DAMLAS Part 2
R, Lines, Tangents, Taylors Theorem, Roots of an equation

James G. Shanahan [1,2]

[1]Church and Duncan Group, *[2]iSchool UC Berkeley, CA*

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

Part 2 Lecture 2

June 12, 2016

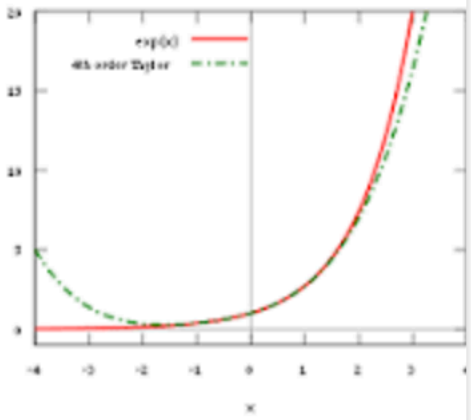
Lecture Outline

- **R Basics**
  - Most algorithms will be demonstrated with examples, code and homework problems
- **Lines, Tangents, Taylors Theorem, Roots of an equation**
- **Gradient Descent, Newton-Raphson**
- **Taylor Series: quadratic approximations**
- **Newton-Raphson quadratic convergence**
- **Multi-Dimensional Approximations (Planes)**
- **Directional Differentials, Total Differentials**
- **Vector plots, contour plots**
- **Gradient Descent**
  - Linear regression

Taylor's Theorem

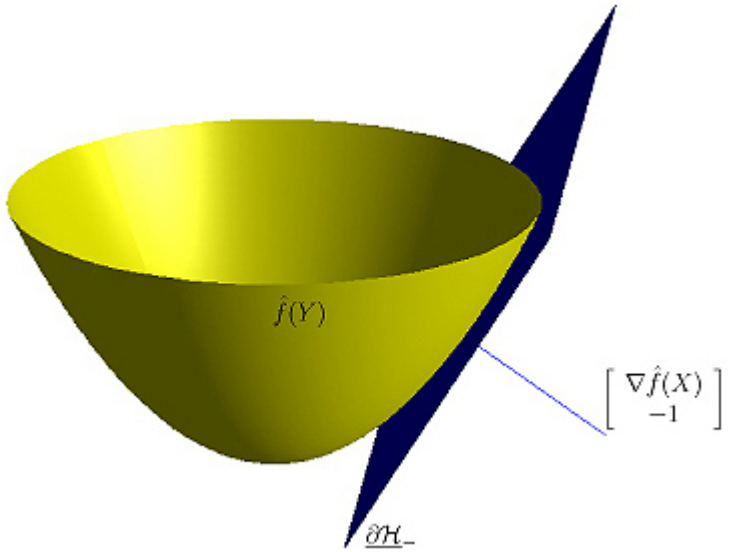
- ..

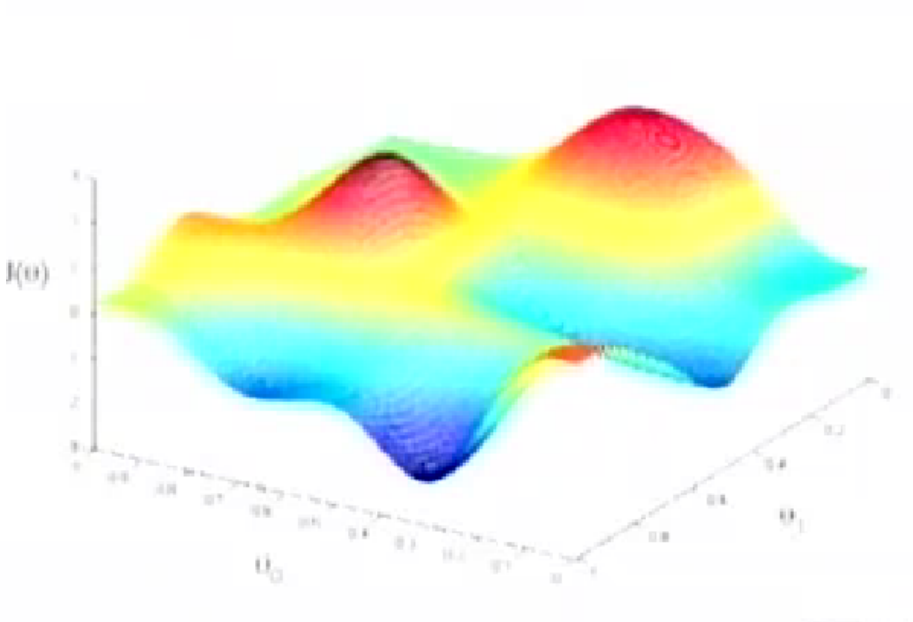
In calculus, **Taylor's theorem** gives an approximation of a k-times differentiable function around a given point by a k-th order **Taylor** polynomial.



Intuitive

- Theory
- Geometry
- Code





R

- The S statistical programming language and computing environment has become the de-facto standard among machine learners, statisticians, operation research (kitchen sink, gateway).
- The S language has two implementations: the commercial product S-PLUS, and the free, open-source R.
- Both are available for Windows and Unix/Linux systems; R, in addition, runs on Macintoshes.
- This lecture series will use R.

R: A History (from 1997 –

- In computing, R is a programming language and software environment for general purpose statistical and analytics computing and graphics.
- It is an implementation of the S programming language with lexical scoping semantics inspired by Scheme.
- R was created by Ross Ihaka and Robert Gentleman[2] at the University of Auckland, New Zealand, and is now developed by the *R Development Core Team*.
- It is named partly after the first names of the first two R authors (Robert Gentleman and Ross Ihaka), and partly as a play on the name of S.[3]
- The R language has become a de facto standard among statisticians/engineers for the development of statistical and engineering software, and is widely used for statistical software development and data analysis.

[Wikipedia]

Scripting languages

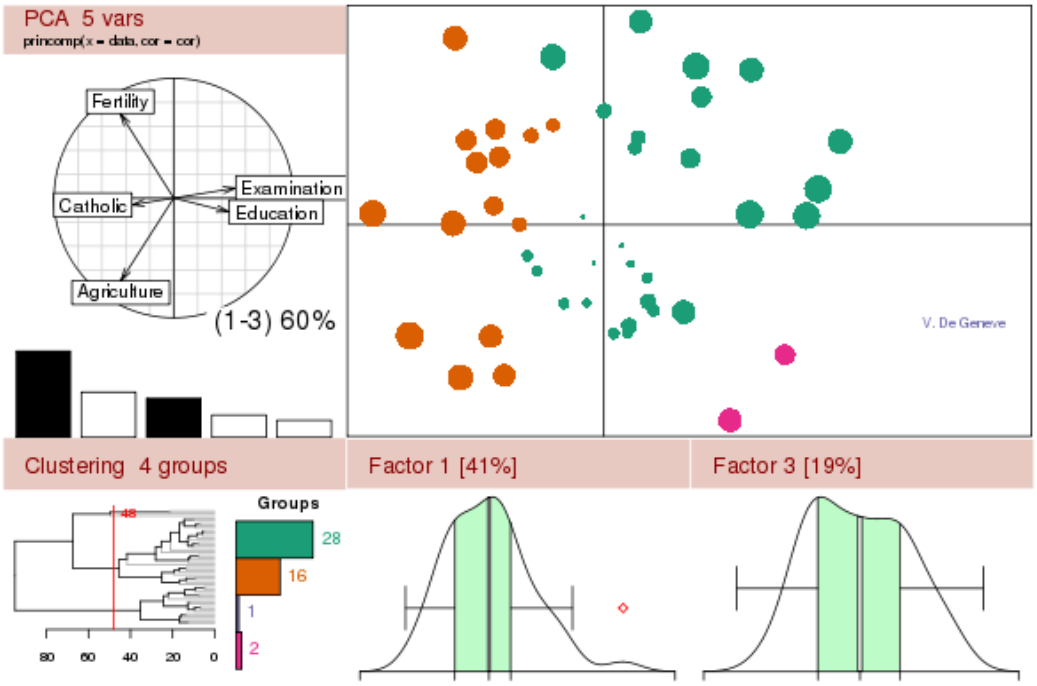
- **R has its own language**
  - R functionality has been made accessible from several scripting languages. E.g.,
    - Python (by the RPy[17] interface package)
    - Perl (by the Statistics::R[18] module).
- **Packages:**
  - Optimization packages are available
  - It can also be used as a general matrix calculation toolbox with comparable benchmark results to GNU Octave and its proprietary counterpart, MATLAB
  - An RWeka[9] interface has been added to the popular data mining software Weka which allows the capability to read/ write into the arff data format thus allowing the usage of data mining capabilities in Weka and statistical in R.

Software and Licenses

- **Available on Windows/Linux/Mac**
- **R is part of the GNU project.**
  - Its source code is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems.
- **R uses a command line interface, though several graphical user interfaces are available.**

R

- **Intro R website**
  - http://cran.r-project.org/doc/manuals/R-intro.html#Graphics
- **Nice examples**
  - http://www.mayin.org/ajayshah/KB/R/index.html



Online Resources

- **R Site with examples (French, Naïve Bayes)**
  - http://zoonek2.free.fr/UNIX/48_R/12.html#2
- **Intro R website**
  - http://cran.r-project.org/doc/manuals/R-intro.html#Graphics
- **Nice examples**
  - http://www.mayin.org/ajayshah/KB/R/index.html
- **Steward Book website**
  - http://www.stewartcalculus.com/media/9_inside_chapters.php?subaction=showfull&id=1090822711&archive=&start_from=&ucat=2&show_cat=2
- **Taylor's page at Stanford**
  - http://www-stat.stanford.edu/~jtaylo//
- **Contour plots**
  - http://online.redwoods.cc.ca.us/instruct/darnold/MULTCALC/grad/grad.pdf
- **Fox's Book**
  - 2009, http://socserv.socsci.mcmaster.ca/jfox/Courses/R-programming/index.html
  - 2008, http://socserv.mcmaster.ca/jfox/Courses/R-course/index.html

Online Resources

- **R** http://www.r-project.org/
- **R books online**
  - http://www.math.ccu.edu.tw/~yshih/Rrefs/Rlecturenotes.pdf
  - http://www.cran.r-project.org/doc/contrib/Faraway-PRA.pdf
- **STATISTICS: AN INTRODUCTION USING R (Crawley)**
  - http://www.bio.ic.ac.uk/research/crawley/statistics/exercises.htm
- **Resources at Stanford**
  - http://www-stat.stanford.edu/~jtaylo/courses/stats191/R/logistic/flu.R
  - http://www-stat.stanford.edu/~jtaylo/courses/stats191/R/logistic/fluout.html

Installing R and an Editor

- **Installing an editor: EditPlus (for Windows)**
  - Useful Editor on Windows (30 temporary license)
  - http://www.brothersoft.com/download-editplus-16751.html
- **Installing R (Windows, also on Linux and Mac)**
  - Click here to download an installer EXE: http://cran.r-project.org/bin/windows/base/R-2.10.0-win32.exe

    The distribution is distributed as a 30Mb installer R-2.10.0-win32.exe.

    Just run this for a Windows-XP style installer. It contains all the R components, and you can select what want installed.

    For more details, including command-line options for the installer and how to uninstall, see the rw-FAQ (http://cran.r-project.org/bin/windows/base/rw-FAQ.html).

Required Files And Things to do

When you see example.ABC () check R script file

- **Examples available as Functions in script files**
  - Download JimisMLCourse_2.R

```
example.learnLSUsingClosedFormSolution = function() {
    dataEx1= matrix(c(……),
    byrow=TRUE,
    ncol = 2)
    colnames(dataEx1)=c("time", "temperature")

    designMatrix=as.matrix(dataEx1[,1])  #input variable data
    X=designMatrix=cbind(1, designMatrix)  #append a
   constant 1 for bias term
…….
    y=targetValues=as.matrix(dataEx1[,2]);
```

Install R Packages

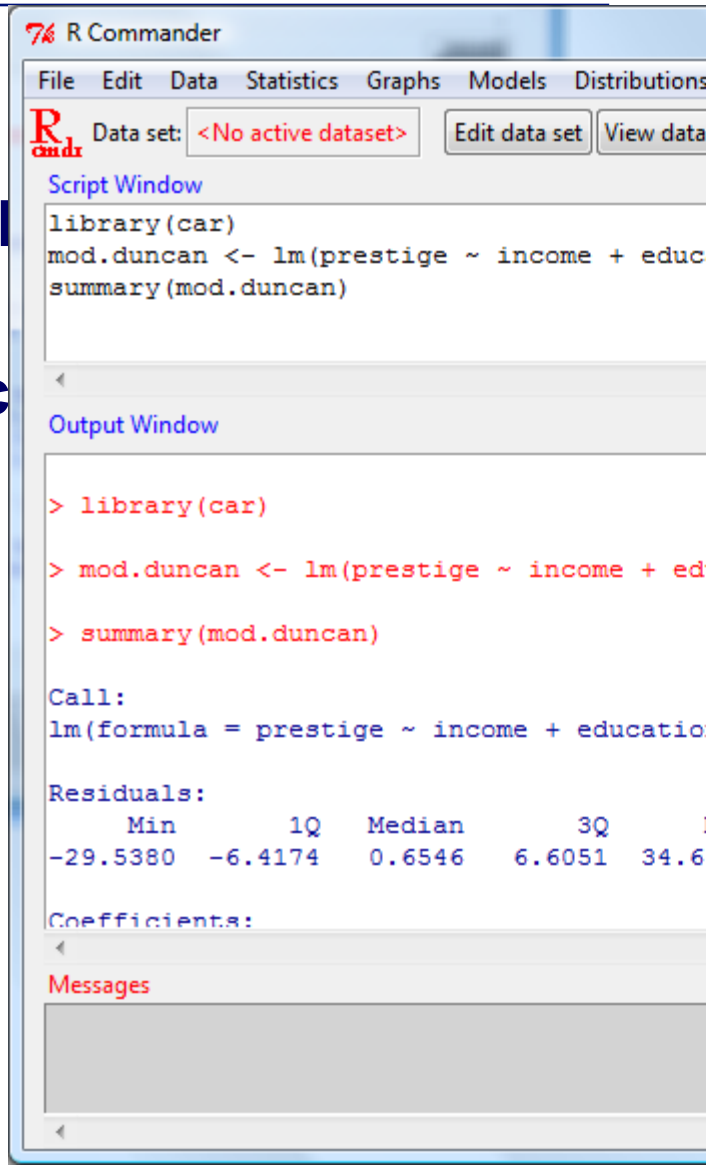
See example.setupPackages() in course R script file

- **Install via command line or via the menu**
  - install.packages("Rcmdr", dependencies=TRUE)
  - install.packages('e1071')
  - Install.packages (“MASS“)
  - Install.packages(“tree“)
  - Install.packages("Rcmdr“)
  - Via MENU
    - Packages->install; then select a repository and the package needed to be installed

- **To use a library just type**
  - library('Rcmdr')
  - library('e1071')

```
# Rcmdr
# http://socserv.mcmaster.ca/jfox/Misc/l
    installation-notes.html
# install.packages("Rcmdr", dependenc
library(Rcmdr)
```

```
library(car)
mod.duncan <- lm(prestige ~
income + education,
data=Duncan)
summary(mod.duncan)
```



R Resources

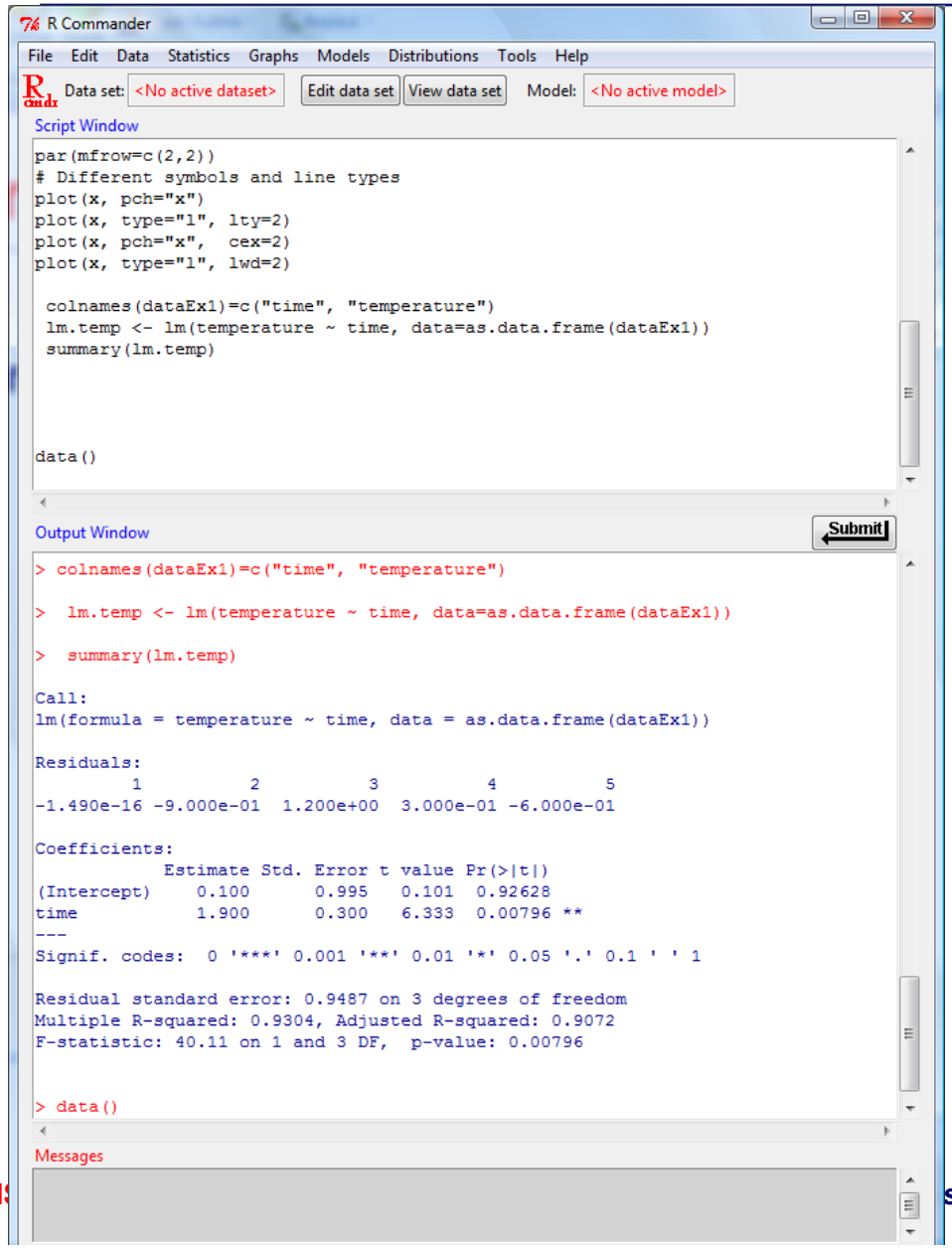
- http://cran.r-project.org/doc/contrib/Lemon-kickstart/index.html

Rcmdr: a tool for demos and teaching

```
# Rcmdr
# http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/
    installation-notes.html
#  install.packages("Rcmdr", dependencies=TRUE)
library(Rcmdr)
```

```
library(car)
mod.duncan <- lm(prestige ~
income + education,
data=Duncan)
summary(mod.duncan)
```

rcmdr()



Demonstrations in R

RCommander

R Commander

File Edit Data Statistics Graphs Models Distributions Tools Help

Data set: Duncan | Edit data set | View data set | Model: <No active model>

Script Window

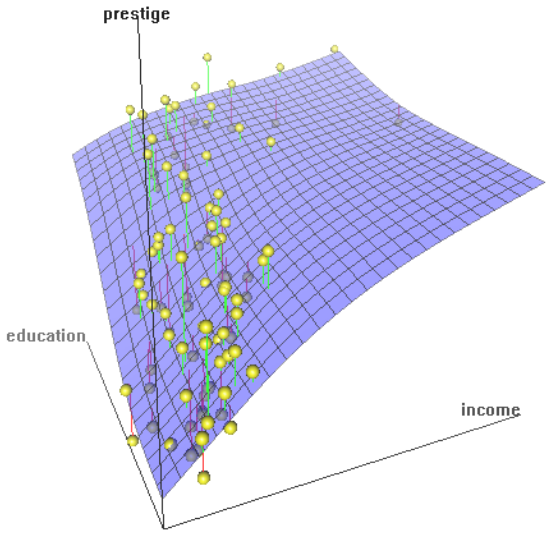
```
data()
Hist(Duncan$education, scale="frequency", breaks="Sturges", col="darkgray")
.Table <- table(Duncan$type)
.Table  # counts for type
100*.Table/sum(.Table)  # percentages for type
remove(.Table)
boxplot(Duncan$education, ylab="education")
{boxplot(income~type, ylab="income", xlab="type", data=Duncan)
} {boxplot(prestige~type, ylab="prestige", xlab="type", data=Duncan)
}
scatter3d(Duncan$education, Duncan$income, Duncan$prestige, fit="linear",
  residuals=TRUE, bg="white", axis.scales=TRUE, grid=TRUE, ellipsoid=FALSE,
  xlab="education", ylab="income", zlab="prestige")
```

Output Window — Submit

```
> plot(lm.temp)

> Hist(Duncan$education, scale="frequency", breaks="Sturges", col="darkgray")

> .Table <- table(Duncan$type)

> .Table  # counts for type

 bc prof   wc
 21   18    6

> 100*.Table/sum(.Table)  # percentages for type

      bc     prof       wc
46.66667 40.00000 13.33333

> remove(.Table)

> boxplot(Duncan$education, ylab="education")

> boxplot(income~type, ylab="income", xlab="type", data=Duncan)

> scatter3d(Duncan$education, Duncan$income, Duncan$prestige, fit="linear",
+   residuals=TRUE, bg="white", axis.scales=TRUE, grid=TRUE, ellipsoid=FALSE,
+   xlab="education", ylab="income", zlab="prestige")
```

Messages

5 Adobe ... | 6 R for W... | 3 Interne... | 3 Windo... | 2 Micros... | EditPlus - [...



RCmdr Output

See example.BocPlotsAnd3DScatterPlots()

```
example.BocPlotsAnd3DScatterPlots = function() {

# data()
  Duncan <- read.table("http://socserv.mcmaster.ca/jfox/Courses/R-course/Duncan.txt")
  Hist(Duncan$education, scale="frequency", breaks="Sturges", col="darkgray")
  .Table <- table(Duncan$type)
  .Table  # counts for type
  100*.Table/sum(.Table)  # percentages for type
  remove(.Table)
  boxplot(Duncan$education, ylab="education")
  #plot income as a function of job type
  boxplot(income~type, ylab="income", xlab="type", data=Duncan)
  #plot prestige as a function of job type
  boxplot(prestige~type, ylab="prestige", xlab="type", data=Duncan)

  library(Rcmdr)
  # 3Dplot income as function of eduction and prestige
  # with residuals
  scatter3d(Duncan$education, Duncan$income, Duncan$prestige, fit="linear",
    residuals=TRUE, bg="white", axis.scales=TRUE, grid=TRUE, ellipsoid=FALSE,
    xlab="education", ylab="income", zlab="prestige")
```

Built in Optimization Tools in R

- **?optim**
  - General-purpose optimization based on Nelder–Mead, quasi-Newton and conjugate-gradient algorithms. It includes an option for box-constrained optimization and simulated annealing.
  - Usage

    optim(par, fn, gr = NULL, ..., method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN"), lower = -Inf, upper = Inf, control = list(), hessian = FALSE)
- **?constrOptim**
  - Minimise a function subject to linear inequality constraints using an adaptive barrier algorithm.

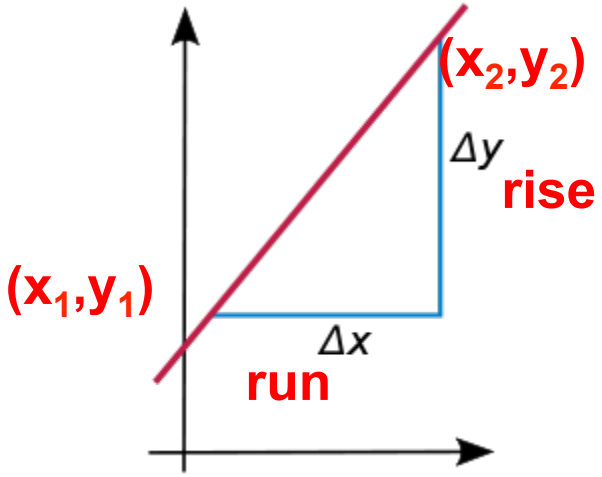
Lecture Outline

- **R Basics**
  - Most algorithms will be demonstrated with examples, code and homework problems
- **Lines, Tangents, Taylors Theorem, Roots of an equation**
- **Gradient Descent, Newton-Raphson**

Slope and Equation of a Line

- Slope = rise/run
- The slope of a line is defined as the rise over the run, $m = \Delta y/\Delta x$.
- Given two points (x_1,y_1) and (x_2,y_2) on a line, the slope m of the line is

$$slope = m = \frac{rise}{run} = \frac{y_2 - y_1}{x_2 - x_1}$$



Equation of Line from slope and intercept

Slope (m), intercept (b)

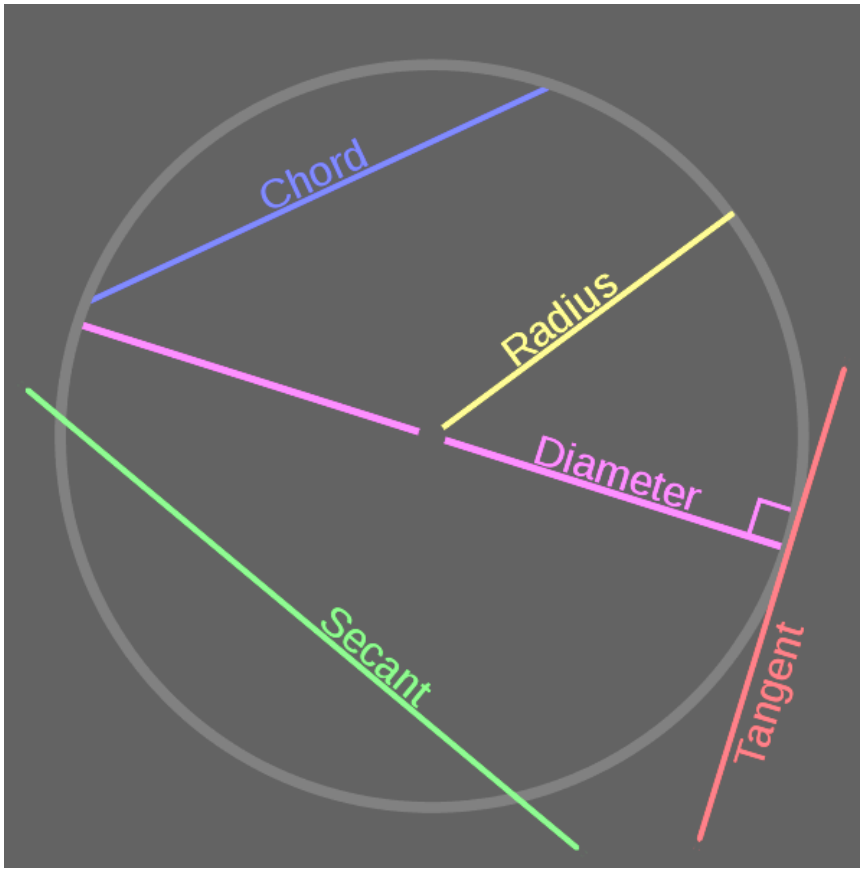
- y = mx +b
- Find the equation of the straight line that has slope *m* = 4 and passes through the point (–1, –6).
- A: In this case, *m* = 4,*x* = –1 and *y* = –6.
- In the slope-intercept form of a straight line, I have *y*, *m*, *x*, and *b*. So the only thing I don't have so far is a value for is *b* (which gives me the *y*-intercept).
- Plug m, y, x and solve for b:
  - $y = mx + b$
    $(-6) = (4)(-1) + b$
    $-6 = -4 + b$
    $-2 = b$
- Then the line equation must be "*y* = 4*x* – 2".

Equation of Line from a point and slope

Point (x1, y1), Slope (m)

- **The other format for straight-line equations is called the "point-slope" form.**
- **For this one, they give you a point (x_1, y_1) and a slope *m*, and have you plug it into this formula:**
  - $y - y_1 = m(x - x_1)$

A secant line of a curve is a line that (locally) intersects two points on the curve.



Tangent Line: Best Approx of curve

- In geometry, the **tangent line** (or simply the **tangent**) to a curve at a given point is the straight line that "just touches" the curve at that point.
- Best straight-line approximation to the curve at that point
  - As it passes through the point of tangency, the tangent line is "going in the same direction" as the curve, and in this sense it is the best straight-line approximation to the curve at that point. The same definition applies to space curves and curves in *n*-dimensional Euclidean space.
- The word "tangent" comes from the Latin *tangere*, meaning "to touch".

Limit of secant's slope is that of the tangent

- It can be used to approximate the tangent to a curve, at some point *f*.
- If the secant to a curve is defined by two points, *P* and *Q*, with *P* fixed and *Q* variable, as *Q* approaches *P* along the curve, the direction of the secant approaches that of the tangent at *P*, assuming there is just one.
- As a consequence, one could say that the limit of the secant's slope, or direction, is that of the tangent.
- In calculus, this idea is the basis of the geometric definition of the derivative. A chord is the portion of a secant that lies within the curve.

