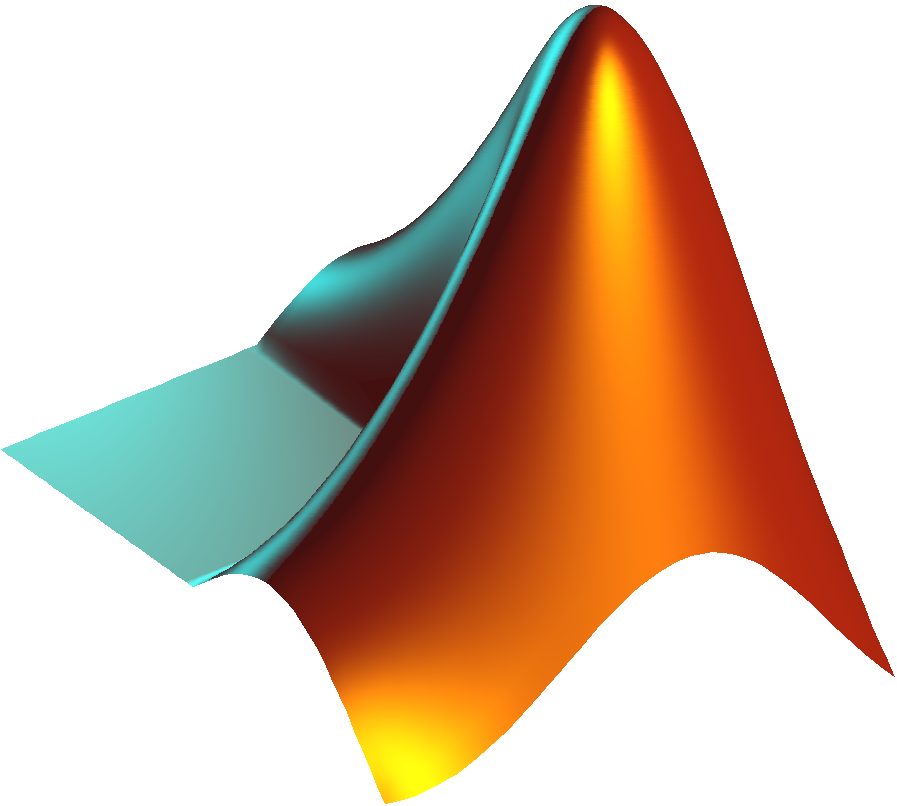
Omar Barakat

Omar Yousri

Karim Waguih ElAzzouni

Michael Wahid Azmy

Mohamed El Feky



Contents

[Linear Equations 4](#_Toc391405768)

[Gauss Elimination 4](#_Toc391405769)

[Code snippets 4](#_Toc391405770)

[Gauss-Jordan 5](#_Toc391405771)

[Code snippets 5](#_Toc391405772)

[LU-Decomposition 7](#_Toc391405773)

[Code snippets 7](#_Toc391405774)

[Root Finding 8](#_Toc391405775)

[Differentiation 8](#_Toc391405776)

[Code snippets 8](#_Toc391405777)

[Bisection 9](#_Toc391405778)

[Code snippets 10](#_Toc391405779)

[Analysis and Pitfalls 11](#_Toc391405780)

[False-position 13](#_Toc391405781)

[Code snippets 14](#_Toc391405782)

[Analysis and Pitfalls 15](#_Toc391405783)

[Fixed Point 15](#_Toc391405784)

[Code snippets 16](#_Toc391405785)

[Pitfalls 17](#_Toc391405786)

[Newton-Raphson 18](#_Toc391405787)

[Code snippets 18](#_Toc391405788)

[Analysis and Pitfalls 19](#_Toc391405789)

[Secant 21](#_Toc391405790)

[Code snippets 22](#_Toc391405791)

[analysis and Pitfalls 23](#_Toc391405792)

[GUI 23](#_Toc391405793)

[Sample Snapshots 24](#_Toc391405794)

[Introductory 24](#_Toc391405795)

[Root Finding 24](#_Toc391405796)

[Linear Equations 25](#_Toc391405797)

[Summary 27](#_Toc391405798)

[Comparison 27](#_Toc391405799)

[Data structures 29](#_Toc391405800)

[GUI 29](#_Toc391405801)

[UI TABLE 29](#_Toc391405802)

[Cell string array 29](#_Toc391405803)

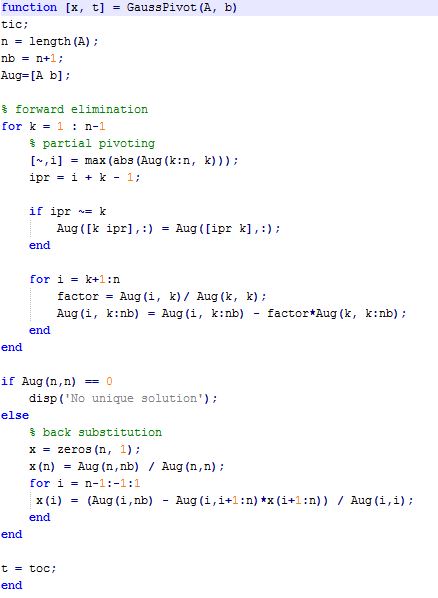
[Behavioral Analysis 30](#_Toc391405804)

[Case study 30](#_Toc391405805)

# Linear Equations

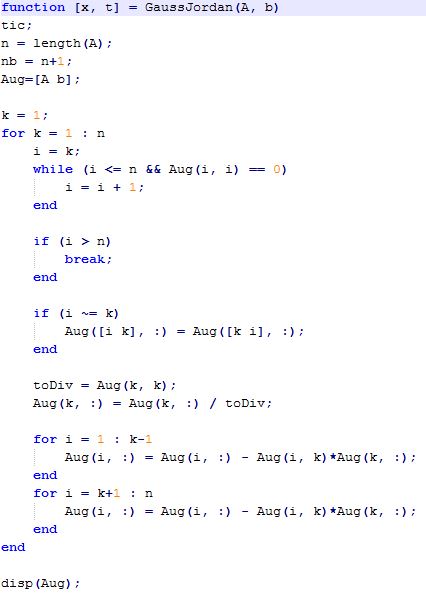
## Gauss Elimination

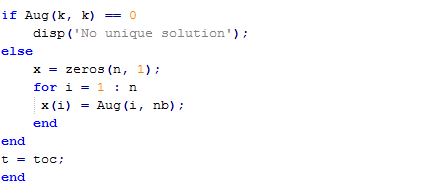
### Code snippets



## Gauss-Jordan

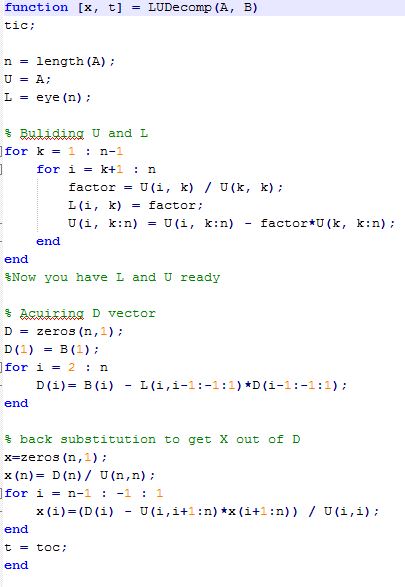
### Code snippets





## LU-Decomposition

### Code snippets

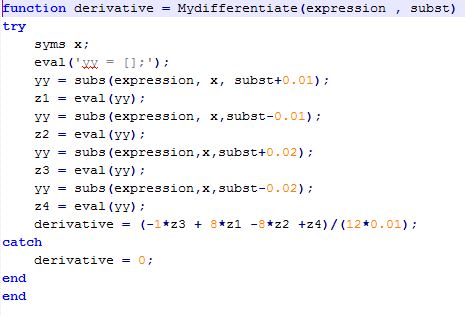


# Root Finding

## Differentiation

We implemented central difference differentiation method.

