ISM 280: Stochastic Gradient Descent

James G. Shanahan[1]

[1] *Independent Consultant*

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

ISM 280

UC Berkeley

Wednesday April 13, 2011

Lecture 2 Outline

- Taylor Series: quadratic approximations
- Newton-Raphson quadratic convergence
- Multi-Dimensional Approximations (Planes)
- Directional Differentials, Total Differentials
- Vector plots, contour plots
- Gradient Descent
  - Linear regression
- Predicting Click Through Rates
  - Linear Regression
  - *Logistic Regression*

Notes

- Gradient Descent
  - Bisection, Newton-Raphson, Secant Method avoid the calculation of derivative f'(x) (see cheney, kincaid book, page 126 for description and examples)
- Linear Regression
  - Closed form
  - Gradient descent
    - My slides
  - Maximum Likelihood
    - http://www.mayin.org/ajayshah/KB/R/index.html (see MLE)
    - http://www.mayin.org/ajayshah/KB/R/html/p3.html
  - Bayesian Model
  - Show in R

Course References

- John Fox (2010), Sage, An R and S-PLUS Companion to Applied Regression (second edition)
- Mathematical Optimization and Economic Theory, Michael Intriligator, SIAM 2002
- Nonlinear Programming, Theory and Algorithms, Mokhtar S. Bazaraa and C.M. Shetty, Wiley 1979
- Dynamic Programming and Optimal Control, Dimitri P. Bertsekas, Athena Scientific 2000
- Linear and nonlinear Programming, David Luenberg, Yinyu Ye, 3rd edition, Springer
- Duda, Hart, & Stork (2000). Pattern Classification. http://rii.ricoh.com/~stork/DHS.html
- Artificial Intelligence: A Modern Approach (Third edition) by Stuart Russell and Peter Norvig.
- *Pattern Recognition* and Machine Learning, Christopher M. Bishop, Springer

Disclaimer

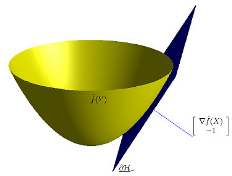
- Living vicariously!

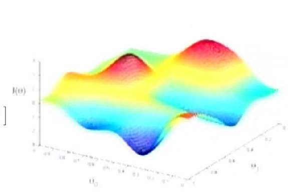
Lecture Outline

- R
- Lines, Tangents, Taylors Theorem, Roots of an equation
- Newton-Raphson quadratic convergence
- Taylor Series: quadratic approximations
- Multi-Dimensional Approximations (Planes)
- Directional Differentials, Total Differentials
- Vector plots, contour plots
- Gradient Descent
  - Linear regression
- Predicting Click Through Rates
  - Linear Regression
  - *Logistic Regression*

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 1993 –

- 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.
  - S was developed at Bell Laboratories in 1976; it was inspired by C and Unix (also developed at Bell Labs)
- 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

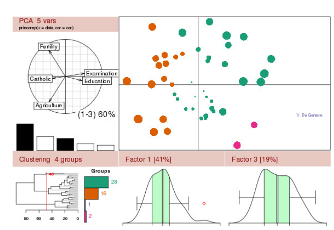
- R has its own language
  - R functionality has been made accessible from several scripting languages. E.g.,
    - Python (by the RPy[17] interface package)
    - Perl (by the Statistics::R[18] module).
- Packages:
  - Optimization packages are available
  - It can also be used as a general matrix calculation 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]);

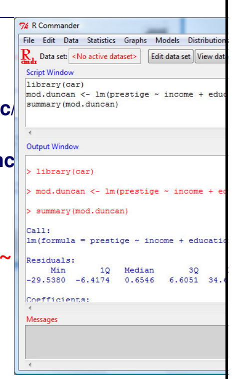
Install R Packages

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/
llation-notes.html
 # install.packages("Rcmdr", dependenc
library(Rcmdr)

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



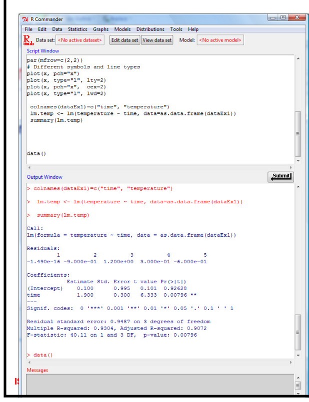
R Resources

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

Rcmdr: a tool for demos and teaching

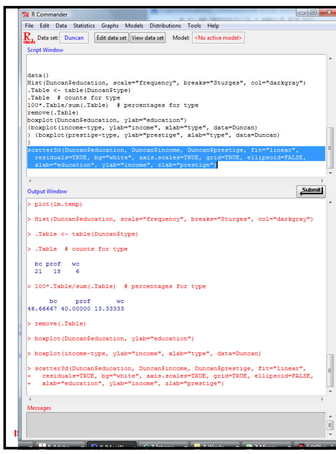
```
# Rcmdr
#
  http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/installation-notes.html
# install.packages("Rcmdr", dependencies=TRUE)
library(Rcmdr)

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

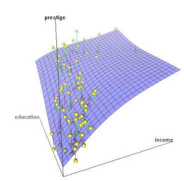

rcmdr()



Demonstrations in R

RCommander





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.

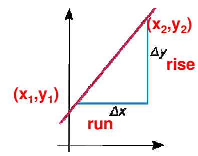
Lecture Outline

- R
- Lines, Tangents, Taylors Theorem
- Newton-Raphson quadratic convergence
- Taylor Series: quadratic approximations
- Multi-Dimensional Approximations (Planes)
- Directional Differentials, Total Differentials
- Vector plots, contour plots
- Gradient Descent
  - Linear regression
- Predicting Click Through Rates
  - Linear Regression
  - *Logistic Regression*

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**

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



y = mx +b

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 in *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 = 4x – 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:**

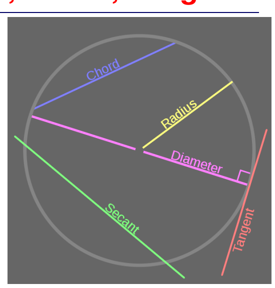
$$y - y_1 = m(x - x_1)$$

versus

$$y = mx + b$$

Secants, Chord, Tangents

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



A chord is the portion of a secant that lies within the curve.

Tangent: Best straight-line approximation to the curve at that point

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.**

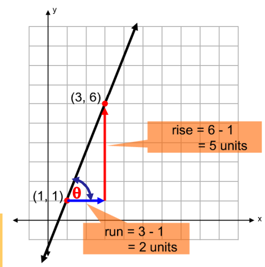
Slope of a Line

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.

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

$$m = \tan\theta.$$

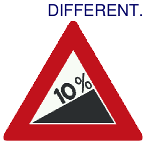
$$\theta = \arctan m$$

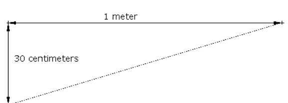


NOTE: The gradient is a generalization of the concept of slope for functions of more than one variable.

How steep is a road or railroad?

- **One is by the angle in degrees, and the other is by the slope (m) in a percentage.**
  - To calculate a *percent slope* simply you apply the following formula:
  - If I cover one meter and I rise 30 cm the percentage of slope is 30%.
  - Make attention don't confuse percentage and degrees. A 100% slope is a 45° slope... (try with the just explained method!)
  - WARNING: Gradeability for vehicles is measured in percentage, and it differs from the slope in degrees, for example, a 100% slope is a 45 degrees slope. The slope in percent and the slope in degrees are DIFFERENT.





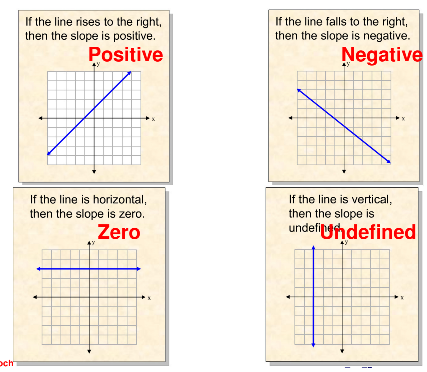
(30/100)/100 =30%

$$m = \tan\theta.$$

$$\theta = \arctan m$$

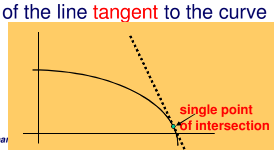
$$\text{angle} = \arctan\frac{\text{slope}}{100},$$

Slope



Slope versus Derivative?

- **In mathematics, the slope or gradient of a line describes its steepness, incline, or grade.**
  - A higher slope value indicates a steeper incline.
- **Derivative (calculus) is**
  - A function of many (independent) variables
  - The **derivative** is a measure of how a function changes as its input changes
  - The process of finding a derivative is called **differentiation**.
  - Corresponds to the slope of the line tangent to the curve (function of one variable)

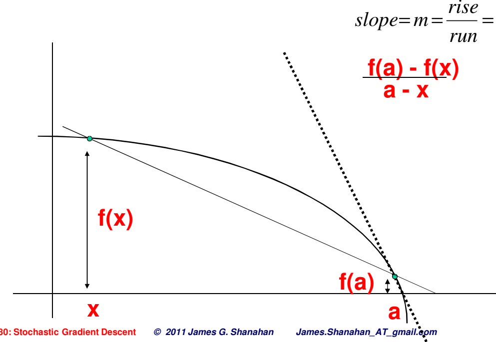


Slope of a secant line

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

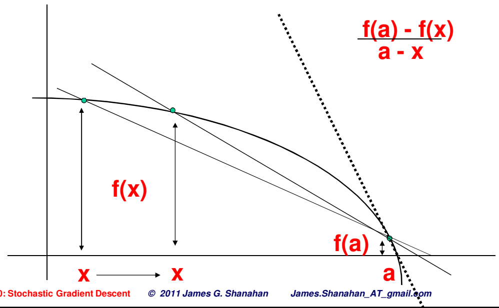
$$slope = m = \frac{rise}{run} = \frac{y_2 - y_1}{x_2 - x_1}$$

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

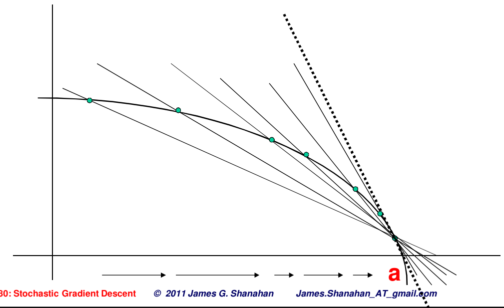


Slope of a (closer) secant line

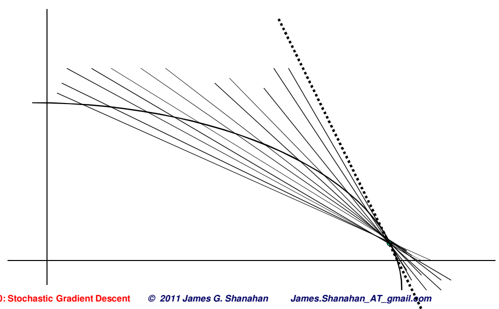
$$slope = m = \frac{rise}{run} = \frac{y_2 - y_1}{x_2 - x_1}$$



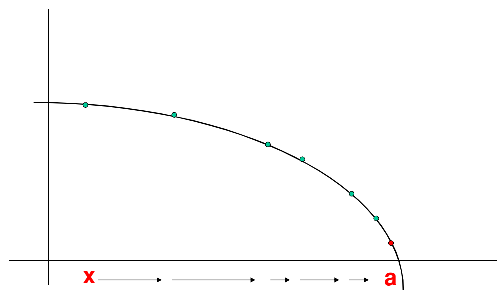
closer and closer…



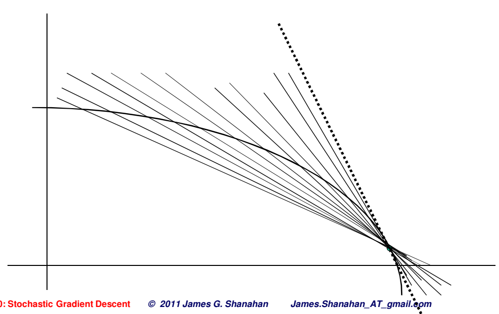
watch the slope...



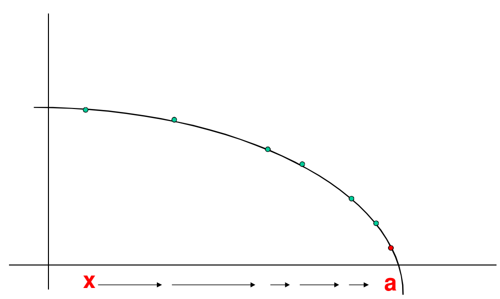
watch what *x* does...



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....

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

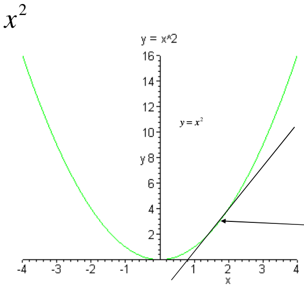
as *x* goes to *a*

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*

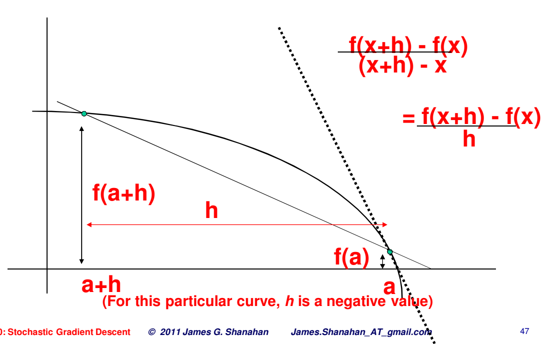
$$slope = m = \frac{rise}{run} = \frac{y_2 - y_1}{x_2 - x_1}$$

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

Now as x → a=2 we get

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

Alternatively...



(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

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

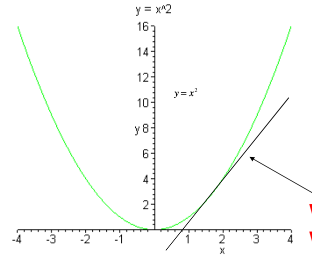
Give us a way to calculate the slope
of the line tangent at *a*!

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 …

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

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

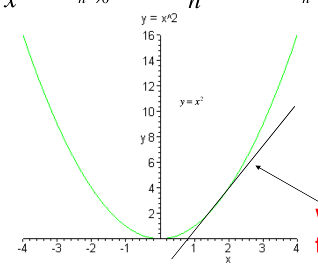
For X=2

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

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/

Calculus gives you a formula for the gradient of the tangent

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.

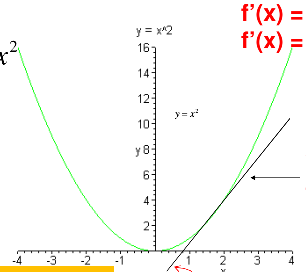
Equation of a line given a pt and slope

- 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)=
the slope is 4

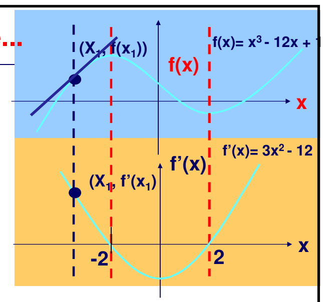
y – y0 = f'(x0)(x – x0)
y – 4 = 4(x – 2)

Given a Pt. and Slope...

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

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

[=0 at maximum and minimum]

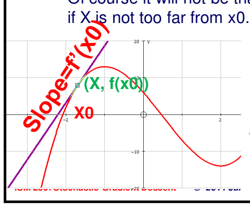


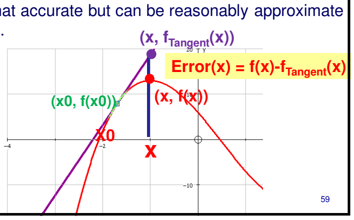
Given a Pt. and Slope... Approximate f(x) with tanger

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.



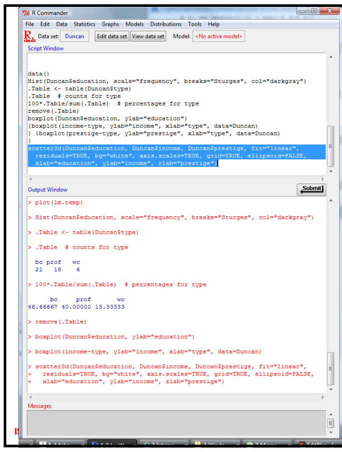


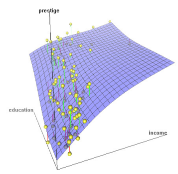
R Basics

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 Basics

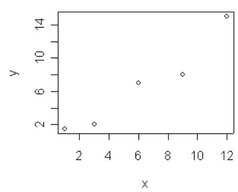
example.GettingStarted.Chapter1.Fox()

- **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

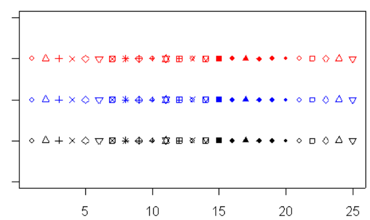
- # Example 1
- # make a very simple plot

```
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.



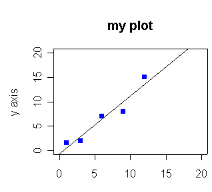
Plot a points and then …a line

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 \"

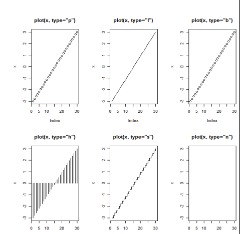
plot(x, type="l", main="plot(x, type=\"l\")")

plot(x, type="b", main="plot(x, type=\"b\")")

plot(x, type="h", main="plot(x, type=\"h\")")

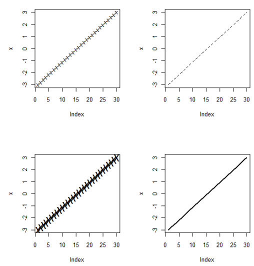
plot(x, type="s", main="plot(x, type=\"s\")")

plot(x, type="n", main="plot(x, type=\"n\")")



Different symbols and line types

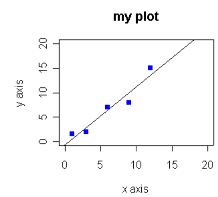
par(mfrow=c(2,2))
 # Different symbols and line types
plot(x, pch="x")
plot(x, type="l", lty=2)
plot(x, pch="x", cex=2)
plot(x, type="l", lwd=2)



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")

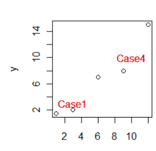
- The text() function allows us to put text on the plot where we want it. An obvious use is to label a line or group of points.

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")

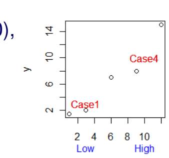
The text() function allows us to put text on the plot where we want it.



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.



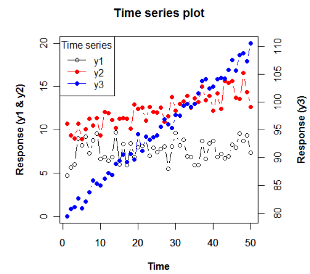
Two y-axis example

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

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

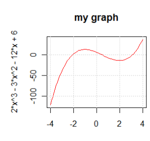
Two y-axis example

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

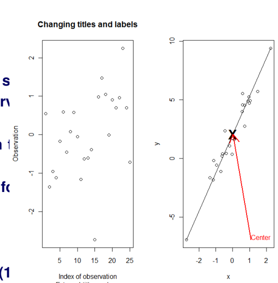
```
## R code and examples for "Modern Applied Statistics Using R"
##      Lecture 3: Graphics
## Alexander.Ploner@ki.se   2007-09-17

# 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()
```

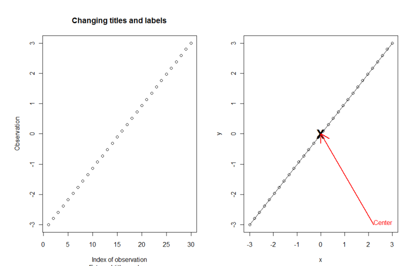
See example.PDF()
Useful for reporting

[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)
```





More Graphic Examples

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

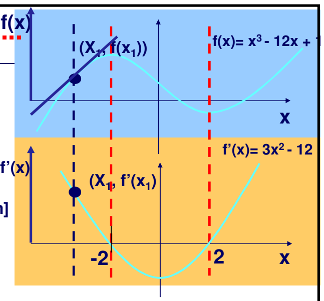
ISM 280: Stochastic Gradient Descent © 2011 James G. Shanahan James.Shanahan_AT_gmail.com 79

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]



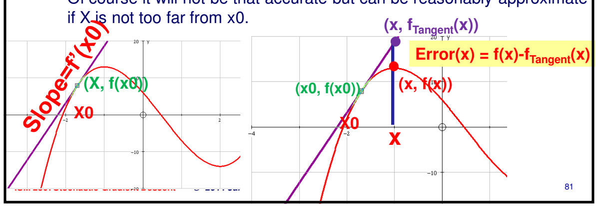
Given a Pt. and Slope... Approximate f(x) with tangent

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

ISM 280: Stochastic Gradient Descent © 2011 James G. Shanahan James.Shanahan_AT_gmail.com 80

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.



81

Guidelines for Homework

- GENERAL Guidelines for Homework
  - Please provide code, graphs and comments in a PDF report. Don't forget to put your name, email and date of submission on each report.
  - Please provide R code in separate file. Please comment your so that I or anybody else can understand it and please cross reference code with problem numbers
  - If you have questions please raise them in class or via email or during office hours
  - Homework is due on TBD.
  - Please submit your homework by email to: James.Shanahan@gmail.com with the subject "ISM 280 2011 Homework 1
  - Have fun!

ISM 280: Stochastic Gradient Descent © 2011 James G. Shanahan James.Shanahan_AT_gmail.com 82

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.

ISM 280: Stochastic Gradient Descent © 2011 James G. Shanahan James.Shanahan_AT_gmail.com 83

Derivatives in R using deriv(), D()

See example.drawTangent()

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

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

ISM 280: Stochastic Gradient Descent © 2011 James G. Shanahan James.Shanahan_AT_gmail.com 84

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=fu ) = 2*x^3 - 3*x^2 - 12*x + 6
  2*x^
}
fprim - 12 }

fprim

#give ope
#chos
x=sec
i=11
x0=x[
f_x0 =
slope
tange y0) {
  y=s
}
y=tan
par(m egio ows and one column (i.e.,
  2
x=sec
```

Exercise

f'(x) or fprime(x)

f'(x) or fprime(x)

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

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

Cntd next Slide

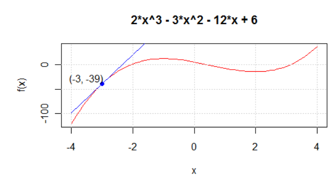
Tangent Example: plotting

See example.drawTangent()

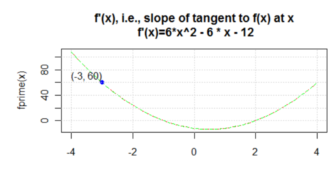
```
pa ion into to 2 rows and one column (i.e., 2 regions)
x=
plo x + 6", xlab="x", ylab="f(x)", pch="")
lin
lin
po , pch=21)
tex f_x0, ")", sep=""))
gri

plo e of tangent to f(x) at x\n f'(x)=6*x^2 - 6 * x - 12",

lin and calculate slope
lin ), lty=2, col="green")  #R-calculated slope
tex ', slope, ")", sep=""))

po ", pch=21)

gri
```

Exercise

ercise 1.1



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



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

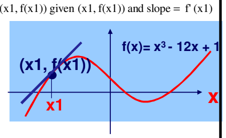
Summary on Tangent Approximations

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

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

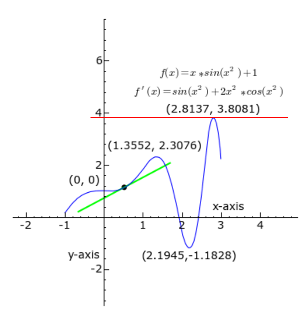
$f(x) = y1 + f'(x1)(x - x1)$

$\boxed{f(x) = f(x1) + f'(x1)(x - x1)}$ 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)

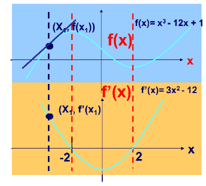
Lecture Outline

- R
- Lines, Tangents, Taylors Theorem
- Turning points, Roots, Newton-Raphson
- Taylor Series: quadratic approximations
- Multi-Dimensional Approximations (Planes)
- Directional Differentials, Total Differentials
- Vector plots, contour plots
- Gradient Descent
  - Linear regression
- Predicting Click Through Rates
  - Linear Regression
  - *Logistic Regression*

At the turning point . . .

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

0



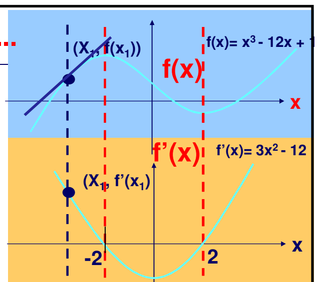
- Find the roots of the gradient function
  - Find the root or zeros of an equation analytically by hand or numerically using iterative approaches such as Newton-Raphson, gradient descent, etc.
  - What value(s) of x will f'(x) =0 (gradient be zero).

Given a Pt. and Slope...