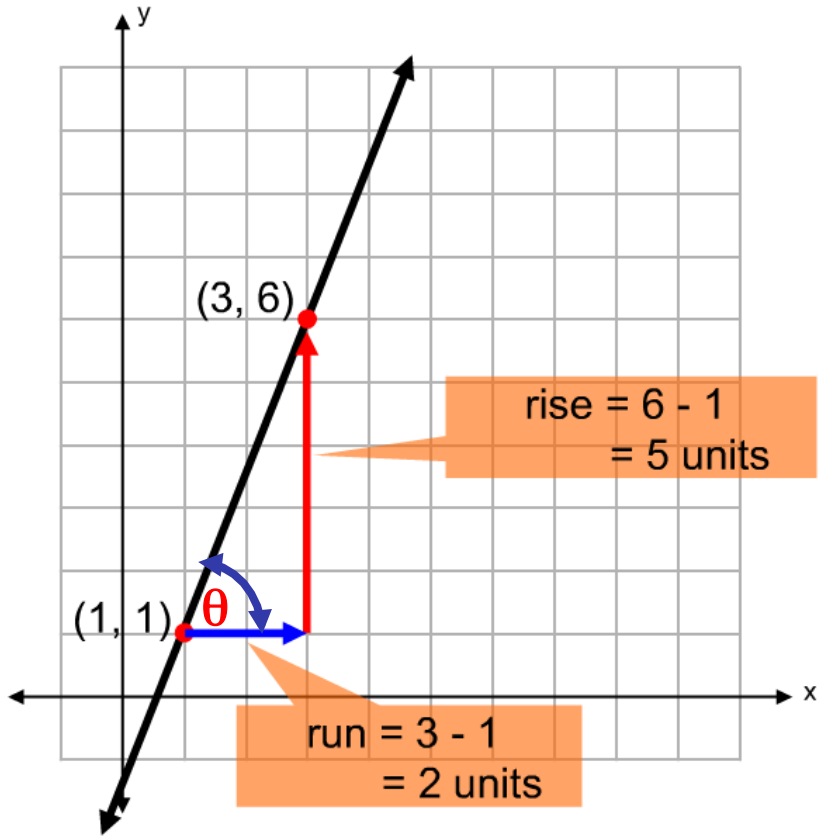
Slope of a Line

The **slope** *m* of a non-vertical line is the number of units the line rises or falls for each unit of horizontal change from left to right.

$$\text{slope} = m = \frac{rise}{run} = \frac{\Delta y}{\Delta x} = \frac{5}{2}$$

$$m = \tan\theta.$$

$$\text{angle} = \arctan\frac{\text{slope}}{100},$$

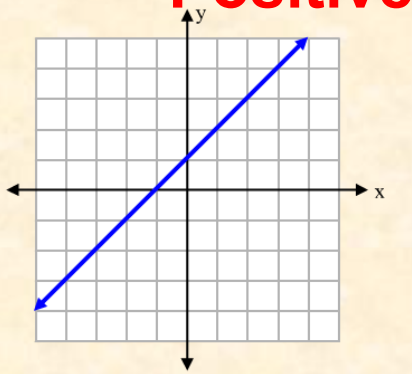


NOTE: The gradient is a generalization of the concept of slope for functions of more than one variable.

Slope

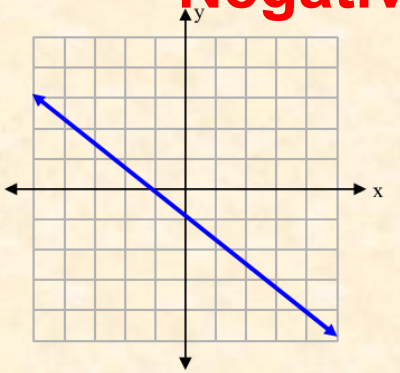
If the line rises to the right, then the slope is positive.

Positive



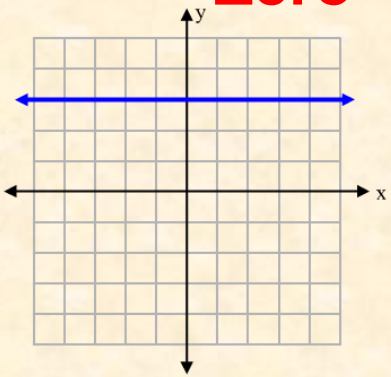
If the line falls to the right, then the slope is negative.

Negative



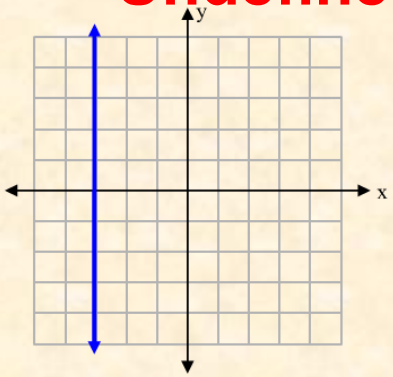
If the line is horizontal, then the slope is zero.

Zero



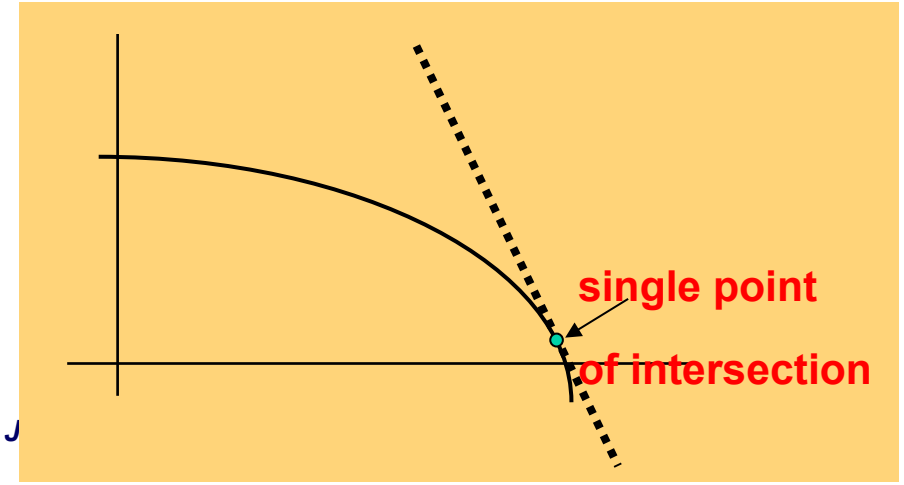
If the line is vertical, then the slope is undefined.

Undefined



Slope versus Derivative?

- **The derivative is the slope of the tangent line at a point on a curve**
- **Derivative is**
  - a function of many (independent) variables
  - the rate of change of a function
  - Corresponds to the slope of the line tangent to the curve (function of one variable)

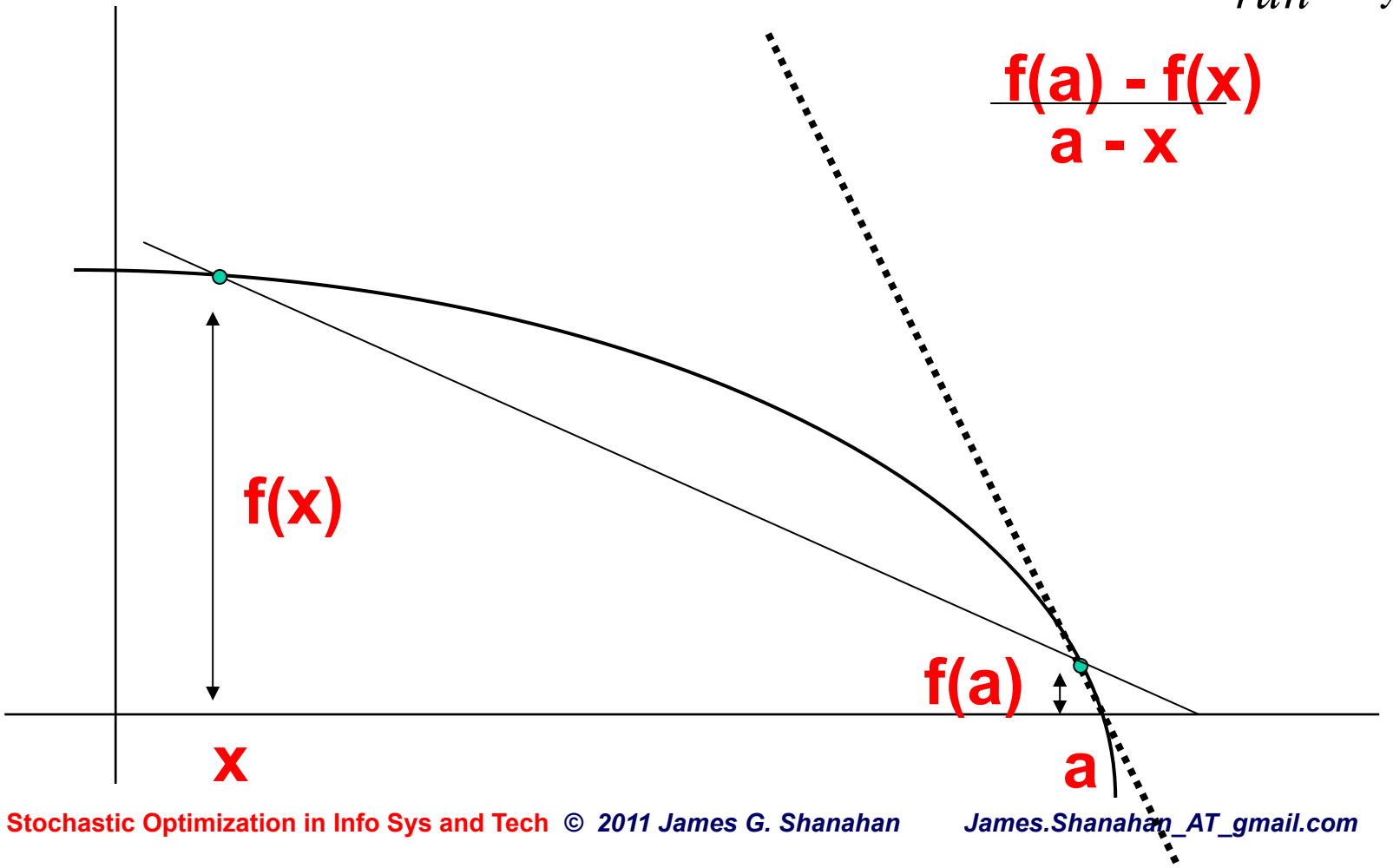


Slope of a secant line

Given two points (x, f(x)), and (a, f(a))

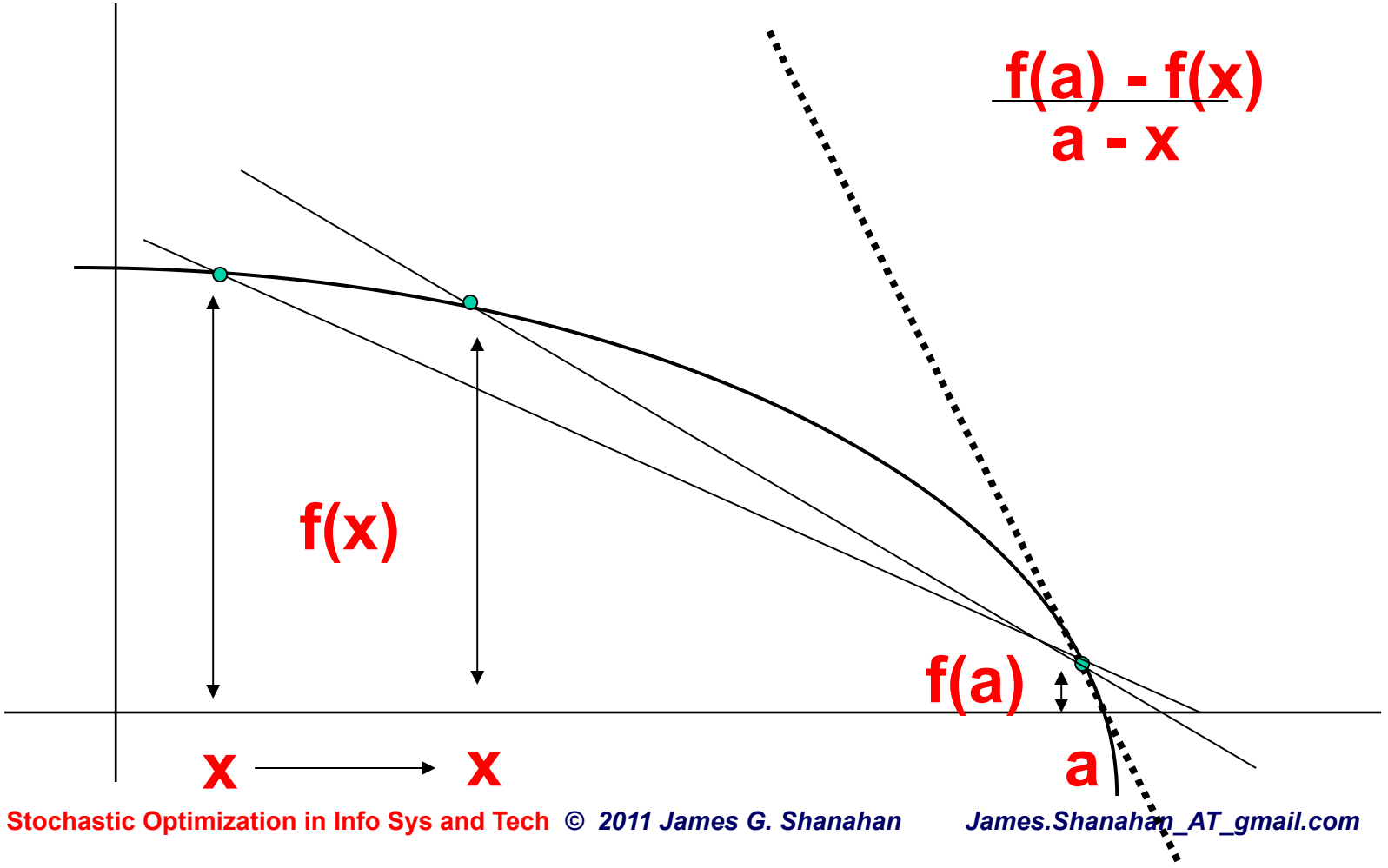
$$slope = m = \frac{rise}{run} = \frac{y_2 - y_1}{x_2 - x_1}$$

$$\frac{f(a) - f(x)}{a - x}$$

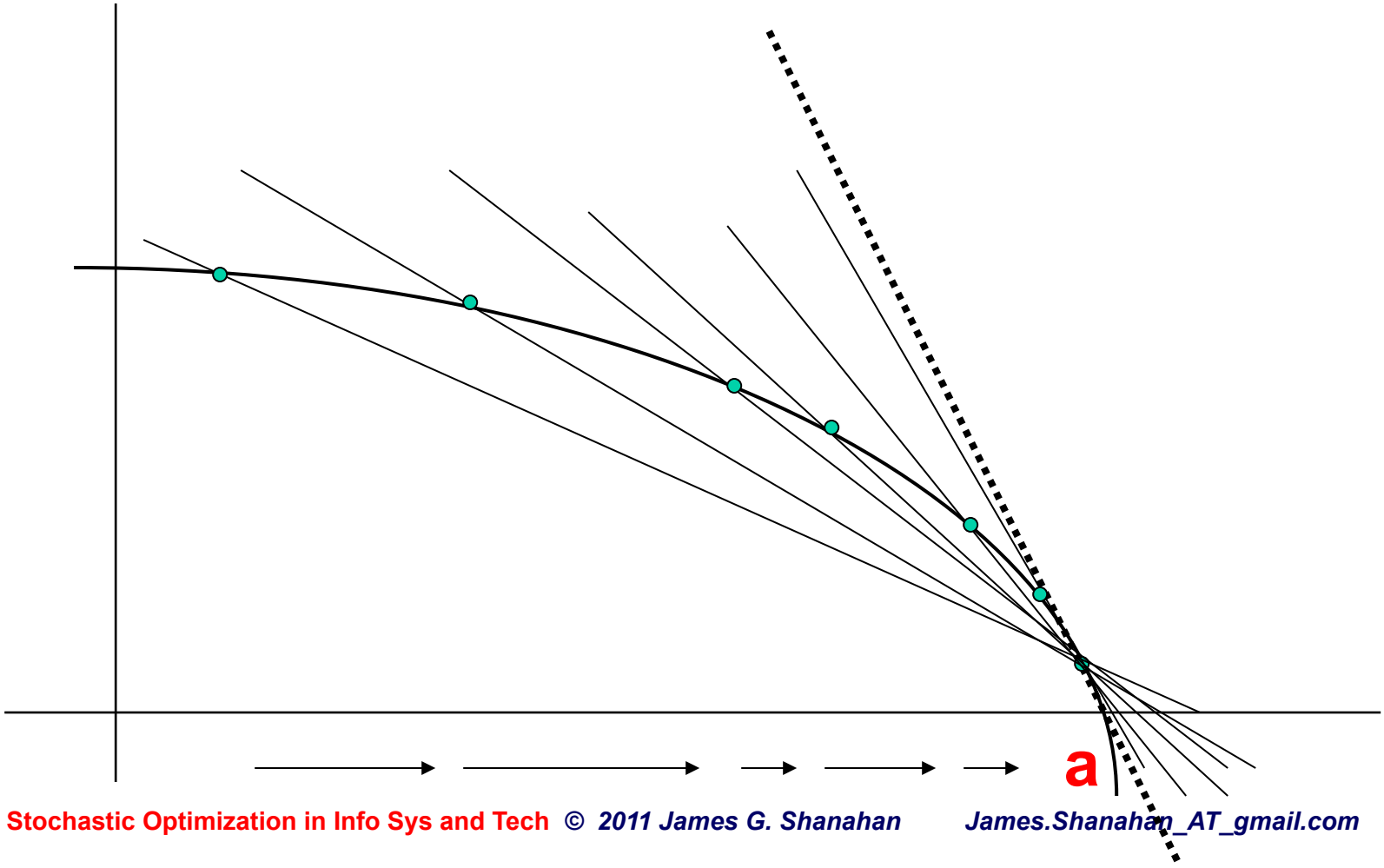


Slope of a (closer) secant line

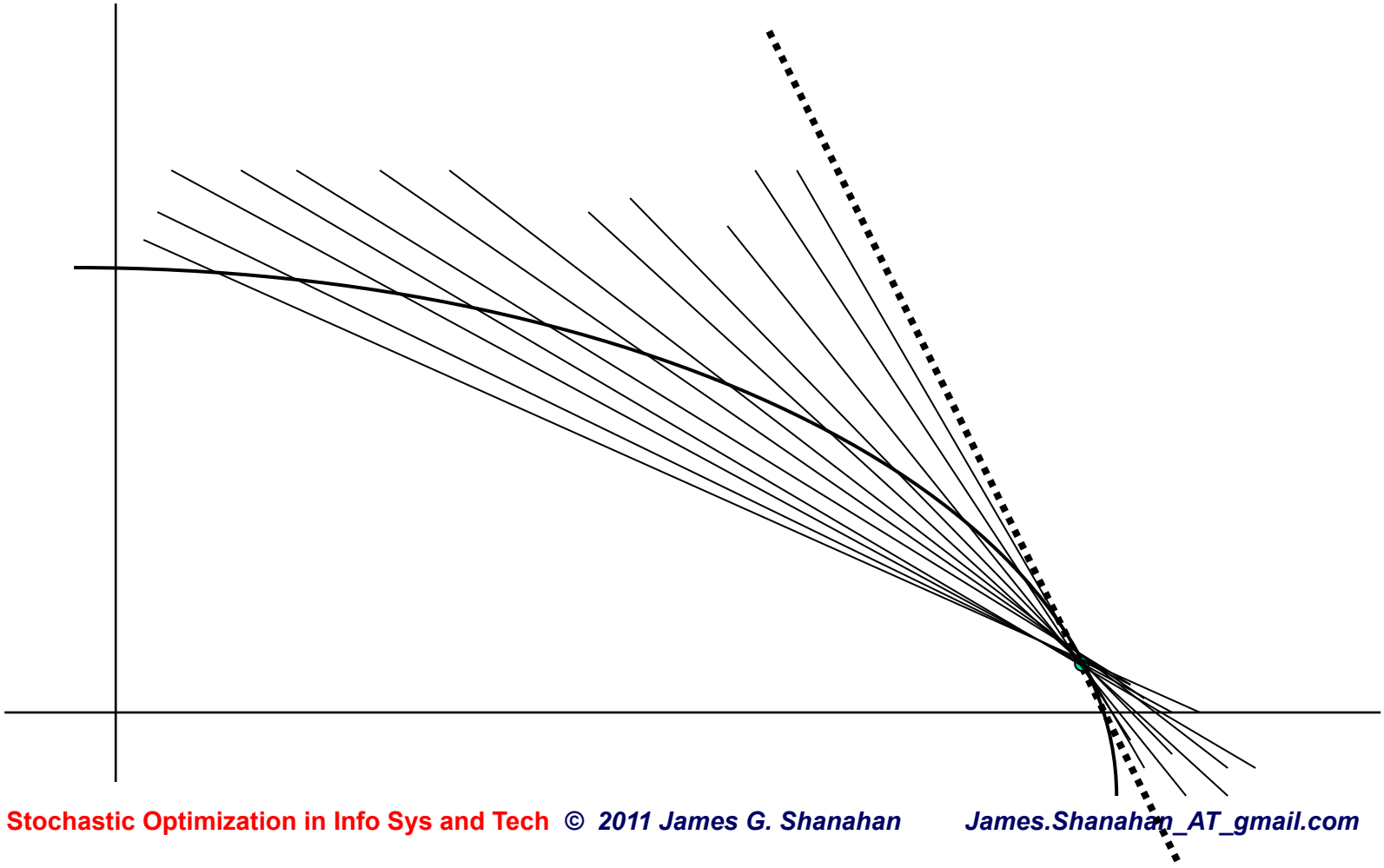
$$slope = m = \frac{rise}{run} = \frac{y_2 - y_1}{x_2 - x_1}$$



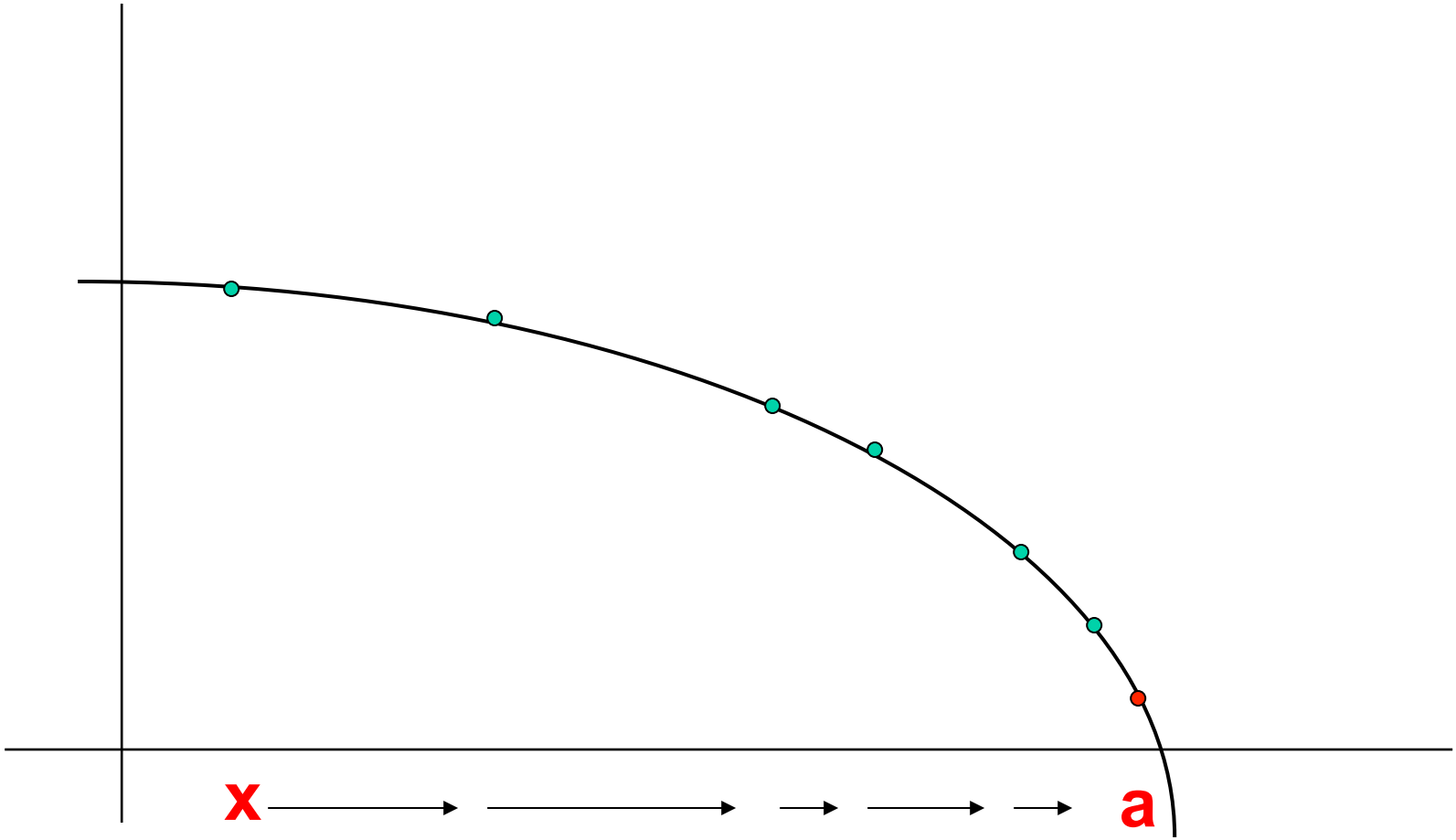
closer and closer…



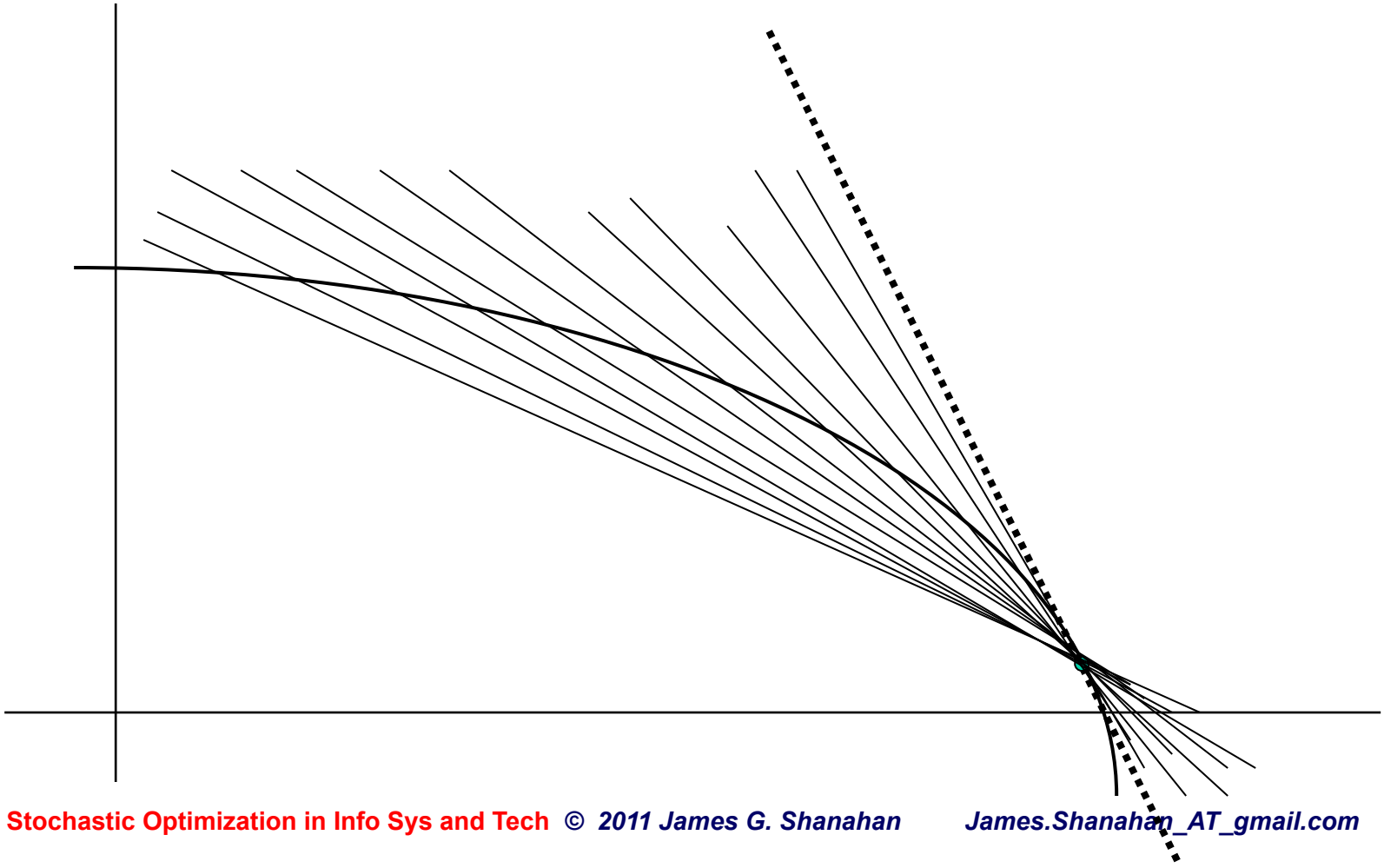
watch the slope...



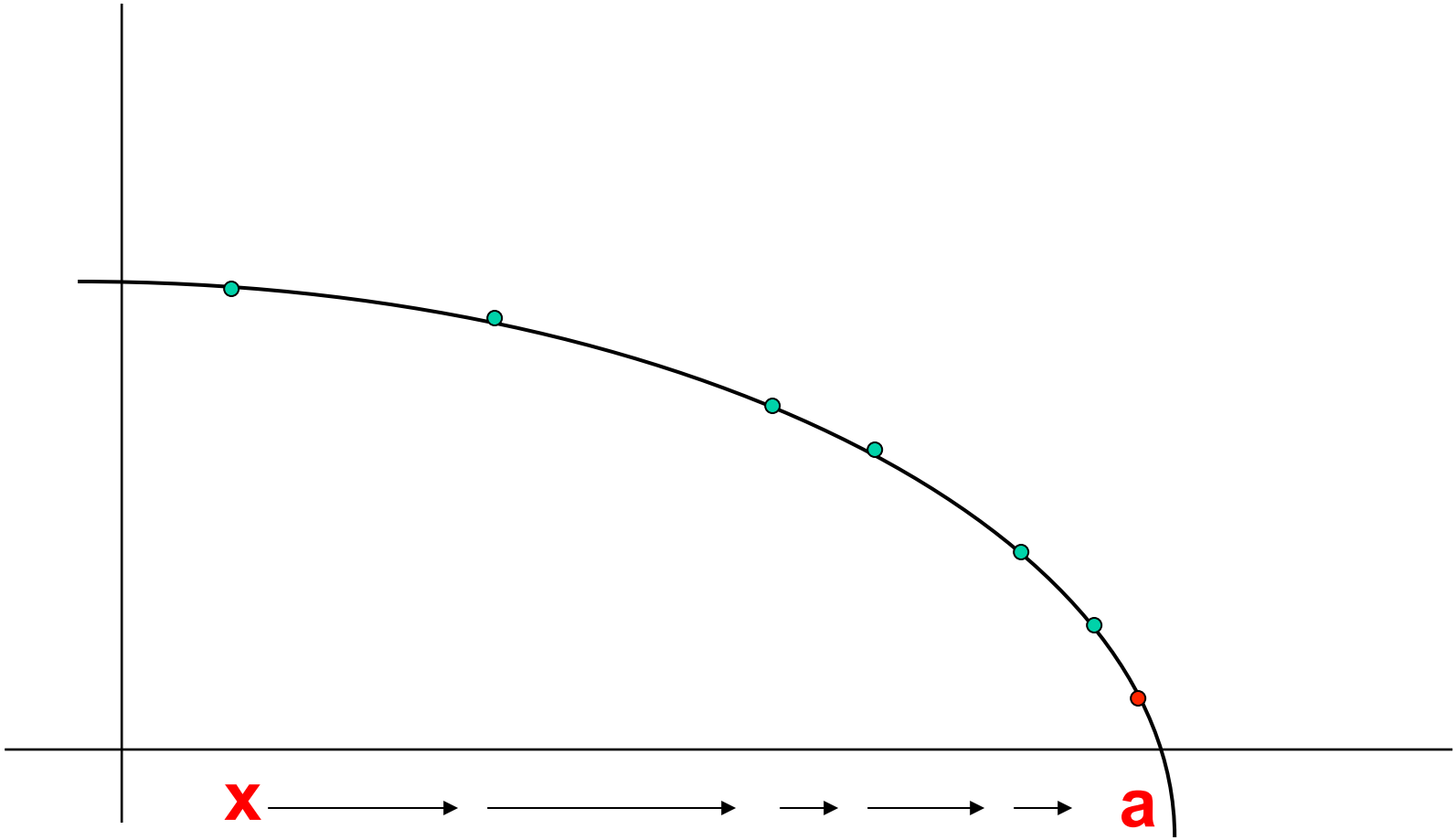
watch what *x* does...