### Code snippets



## Bisection

The method is applicable for solving the equation f(x) = 0 for the real variable x, where f is a continuous function defined on an interval [a, b] and f(a) and f(b) have opposite signs. In this case, a and b are said to bracket a root since, by the intermediate value theorem, the continuous function f must have at least one root in the interval (a, b).

The algorithm works as follows:

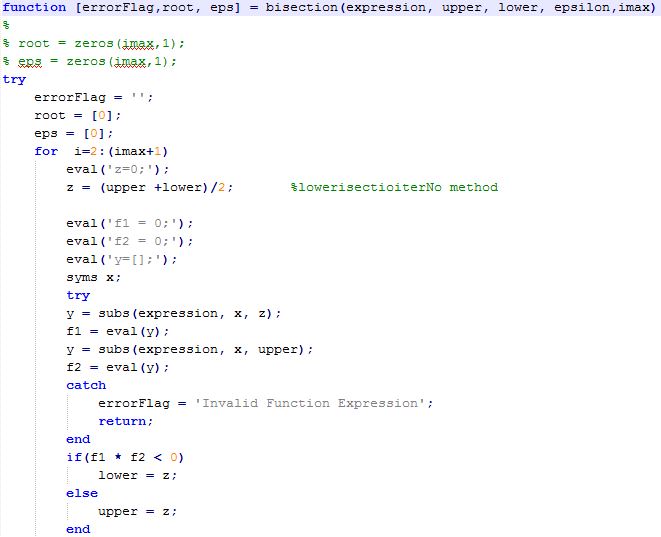
1. Calculate c, the midpoint of the interval, c = 0.5 \* (a + b).

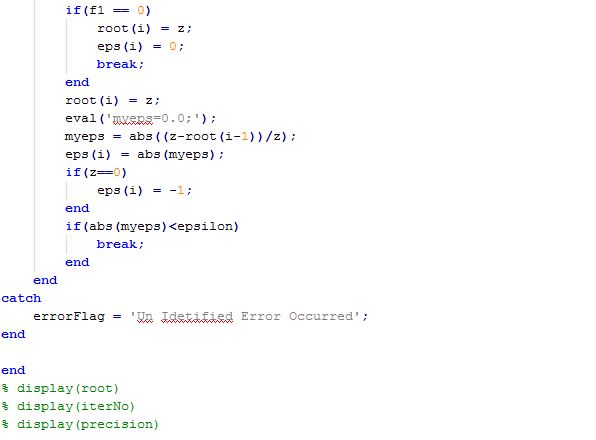
2. Calculate the function value at the midpoint, f(c).

3. If convergence is satisfactory (that is, a - c is sufficiently small, or f(c) is sufficiently small), return c and stop iterating.

4. Examine the sign of f(c) and replace either (a, f(a)) or (b, f(b)) with (c,f(c)) so that there is a zero crossing within the new interval.

### Code snippets





### Analysis and Pitfalls

The method is guaranteed to converge to a root of the function f if f is a continuous function on the interval [a, b] and f(a) and f(b) have opposite signs. The absolute error is halved at each step so the method converges linearly, which is comparatively slow.

Specifically, if c1 = (a+b)/2 is the midpoint of the initial interval, and cn is the midpoint of the interval in the nth step, then the difference between cn and a solution c is bounded by[7]

This formula can be used to determine in advance the number of iterations that the bisection method would need to converge to a root

|c_n-c|\le\frac{|b-a|}{2^n}.

Pros :

•Easy

•Always finds a root

•Number of iterations required to attain an absolute error can be computed a priori.

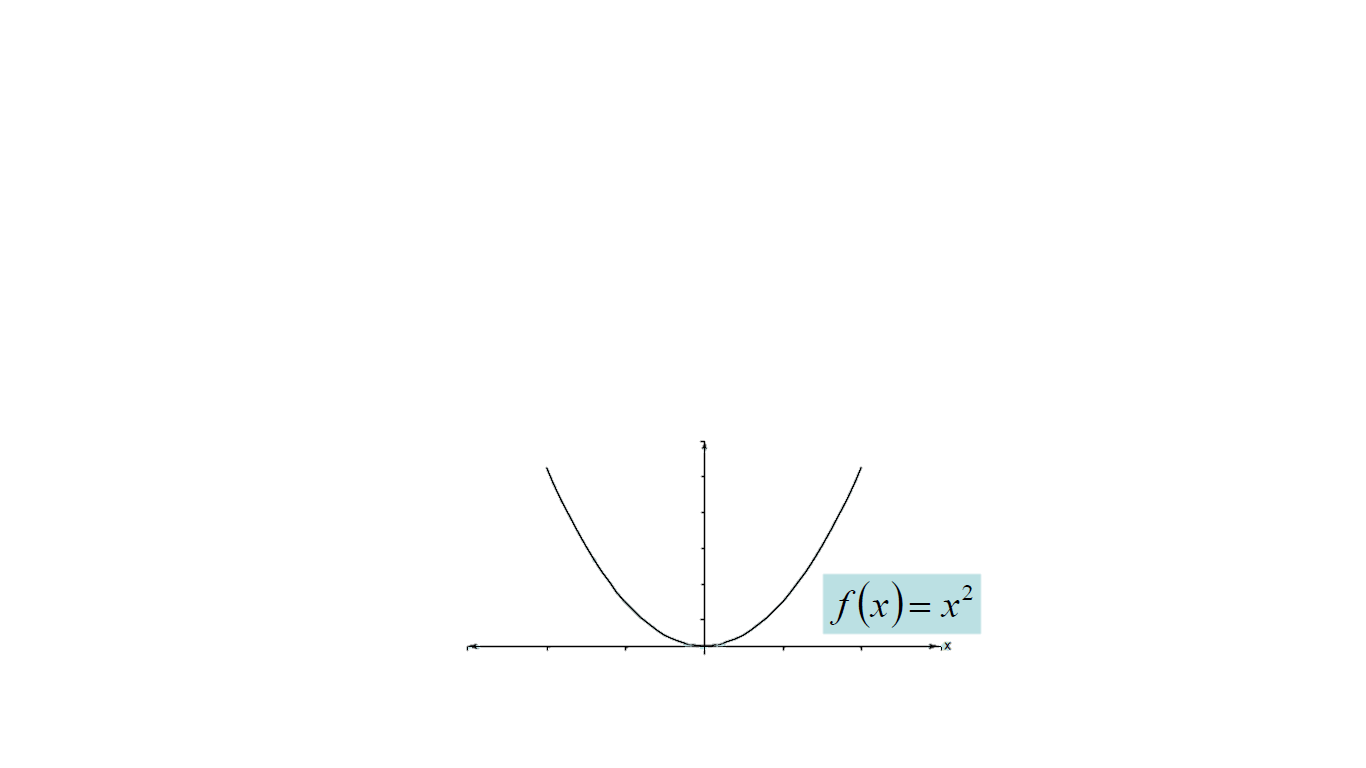
Cons:

•Relatively Slow

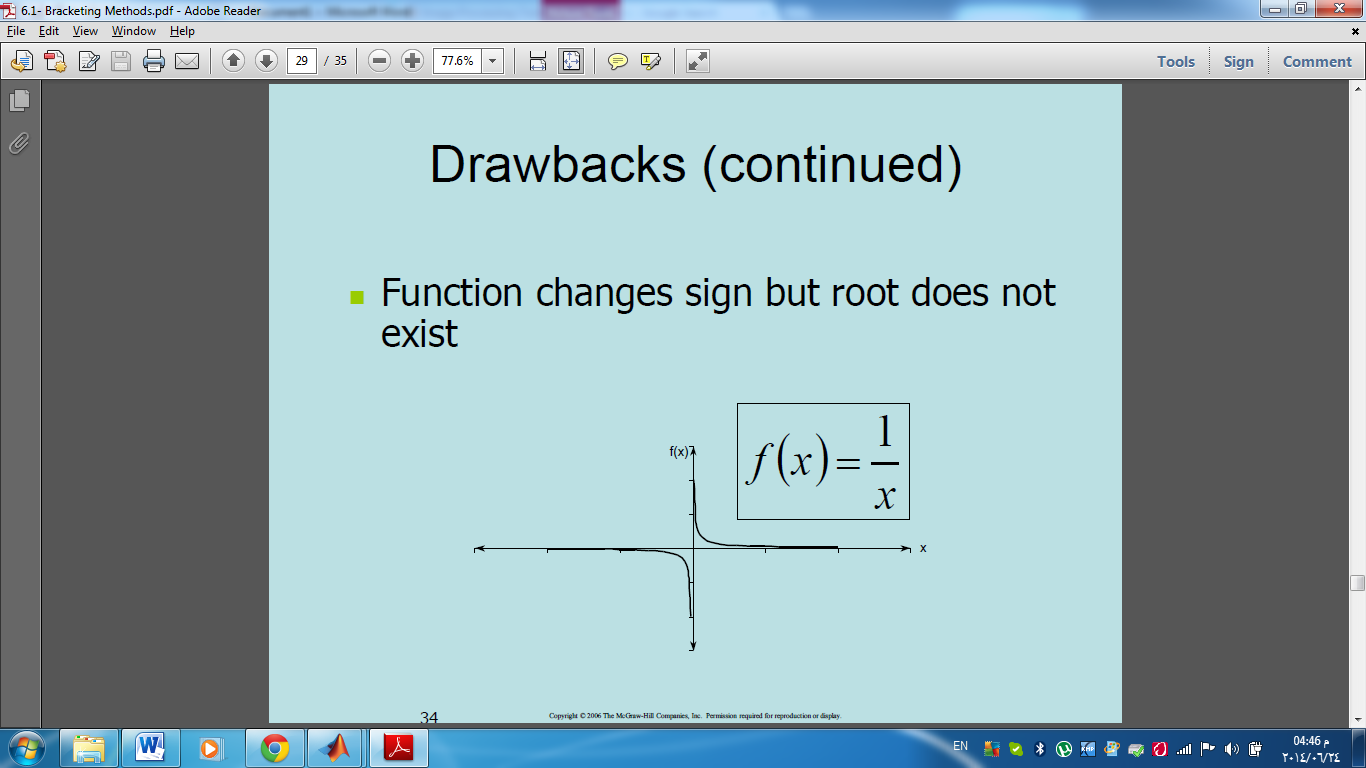
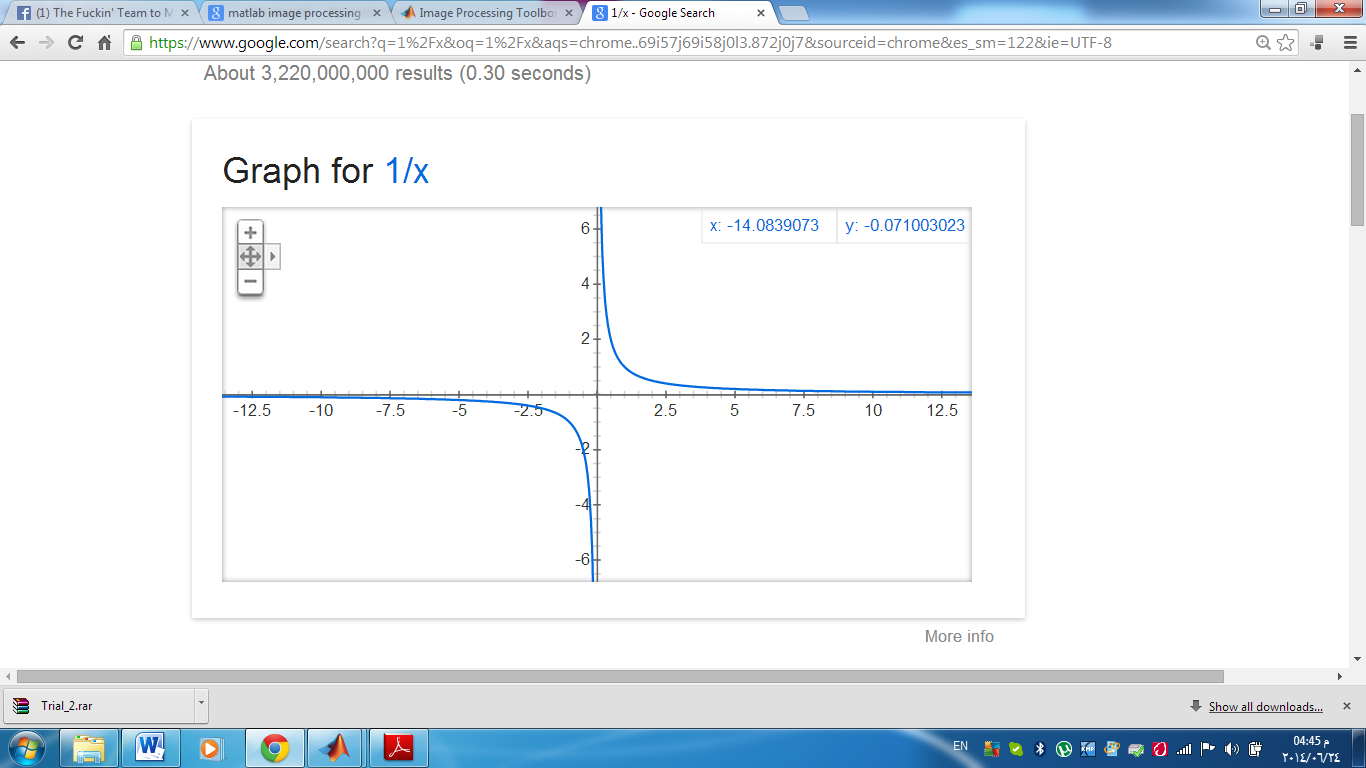
•Need to find initial guesses for xl and xu

•No account is taken of the fact that if f(xl) is closer to zero, it is likely that root is closer to xl .