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

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

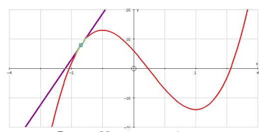
[=0 at maximum and minimum]

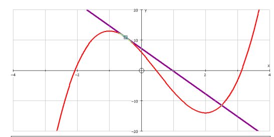


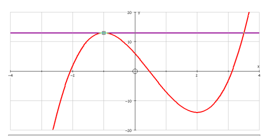
Given a Pt. and Slope... Approximate f(x) with tanger

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

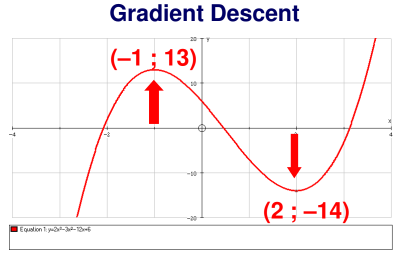
When $x = -1$, $f(-1) = 2(-1)^3 - 3(-1)^2 - 12(-1) + 6 = 13$

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

For the gradient function in our case

- 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

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

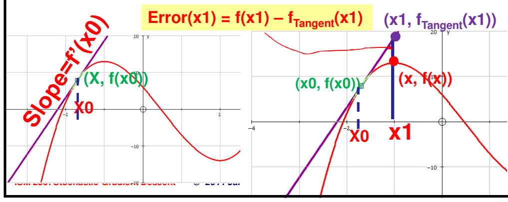
In our case f(x) is f'(x) since w… f'(x)=0

- **An important problem i…**
  statistics is findin… = 0.
  - Such values a… and also known as…
- **Can s… d above OR**
- **… numerically determine … on 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 $f(x)=f(a)+f'(a)(x-a) + f''(a)(x-a)^2/2\ldots$..
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

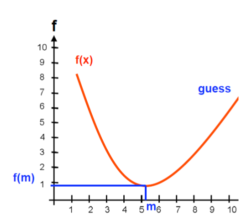
Recall: approximate a curve using a Tangent

- **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)

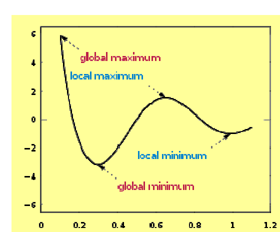


Focus on Convex Univariate problems

For the moment



Convex problem



Local and global maxima and minima for cos(3πx)/x, 0.1≤x≤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..

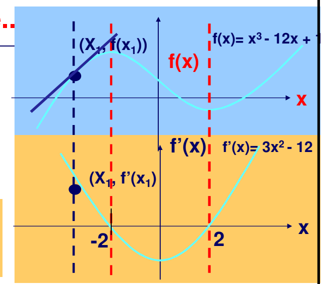
$f(x)= x^3 - 12x + 1$

First derivative

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

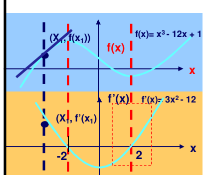
[=0 at maximum and minimum]

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

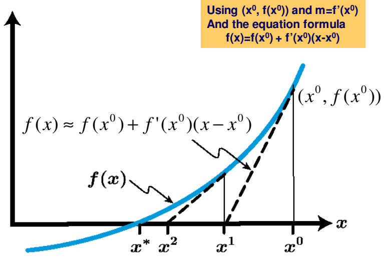


Find roots of f'(x) to give us candidate turning points

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))**

$f(x) \approx f(x^0)+f'(x^0)(x-x^0)$ $\forall$x surrounding x^0

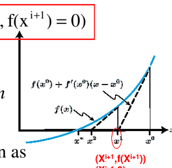
$f(x^{i+1}) \approx f(x^i)+f'(x^i)(x^{i+1}-x^i)$ $\forall$x surrounding x^i

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

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

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

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

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

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

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

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

Newton-Raphson Method – Algorithm

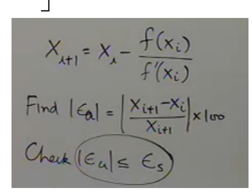
x^1, x^2, x^3…, … ,x^*

Define iteration

Do $k = 0$ to ….

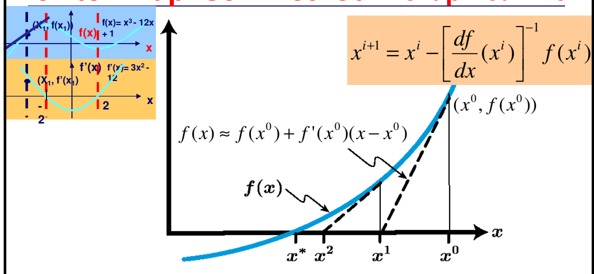
$$x^{i+1} = x^i - \left[\frac{df}{dx}(x^i)\right]^{-1} f(x^i)$$

until convergence



Newton-Raphson Method – Graphical View

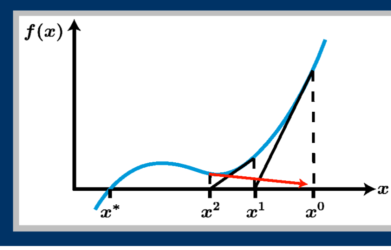
$$x^{i+1} = x^i - \left[\frac{df}{dx}(x^i)\right]^{-1} f(x^i)$$



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^*

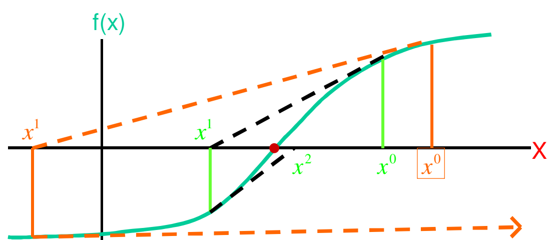


demo2

Newton-Raphson Method – Convergence

Local Convergence

Convergence Depends on a Good Initial Guess

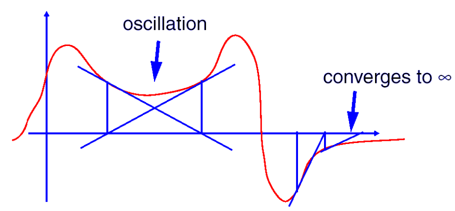


Newton-Raphson Method – Convergence

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.

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

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

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

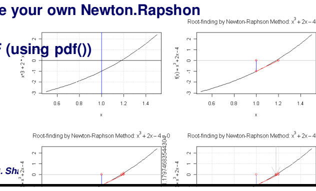
$$x_2 = 1 - \frac{-1}{5} = 1 + \frac{1}{5} = 1.2$$

Exercise 1.2

example.FindZerosOfDerivativeFunction()

- **In R write a function that:**

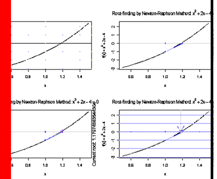
1. **Find the zeros of the function x^3+2*x-4, 1 in the interval c(0.5, 1.5) starting with an initial guess of 1.4 using the Newton-Raphson method.**
2. **Plot the progress of the algorithm (See figure below for inspiration)**
3. **Comment on the convergence**
4. **HINT: you can use a publicly available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

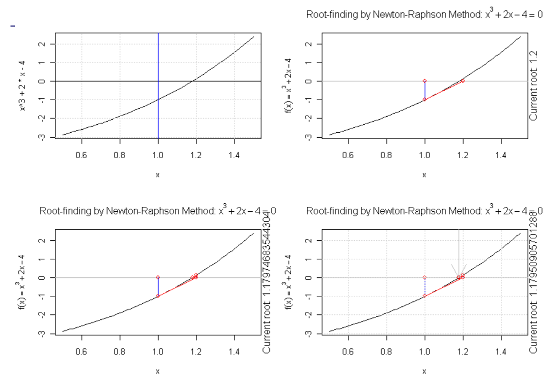


Exercise 1.2: Solution

example.FindZerosOfDerivativeFunction()

```
#find sequence of NewtonRaphson zero estimates for a function
# plots
examp

 par(m
 x = s
 plot(x
 grid()
 ablin
 ablin
 newt
 grid()
}
```





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

Let's assume 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.

F(x)=X^2 - 4

```
#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}
```

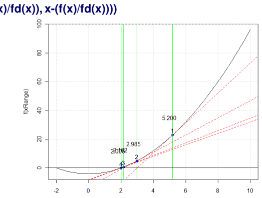
Newton-Raphson Algorithm

```
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)
```

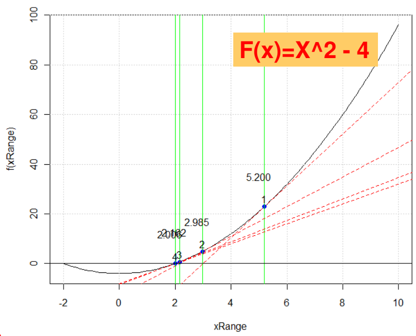

Find zero of f(x) in R with Graphics

F(x)=X^2 - 4

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

newtonRap                                   , fd){
 x=x0
 plot(xRang
 grid()
 for (i in 1:
  print(pas                                  (f(x)/fd(x)), x-(f(x)/fd(x))))
  slope = f
  intercept
  abline(int
  points(x,
  abline(v=
  text(x-.01
  text(x-.2,
  x=x-(f(x)/
 }
 list(x) # re
}
newtonRaphsonInOneDim(5.2, 4, xRange, f, fd)
```



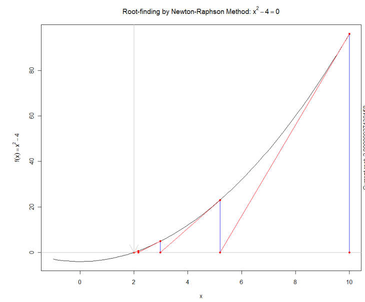
Find the root of f(x) = x^2 - 4

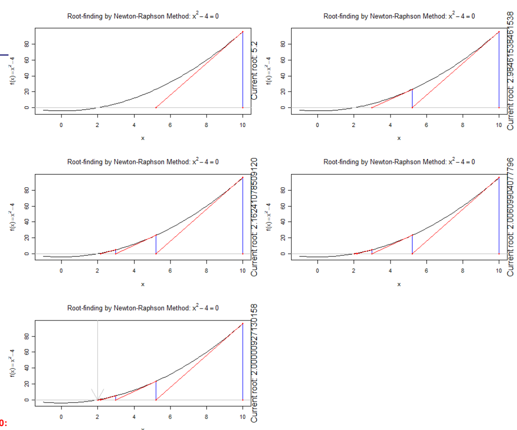


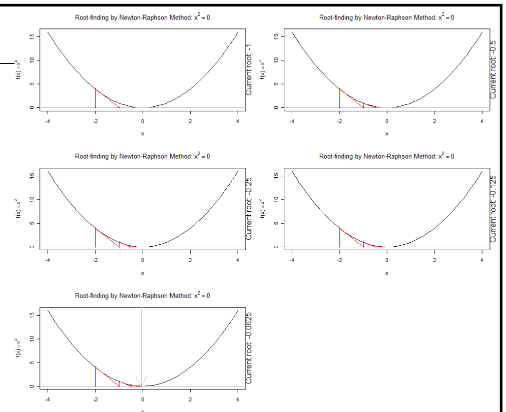
Using built-in method for NR method

par(mfrow=c(3,2))

newton.method(function(x) x^2, -2, c(-4, 4))







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.

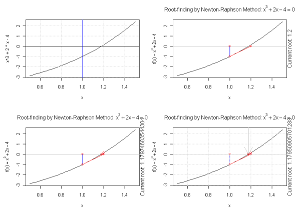
$f(x)=x^3+2x-4 \qquad f'(x)=3x^2+2 \qquad f(x) \approx f(x^0) + f'(x^0)(x - x^0)$

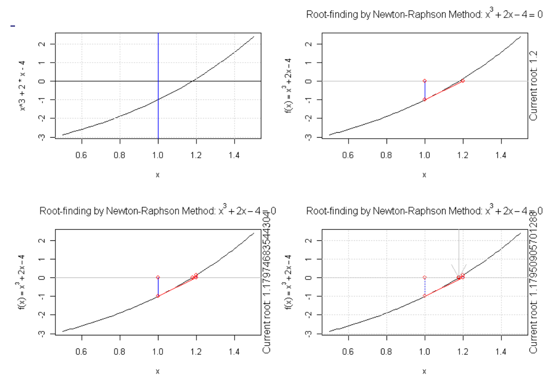
$f(1) = 1 + 2 - 4 = -1$

$f'(1) = 3 + 2 = 5 \qquad x_2 = 1 - \frac{-1}{5} = 1 + \frac{1}{5} = 1.2$

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()
}
```

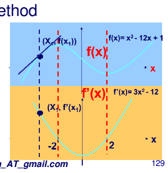




Finding the roots f(x) iteratively

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
    - Bisecting Method [wont discuss here]
    - Secant Method [wont discuss here]



Finding the roots f(x) iteratively

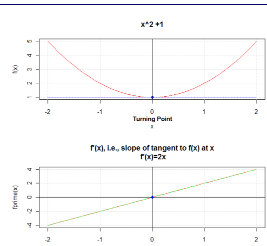
In our case f(x) is f'(x) since we … f'(x)=0

- **An important problem … statistics is finding … = 0.**
  - Such values a… and also known as…
- **Can s… above OR**
- **… numerically determine … on 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).
  - …secting 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**

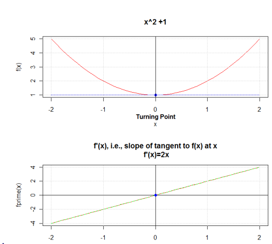


$$x^{i+1} = x^i - \frac{g(x^i)}{g'(x^i)} \quad \textit{Iteration function} \quad \text{where } g = f'(x)$$

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

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

Newton's Method in Optimization

- This iterative scheme can be generalized to several dimensions by replacing the derivative with the gradient, *f'(X),* and the reciprocal of the second derivative with the inverse of the Hessian matrix, . One obtains the Newton- iterative scheme

$$
\begin{aligned}
x^{i+1} &= -\frac{f'(x^i)}{f''(x^i)} + x^i \quad \text{For one variable} \\
&= x^i - \left[\frac{d2f}{dx}(x^i)\right]^{-1} f'(x^i) \quad \text{in matrix form} \\
X^{i+1} &= X^i - \left[f''(X^i)\right]^{-1} f'(X^i) \quad \text{For multivariable (i.e., X is a vector)}
\end{aligned}
$$

http://en.wikipedia.org/wiki/Newton's_method_in_optimization

•R Break

Solve a System of Equations in R

- Solve the system of linear equations.
  -2x + 3y = 8
  3x - y = -5
- multiply all terms in the second equation by 3
  -2x + 3y = 8
  9x - 3y = -15

  7x = -7 # add the two equations

  Note: y has been eliminated, hence the name: method of elimination
  solve the above equation for x
  x = -1
  substitute x by -1 in the first equation
  -2(-1) + 3y = 8
  solve the above equation for y
  2 + 3y = 8
  3y = 6
  y = 2
- write the solution as ordered pair (-1,2

```
>A <- matrix(c(-2,3, 3,-1 ), 2)
> A
     [,1] [,2]
[1,]   -2    3
[2,]    3   -1
> b
[1] 8 5
> b=c(8,-5)
> qr.solve(A, b) # or solve(qr(A), b)
[1] -1  2
```


Matrices

See example.Matrices()

See local file MatricesInR.doc

- To calculate inverse of a matrix
  - # division for matrices
  - ginv() # from library(MASS)
- Other useful matrix commands
  - matrix()
  - det()
  - diag()
  - t() #transpose of a matrix
  - eigen()
  - solve() #compute inverse or solve system of equations

Matrix Algebra,The R Book, M. Crawley page 259

Data in R

See example.DataFrames()

- Dataframes, matrices etc…
- Data input:
  - From the keyboard.
  - From an ascii (plain text) file.
  - From the clipboard.
  - Importing data (e.g., from SPSS).
  - From a database-management system.
  - From an R package.
- The R search path.
- Missing data.
- Numeric variables, character variables, and factors
- http://socserv.mcmaster.ca/jfox/Courses/R-course/Session-3-script.R

Matrices in R

See example.Matrices()

- Linear equations
- Determinant

- See presentation in local dir Matrices and Singular values
- MatricesLectureSingularValues.pptx

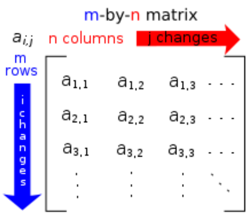
Matrices, Vectors (in R)

- **For more background see**
  - http://en.wikipedia.org/wiki/Euclidean_vector
  - http://en.wikipedia.org/wiki/Matrix_(mathematics)

In mathematics, a **matrix** (plural **matrices**, or less commonly **matrixes**) of numbers, such as

$$\begin{bmatrix} 1 & 2 & 3 \\ 6 & 5 & 4 \end{bmatrix}.$$

An item in a matrix is called an **entry** or an **element**. In the example, e.g Entries are often denoted by a variable with two subscripts, as shown on the same size can be added and subtracted entrywise and matrices of co multiplied. These operations have many of the properties of ordinary arithi matrix multiplication is not commutative, that is, **AB** and **BA** are not equa consisting of only one column or row are called vectors, while higher-dime dimensional, arrays of numbers are called tensors. Matrices with entries are also studied.



Matrices are a key tool in linear algebra. One use of matrices is to represent linear transformations, which are higher-dimensional analogs of linear functions of the form $f(x) = cx$, w corresponds to composition of linear transformations. Matrices can also keep track of the coeffic

Vectors

- In elementary mathematics, physics, and engineering, a **vector** (sometimes called a **geometric**[1] or **spatial vector**) is a geometric object that has both a magnitude (or length) and direction.
- A vector is frequently represented by a line segment with a definite direction, or graphically as an arrow, connecting an **initial point** *A* with a **terminal point** *B*, and denoted by $\overrightarrow{AB}$.

http://en.wikipedia.org/wiki/Euclidean_vector

Length of a Vector

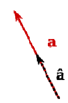
- The length or magnitude or norm of the vector **a** is denoted by ||**a**|| or, less commonly, |**a**|, which is not to be confused with the absolute value (a scalar "norm").
- The length of the vector **a** can be computed with the Euclidean norm

$$\|\mathbf{a}\| = \sqrt{\mathbf{a}\cdot\mathbf{a}}.$$

http://en.wikipedia.org/wiki/Euclidean_vector

Unit Vector, Dot Product

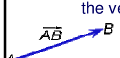
- **For more details see**
  - http://en.wikipedia.org/wiki/Euclidean_vector



Vectors in Cartesian Space: Bound Vector

- In the Cartesian coordinate system, a vector can be represented by identifying the coordinates of its initial and terminal point. For instance, the points *A* = (1,0,0) and *B* = (0,1,0) in space determine the free vector $\overrightarrow{AB}$ pointing from the point *x*=1 on the *x*-axis to the point *y*=1 on the *y*-axis.
- Typically in Cartesian coordinates, one considers primarily bound vectors. **A bound vector (aka position vector) is determined by the coordinates of the terminal point, its initial point always having the coordinates of the origin *O* = (0,0,0).**
- Thus the **bound vector** represented by (1,0,0) is a vector of unit length pointing from the origin up the positive *x*-axis.
- The coordinate representation of vectors allows the algebraic features of vectors to be expressed in a convenient numerical fashion. For example, the sum of the vectors (1,2,3) and (−2,0,4) is the vector

$$(1,\,2,\,3) + (-2,\,0,\,4) = (1-2,\,2+0,\,3+4) = (-1,\,2,\,7).$$



R Notes

- **Matrix Ops**
- **Solve(a, b)**
  - #solve a system of equations Ax=b by b=A^{-1}b; b is combination of the column in A.
  - This generic function solves the equation a %*% x = b for x, where b can be either a vector or a matrix.
  - a: a square numeric or complex matrix containing the coefficients of the linear system.
  - b: a numeric or complex vector or matrix giving the right-hand side(s) of the linear system.
  - If missing, b is taken to be an identity matrix and solve will return the **inverse of a**.

Solve a System of Equations in R

- Example 1: Solve the system of linear equations.

  -2x + 3y = 8
  3x - y = -5
- multiply all terms in the second equation by 3
  -2x + 3y = 8
  9x - 3y = -15
  add the two equations
  7x = -7

  Note: y has been eliminated, hence the name: elimination solve the above equation for x
  x = -1
  substitute x by -1 in the first equation
  -2(-1) + 3y = 8
  solve the above equation for y

  2 + 3y = 8
  3y = 6

```
>A <- matrix(c(-2,3, 3,-1 ), 2)
> A
     [,1] [,2]
[1,]   -2    3
[2,]    3   -1
> b
[1] 8 5
> b=c(8,-5)
> qr.solve(A, b) # or solve(qr
[1] -1  2
```


Lists

```
> (z <- list( a=list( b=9, c='hello'), d=1:5))
> z
$a
$a$b
[1] 9

$a$c
[1] "hello„

$d
[1] 1 2 3 4 5

> z[[1]]
$b
[1] 9

$c
[1] "hello„
> z[[1]][[1]]
[1] 9
```

```
> z[[1]][2]
$c
[1] "hello"

> z[[1]]["c"]
$c
[1] "hello"
```


Debugging in R

- http://www.stats.uwo.ca/faculty/murdoch/software/debuggingR/debug.shtml

Debugging in R

- **Use browser() #?browser** commands like c/c++ debugger
  - n #next
  - c # continue
  - Q quit
- **For more details on debugging on R RTFM (see next slide for useful example) !!**
  - http://www.stats.uwo.ca/faculty/murdoch/software/debuggingR/debug.shtml
- **Locating an error: traceback().**
  - Setting a breakpoint and examining the local environment of an executing function: browser().
  - A simple interactive debugger: debug().
  - A more sophisticated debugger: the debug package.
  - There is also a "postmortem" debugger: debugger(). (which I'll not discuss)

R debuging via browser()

- This kind of use of browser can be useful if you have a vague idea as to where a bug may be in your program.
- Notice that the first two lines in the function were not printed.

3.2.1 Explicit Calls to browser

It is possible to do a kind of "manual debugging" if you don't feel like stepping through a function line by line. The function browser can be used to suspend execution of a function so that the user can browse the local environment. Suppose we edited the SS function from above to look like:

```
SS <- function(mu, x) {
    d <- x - mu
    d2 <- d^2
    browser()
    ss <- sum(d2)
    ss
}
```

Now, when the function reaches the third statement in the program, execution will suspend and you will get a Browse[1]> prompt, much like in the debugger.

```
> SS(2, x)
Called from: SS(2, x)
Browse[1]> ls()
[1] "d"  "d2" "mu" "x"
Browse[1]> print(mu)
[1] 2
Browse[1]> mean(x)
[1] 0.02176075
Browse[1]> n
debug: ss <- sum(d2)
Browse[1]> c
[1] 503.814
```

Commands in debug mode

When the debugger is invoked, you are left in a browser(). Expressions typed at the prompt are evaluated in the local environment.

<RET>
  Go to the next statement if the function is being debugged. Continue execution if the browser was invoked.
c or *cont*
  Continue execution without single stepping.
n
  Execute the next statement in the function. This works from the browser as well.
where
  Show the call stack.
Q
  Halt execution and jump to the top-level immediately.

To view the value of a variable whose name matches one of these commands, use the print() function, e.g. print(n).

When the debugger is invoked, you are left in a browser(). Expressions typed at the prompt are evaluated in the local environment. The following commands are available.

<RET>
  Go to the next statement if the function is being debugged. Continue execution if the browser was invoked.
c or *cont*
  Continue execution without single stepping.
n
  Execute the next statement in the function. This works from the browser as well.
where
  Show the call stack.
Q
  Halt execution and jump to the top-level immediately.

To view the value of a variable whose name matches one of these commands, use the print() function, e.g. print(n).

Here is a sample session, based on the one in the R Language manual.

```
> debug(mean.default)
> mean(1:10)
debugging in: mean.default(1:10)
debug: {
    if (na.rm)
        x <- x[!is.na(x)]
    trim <- trim[1]
    n <- length(c(x, recursive = TRUE))
    if (trim > 0) {
        if (trim >= 0.5)
            return(median(x, na.rm = FALSE))
        lo <- floor(n * trim) + 1
        hi <- n + 1 - lo
        x <- sort(x, partial = unique(c(lo, hi)))[lo:hi]
        n <- hi - lo + 1
    }
    sum(x)/n
}
Browse[1]> where
where 1: mean.default(1:10)
where 2: mean(1:10)

Browse[1]>
debug: if (na.rm) x <- x[!is.na(x)]
Browse[1]>
debug: trim <- trim[1]
Browse[1]> trim
[1] 0
Browse[1]>
debug: n <- length(c(x, recursive = TRUE))
```

http://www.stats.uwo.ca/faculty/murdoch/software/debuggingR/debug.shtml

Recap: Lines, Tangents, Slopes

- Approximate *f(X) for X* around point *a* by the tangent at a point (a, f(a))

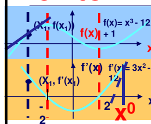
$$y - y1 = m(x - x1)$$
$$f(x) - y1 = m(x - x1) \qquad f(x) = f(a) + f'(a)(x-a)$$
$$f(x) = y1 + m(x - x1)$$
$$f(x) = f(a) + f'(a)(x-a) \quad \text{AT} \quad (a, f(a)) \quad \text{slope} = f'(a)$$

- Taylor Series explores different approximations of f(X);
  - the above tangential form is linear approximation
- General Form of a Taylor Series

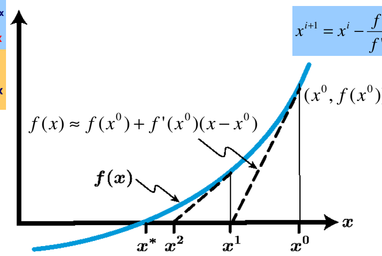
$$f(x) = \boxed{f(a) + f'(a)(x-a)} + \tfrac{f''(a)}{2!}(x-a)^2 + \ldots + \tfrac{f^{(n)}(a)}{n!}(x-a)^n$$

More compactly $f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k$

Newton-Raphson Method – Graphical View



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



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

Lecture Outline

- R
- Lines, Tangents, Taylors Theorem
- Turning points, Roots, Newton-Raphson
- Taylor Series: quadratic approximations
- Multi-Dimensional Approximations (Planes)
- Directional Differentials, Total Differentials
- Vector plots, contour plots
- Gradient Descent
  - Linear regression
- Predicting Click Through Rates
  - Linear Regression
  - *Logistic Regression*

Make Tangential Approximation Better?

- Approximate *f(X) for X* around point *a* by the tangent at a point (a, f(a))

$$y - y1 = m(x - x1)$$
$$f(x) - y1 = m(x - x1) \qquad f(x) = f(a) + f'(a)(x-a)$$
$$f(x) = y1 + m(x - x1)$$
$$f(x) = f(a) + f'(a)(x-a) \quad \text{AT} \quad (a, f(a)) \quad \text{slope} = f'(a)$$

- Taylor Series explores different approximations of f(X);
  - the above tangential form is linear approximation
- General Form of a Taylor Series

$$f(x) = \boxed{f(a) + f'(a)(x-a)} + \tfrac{f''(a)}{2!}(x-a)^2 + \ldots + \tfrac{f^{(n)}(a)}{n!}(x-a)^n$$

More compactly $f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k$

Taylor Series And Tangent Approximations

- Taylor series is a representation of a function as an infinite sum of terms calculated from the values of its derivatives at a single point.
- If the series is centered at zero, the series is also called a Maclaurin series, named after the Scottish mathematician Colin Maclaurin.
- It is common practice to use a finite number of terms of the series to approximate a function.