The slope of the secant line gets closer and closer to the slope of the tangent line...



As the values of *x* get closer and closer to *a*!



The slope of the secant lines
gets closer
to the slope of the tangent line...

...as the values of *x*
get closer to *a*

Translates to….

$$\lim_{x \to a} \frac{f(x) - f(a)}{x - a}$$

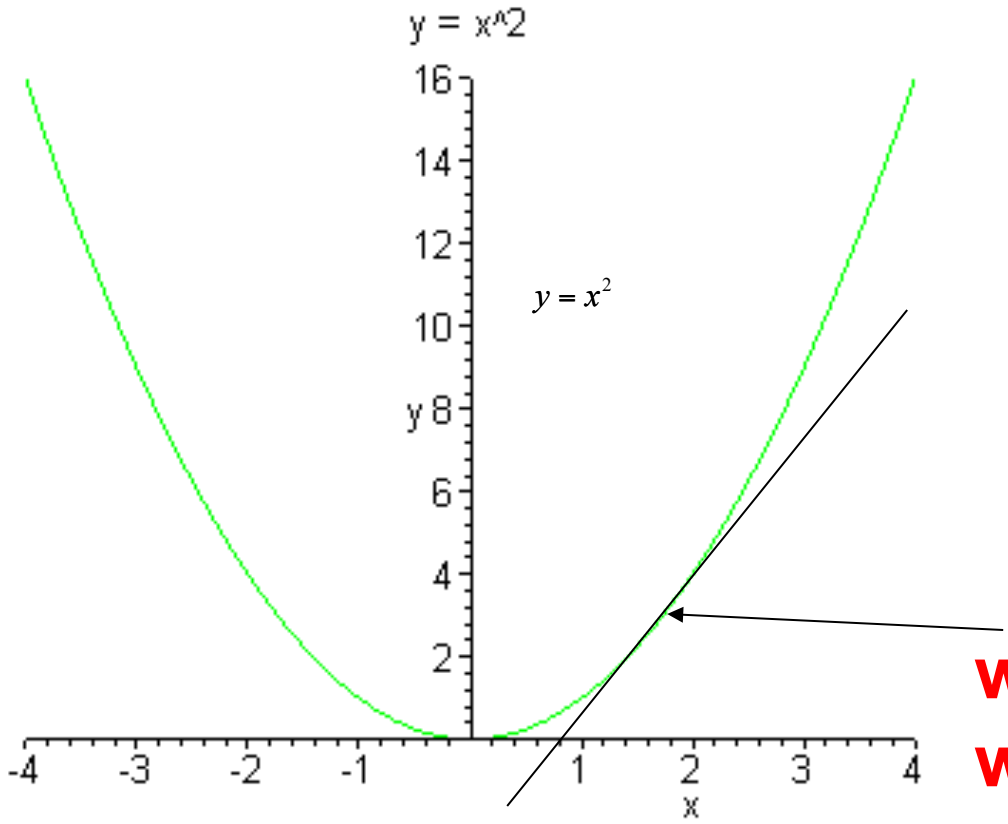
as *x* goes to *a*

Equation for the slope

Which gives us the the exact slope of the line tangent to the curve at *a*!

A VERY simple example...

$$y = x^2$$



want the slope where a=2

In the limit as x tends towards a

$$slope = m = \frac{rise}{run} = \frac{y_2 - y_1}{x_2 - x_1}$$

$$\lim \frac{f(x) - f(a)}{x - a} = \lim \frac{x^2 - a^2}{x - a} = \lim \frac{(x - a)(x + a)}{x - a}$$

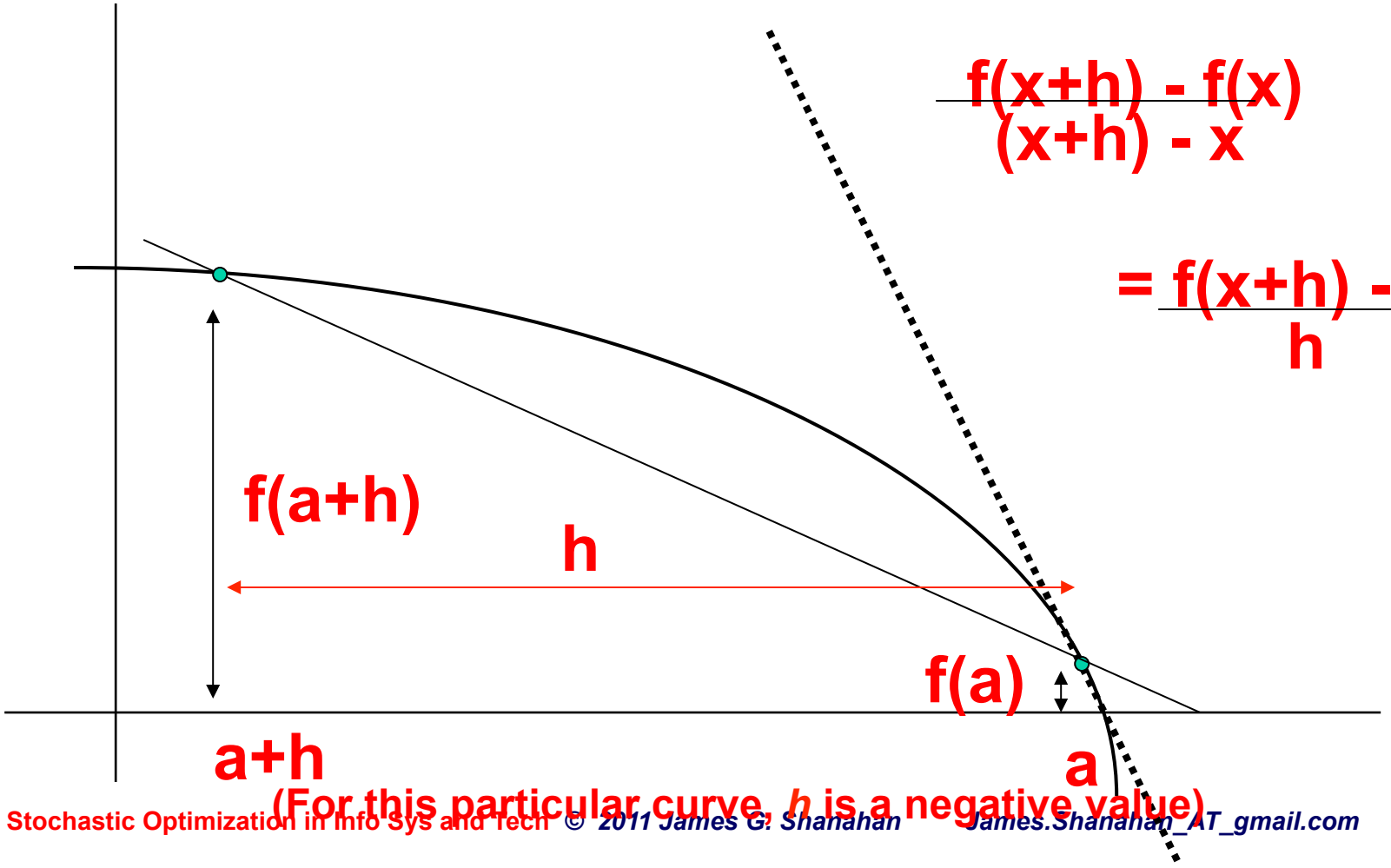
Now as x → a=2 we get

$$\lim(x + a) = \lim(x + 2) = 4$$

Alternatively...

$$\frac{f(x+h) - f(x)}{(x+h) - x}$$

$$= \frac{f(x+h) - f(x)}{h}$$



(For this particular curve, *h* is a negative value)

Thus as h tends towards zero...

$$\lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$

Or X→ a then

$$\lim_{x \to a} \frac{f(x) - f(a)}{x - a}$$

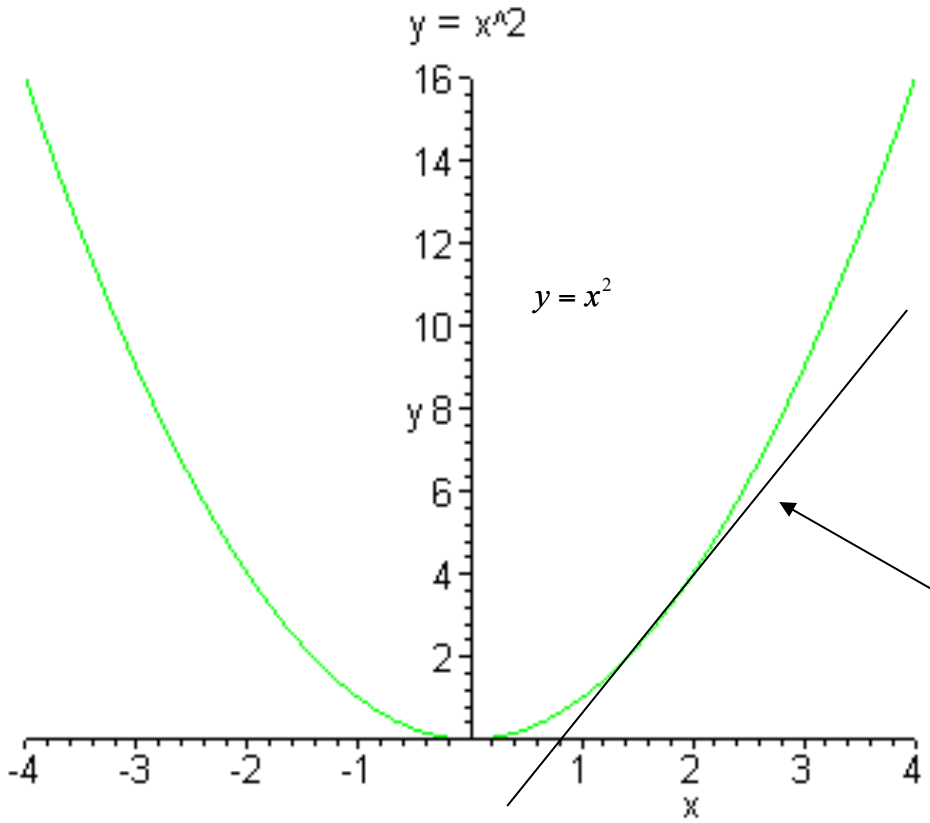
Give us a way to calculate the slope of the line tangent at *a*!

Which one should I use?

(doesn't *really* matter)

A VERY simple example...

$$y = x^2$$



want the slope where a=2

Give two points on the secant …

$$\lim \frac{f(x+h)-f(x)}{h} = \lim \frac{(x+h)^2 - x^2}{h}$$

$$= \lim \frac{x^2 + 2xh + h^2 - x^2}{h} = \lim \frac{h(2x+h)}{h}$$

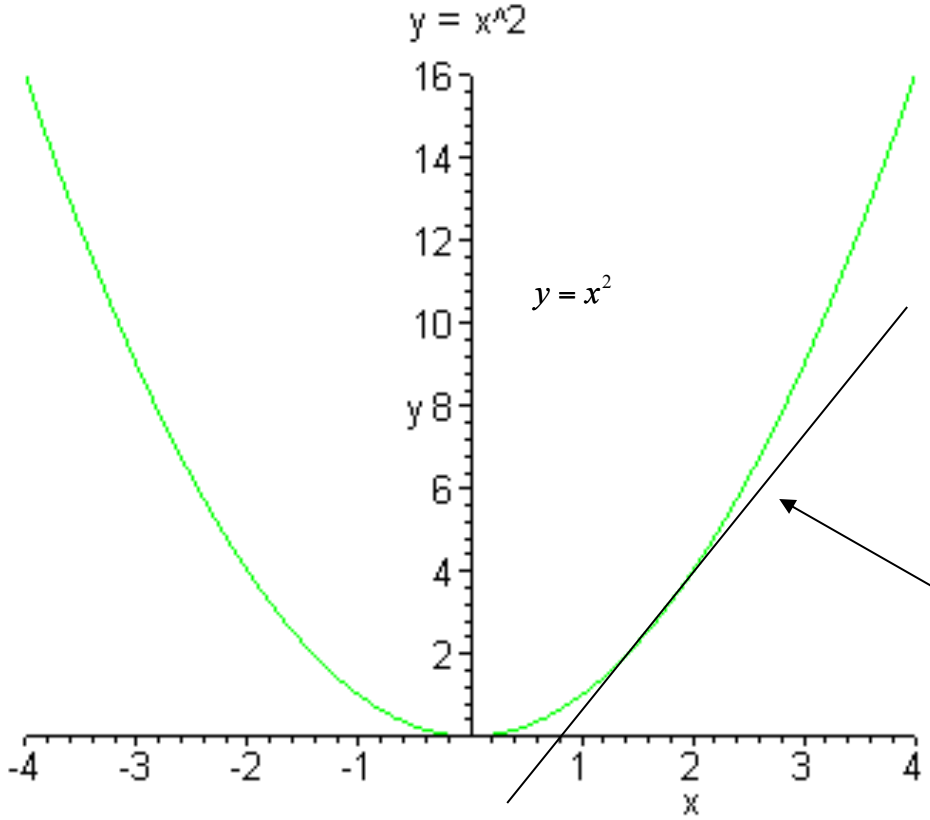
For X=2

$$\lim(2x+h) = 4$$

As h → 0

back to our example...

$$y = x^2 \qquad \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} = \lim_{h \to 0} \frac{(x+h)^2 - x^2}{h}$$



When a=2, the slope is 4

in conclusion...

- The derivative is the slope of the line tangent to the curve (evaluated at a point); having contact at a single point or along a line without crossing
- It is a limit (2 ways to define it)
- The rules of derivatives WILL help one forget these limit definitions..see next
- cool site to go to for additional explanations: http://archives.math.utk.edu/visual.calculus/2/

Slope via Differential Calculus

- **Through differential calculus, one can calculate the slope of the tangent line to a curve f(x) at a point x0.**
  - Slope =f'(x0)
- **At each point x0, the derivative is the slope of a line that is tangent to the curve.**
- **Differentiation is a method to compute the rate at which a dependent output *y* changes with respect to the change in the independent input *x*. This rate of change is called the derivative of *y* with respect to *x*.**
  - In more precise language, the dependence of *y* upon *x* means that *y* is a function of *x*. This functional relationship is often denoted $y = f(x)$, where f denotes the function. If *x* and *y* are real numbers, and if the graph of *y* is plotted against *x*, the derivative measures the slope of this graph at each point.

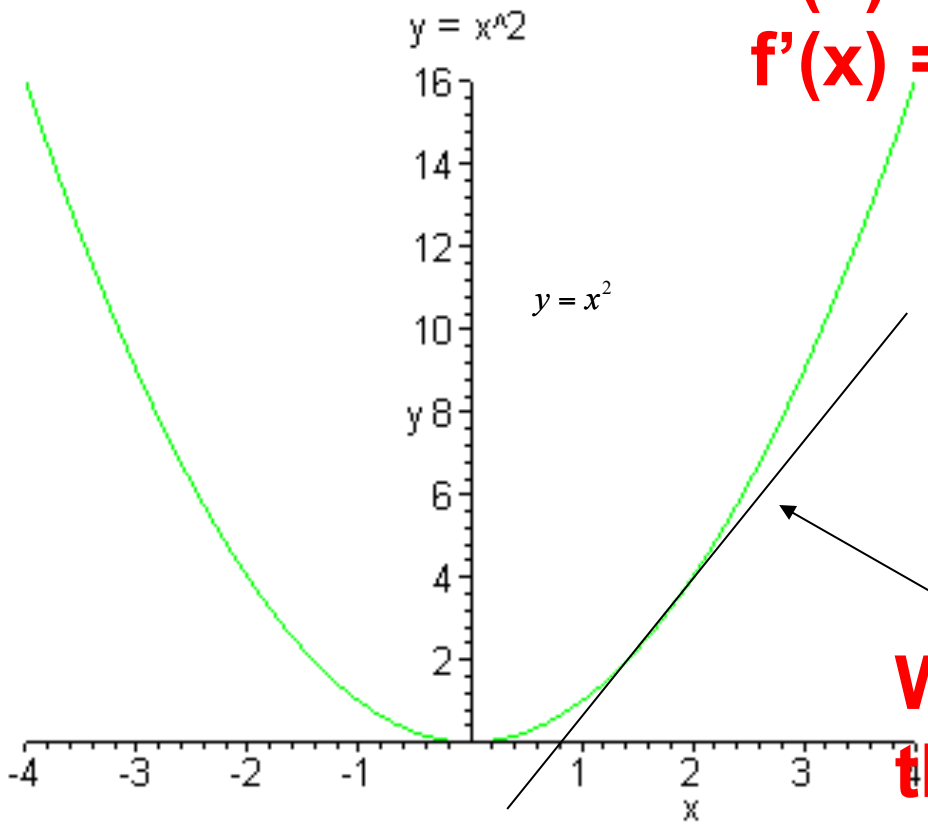
Calculus
gives you a formula
for the gradient of the tangent

Equation of a line given a pt and slope

- **Equation of a tangent line:**
- **y - y0 = f'(x0)(x - x0) ## y-y0 = m(x-x0)**
- **Give a point *(a, f(a))* and Tangent line to the curve at *(a, f(a))*, we can approximate *f(x)* in the vicinity of *a*.**
  - Approximate f(x) linearly by the tangent
- **(i.e., take n=1 in the Taylor series)**

Using derivatives….

$$y = x^2$$



f'(x) = 2x when a =2
f'(x) = 2*2=4

When a=2, f'(2)=4, the slope is 4

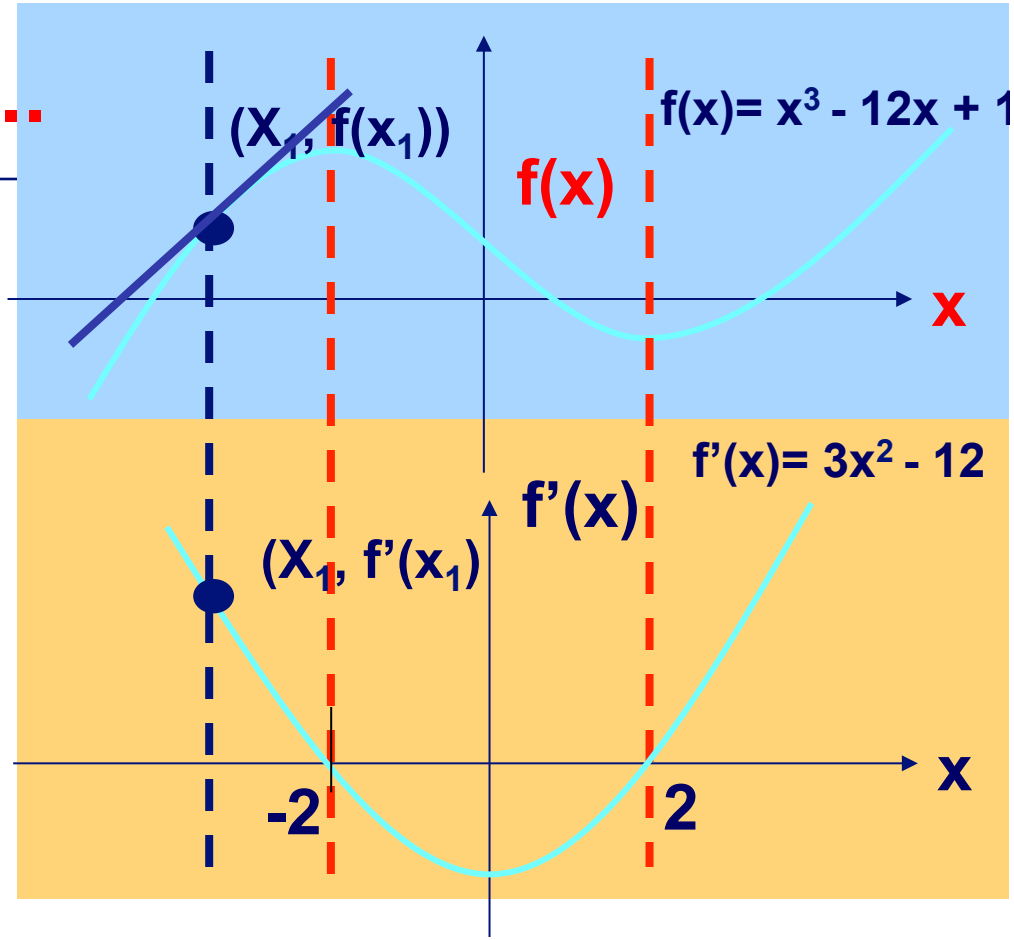
Given a Pt. and Slope...

$$f(x)= x^3 - 12x + 1$$

First derivative

$$f'(x) = 3x^2 - 12$$

[=0 at maximum and minimum]

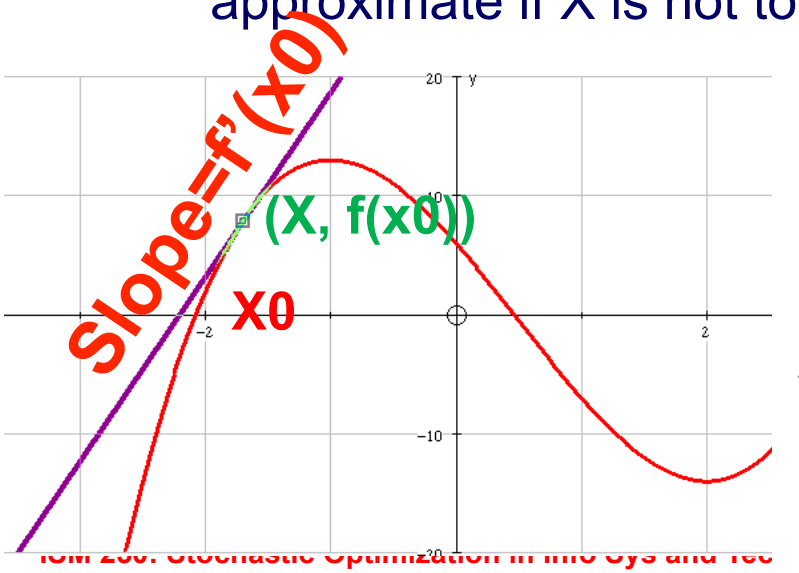


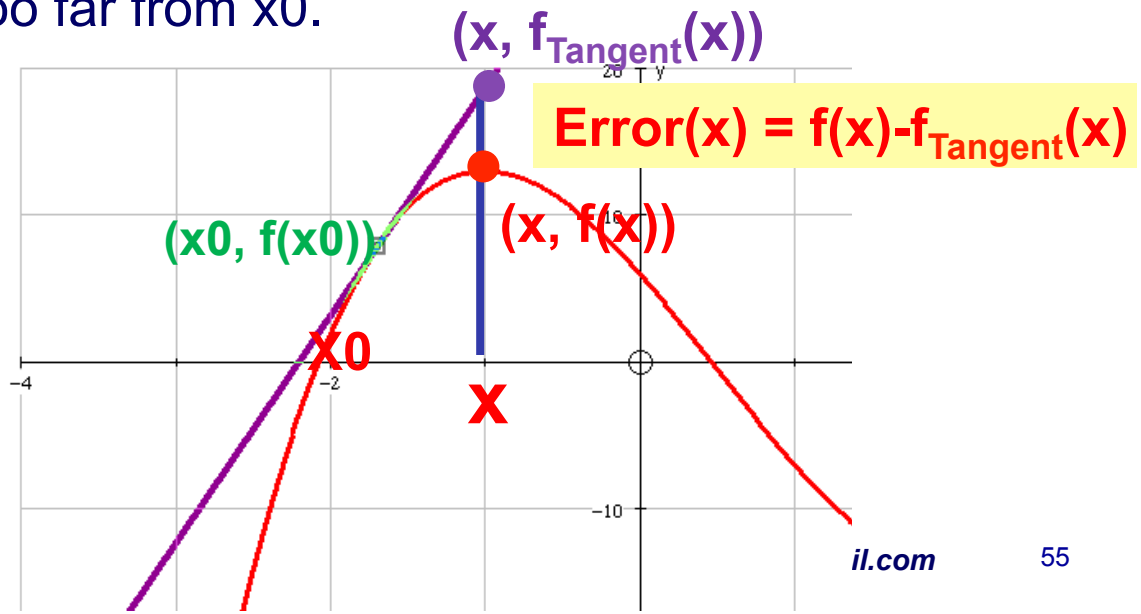
Given a Pt. and Slope... Approximate f(x) with tanger

Using $(x_1, f(x_1))$ and $m=f'(x_1)$
And the equation formula
$y-y0=m(x-x0)$
Plot the tangent line

Approximate a curve using a Tangent

- **Given a point on the curve, (x0, f(x0)) and a slope, f'(x0), we can calculate the equation of the tangent at (x0, f(x0)) as follows:**
  - y - y0 = f'(x0)(x - x0) ## y-y0 = m(x-x0)
  - f(X) – f(x0) = f'(x0)(X-x0) where X is a free variable, f'(x) is the slope
  - Then for any X in the neighbourhood of X0 we can approximate it by the tangent at (x, f(x0))
  - Of course it will not be that accurate but can be reasonably approximate if X is not too far from x0.





R Basics

example.GettingStarted.Chapter1.Fox()

- **R via a GUI R Commander**
  - Examine data; plot data
- **Scripting in R**
  - Variables, vectors, data.frames, functions, graphics

- **Check out example.GettingStarted.Chapter1.Fox()**

RCommander

R Commander

File Edit Data Statistics Graphs Models Distributions Tools Help

Data set: Duncan | Edit data set | View data set | Model: <No active model>

Script Window

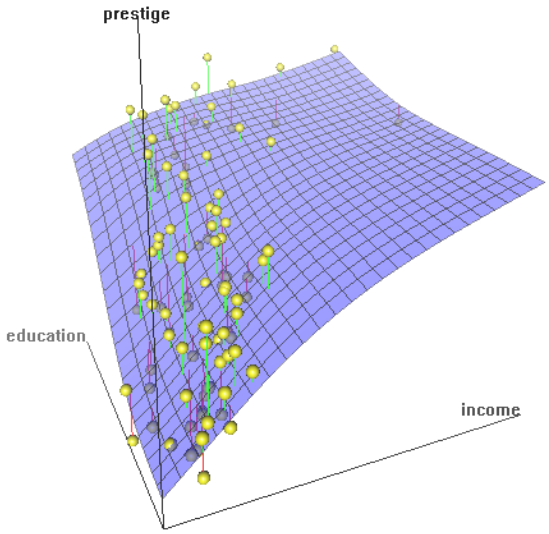
```
data()
Hist(Duncan$education, scale="frequency", breaks="Sturges", col="darkgray")
.Table <- table(Duncan$type)
.Table  # counts for type
100*.Table/sum(.Table)  # percentages for type
remove(.Table)
boxplot(Duncan$education, ylab="education")
{boxplot(income~type, ylab="income", xlab="type", data=Duncan)
} {boxplot(prestige~type, ylab="prestige", xlab="type", data=Duncan)
}
scatter3d(Duncan$education, Duncan$income, Duncan$prestige, fit="linear",
  residuals=TRUE, bg="white", axis.scales=TRUE, grid=TRUE, ellipsoid=FALSE,
  xlab="education", ylab="income", zlab="prestige")
```

Output Window — Submit

```
> plot(lm.temp)

> Hist(Duncan$education, scale="frequency", breaks="Sturges", col="darkgray")

> .Table <- table(Duncan$type)

> .Table  # counts for type

  bc prof   wc
  21   18    6

> 100*.Table/sum(.Table)  # percentages for type

      bc     prof       wc
46.66667 40.00000 13.33333

> remove(.Table)

> boxplot(Duncan$education, ylab="education")

> boxplot(income~type, ylab="income", xlab="type", data=Duncan)

> scatter3d(Duncan$education, Duncan$income, Duncan$prestige, fit="linear",
+   residuals=TRUE, bg="white", axis.scales=TRUE, grid=TRUE, ellipsoid=FALSE,
+   xlab="education", ylab="income", zlab="prestige")
```

Messages

5 Adobe ... | 6 R for W... | 3 Interne... | 3 Windo... | 2 Micros... | EditPlus - [...



R Basics

example.GettingStarted.Chapter1.Fox()

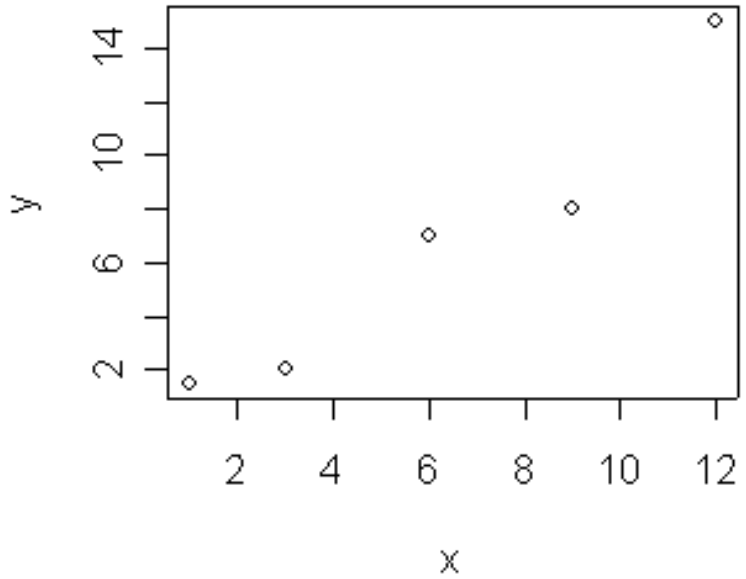
- **R via a GUI R Commander**
  - Examine data; plot data
- **Scripting in R**
  - Variables, vectors, data.frames, functions, graphics

- **Check out example.GettingStarted.Chapter1.Fox()**

Simple Plotting Example

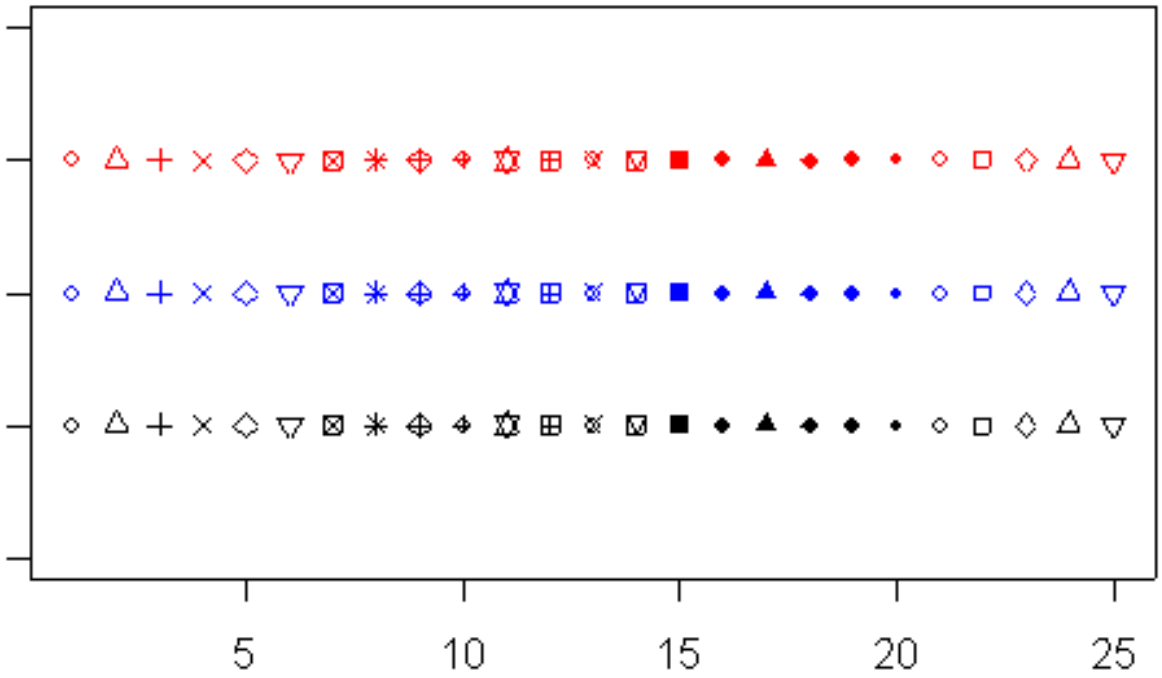
- # Example 1
- # make a very simple plot

```
x <- c(1,3,6,9,12)
y <- c(1.5,2,7,8,15)
plot(x,y)
```



Plotting in R: plot character

- Plot symbols are set within the **plot()** function by setting the **pch** parameter (plot character?) equal to an integer between 1 and 25.

