Introduction

# Lecture 01 – Introductory Remarks

## What is Computational Science?

* Also known as *Numerical Analysis* and *Scientific Computing*
* Possible definitions:
  + The study of algorithms for problems of continuous mathematics
  + Appropriate topics include the rigorous study of convergence of algorithms, their accuracy, their stability, and their computational complexity
  + The art of producing an approximate, numerical solution to a mathematical problem

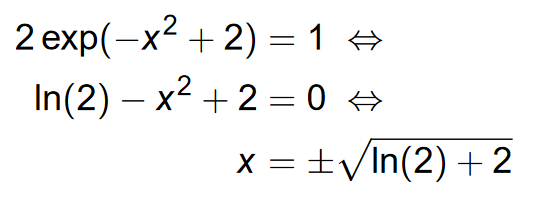
## Typical Applications

* In data mining, computational science is used to extra valuable information from large data sets (e.g. Google’s PageRank algorithm)
* In physics and engineering, computational science is used to model (simulate) physical processes.

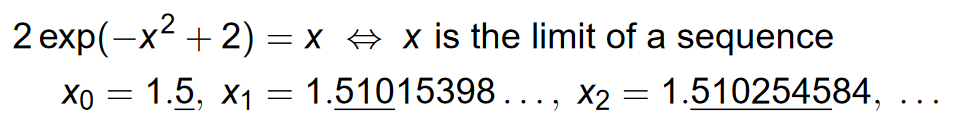
## Some Definitions

### Symbolic vs. Numerical Computation

* ***Symbolic Computation***: Application of the rules of algebra and mathematical identities to manipulate an expression involving symbols and variables.



* ***Numerical Computation***: Performing arithmetic on (approximate) numerical quantities. The result is a (set of) number(s).



* MAPLE and MATHEMATICA are **symbolic computation**programs
* PYTHON and MATLAB are best suited for **numerical computation**.
* Programming languages like FORTRAN and C are commonly used for numerical computation.
* Symbolic Computations are **exact**
* Numerical Computations are **approximate**
* ***Algorithm***: A thoroughly specified sequence of actions to accomplish some task (e.g. A recipe)
  + Most common ingredients: *loops* and *conditionals*
* *Computer*: A machine designed to manipulate symbols (usually 0’s and 1’s) and quickly and reliable execute algorithms involving large amounts of data and complicated repetitive procedures
* ***Hardware***: The physical, mechanical, magnetic, electronic, and electrical components making up a computer system.
* ***Software***: A computer program or a collection of computer programs designed to instruct a computer on how to perform one or more tasks.
  + Two important kinds of computer software:
    1. Application Software (e.g. Word Processors, Web Browsers, Databases)
    2. System Software (e.g. Hardware Drivers, Compilers, Editors)

### Operating Systems

* ***Operating System (OS)***: The foundation systems software that controls the execution of computer programs. It may provide various services (E.g. Operating the internet connection, Outputting to the screen, etc.)

### Programming Languages

* ***Programming Language***: An artificial language designed for issuing instructions to a computer.
* ***High-level programming language***: Problem-oriented, requiring little knowledge of the computer hardware or operating system on which it’ll be run (Java, Python, etc.)
* ***Low-level programming language***: Any language, which is meant to be closely related to, and easily translated into, machine language.
  + Allows tuning to make more efficient programs; however, they tend to be more difficult for programmers to master. (Assembly, Postscript, etc.)
* FORTRAN and C can be placed in between these two

### Compiler vs. Interpreter

* ***Compiler***: A computer program that reads high-level programming languages and **translates** it to a low-level (machine) language (also called object code)
  + *Object code is* typically put into a file called an **executable** that can be run at a later time
* ***Interpreter***: A computer program that reads input code in a high-level programming language and **immediately executes the input program**
  + A programming language executed by an interpreter is called an **interpreted programming language** (contrasted with a compiled programming language)
* *Interpreted code* runs slower than compiled code because the interpreter **reads and executes** each statement as it goes along whereas an *executable machine code* output from compiler only needs to **execute** each statement.
* When testing/developing code, interpreted language are usually easier
* Programs like MATLAB & MAPLE are **interpreters**. They allow development of scientific algorithms with lots of debugging help.
* When developing a complicated program, we often use an interpreted language first, then translate into a lower-level language like FORTRAN or C.