•If a function f(x) is such that it just touches the x-axis it will be unable to find the lower and upper guesses

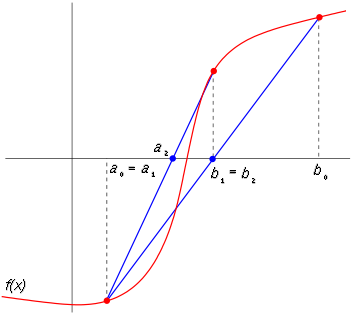


* Function changes sign but root does not exist it converge to a false root



## False-position

Like the bisection method, the false position method starts with two points*a*0 and *b*0 such that *f*(*a*0) and *f*(*b*0) are of opposite signs. The method proceeds by producing a sequence of shrinking intervals [*ak*, *bk*] that all contain a root of  *f*.



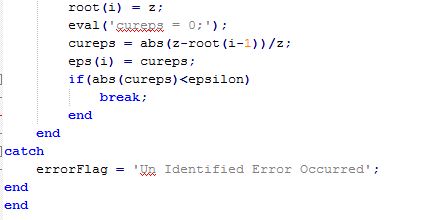
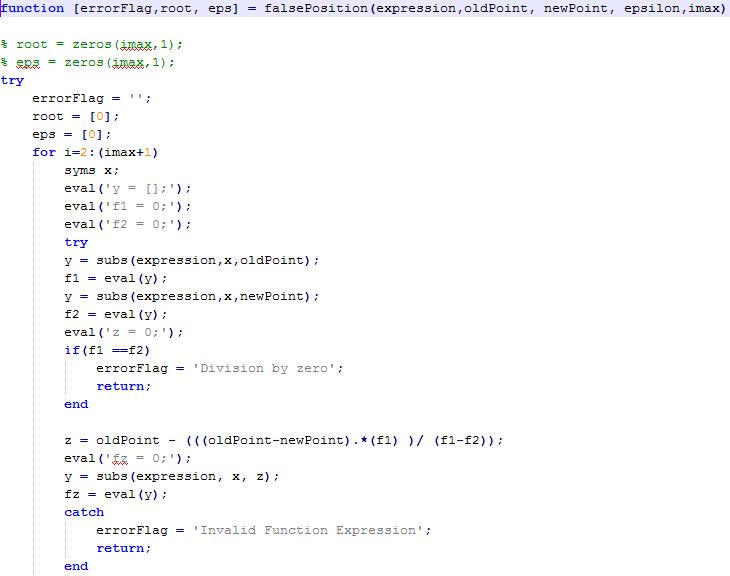
At iteration number *k*, the number:

 c_k = b_k-\frac{f(b_k) (b_k-a_k)}{f(b_k)-f(a_k)} 

is computed. As explained below, *ck* is the root of the secant line through (*ak*, f(*ak*)) and (*bk*, f(*bk*)). If f(*ak*) and f(*ck*) have the same sign, then we set *ak*+1 =*ck* and *bk*+1 = *bk*, otherwise we set *ak*+1 = *ak* and *bk*+1 = *ck*. This process is repeated until the root is approximated sufficiently well.

The above formula is also used in the secant method, but the secant method always retains the last two computed points, while the false position method retains two points which certainly bracket a root. On the other hand, the only difference between the false position method and the bisection method is that the latter uses *ck* = (*ak* + *bk*) / 2.

### Code snippets



### Analysis and Pitfalls

One of the pitfalls is the “one sided nature”. This occurs when only one of the two points moves along while the other remains in position (new point comes at the same side every time). if one of the end points becomes fixed, it can be shown that it is still an O(h) operation, that is, it is the same rate as the bisection method, usually faster, but possibly slower.

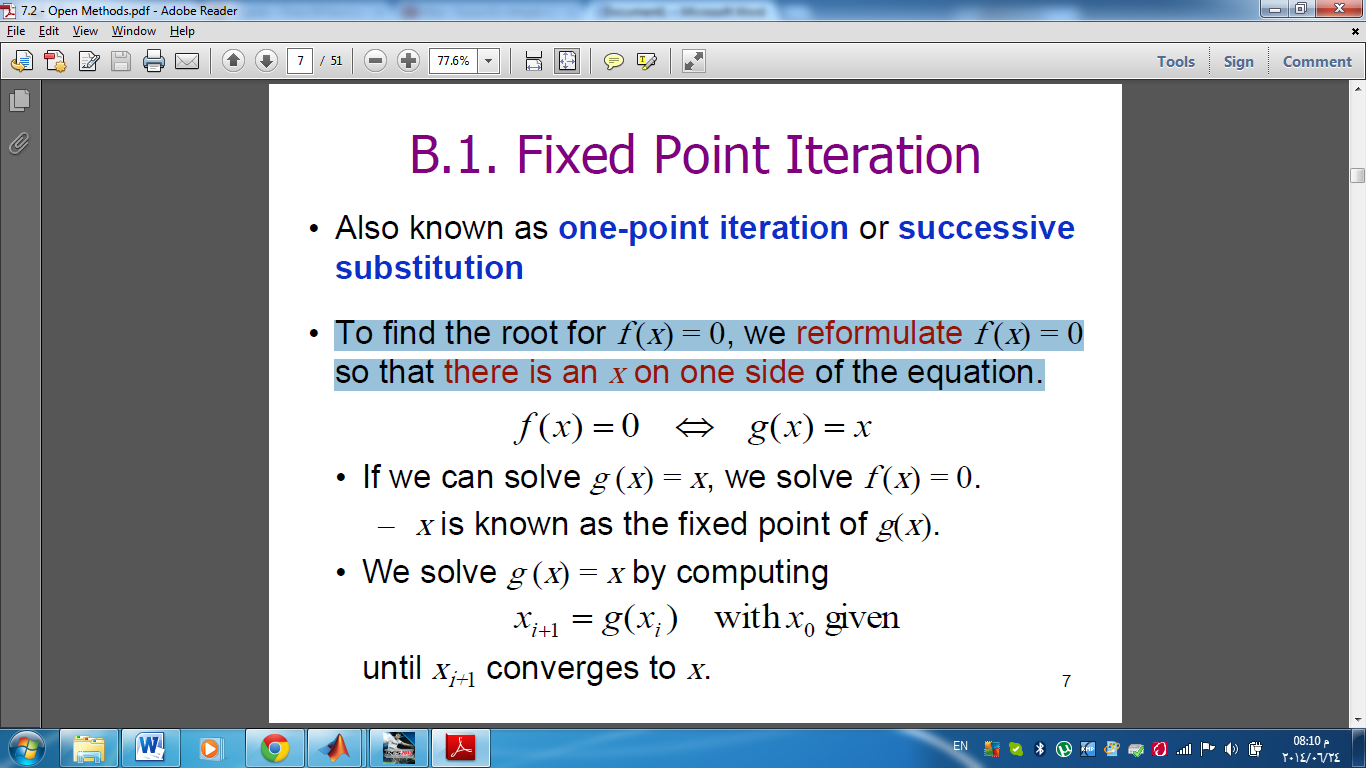


One way to mitigate the “one-sided” nature of the false position (i.e. the pitfall case) is to have the algorithm detect when one of the bounds is stuck.