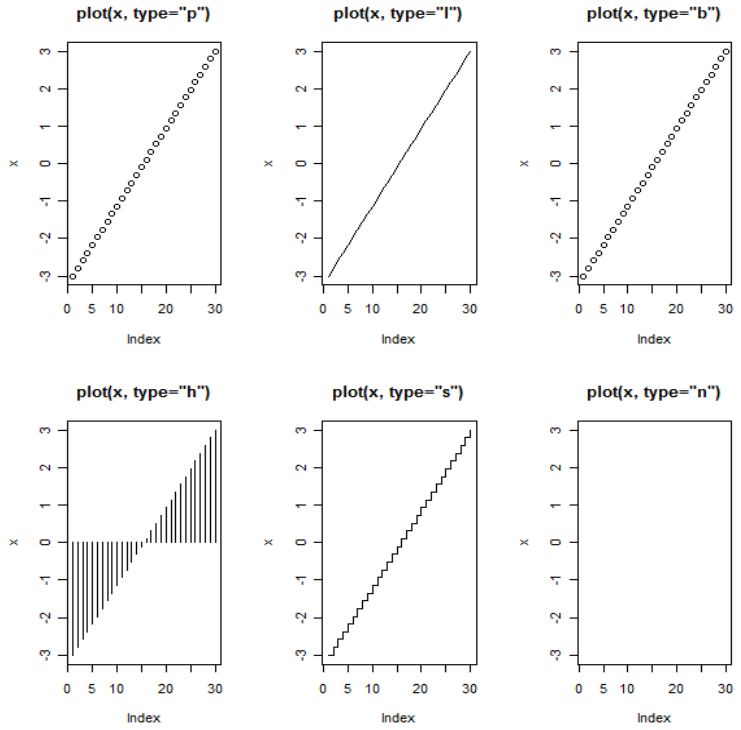


Plot a points and then …a line

```
x <- c(1,3,6,9,12)
y <- c(1.5,2,7,8,15)
# Example 2. Draw a plot, set a bunch of parameters.
plot(x,y, xlab="x axis", ylab="y axis", main="my plot",
    ylim=c(0,20), xlim=c(0,20), pch=15, col="blue")
```

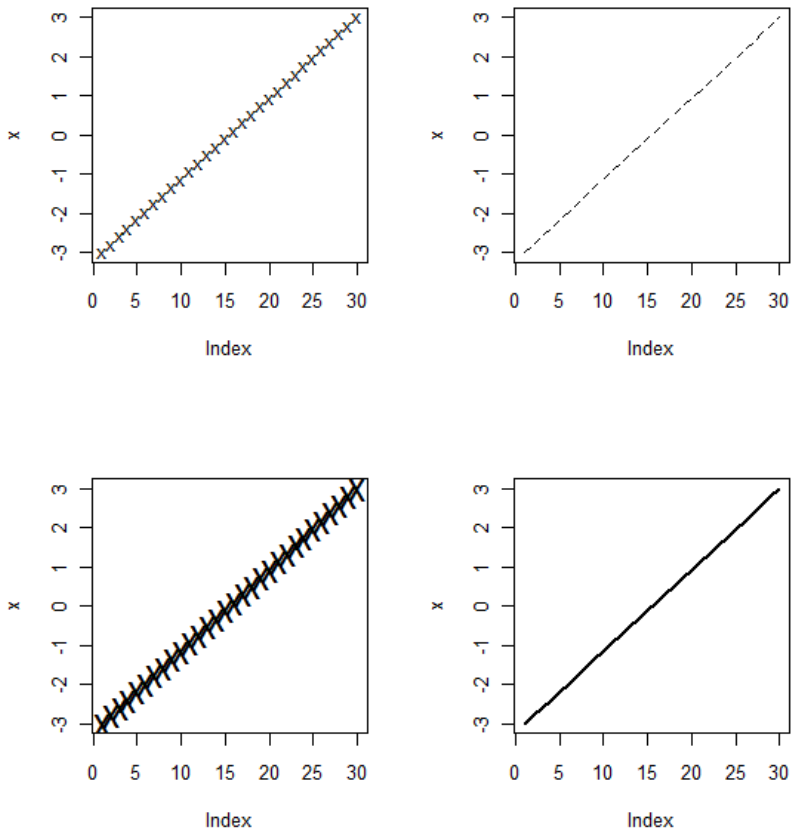
Plotting examples

```
par(mfrow=c(2,3))
plot(x, type="p", main="plot(x,type=\"p\")") # Note the escaped
    quotes \"
plot(x, type="l", main="plot(x, type=\"l\")")
plot(x, type="b“, main="plot(x, type=\"b\")")
plot(x, type="h“, main="plot(x, type=\"h\")")
plot(x, type="s", main="plot(x, type=\"s\")")
plot(x, type="n", main="plot(x, type=\"n\")")
```



Different symbols and line types

```
par(mfrow=c(2,2))
# Different symbols and line types
plot(x, pch="x")
plot(x, type="l", lty=2)
plot(x, pch="x",  cex=2)
plot(x, type="l", lwd=2)
```



Plot a line

```
x <- c(1,3,6,9,12)
y <- c(1.5,2,7,8,15)
# Example 2. Draw a plot, set a bunch of parameters.
plot(x,y, xlab="x axis", ylab="y axis", main="my plot",
    ylim=c(0,20), xlim=c(0,20), pch=15, col="blue")
# fit a line to the points
myline.fit <- lm(y ~ x)
# get information about the fit
summary(myline.fit)


# draw the fit line on the plot
abline(myline.fit)
```

Add points to graph

```
# Example 3
# add some more points to the graph
x2 <- c(0.5, 3, 5, 8, 12)
y2 <- c(0.8, 1, 2, 4, 6)
points(x2, y2, pch=16, col="green")
```

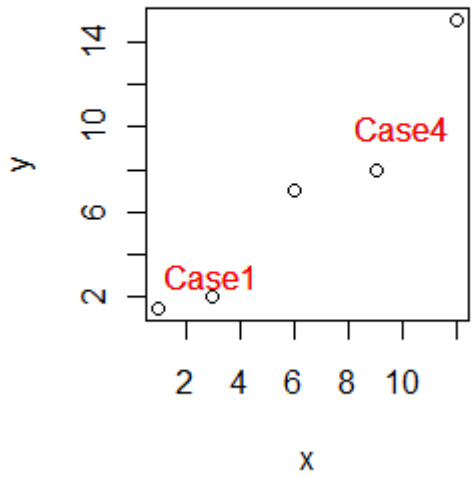
- The text() function allows us to put text on the plot where we want it. An obvious use is to label a line or group of points.

```
text(c(2,2),c(37,35),labels=c("Non-case","Case"))
```

Simple Plotting Example with text

```
# Example 1
# make a very simple plot
x <- c(1,3,6,9,12)
y <- c(1.5,2,7,8,15)
plot(x,y)
text(c(3,10),c(3,10),labels=c("Case1","Case4"),
    col="red")
```

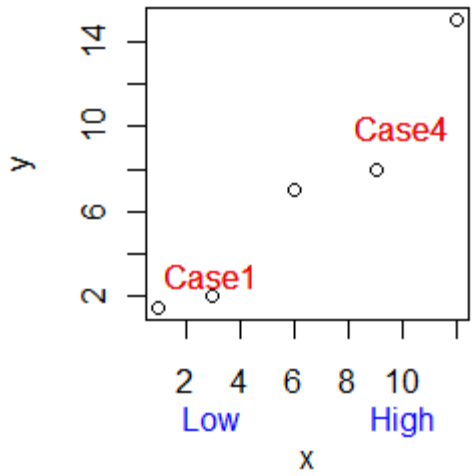
The text() function allows us to put text on the plot where we want it.



Plotting Example with margin text

```
# Example 1
# make a very simple plot
x <- c(1,3,6,9,12)
y <- c(1.5,2,7,8,15)
plot(x,y)
text(c(3,10),c(3,10),labels=c("Case1","Case4"),
    col="red")
mtext(c("Low","High"),side=1,line=2,at=c(3,10),
    col="blue")
```

Text labels can also be placed in the margins of a plot using the mtext() function. This would place the words "Low" and "High" on the second line below the X axis centered at 3 and 10 units.



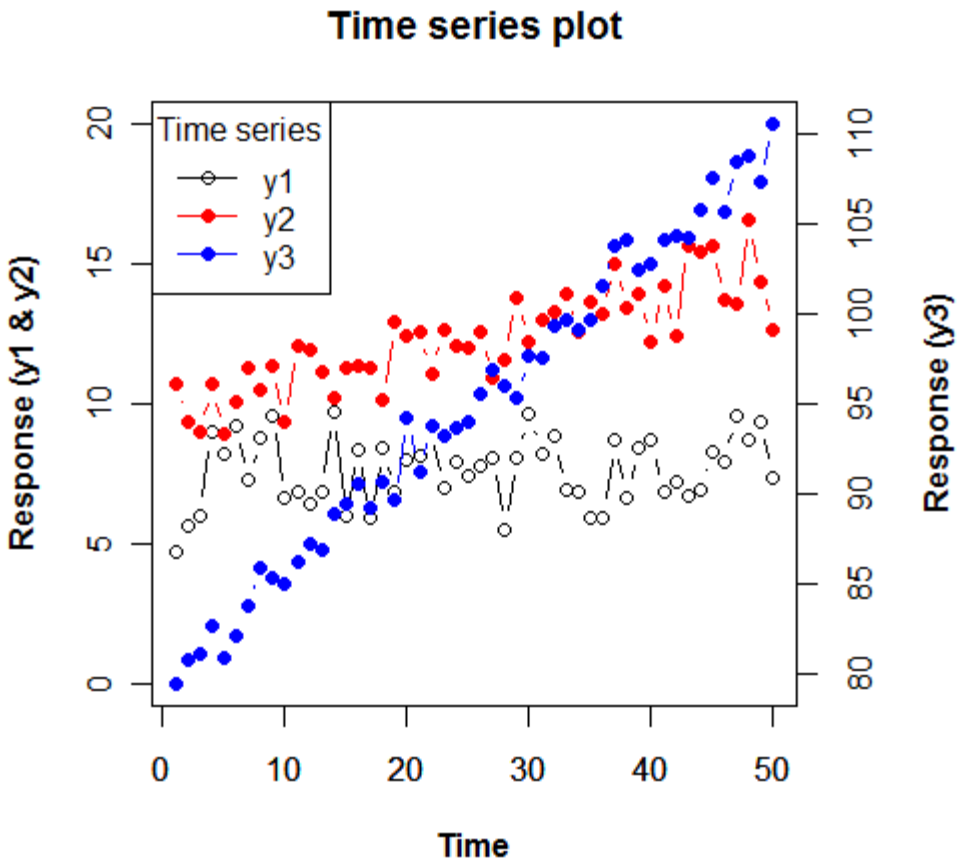
Two y-axis example

http://rgraphics.limnology.wisc.edu/line.php

```
#http://rgraphics.limnology.wisc.edu/line.php
rm(list = ls())      # Clear all variables
graphics.off()    # Close graphics windows

# Generate sample time series data
ti = 1:50                          # Generate 50 sample time steps
# Generate 50 stochastic data points for time series y1
y1 = 8 + rnorm(50)

# Plot the y1 data
par(oma=c(2,2,2,4))     # Set outer margin areas (only necessary in order to plot extra y-axis)
plot(ti, y1,                          # Data to plot - x, y
    type="b",                         # Plot lines and points. Use "p" for points only, "l" for lines only
    main="Time series plot",     # Main title for the plot
    xlab="Time",                      # Label for the x-axis
    ylab="Response (y1 & y2)", # Label for the y-axis
    font.lab=2,          # Font to use for the axis labels: 1=plain text, 2=bold, 3=italic, 4=bold italic
    ylim=c(0,20),       # Range for the y-axis; "xlim" does same for x-axis
    xaxp=c(0,50,5),    # X-axis min, max and number of intervals; "yaxp" does same for y-axis
    bty="l")              # Box around plot to contain only left and lower lines
```

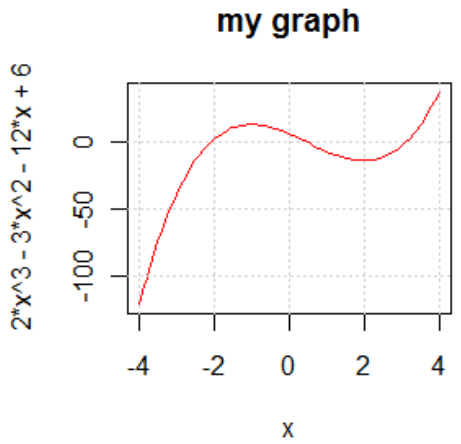
Two y-axis example

http://rgraphics.limnology.wisc.edu/line.php



Functions in R

```
fx=function(x) {
  2*x^3 - 3*x^2 - 12*x + 6
}

x=seq(-4, 4, by=0.1)
plot(x, fx(x), main="my graph", xlab="x",
    ylab="2*x^3 - 3*x^2 - 12*x + 6", pch=", " type="l")
grid()
```



R Graphics Basics: Save plots to PDF

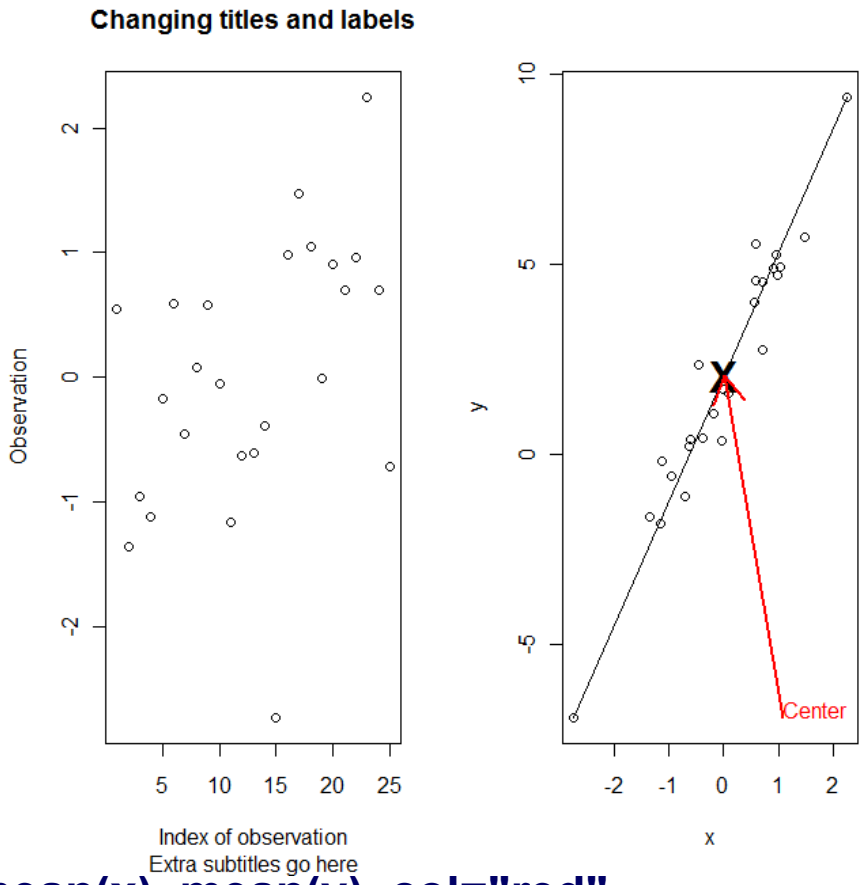
```
## R code and examples for "Modern Applied Statistics Using R"
##      Lecture 3: Graphics
## Alexander.Ploner@ki.se   2007-09-17

# The basic high-level plot
x = rnorm(25)
y = 2 + 3*x + rnorm(25)
plot(x)

# Create a pdf file in home directory
setwd("~")
pdf("test.pdf")
plot(x)
dev.off()
# Nice trick - works if a pdf viewer is installed
viewer = options()$pdfviewer
system(paste(viewer, "test.pdf"))

## Note: we can easily create multipage plots
pdf("test2.pdf")
plot(x, main="Page 1")
plot(y, main="Page 2")
dev.off()
```

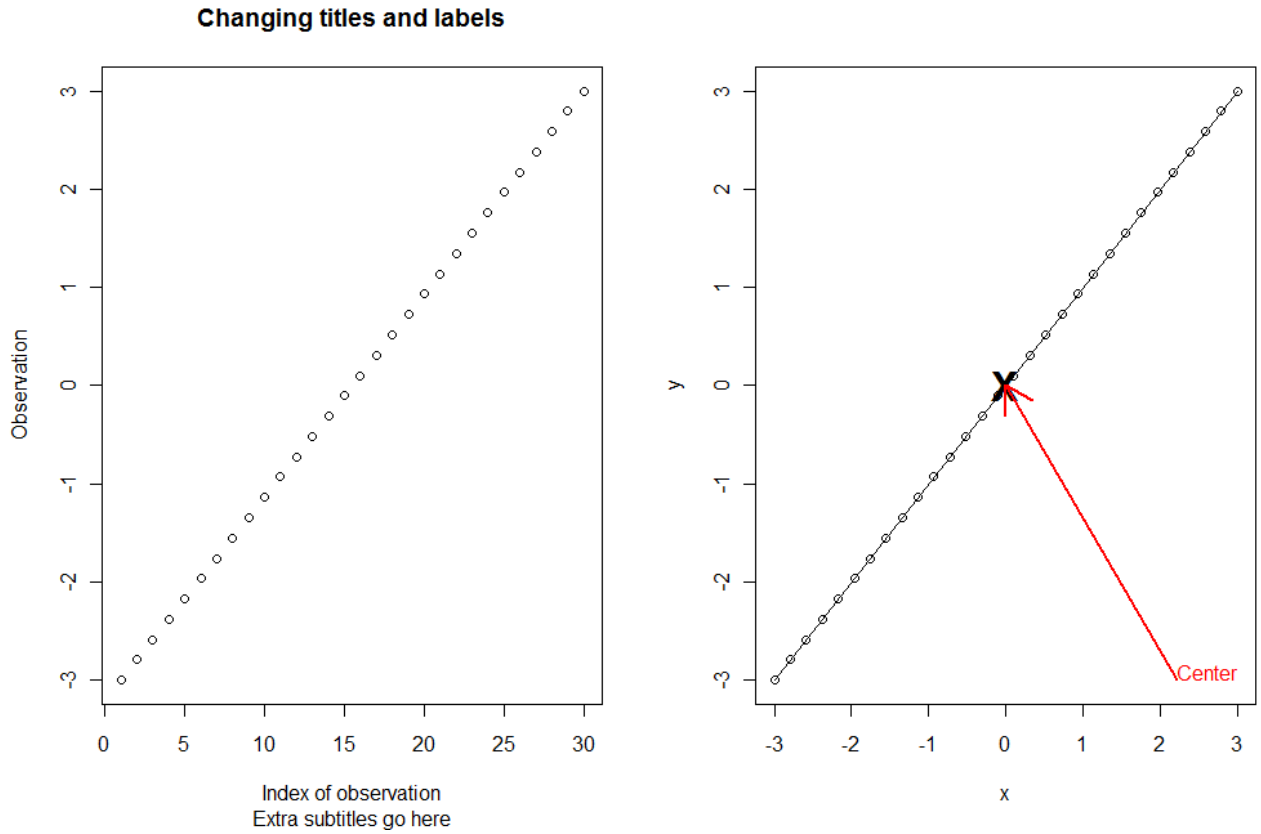
See example.PDF()
Useful for reporting

[http://www.meb.ki.se/~aleplo/R2007/Rcourse03.R]

Putting it all together …(PIAT)

```
 x = rnorm(25)
  y = 2 + 3*x + rnorm(25)
par(mfrow=c(1,2)
# Common text elements
plot(x, main="Changing titles and labels", s
  xlab="Index of observation", ylab="Obser

# Adding extra points and lines; we switch t
plot(x,y)
points(mean(x), mean(y), pch="X", cex=2, fo
lines(range(x), range(y))

# Adding text and arrows
text(max(x), min(y), "Center", col=2, adj=c(1
arrows(max(x)-strwidth("Center"), min(y), mean(x), mean(y), col="red",
     lwd=2)
```




Changing titles and labels
Observation
Index of observation
Extra subtitles go here
y
x
Center

More Graphic Examples

```
####
#incorporate more examples from http://
   www.meb.ki.se/~aleplo/R2007/Rcourse03.R
####
```

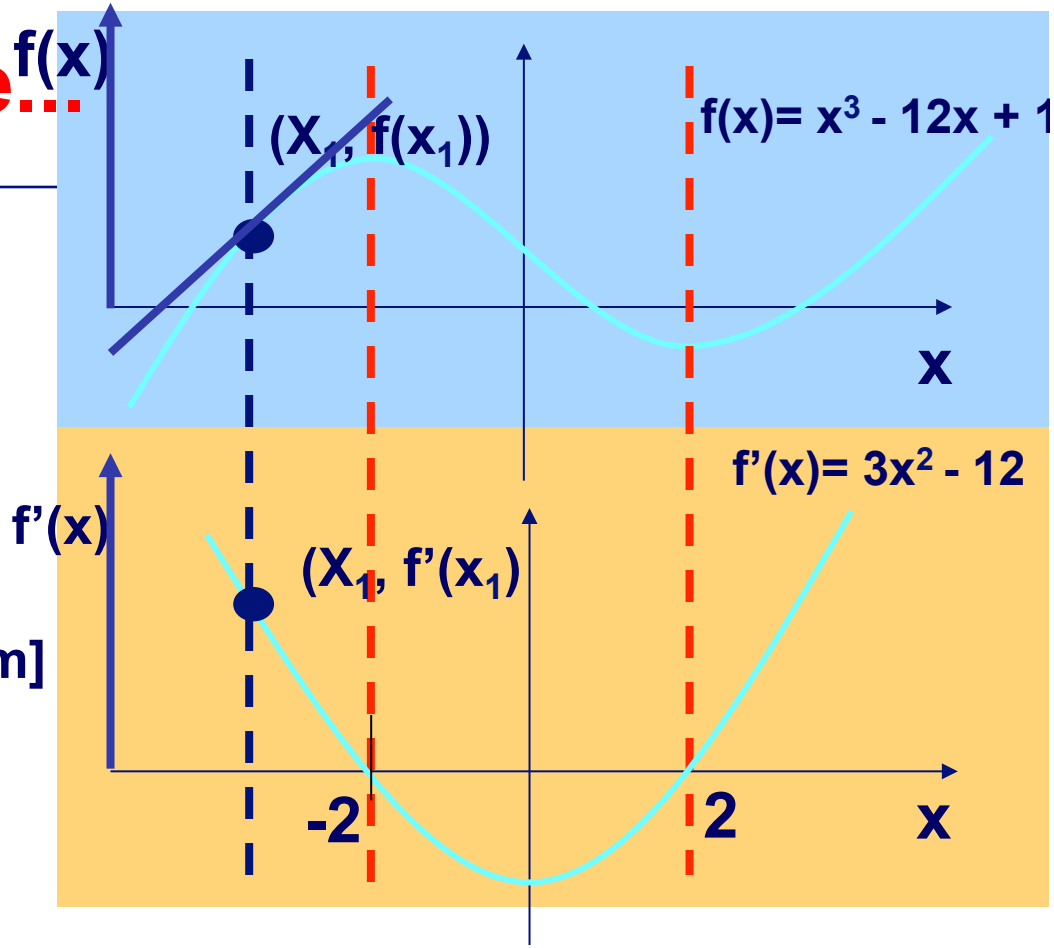
Given a Pt. and Slope...

$$f(x) = x^3 - 12x + 1$$

First derivative

$$f'(x) = 3x^2 - 12$$

[f(x)==0 at maximum and minimum]

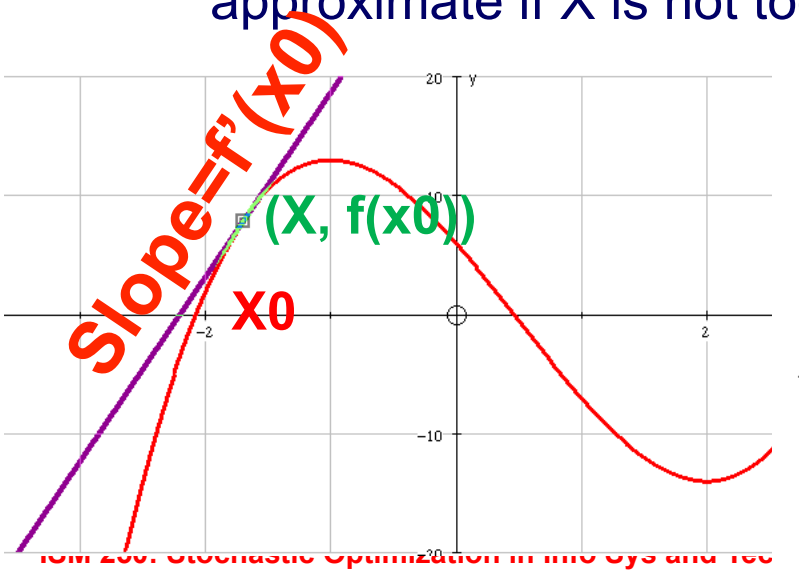


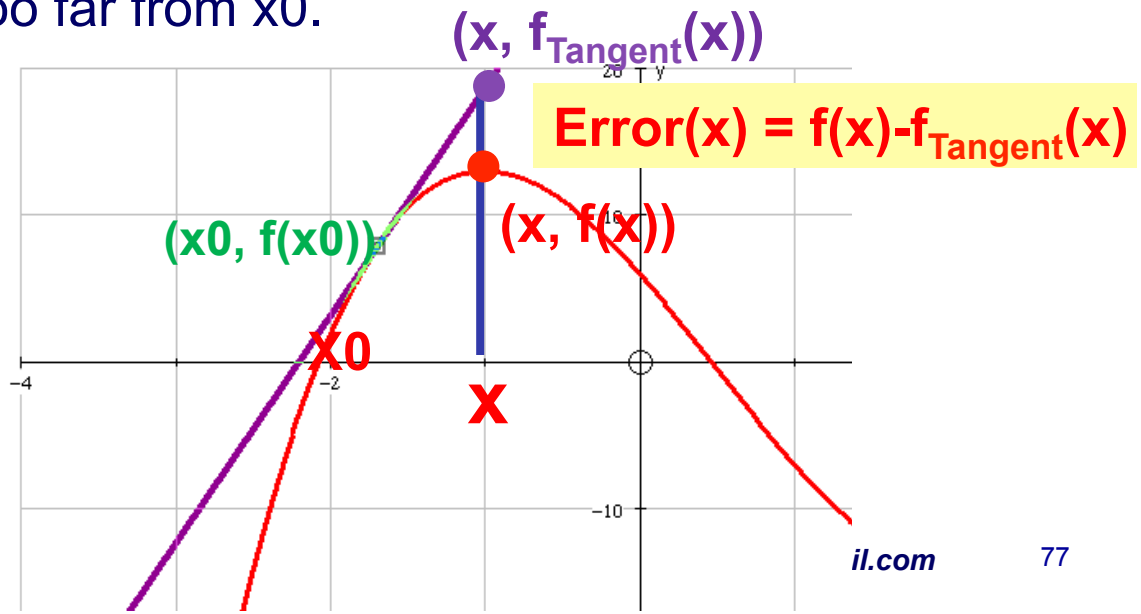
Given a Pt. and Slope... Approximate f(x) with tanger

Using $(x_1, f(x_1))$ and $m=f'(x_1)$
And the equation formula
$y-y0=m(x-x0)$
Plot the tangent line

Approximate a curve using a Tangent

- **Given a point on the curve, (x0, f(x0)) and a slope, f'(x0), we can calculate the equation of the tangent at (x0, f(x0)) as follows:**
  - y - y0 = f'(x0)(x - x0) ## y-y0 = m(x-x0)
  - f(X) – f(x0) = f'(x0)(X-x0) where X is a free variable, f'(x) is the slope
  - Then for any X in the neighbourhood of X0 we can approximate it by the tangent at (x, f(x0))
  - Of course it will not be that accurate but can be reasonably approximate if X is not too far from x0.





Exercise 1.1

- Given function $f(x)= x^3 - 12x + 1$ approximate the curve at (-1, f(-1)) in the x range of [-3, 3] using the tangent to (-1, f(-1)) [also know as the first order Taylor approximation]
- In R, plot the curves f(x), f'(x) and the tangent approximation and label appropriately
- Add text and arrows to highlight (-1,f(-1)) and its tangent line
- Comment on the approximation of f(x) at x = -3

  $f_{Tangent}$ (x= -3)

- HINT: review material on slides before this and after this.