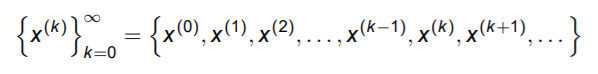
Solving nonlinear equations

# Lecture 02 – Introduction to Root Finding and Iterative Methods

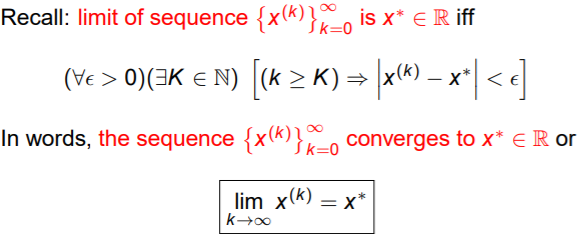
## Root Finding: Introduction

## Iterative Methods

* The two *iterative* methods we will study is:
  + Bisection
  + Newton’s method (an adaption called the *secant* method)
* Algorithms for solving f(x\*) = 0 are usually **iterative**
* Rule: x (k+1) = φ x(k) for k ≥ 0 is a **recurrence relation**
* Any iteration of x (k+1) = φ x(k) generates a **sequence**



* Another word to describe a convergence (of iterations) is the limit.



## First Method: Bisection

* The Bisection Method is an application of the Intermediate Value Theorem (IVT)
  + IVT Review: A continuous function (f) in a closed interval [a,b] where one endpoint is positive number and another endpoint is negative number. If this is true, then there exists a point that crosses the x-axis where f(x0) = 0
  + IVT doesn’t give you an idea on how to find the xto find that 0.

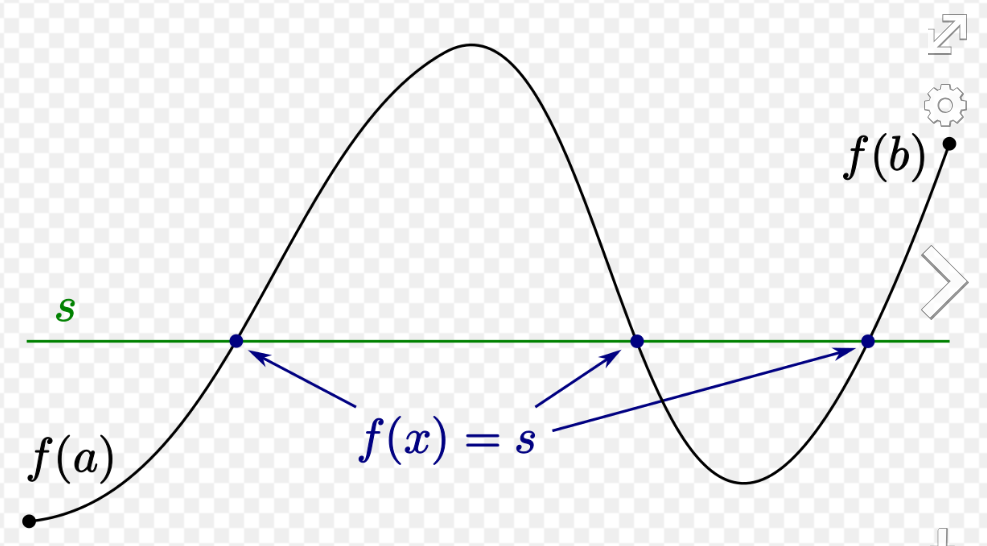
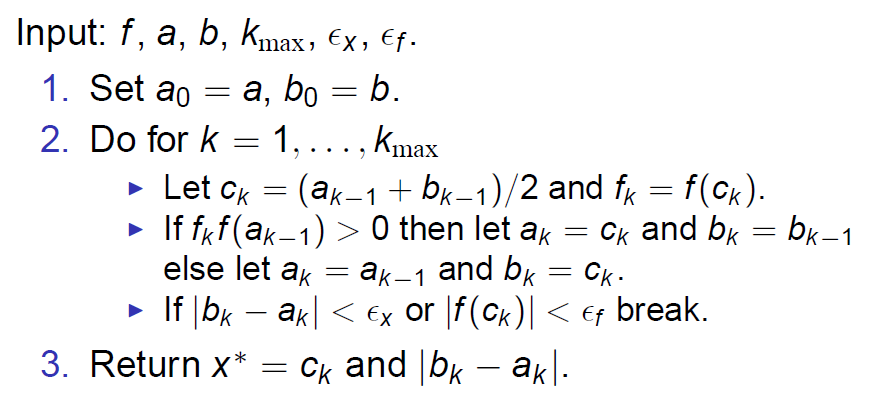