Taylor Series: Written Different Ways

- In mathematics, the Taylor series is a representation of a function as an infinite sum of terms calculated from the values of its derivatives at a single point.

$$f(x) = f(a) + f'(a)(x-a) + \tfrac{f''(a)}{2!}(x-a)^2 + \ldots + \tfrac{f^{(n)}(a)}{n!}(x-a)^n$$

$$F(x) = F(x^*) + \frac{d}{dx}F(x)\Big|_{x=x^*}(x-x^*) + \frac{1}{2}\frac{d^2}{dx^2}F(x)\Big|_{x=x^*}(x-x^*)^2 + \frac{1}{n!}\frac{d^n}{dx^n}F(x)\Big|_{x=x^*}(x-x^*)^n + \ldots..$$

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k$$

$F(x)$ = cos(x): Taylor series expansion,$^{x^*=0}$

- **Given F(X)=cos(x) and a Taylor Series expansion at x* = 0**

$$F(x) = \cos(x)$$
$$= \cos(0) - \sin(0)(x-0) - \frac{1}{2}\cos(0)(x-0)^2 + \frac{1}{6}\sin(0)(x-0)^3 + \ldots$$
$$= 1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 + \ldots.$$

- **The zeroth-order approximation of $F(x)$ is**

$$F(x) \approx F_0(x) = 1 = F_1(x)$$

  - (Note that in this case the first-order approximation is the same as the zeroth-order approximation, since the first derivative is zero, i.e., sin(0)=0).

- **The second-order approximation is**

$$F(x) \approx F_2(x) = 1 - \frac{1}{2}x^2 = F_3(x)$$

- **The fourth-order approximation is**

$$F(x) \approx F_4(x) = 1 - \frac{1}{2}x^2 + \frac{1}{24}x^4$$

Taylor Series Approxn. Of Sin(x) at 0

- **Approximating f(x) = sin x when it is centred around 0 (I.e., a=0), Taylor Polynomial of degree 7 (sin(0)=; cos(0) =1)**

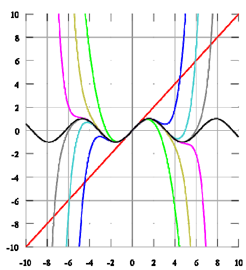
$$f(x) \;=\; f(a) + f'(a)(x-a) + \tfrac{f''(a)}{2!}(x-a)^2 + \ldots + \tfrac{f^{(n)}(a)}{n!}(x-a)^n$$
$$\sin_{T7}(x) \approx \sin(0) + \cos(0)(x-0) - \frac{\sin(0)}{2!}(x-0)^0 - \frac{\cos(0)}{3!}(x-0)^3 + \ldots\ldots$$
$$\sin x \approx M_7(x) = 0 + x + 0\cdot\frac{x^2}{2!} - \frac{x^3}{3!} +$$
$$0\cdot\frac{x^4}{4!} + \frac{x^5}{5!} + 0\cdot\frac{x^6}{6!} - \frac{x^7}{7!}$$

$$\frac{d(\sin x)}{dx} = \cos x$$
$$\frac{d(\cos x)}{dx} = -\sin x$$
$$\frac{d(\tan x)}{dx} = \sec^2 x$$

Taylor Series Approximations of f(x) at a

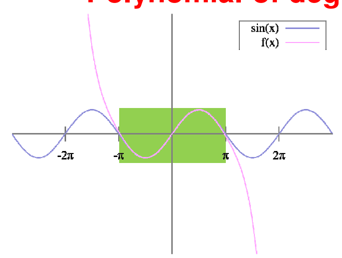
$$f(x) = f(a) + f'(a)(x-a) + \tfrac{f''(a)}{2!}(x-a)^2 + \ldots + \tfrac{f^{(n)}(a)}{n!}(x-a)^n$$

- As the degree of the Taylor polynomial rises, it approaches the correct function.
- This image shows **sin*x*** (in black) and Taylor approximations about a=0, polynomials of degree 1, 3, 5, 7, 9, 11 and 13.
- **What does the Taylor Approximation of degree zero look like at ?**



f(x) ~f(a)= sin(0)) = 0; f(x)=0 ∀ x in the neighborhood of 0

Sine Function Approximated by Taylor Polynomial of degree 7



The sine function (blue) is closely approximated by its Taylor polynomial of degree 7 (pink) for a full period centered at the origin.

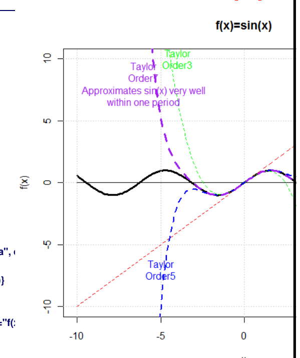
Problem: Plot Taylor Approxns of sin(x)

```
#----------------------------------------------------
# Taylor series in one-dim for f(x) = sin(x) at a=0
#----------------------------------------------------
# plot sin(x) and its Taylor series approximation in
# the range [-10, 10]

## Higher derivatives (boiler plate):
DD <- function(expr,name, order = 1) {
  if(order < 1) stop("'order' must be >= 1")
  if(order == 1) D(expr,name)
  else DD(D(expr, name), name, order - 1)
}
#e.g., DD(expression(sin(x^2)), "x", 3)


f = function(x){sin(x)}
fPrime.order =function(a, order){eval(DD(expression(sin(a)), "a", order))}

taylorTerm_n=function(a, n){fPrime.order(a, n)*x^n/factorial(n)}
```

See example.TaylorSeries()

Plot Taylor Approximations of Sin(x)

```
#----------------------------------------------------
# Taylor series in one-dim for f(x) = sin(x) at a=0
#----------------------------------------------------
# plot sin(x) and its Taylor series approximation in
# the range [-10, 10]

## Higher derivatives (boiler plate):
DD <- function(expr,name, order = 1) {
  if(order < 1) stop("'order' must be >= 1")
  if(order == 1) D(expr,name)
  else DD(D(expr, name), name, order - 1)
}
#e.g., DD(expression(sin(x^2)), "x", 3)


f = function(x){sin(x)}
fPrime.order =function(a, order){eval(DD(expression(sin(a)), "a", 

taylorTerm_n=function(a, n){fPrime.order(a, n)*x^n/factorial(n)}

x=seq(-10, 10, by=0.1)
plot(x, f(x),  ylim=c(-10, 10), main="f(x)=sin(x)", xlab="x", ylab="f(
grid()
lines(x, rep(0, length(x)))  #term_0(x) reduces to f(0)=sin(0)=0
lines(x, rep(0, length(x)) + taylorTerm_n(0,1), col="red", lty=2)
#same order 1 Taylor Approx as taylorTerm_n(0,2) is 0
```



Taylor Approximations of Different Degrees

- **Linear Approximation of the function f at a**

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

- **Quadratic Approximation**

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

- **General Form of a Taylor Series**

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$

More compactly $f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k$

Multivariate Taylor Series

A second-order Taylor series expansion of a scalar-valued function of more than one variable can be written compactly as

$$T(\mathbf{x}) = f(\mathbf{a}) + (\mathbf{x}-\mathbf{a})^T \mathrm{D} f(\mathbf{a}) + \frac{1}{2!}(\mathbf{x}-\mathbf{a})^T \{\mathrm{D}^2 f(\mathbf{a})\}(\mathbf{x}-\mathbf{a}) + \cdots,$$

where *Df(a)* is the gradient (partial derivatives) of *f* evaluated at *x=a* (*Df() is* sometimes written as ∇f)

$$\text{Df(a)=}\quad \nabla f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right).$$

and *D²f(a)* is the Hessian matrix. sometimes represented as *H(f)* as follows:.

$$\text{D}^2\text{f(a)=}\ H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \frac{\partial^2 f}{\partial x_2\,\partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\,\partial x_1} & \frac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Gradient and Hessian

- ***Find***
- ***F*(x) = *F*(x_1, x_2) = (x_2 - x_1)⁴ +8$x_1$$x_2$ - x_1 + x_2 + 3**

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

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

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

Taylor Approx around x¹=[-0.42 0.42]ᵀ

- **Noting that for the minima at x¹=[-0.42 0.42]ᵀ, the gradient is zero (so drop linear term) and *F*(x¹) = 2.93 (by direct substitution). Here is the expansion:**

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

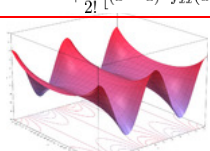
$$= 2.93 + \frac{1}{2}\begin{bmatrix} x_1 + 0.42 & x_2 - 0.42 \end{bmatrix}\begin{bmatrix} 8.42 & -0.42 \\ -0.42 & 8.42 \end{bmatrix}\begin{bmatrix} x_1 + 0.42 \\ x_2 - 0.42 \end{bmatrix}$$

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

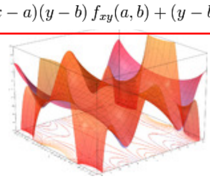
Multivariate Taylor Series

$$\text{f(X)} \quad = \sum_{n_1=0}^{\infty} \cdots \sum_{n_d=0}^{\infty} \frac{(x_1-a_1)^{n_1}\cdots(x_d-a_d)^{n_d}}{n_1!\cdots n_d!}\left(\frac{\partial^{n_1+\cdots+n_d} f}{\partial x_1^{n_1}\cdots\partial x_d^{n_d}}\right)(a_1,\ldots,a_d).$$

$$f(x,y) \approx f(a,b) + (x-a)\,f_x(a,b) + (y-b)\,f_y(a,b) + \frac{1}{2!}\left[(x-a)^2\,f_{xx}(a,b) + 2(x-a)(y-b)\,f_{xy}(a,b) + (y-b)^2\,f_{yy}(a,b)\right]$$



The real part of the cosine function in the complex plane.



Overlaid with an 8th degree approximation at (0, 0) of the cosine function in the complex plane.

Taylors Series Example: MultiDim

Compute a second-order Taylor series expansion around point $(a,b) = (0,0)$ of a function

$$f(x,y) = e^x \log(1+y).$$

Firstly, we compute all partial derivatives we need

$$f_x(a,b) = e^x \log(1+y)\Big|_{(x,y)=(0,0)} = 0,$$

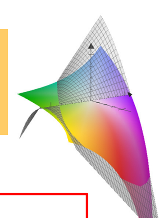
$$f_y(a,b) = \frac{e^x}{1+y}\Big|_{(x,y)=(0,0)} = 1,$$

$$f_{xx}(a,b) = e^x \log(1+y)\Big|_{(x,y)=(0,0)} = 0,$$

$$f_{yy}(a,b) = -\frac{e^x}{(1+y)^2}\Big|_{(x,y)=(0,0)} = -1,$$

$$f_{xy}(a,b) = f_{yx}(a,b) = \frac{e^x}{1+y}\Big|_{(x,y)=(0,0)} = 1.$$

Second-order Taylor series approximation (in gray) of a function *f*(*x,y*) = *e*ˣlog(1 + *y*) around origin.



The Taylor series is

$$T(x,y) = f(a,b) + (x-a)\,f_x(a,b) + (y-b)\,f_y(a,b) + \frac{1}{2!}\left[(x-a)^2\,f_{xx}(a,b) + 2(x-a)(y-b)\,f_{xy}(a,b) + (y-b)^2\,f_{yy}(a,b)\right] + \cdots,$$

which in this case becomes

$$T(x,y) = 0 + 0(x-0) + 1(y-0) + \frac{1}{2}\left[0(x-0)^2 + 2(x-0)(y-0) + (-1)(y-0)^2\right] + \cdots$$

$$= y + xy - \frac{y^2}{2} + \cdots.$$

Since log(1 + *y*) is analytic in |*y*| < 1, we have

$$e^x \log(1+y) = y + xy - \frac{y^2}{2} + \cdots$$

for |*y*| < 1.

Lagrange Remainder

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f^{(2)}(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + R_n(x).$$

- Here, $n!$ denotes the factorial of n, and $R_n(x)$ is a remainder term, denoting the difference between the Taylor polynomial of degree n and the original function.

- **The remainder term $R_n(x)$ depends on x and is small if x is close enough to a. Several expressions are available for it.**
- **The Lagrange form[1] of the remainder term states that there exists a number ξ between a and x such that**

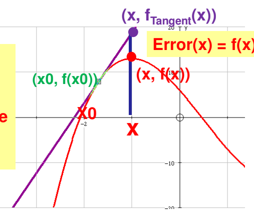
$$R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-a)^{n+1}.$$

http://en.wikipedia.org/wiki/Lagrange_remainder

Lagrange Remainder

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f^{(2)}(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + R_n(x).$$

The quality of the approximation is controlled by the remainder term as we move away from *a=x0*



$$R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-a)^{n+1}.$$ **Lagrange Remainder**

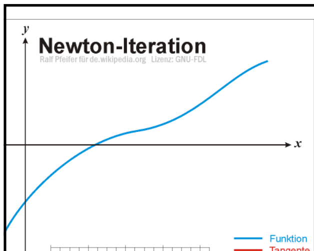
Why/where are they used?

- **Taylor polynomials (finite versions of Taylor series) approximate functions near the center.**
- **The more terms you take, the better your estimate of *f(x).***
- **Used extensively in finding the roots of an equation or system of equations (e.g., f'(x))and therefore maxima or minima (of f(x)),**
  - in operations research,
  - machine learning
- **Tells us about convexity and concavity of a function**
  - If concave or convex then global max or min exists and numerical approaches can be used to iteratively find the global min/max
  - Otherwise need to resort to heuristic approaches to find min/max (generally, these will be local min or max)

Lecture Outline

- **R**
- **Lines, Tangents, Taylors Theorem**
- **Turning points, Roots, 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
- **Predicting Click Through Rates**
  - Linear Regression
  - *Logistic Regression*

Newton-Rhapson



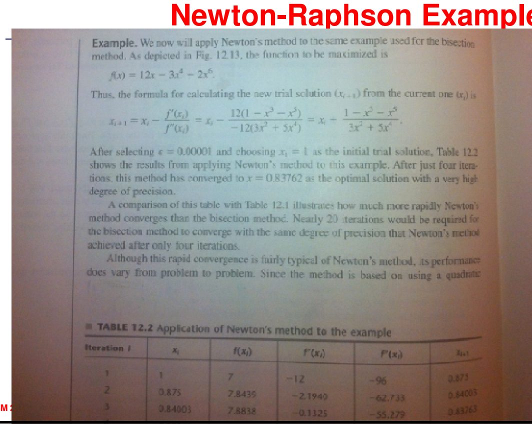
$$f(x) = f(a) + f'(a)(x-a) + \tfrac{f''(a)}{2!}(x-a)^2 + \ldots + \tfrac{f^{(n)}(a)}{n!}(x-a)^n$$

$$f(x) \approx f(a) + f'(a)(x-a) + \tfrac{f''(a)}{2!}(x-a)^2$$

$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1}-x_i) + \tfrac{f''(x)}{2!}(x_{i+1}-x_i)^2$$

where $f(x_i)$, $f'(x_i)$ and $f''(x_i)$ are constants

Newton-Raphson Example



Example. We now will apply Newton's method to the same example used for the bisection method. As depicted in Fig. 12.13, the function to be maximized is

$$f(x) = 12x - 3x^4 - 2x^6.$$

Thus, the formula for calculating the new trial solution (x_{i+1}) from the current one (x_i) is

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)} = x_i - \frac{12(1-x^3-x^5)}{-12(3x^2+5x^4)} = x_i + \frac{1-x^3-x^5}{3x^2+5x^4}.$$

After selecting $\epsilon = 0.00001$ and choosing $x_1 = 1$ as the initial trial solution, Table 12.2 shows the results from applying Newton's method to this example. After just four iterations, this method has converged to $x = 0.83762$ as the optimal solution with a very high degree of precision.

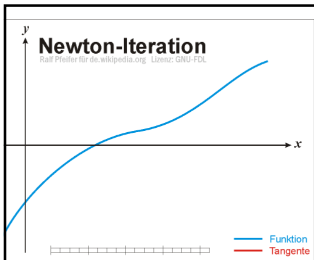
A comparison of this table with Table 12.1 illustrates how much more rapidly Newton's method converges than the bisection method. Nearly 20 iterations would be required for the bisection method to converge with the same degree of precision that Newton's method achieved after only four iterations.

Although this rapid convergence is fairly typical of Newton's method, its performance does vary from problem to problem. Since the method is based on using a quadratic

TABLE 12.2 Application of Newton's method to the example

Iteration i	x_i	$f(x_i)$	$f'(x_i)$	$f''(x_i)$	x_{i+1}
1	1	7	−12	−96	0.875
2	0.875	7.8439	−2.1940	−62.733	0.84003
3	0.84003	7.8838	−0.1325	−55.279	0.83763

Newton-Rhapson



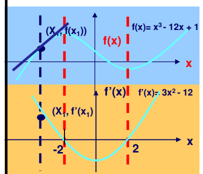
$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2$$

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

where $f(x_i)$, $f'(x_i)$ and $f''(x_i)$ are constants

Newton Rhapson (sometimes known as Newton



$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2$$

$$f(x_{i+1}) \approx f(x_i) + f'(x_i)(x_{i+1} - x_i) + \frac{f''(x)}{2!}(x_{i+1} - x_i)^2$$

where $f(x_i)$, $f'(x_i)$ and $f''(x_i)$ are constants

Approximating f(x) using this quadratic approximation, one can maximize it by taking the first derivative and setting it to zero:

$$f'(x_{i+1}) \approx \frac{\partial f(x_{i+1})}{\partial x_{i+1}} = 0 + f'(x_i)(1-0) + 2\frac{f''(x)}{2!}(x_{i+1} - x_i)^2(1-0)$$

$$0 = f'(x_i) + f''(x_i)(x_{i+1} - x_i)$$

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$

Quadratic Convergence 1/2

Quadratic convergence for Newton's iterative method

According to Taylor's theorem, any function $f(x)$ which has a continuous second derivative can be represented by an expansion about a point that is close to a root of f(x). Suppose this root is α. Then the expansion of f(α) about x_n is:

$$f(\alpha) = f(x_n) + f'(x_n)(\alpha - x_n) + R_1$$

where the Lagrange form of the Taylor series expansion remainder is

$$R_1 = \frac{1}{2!}f''(\xi_n)(\alpha - x_n)^2,$$

where ξ_n is in between x_n and α.

Since α is the root, (1) becomes:

$$0 = f(\alpha) = f(x_n) + f'(x_n)(\alpha - x_n) + \frac{1}{2}f''(\xi_n)(\alpha - x_n)^2$$

Dividing equation (2) by $f'(x_n)$ and rearranging gives

$$\frac{f(x_n)}{f'(x_n)} + (\alpha - x_n) = \frac{-f''(\xi_n)}{2f'(x_n)}(\alpha - x_n)^2 \quad (3)$$

Remembering that x_{n+1} is defined by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

one finds that

$$\underbrace{\alpha - x_{n+1}}_{\epsilon_{n+1}} = \frac{-f''(\xi_n)}{2f'(x_n)}(\underbrace{\alpha - x_n}_{\epsilon_n})^2.$$

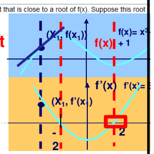
That is,

$$\epsilon_{n+1} = \frac{-f''(\xi_n)}{2f'(x_n)}{\epsilon_n}^2. \quad (5)$$

Let α (target) be the root

R_1 is the Remainder

Error, $\varepsilon_{n+1} = \alpha - x_{n+1}$



For $\varepsilon_{n+1} < 1$then quadratic convergence

$$|\epsilon_{n+1}| = \frac{|f''(\xi_n)|}{2|f'(x_n)|}{\epsilon_n}^2$$

Assume $\in_{n+1} = 1 \cdot \in_n^2$ and $\in_i < 1$

$E.g.$, assume $\in_n = 0.9$

then $\in_{n+1} = 0.81$ and $\in_{n+2} = 0.64$ etc

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

Quadratic Convergence 2/2

Quadratic convergence for Newton's iterative method

That is,

$$\epsilon_{n+1} = \frac{-f''(\xi_n)}{2f'(x_n)}{\epsilon_n}^2. \quad (5)$$

Taking absolute value of both sides gives

$$|\epsilon_{n+1}| = \frac{|f''(\xi_n)|}{2|f'(x_n)|}{\epsilon_n}^2 \quad (6)$$

Quadratic convergence holds if the conditions met

Equation (6) shows that the rate of convergence is quadratic if following conditions are satisfied:

1. $f'(x) \neq 0; \forall x \in I$, where I is the interval $[\alpha - r, \alpha + r]$ for some $r \geq |(\alpha - x_0)|$;
2. $f''(x)$ is finite, $\forall x \in I$;
3. x_0 *sufficiently* close to the root α

The term *sufficiently* close in this context means the following:

(a) Taylor approximation is accurate enough such that we can ignore higher order terms,

(b) $\frac{1}{2}\left|\frac{f''(x_n)}{f'(x_n)}\right| < C\left|\frac{f''(\alpha)}{f'(\alpha)}\right|$, for some $C < \infty$,

(c) $C\left|\frac{f''(\alpha)}{f'(\alpha)}\right|\epsilon_n < 1$, for $n \in \mathbb{Z}^+ \cup \{0\}$ and C satisfying condition (b).

Finally, (7) can be expressed in the following way:

$$|\epsilon_{n+1}| \leq M\epsilon^2_n$$

where M is the supremum of the variable coefficient of ϵ^2_n on the interval I defined in the condition 1, that is:

$$M = \sup_{x \in I} \frac{1}{2}\left|\frac{f''(x)}{f'(x)}\right|.$$

The initial point x_0 has to be chosen such that conditions 1 through 3 are satisfied, where the third condition requires that $M|\epsilon_0| < 1$.

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

Newton-Raphson Method – Algorithm

Define iteration

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

Do $k = 0$ to ….

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

Until convergence

(e.g., $|x^{i+1} - x^i| < \varepsilon$ (i.e., $|x^{i+1} - x^i|$ has become sufficiently small))

Exercise (not required)

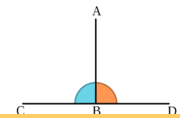
- **One algorithmic criterion for the convergence of the Newton-Rhapson root finding algorithm is $|x_{i+1} - x_i| < \varepsilon$ (i.e., has become sufficiently small).**
  - Can you describe at least one other criterion for convergence besides the one described here?
  - Can you describe a third criterion for extra points?

Lecture Outline

- R
- Lines, Tangents, Taylors Theorem
- Turning points, Roots, 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
- Predicting Click Through Rates
  - Linear Regression
  - Logistic Regression

Orthogonality

- In mathematics, two vectors A and B are **orthogonal** if they are perpendicular, i.e., they form a right angle and A*B=0.
- The vectors (1, 3, 2), (3, –1, 0), (1/3, 1, –5/3) are orthogonal to each other
  - since (1)(3) + (3)(–1) + (2)(0) = 0, (3)(1/3) + (–1)(1) + (0)(–5/3) = 0, (1)(1/3) + (3)(1) – (2)(5/3) = 0.
  - Observe also that the dot product of the vectors with themselves are the norms of those vectors, so to check for orthogonality, we need only check the dot product with every other vector.



AB and CD are orthogonal to each other.

Continuous Function

3 Definition A function f of two variables is called **continuous at** (a, b) if

$$\lim_{(x,y)\to(a,b)} f(x, y) = f(a, b)$$

We say f is **continuous on** D if f is continuous at every point (a, b) in D.

The intuitive meaning of continuity is that if the point (x, y) changes by a small amount, then the value of $f(x, y)$ changes by a small amount. This means that a surface that is the graph of a continuous function has no hole or break.

Small change in (x,y) implies small change in f(x,y)

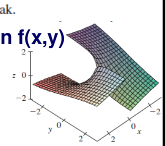
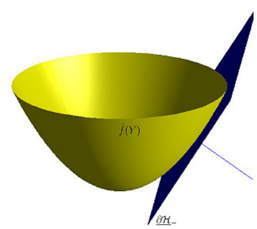
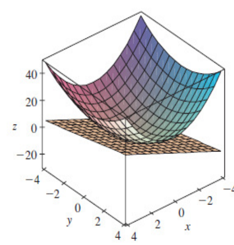


FIGURE 7
The function $h(x, y) = \arctan(y/x)$ is discontinuous where $x = 0$.

Tangent Approximations





A plane from a point and orthogonal vector

- Although a line in space is determined by a point and a direction, a plane in space is more difficult to describe.
- A single vector parallel to a plane is not enough to convey the "direction" of the plane, but a vector perpendicular to the plane does completely specify its direction.
- Thus, a plane in space is determined by a point in the plane and a vector that is orthogonal to the plane. This orthogonal vector is called a normal vector.

7 $$ax + by + cz + d = 0$$

where $d = -(ax_0 + by_0 + cz_0)$. Equation 7 is called a **linear equation** in x, y, and z.

SECTION 9.5 EQUATIONS OF LINES AND PLANES ◆ 679

But if we solve the first two equations, we get $t = \frac{11}{5}$ and $s = \frac{8}{5}$, and these values don't satisfy the third equation. Therefore, there are no values of t and s that satisfy the three equations. Thus, L_1 and L_2 do not intersect. Hence, L_1 and L_2 are skew lines.

Planes

Although a line in space is determined by a point and a direction, a plane in space is more difficult to describe. A single vector parallel to a plane is not enough to convey the "direction" of the plane, but a vector perpendicular to the plane does completely specify its direction. Thus, a plane in space is determined by a point $P_0(x_0, y_0, z_0)$ in the plane and a vector $\mathbf{n}$ that is orthogonal to the plane. This orthogonal vector $\mathbf{n}$ is called a **normal vector**. Let $P(x, y, z)$ be an arbitrary point in the plane, and let $\mathbf{r}_0$ and $\mathbf{r}$ be the position vectors of P_0 and P. Then the vector $\mathbf{r} - \mathbf{r}_0$ is represented by $\overrightarrow{P_0P}$. (See Figure 6.) The normal vector $\mathbf{n}$ is orthogonal to every vector in the given plane. In particular, $\mathbf{n}$ is orthogonal to $\mathbf{r} - \mathbf{r}_0$ and so we have

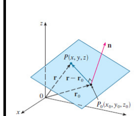


FIGURE 6

4 $$\mathbf{n} \cdot (\mathbf{r} - \mathbf{r}_0) = 0$$

Derivation of vector equation of a plane from:
1. **a normal vector (derivative of the function)**
2. **and point on the plane**

which can be rewritten as

5 $$\mathbf{n} \cdot \mathbf{r} = \mathbf{n} \cdot \mathbf{r}_0$$

Either Equation 4 or Equation 5 is called a **vector equation of the plane**.