Derivatives in R using deriv(), D()

```
fx=function(x) {
  2*x^3 - 3*x^2 - 12*x + 6
}
fprime = function (x){
   6*x^2 - 6 * x - 12
}
```

See example.drawTangent()

F(x) =X^2; f'(x) = 2x; df(x)/dx)=f'(x)

#deriv() Compute derivatives of simple expressions, symbolically

```
dx2x <- deriv(~ x^2, "x", TRUE)
> dx2x
function (x)
{
    .value <- x^2
    .grad <- array(0, c(length(.value), 1L), list(NULL, c("x")))
    .grad[, "x"] <- 2 * x
    attr(.value, "gradient") <- .grad
    .value
}
```

```
>dx2x(2)
4
```

Derivatives in R using deriv(), D()

F(x) =X^2; f'(x) = 2x; df(x)/dx)=f'(x)

```
> dx2x <- deriv(~ x^2, "x", TRUE) ; dx2x
function (x)
{
    .value <- x^2
    .grad <- array(0, c(length(.value), 1L), list(NULL,
    c("x")))
    .grad[, "x"] <- 2 * x
    attr(.value, "gradient") <- .grad
    .value
}
```

See example.drawTangent()

C:\jimi\Projects\R\GradientDescent\JimisMLCourse.R

Tangent Example: f(x), f'(x)

```
fx=function(x) {
  2*x^3 - 3*x^2 - 12*x + 6
}
fprime = function (x){   6*x^2 - 6 * x – 12 }

fprime.deriv <- deriv(~ 2*x^3 - 3*x^2 - 12*x + 6, "x", TRUE)

#given a point on the curve and a slope
#chose x at index 10
x=seq(-4, 4, by=0.1)
i=11
x0=x[i]
f_x0 = fx(x0) #or y0
slope = fprime(x0)
tangentLine = function(x, slope, x0, y0) {
   y=slope*(x-x0) +y0
}
y=tangentLine(x, slope, x0, f_x0)
par(mfrow = c(2, 1)) # split display regio    ows and one column (i.e., 2
    regions)
x=seq(-4, 4, by=0.1)
```

f(x) = 2*x^3 - 3*x^2 - 12*x + 6

f'(x) or fprime(x)

f'(x) or fprime(x)

approx curve using tangent at (x0, f(x0))

tangent given (x0, f(x0)) and slope

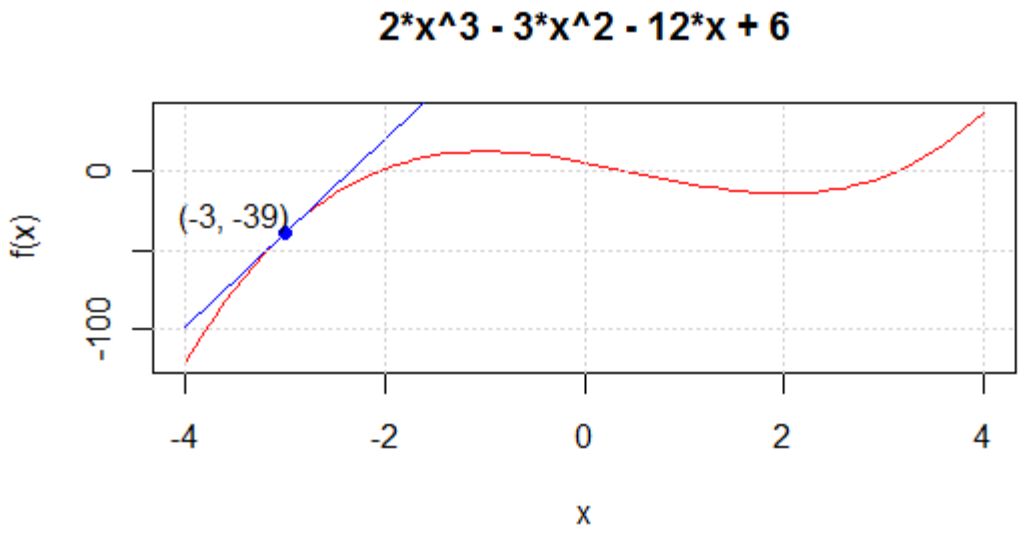
Cntd next Slide

Tangent Example: plotting

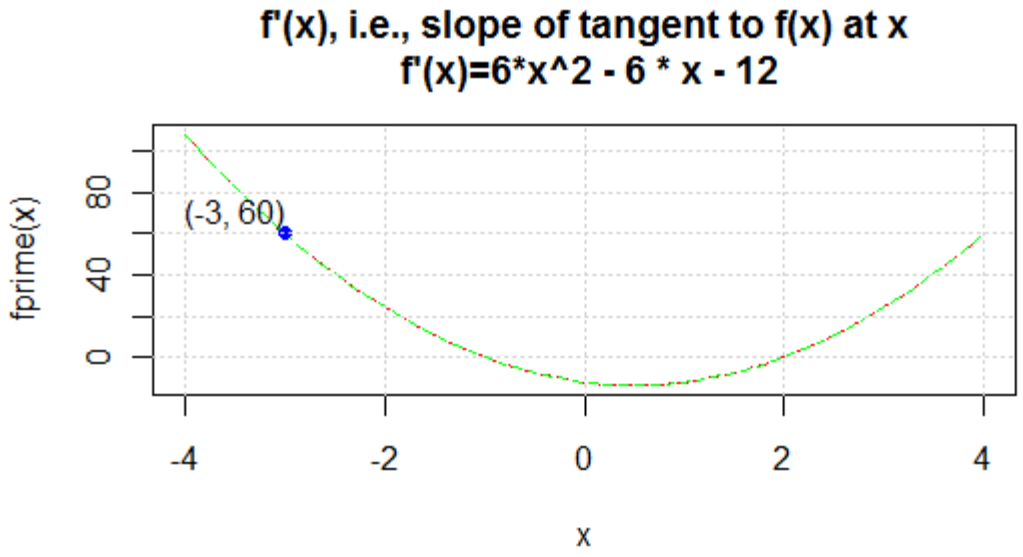
See example.drawTangent()

```
par(mfrow = c(2, 1)) # split display region into to 2 rows and one column (i.e., 2 regions)
x=seq(-4, 4, by=0.1)
plot(x, fx(x),  main="2*x^3 - 3*x^2 - 12*x + 6", xlab="x", ylab="f(x)", pch="")
lines(x, fx(x), lty=1, col="red")
lines(x, y,col="blue")
points(x0, f_x0, col="blue", bg="blue", pch=21)
text(x0-0.5, f_x0+10, paste("(", x0, ", ", f_x0, ")", sep=""))
grid()

plot(x, fprime(x), main="f'(x), i.e., slope of tangent to f(x) at x\n f'(x)=6*x^2 - 6 * x - 12
    ylab="fprime(x)", pch="")
lines(x, fprime(x), lty=3, col="red")  #hand calculate slope
lines(x, attr(fprime.deriv(x), "gradient"), lty=2, col="green")  #R-calculated slope
text(x0-0.5, slope+10, paste("(", x0, ", ", slope, ")", sep=""))

points(x0, slope, col="blue", bg="blue", pch=21)

grid()
```

ercise 1.1



the point of contact of the tangent and the curve is (-3, -39)



Slope is 60 (f'(-3)=60)

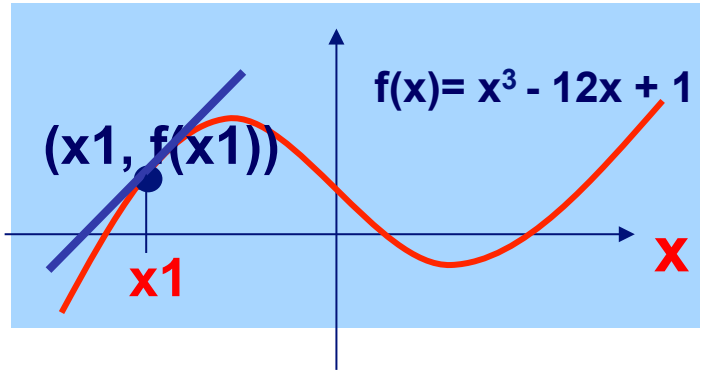
Summary on Tangent Approximations

$y - y1 = m(x - x1)$ give a point (x1, y1) and a slope m

$f(x) - y1 = f'(x1)(x - x1)$ let y = f(x) and m = $f'(x1)$

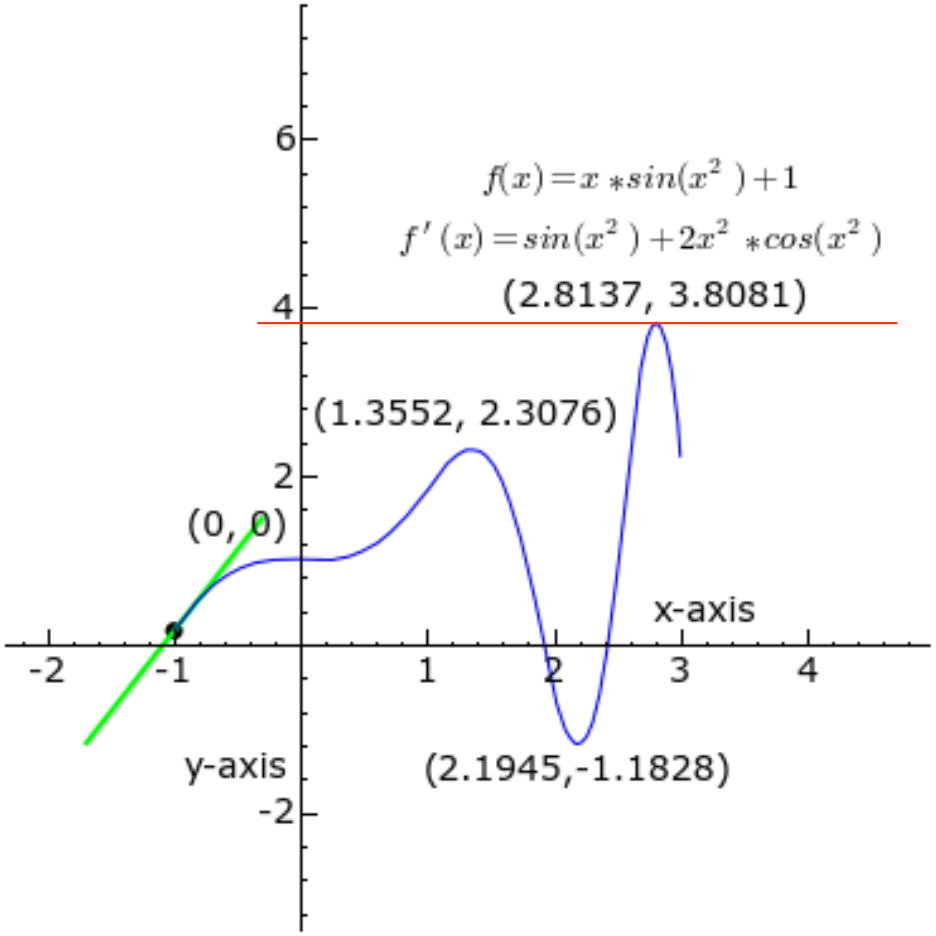
$f(x) = y1 + f'(x1)(x - x1)$

$f(x) = f(x1) + f'(x1)(x - x1)$ eqn of tangent at point (x1, f(x1)) given (x1, f(x1)) and slope = f' (x1)



- **Remember**
  - Every point on the curve has a tangent
  - A tangent is a straight line
  - The tangent has its own equation
  - The tangent has equation y = mx + c
  - This equation is different for every position of the tangent since the slope (f'(x)) is different.

A more complicated example



Y=mx+c

y-y0=m(x-x0)
Let m = f'(x0)

Lecture Outline

- **R Basics**
  - Most algorithms will be demonstrated with examples, code and homework problems
- **Lines, Tangents, Taylors Theorem, Roots of an equation**
- **Gradient Descent, Newton-Raphson**

At the turning point . . .

- The tangent will be horizontal
- The gradient of the tangent must be ???

0

- Find the root or zeros of an equation analytically by hand or numerically using iterative approaches such as Newton-Raphson, gradient descent, etc.
  - What value(s) of x will f'(x) =0 (gradient be zero).

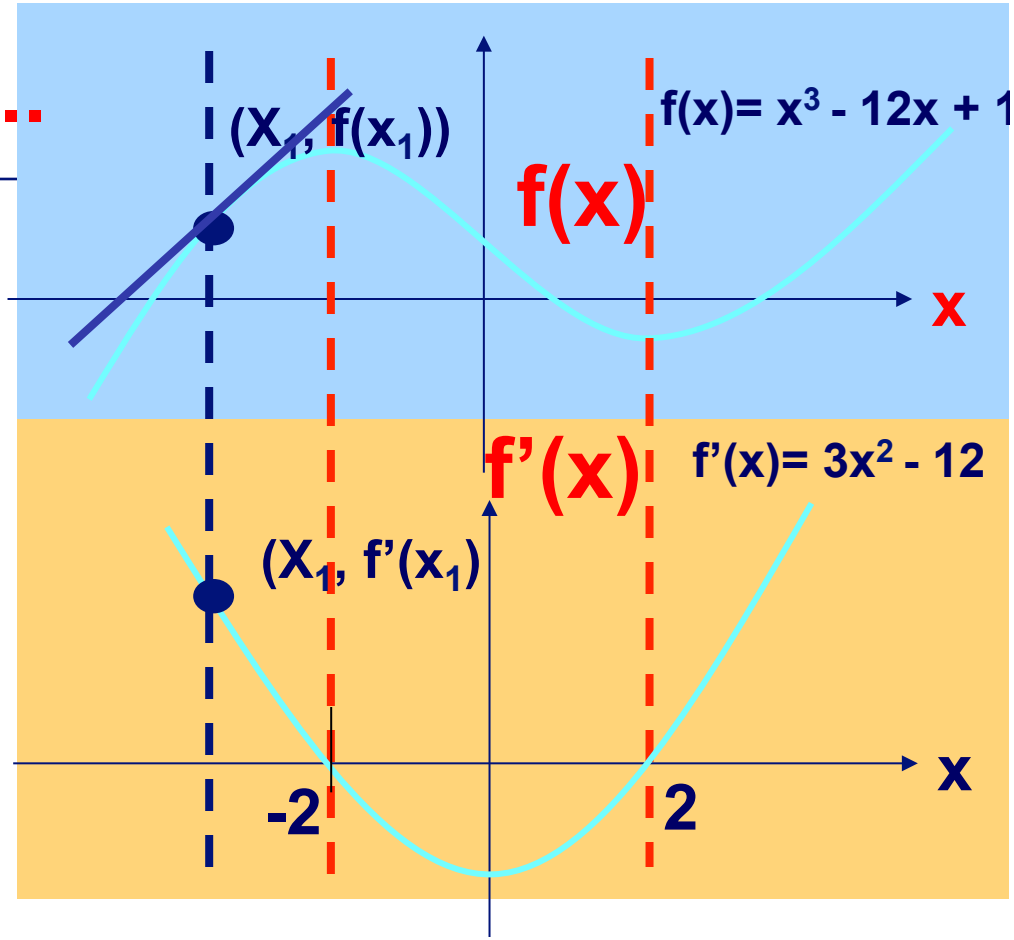
Given a Pt. and Slope...

$$f(x)= x^3 - 12x + 1$$

First derivative

$$f'(x) = 3x^2 - 12$$

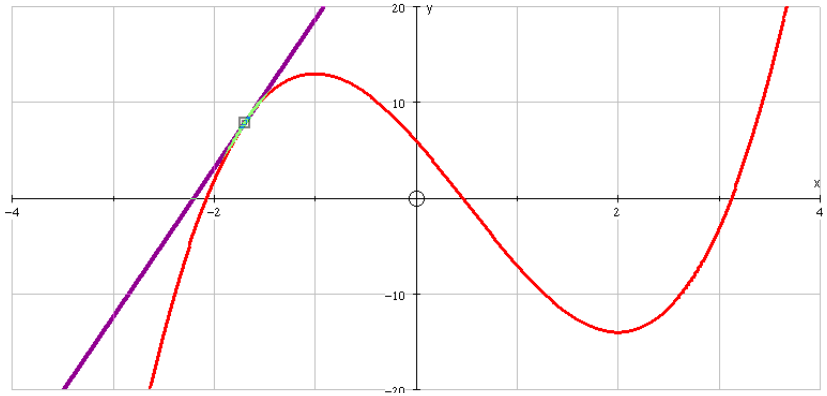
[=0 at maximum and minimum]



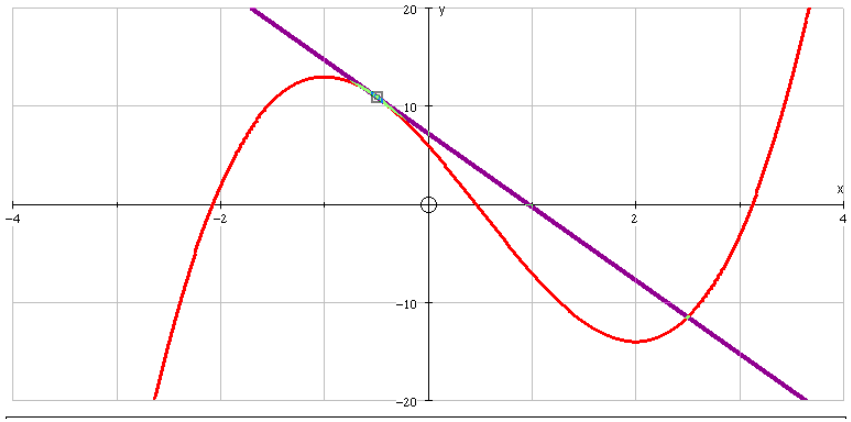
Given a Pt. and Slope... Approximate f(x) with tanger

Using $(x_1, f(x_1))$ and $m=f'(x_1)$
And the equation formula
$y-y0=m(x-x0)$
Plot the tangent line

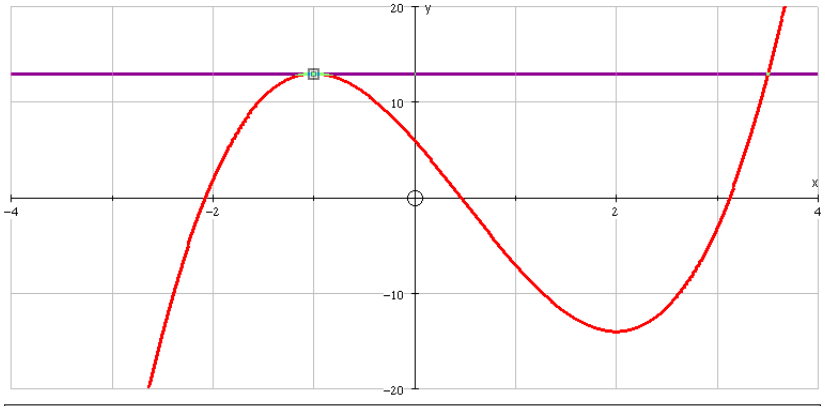
focus on the gradient of the tangent



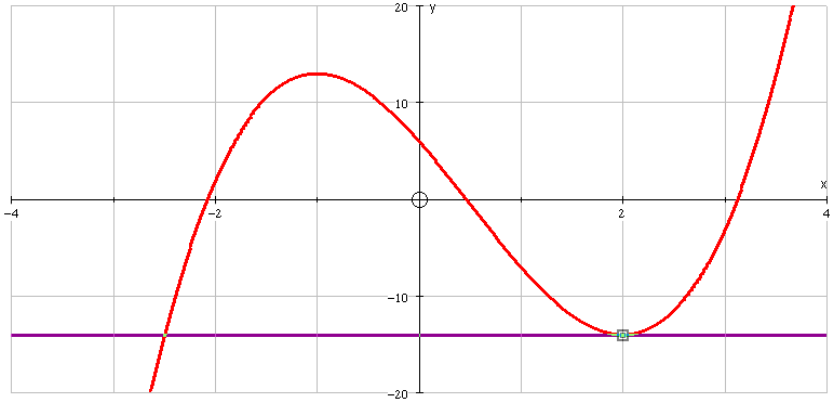
Gradient > 0



Gradient < 0



Gradient = 0



Gradient = 0

Finding turning points of f(x) by hand via f'(x)=0

$f(x) = 2x^3 - 3x^2 - 12x + 6$ # Function

STEP 1 $f'(x) = 6x^2 - 6x - 12$ # Gradient formula

STEP 2 Since the gradient of the tangent at the turning point is 0

$$6x^2 - 6x - 12 = 0$$

$$x^2 - x - 2 = 0$$

$$(x - 2)(x - 1) = 0$$

$x = 2$ or $x = -1$ Two turning points

When $x = 2$, $f(2) = 2(2)^3 - 3(2)^2 - 12(2) + 6 = -14$

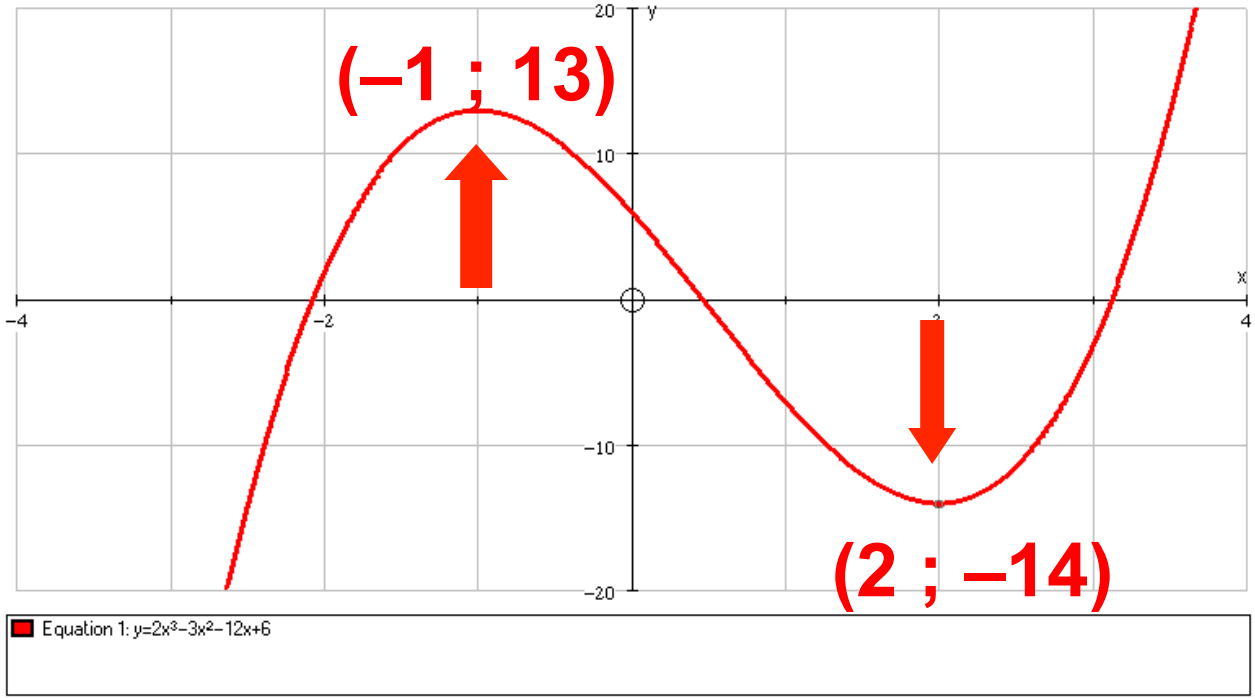
When $x = -1$, $f(-1) = 2(-1)^3 - 3(-1)^2 - 12(-1) + 6 = 13$

Turning Points (-1, 13) and (2, -14)

Calculate the coordinates of the turning points of the graph

USING CALCULUS and

Gradient Descent



Root Finding Algorithms

- Newton's Method Module
- Bisection Method Module [wont discuss here]
- Regula Falsi Method Module [wont discuss here]
- Fixed Point Iteration Module [wont discuss here]
- Secant Method Module [wont discuss here]

- **Click for Animations of the different approaches**
  - http://math.fullerton.edu/mathews/a2001/Animations/Animations2.html
-

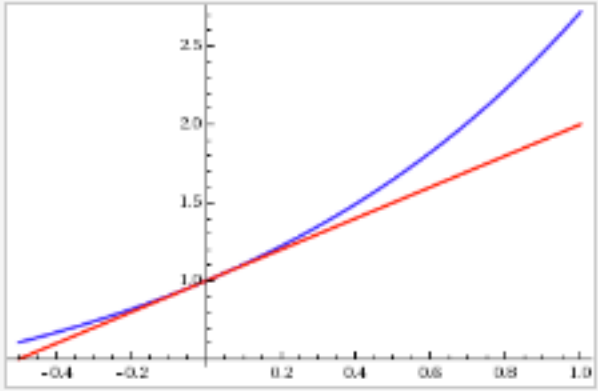
Newton-Raphson Method: A History

- **Solving a nonlinear equation of the form f(x)=0**
- **Isaac Newton developed an initial version of this algorithm in 1669 and published it in 1685; Raphson tweaked it 1690**

- **Extending it to a system of two equations**
  - In 1740, Thomas Simpson described Newton's method as an iterative method for solving general nonlinear equations using fluxional calculus, essentially giving the description above
  - In the same publication, Simpson also gives the generalization to systems of two equations and notes that Newton's method can be used for solving optimization problems by setting the gradient to zero.

http://en.wikipedia.org/wiki/Newton%27s_method

If a real-valued function f is differentiable at the point a then it has a linear approximation at the point a. This means that there exists a function h_1 such that



Graph of $f(x) = e^x$ (blue) with its linear approximation $P_1(x) = 1 + x$ (red) at $a = 0$.

$$f(x) = f(a) + f'(a)(x-a) + h_1(x)(x-a), \qquad \lim_{x \to a} h_1(x) = 0.$$

Here

$$P_1(x) = f(a) + f'(a)(x-a)$$

is the linear approximation of f at the point a. The graph of $y = P_1(x)$ is the tangent line to the graph of f at $x = a$. The error in the approximation is

$$R_1(x) = f(x) - P_1(x) = h_1(x)(x-a).$$

Finding the roots f(x) iteratively

In our case f(x) is f'(x) since we wish to solve f'(x)=0

- **An important problem in mathematics and statistics is finding values of x to satisfy f(x) = 0.**
  - Such values are called the roots of the equation and also known as the zeros of f(x).
- **Can solve analytically as we did above**
- **OR**
- **Various methods exist to numerically determine the roots of an equation or multiple equations**
  - Newton's method or the Newton-Raphson method is a procedure or algorithm for approximating the zeros of a function f (or, equivalently, the roots of an equation f(x) = 0).
  - Bisecting Method [wont discuss here]
  - Secant Method [wont discuss here]

Finding the roots f(x) i atively

In our case f(x) is f'(x) since w f'(x)=0

- **An important problem i statistics is findin = 0.**
  - Such values a and also known as
- **Can a above OR**
- **numerically determine ion or multiple equations**

or the Newton-Raphson method is a algorithm for approximating the zeros of a (or, equivalently, the roots of an equation f(x) = 0).