If this occurs, then the original bisection formula xr = (xl + xu)/2 can be used.

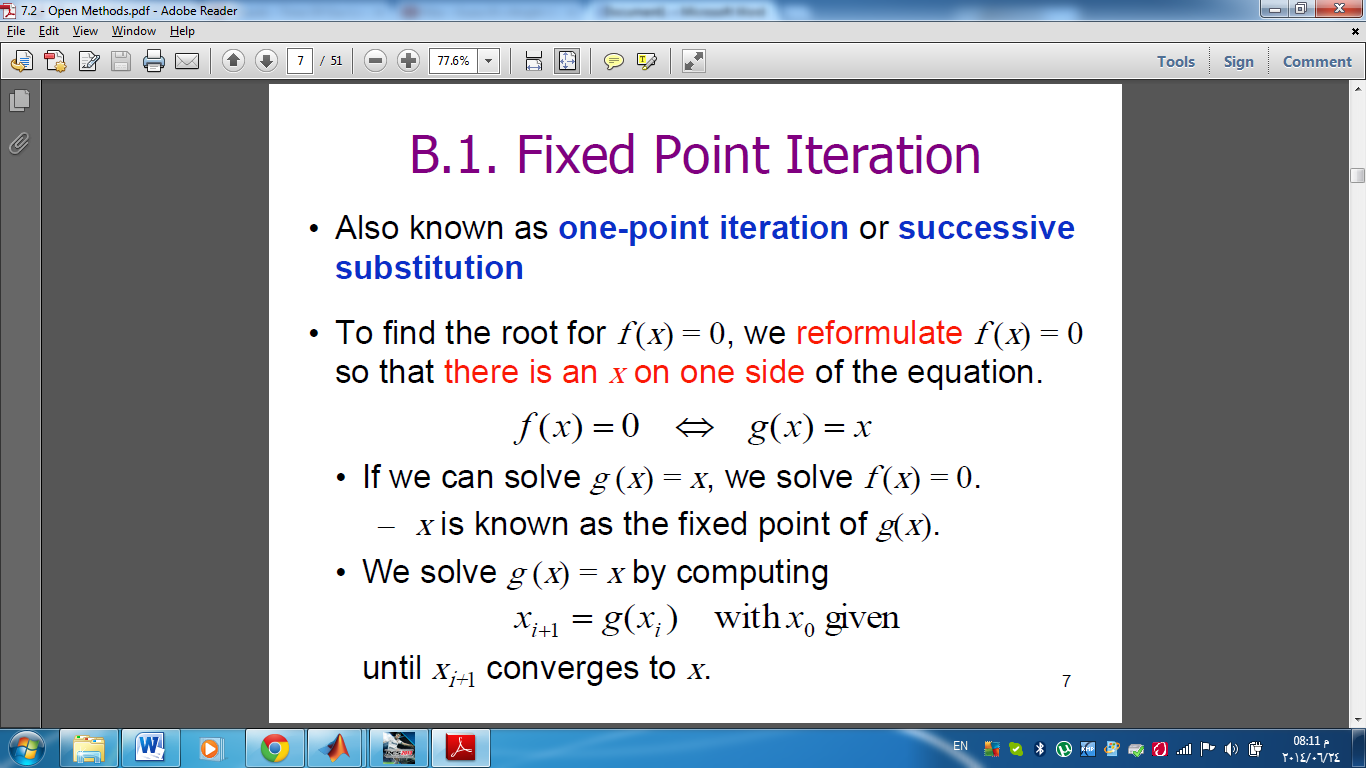
## Fixed Point

To find the root for f (x) = 0, we reformulate f (x) = 0 so that there is an x on one side of the equation.

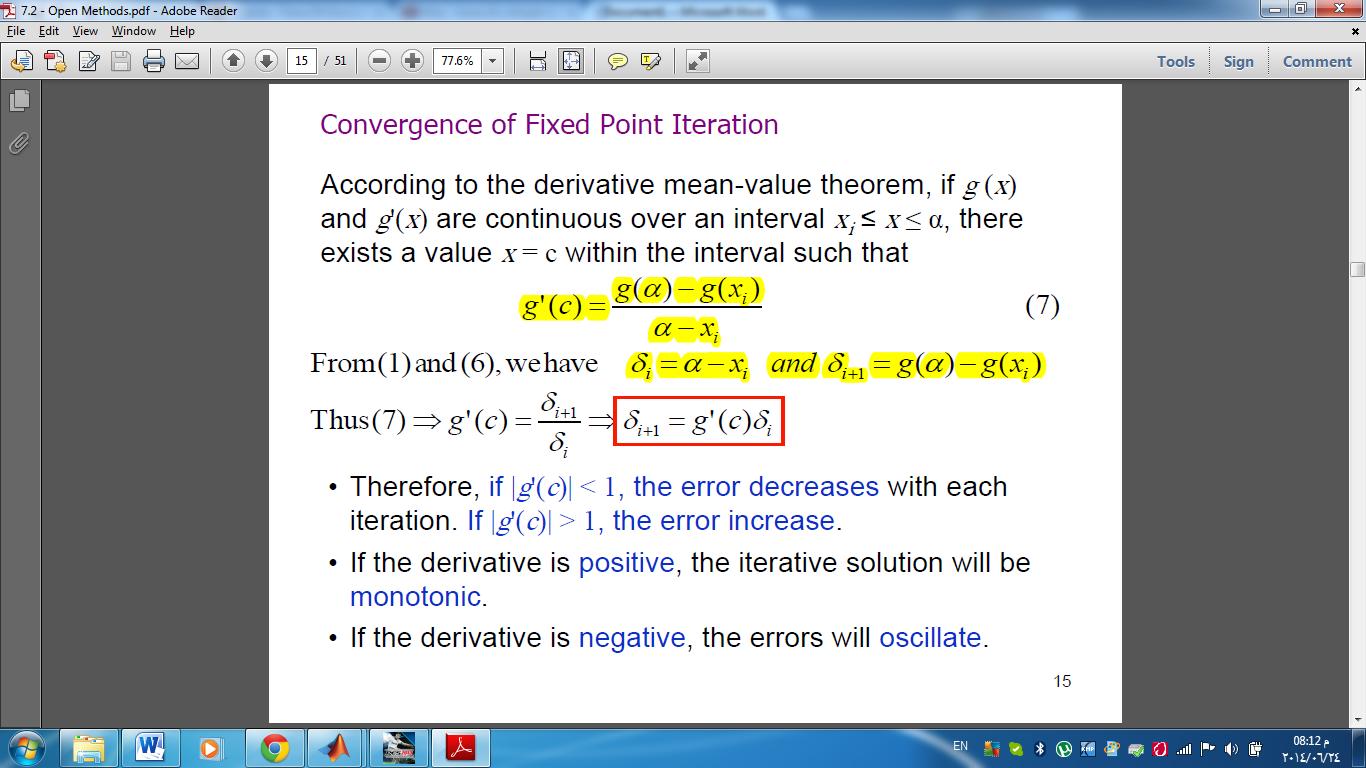


• If we can solve g (x) = x, we solve f (x) = 0. X is called fixed point of g(x)