To obtain a scalar equation for the plane, we write $\mathbf{n} = \langle a, b, c\rangle$, $\mathbf{r} = \langle x, y, z\rangle$, and $\mathbf{r}_0 = \langle x_0, y_0, z_0\rangle$. Then the vector equation (4) becomes

$$\langle a, b, c\rangle \cdot \langle x - x_0, y - y_0, z - z_0\rangle = 0$$

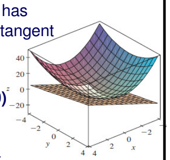
or

6 $$a(x - x_0) + b(y - y_0) + c(z - z_0) = 0$$

Equation 6 is the **scalar equation of the plane through** $P_0(x_0, y_0, z_0)$ **with normal vector** $\mathbf{n} = \langle a, b, c\rangle$.

Tangent Planes and Linear Approximation

- Just as we can visualize the line tangent to a curve at a point in 2-space, in 3-space we can picture the **plane** tangent to a **surface** at a point.
- Consider the surface given by $z=f(x,y)$. Let ($x0$, $y0$, $z0$) be any point on this surface.
  - If $f(x,y)$ is differentiable at ($x0$, $y0$), then the surface has a tangent plane at ($x0$, $y0$, $z0$). The equation of the tangent plane at **(x0,y0, z0)** is given by:



Tangent Plane $(z-z0)= f_x(x0,y0)*(x-x0)+f_y(x0,y0) *(y-y0)$

Tangent Line similar form: $(y-y0)= f_x(x0) *(x-x0)$

  - where $f_x(x0,y0)$ is the partial derivative of f() WRT x calculated at $x0,y0$; similarly for $f_y(x0,y0)$

http://www.math.hmc.edu/calculus/tutorials/tangentplanes/

Notation

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

1D Linear Approximation

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

where

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

I.E.

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

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

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

Multi Variable Linear Approx.

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

Tangent Plane

$(z-z0)= f_x(x0,y0)(x-x0)+f_y(x0,y0)(y-y0)$

Function of variables

Calculate gradient vector by evaluating partial derivatives at the tangential point
Gradient vector at (1, 1) is (4, 2);
f'(1,1) = (4,2)
f(1,1)=3

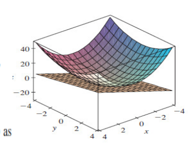
Tangent plane at (1, 1, 3) with gradient (4,2)

EXAMPLE 1 Find the tangent plane to the elliptic paraboloid $z = 2x^2 + y^2$ at the point (1, 1, 3).

SOLUTION Let $f(x, y) = 2x^2 + y^2$. Then

$$f_x(x, y) = 4x \qquad f_y(x, y) = 2y$$

$$f_x(1, 1) = 4 \qquad f_y(1, 1) = 2$$



(a)

Then (2) gives the equation of the tangent plane at (1, 1, 3) as

$$z - 3 = 4(x - 1) + 2(y - 1)$$

or

$$z = 4x + 2y - 3$$

Figure 2(a) shows the elliptic paraboloid and its tangent plane at (1, 1, 3) that we found in Example 1. In parts (b) and (c) we zoom in toward the point (1, 1, 3) by restricting the domain of the function $f(x, y) = 2x^2 + y^2$. Notice that the more we zoom in, the flatter the graph appears and the more it resembles its tangent plane.

[Adapted from Multivariable Calculus: Concepts and Contexts, James Stewart]

Tangent Plane Example

SECTION 11.4 TANGENT PLANES AND LINEAR APPROXIMATIONS ◆ 781

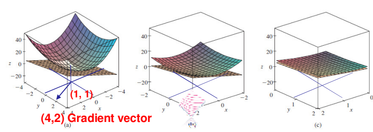


FIGURE 2 The elliptic paraboloid $z = 2x^2 + y^2$ appears to coincide with its tangent plane as we zoom in toward (1, 1, 3).

In Figure 3 we corroborate this impression by zooming in toward the point (1, 1) on a contour map of the function $f(x, y) = 2x^2 + y^2$. Notice that the more we zoom in, the more the level curves look like equally spaced parallel lines, which is characteristic of a plane.

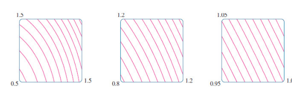


FIGURE 3 Zooming in toward (1, 1) on a contour map of $f(x, y) = 2x^2 + y^2$

Tangent Plane to Ellipoid Example

EXAMPLE 8 Find the equations of the tangent plane and normal line at the point (−2, 1, −3) to the ellipsoid

$$\frac{x^2}{4} + y^2 + \frac{z^2}{9} = 3$$

SOLUTION The ellipsoid is the level surface (with $k = 3$) of the function

$$F(x, y, z) = \frac{x^2}{4} + y^2 + \frac{z^2}{9}$$

▲ Figure 10 shows the ellipsoid, tangent plane, and normal line in Example 8.

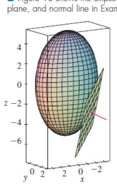


FIGURE 10

Therefore, we have

$$F_x(x, y, z) = \frac{x}{2} \qquad F_y(x, y, z) = 2y \qquad F_z(x, y, z) = \frac{2z}{9}$$

$$F_x(-2, 1, -3) = -1 \qquad F_y(-2, 1, -3) = 2 \qquad F_z(-2, 1, -3) = -\tfrac{2}{3}$$

Then Equation 19 gives the equation of the tangent plane at (−2, 1, −3) as

$$-1(x + 2) + 2(y - 1) - \tfrac{2}{3}(z + 3) = 0$$

which simplifies to $3x - 6y + 2z + 18 = 0$.

By Equation 20, symmetric equations of the normal line are

$$\frac{x+2}{-1} = \frac{y-1}{2} = \frac{z+3}{-\frac{2}{3}}$$

Lecture 2 Outline

- **Taylor Series: quadratic approximations**
- **Newton-Raphson quadratic convergence**
- **Multi-Dimensional Approximations (Planes)**
- **Directional Differentials, Total Differentials**
- **Vector plots, contour plots**
- **Gradient Descent**
  - Linear regression
- **Predicting Click Through Rates**
  - Linear Regression (using gradient descent, MCMC version on 1/26)
  - *Logistic Regression (using gradient descent, MCMC on 1/26)*
- ***Convexity, extreme values, mathematical programming***

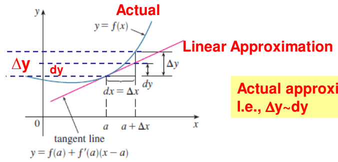
Approximate Δy with dy via the tangent

For a function of one variable, $y = f(x)$, we define the differential dx to be an independent variable; that is, dx can be given the value of any real number. The differential of y is then defined as

Difference in f(x); i.e., second term in Linear Taylor Expansion
$$f(x) = f(a) + f'(a)(x-a)$$

9 $$dy = f'(x)\,dx$$

(See Section 3.8.) Figure 6 shows the relationship between the increment Δy and the differential dy: Δy represents the change in height of the curve $y = f(x)$ and dy represents the change in height of the tangent line when x changes by an amount $dx = \Delta x$.



Actual approximated by Predicted
I.e., Δy~dy

dy is the predicted difference in (f(x) given the linear approximation)
Can change Δx as much as we like but the bigger the Δx the bigger the gap between the tangent approximation and the actual function (and dy and Δy)

Linear and Quadratic Approximations

- Approximate *f(X) for X* around point *a* by the tangent at a point (a, f(a))

$$y - y1 = m(x - x1)$$
$$f(x) - y1 = m(x - x1) \qquad f(x) = f(a) + f'(a)(x-a)$$
$$f(x) = y1 + m(x - x1)$$
$$f(x) = f(a) + f'(a)(x-a) \qquad \text{AT} \quad \text{(a, f(a))} \quad \text{slope} = \text{f'(a)}$$

- Taylor Series explores different approximations of f(X);
  - the above tangential form is linear approximation

- General Form of a Taylor Series

$$f(x) = \boxed{f(a) + f'(a)(x-a)} + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$

More compactly $f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k$

Total Differential, dz, for z=f(x, y) in 2D

For a differentiable function of two variables, $z = f(x, y)$, we define the **differentials** dx and dy to be independent variables; that is, they can be given any values. Then the **differential** dz, also called the **total differential**, is defined by

10 $$dz = f_x(x, y)\,dx + f_y(x, y)\,dy = \frac{\partial z}{\partial x}dx + \frac{\partial z}{\partial y}dy$$ Total Change in f(x,y)

$$f(\vec{x}) \approx f(\vec{a}) + dz$$
$$f(\vec{x}) \approx f(\vec{a}) + \nabla f(\vec{a}) \cdot (\vec{x} - \vec{a})$$

(Compare with Equation 9.) Sometimes the notation df is used in place of dz.

If we take $dx = \Delta x = x - a$ and $dy = \Delta y = y - b$ in Equation 10, then the differential of z is

$$dz = f_x(a, b)(x - a) + f_y(a, b)(y - b)$$

- Estimated change in z using total differential
- Total Differential in 2D (estimated change in z=f(x) using a linear approximation)
-
- This corresponds to the second term (the linear term) in Taylors expansion

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

Total Differential in 2D (estimated change in z)

So, in the notation of differentials, the linear approximation (4) can be written as

$$f(x, y) \approx f(a, b) + dz$$ First-order Taylor Series

Figure 7 is the three-dimensional counterpart of Figure 6 and shows the geometric interpretation of the differential dz and the increment Δz: dz represents the change in height of the tangent plane, whereas Δz represents the change in height of the surface $z = f(x, y)$ when (x, y) changes from (a, b) to $(a + \Delta x, b + \Delta y)$.

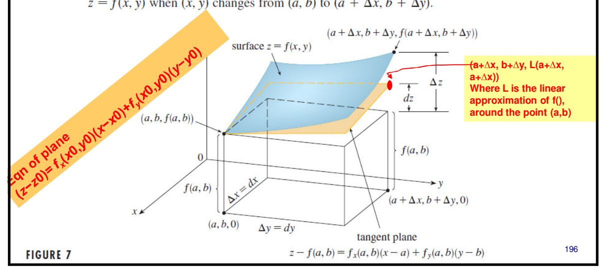


FIGURE 7

Total Differential in 2D (estimated change in z)

So, in the notation of differentials, the linear approximation (4) can be written as

$$f(x, y) \approx f(a, b) + dz$$

Figure 7 is the three-dimensional counterpart of Figure 6 and shows the geometric interpretation of the differential dz and the increment Δz: dz represents the change in height of the tangent plane, whereas Δz represents the change in height of the surface $z = f(x, y)$ when (x, y) changes from (a, b) to $(a + \Delta x, b + \Delta y)$.

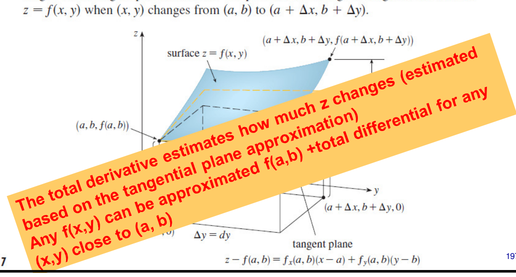


FIGURE 7

Total Differential in 2D: An Example

(a) If $z = f(x, y) = x^2 + 3xy - y^2$, find the differential dz. change in height of f(x)
(b) If x changes from 2 to 2.05 and y changes from 3 to 2.96, compare the values of Δz and dz. change in height of f(x) when I travel in (0.05, -0.04)

SOLUTION

(a) Definition 10 gives Total Differential

$$dz = \frac{\partial z}{\partial x}dx + \frac{\partial z}{\partial y}dy = (2x + 3y)\,dx + (3x - 2y)\,dy$$

▲ In Example 4, dz is close to Δz because the tangent plane is a good approximation to the surface $z = x^2 + 3xy - y^2$ near (2, 3, 13). (See Figure 8.)

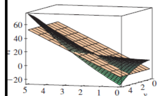


FIGURE 8

(b) Putting $x = 2$, $dx = \Delta x = 0.05$, $y = 3$, and $dy = \Delta y = -0.04$, we get

$$dz = [2(2) + 3(3)]0.05 + [3(2) - 2(3)](-0.04)$$
$$= 0.65$$

Estimated z difference between f(x, y) and f(a, b) $z=f_{Tang(a,b)}$ (x,y)-f(a,b))

The increment of z is

$$\Delta z = f(2.05, 2.96) - f(2, 3)$$
$$= [(2.05)^2 + 3(2.05)(2.96) - (2.96)^2] - [2^2 + 3(2)(3) - 3^2]$$
$$= 0.6449$$

Actual z difference (i.e., z=f (x,y)-f(a,b))

Notice that $\Delta z \approx dz$ but dz is easier to compute.

Total Differential and Directional Derivative

So, in the notation of differentials, the line[...]n as

Figure 7 is [...] geometric int[...] presents the change in [...]hange in height of the surface [...] $(a + \Delta x, b + \Delta y)$.

he total derivative tells us how much z changes (only estimated as it is based on the tangential plane approximation) when we travel in a particular direction u, i.e.,$D_u f(x,y)=\nabla f(x,y) \bullet u$
Where u=(x-a, y-b)
Any f(x,y) can be approximated by f(a,b) +total differential

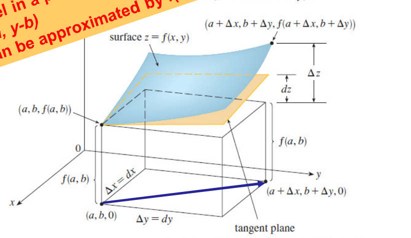
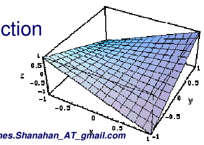


FIGURE 7

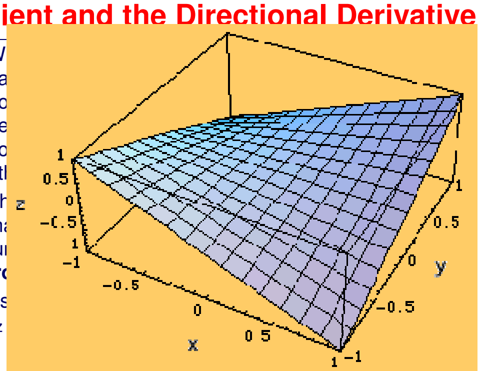
Gradient and the Directional Derivative

- When you are hiking on a mountain or a slope you have a choice of many directions in which you can go. Starting at the same point some directions head generally upward; some directions head generally downward; and some directions are steeper than others.
- The **directional derivative** of a function, **z = f(x, y)**, that is, the slope of the surface described by this function **as we go in different directions starting from the same point.**
- As an example consider the function
  z = xy



Gradient and the Directional Derivative

- W[...] ha[...] go[...] ge[...] do[...] ot[...]
- Th[...] th[...] fu[...] fr[...]
- As[...]
  z



Directional (versus Partial) Derivatives

Partial derivatives measure the differences in the direction of the coordinate axes

Partial derivatives

- **If f is a real-valued function on R^n, then the partial derivatives of f measure its variation in the direction of the coordinate axes.**
  - For example, if f is a function of x and y, then its partial derivatives measure the variation in f in the x direction and the y direction.
- **They do not, however, directly measure the variation**

(See other notations below.) If the function f is differentiable at $\vec{x}$, then the directional derivative exists along any unit vector $\vec{u}$, and one has

$$\nabla_{\vec{u}} f(\vec{x}) = \nabla f(\vec{x}) \cdot \vec{u}$$

where the ∇ on the right denotes the gradient and $\cdot$ is the Euclidean inner product. At any point $\vec{x}$, the directional derivative of f intuitively represents the rate of change in f along $\vec{u}$ at the point $\vec{x}$.

One sometimes permits non-unit vectors, allowing the directional derivative to be taken in the direction of $\vec{v}$, where $\vec{v}$ is any nonzero [...] one must modify the definitions to account for the fact that $\vec{v}$ may not be normalized, so one has

irection derivatives measure the differences in ny other direction

Directional Derivative== Total Deriv. == Change in f(x)

If f is a real-valued function on $\mathbf{R}^n$, then the partial derivatives of f measure its variation in the direction of the coordinate axes. For example, if f is a function of x and y, then its partial derivatives measure the variation in f in the x direction and the y direction. They do not, however, directly measure the variation of f in any other direction, such as along the diagonal line $y = x$. These are measured using directional derivatives. Choose a vector

$$\mathbf{v} = (v_1, \ldots, v_n).$$

The **directional derivative** of f in the direction of $\mathbf{v}$ at the point $\mathbf{x}$ is the limit

$$D_{\mathbf{v}} f(\boldsymbol{x}) = \lim_{h \to 0} \frac{f(\boldsymbol{x} + h\mathbf{v}) - f(\boldsymbol{x})}{h}.$$

Let λ be a scalar. The substitution of h/λ for h changes the λ**v** direction's difference quotient into λ times the **v** direction's difference quotient. Consequently, the directional derivative in the λ**v** direction is λ times the directional derivative in the **v** direction. Because of this, directional derivatives are often considered only for unit vectors **v**.

If all the partial derivatives of f exist and are continuous at $\boldsymbol{x}$, then they determine the directional derivative of f in the direction **v** by the formula:

$$D_{\mathbf{v}} f(\boldsymbol{x}) = \sum_{j=1}^{n} v_j \frac{\partial f}{\partial x_j}.$$

Change in f(x) when we travel in direction v
We can approximate the change in z (i.e., f(x) – f(x+v)) using $D_v f(x,y)=\nabla f(x,y) \bullet v$

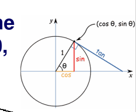
This is a consequence of the definition of the total derivative. It follows that the directional derivative is linear in **v**.

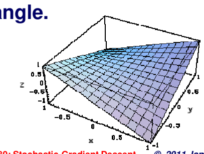
The same definition also works when f is a function with values in $\mathbf{R}^m$. We just use the above definition in each component of the vectors. In this case, the directional derivative is a vector in $\mathbf{R}^m$.

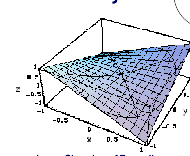
$D_A f(x,y)=\nabla f(x,y) \bullet (x-A1,y-A2)$

Directional Derivatives

- Suppose that we start at the point **(0, 0)** and go one unit in several different directions.
  - The graph below shows four different directions marked by curves starting at the origin and it also shows all the points we would reach if we tried all possible directions and walked one unit.
- **When we say "we walk one unit," we mean one unit in the xy-direction. Thus, we walk from (0, 0) to the point (cos θ, sin θ) where θ is any angle.**







Directional Derivative

- We are interested in the rate at which **z** changes as we move away from **x** in various different directions. Each possible direction is indicated by a **unit vector**,
- **u = (u_1, u_2, ... u_n)**
- The directional derivative in the direction **u** is given by

$$D_{\vec{u}}z = \lim_{h \to 0} \frac{f(x_1 + hu_1, x_2 + hu_2, \ldots x_n + hu_n) - f(x_1, x_2, \ldots x_n)}{h}$$

- Sometimes we use the notation $\mathbf{f}_u$ for the directional derivative.
  - f_u = grad f · u

Total Differential (Directional derivative)

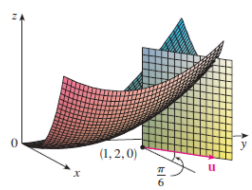
- **Most relationships depend on several variables**
  - $y = f(x_1, x_2, x_3, \ldots, x_N)$
- **Recall that the partial derivative, $\partial y/\partial x_1$, is the change in *y* when we change x_1, etc.**
- **Now we're interested in the total effect on *y* when all the *x*'s are changed by a small amount.**
- **This is the Total Differential of *f* and is denoted by *dy* in direction dx at df/dx|**

$$dy = \frac{\partial f}{\partial x_1}dx_1 + \frac{\partial f}{\partial x_2}dx_2 + \ldots + \frac{\partial f}{\partial x_N}dx_N$$

Directional Derivatives

- **The directional derivative of a multivariate differentiable function along a given vector V at a given point P intuitively represents the instantaneous rate of change of the function, moving through P, in the direction of V.**
- **It therefore generalizes the notion of a partial derivative, in which the direction is always taken parallel to one of the coordinate axes.**

▲ The directional derivative $D_{\mathbf{u}}f(1, 2)$ in Example 2 represents the rate of change of z in the direction of $\mathbf{u}$. This is the slope of the tangent line to the curve of intersection of the surface $z = x^3 - 3xy + 4y^2$ and the vertical plane through $(1, 2, 0)$ in the direction of $\mathbf{u}$ shown in Figure 5.



Questions for Thought

1. **Gradient Vector: Find the rate of change in the direction of a given vector WRT a given point (and tangential approximation)?**
2. **In what direction does f() have the maximum rate of change?**
3. **What is this maximum rate of change?**

Maximizing the Directional Derivative

Suppose we have a function f of two or three variables and we consider all possible directional derivatives of f at a given point. These give the rates of change of f in all possible directions. We can then ask the questions: In which of these directions does f change fastest and what is the maximum rate of change? The answers are provided by the following theorem.

15 Theorem Suppose f is a differentiable function of two or three variables. The maximum value of the directional derivative $D_{\mathbf{u}}f(\mathbf{x})$ is $|\nabla f(\mathbf{x})|$ and it occurs when $\mathbf{u}$ has the same direction as the gradient vector $\nabla f(\mathbf{x})$.

Proof From Equation 9 or 14 we have

$$D_{\mathbf{u}}f = \nabla f \cdot \mathbf{u} = |\nabla f||\mathbf{u}|\cos\theta = |\nabla f|\cos\theta$$

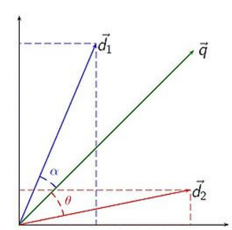
where θ is the angle between ∇f and $\mathbf{u}$. The maximum value of $\cos\theta$ is 1 and this occurs when $\theta = 0$. Therefore, the maximum value of $D_{\mathbf{u}}f$ is $|\nabla f|$ and it occurs when $\theta = 0$, that is, when $\mathbf{u}$ has the same direction as ∇f. ■

Cosine of the angle

- **the cosine of the angle between the vectors instead of the angle**

$$\cos\theta = \frac{\langle q, d_1 \rangle}{\|q\| * \|d_1\|}$$

$$\langle q, d_1 \rangle = \|q\| * \|d_1\| \cos\theta$$



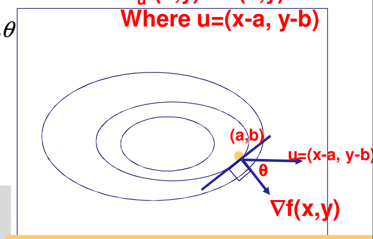
Which direction is maximising f(x,y)?

$f(x) = f(a) + f'(a)(x-a)$
$f(x) = f(a) + \nabla f(a)(x-a)$

Recall: Directional derivative $D_u f(x,y)$ (approximated difference in *f(x,y)* if we travel in direction *u* from *(x, y)*)

$$\begin{aligned} D_u f &= \nabla f^T \cdot u \\ \nabla f^T \cdot u &= \|\nabla f\| \cdot \|u\| \cos\theta \\ &= \|\nabla f\| \cos\theta \\ \nabla f^T \cdot u &= \|\nabla f\| \\ \nabla f^T \cdot u &= \nabla f^T \cdot \nabla f \\ u &= \nabla f \end{aligned}$$

$D_u f(x,y) = \nabla f(x,y) \bullet u$
Where u=(x-a, y-b)

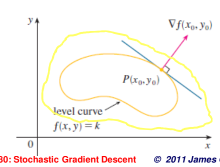


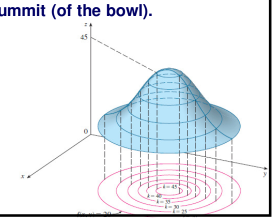
- cos(θ) maxes at 1 when θ=0
- So $D_u f(x,y)$ maxes when u equals ∇f
- Gradient vector is the steepest change

Thus, the direction of **steepest ascent** is ∇f(x,y) and the direction of **steepest descent** is -∇f(x,y)

Significance of the Gradient Vector

- **The gradient vector, ∇f(x,y), gives the direction of fastest increase of f(x, y) (assuming a two-variable function here). [Newton-Raphson]**
- **The gradient vector, ∇f(x,y), is orthogonal to the contour lines**
- **Imagine climbing an upside-down bowl from below, where I can move in any <x, y> direction (NOTE I cant move in z); x, and y are independent variables.**
- **If I follow the level curve (f(x,y)=k) then I make no progress to the summit or bottom but if I move perpendicular to the level curve then I make the quickest progress to the summit (of the bowl).**





Steepest Ascent/Descent

- Since **u** is a unit vector and
- **f_u = grad f · u = ||u|| ||grad f|| cos(θ)**
- **= ||grad f|| cos(θ)**
- where **theta** is the angle between **grad f** and **u**, we see that the directional derivative is at its maximum when **u** is pointing in the same direction as **grad f** and is at a minimum when **u** is pointing in the opposite direction. (zero angle (cos(0) =1)
- Thus, the direction of **steepest ascent** is **grad f** and the direction of **steepest descent** is **-grad f**.

Examples:
http://www.math.montana.edu/frankw//ccp/multiworld/twothree/gradient/learn.htm

Gradients and Contour Curves

- **Examine the relation between contour curves, or by another name, level sets, and gradients**
  - [moving along a direction bring no change in f(x,y)]
- **Theorem on Gradients and Contour Curves**
  - Consider any point (x_0,y_0), and the level curve of f through this point (i.e., the level curve of f at value $f(x_0,y_0)$).
  - Then the gradient of f at $f(x_0,y_0)$ is perpendicular to the tangent direction along the level curve of f though (x_0,y_0).
  - It is very easy to see why this theorem is true.
  - Suppose that (a,b) is any vector that is tangent to the level curve of f through (x_0,y_0). Then, as you move in the (a,b) direction, you are at that instant moving along the level curve, and the value of f does not change.
  - So the directional derivative in this direction is zero; i.e., dot(gradf(x_0,y_0),(a,b)) = 0
  - This is the perpendicularity that we wanted to establish.