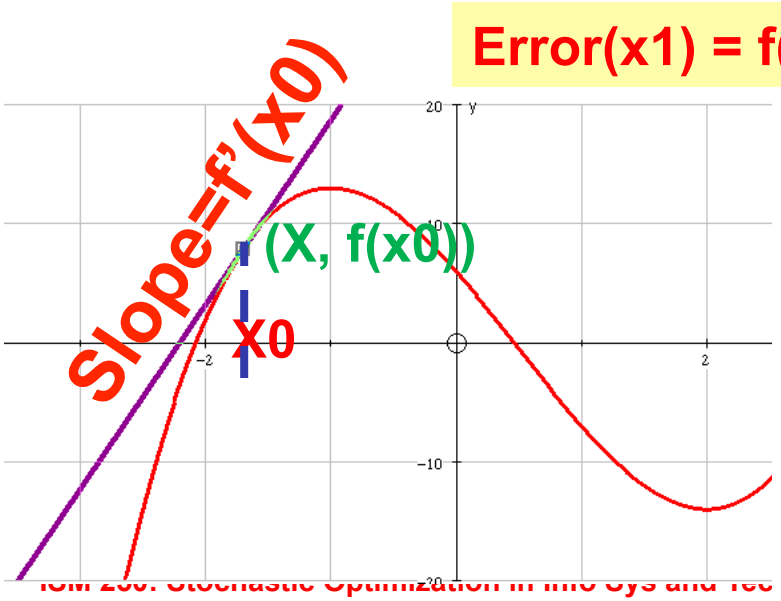
ecting Method [wont discuss here]

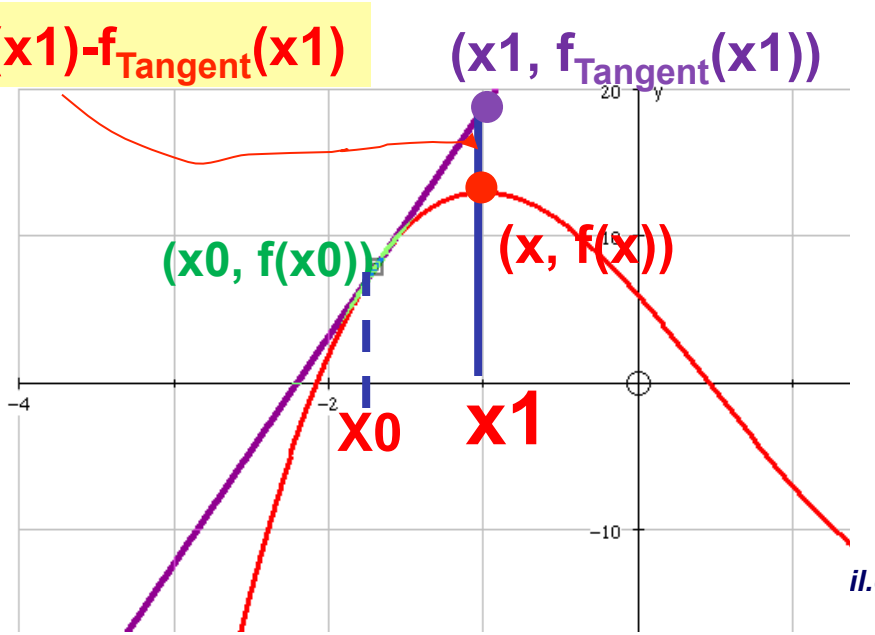
- Secant Method [wont discuss here]

Taylor Series tell us everything we need to know about finding a min or max f(x)=f(a)+f'(a)(x-a) + f''(a)(x-a)²/2..... Goal: find roots of f'(x) so we can locate Turning Points (candidate Max, Min) (Note Newton uses f'(x) and f''(x)) And use f''(x) to determine if candidate is a max or min

Recall: approximate a curve using a Tangent

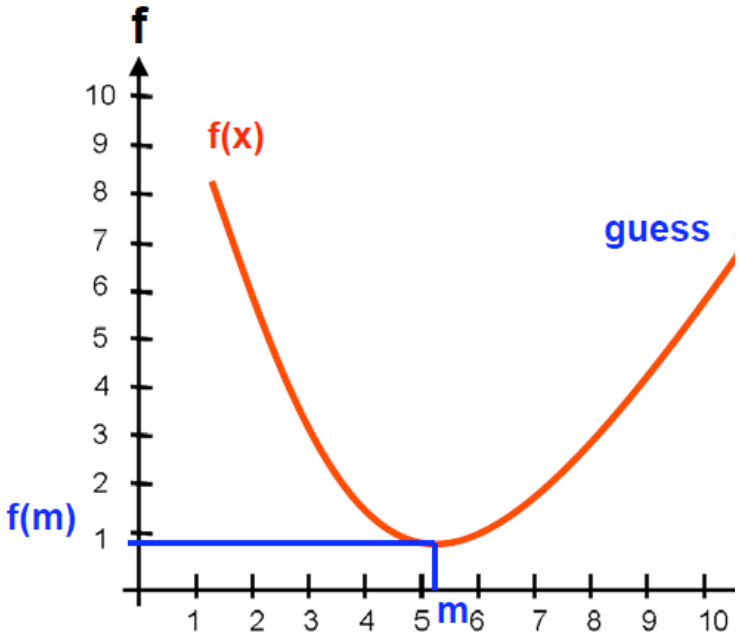
- **Given a point on the curve, (x0, f(x0)) and a slope, f'(x0), we can calculate the equation of the tangent at (x0, f(x0))**
  - y - y0 = f'(x0)(x - x0) ## y-y0 = m(x-x0)
  - f(X) = f(x0) + f'(x0)(X-x0) where X is a free variable
- **Then for any X1 in the neighbourhood of X0 we can approximate f(x1) it by the tangent at (x, f(x0)),**
  - i.e., f(**x1**) ~ $f_{tangent}$(**x1**) =f(x0) + f'(x0)(**X1**-x0)



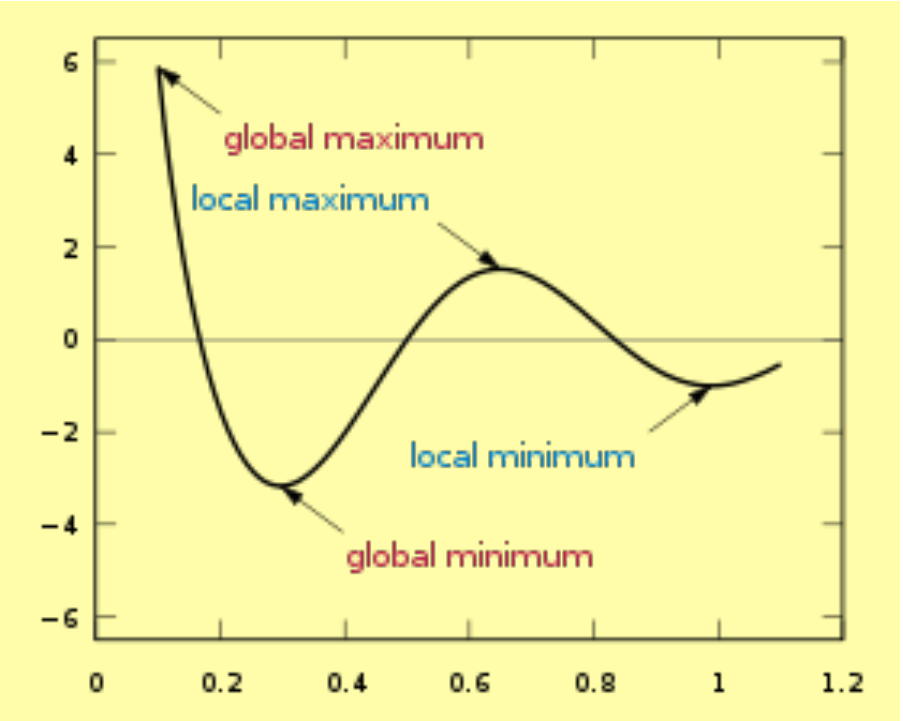


Focus on Convex Univariate problems

For the moment



Convex problem



Local and global maxima and minima for $\cos(3\pi x)/x$, $0.1 \le x \le 1.1$

Nonlinear Equations – Iterative Methods

Find the roots

- **Computers can NOT solve the roots in closed form (easily)**
- **Iterative Algorithm**
  - Start from an initial value x^0 as a candidate root (and also bracket the extrema).
  - Generate a sequence of iterate x^{n-1}, x^n, x^{n+1} which hopefully converges to the solution x^* (the root of f(x))
  - Iterates are generated according to an iteration function $F: x^{n+1}=F(x^n)$

Question

- When does it converge to correct solution ?
- What is the convergence rate ?

Given a Pt. and Slope..

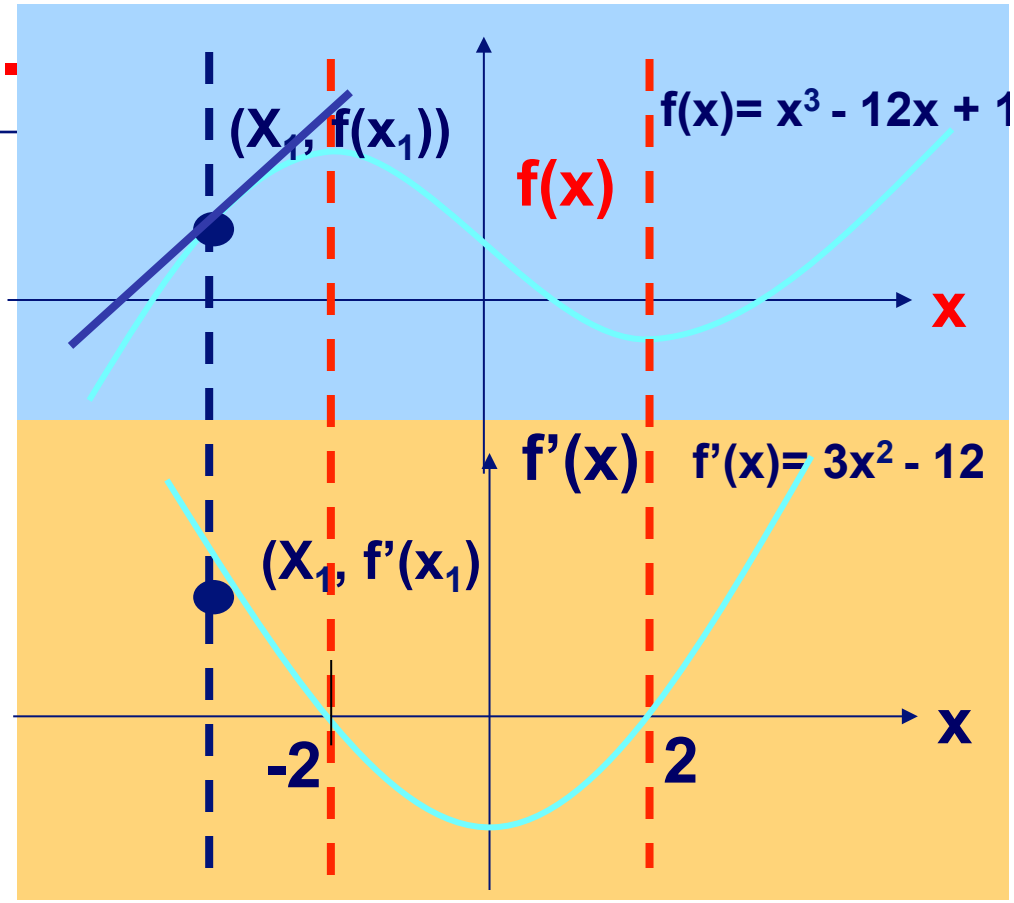
$$f(x)= x^3 - 12x + 1$$

First derivative

$$f'(x) = 3x^2 - 12$$

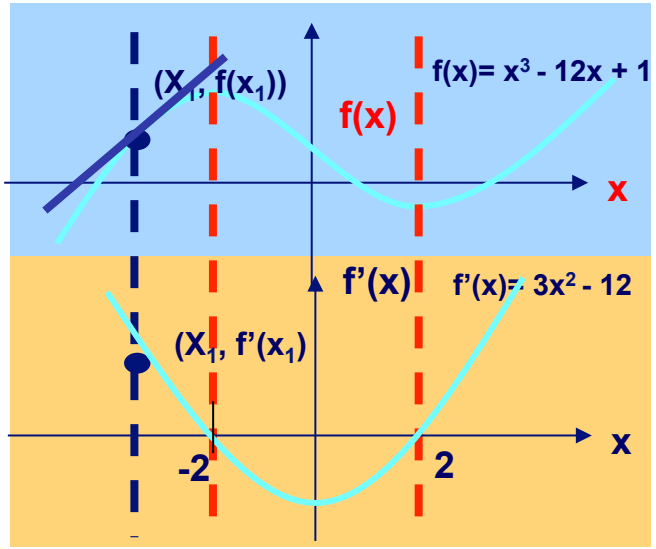
[=0 at maximum and minimum]

Using $(x_1, f(x_1))$ and $m=f'(x_1)$
And the equation formula
y-y0=m(x-x0)

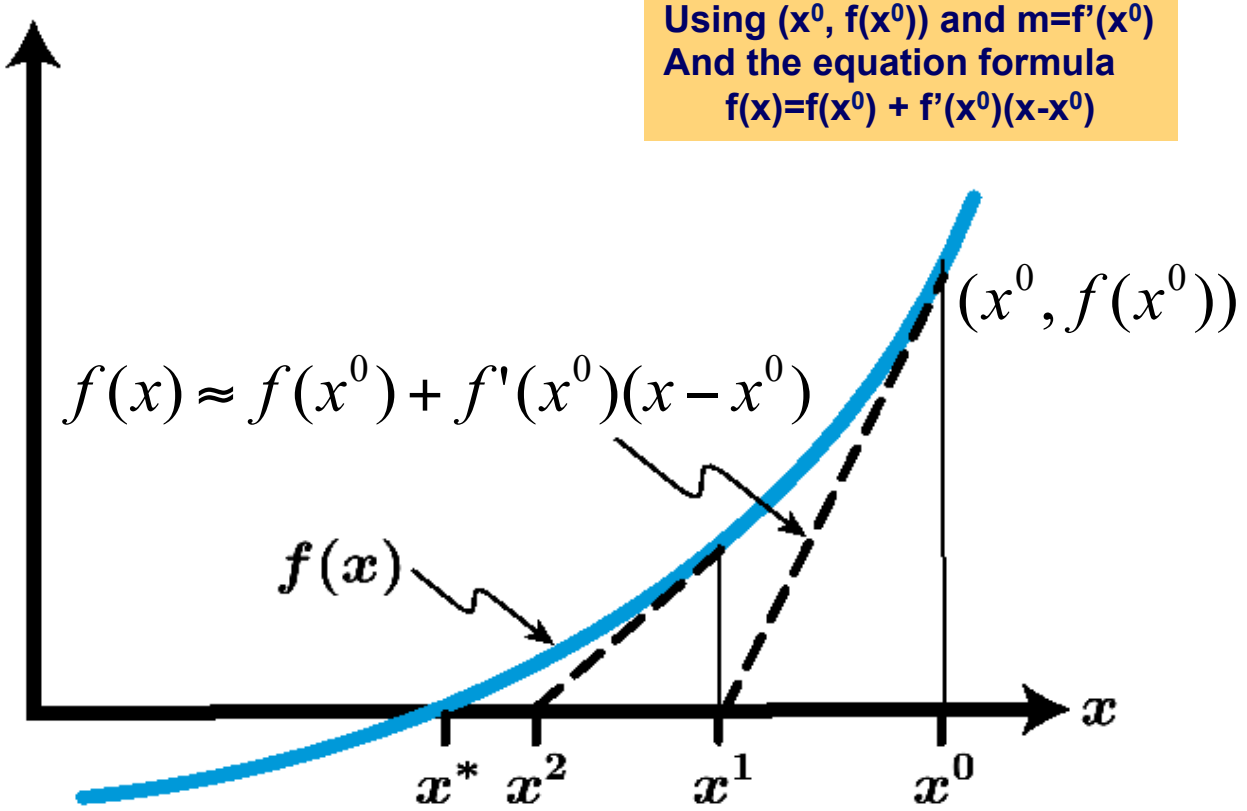


Find roots of f'(x) to give us candidate turning points

Newton-Raphson Method – Graphical View



Using $(x^0, f(x^0))$ and $m=f'(x^0)$
And the equation formula
$f(x)=f(x^0) + f'(x^0)(x-x^0)$



1. Initial guess: x^0 Letting i=0 $x^i = x^0$
2. Approximate f(x) by tangent at $(x^i, f(x^i))$ *# $(x^0, f(x^0))$ for the first iteration*
3. Find where $f_{Tangent\_x0}(x) = 0$, i.e., x^{i+1}; better approx. of the root (x^*)
4. Repeat until convergence

Deriving Newton-Raphson Method

- **Solving a nonlinear equation of the form f(x)=0**
- **Generate a sequence of iterate x^{n-1}, x^{n}, x^{n+1} which hopefully converges to the solution x^* (the root of f(x))**

$$f(x) \approx f(x^0) + f'(x^0)(x - x^0) \quad \forall x \text{ surrounding } x^0$$

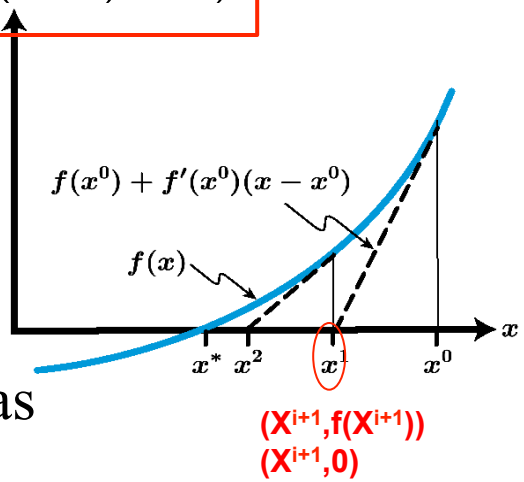
$$f(x^{i+1}) \approx f(x^i) + f'(x^i)(x^{i+1} - x^i) \quad \forall x \text{ surrounding } x^i$$

$$0 = f(x^i) + f'(x^i)(x^{i+1} - x^i) \qquad \text{We desire a root (i.e., } f(x^{i+1}) = 0)$$

$$f'(x^i)(x^{i+1} - x^i) = -f(x^1)$$

$$x^{i+1} = -\frac{f(x^i)}{f'(x^i)} + x^i \qquad \textit{Iteration function}$$

$$x^{i+1} = x^i - \left[\frac{df}{dx}(x^i)\right]^{-1} f(x^i) \qquad \text{sometimes written as}$$



Newton-Raphson (NR) Method

Consists of linearizing the system.

Want to solve f(x)=0 → Replace f(x) with its linearized version and solve.

$$f(x) = f(x^*) + \frac{df}{dx}(x^*)(x - x^*) \qquad 1^{st}\ order\ Taylor\ Series$$

$$f(x^{k+1}) = f(x^k) + \frac{df}{dx}(x^k)(x^{k+1} - x^k)$$

$$\Rightarrow x^{k+1} = x^k - \left[\frac{df}{dx}(x^k)\right]^{-1} f(x^k) \qquad Iteration\ function$$

Note: at each step need to evaluate *f* and *f'*

Newton-Raphson Method – Algorithm

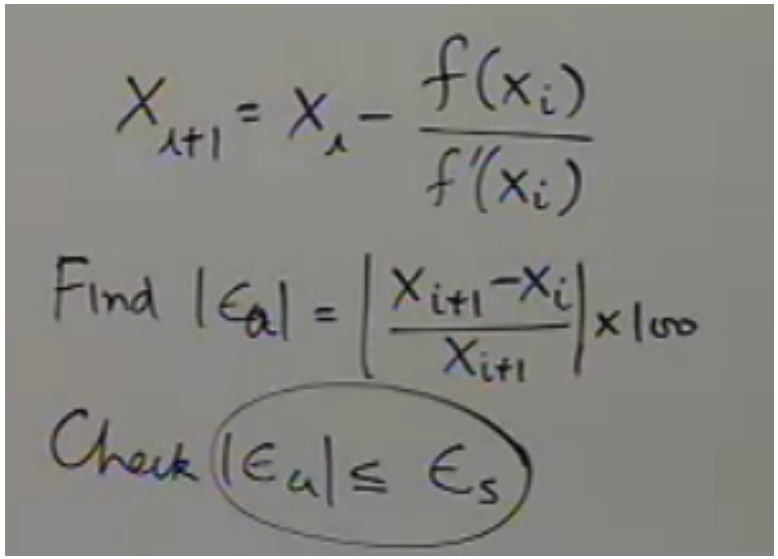
$x^1, x^2, x^3 \ldots, \ldots, x^*$

Define iteration

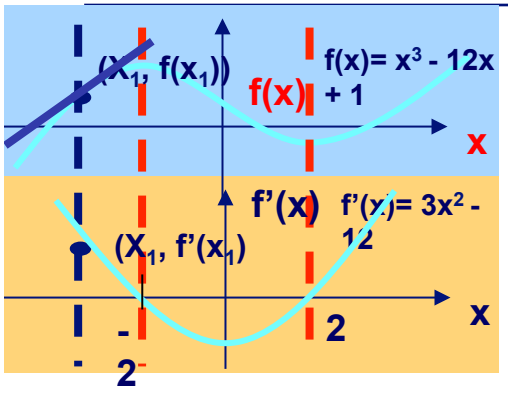
Do $k = 0$ to ….

$$x^{i+1} = x^i - \left[\frac{df}{dx}(x^i)\right]^{-1} f(x^i)$$

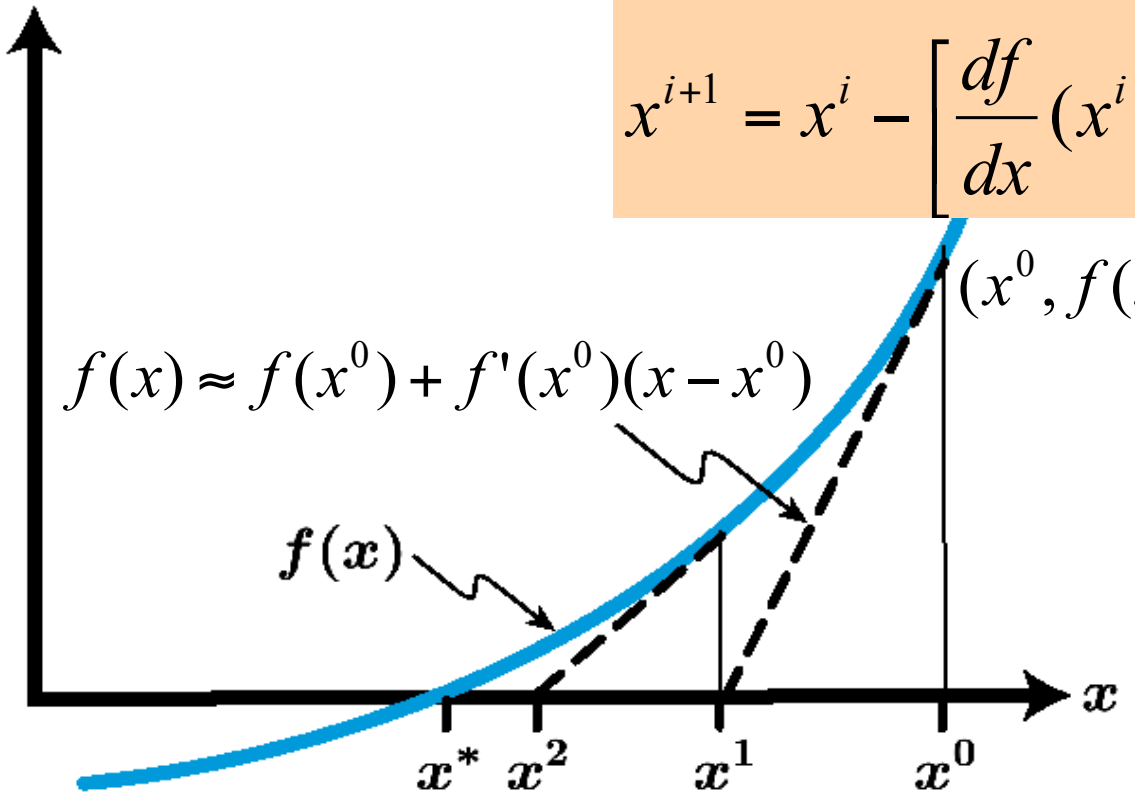
until convergence



Newton-Raphson Method – Graphical View



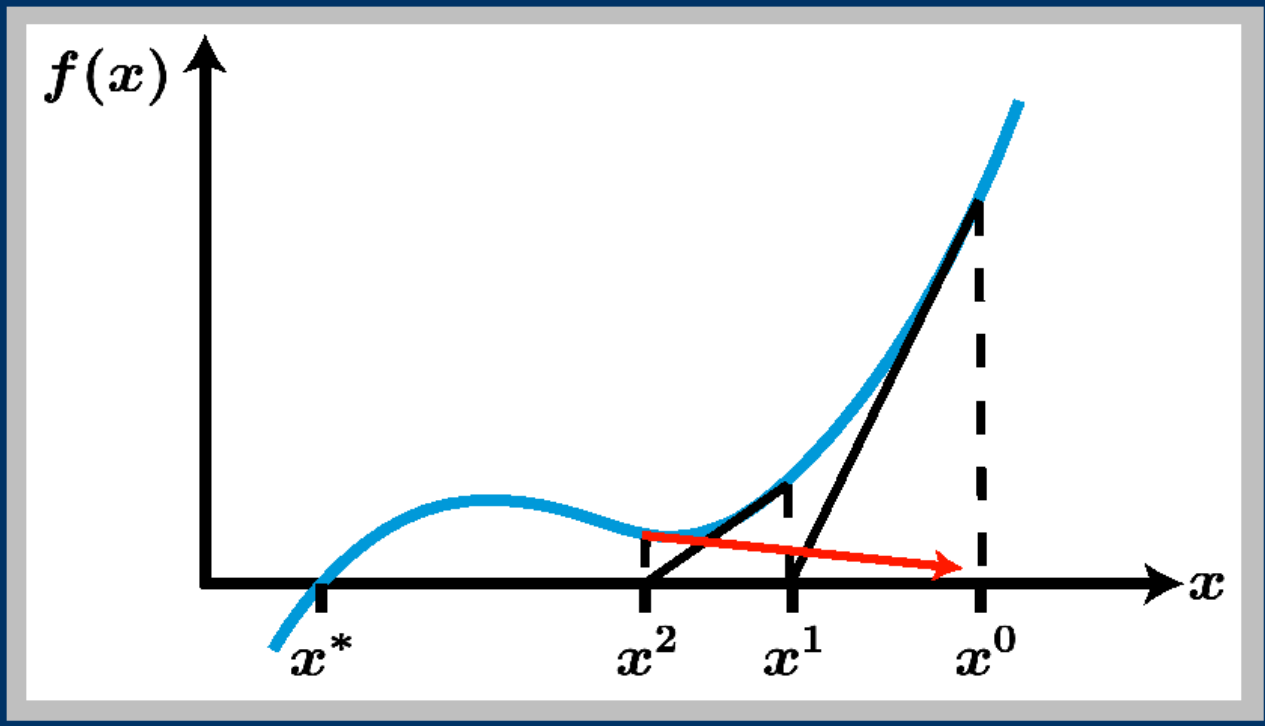
$$x^{i+1} = x^i - \left[\frac{df}{dx}(x^i)\right]^{-1} f(x^i)$$



1. Initial guess: x^0 Letting i=0 $x^i = x^0$
2. Approximate f(x) by tangent at $(x^i, f(x^i))$ *# $(x^0, f(x^0))$ for the first iteration*
3. Find where $f_{Tangent\_x0}(x) = 0$, i.e., x^{i+1}; better approx. of the root (x^*)
4. Repeat until convergence

Newton-Raphson Method – Convergence

We require that x^0 be "**close**" to the solution x^*

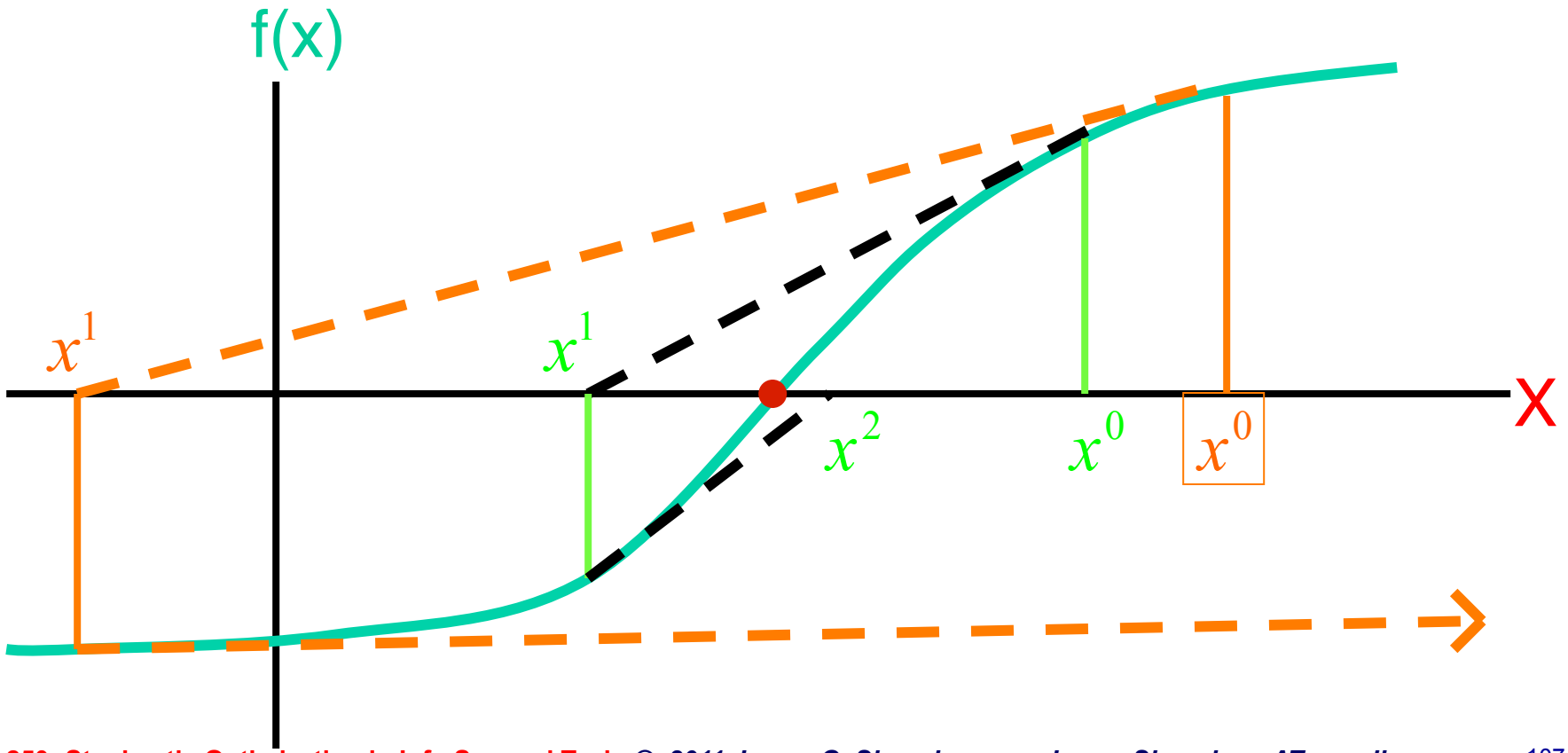


demo2

Newton-Raphson Method – Convergence

Local Convergence

Convergence Depends on a Good Initial Guess

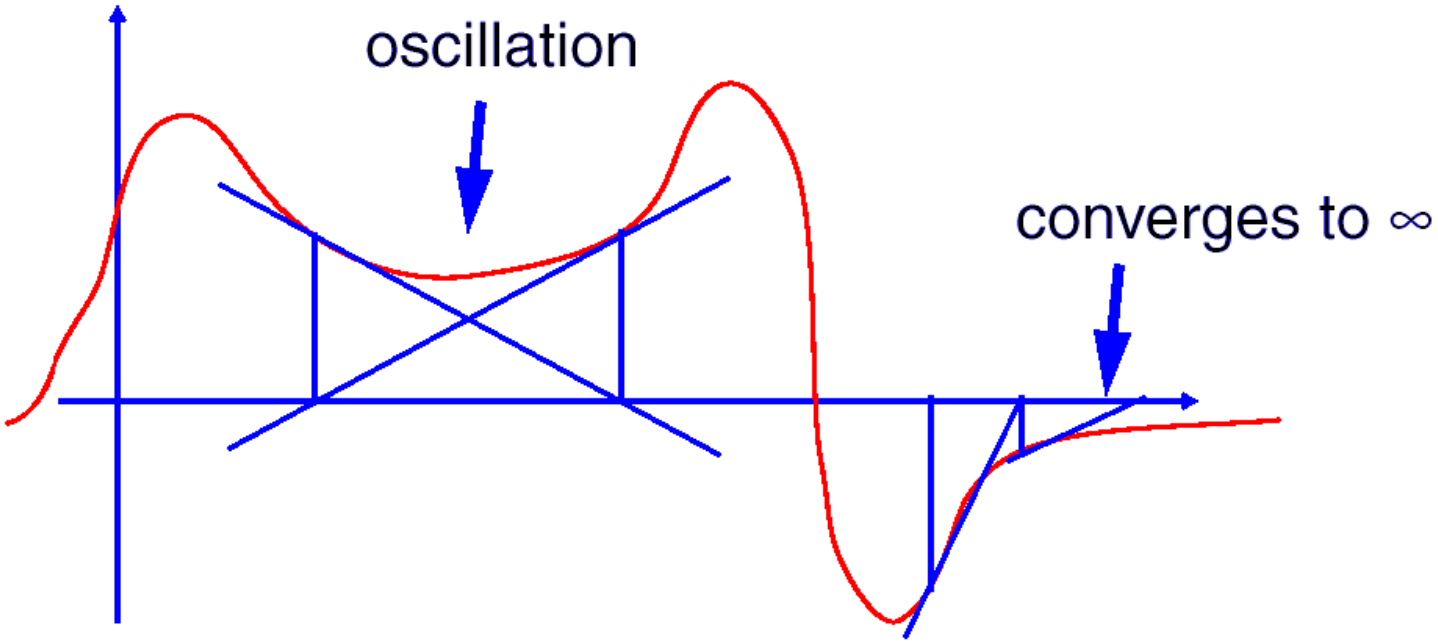


Newton-Raphson Method – Convergence

Local Convergence

Convergence Depends on a Good Initial Guess

Example:



Homework Problem: Find 2nd approx.

$$x^{i+1} = -\frac{f(x^i)}{f'(x^i)} + x^i = x^i - \left[\frac{df}{dx}(x^i)\right]^{-1} f(x^i) \quad \textit{Iteration function}$$

Taking 1 as the first approximation of a root of $x^3+2x-4=0$, use the Newton-Raphson method to calculate the second approximation of this root.