• We solve g (x) = x by computing

 until xi+1 converges to x.

**Error analysis:**

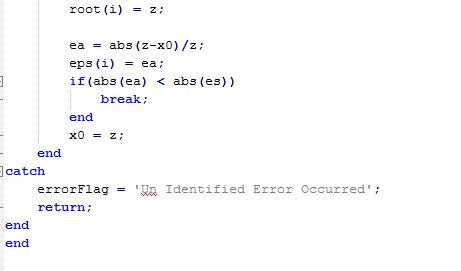
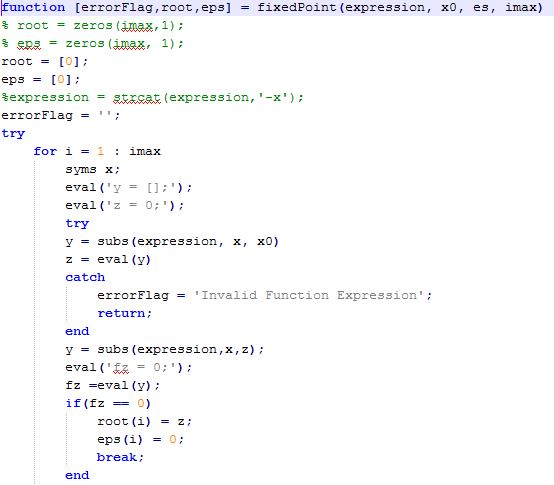


• Therefore, if |g'(c)| < 1, the error decreases with each iteration. If |g'(c)| > 1, the error increase.

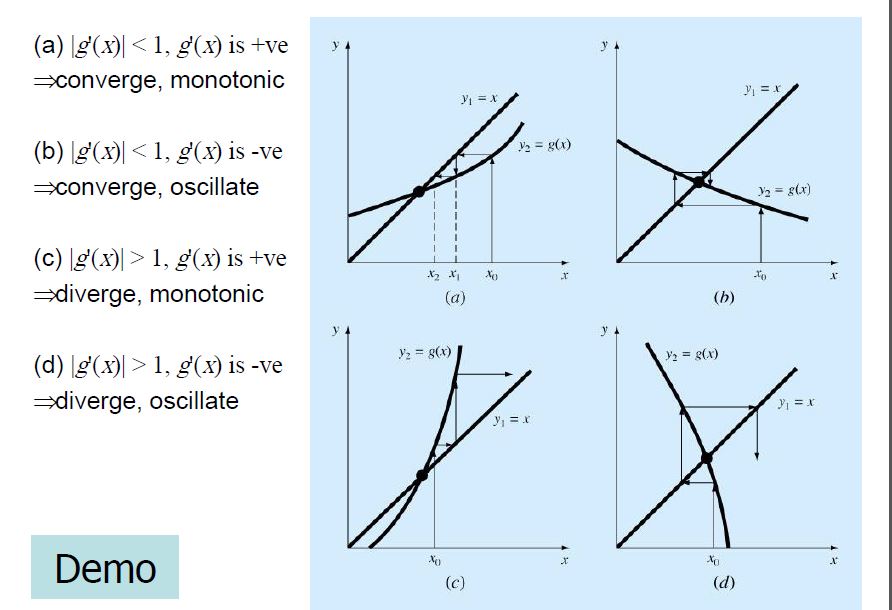
• If the derivative is positive, the iterative solution will be monotonic.

• If the derivative is negative, the errors will oscillate.

### Code snippets

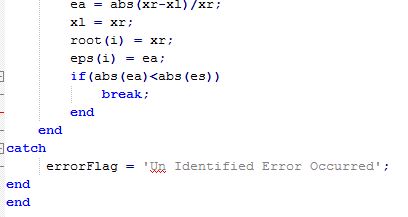
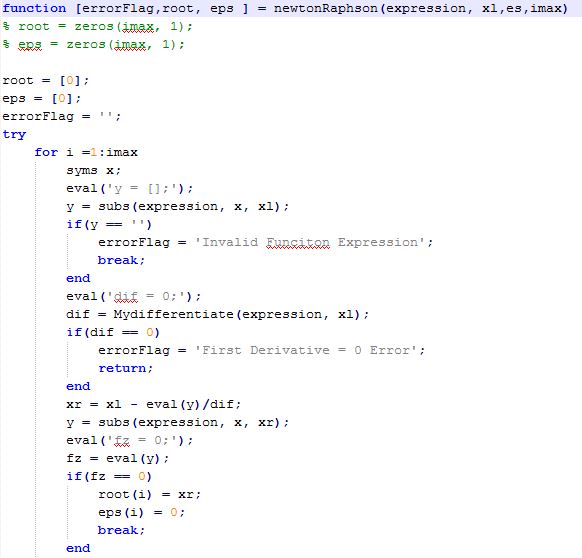