Directional Derivative Examples

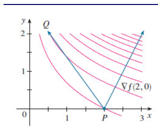


FIGURE 7

▲ At (2, 0) the function in Example 6 increases fastest in the direction of the gradient vector $\nabla f(2,0) = \langle 1, 2\rangle$. Notice from Figure 7 that this vector appears to be perpendicular to the level curve through (2, 0). Figure 8 shows the graph of f and the gradient vector.

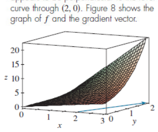


FIGURE 8

EXAMPLE 6
(a) If $f(x, y) = xe^y$, find the rate of change of f at the point $P(2, 0)$ in the direction from P to $Q(\frac{1}{2}, 2)$.
(b) In what direction does f have the maximum rate of change? What is this maximum rate of change?

SOLUTION
(a) We first compute the gradient vector:

$$\nabla f(x,y) = \langle f_x, f_y\rangle = \langle e^y, xe^y\rangle$$
$$\nabla f(2,0) = \langle 1, 2\rangle$$

The unit vector in the direction of $\overrightarrow{PQ} = \langle -1.5, 2\rangle$ is $\mathbf{u} = \langle -\frac{3}{5}, \frac{4}{5}\rangle$, so the rate of change of f in the direction from P to Q is

$$\begin{aligned} D_{\mathbf{u}} f(2,0) &= \nabla f(2,0)\cdot \mathbf{u} = \langle 1,2\rangle \cdot \langle -\tfrac{3}{5}, \tfrac{4}{5}\rangle \\ &= 1(-\tfrac{3}{5}) + 2(\tfrac{4}{5}) = 1 \end{aligned}$$

(b) According to Theorem 15, f increases fastest in the direction of the gradient vector $\nabla f(2,0) = \langle 1, 2\rangle$. The maximum rate of change is

$$|\nabla f(2,0)| = |\langle 1,2\rangle| = \sqrt{5}$$

EXAMPLE 7 Suppose that the temperature at a point (x, y, z) in space is given by $T(x, y, z) = 80/(1 + x^2 + 2y^2 + 3z^2)$, where T is measured in degrees Celsius and x, y, z in meters. In which direction does the temperature increase fastest at the point $(1, 1, -2)$? What is the maximum rate of increase?

Gradients, Gradient Plots and Tangent Planes

- http://www-users.math.umd.edu/~jmr/241/gradients.html

4D function: function of 3 Variables

EXAMPLE 8 Find the equations of the tangent plane and normal line at the point $(-2, 1, -3)$ to the ellipsoid

$$\frac{x^2}{4} + y^2 + \frac{z^2}{9} = 3$$

SOLUTION The ellipsoid is the level surface (with $k = 3$) of the function

Therefore, we have

$$F_x(x, y, z) = \frac{x}{2} \qquad F_y(x, y, z) = 2y \qquad F_z(x, y, z) = \frac{2z}{9}$$

$$F_x(-2, 1, -3) = -1 \qquad F_y(-2, 1, -3) = 2 \qquad F_z(-2, 1, -3) = -\tfrac{2}{3}$$

Then Equation 19 gives the equation of the tangent plane at $(-2, 1, -3)$ as

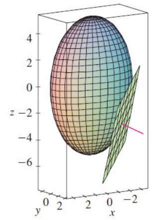
$$-1(x + 2) + 2(y - 1) - \tfrac{2}{3}(z + 3) = 0$$

which simplifies to $3x - 6y + 2z + 18 = 0$.

By Equation 20, symmetric equations of the normal line are

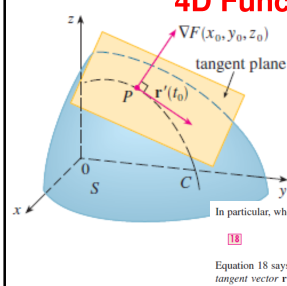
$$\frac{x + 2}{-1} = \frac{y - 1}{2} = \frac{z + 3}{-\frac{2}{3}}$$

▲ Figure 10 shows the ellipsoid, tangent plane, and normal line in Example 8.



Level surface with 3D Tangent Plane; with a 3D nor (as opposed to 2D contour plot with a 2D normal

4D Function: Tangent Plane



In particular, when $t = t_0$ we have $\mathbf{r}(t_0) = \langle x_0, y_0, z_0 \rangle$, so

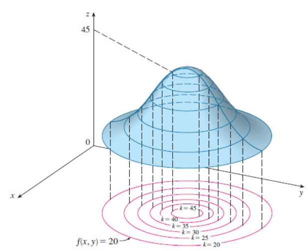
$$\boxed{18} \qquad \nabla F(x_0, y_0, z_0) \cdot \mathbf{r}'(t_0) = 0$$

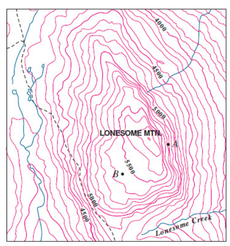
Equation 18 says that *the gradient vector at P, $\nabla F(x_0, y_0, z_0)$, is perpendicular to the tangent vector $\mathbf{r}'(t_0)$ to any curve C on S that passes through P.* (See Figure 9.) If $\nabla F(x_0, y_0, z_0) \neq \mathbf{0}$, it is therefore natural to define the **tangent plane to the level surface $F(x, y, z) = k$ at $P(x_0, y_0, z_0)$** as the plane that passes through P and has normal vector $\nabla F(x_0, y_0, z_0)$. Using the standard equation of a plane (Equation 9.5.6), we can write the equation of this tangent plane as

$$\boxed{19} \quad F_x(x_0, y_0, z_0)(x - x_0) + F_y(x_0, y_0, z_0)(y - y_0) + F_z(x_0, y_0, z_0)(z - z_0) = 0$$

Video: http://academicearth.org/lectures/tangent-planes-and-linear-approximation

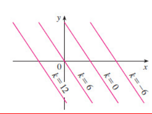
Level Curves/Surfaces





Plotting Level Curves

FIGURE 7



EXAMPLE 7 Sketch the level curves of the function $f(x, y) = 6 - 3x - 2y$ for the values $k = -6, 0, 6, 12$.

SOLUTION The level curves are

$$6 - 3x - 2y = k \qquad \text{or} \qquad 3x + 2y + (k - 6) = 0$$

This is a family of lines with slope $-\frac{3}{2}$. The four particular level curves with $k = -6, 0, 6$, and 12 are $3x + 2y - 12 = 0$, $3x + 2y - 6 = 0$, $3x + 2y = 0$, and $3x + 2y + 6 = 0$. They are sketched in Figure 8. The level curves are equally spaced parallel lines because the graph of f is a plane (see Figure 4 in Section 9.6).

$$g(x, y) = \sqrt{9 - x^2 - y^2} \qquad \text{for} \quad k = 0, 1, 2, 3$$

SOLUTION The level curves are

$$\sqrt{9 - x^2 - y^2} = k \qquad \text{or} \qquad x^2 + y^2 = 9 - k^2$$

This is a family of concentric circles with center $(0, 0)$ and radius $\sqrt{9 - k^2}$. The cases $k = 0, 1, 2, 3$ are shown in Figure 9. Try to visualize these level curves lifted up to form a surface and compare with the graph of g (a hemisphere) in Figure 2.

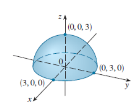
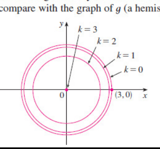


FIGURE 2
Graph of $g(x, y) = \sqrt{9 - x^2 - y^2}$

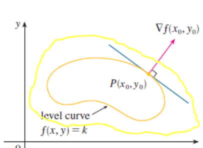


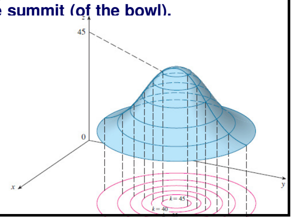
In R try:

```
x <- seq(-3, 3, length= 30)
contour(outer(x, x, "+"), method = "edge")
contour(outer(x, x, "*"), method = "edge")
```

Significance of the Gradient Vector

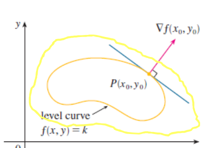
- The gradient vector, ∇f(x,y), gives the direction of fastest increase of f(x, y) (assuming a two-variable function here).
- The gradient vector, ∇f(x,y), is orthogonal to the contour lines
- Imagine climbing an upside-down bowl from below, where I can move in any <x, y> direction (NOTE I cant move in z); x, and y are independent variables.
- If I follow the level curve (f(x,y)=k) then I make no progress to the summit or bottom but if I move perpendicular to the level curve then I make the quickest progress to the summit (of the bowl).

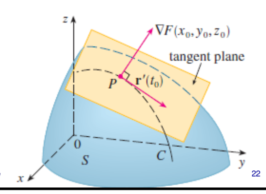




Significance of the Gradient Vector

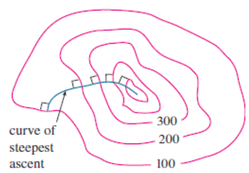
- The gradient vector, ∇f(x,y,z), gives the direction of fastest increase of f(x, y, z) (assuming a three-variable function here).
- The gradient vector, ∇f(x,y,z), is orthogonal to the level surface S of f through P (i.e., x_0, y_0, z_0))
- Imagine climbing an upside-down bowl from below; where I can move in any <x, y> direction (I cant move in z). If I follow the level curve (f(x,y)=k) then I make no progress to the summit or bottom but if I move perpendicular to the level curve then I make the quickest progress to the summit (of the bowl).

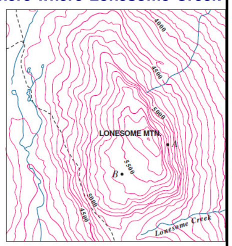




Gradient Vector: Hiker's Perspective

- If we consider a topographical map of a hill and let f(x, y) represent the height above sea level at a point with coordinates (x, y) , then a curve of steepest ascent can be drawn by making it perpendicular to all of the contour lines.
- This phenomenon can also be noticed here where Lonesome Creek follows a curve of steepest descent.





Lecture Outline

- R
- Lines, Tangents, Taylors Theorem
- Turning points, Roots, 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
- Predicting Click Through Rates
  - Linear Regression
  - *Logistic Regression*

Follow Gradient for Maximization

Follow negative of Gradient for Minimization

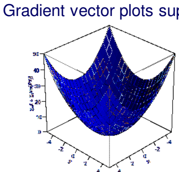
- Gradient descent is a first-order optimization algorithm.
- To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point.
- If instead one takes steps proportional to the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent.

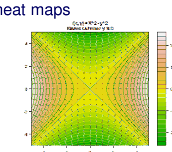
Gradient Vector Plots

- The gradient gives us a vector at each point (x, y) that is pointing uphill
- We can visualize these using a gradient vector plot

See example.gradientPlots()

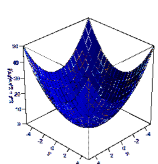
  - Plot 3 dimensional surfaces f(x,y)
  - Heat maps
  - Gradient vector plots
  - Gradient vector plots superimposed on heat maps





Plotting 3D Surfaces in R

```
#put two plots side by side (i.e., 1 row and 2 columns)
par(mfrow=c(1, 2))
x <- seq(-3, 3, length= 30)
y <- x
f <- function(x,y) { x^2 + 2 * y^2 }
z <- outer(x, y, f)
#Plot 3D surface of function
#Modify theta and phi for different perspective
persp(x, y, z, theta = 135, phi = 30, col = "blue", scale = FALSE,
    ltheta = -120, shade = 0.75, ticktype = "detailed", expand = 0.2,
)
```

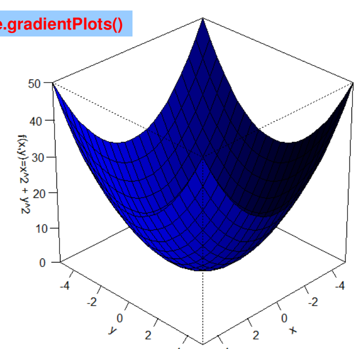


See example.gradientPlots()

R: outer()
The outer product of the arrays X and Y is the array A with dimension c(dim(X), dim(Y)) where element
A[c(arrayindex.x, arrayindex.y)] = FUN(X[arrayindex.x], Y[arrayindex.y], ...).

3D Plot of f(x,y)=x^+y^2

See example.gradientPlots()



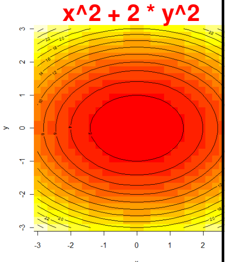
HeatMap and Contour Plot

```
x <- seq(-3, 3, length= 30);   y <- x
f <- function(x,y) { x^2 + 2 * y^2 }
z <- outer(x, y, f)
```

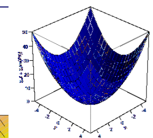
See example.gradientPlots()

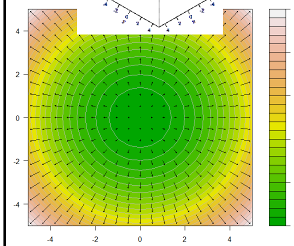
- #new plot of isolines and heatmap
- image(x,y,z)   #heat image of surface
- contour(x,y,z, add=TRUE) #add contours

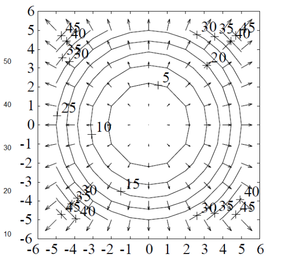
x^2 + 2 * y^2



Gradient Vector Plots of f(x,y)=x^+y^2







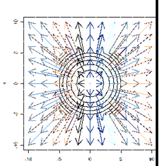
3. Superimpose the level curves of $f(x, y) = x^2 + y^2$

http://online.redwoods.cc.ca.us/instruct/darnold/MULTCALC/grad/grad.pdf

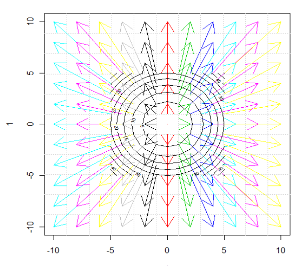
Ugly Unnormalized Gradient Plot

```
#ugly unnormalized gradient vector plot
 # plot vector plot and contour plot
 x <- -5:5
 y <- x
 f <- function(x,y) { x^2 + y^2 }
 z <- outer(x, y, f)
 u=2*x  #fprime_x(x,y) = df/dx =d(x^2 + y^2)/dx=2x
 v=2*y
 plot(1, 1, xlim=c(-10, 10), ylim=c(-10, 10), pch="")
 grid(length(x))
 for(i in x) {
  for (j in y) {
   arrows(i,j, 2*i, 2*j, col=i+10)  #(f'x(x,y), f'y(x,y), col=colour
  }
 }
 contour(x,y,z, add=TRUE) #add contours
```

In example.gradientPlots()

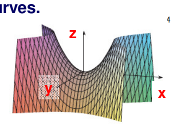


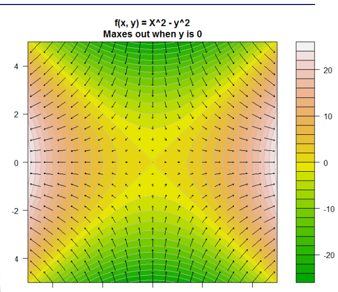
Ugly Unnormalized Gradient Plot



Gradient Vector Field for f(x,y) = $x^2 - y^2$

- Sampled gradient vector plot superimposed on a function heat map of f(x, y) = $x^2 - y^2$
- Each gradient vector is plotted starting at the point
- *As expected, the gradient vectors point "uphill" and are perpendicular* to the level curves.

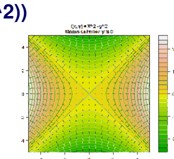




Prettified Gradient Plot

(looks like a quiver!)

```
#prettified gradient vector plot using quiver()
#f <- expression( (3*x^2 + y) * exp(-x^2-y^2))
f <- expression( (x^2) - (y^2))
#f <-expression((x^2+x^2))
x <- y <- seq(-5, 5, by=0.5)
par(mar=c(3,3,3,3))
quiver2(f,x,y, color.palette=terrain.colors,
  main="f(x, y) = X^2 - y^2\nMaxes out when y is 0")
```

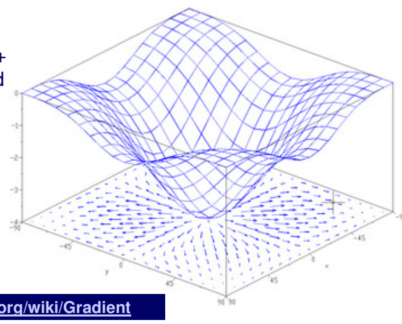


Gradient Vector at Extrema is <0, 0, ..>

- The gradient is a fancy word for derivative, or the rate of change of a function.
- It's a vector (a direction to move) that points in the direction of greatest increase of a function is zero at a local maximum or local minimum (because there is no single direction of increase); the magnitude of the vector is zero. Gradient at turning points =<0, 0, 0…,0)
- The term gradient typically refers to the derivative of vector functions, or functions of more than one variable. Yes, you can say a line has a gradient (its slope), but using the term gradient for single-variable functions is unnecessarily confusing. Keep it simple.
- http://betterexplained.com/articles/vector-calculus-understanding-the-gradient/

Gradient Example

- The gradient of the function $f(x,y) = -(\cos^2 x + \cos^2 y)^2$ depicted as a vector field on the bottom plane



http://en.wikipedia.org/wiki/Gradient

Exercise 1.4 : Vector Field vs. 3DPlot

quiver2() available in R code

- Compute the gradient of the following function.
  - $z=f(x,y) = x^2 + y$
- This gives us a vector at each point (x, y) that is pointing uphill.
- In R plot the these vectors. This plot is also known as a vector field. Hint: use quiver2(); provided in R Code.
- Plot the 3D of this function
- Compare the vector field plot with a three-dimensional plot of the indicated function. Does the vector field appear to be pointing upward?

quiver2() 1/2

```
#got this from http://addictedtor.free.fr/graphiques/graphcode.php?graph=128
#plot a normalized gradient plot (which looks like a quiver of arrows)
 quiver2 <- function(expr,       x,          y,           nlevels=20,            length=0.05,          ...){
   z <- expand.grid(x,y)
   xx  <- x
   x   <- z[,1]
   yy  <- y
   y   <- z[,2]

   #browser()
   fxy <- eval(expr)
   grad_x <- eval(D(expr, "x"))
   grad_y <- eval(D(expr, "y"))

  dim(fxy) <- c(length(xx), length(yy))  #pour vector into table
  dim(grad_x) <- dim(fxy)
  dim(grad_y) <- dim(fxy)

  maxlen <- min(diff(xx), diff(yy)) * .9
  grad_x <- grad_x / max(grad_x) * maxlen #normalize gradient components
  grad_y <- grad_y / max(grad_y) * maxlen

  filled.contour(xx,  yy, fxy, nlevels=nlevels,
   plot.axes = {
    contour(xx, yy, fxy, add=T, col="gray",
          nlevels=nlevels, drawlabels=FALSE)
```


quiver2() 2/2

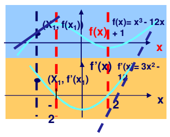
```
…..
 filled.contour(xx,  yy, fxy, nlevels=nlevels,
   plot.axes = {
    contour(xx, yy, fxy, add=T, col="gray",
          nlevels=nlevels, drawlabels=FALSE)

    arrows(x0  = x,
          x1  = x + grad_x,
          y0  = y,
          y1  = y + grad_y,
          length = length*min(par.uin()))

    axis(1)
    axis(2)
   },
   ...)
}
```


Use of quiver2()

```
#prettified gradient vector plot using quiver()
  #f <- expression( (3*x^2 + y) * exp(-x^2-y^2))
  f <- expression( (x^2) - (y^2))
  #f <-expression((x^2+y))
  x <- y <- seq(-5, 5, by=0.5)
  par(mar=c(3,3,3,3))
  quiver2(f,x,y, color.palette=terrain.colors, main="f(x, y) = X^2
     - y^2\nMaxes out when y is 0")
```

Recap on finding minimum or maximum

$$\max/\min \quad f(x) = f(x_1, \dots, x_n)$$
$$\text{subject to} \quad x \in R^n$$



- **Given f(x1, x2, …) find a candidate minimum or maximum (stationary points)**
- **Assume f'(x) and *H(x)* exists for all $x \in S$**
- **Locate candidate extrema using f'(x) = 0 and boundary points**
- **Steps**
  - Find roots of the gradient equation f'(X)
    - Use Newton-Raphson

Multivariate Newton's Method

Suppose that the objective f is a function of multiple arguments, $f(w_1, w_2, \dots w_p)$. Let's bundle the parameters into a single vector, $\vec{w}$. Then the Newton update is

$$\vec{w}_{n+1} = \vec{w}_n - H^{-1}(w_n)\nabla f(\vec{w}_n) \quad (16)$$

Find the roots of an equation or system of equations
Calculating gradient and Hessian not very time-consuming but calculating the inverse of H is!

where ∇f is the **gradient** of f, its vector of partial derivatives $[\partial f/\partial w_1, \partial f/\partial w_2, \dots \partial f/\partial w_p]$, and H is the **Hessian** of f, its matrix of second partial derivatives, $H_{ij} = \partial^2 f/\partial w_i \partial w_j$.

Calculating H and ∇f isn't usually very time-consuming, but taking the inverse of H is, unless it happens to be a diagonal matrix. This leads to various **quasi-Newton** methods, which either approximate H by a diagonal matrix, or take a proper inverse of H only rarely (maybe just once), and then try to update an estimate of $H^{-1}(w_n)$ as w_n changes. (See section 8.3 in the textbook for more.)

In R, have a look at
?optim #method=BFGS

[http://www.stat.cmu.edu/~cshalizi/350/2008/lectures/29/lecture-29.pdf]
[Hand, Manilla, Smith, Data Mining, Section 8.3]

Operational Algorithms

- **Quasi-Newton (BFGS) - Popular in practice**
  - **Avoid computing the inverse of Hessian matrix**
  - But, it still requires computing the **B** matrix (approximate Hessian) → large storage
  - Broyden–Fletcher–Goldfarb–Shanno (BFGS) method is a method for solving nonlinear optimization problems
  - The BFGS method approximates Newton's method,
    - a class of hill-climbing optimization techniques that seeks a stationary point of a (twice continuously differentiable) function:
    - For such problems, a necessary condition for optimality is that the gradient be zero.
  - Newton's method and the BFGS methods need not converge unless the function has a quadratic Taylor expansion near an optimum. These methods use the first and second derivatives.
- **Limited-Memory Quasi-Newton (L-BFGS)**
  - Even avoid explicitly computing **B** matrix

Quasi-Newton Method

- **Approximate the Hessian matrix H^{-1} with another B matrix:**

$$\vec{x}^{new} \leftarrow \vec{x}^{old} - \mathbf{B}\frac{\partial f(\vec{x})}{\partial \vec{x}}$$

- **B is updated iteratively (BFGS):**

$$\mathbf{B}_{k+1} = \mathbf{B}_k - \frac{(\mathbf{B}_k \vec{p}_k)(\mathbf{B}_k \vec{p}_k)^T}{\vec{p}_k^T \mathbf{B}_k \vec{p}_k} + \frac{\vec{y}_k \vec{y}_k^T}{\vec{y}_k^T \vec{p}_k}$$
$$\vec{p}_k = \vec{x}_{k+1} - \vec{x}_k,\ \vec{y}_k = \vec{g}_{k+1} - \vec{g}_k$$

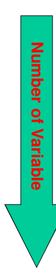
  - Utilizing derivatives of previous iterations

Efficiency

Small

Medium

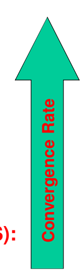
Large



Standard Newton-Raphson method: $O(n^3)$

Quasi Newton method (BFGS): $O(n^2)$

Limited-memory Quasi Newton method (L-BFGS): O(n)

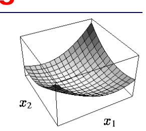


V-Fast

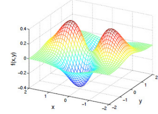
Fast

R-Fast

General Approach to Finding Extrema



- **Well-behaved version spaces**
  - Convex or concave function (±definiteness)
  - Algorithms seek a local extrema knowing that it will be global
    - If f() is a concave function then local maximum is a global maximum
    - If f() is a convex function then local minimum is a global minimum
- **Otherwise**
  - We resort to local approximations
    - Hill-Climbing
    - Simulated annealing
    - Commonly used in Neural Networks

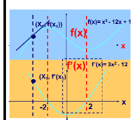


Lecture Outline

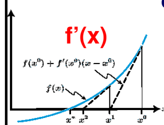
- R
- Lines, Tangents, Taylors Theorem
- Turning points, Roots, Newton-Raphson
- Taylor Series: quadratic approximations
- Newton-Raphson quadratic convergence
- Multi-Dimensional Approximations (Planes)
- Directional Differentials, Total Differentials
- Vector plots, contour plots
- **Gradient Descent**
  - Linear regression
- Predicting Click Through Rates
  - Linear Regression
  - Logistic Regression

Gradient Descent (a simpler root finder)

$$x^{i+1} = x^i - \frac{f'(x^i)}{f''(x^i)}$$ *Iteration function* **Newton-Raphson In 1-Dimension**



- Calculating f''(x), the Hessian H, and inverting it is complex so simpler algorithms have been developed such gradient descent



$$x^{i+1} = x^i - a^i f'(x^i)$$ **Gradient Descent**

- How large should I step in the positive gradient direction (gradient ascent)
  - or in the negative gradient direction (gradient descent)

Gradient Descent

Initialization: Select ϵ and any initial trial solution x'. Go first to the stopping rule.
Iteration:

1. Express $f(\mathbf{x}' + t\,\nabla f(\mathbf{x}'))$ as a function of t by setting

$$x_j = x_j' + t\left(\frac{\partial f}{\partial x_j}\right)_{\mathbf{x}=\mathbf{x}'}, \qquad \text{for } j = 1, 2, \ldots, n,$$

and then substituting these expressions into $f(\mathbf{x})$.

2. Use the one-dimensional search procedure (or calculus) to find $t = t^*$ that maximizes $f(\mathbf{x}' + t\,\nabla f(\mathbf{x}'))$ over $t \geq 0$.
3. Reset $\mathbf{x}' = \mathbf{x}' + t^*\,\nabla f(\mathbf{x}')$. Then go to the stopping rule.