$f(x)=x^3+2x-4$ $\quad$ $f'(x)=3x^2+2$ $\quad$ $f(x) \approx f(x^0) + f'(x^0)(x-x^0)$

$f(1)=1+2-4=-1$

$f'(1)=3+2=5$

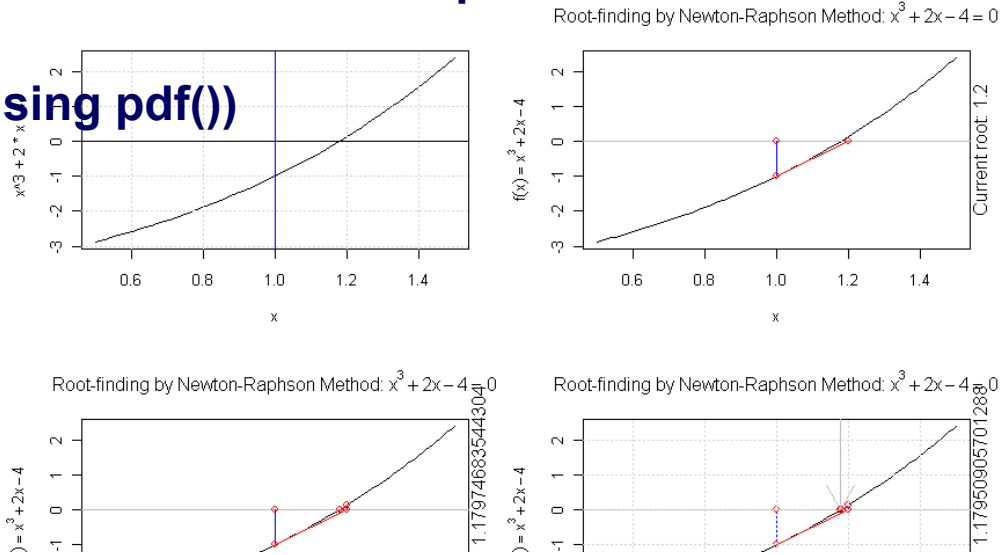
$$x_2 = 1 - \frac{-1}{5} = 1 + \frac{1}{5} = 1.2$$

Exercise 1.2

example.FindZerosOfDerivativeFunction()

- In R write a function that:

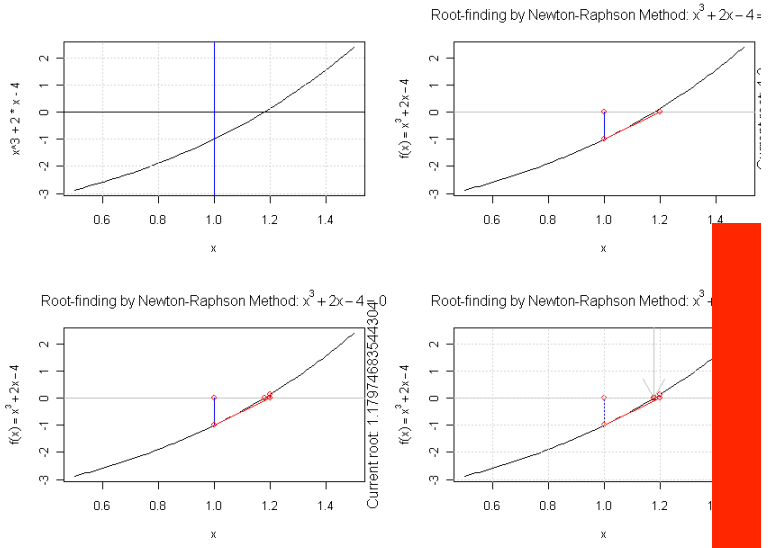
1. Find the zeros of the function x^3+2*x-4, 1 in the interval c(0.5, 1.5) starting with an initial guess of 1.4 using the Newton-Raphson method.
2. Plot the progress of the algorithm (See figure below for inspiration)
3. Comment on the convergence
4. HINT: you can use a publicly available function: *newton.method(function(x) x^3+2*x-4, 1, c(0.5, 1.5))* but for an extra little challenge please code your own Newton.Rapshon method and plot the progress
5. Save graphic animations to PDF (using pdf())

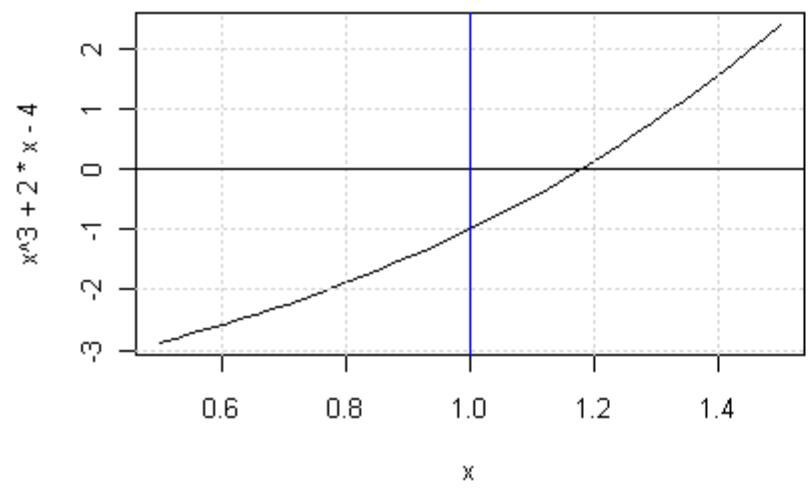


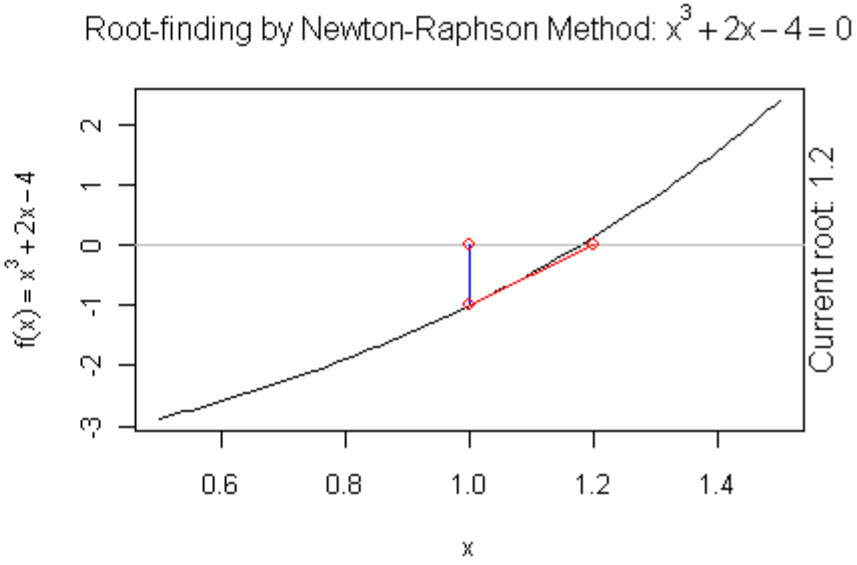
Exercise 1.2: Solution

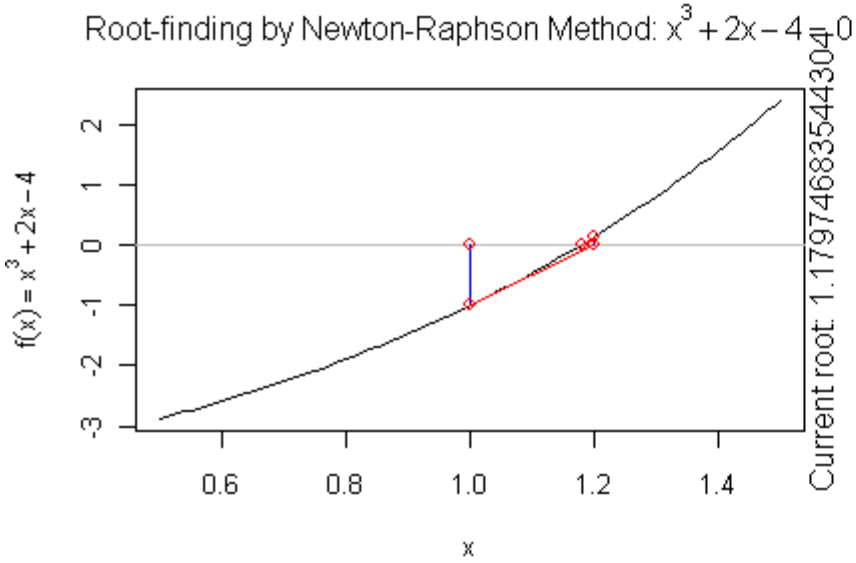
example.FindZerosOfDerivativeFunction()

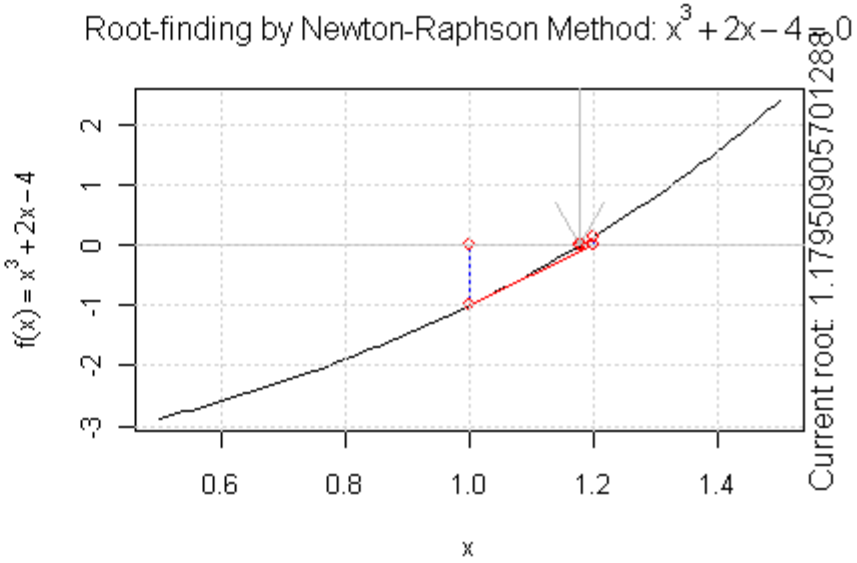
```
#find sequence of NewtonRaphson zero estimates for a function
# plots the sequence
example.FindZerosOfDerivativeFunction() {

  par(mfrow=c(2,2))
  x = seq(0.5, 1.5, len=40)
  plot(x, x^3+2*x-4, type="l")
  grid()
  abline(h=0)
  abline(v=1, col="blue")
  newton.method(function(x) x^3+2*x-4, 1, c(0.5, 1.5))
  grid()
}
```




x^3 + 2 * x - 4
x


Root-finding by Newton-Raphson Method: x^3 + 2x - 4 = 0
f(x) = x^3 + 2x - 4
Current root: 1.2
x


Root-finding by Newton-Raphson Method: x^3 + 2x - 4 = 0
f(x) = x^3 + 2x - 4
Current root: 1.179746835443041
x


Root-finding by Newton-Raphson Method: x^3 + 2x - 4 = 0
f(x) = x^3 + 2x - 4
Current root: 1.179509057012881
x

Exercise 1.3

Taking $x_1=1$, and using two iterations, obtain an approximation to a root of the equation. x^3+3x^2-x-2 by the Newton-Raphson method.

f

u

6.547

Exercise 1.3

Taking $x_1=1$, and using two iterations, obtain an approximation to a root of the equation. x^3+3x^2-x-2 by the Newton-Raphson method.

$$f(1)=1+3-1-2=1$$

$$f'(x)=3x^2+6x-1$$

$$f'(1)=3+6-1=8$$

$$u_2=1-\frac{1}{8}=.875$$

$$f(.875)=(.875)^3+3(.875)^2-(.875)-2$$

$$=.6699+2.2969-.875-2=.0918$$

$$f'(.875)=3(.875)^2+6(.875)-1=2.297+5.25-1=6.547$$

$$u_3=.875-\frac{.0918}{6.547}=.875-.0140=.8610$$

Let's simplify to convex problems

- **One global maximum or minimum of a univariate function, e.g., f(x) =x^2**
  - Will provide more formal definition shortly
- **Assume function f(x) =x^2, find the x value that minimizes f(x)**

$$\arg\min_{x \in \Omega_X} f(x)$$

The value of *x* that maximises *f(x).* For example,

$$\arg\min_{x \in \{1,2,-3\}} f(x^2) = 1$$

Root-Finding of f(x) in R using NR Alg.

```
#find the squareroot of 10;  #x^2 = 4 then f(x) = x^2-4
 f=function(x){x^2-4}
 fd=function(x){2*x}
```

F(x)=X^2 - 4

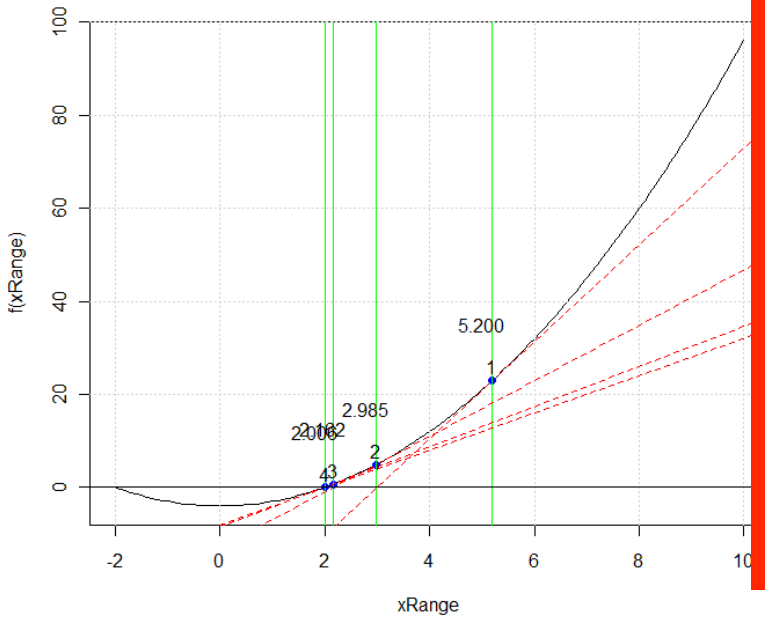
Newton-Raphson Algorithm

```
newtonRaphsonInOneDim=function(x0, n, xRange, f, fd){
 x=x0
 for (i in 1:n){
     x=x-(f(x)/fd(x))    #browser()
 }
 list(x) # return x
}
root = newtonRaphsonInOneDim(5.2, 4, xRange, f, fd)
```

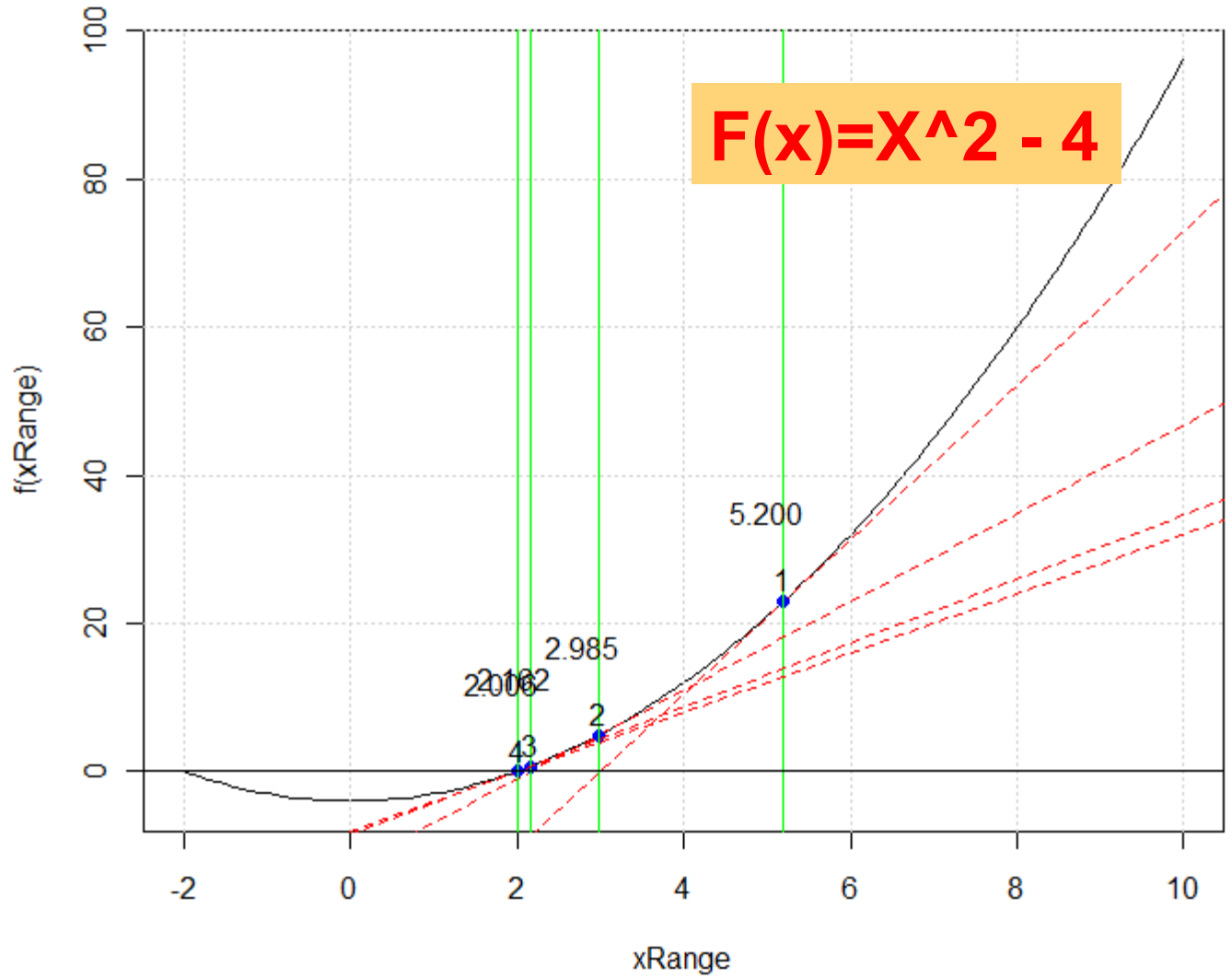
Find zero of f(x) in R with Graphics

```
#find the squareroot of 10;  #x^2 = 4 then f(x) = x^2-4
 f=function(x){x^2-4}    #f(x)
 fd=function(x){2*x}      #f'(x)

newtonRaphsonInOneDim=function(x0, n, xRange, f, fd){
 x=x0
 plot(xRange, f(xRange), type="l")
 grid()
 for (i in 1:n){
   print(paste("interation", i, "x, -f(x)/fd(x), newX", x, -(f(x)/fd(x)), x-(f(x)/fd(x))))
   slope = fd(x)
   intercept = -1*slope*x +f(x)
   abline(intercept, slope, lty=2, col="red")
   points(x, f(x), col="blue", bg="blue", pch=21)
   abline(v=x, col="green");   abline(h=0)
   text(x-.01, f(x)+3, i)
   text(x-.2, f(x)+12, format(x, digits = 2, nsmall = 3))
   x=x-(f(x)/fd(x))    #browser()
 }
 list(x) # return x
}
 newtonRaphsonInOneDim(5.2, 4, xRange, f, fd)
```

F(x)=X^2 - 4



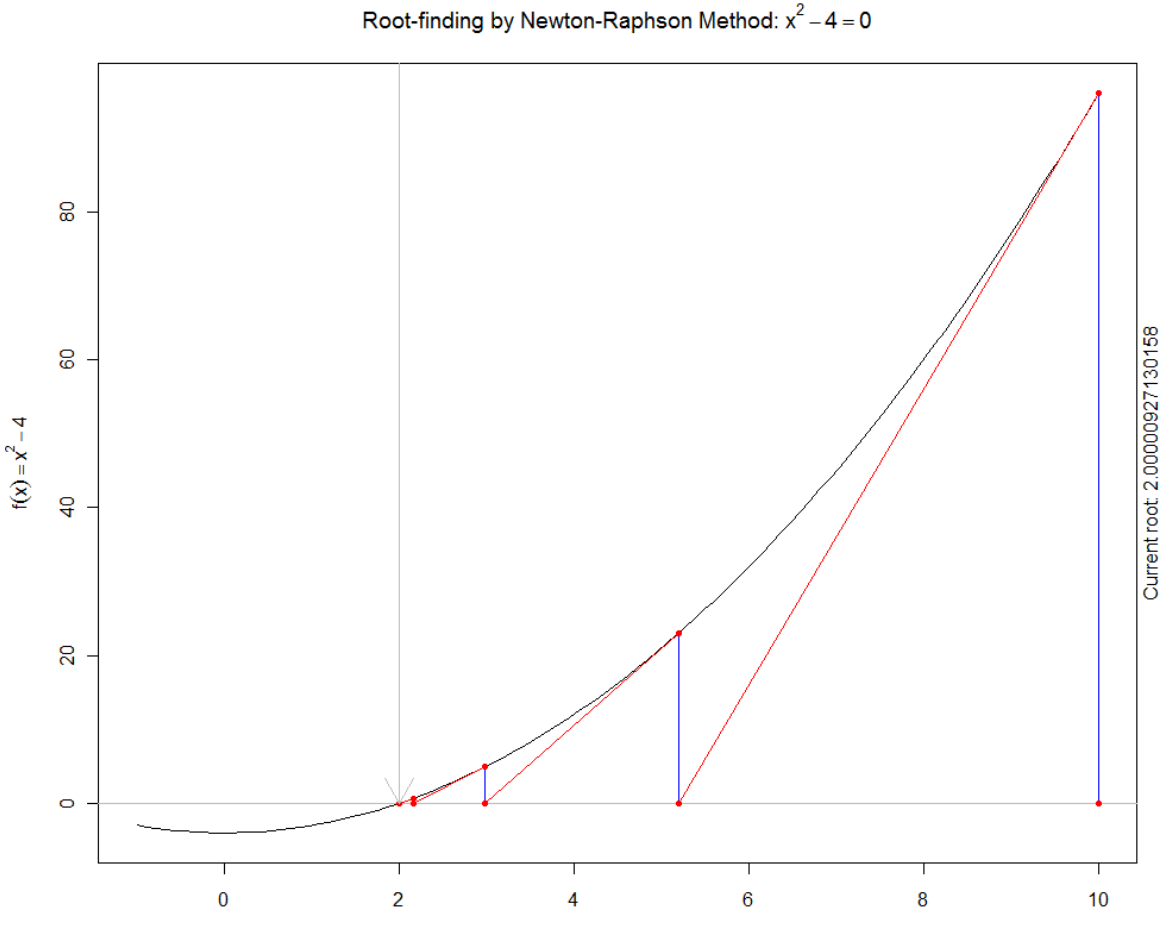
Find the root of f(x) = x^2 - 4

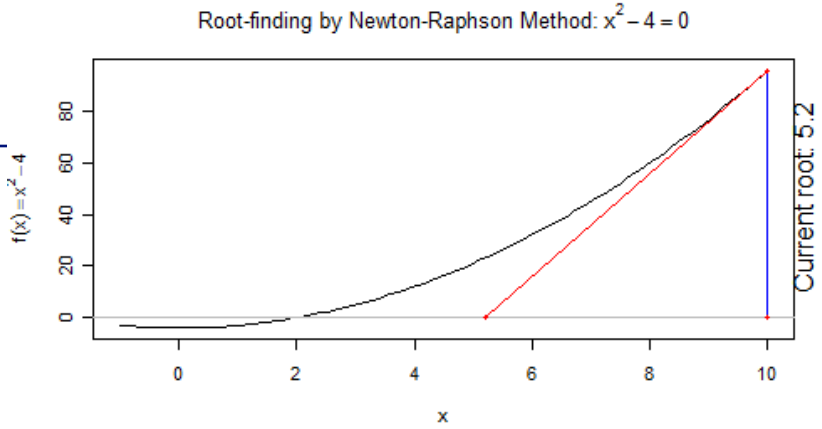


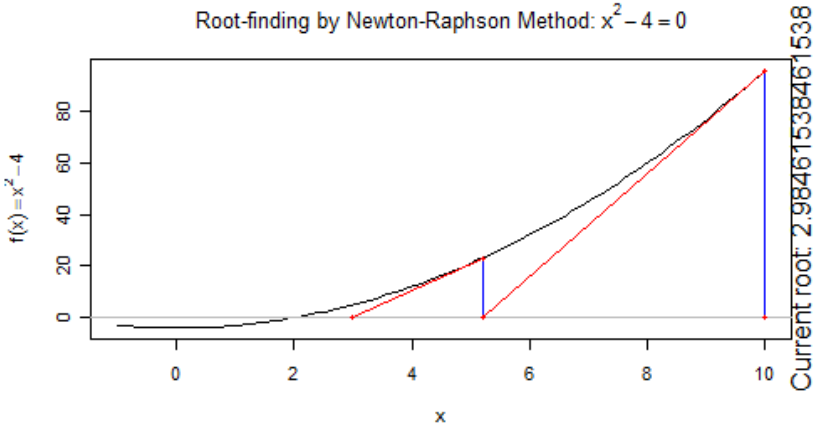
T_gmail.com

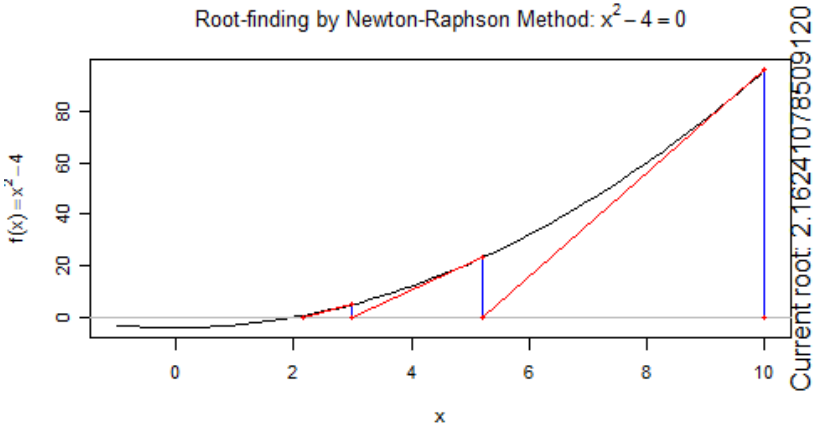
Using built-in method for NR method

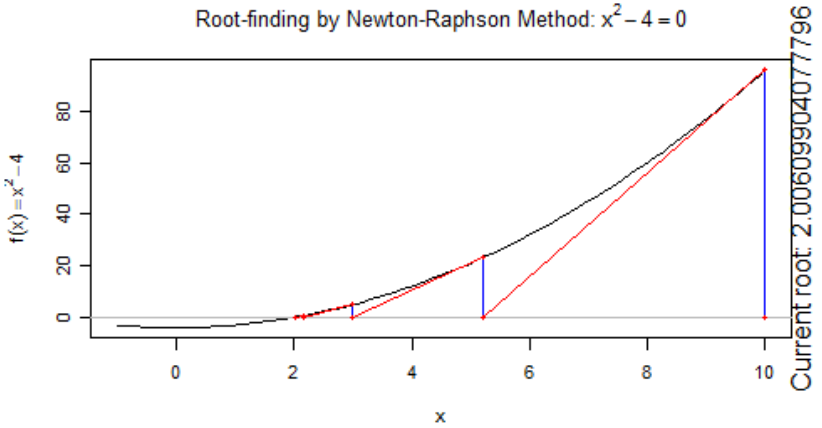
```
par(mfrow=c(3,2))
newton.method(function(x) x^2, -2, c(-4, 4))
```

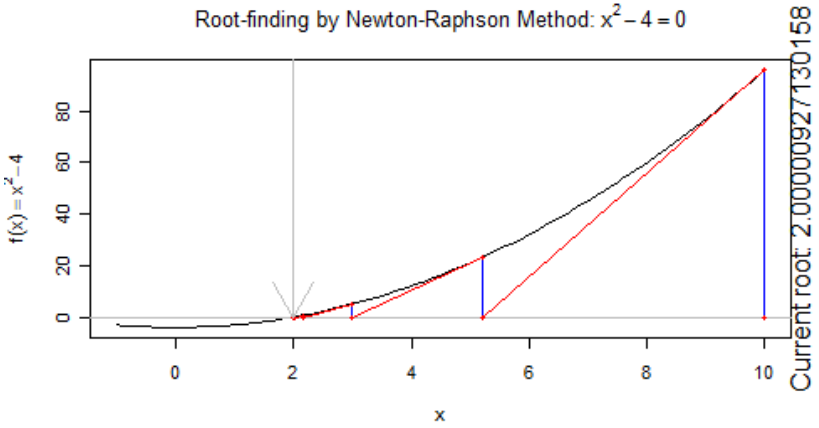


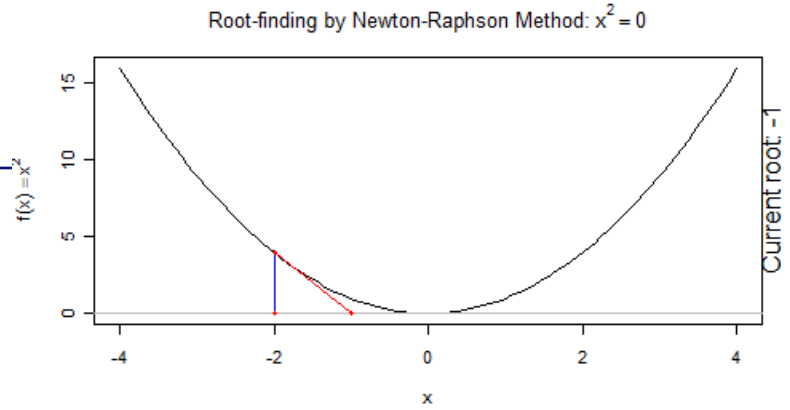

Root-finding by Newton-Raphson Method: $x^2 - 4 = 0$
f(x) = x^2 − 4
x
Current root: 5.2

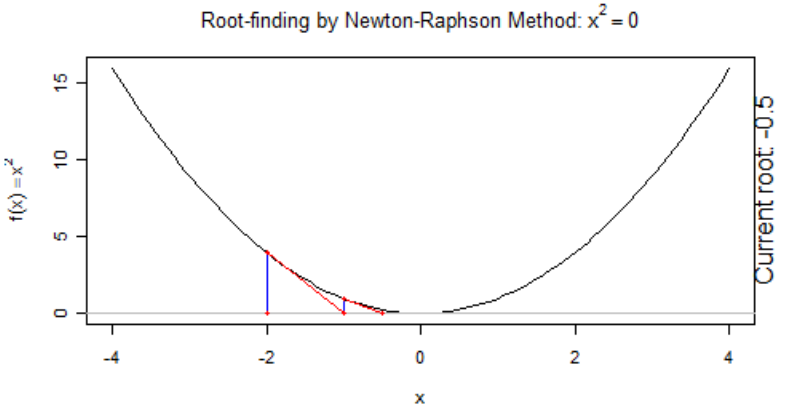

Root-finding by Newton-Raphson Method: $x^2 - 4 = 0$
f(x) = x^2 − 4
x
Current root: 2.98461538461538