Figure 1. Intermediate Value Theorem

* The Bisection Method allows you to find the zeroes of a function in a closed interval



* f = function
* a = left end-point of the function
* b = right end-point of the function
* Ɛx = residual |f(x)=0| (X-length between the 2 points)
* Ɛf = error || (Y-length between the x-axis and the y-value of the coordinate)

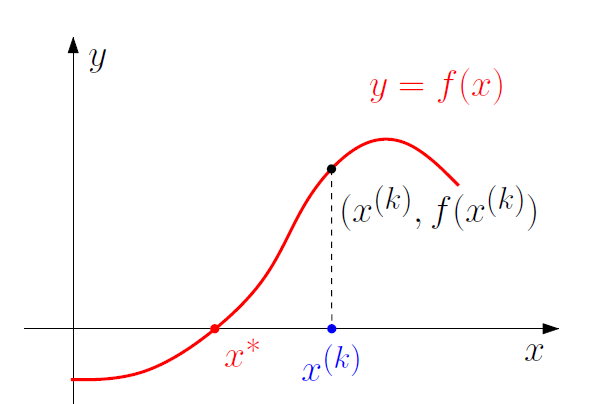
# Lecture 03 – Bisection and Newton Iteration

## Bisection

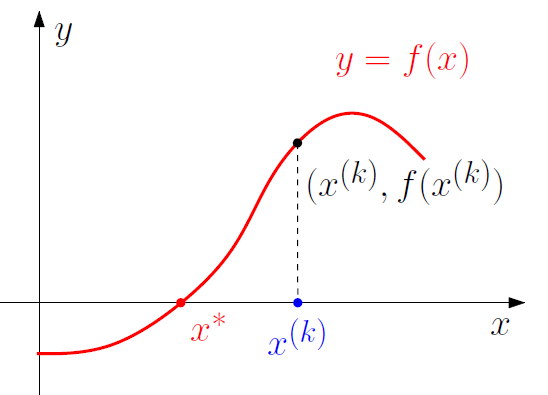
## Newton Iteration

* Derivation

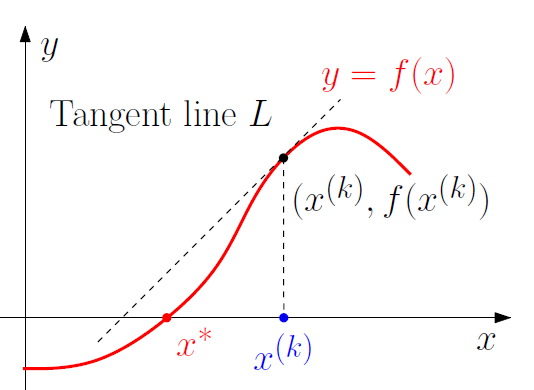
1. Start with x(k) (intended to approximate x\* such that f(x\*) = 0)