Stopping rule: Evaluate $\nabla f(\mathbf{x}')$ at $\mathbf{x} = \mathbf{x}'$. Check if

$$\left|\frac{\partial f}{\partial x_j}\right| \leq \epsilon \qquad \text{for all } j = 1, 2, \ldots, n.$$

If so, stop with the current $\mathbf{x}'$ as the desired approximation of an optimal solution $\mathbf{x}^*$. Otherwise, perform another iteration.

Steepest Descent Method: example

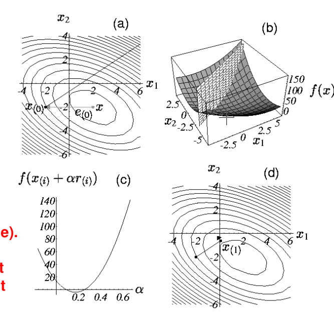
$$\begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ -8 \end{bmatrix}$$

a) Starting at (-2,-2) take the direction of steepest descent of f

b) Find the point on the intersection of these two surfaces that minimizes f

c) Intersection of surfaces (a plane).

d) The gradient at the bottommost point is orthogonal to the gradient of the previous step



A *line search* is a procedure that chooses α to minimize f along a line. Figure 6(b) illustrates this task: we are restricted to choosing a point on the intersection of the vertical plane and the paraboloid. Figure 6(c) is the parabola defined by the intersection of these surfaces. What is the value of α at the base of the parabola?

From basic calculus, α minimizes f when the *directional derivative* $\frac{d}{d\alpha}f(x_{(1)})$ is equal to zero. By the chain rule, $\frac{d}{d\alpha}f(x_{(1)}) = f'(x_{(1)})^T\frac{d}{d\alpha}x_{(1)} = f'(x_{(1)})^T r_{(0)}$. Setting this expression to zero, we find that α should be chosen so that $r_{(0)}$ and $f'(x_{(1)})$ are orthogonal (see Figure 6(d)).

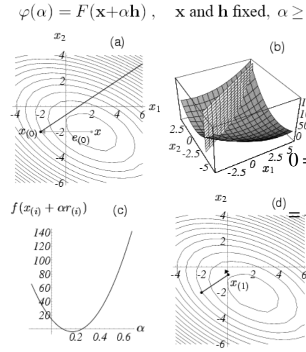
Line search: Find Minimum

$\varphi(\alpha) = F(\mathbf{x}+\alpha\mathbf{h})$, x and h fixed, $\alpha \geq 0$

Find α so that

$$\varphi(\alpha) = \mathbf{F}(\mathbf{x}_0 + \alpha \cdot \nabla f(\mathbf{x}_0))$$

is minumum



Use chain rule df/du*du/dx

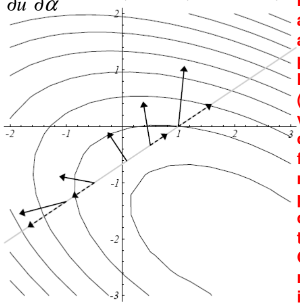
$$0 = \frac{\partial \varphi(\alpha)}{\partial \alpha} = \frac{\partial \mathbf{F}(\mathbf{x}_0 + \alpha \cdot \nabla f(\mathbf{x}_0))}{\partial \alpha}$$

$$= \frac{\partial \mathbf{F}}{\partial u}\frac{\partial u}{\partial \alpha} = \nabla f(\mathbf{x}_0)^{\mathrm{T}}\mathbf{F}'(\mathbf{x}_0 + \alpha \cdot \nabla f(\mathbf{x}$$

$$0=\frac{\partial\varphi(\alpha)}{\partial\alpha}=\frac{\partial \mathbf{F}(\mathbf{x}_0+\alpha\cdot\nabla f(\mathbf{x}_0))}{\partial\alpha}$$

Line search

$$=\frac{\partial \mathbf{F}}{\partial u}\frac{\partial u}{\partial \alpha}=\nabla f(\mathbf{x}_0)^{\mathrm{T}}\mathbf{F}'(\mathbf{x}_0+\alpha\cdot\nabla f(\mathbf{x}_0))$$



The gradient candidate $\nabla f(x_0)^i$ is shown at several locations along the search line (solid arrows). Each gradient's projection onto the line [in our line search] is also shown (dotted arrows). The gradient vectors represent the direction of steepest increase of f (our function that is being minimized), and the projections represent the rate of increase as one traverses the search line. On the search line, f is minimized where the gradient is orthogonal to the search line.

[Duda and Hart Stork page 226]

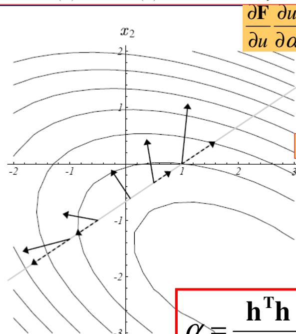
Notes on previous two slides

- There is an intuitive reason why we should expect these vectors to be orthogonal at the minimum.
- Figure on previous slide shows the gradient vectors at various points along the search line. The slope of the parabola (Figure (c) of the second previous slide) at any point is equal to the magnitude of the projection of the gradient onto the line (Figure on previous slide,dotted arrows). These projections represent the rate of increase of f as one traverses the search line.
- f is minimized where the projection is zero—where the gradient is orthogonal to the search line.

$$F(\mathbf{x}+\alpha\mathbf{h})=F(\mathbf{x})+\alpha\mathbf{h}^{\mathrm{T}}\mathbf{F}'(\mathbf{x})+O(\alpha^2)$$
$$\simeq F(\mathbf{x})+\alpha\mathbf{h}^{\mathrm{T}}\mathbf{F}'(\mathbf{x})\quad\text{for }\alpha\text{ sufficiently small.}$$

Line search

$$\frac{\partial \mathbf{F}}{\partial u}\frac{\partial u}{\partial \alpha}=\nabla f(\mathbf{x}_0)^{\mathrm{T}}\mathbf{F}'(\mathbf{x}_0+\alpha\cdot\nabla f(\mathbf{x}_0))$$



$$\text{Let } \mathbf{h}=\nabla f(\mathbf{x}_0)$$

$$\mathbf{h}^{\mathrm{T}}\mathbf{F}'(\mathbf{x}_0+\alpha\mathbf{h})=0$$

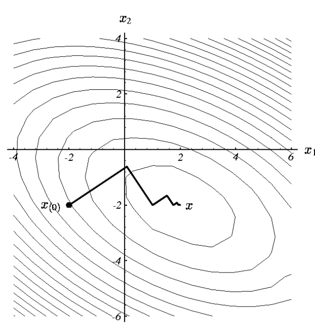
$$\text{since } \mathbf{F}'(\mathbf{x}_0+\alpha\mathbf{h})=\mathbf{F}'(\mathbf{x}_0)+\alpha\mathbf{F}''(\mathbf{x}_0)^{\mathrm{T}}\mathbf{h}$$

$$\mathbf{h}^{\mathrm{T}}\mathbf{F}'(\mathbf{x}_0+\alpha\mathbf{h})$$
$$=\mathbf{h}^{\mathrm{T}}\left(\mathbf{F}'(\mathbf{x}_0)+\alpha\mathbf{F}''(\mathbf{x}_0)^{\mathrm{T}}\mathbf{h}\right)$$
$$=-\mathbf{h}^{\mathrm{T}}\mathbf{h}+\alpha\mathbf{h}^{\mathrm{T}}\mathbf{H}\mathbf{h}=0$$

$$\boxed{\alpha=\frac{\mathbf{h}^{\mathrm{T}}\mathbf{h}}{\mathbf{h}^{\mathrm{T}}\mathbf{H}\mathbf{h}}=\frac{\nabla f(x_0)^{\mathrm{T}}\nabla f(x_0{}^{\mathrm{T}})}{\nabla f(x_0)^{\mathrm{T}}H\nabla f(x_0{}^{\mathrm{T}})}}$$

[Duda and Hart Stork page 226]

Iterations of Steepest Descent Method



Gradient Descent Example

Example. Consider the following two-variable problem:

Maximize $\quad f(\mathbf{x})=2x_1x_2+2x_2-x_1^2-2x_2^2.$

Thus,

$$\frac{\partial f}{\partial x_1}=2x_2-2x_1,$$
$$\frac{\partial f}{\partial x_2}=2x_1+2-4x_2.$$

2(0) – 2(0)=0
2(0) + 2 – 2(0)=2
∇f^1 = (0,2)

We also can verify (see Appendix 2) that $f(\mathbf{x})$ is concave. To begin the gradient search procedure, suppose that $\mathbf{x}=(0,0)$ is selected as the initial trial solution. Because the respective partial derivatives are 0 and 2 at this point, the gradient is

$\nabla f(0,0)=(0,2).$ ∇f^2 X^1=(0,0)

Therefore, to begin the first iteration, set

$$x_1=0+t(0)=0,$$
$$x_2=0+t(2)=2t,$$

and then substitute these expressions into $f(\mathbf{x})$ to obtain

$$\begin{aligned}f(\mathbf{x}'+t\,\nabla f(\mathbf{x}'))&=f(0,2t)\\&=2(0)(2t)+2(2t)-0^2-2(2t)^2\\&=4t-8t^2.\end{aligned}$$

Linesearch

Because

$$f(0,2t^*)=\max_{t\ge0} f(0,2t)=\max_{t\ge0}\{4t-8t^2\}$$

and

$$\frac{d}{dt}(4t-8t^2)=4-16t=0,$$

it follows that

$$t^*=\frac{1}{4},$$

so

Reset $\quad \mathbf{x}'=(0,0)+\frac{1}{4}(0,2)=\left(0,\frac{1}{2}\right).$ x^2

For this new trial solution, the gradient is

$\nabla f\left(0,\frac{1}{2}\right)=(1,0).$ ∇f^2 Gradient a x^2

Thus, for the second iteration, set

$$\mathbf{x}=\left(0,\frac{1}{2}\right)+t(1,0)=\left(t,\frac{1}{2}\right),$$

so

$$\begin{aligned}f(\mathbf{x}'+t\,\nabla f(\mathbf{x}'))&=f\left(0+t,\frac{1}{2}+0t\right)=f\left(t,\frac{1}{2}\right)\\&=(2t)\left(\frac{1}{2}\right)+2\left(\frac{1}{2}\right)-t^2-2\left(\frac{1}{2}\right.\\&=t-t^2+\frac{1}{2}.\end{aligned}$$

Because

$$f\left(t^*,\frac{1}{2}\right)=\max_{t\ge0} f\left(t,\frac{1}{2}\right)=\max_{t\ge0}\left\{t-t^2+\frac{1}{2}\right\}$$

and

$$\frac{d}{dt}\left(t-t^2+\frac{1}{2}\right)=1-2t=0,$$

then

$$t^*=\frac{1}{2},$$

so

Reset $\quad \mathbf{x}'=\left(0,\frac{1}{2}\right)+\frac{1}{2}(1,0)=\left(\frac{1}{2},\frac{1}{2}\right).$ x

Example Contd.

A nice way of organizing this work is to write out a table such as Table 13.2 which summarizes the preceding two iterations. At each iteration, the second column shows the current trial solution, and the rightmost column shows the eventual new trial solution, which then is carried down into the second column for the next iteration. The fourth column gives the expressions for the x_j in terms of t that need to be substituted into $f(\mathbf{x})$ to give the fifth column.

By continuing in this fashion, the subsequent trial solutions would be $(\frac{1}{2},\frac{3}{4})$, $(\frac{3}{4},\frac{3}{4})$, $(\frac{3}{4},\frac{7}{8})$, $(\frac{7}{8},\frac{7}{8})$, . . . , as shown in Fig. 13.14. Because these points are converging to $\mathbf{x}^*=(1,1)$, this solution is the optimal solution, as verified by the fact that

$$\nabla f(1,1)=(0,0).$$

TABLE 13.2 Application of the gradient search procedure to the example

Iteration	$\mathbf{x}'$	$\nabla f(\mathbf{x}')$	$\mathbf{x}'+t\,\nabla f(\mathbf{x}')$	$f(\mathbf{x}'+t\,\nabla f(\mathbf{x}'))$	t^*	$\mathbf{x}'+t^*\,\nabla f(\mathbf{x}')$
1	(0, 0)	(0, 2)	(0, 2t)	$4t-8t^2$	$\frac{1}{4}$	$\left(0,\frac{1}{2}\right)$
2	$\left(0,\frac{1}{2}\right)$	(1, 0)	$\left(t,\frac{1}{2}\right)$	$t-t^2+\frac{1}{2}$	$\frac{1}{2}$	$\left(\frac{1}{2},\frac{1}{2}\right)$

However, because this converging sequence of trial solutions never reaches its limit, the procedure actually will stop somewhere (depending on ϵ) slightly below (1, 1) as its final approximation of $\mathbf{x}^*$.

As Fig. 13.14 suggests, the gradient search procedure zigzags to the optimal solution rather than moving in a straight line. Some modifications of the procedure have been developed that *accelerate* movement toward the optimal solution by taking this zigzag behavior into account.

If $f(\mathbf{x})$ were *not* a concave function, the gradient search procedure still would converge to a *local* maximum. The only change in the description of the procedure for this case is that t^* now would correspond to the *first local maximum* of $f(\mathbf{x}'+t\,\nabla f(\mathbf{x}'))$ as t is increased from 0.

If the objective were to *minimize* $f(\mathbf{x})$ instead, one change in the procedure would be to move in the *opposite* direction of the gradient at each iteration. In other words, the rule for obtaining the next point would be

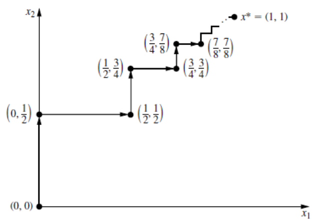
Reset $\quad \mathbf{x}'=\mathbf{x}'-t^*\,\nabla f(\mathbf{x}').$

The only other change is that t^* now would be the nonnegative value of t that *minimizes* $f(\mathbf{x}'-t\,\nabla f(\mathbf{x}'))$; that is,

$$f(\mathbf{x}'-t^*\,\nabla f(\mathbf{x}'))=\min_{t\ge0} f(\mathbf{x}'-t\,\nabla f(\mathbf{x}')).$$

Example Continued

FIGURE 13.14
Illustration of the gradient search procedure when $f(x_1, x_2) = 2x_1x_2 + 2x_2 - x_1^2 - 2x_2^2$.



Method of steepest ascent

- **Start at a point *x***
- **Follow the direction of steepest ascent:**

$$d = \nabla f(x)$$

- **Move to the best point in the direction of steepest ascent**
- **Stop as soon as**
  - This point is approximately *stationary*

$$\|\nabla f(x)\| < \varepsilon$$

Exercise (not required)

- **Perform iteration 3 and 4 of the of the above example; show the worked out details by hand (follow the style of the example present here)**
- **Plot the isolines plot and the vector plot (quiver2)**
- **Overlay the path for steps 1, 2, 3, and 4 on the isoline-vector plot**
- **Plot the surface plot (3D) also**

Unconstrained optimization

For noncovex/non-concave

$$\max/\min \quad f(x) = f(x_1, \ldots, x_n)$$

subject to

$$x \in R^n$$

- **Assume f'(x) and *H(x)* exists for all $x \in S$**
- **Locate candidate extrema using f'(x) = 0 and boundary points**
- **Then candidate x is**
  - *f(x)* is a convex function on *S* if and only if all principal minors of *H(x)* are nonnegative for all $x \in S$
  - *f(x)* is a concave function on *S* if and only if the principal minors of *H(x)* of order *k* have the same sign as $(-1)^k$ for all $x \in S$ and all *k*

Unconstrained optimization

- **A point *x* where $\nabla f(x) = 0$ is called a *stationary point* of *f***
- **Let *x** be a stationary point, i.e., $\nabla f(x^*) = 0$**
  - If all leading principal minors of $H(x^*)$ are positive then x^* is a *local minimum*
  - If the leading principal minors of $H(x^*)$ of order *k* has the same sign as $(-1)^k$ (for all *k*) then x^* is a *local maximum*

Issues in Gradient Descent

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

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

$$x_{(i+1)} = x_{(i)} - a_{(i)} p_{(i)}$$

$p_{(i)}$ Adjustment Direction

$a_{(i)}$ Step Size

- **How large should I step in the positive gradient direction (gradient ascent) or in the negative gradient direction (gradient descent)**

Let W = (0,0,….)
Repeat
For j in 0..n #each variable

$$W_{j,t+1} = W_{j,t} - \alpha * \nabla J_{w_j}(W_t)$$

$$W_{j,t+1} = W_{j,t} + \alpha * \sum_{i=1}^{m} (y^i - WX^i) X_j^i$$

until convergence (i.e., no big changes in W or erro

Non-stationary Iterative Method

- Start from initial guess x0, adjust it until close enough to the exact solution

$$x_{(i+1)} = x_{(i)} + a_{(i)} p_{(i)} \quad \text{i=0,1,2,3,......}$$

$p_{(i)}$ Adjustment Direction

$a_{(i)}$ Step Size

- How to choose direction and step size?

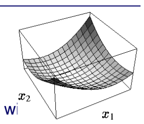
Closed Form versus Iterative Procs

- Linear least squares problems are convex and have a closed-form solution that is unique, except in special degenerate situations.
- In contrast, non-linear least squares problems generally must be solved by an iterative procedure, and often are non-convex with multiple local solutions.

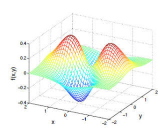
Lecture Outline

- R
- Lines, Tangents, Taylors Theorem
- Turning points, Roots, 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
- Predicting Click Through Rates
  - Linear Regression
  - Logistic Regression

General Approach to Finding Extrema



- Well-behaved version spaces
  - Convex or concave function (±definiteness)
  - Algorithms seek a local extrema knowing that it wi
    - If f() is a concave function then local maximum is a global maximum
    - If f() is a convex function then local minimum is a global minimum
    - Newton-Raphson, Gradient Descent, Conjugate Gradient Descent



- Otherwise
  - We resort to local approximations
    - Hill-Climbing
    - Simulated annealing
    - Commonly used in Neural Networks

Lecture Outline

- R
- Lines, Tangents, Taylors Theorem
- Turning points, Roots, 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
- Predicting Click Through Rates
  - Linear Regression
  - Logistic Regression

Machine Learning in one slide

- Machine learning, a branch of artificial intelligence, is a scientific discipline that is concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data, such as from sensor data or databases.
- A learner can take advantage of examples (data) to capture characteristics of interest of their unknown underlying probability distribution. Data can be seen as examples that illustrate relations between observed variables.
- A major focus of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on data; the difficulty lies in the fact that the set of all possible behaviors given all possible inputs is too large to be covered by the set of observed examples (training data).
- Hence the learner must generalize from the given examples, so as to be able to produce a useful output in new cases. Machine learning, like all subjects in artificial intelligence, require cross-disciplinary proficiency in several areas, such as probability theory, statistics, pattern recognition, cognitive science, data mining, adaptive control, computational neuroscience and theoretical computer science.

[WikiPedia]

What is the Learning Problem?

Learning = Improving with experience at some task

- **Improve over Task T**
- **with respect to performance measure P**
- **based on experience E**

Types of Learning

- Supervised learning - **Generates a function that maps inputs to desired outputs. For example, in a** classification **problem, the learner approximates a function mapping a vector into classes by looking at input-output examples of the function.**
- Unsupervised learning - **Models a set of inputs: like clustering**
- Semi-supervised learning - **Combines both labeled and unlabeled examples to generate an appropriate function or classifier.**
- Reinforcement learning - **Learns how to act given an observation of the world. Every action has some impact in the environment, and the environment provides feedback in the form of rewards that guides the learning algorithm.**
  Transduction - **Tries to predict new outputs based on training inputs, training outputs, and test inputs.**

Supervised Learning :Regression

- **Regression**
  - Linear Regression
- **Classification**
  - Logistic Regression
- **Generalized Linear Models (GLMs)**
  - Broader family of models (that subsume Linear Regression and logistic regress and more
  - In R checkout ?glm()

Parametric Approaches vs. Non-parametric
Convex/Concave
Discriminative versus generative

Terminology: linear regression

W_i are the model coefficients

$$y = W_0 + W_1X_1 + W_2X_2 + \ldots + W_nX_n$$

y
Predicted
Response variable
Outcome variable
Dependent

X_i's
Predictor variables
Explanatory variables
Covariables
Independent variables

Y-intercept/threshold

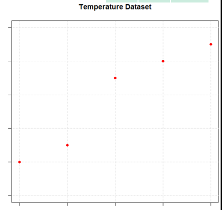
Pr(Click): Advertising Problem

- **Predict Pr(Click|dwellTimeOnWebpage)**
  - at the times 1, 2, 3, 4, and 5 seconds after loading the page.
- **Graph each data point with time on the *x*-axis and CTR on the *y*-axis. Your data should follow a straight line.**
- **Use locator() to input data**
- **Find the equation of this line.**

#	x	y%
1	1	2
.	2	3
.	3	7
.	4	8
m	5	9

Temperature Dataset

F(x)



X

X are features, aka variables, continuous, discrete, ordinal (X ε ℜ^n)

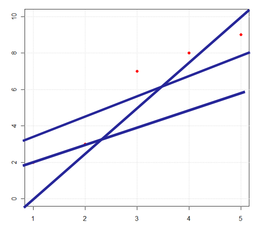
Generate Your Own Data

```
# You can generate data by clicking on a plot.
# Create data that illustrates the effect of varying 'f' and 'iter' in 'lowess'.
example.generateYourOwnData = function(){

 #change range of X and Y and experiment
 plot( c(0,1), c(0,1), type = 'n')
 xy <- locator(type = "p")     # create your own data set by clicking on the
                               # left mouse button, then with the right mouse button
                               # to finish. With a Macintosh cntrl-click outside the
                               # plot to finish.
data1=data.frame(x=xy$x, y=xy$y)  #PLOT LINE
abline(coef(lm(y~x, data=data1)), col="red")

lines(lowess(xy, f = 2/3, iter = 3))  # here I've used the defaults
                                      # for f and iter,
                                      # experiment with other values
```

Least Square Fit Approximations

Suppose we want to fit the data set.

#	x	y
1	1	2
.	2	3
.	3	7
.	4	8
m	5	9

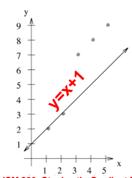


We would like to find the best straight line to fit the data?

Fit a line based on…

- If we assume that the first two points are correct and choose the line that goes through them, we get the line $y = 1 + x$.
- If we substitute our points (x-values) into this equation, we get the following chart.
- How good is this line?
  - The sum of the squares of the errors is 27.

$$y = mx + b$$
$$m = \frac{y_2 - y_1}{x_2 - x_1}$$



x	y	predicted y	error	$(\text{error})^2$
1	2	2	0	0
2	3	3	0	0
3	7	4	3	9
4	8	5	3	9
5	9	6	3	9

SSE = 27

Do you think that we can do better than this?

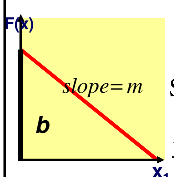
Linear Model More Generally

- E.g., y=mx+b can be more generally seen a function of the form
- Here the W's are the parameters (also called weights) parametering the space of linear function mapping from X → Y=F(x)

#	X0	x1	y
1	1	1	2
.	1	2	3
.	1	3	7
.	1	4	8
m	1	5	9

$$y = f(x_0, x_1) = w_0 x_0 + w_1 x_1$$

$$= \sum_{i=1}^{n} w_i x_i = W^T X$$



Sometimes use θ instead of W

$$y = f(x_0, x_1) = \sum_{i=1}^{n} \theta_i x_i = \theta^T X$$

Linear Model: Ordinary Least Squares

Measuring Quality

- How do we pick, or learn, the parameters W (aka θ)?
- One reasonable method seems to be to make f(x) close to y, at least for the training examples.
- To formalize, let's define a function that measures, for each possible model/hypothesis, W, how close $f_\theta(x^i)$'s are to the corresponding y^i's:

$$J(W) = \sum_{i=1}^{m} \left| WX^i - y^i \right|$$

This error minimization is going to have problems?

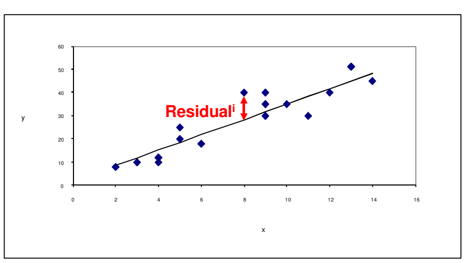
$$J(W) = \tfrac{1}{2} \sum_{i=1}^{m} \left(WX^i - y^i \right)^2$$

Residual sum of squares

- Sum of squared error
- AKA Residual Sum of Squares (Residual squared)

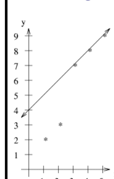
Residual

$$\text{Residual}^i = \left(WX^i - y^i \right)$$



Which Line is it anyway?

- Select another two points and build a line
- If we choose the line that goes through the points when $x = 3$ and 4, we get the line $y = 4 + x$. Will we get a better fit? Let's look at it.

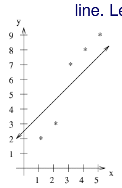


x	y	predicted y	error	$(\text{error})^2$
1	2	5	−3	9
2	3	6	−3	9
3	7	7	0	0
4	8	8	0	0
5	9	9	0	0

SSE = 18. Getting better but can we do better?

Can we do better than guesswork?

- **Let's try the line that is half way between these two lines. The equation would be $y = 2.5 + x$.**
- **Is there a more scientific or efficient way than guessing at which line would give the best fit.**
  - Surely there is a methodical way to determine the best fit line. Let's think about what we want.



x	y	predicted y	error	$(\text{error})^2$
1	2	3.5	−1.5	2.25
2	3	4.5	−1.5	2.25
3	7	5.5	1.5	2.25
4	8	6.5	1.5	2.25
5	9	7.5	1.5	2.25

SSE = 11.25. Getting better but can we do better?