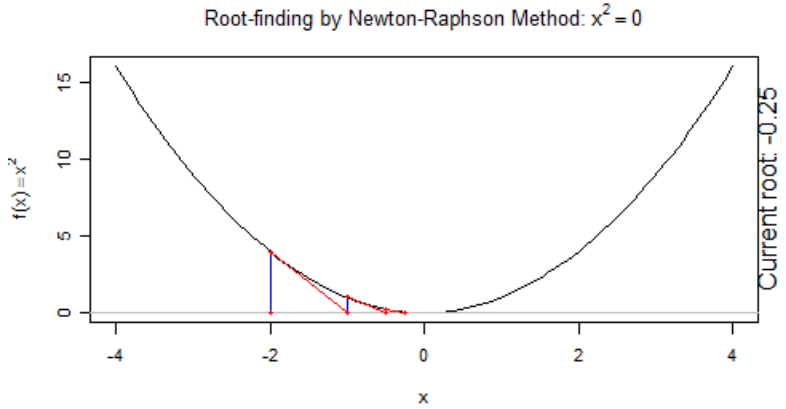

Root-finding by Newton-Raphson Method: $x^2 - 4 = 0$
f(x) = x^2 − 4
x
Current root: 2.16241078509120

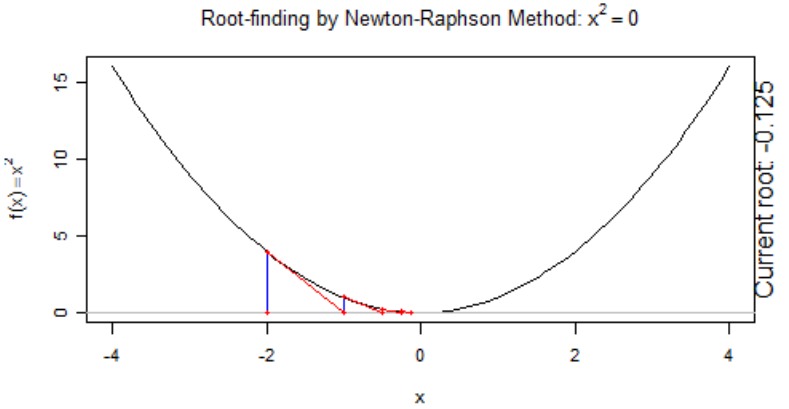

Root-finding by Newton-Raphson Method: $x^2 - 4 = 0$
f(x) = x^2 − 4
x
Current root: 2.00609904077796

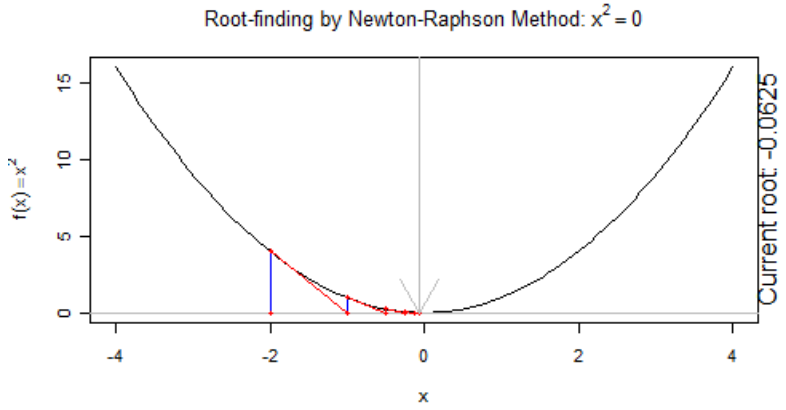

Root-finding by Newton-Raphson Method: $x^2 - 4 = 0$
f(x) = x^2 − 4
x
Current root: 2.00000927130158


Root-finding by Newton-Raphson Method: $x^2 = 0$
$f(x) = x^2$
Current root: -1
x


Root-finding by Newton-Raphson Method: $x^2 = 0$
$f(x) = x^2$
Current root: -0.5
x


Root-finding by Newton-Raphson Method: $x^2 = 0$
$f(x) = x^2$
Current root: -0.25
x


Root-finding by Newton-Raphson Method: $x^2 = 0$
$f(x) = x^2$
Current root: -0.125
x


Root-finding by Newton-Raphson Method: $x^2 = 0$
$f(x) = x^2$
Current root: -0.0625
x

Exercise (not required)

- **Calculate the root of the following equation**
  - x^3
  - HINT: use newton.method(function(x) x^3, -4, c(-10, 4))
  - How many iterations does of the Newton-Raphson algorithm?
- **Save graphic animations to PDF (using pdf())**

Homework Problem: Find 2nd approx.

$$x^{i+1} = -\frac{f(x^i)}{f'(x^i)} + x^i \quad \textit{Iteration function}$$

Taking 1 as the first approximation of a root of $x^3+2x-4=0$, use the Newton-Raphson method to calculate the second approximation of this root.

$f(x)=x^3+2x-4$ $\quad$ $f'(x)=3x^2+2$ $\quad$ $f(x) \approx f(x^0) + f'(x^0)(x - x^0)$

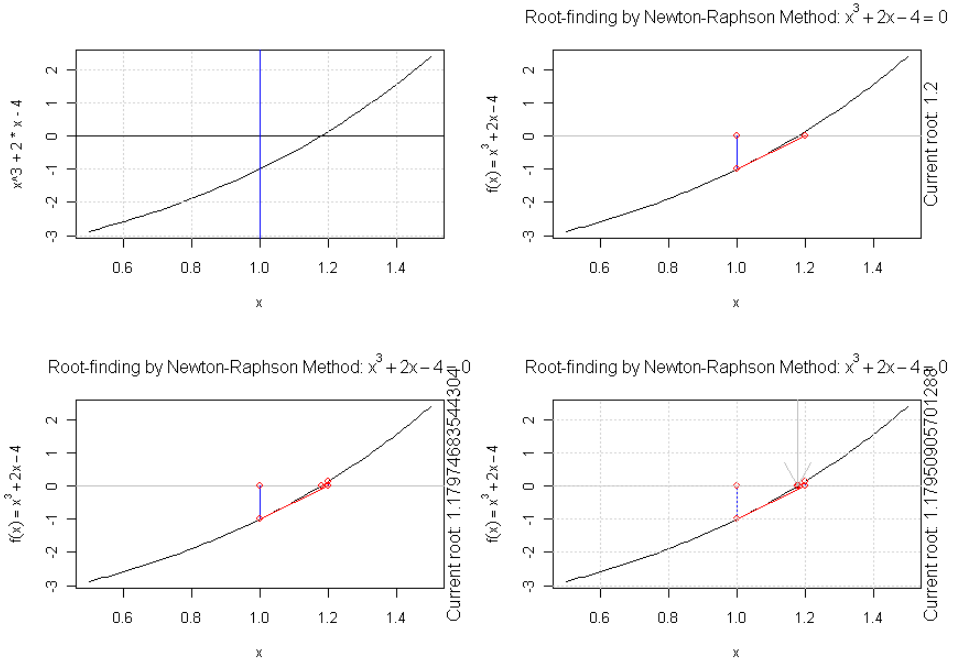
$f(1)=1+2-4=-1$

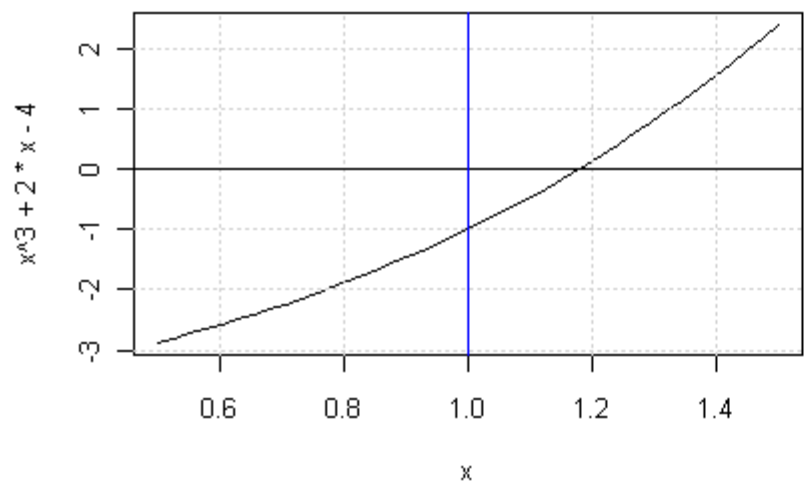
$f'(1)=3+2=5$

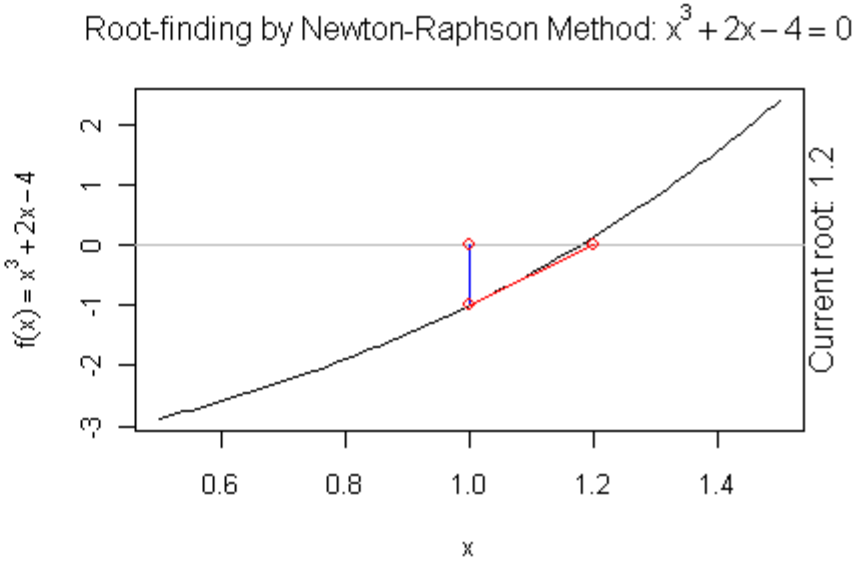
$$x_2 = 1 - \frac{-1}{5} = 1 + \frac{1}{5} = 1.2$$

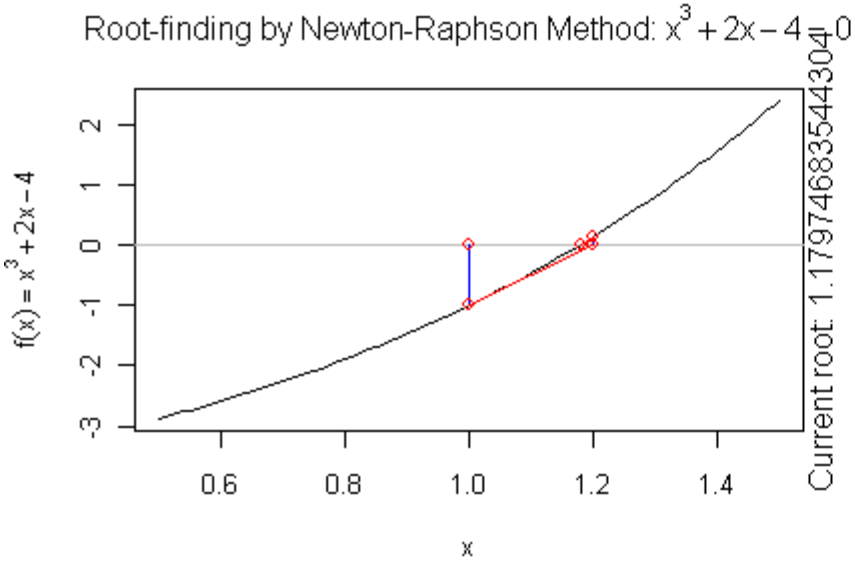
Homework Solution: Plot 2nd …

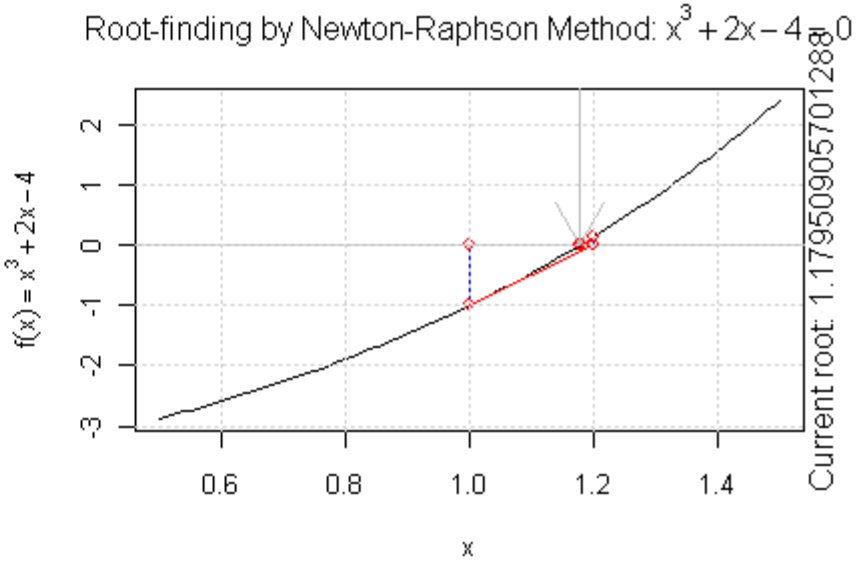
```
#find sequence of NewtonRaphson zero estimates for a
    function
# plots the sequence
example.FindZerosOfDerivativeFunction() {

  par(mfrow=c(2,2))
  x = seq(0.5, 1.5, len=40)
  plot(x, x^3+2*x-4, type="l")
  grid()
  abline(h=0)
  abline(v=1, col="blue")
  newton.method(function(x)
  grid()
}
```




x^3 + 2 * x - 4
x


Root-finding by Newton-Raphson Method: x^3 + 2x - 4 = 0
f(x) = x^3 + 2x - 4
Current root: 1.2
x


Root-finding by Newton-Raphson Method: x^3 + 2x - 4 = 0
f(x) = x^3 + 2x - 4
Current root: 1.17974683544304
x


Root-finding by Newton-Raphson Method: x^3 + 2x - 4 = 0
f(x) = x^3 + 2x - 4
Current root: 1.17950905701288
x

Finding the roots f(x) iteratively

In our case f(x) is f'(x) since we wish to solve f'(x)=0

- **An important problem in mathematics and statistics is finding values of x to satisfy f(x) = 0.**
  - Such values are called the roots of the equation and also known as the zeros of f(x).
- **Can solve analytically as we did above OR**
- **Various methods exist to numerically determine the roots of an equation or multiple equations**
  - Newton's method or the Newton-Raphson method is a procedure or algorithm for approximating the zeros of a function f (or, equivalently, the roots of an equation f(x) = 0).
  - Bisecting Method [wont discuss here]
  - Secant Method [wont discuss here]

Finding the roots f(x) i atively

In our case f(x) is f'(x) since w f'(x)=0

- **An important problem i statistics is findin = 0.**
  - Such values a and also known as
- **Can a above OR**
- **numerically determine ion or multiple equations**

or the Newton-Raphson method is a algorithm for approximating the zeros of a (or, equivalently, the roots of an equation f(x) = 0).

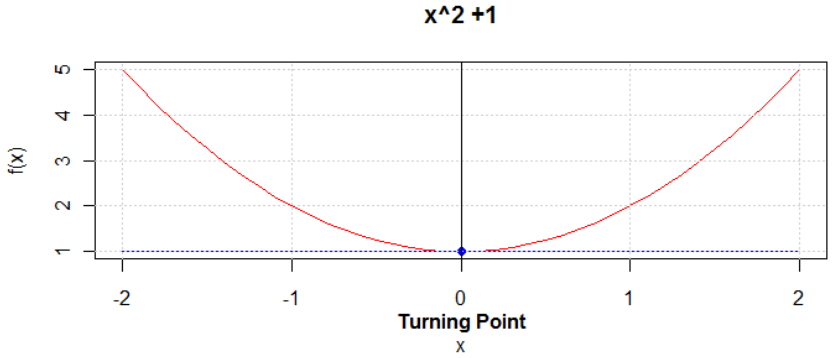
ecting Method [wont discuss here]

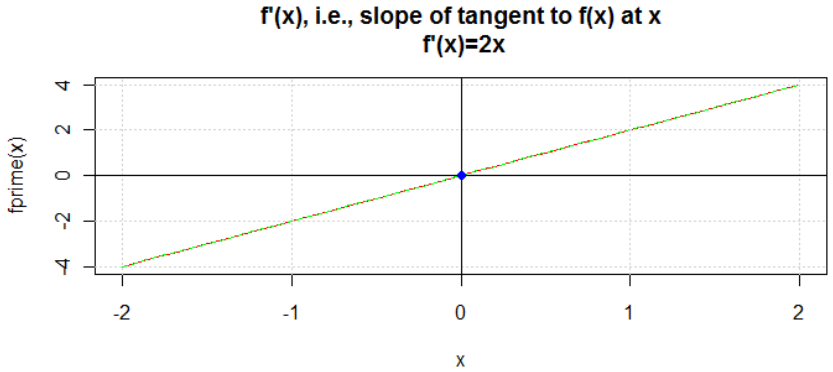
- Secant Method [wont discuss here]

Taylor Series tell us everything we need to know about finding a min or max f(x)=f(a)+f'(a)(x-a) + f''(a)(x-a)²/2..... Goal: find roots of f'(x) so we can locate Turning Points (candidate Max, Min) (Note Newton uses f'(x) and f''(x)) And use f''(x) to determine if candidate is a max or min

Find Turning Points via Zeros of Derivative

- **Turning points correspond to zeros of the derivative function**
- **Find Roots using an iterative method such as the Newton-Raphson Method**



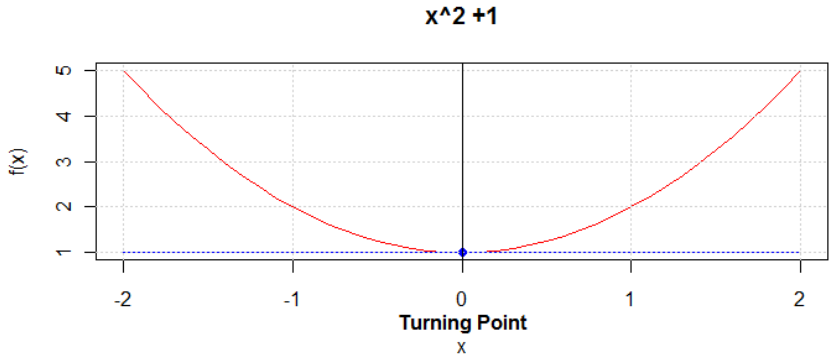


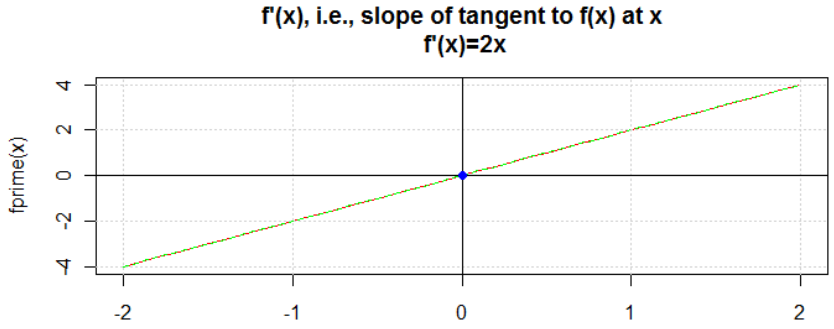
$$x^{i+1} = x^i - \frac{g(x^i)}{g'(x^i)} \quad Iteration\ function \qquad \text{where } g = f'(x)$$

$$x^{i+1} = x^i - \frac{f'(x^i)}{f''(x^i)} \quad Iteration\ function \qquad \text{for finding roots of } f(x)$$

Find roots using F(x) = x^2+1

```
#-------------------------------------------------------------
#simple convex problem
#-------------------------------------------------------------
#tangent example for f(x) = x^2 +1
fx=function(x) {   x^2+ 1 }
fprime = function (x){   2*x }
fprime.deriv <- deriv(~ x^2 + 1, "x", TRUE)

#given a point on the curve and a slope
#chose x at index 10
i=11
x0=0
f_x0 = fx(x0) #or y0
slope = fprime(x0)

tangentLine = function(x, slope, x0, y0) {
  y=slope*(x-x0) +y0
}
y=tangentLine(x, slope, x0, f_x0)
par(mfrow = c(2, 1)) # split display region into to 2 rows and one colur
x=seq(-2, 2, by=0.1)
plot(x, fx(x),  main="x^2 +1", xlab="x", yl
lines(x, y,col="blue")
points(x0, f_x0, col="blue", bg="blue", pc
text(x0-0.5, f_x0+10, paste("(", x0, ", ", f_x
grid()
abline(v=0, lty=1)
mtext("Turning Point",          # Add second y-axis label
    side=1,                 # Add to right hand side of plot
    line=2,                 # Add to line 3 from the margin
```





See example.FindTurningPoints()
Newton-Raphson not necessary here

MultiVariate Taylor

$$F(x) = F(\mathbf{x}^*) + \nabla F(\mathbf{x})^T\big|_{\mathbf{x}=\mathbf{x}^*}(\mathbf{x}-\mathbf{x}^*) + \ldots\ldots$$

where

$\nabla F(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}^*}$ is the gradient of F(x) evaluated at x *

I.E.

$$\nabla F(\mathbf{x}) = \left[\frac{\partial}{\partial x_1}F(\mathbf{x}), \frac{\partial}{\partial x_2}F(\mathbf{x}), \ldots \frac{\partial}{\partial x_n}F(\mathbf{x})\right]^T$$

$$\nabla F(\mathbf{x}) = \left[F_{x_1}(\mathbf{x}), F_{x_1}(\mathbf{x}), F_{xn}(\mathbf{x}),\right]^T$$

$$\nabla F(\mathbf{x}) = \left[F_{x_1}'(\mathbf{x}), F_{x_1}'(\mathbf{x}), F_{xn}'(\mathbf{x}),\right]^T$$

$$x^{i+1} = x^i - \frac{g(x^i)}{g'(x^i)} \quad \textit{Iteration function} \quad \text{where } g = f'(x)$$

$$x^{i+1} = x^i - \frac{f'(x^i)}{f''(x^i)} \quad \textit{Iteration function} \quad \text{for finding roots of } f'(x)$$

MultiVariate Taylor

$$F(x) = F(\mathbf{x}^*) + \nabla F(\mathbf{x})^T\big|_{\mathbf{x}=\mathbf{x}^*}(\mathbf{x}-\mathbf{x}^*) + \frac{1}{2}(\mathbf{x}-\mathbf{x}^*)^T\nabla^2 F(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}^*}(\mathbf{x}-\mathbf{x}^*)$$

where

$\nabla F(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}^*}$ is the gradient of F(x) evaluated at x *

and

$\nabla^2 F(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}^*}$ is the Hessian of F(x) evaluated at x *

Here

$$\nabla F(\mathbf{x}) = \left[\frac{\partial}{\partial x_1}F(\mathbf{x})\frac{\partial}{\partial x_2}F(\mathbf{x})\ldots\frac{\partial}{\partial x_n}F(\mathbf{x})\right]^T$$

and

$$\nabla^2 F(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2}F(\mathbf{x}) & \frac{\partial^2}{\partial x_1 \partial x_2}F(\mathbf{x}) & \ldots\frac{\partial^2}{\partial x_1 \partial x_n}F(\mathbf{x}) \\ \frac{\partial^2}{\partial x_2 \partial x_1}F(\mathbf{x}) & \frac{\partial^2}{\partial x_2^2}F(\mathbf{x}) & \ldots\frac{\partial^2}{\partial x_2 \partial x_n}F(\mathbf{x}) \\ . & . & . \\ . & . & . \\ \frac{\partial^2}{\partial x_n \partial x_1}F(\mathbf{x}) & \frac{\partial^2}{\partial x_n \partial x_2}F(\mathbf{x}) & \ldots\frac{\partial^2}{\partial x_n^2}F(\mathbf{x}) \end{bmatrix}$$

$$x^{i+1} = x^i - \frac{g(x^i)}{g'(x^i)} \quad \textit{Iteration function} \quad \text{where } g = f'(x)$$

$$x^{i+1} = x^i - \frac{f'(x^i)}{f''(x^i)} \quad \textit{Iteration function} \quad \text{for finding roots of } f'(x$$

Extrema Example

- A sufficient condition for an extreme point x* (i.e., $F(x^*) = 0$) to be a minimum is to have a positive definite Hessian at x* (i.e., has positive (nonzero) eigenvalues.
- Consider a function of two variables

$$F(x) = F(x_1, x_2) = (x_2 - x_1)^4 + 8x_1x_2 - x_1 + x_2 + 3$$

- Find all extreme (stationary) points.
- Test extreme points to determine all minima.
- Find the second-order Taylor series expansion at each minima.
- Plot $F(x)$ and the two Taylor series expansions in part 3.
- See Local Document at C:\jimi\Publications\Conferences\ESSIR-RUSSIR-2009\Ancestry\taylorSeries_PositiveDefiniteGreatExamples.docx

Gradient and Hessian

- $F(\mathbf{x}) = F(x_1, x_2) = (x_2 - x_1)^4 + 8x_1x_2 - x_1 + x_2 + 3$

$$\nabla F(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} -4(x_2 - x_1)^3 + 8x_2 - 1 \\ +4(x_2 - x_1)^3 + 8x_1 + 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\nabla^2 F(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2} F(\mathbf{x}) & \frac{\partial^2}{\partial x_1 \partial x_2} F(\mathbf{x}) \\ \frac{\partial^2}{\partial x_2 \partial x_1} F(\mathbf{x}) & \frac{\partial^2}{\partial x_2^2} F(\mathbf{x}) \end{bmatrix} =$$

$$\begin{bmatrix} 12(x_2 - x_1)^2 & -12(x_2 - x_1)^2 + 8 \\ -12(x_2 - x_1)^2 + 8 & 12(x_2 - x_1)^2 \end{bmatrix}$$

Positive Definite or not

- If we evaluate the Hessian at $x^1=[-0.42\ 0.42]^T$, we find

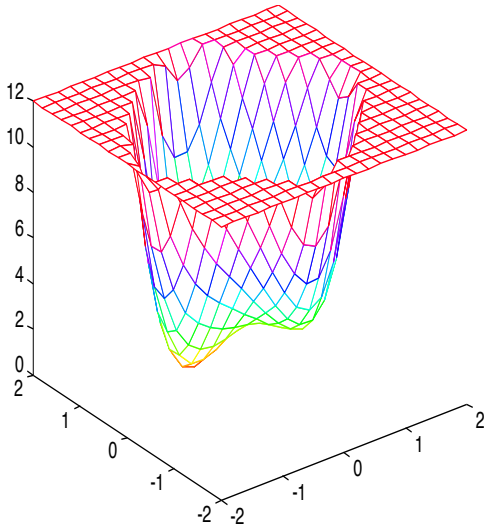
$$\nabla^2 F(\mathbf{x}^1) = \begin{bmatrix} 8.42 & -0.42 \\ -0.42 & 8.42 \end{bmatrix}$$

- The eigenvalues are the solution of the quadratic equation (characteristic equation) obtained from:

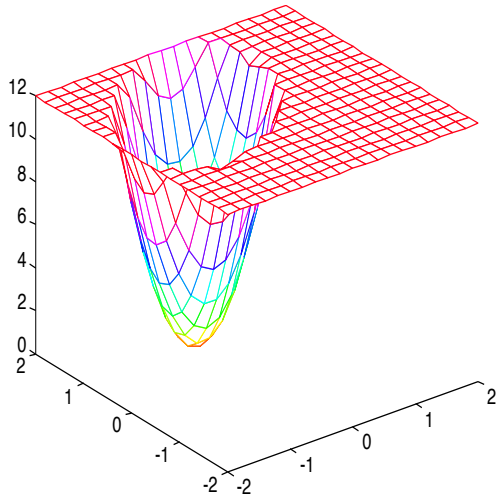
$$\begin{vmatrix} 8.42-\lambda & -0.42 \\ -0.42 & 8.42-\lambda \end{vmatrix} = (8.42-\lambda)^2 - 0.42^2 = 0$$

- Solving we get $\lambda_1 = 8.84$, $\lambda_2 = 8.0$, therefore x^1 is a strong minimum.

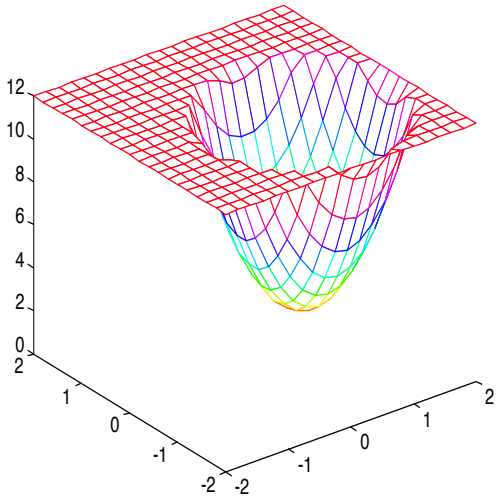
F(x) with it's two candidate minima



$F(x)$



$F_{taylor=[0.55\ -0.55]}(x)$



$F_{taylor=[-0.42\ 0.42]}(x)$

Two approximations at its the two minima $x^1=[-0.42\ 0.42]^T$ and $x^3=[0.55\ -0.55]^T$ employing second-order Taylor series expansion

Taylor Approx around $x^1=[-0.42 \;\; 0.42]^T$

- **Noting that for the minima at $x^1=[-0.42 \;\; 0.42]^T$, the gradient is zero and $F(x^1) = 2.93$ (by direct substitution). Here is the expansion:**

$$F^1(\mathbf{x}) = F(\mathbf{x}^1) + \frac{1}{2}(\mathbf{x}-\mathbf{x}^1)^T \nabla^2 F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^1}(\mathbf{x}-\mathbf{x}^1)$$

$$= 2.93 + \frac{1}{2}\begin{bmatrix} x_1 + 0.42 & x_2 - 0.42 \end{bmatrix}\begin{bmatrix} 8.42 & -0.42 \\ -0.42 & 8.42 \end{bmatrix}\begin{bmatrix} x_1 + 0.42 \\ x_2 - 0.42 \end{bmatrix}$$

$$= 4.49 - (-3.7128x_1 + 3.7128x_2) + \frac{1}{2}\begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} 8.42 & -0.42 \\ -0.42 & 8.42 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Lecture Outline

- **R Basics**
  - Most algorithms will be demonstrated with examples, code and homework problems
- **Lines, Tangents, Taylors Theorem, Roots of an equation**
- **Gradient Descent, Newton-Raphson**
- **Taylor Series: quadratic approximations**
- **Newton-Raphson quadratic convergence**
- **Multi-Dimensional Approximations (Planes)**
- **Directional Differentials, Total Differentials**
- **Vector plots, contour plots**
- **Gradient Descent**
  - Linear regression