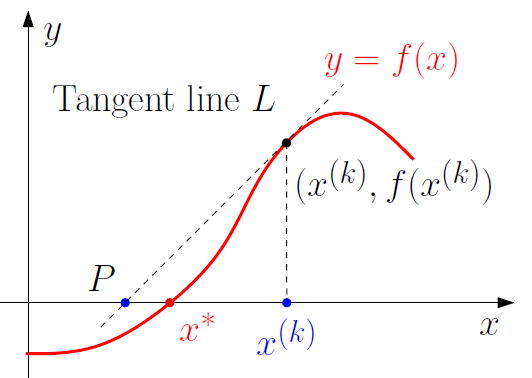
1. Evaluate f at x(k)



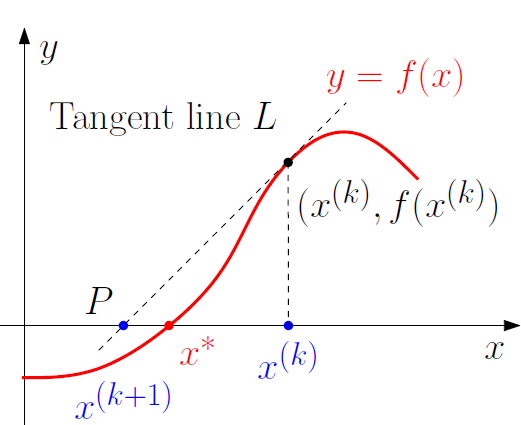
1. Extend tangent line *L* from (x(k), f(x(k)) (using f’ (x(k)))

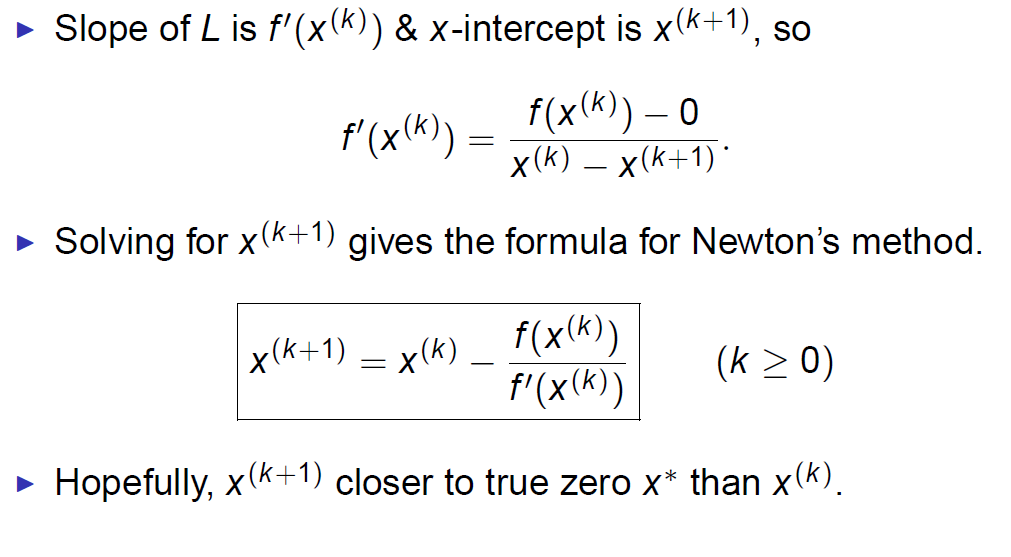


1. Follow *L* to *P* (where it cuts x-axis)



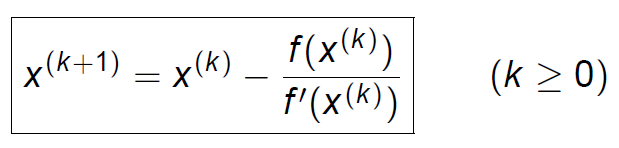
1. x(k+1) defined as x-coordinate of point *P*





## Newton’s Method

* Given an iterate x(k) approximating a zero of *f*, the next iterate is:



* Iterative procedure to locate zeros of *f*
* Requires initial iterate x(0) to start
* Near true zero x\* of *f*, iteration converges quickly

<https://www.youtube.com/watch?v=-5e2cULI3H8>

## Comparing the Two

## Secant Method

## Recursion

# Lecture 04 – Properties of Bisection and Newton Iteration, Newton-Raphson Iteration