Hypothesis Space of Linear Models

- **Here the W's are the parameters (also called weights) parameterizing the space of linear function mapping from X → Y = f(X)**
- **Augment Training Data with dummy intercept variable (simplifies notation and modeling)**

#	X0	x1	y
1	1	1	2
.	1	2	3
.	1	3	7
.	1	4	8
m	1	5	9

$$y = f(x_0, x_1) = w_0 x_0 + w_1 x_1$$

$$= \sum_{i=1}^{n} w_i x_i = W^T X$$

Sometimes use θ instead of W

$$y = f(x_0, x_1) = \sum_{i=1}^{n} \theta_i x_i = \theta^T X$$

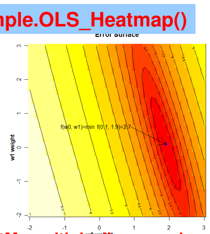
Space of Hypotheses: Weights

- Each model is in our case a coefficient for the y-intercept (bias) and a coefficient for the feature-variable (time)
- Plot weight-space in 2D where the third dimesion is the error

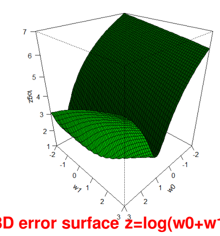
$$J(W) = \tfrac{1}{2} \sum_{i=1}^{m} \left(W^i X^i - y^i\right)^2$$

- Select combination that minimizes the sum of square error

example.OLS_Heatmap()



HeatMap with isolines overlayed



3D error surface z=log(w0+w1∗x)

Find a line that fits all datapoints?

- **Recall, a line in slope-intercept form looks like y= $w_0 + w_1 x$ where w_0 is the y-intercept and w_1 is the slope.**
- **We want to find w_0 and w_1 such that $w_0 + w_1 x_i = y_i$ is true for all our data points:**

$$\begin{aligned} w_0 + 1w_1 &= 2 \\ w_0 + 2w_1 &= 3 \\ w_0 + 3w_1 &= 7 \\ w_0 + 4w_1 &= 8 \\ w_0 + 5w_1 &= 9 \end{aligned}$$

- **We know that there <u>may not exist</u> w_0 and w_1 that fit all these equations, so we try to find the best fit.**

Find the best line: Several Approaches

- **Determine $w_0, w_1 \ldots w_n$**

$$y = f(x_0, x_1) = w_0 x_0 + w_1 x_1$$

$$= \sum_{i=1}^{n} w_i x_i = W^T X$$

- **Several Approaches to finding the best-fit line**
  - Select a couple of data points and solve analytically (high variance)
  - Brute-force Search
  - Iterative approaches (via the gradient)
  - Closed Form
  - Probabilistic interpretation/justification via maximum likelihood
  - Bayesian modeling [will be covered in Lecture 4]

Hypothesis Space of Linear Models

- **Here the W's are the parameters (also called weights) parameterizing the space of linear function mapping from X → YF(x)**
- **Augment Training Data with dummy intercept variable (simplifies notation and modeling)**

#	X0	x1	y
1	1	1	2
.	1	2	3
.	1	3	7
.	1	4	8
m	1	5	9

$$y = f(x_0, x_1) = w_0 x_0 + w_1 x_1$$

$$= \sum_{i=1}^{n} w_i x_i = W^T X$$

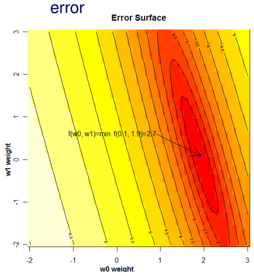
Sometimes use θ instead of W

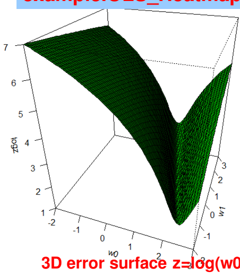
$$y = f(x_0, x_1) = \sum_{i=1}^{n} \theta_i x_i = \theta^T X$$

Brute Force Search of Weights

- Enumerate all possible coefficient combinations (in our case coefficient for the y-intercept (bias) and for the c-variable (time)
- Select the weight combination that minimizes the sum of square error

example.OLS_Heatmap()



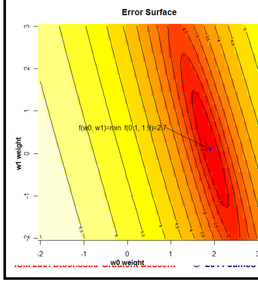


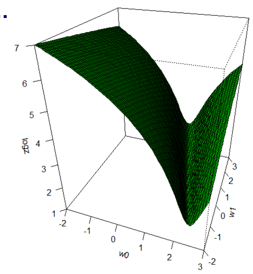
HeatMap with isolines overlayed

3D error surface z=log(w0+w1∗x)

Brute Force Search of Weights

- **Very inefficient; at best we can only approximate the surface**
- **Not scaleable**
- **Avoid this approach…**





Iterative approach to Learning the Line

- **Can we navigate the error surface in an efficient manner in the hope of getting to minimum?**
- **Can we leverage other properties of the function? (Hint convexity)**
- **Yes we can!**
  - We can navigate this surface using the gradient (slope)
  - OLS is convex so what [well-behaved function! More about this later this lecture and next lecture]

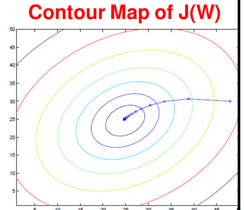
Exploiting the Gradient of Error Surface

- **Gradient Descent (a simpler alternative to Newton-Raphson)**
  - A work horse
- **Newton-Raphson**
  - Quasi-versions
  - Commonly used
- **Conjugate-Gradient Descent**
  - Not covered here but effective and commonly used
- **Practically speaking we will use off-the-shelf software**
  - R built-in solvers such as optim()
  - Or built-in linear regression algorithms, glm(), lm()) etc.

Gradient Descent: surf downhill

- **Goal: Choose W so as to minimize J(W)** $J(W)=\frac{1}{2}\sum_{i=1}^{m}\left(W^iX^i-y^i\right)^2$
- **Algorithm**
  - Start with some random guess for W
  - Repeat
    - Use gradient to travel downhill
      - Update each weight w_i
  - Until convergence (to global minimum)

Contour Map of J(W)



Let $W=(0,0,....)$

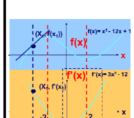
Repeat

$$W_{i,t+1}=W_{i,t}-\alpha*\nabla J(W_{i,t})$$

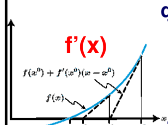
until convergence (i.e., no big changes in W or error)

Gradient Descent (a simpler root finder)

$$x^{i+1}=x^i-\frac{f'(x^i)}{f''(x^i)}$$ *Iteration function* **Newton-Raphson In 1-Dimension**



- **Calculating f''(x), the Hessian H, and inverting it is complex so simpler algorithms have been developed such gradient descent**



$$x^{i+1}=x^i-a^if'(x^i)$$ **Gradient Descent**

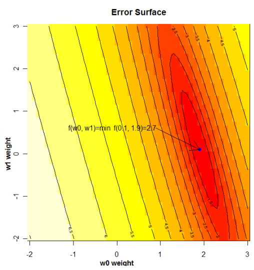
- **How large should I step in the positive gradient direction (gradient ascent)**
  - or in the negative gradient direction (gradient descent)

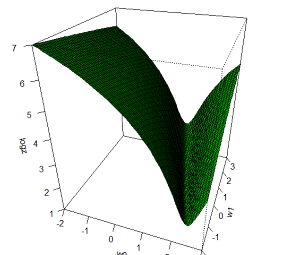
Build Error Surface

R code example: JimisMLCourse.R

- **R code**

Difficult surface to navigate!





HeatMap with isolines overlayed

3D error surface z=log(w0+w1*x)

OLS Via Gradient Descent: The Gradient

$$W_{j,t+1} = W_{j,t} - \alpha * \nabla J_{w_j}(W_{j,t})$$

- In order to implement this algorithm, we have to work out what is the partial derivative term at time t on the right hand side $\nabla f_{wj}(W)=dF(W)/dw_j$.
- Assume we have only one training example (x, y), so that we can drop the sum in the definition of J.

$$\begin{aligned}\nabla J_{W_j}(W) = \frac{\partial}{\partial W_j} J(W) &= \frac{\partial}{\partial W_j}\left(\frac{1}{2}\left(f_W(x)-y\right)^2\right) \\ &= 2*\frac{1}{2}\left(f_W(x)-y\right)\frac{\partial}{\partial W_j}\left(f_W(x)-y\right) \\ &= \left(f_W(x)-y\right)\frac{\partial}{\partial W_j}\left(\left(\sum_{i=0}^{n} w_i x_i\right)-y\right) \\ &= \left(f_W(x)-y\right)x_j\end{aligned}$$

Use chain rule df/du*du/dx

Assume a single training example

For a single w_j

Recall

$$\frac{\partial}{\partial W_j}\left(\left(\sum_{i=0}^{n} w_i x_i\right)-y\right) = \frac{\partial}{\partial W_j}\left(w_0 x_0 + w_1 x_1 \ldots + w_j x_j + \ldots w_n x_n\right) = 0+0+\ldots x_j + \ldots 0$$

OLS Via Gradient Descent

$$\nabla f_{w_j}(W_{j,t}) = \frac{\partial}{\partial W_j} J(W) = \left(f_W(x)-y\right)x_j$$

Widrow-Hoff Learning Rule

- **Assume we have only one training example (x, y), so that we can drop the sum in the definition of J.**

$$W_{j,t+1} = W_{j,t} - \alpha * \nabla f_{w_j}(W_{j,t})$$

$$W_{j,t+1} = W_{j,t} - \alpha * (y - f_{W_t}(x))x_j$$

$$W_{j,t+1} = W_{j,t} + \alpha * (f_{W_t}(x) - y)x_j$$

Least Mean Squares Rule AKA Widrow-Hoff Learning Rule

- **This rule has intuitive properties**
  - The magnitude of the update is proportional to the error term *(y^i – $f(x^i)$);*
  - If we are encountering a training example on which our prediction nearly matches the actual value of y^i, then we find that there is little need to change the parameters;
  - In contrast, a larger change to the parameters will be made if our prediction $f_W(x^i)$ has a large error (i.e., if it is very far from y^i).

LMS Rule for Multiple Examples

BATCH Update Rule

- **General Update Rule for m training examples**
  - W(k+1)=W(k) - α∇J (W(k))

OLS Objective Function

$$J(W, X_1^L) = \sum_{i=1}^{m}(W^T X_i - y_i)^2$$

True gradient is used to update the parameters of the model, corresponding to the sum of the gradients caused by each training example (one sweep)

$$\nabla J = \frac{\partial(J_P(W, X_1^m))}{\partial W} = \sum_{i=1}^{m}(W^T X_i - y_i)X_i$$

Gradient of OLS Objective Func

$$W(k+1) = W(k) + \alpha(k)\sum_{i=1}^{m}(y_i - WX_i)X_i$$

OLS BATCH Update Rule

Intuitively, drag weight vector closer to the misclassified examples

OLS using Gradient Descent (LMS Rule)

- **Batch update**

$\nabla J_{wj}(W_t)$
Partial derivate WRT to variable w_j of error function J(W) at point W_t

$$Let\ \ W = (0,0,\ldots.)$$

Repeat

For j in 0..n #each variable

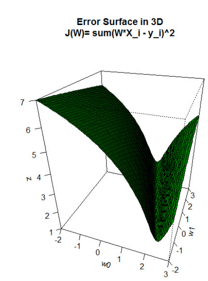
$$W_{j,t+1} = W_{j,t} - \alpha * \nabla J_{w_j}(W_t)$$

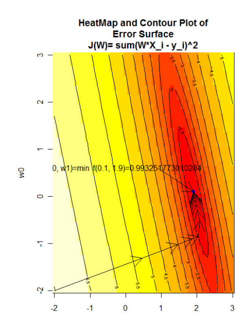
$$W_{j,t+1} = W_{j,t} + \alpha * \underbrace{\sum_{i=1}^{m}(y^i - WX^i)X_j^i}_{\text{Scalar}}$$

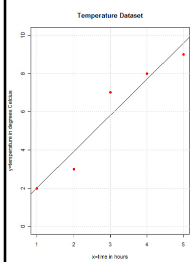
$(y^i - WX^i)^T X_j^i$

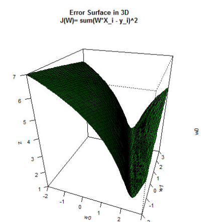
Scalar

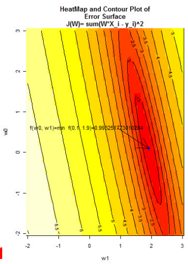
until convergence (i.e., no big changes in W or error)











$coefficients
[1] 0.09644596 1.90098441
$iterations
[1] 658
$SSE
[1] 2.700011

OLS Via Gradient Descent in R

```
OLSUsingGradientDescent = function (..){
.....
  w=rep(-2, numVariables);  #initialize weight vector
  wOld = w;
  it = 1 # iteration index
  while (it <= max.iter){
    p = designMatrix %*% w #prediction for each training example
    w = w + alpha* drop(t(designMatrix) %*% (targetValues - p))
     #drop yields a scalar from errV*X_j

    if (t(w - wOld) %*% (w - wOld) < tol)
      break
    it = it + 1  # ir
    wOld=w
    if (it > max.ite
  }
}
```


Ordinary Least Squares Algorithm

Single-sample Primal For

- **Given Training data S where each example *i* is of the form $(x_{i,1},\ldots, x_{i,n}, y_i)$, and a learning rate η**
- **Set W_o to zeros; k=0;**
- **Repeat**
  - For i = 1 to */Train/* do
    $W_{k+1}= W_k+ \eta\ (<W_k, X_i> - y_i)X_i$
  - End-For
- **Until convergence**
- **Return *W***

Iterative, gradient descent based algorithm (as opposed to other versions, such as closed form version, quadratic programming version, maximum likelihood. What could they look like?)

R: Linear Regression via lm()

example.lm ()

Pay attention to
1. **Residual standard error**
2. **Or Deviance (SSE),**
3. **And variable significance**

```
Output Window
> colnames(dataEx1)=c("time", "temperature")
>  lm.temp <- lm(temperature ~ time, data=as.
>  summary(lm.temp)

Call:
lm(formula = temperature ~ time, data = as.data.frame(dataEx1))

Residuals:
         1          2          3          4          5
-1.490e-16 -9.000e-01  1.200e+00  3.000e-01 -6.000e-01

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    0.100      0.995   0.101  0.92628
time           1.900      0.300   6.333  0.00796 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9487 on 3 degrees of freedom
Multiple R-squared: 0.9304, Adjusted R-squared: 0.9072
F-statistic: 40.11 on 1 and 3 DF,  p-value: 0.00796
> deviance(lm.temp)
[1] 2.7
```

Residuals = $(WX^i - y^i)$

Variable significance

Residual standard error

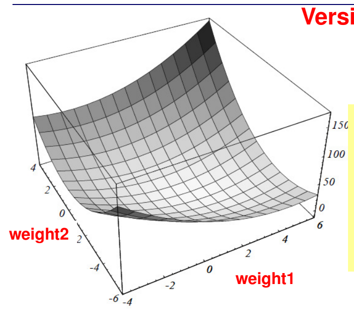
Residual standard error =σ=Sqrt(deviance/m-n-1)

$$\sigma = \sqrt{\left(\frac{1}{m-n-1}\right)\sum_{i=1}^{m}\left(WX^i - y^i\right)^2} = \sqrt{1/3*2.7}$$

residualStandardError=sqrt((t(lm.temp$residuals) %*% lm.temp$residuals)/3)

Gradient Descent for Ordinary Least Squares

Version space (weights)



Error surface; each point corresponds to a different linear model (hypothesis). The vertical axis indicates the squared error for the training dataset WRT that weight vector.
Q: Will this surface change for different datasets?

OLS with this objective has no local minima (convex as the Hessian, n by n matrix of second derivatives, of the objective function is positive definite); in this case n=2 variables.
Iterative versus closed form solution

OLS using Gradient Descent (LMS Rule)

Stochastic Gradient Descent

- **Stochastic update**

$Let\ \ W = (0,0,....)$

$\nabla J_{wj}(W_t)$
Partial derivate WRT to variable of error function J(W) at point W

Repeat

For j in 0..n #each variable

For i in 1..m #each example

$$W_{j,t+1} = W_{j,t} + \alpha * (y^i - WX^i)X^i_j$$

until convergence (i.e., no big changes in W or error)

OLS using Gradient Descent

Stochastic Gradient Descent Online/Single Update Rule

- General Update Rule
  - W(k+1)=W(k) - η∇J (W(k))

$$J_q(W, X_1^L) = \sum_{i=1}^{m} (W^T X_i - y_i)^2$$

OLS Objective Function

True gradient is approximated the gradient of the cost function only evaluated at one example; adjust parameters proportional to this approx. gradient. This can be much better for large datasets.
E.g., Stochastic Gradient Decision Trees; perceptron

$$W(k+1) = W(k) + \eta(k)(WX_i - y_i)X_i$$

OLS Single Update Rule

Intuitively, drag weight vector closer to the misclassified example

Stochastic Gradient Decent vs. Batch

- Stochastic gradient descent can start making progress right away, and continues to make progress with each example it looks at.
- Often, stochastic gradient descent gets W "close" to the minimum much faster than batch gradient descent.
  - Note however that it may never "converge" to the minimum, and the parameters W will keep oscillating around the minimum of J(W); but in practice most of the values near the minimum will be reasonably good approximations to the true minimum.
- For these reasons, particularly when the training set is large, stochastic gradient descent is often preferred over batch gradient descent.

Learning Rate: α

- Fixed learning rate
  - While it is more common to run stochastic gradient descent as we have described it and with a fixed learning rate
- Dynamic, decreasing learning rate
  - by slowly letting the learning rate decrease to zero as the algorithm runs, it is also possible to ensure that the parameters will converge to the global minimum rather then merely oscillate around the minimum
- Or it can be calculated

$$\alpha = \frac{\mathbf{h}^T\mathbf{h}}{\mathbf{h}^T\mathbf{H}\mathbf{h}} = \frac{\nabla f(x_0)^T \nabla f(x_0{}^T)}{\nabla f(x_0)^T H \nabla f(x_0{}^T)}$$

Exercise 1.5 : Code up OLS using LMS

- The learning objective function for weighted ordinary least squares (WOLS) is defined as follows:

$$JWGT(W) = \tfrac{1}{2}\sum_{i=1}^{m} wgt^i \left(W^i X^i - y^i\right)^2$$

- Derive the gradient for this weighted OLS by hand; showing each step and also explaining the step
- Train a weighted OLS model using gradient descent
  - Train OLS model using using LMS (Least Mean Squares) Rule algorithm to predict y (the CTR) given x (the dwelltime on a page)
  - Train a model using lm(.) (in R) using the same weights.
  - Train a model using lm() without the weights
- Analysis
  - Use the following dataset (sometimes known as the design matrix)[See next slide]
  - Plot the error surface
  - Plot the heatmap and contour plot
  - Plot the path to convergence
  - Comment on convergence and on the mean squared error using your algorithm and the lm(..);
  - Comment on the weighted linear and unweighted linear model.
- NOTE: See slides for inspiration

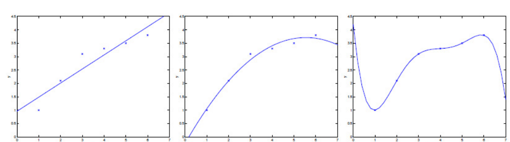
Exercise 1.5 : Continued

$$JWGT(W) = \tfrac{1}{2}\sum_{i=1}^{m} wgt^i \left(W^i X^i - y^i\right)^2$$

- Dataset

#	Weight	x	y
1	0.5	1	2
2	1	2	3
3	5	3	7
4	1	4	8
5	7	5	9

Overfiting versus Underfitting



Instead, if we had added an extra feature x^2, and fit $y = \theta_0 + \theta_1 x + \theta_2 x^2$, then we obtain a slightly better fit to the data. (See middle figure) Naively, it might seem that the more features we add, the better. However, there is also a danger in adding too many features: The rightmost figure is the result of fitting a 5-th order polynomial $y = \sum_{j=0}^{5} \theta_j x^j$. We see that even though the fitted curve passes through the data perfectly, we would not expect this to be a very good predictor of, say, housing prices (y) for different living areas (x). Without formally defining what these terms mean, we'll say the figure on the left shows an instance of **underfitting**—in which the data clearly shows structure not captured by the model—and the figure on the right is an example of **overfitting**.

[Ng, 2008 Stanford]

MultiVariate Linear Regression

R Console

```
> source("C:\\jimi\\Publications\\Conferences\\ESSIR-RUSSIR-2009\\Ancestry\\JimisMLCourse_2.R")
> example.lm()

Call:
lm(formula = temperature ~ time + age, data = as.data.frame(dataEx1))

Residuals:
       1        2        3        4        5
 0.27851 -0.63415  0.06065  0.66712 -0.37213

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.60667    2.14563   1.681   0.2348
time         1.40628    0.36466   3.856   0.0611 .
age         -0.12659    0.07235  -1.750   0.2223
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7304 on 2 degrees of freedom
Multiple R-squared: 0.9725,     Adjusted R-squared: 0.945
F-statistic: 35.37 on 2 and 2 DF,  p-value: 0.0275
```

#	time	age	y
1	1	25	2
.	2	22	3
.	3	7	7
.	4	22	8
m	5	10	9

Normal Equations → Closed From Soln. to OLS

- **Gradient descent gives one way of minimizing J(W).**
- **An alternative is to performing the minimization explicitly and without resorting to an iterative algorithm**
  - In this method, we will minimize J by explicitly taking its derivatives with respect to the W_j's, and setting them to zero.
  - Do this via calculus with matrices.

Closed form solution to OLS

- **To minimize J, we set its derivatives to zero, and obtain the normal equations:**
  - $X^TXW = X^Ty$

$$RSS = Variance \text{ of } \varepsilon$$

$$0 = \frac{\partial \sum \hat{\varepsilon}_i^2}{\partial W} = \frac{\partial (y_i - XW)^2}{\partial W} \qquad 0 = \frac{\partial \sum \hat{\varepsilon}_i^2}{\partial \hat{\beta}_1} = \frac{\partial \sum \left(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i\right)^2}{\partial \hat{\beta}_1}$$

$$= -2X\left(y - \hat{\beta}_0 - \hat{\beta}_1 x_i\right) \qquad = -2\sum x_i \left(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i\right)$$

$$= -2\sum x_i \left(y_i - \bar{y} + \hat{\beta}_1 \bar{x} - \hat{\beta}_1 x_i\right) = -2\sum x_i \left(y_i - \bar{y} + \hat{\beta}_1 \bar{x} - \hat{\beta}_1 x_i\right)$$

For another derivation see: http://www.stanford.edu/class/cs229/notes/cs229-notes1.pdf

Closed form solution to OLS

- **To minimize J, we set its derivatives to zero, and obtain the normal equations:**
  - $X^TXW = X^ty$
  - Thus the value of W that minimizes J(W) is give in closed form

$$\begin{aligned}
\nabla J_{W_j}(W) = \tfrac{\partial}{\partial W_j} J(W) &= \tfrac{\partial}{\partial W_j}\left(\tfrac{1}{2}\left(f_W(x) - y\right)^2\right) \\
&= 2 * \tfrac{1}{2}\left(f_W(x) - y\right)\tfrac{\partial}{\partial W_j}\left(f_W(x) - y\right) \\
&= \left(f_W(x) - y\right)\tfrac{\partial}{\partial W_j}\left(\left(\sum_{i=0}^{n} w_i x_i\right) - y\right) \\
& \left(f_W(x) - y\right)x_j \quad \text{for each j in 1:n} \\
& (XW - Y)^T X \quad \text{overall and in terms of data} \\
&= X^T XW - X^T Y = 0 \\
& X^T XW = X^T Y \quad \text{Normal Equations} \\
& \mathrm{W} = \left(X^T X\right)^{-1} X^T Y
\end{aligned}$$

- **For a full derivation see:** http://www.stanford.edu/class/cs229/notes/cs229-notes1.pdf

Closed form solution to OLS

How do we minimize (3.2)? Denote by $\mathbf{X}$ the $N \times (p+1)$ matrix with each row an input vector (with a 1 in the first position), and similarly let $\mathbf{y}$ be the N-vector of outputs in the training set. Then we can write the residual sum-of-squares as

β is W in our notation

$$\mathrm{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta). \tag{3.3}$$

This is a quadratic function in the $p+1$ parameters. Differentiating with respect to β we obtain

$$\begin{aligned}
\frac{\partial \mathrm{RSS}}{\partial \beta} &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) \\
\frac{\partial^2 \mathrm{RSS}}{\partial \beta \partial \beta^T} &= 2\mathbf{X}^T\mathbf{X}.
\end{aligned} \tag{3.4}$$

Assuming (for the moment) that $\mathbf{X}$ has full column rank, and hence $\mathbf{X}^T\mathbf{X}$ is positive definite, we set the first derivative to zero

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0$$

W is computed directly in closed form

to obtain the unique solution

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{3.6}$$

[Friedman et al. 2001]

Lecture Outline

- **R**
- **Lines, Tangents, Taylors Theorem**
- **Turning points, Roots, 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
- **Predicting Click Through Rates**
  - Linear Regression
  - *Logistic Regression*

- End of lecture

Derivation of Parameter Equations

- An Alternative Derivation treating the y-intercept and the variable coefficients separately; here we represent W as β.
- Goal: Minimize squared error (WRT to the y-intercept)

$$0 = \frac{\partial \sum \hat{\varepsilon}_i^2}{\partial \hat{\beta}_0} = \frac{\partial \sum \left(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i\right)^2}{\partial \hat{\beta}_0}$$

$$= \sum -2\left(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i\right)$$

$$= -2\left(n\bar{y} - n\hat{\beta}_0 - n\hat{\beta}_1 \bar{x}\right)$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

Derivation of Parameter Equations

Derive variable coefficients; here we represent W as β

$$0 = \frac{\partial \sum \hat{\varepsilon}_i^2}{\partial \hat{\beta}_1} = \frac{\partial \sum \left(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i\right)^2}{\partial \hat{\beta}_1}$$

$$= -2\sum x_i\left(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i\right)$$

$$= -2\sum x_i\left(y_i - \bar{y} + \hat{\beta}_1 \bar{x} - \hat{\beta}_1 x_i\right)$$

$$\hat{\beta}_1 \sum x_i (x_i - \bar{x}) = \sum x_i (y_i - \bar{y})$$

$$\hat{\beta}_1 \sum (x_i - \bar{x})(x_i - \bar{x}) = \sum (x_i - \bar{x})(y_i - \bar{y})$$

$$\hat{\beta}_1 = \frac{SS_{xy}}{SS_{xx}}$$

Exercise 1.6 :OLS: Closed Form Solution in R

- Using matrices and the closed from solution estimate the OLS weight (maximum likelihood) using the dataset in exercise 1.5
- To calculate inverse of a matrix
  - ginv() # from library(MASS)
- Other useful matrix commands
  - matrix()
  - det() # division matrix style of a square matrix
  - diag()
  - t() #transpose of a matrix
  - eigen()
  - solve()

Matrix Algebra,The R Book, M. Crawley page 259

Exercise:OLS: Closed Form Solution in R

- Using matrices and the closed from solution estimate the OLS weight (maximum likelihood)

See example.learnLSUsingClosedFormSolution()

```
……preamble
 #dataEx is a training set dataframe
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]);
numVariables=ncol(designMatrix);
w=rep(-2, numVariables);  #initialize weight vector
library(MASS) #make ginv() the inverse of a matrix available
w = ginv(t(X) %*% X) %*% t(X) %*% y;
print(paste("OLS: Closed From Weight is ", w));
```

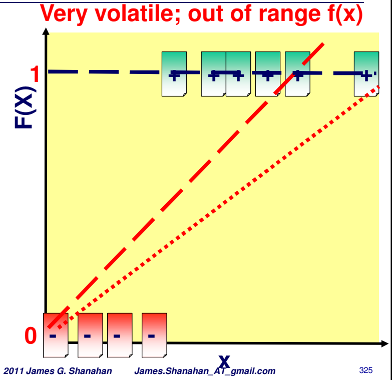
Model classification as a regression?