### Pitfalls

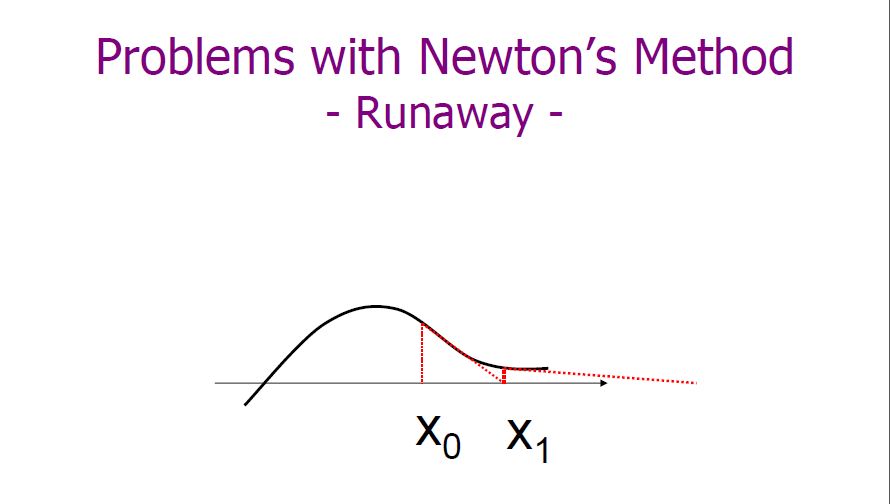
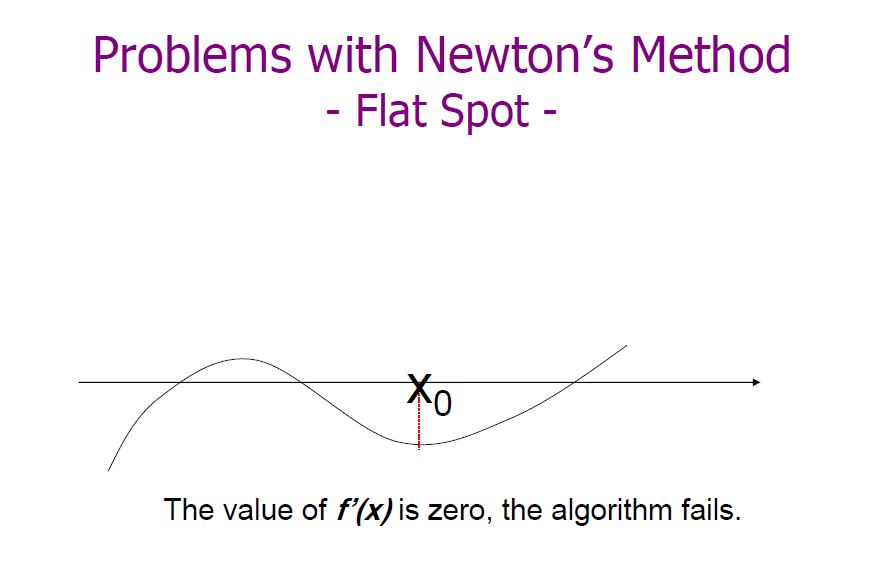
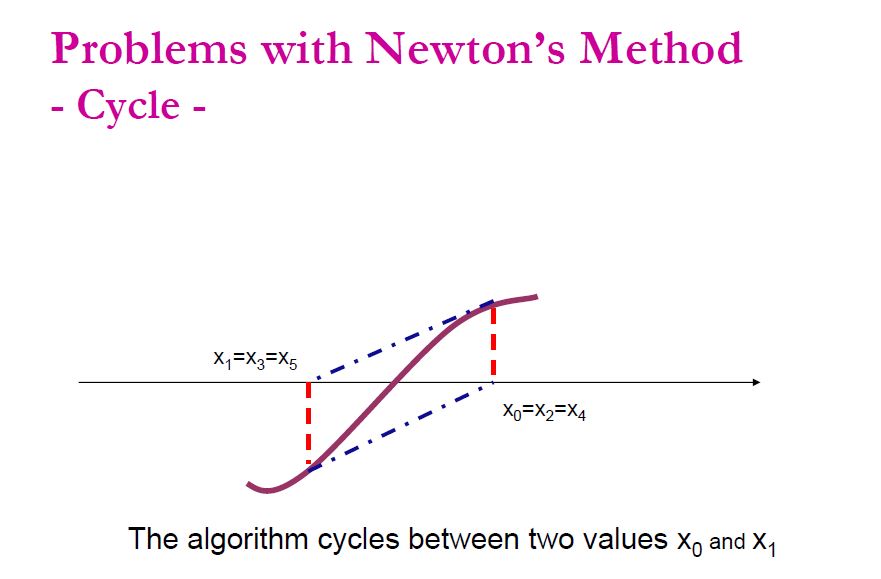
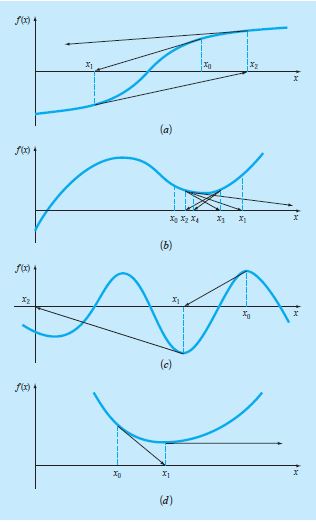


## Newton-Raphson

### Code snippets



### Analysis and Pitfalls



## Secant

The secant method is defined by the [recurrence relation](http://en.wikipedia.org/wiki/Recurrence_relation)


x_n
=x_{n-1}-f(x_{n-1})\frac{x_{n-1}-x_{n-2}}{f(x_{n-1})-f(x_{n-2})}
=\frac{x_{n-2}f(x_{n-1})-x_{n-1}f(x_{n-2})}{f(x_{n-1})-f(x_{n-2})}


As can be seen from the recurrence relation, the secant method requires two initial values, *x*0 and *x*1, which should ideally be chosen to lie close to the root.

The recurrence formula of the secant method can be derived from the formula for [Newton's method](http://en.wikipedia.org/wiki/Newton%27s_method)

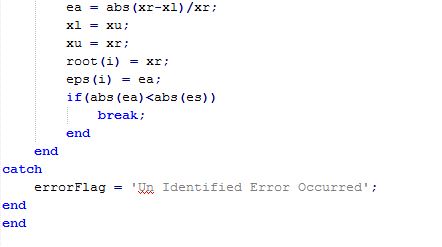
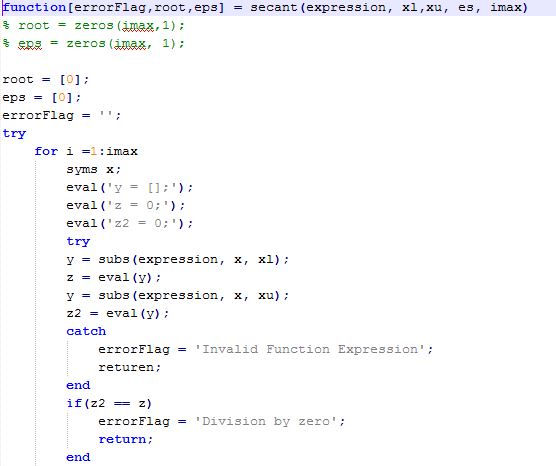
 x_{n} = x_{n-1} - \frac{f(x_{n-1})}{f^\prime(x_{n-1})} 

by using the [finite difference](http://en.wikipedia.org/wiki/Finite_difference) approximation

 f^\prime(x_{n-1}) \approx \frac{f(x_{n-1}) - f(x_{n-2})}{x_{n-1} - x_{n-2}}.

If we compare Newton's method with the secant method, we see that Newton's method converges faster (order 2 against α ≈ 1.6). However, Newton's method requires the evaluation of both f and its derivative f^\prime at every step, while the secant method only requires the evaluation of f.

### Code snippets



### analysis and Pitfalls

\* The convergence of the secant method is superlinear, but not quite [quadratic](http://en.wikipedia.org/wiki/Quadratic_convergence). (approximately 1.618).

\* If the initial values are not close enough to the root, then there is no guarantee that the secant method converges. There is no general definition of "close enough", but the criterion has to do with how "wiggly" the function is on the interval [~x_0,~x_1~]. For example, if f is differentiable on that interval and there is a point where f^\prime = 0 on the interval, then the algorithm may not converge.

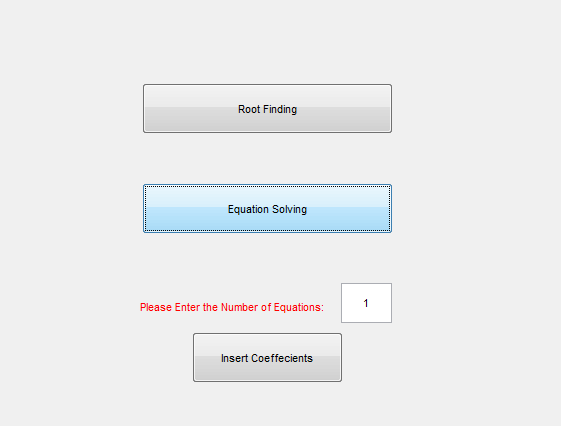
\* The secant method does not require that the root remain bracketed like the [false position method](http://en.wikipedia.org/wiki/Bisection_method) does, and hence it does not always converge.

# GUI

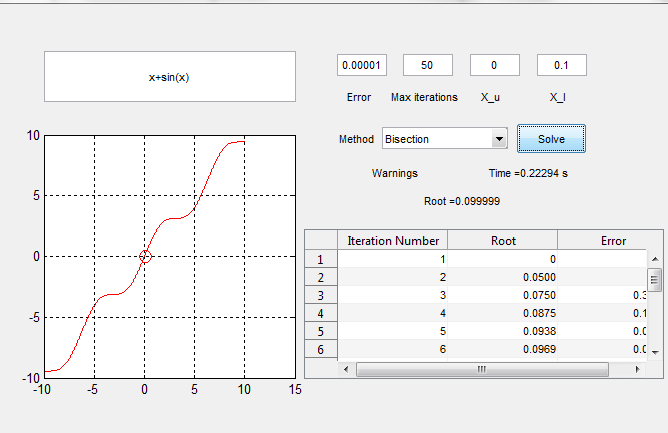
* Explanation here

## Sample Snapshots

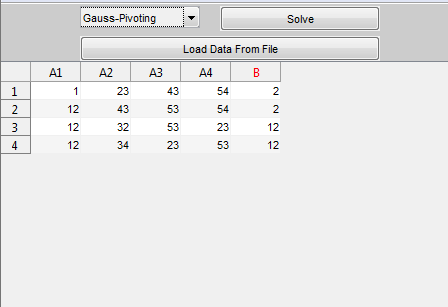
### Introductory

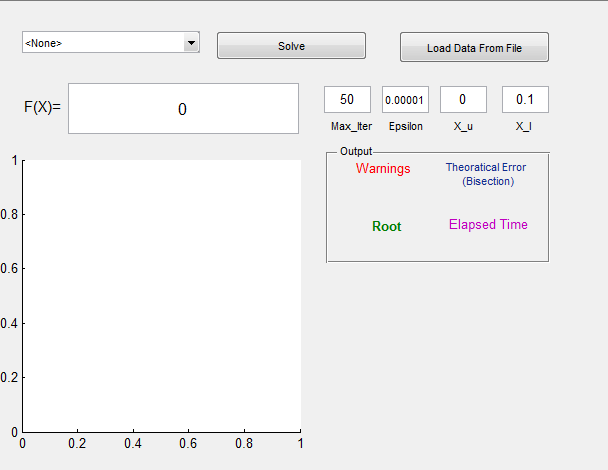


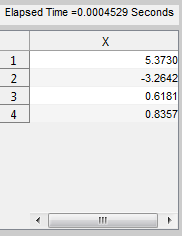
### Root Finding



### Linear Equations

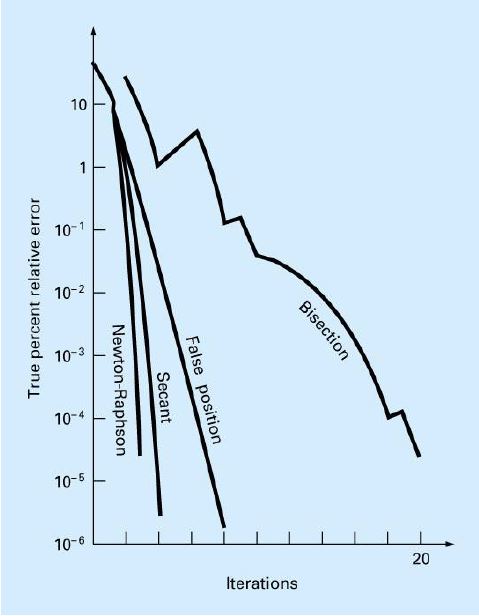
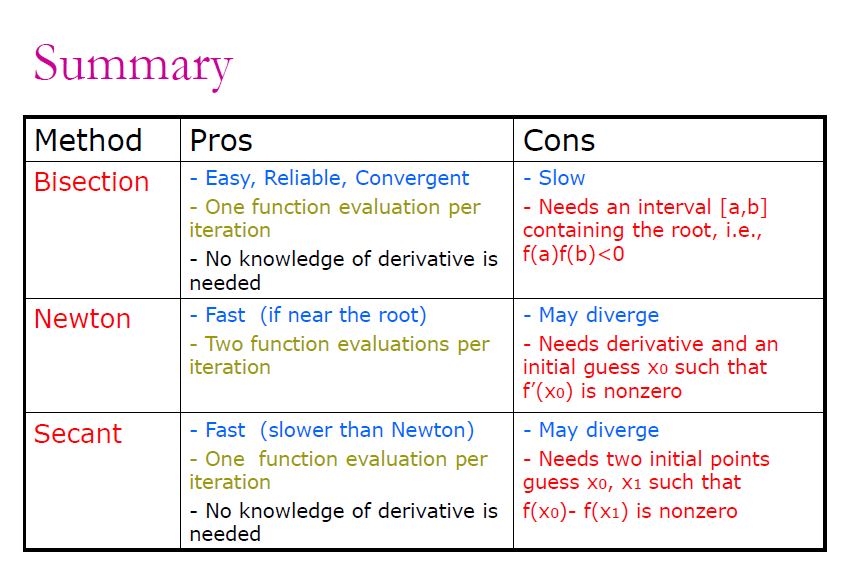






# Summary

## Comparison



# Data structures

## GUI

### UI TABLE

uitable creates an empty uitable object in the current figure window, using default property values. If no figure exists, a new figure window opens.

This data structure creates a 2-D graphic table GUI component, and provide properties to handle it.

It was pretty useful to us in showing the steps for the root finding method with its three columns {Iteration Number, Root, and Epsilon} and we used it also in Linear Algebra methods for the handling the coefficient matrix and showing the results as well

### Cell string array

Create cell array of strings from character array.

It was very useful while creating the names of the variables (Row names) in the Part B of the Assignment

# Behavioral Analysis

### Case study