A linear regression function is linear in the components of X

E.g., $y = ax_1 + bx_2 + c$

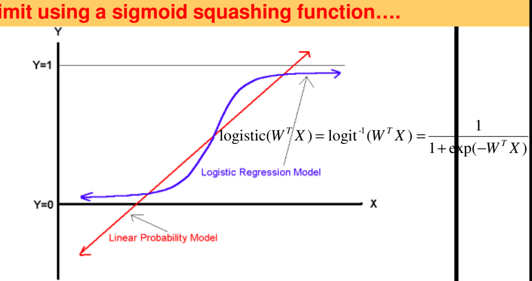
Training Data

E.g.	x_1	x_2	y
1	3	0	0
2			+1
…	…	…	…
L	0	4	0

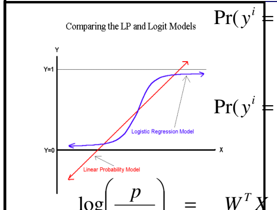


Limit range of f(x) using a logit function

Intuitively it does not make sence to have f(x) >>1 or f(x)<<0
So limit using a sigmoid squashing function….



Logistic Regression: From WX^i to p^i



$$\Pr(y^i = 1 \mid X^i; W) = \text{logit}^{-1}(W^T X^i) = \frac{1}{1+\exp(-W^T X^i)}$$

$$\Pr(y^i = 0 \mid X^i; W) = 1 - \frac{1}{1+\exp(-W^T X^i)} = \frac{\exp(-W^T X^i)}{1+\exp(-W^T X^i)}$$

$$\log\left(\frac{p}{1-p}\right) = W^T X$$

$$\frac{p}{1-p} = \exp(W^T X)$$

$$p = (1-p)\exp(W^T X)$$

$$p = \frac{\exp(W^T X)}{1+\exp(W^T X)} = \frac{1}{1+\exp(-W^T X^i)}$$

As shown earlier

LR: Maximum Likelihood Estimates

- The expression to the right of the *argmax* is the conditional data likelihood.

$$W \leftarrow \arg\max_W \prod_l P(Y^l | X^l, W)$$

Select W s:t likelihood of W generating the data is maximized

$$L(\vec{w}, b) = \prod_{i=1}^{n} p(\vec{x}_i)^{y_i}(1 - p(\vec{x}_i))^{1-y_i}$$

Y can take only values 0 or 1, so only one of the two terms in the expression will be non-zero for any given *Y^l; recall m^0 =1 .*

$$l(W) = \sum_l Y^l \ln P(Y^l = 1 | X^l, W) + (1 - Y^l) \ln P(Y^l = 0 | X^l, W)$$

Working with logs is simpler and more effective computationally; amenable to off-the-shelf optimization approaches; concave function in W so gradient ascent will converge to global maximum (though many may exist). L(W) continuous, differentiable

Conditional Data Likelihood

$$P(Y = 0 | X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^{n} w_i X_i)}$$

$$P(Y = 1 | X) = \frac{\exp(w_0 + \sum_{i=1}^{n} w_i X_i)}{1 + \exp(w_0 + \sum_{i=1}^{n} w_i X_i)}$$

$$\begin{aligned} l(W) &= \sum_l Y^l \ln P(Y^l = 1 | X^l, W) + (1 - Y^l) \ln P(Y^l = 0 | X^l, W) \\ &= \sum_l Y^l \ln \frac{P(Y^l = 1 | X^l, W)}{P(Y^l = 0 | X^l, W)} + \ln P(Y^l = 0 | X^l, W) \\ &= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + \exp(w_0 + \sum_i^n w_i X_i^l)) \end{aligned}$$

Estimating Parameters using Gradient Descent

- Unfortunately, there is no closed form solution to maximizing l(W) with respect to W. Therefore, one common approach is to use gradient ascent, in which we work with the gradient, which is the vector of partial derivatives. The ith component of the vector gradient has the form

$$l(W) = \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + \exp(w_0 + \sum_i^n w_i X_i^l))$$

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

Beginning with initial weights of zero, we repeatedly update the weights in the direction of the gradient, changing the *ith* weight according to *this formula,* where η is a small constant *(e.g., 0:01) which determines the step size. Effectively we are p*ulling weight vector closer to the examples where we make mistakes

Logistic Regression via Gradient Descent

Stochastic Gradient Descent

- **Stochastic update**

$Let\ \ W = (0,0,....)$

$\nabla_{wj}\ l(W_t)$
Partial derivate WRT to variable w_j of error function l(W) at point W_t

Repeat

For j in 0..n #each variable

For i in 1..m #each example

$$W_{j,t+1} = W_{j,t} + \alpha * (y - p) X_j$$

until convergence (i.e., no big changes in W or error)

the term inside the parentheses is simply the prediction error; pulling the W weight vector closer to the example
Batch LR: do a batch update of W_j after a sweep of the data

Logistic Regression in R

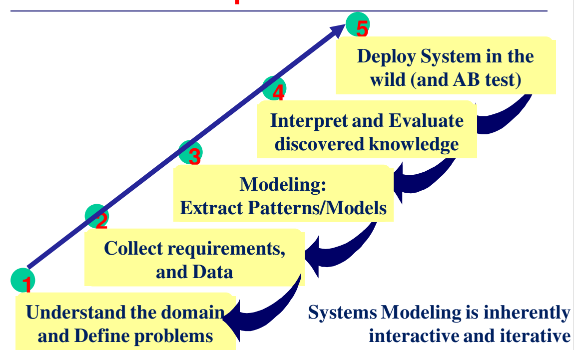
- **Explore Logistic Regression in R**
  - Using Newton-Raphson
  - Using general optimization
  - Using GLM built-in function
  - Using Gradient Descent (homework)

- **Book: John Fox (2002), Sage,** An R and S-PLUS Companion to Applied Regression
  - http://socserv.mcmaster.ca/jfox/Courses/R-course/
- **Accessing man pages in R**
  - ?glm
  - ?solve
  - help.search("solve system") in R

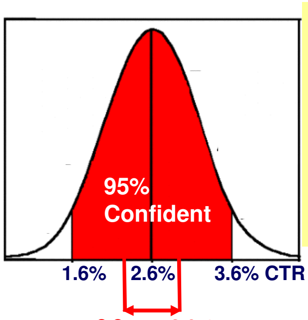
Lecture 2 Outline

- **Taylor Series: quadratic approximations**
- **Newton-Raphson quadratic convergence**
- **Multi-Dimensional Approximations (Planes)**
- **Directional Differentials, Total Differentials**
- **Vector plots, contour plots**
- **Gradient Descent**
  - Linear regression
- **Predicting Click Through Rates**
  - Linear Regression (using gradient descent, MCMC version on 1/26)
  - *Logistic Regression (using gradient descent, MCMC on 1/26)*
- ***Convexity, extreme values, mathematical programming***

Stochastic Optimization in Practice



Estimating CTR (and later AR)



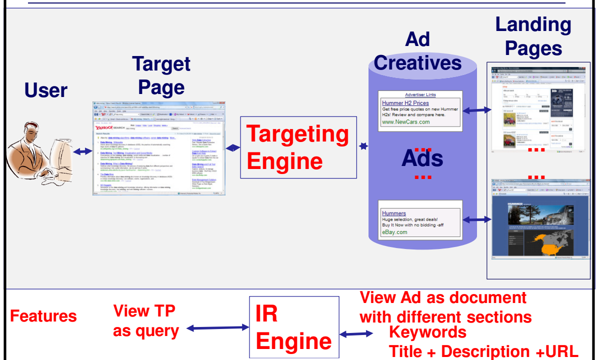
For a network of
~10^9 target pages,
~10^6 ads
~10^7 users
……
- Cannot afford this evaluation/auditioning
- Borrow strength, marginalize
- CoD (curse of dim^{ality})

Accurate CTR Estimates are Crucial

$$ECPM_{Ad} = \boxed{CTR_{Ad}} * Bid_{Ad} * 1000$$

- **Very important to have accurate estimates of CTR_{Ad} for a keyword or publisher page**
  - for ranking and for revenue purposes
  - CTR drop exponentially with position [enquiro.com] ; NDCG Metric
- **E.g., A true CTR for an Ad is 2.6% must be shown 1,000 times before we are 95% confident that this estimate is within 1% of the true CTR, i.e., [1.6, 3.6]**
  - Very noisy!!

Ranking Ads using IR



Estimating CTRs using ML

- **Estimate CTR using Pr_{Ad}(Click|Keyword)**
- **Frame as machine learning problem**
  - E.g., Matthew Richardson, Ewa Dominowska, Robert Ragno: Predicting clicks: estimating the click-through rate for new ads. WWW 2007 pages 521-530
    - Model using Logistic Regression and MART (P... decision trees using stochastic gradient de... [Friedman 2000])
  - Esteban Feuerstein, Pablo Heiber, Ja... ...tinez-Viademonte and Ricardo Baeza-Y... ...ew Stochastic Algorithms for Placing Ads in S... ...ed Search. LA-Web, Santiago, Chile 2007

What features could be used?

ML Features 1/2

Features(KW,AD, LP)->CTR
X_i ->CTR_i

- **Historical data**
  - CTR of KW based on other ads with this KW
  - Related terms CTRs
- **Appearance**

$$p_i = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_{1,i} + \cdots + \beta_k x_{k,i})}}.$$

  - #words in title/body; capitalization; punctuation; word length
- **Attention Capture**
  - Title/body contain action words, e.g., buy/join/etc
- **Reputation**
  - .com/.net/etc, length of URL, #segments in URL, numbers in URL
- **Landing page quality**
  - Contains flash? Fraction of page in images? W3C compliant
- **Text Relevance**
  - keyword match with ad title/body; fraction of match

[Richardson et al., 2007]

ML Features 2/2

- Historical data
- Related terms CTRs
- Appearance
- Attention Capture
- Reputation
- Landing page quality
- **Text Relevance**
  - keyword match with ad title/body; fraction of match
- **10K unigrams (appearing in Ad title and Ad body); bi/trigrams did not bring significant improvement;**
  - Binary feature; 1 if term occurs in ad 0 otherwise
- **Freq of term on web; in query logs**
- **Many others could be used!!!**

[Richardson et al.]

Learning Setup

- **Logistic Regression**

$$p_i = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_{1,i} + \cdots + \beta_k x_{k,i})}}.$$

  - Used a cross entropy loss function
  - Standardized all features using training data
    - (mean and variance, of 0 and 1)
  - Thresholded data beyond 5 std deviations
  - Added derived features
    - (i.e., foreach feature f, log(f +1) and f^2)
- **Baseline**
  - Predict the average CTR of the training dataset
- **MART (Boosted decision trees using stochastic gradient descent [Friedman 2000])**
  - Experiments did not show significant improvement over LR
  - LR is a more transparent model

For LR see:
1. http://en.wikipedia.org/wiki/Logistic_regression
2. http://statgen.iop.kcl.ac.uk/bgim/mle/sslike_4.html

Learning Setup

- **Error measures**
  - Mean Squared error between predicted CTR and true CTR
  - KL Divergence between the predicted CTR and true CTR (in both cases lower is better; 0 is best)
- **Issues?**
  - Weighted?
  - ??

Dataset

- **10,000 Advertisers**
- **1 Million examples of <Keyword, Ad> -> CTR**
  - (view <Keyword, Ad> as <TP, Ad>)
- **Keywords are both exact and broad match**
- **100,000 unique ad texts**
- **Required that each example had more than 100 views**
- **70-10-20 data split (train, validation, test)**

[Richardson et al.]

Proportion Data versus Binary Event

- **An important class of problems involves data on proportions:**
  - Proportion of click responses to impressions
  - data on percentage mortality
  - infection rates of diseases
  - proportion responding to clinical treatment
  - proportion admitting to particular voting intentions
  - data on proportional response to an experimental treatment
- **Model as proportion data or as Bernoulli/binary event**

$$2\sum y \log\left[\frac{y}{\hat{y}}\right] + (n-y)\log\left[\frac{(n-y)}{(n-\hat{y})}\right]$$

LR Modeling of Clicks and Impression

- **In R, using glm()**

```
y <- clicks/impressions
model.CTRPredict  = glm(y ~ log(dose), binomial,
                        weights=impressions)
summary(model.CTRPredict)
```

- **and internally glm converts models with a two-column response to this form, for it is in this form the binomial fits into the GLM framework.**

For Proportion Data

- **In R, using glm() or you can pass in**
- **number.of.failures = binomial.denominator – number.of.successes**
- **y <- cbind(number.of.successes, number.of.failures)**

Bernoulli trials vs. Proportion Data

The response variable contains only 0's or 1's; for example, 0 to represent dead individuals and 1 to represent live ones. There is a single column of numbers, in contrast to proportion data (see above). The way SPlus treats this kind of data is to assume that the 0's and 1's come from *a binomial trial with sample size 1*. If the probability that an animal is dead is p, then the probability of obtaining y (where y is either dead or alive, 0 or 1) is given by an abbreviated form of the binomial distribution with $n = 1$, known as the Bernoulli distribution:

$$P(y) = p^y (1-p)^{(1-y)}$$

The random variable y has a mean of p and a variance of $p(1\text{-}p)$, and the object is to determine how the explanatory variables influence the value of p.

The trick to using binary response variables effectively is to know when it is worth using them and when it is better to lump the successes and failures together and analyse the *total counts* of dead individuals, occupied patches, insolvent firms or whatever. The question you need to ask yourself is this:

do I have unique values of one or more explanatory variables for each and every individual case?

If the answer is 'yes', then analysis with a binary response variable is likely to be fruitful. If the answer is 'no', then there is nothing to be gained, and you should reduce your data by aggregating the counts to the resolution at which each count *does* have a unique set of explanatory variables. For example, suppose that all your explanatory variables were categorical (say sex (male or female), employment

http://www.bio.ic.ac.uk/research/crawley/statistics/exercises/R10Proportiondata.pdf Page 2

Logistic Regression: L-BFGS

The logistic regression was trained using the limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method [16]. We used a cross-entropy loss function, with zero-mean Gaussian weight priors with a standard-deviation of σ. The best σ was chosen on the validation set from the values [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 100]. In all experiments, σ=0.1 was the best. As is commonly done, we also added a bias feature that is always set to 1.

Used regularized LR

Results

Table 7: Comparison of results for a model trained and tested on ads with over 100 views vs. over 1000 views.

Features	%Imprv	
	>100 views	>1000 views
Baseline ($\overline{CTR}$)	-	-
+Term CTR	13.28	25.22
+Related CTR	19.67	32.92
+Ad Quality	23.45	33.90
+Order Specificity	28.97	40.51
+Search Data	29.47	41.88

Transparency of Results

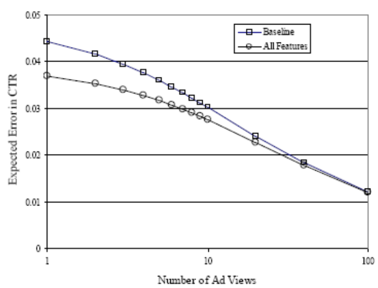
Table 5: Non-unigram features with highest (lowest) weight

Top ten features	*Bottom ten features*
log(#chars in term)	log(# terms in order)
v_{12}	log(v_{0*})
v_{22}	sqr(p_{00})
log(order category entropy)	sqr(order category entropy)
log(#most common word)	log(#chars in landing page)
sqr(#segments in displayurl)	log(a_{01})
sqr(#action words in body)	a_{13}
p_{10}	sqr(p_{0*})
p_{**}	log(#chars in body)
log(v_{00})	sqr(#chars in term)

Table 6: Unigrams with highest (and lowest) weight.

Top ten unigrams		*Bottom ten unigrams*	
official	body	quotes	title
download	title	hotels	title
photos	body	trial	body
maps	body	deals	body
official	title	gift	body
direct	body	have	text
costumes	title	software	title
latest	body	engine	body
version	body	compare	title
complete	body	secure	body

CTR Evolution



$$\hat{p} = \frac{\alpha p_0 + clicks}{\alpha + views}$$

Model predicted CTR using a Beta distribution where p_0 is the predicted (and baseline) CTR
α: strength of prior as measured in terms of impressions

Figure 6: Expected mean absolute error in CTR as a function of the number of times an ad is viewed.

Estimating CTRs using ML

Intermediate Conclusions

- **Richardson et al. report a very interesting approach and case study**
  - Despite realistic problem setting results are preliminary
- **Transparency of model**
- **Using many features helps insulate from adversarial attacks (can be useful in adversarial detection)**
- **Applied to new ads but could be extended to deal with existing ads, display/graphical ads**
  - Homework!!
- **But many issues remain!!**

Modeling CTR Challenges

- **Extremely rare events (Typical CTRs < 1% for contextual)**
- **Biased dataset (the rich get richer; suboptimal locking)**
- **Very sparse (only a small percentage of <TP, Ad> get impressions; can impede generalization)**
  - Missed opportunities
- **Accuracy of estimates**
  - ML approaches are hugely biased; bias correction [see Provost and Domingos; Platt]
- **Scale and Speed**
- **Non-Stationary, new ads, changes in network**
- **Marginalization versus segmentation (resolution vs. sufficient data)**
- **….**

Exercise

- **Mobile advertising is defined as showing ads on mobile phone contexts such within a browser or app (application).**
- **What types of features could be leverage within a mobile context to better target consumers?**
- **Are these features real-valued, nominal, categorical?**

Thanks

EMAIL:
James_DOT_Shanahan_AT_gmail_DOT_com

•R Break

Solve a System of Equations in R

- **Solve the system of linear equations.**
  -2x + 3y = 8
  3x - y = -5
- **multiply all terms in the second equation by 3**
  -2x + 3y = 8
  9x - 3y = -15

  7x = -7 # add the two equations

  Note: y has been eliminated, hence the name: method of elimination
  solve the above equation for x
  x = -1
  substitute x by -1 in the first equation
  -2(-1) + 3y = 8
  solve the above equation for y
  2 + 3y = 8
  3y = 6
  y = 2
- **write the solution as ordered pair (-1,2**

```
>A <- matrix(c(-2,3, 3,-1 ), 2)
> A
     [,1] [,2]
[1,]   -2    3
[2,]    3   -1
> b
[1] 8 5
> b=c(8,-5)
> qr.solve(A, b) # or solve(qr(A), b)
[1] -1  2
```


Matrices

See example.Matrices()

See local file MatricesInR.doc

- **To calculate inverse of a matrix**
  - # division for matrices
  - ginv() # from library(MASS)
- **Other useful matrix commands**
  - matrix()
  - det()
  - diag()
  - t() #transpose of a matrix
  - eigen()
  - solve() #compute inverse or solve system of equations

Matrix Algebra,The R Book, M. Crawley page 259

Data in R

See example.DataFrames()

- **Dataframes, matrices etc…**
- **Data input:**
  - From the keyboard.
  - From an ascii (plain text) file.
  - From the clipboard.
  - Importing data (e.g., from SPSS).
  - From a database-management system.
  - From an R package.
- **The R search path.**
- **Missing data.**
- **Numeric variables, character variables, and factors**
- http://socserv.mcmaster.ca/jfox/Courses/R-course/Session-3-script.R

Matrices in R

See example.Matrices()

- **Linear equations**
- **Determinant**

- **See presentation in local dir Matrices and Singular values**
- MatricesLectureSingularValues.pptx

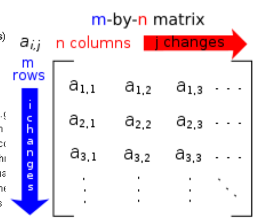
Matrices, Vectors (in R)

- **For more background see**
  - http://en.wikipedia.org/wiki/Euclidean_vector
  - http://en.wikipedia.org/wiki/Matrix_(mathematics)

In mathematics, a **matrix** (plural **matrices**, or less commonly **matrixes**) of numbers, such as

$$\begin{bmatrix} 1 & 2 & 3 \\ 6 & 5 & 4 \end{bmatrix}.$$

An item in a matrix is called an **entry** or an **element**. In the example, e.g Entries are often denoted by a variable with two subscripts, as shown on the same size can be added and subtracted entrywise and matrices of c multiplied. These operations have many of the properties of ordinary arith matrix multiplication is not commutative, that is, **AB** and **BA** are not equa consisting of only one column or row are called vectors, while higher-dime dimensional, arrays of numbers are called tensors. Matrices with entries are also studied.



Matrices are a key tool in linear algebra. One use of matrices is to represent linear transformations, which are higher-dimensional analogs of linear functions of the form $f(x) = cx$, w corresponds to composition of linear transformations. Matrices can also keep track of the coeffic

R Notes

- **Matrix Ops**
- **Solve(a, b)**
  - #solve a system of equations Ax=b by b=A^{-1}b; b is combination of the column in A.
  - This generic function solves the equation a %*% x = b for x, where b can be either a vector or a matrix.
  - a: a square numeric or complex matrix containing the coefficients of the linear system.
  - b: a numeric or complex vector or matrix giving the right-hand side(s) of the linear system.
  - If missing, b is taken to be an identity matrix and solve will return the **inverse of a**.

Solve a System of Equations in R

- **<u>Example 1:</u> Solve the system of linear equations.**

-2x + 3y = 8
3x - y = -5

- **multiply all terms in the second equation by 3**

-2x + 3y = 8
9x - 3y = -15
add the two equations
7x = -7

Note: y has been eliminated, hence the name: elimination solve the above equation for x
x = -1
substitute x by -1 in the first equation
-2(-1) + 3y = 8
solve the above equation for y

2 + 3y = 8
3y = 6

```
➢A <- matrix(c(-2,3, 3,-1 ), 2)
> A
    [,1] [,2]
[1,]   -2    3
[2,]    3   -1
> b
[1] 8 5
> b=c(8,-5)
> qr.solve(A, b) # or solve(qr
[1] -1  2
```


Debugging in R

- http://www.stats.uwo.ca/faculty/murdoch/software/debuggingR/debug.shtml

Debugging in R

- **Use browser() #?browser commands like c/c++ debugger**
  - n #next
  - c # continue
  - Q quit
- **For more details on debugging on R RTFM (see next slide for useful example) !!**
  - http://www.stats.uwo.ca/faculty/murdoch/software/debuggingR/debug.shtml
- **Locating an error: traceback().**
  - Setting a breakpoint and examining the local environment of an executing function: browser().
  - A simple interactive debugger: debug().
  - A more sophisticated debugger: the debug package.
  - There is also a "postmortem" debugger: debugger() (which I'll not discuss)

R debuging via browser()

- **This kind of use of browser can be useful if you have a vague idea as to where a bug may be in your program.**
- **Notice that the first two lines in the function were not printed.**

3.2.1 Explicit Calls to browser

It is possible to do a kind of "manual debugging" if you don't feel like stepping through a function line by line. The function browser can be used to suspend execution of a function so that the user can browse the local environment. Suppose we edited the SS function from above to look like:

```
SS <- function(mu, x) {
    d <- x - mu
    d2 <- d^2
    browser()
    ss <- sum(d2)
    ss
}
```

Now, when the function reaches the third statement in the program, execution will suspend and you will get a Browse[1]> prompt, much like in the debugger.

```
> SS(2, x)
Called from: SS(2, x)
Browse[1]> ls()
[1] "d"  "d2" "mu" "x"
Browse[1]> print(mu)
[1] 2
Browse[1]> mean(x)
[1] 0.02176075
Browse[1]> n
debug: ss <- sum(d2)
Browse[1]> c
[1] 503.814
```

Commands in debug mode

When the debugger is invoked, you are left in a browser(). Expressions typed at the prompt are evaluated in the local environment.

<RET>
Go to the next statement if the function is being debugged. Continue execution if the browser was invoked.
c or *cont*
Continue execution without single stepping.
n
Execute the next statement in the function. This works from the browser as well.
where
Show the call stack.
Q
Halt execution and jump to the top-level immediately.

To view the value of a variable whose name matches one of these commands, use the print() function, e.g. print(n).

When the debugger is invoked, you are left in a `browser()`. Expressions typed at the prompt are evaluated in the local environment. The following commands are available.

<RET>
Go to the next statement if the function is being debugged. Continue execution if the browser was invoked.

c or *cont*
Continue execution without single stepping.

n
Execute the next statement in the function. This works from the browser as well.

where
Show the call stack.

Q
Halt execution and jump to the top-level immediately.

To view the value of a variable whose name matches one of these commands, use the `print()` function, e.g. `print(n)`.

Here is a sample session, based on the one in the R Language manual.

```
> debug(mean.default)
> mean(1:10)
debugging in: mean.default(1:10)
debug: {
    if (na.rm)
        x <- x[!is.na(x)]
    trim <- trim[1]
    n <- length(c(x, recursive = TRUE))
    if (trim > 0) {
        if (trim >= 0.5)
            return(median(x, na.rm = FALSE))
        lo <- floor(n * trim) + 1
        hi <- n + 1 - lo
        x <- sort(x, partial = unique(c(lo, hi)))[lo:hi]
        n <- hi - lo + 1
    }
    sum(x)/n
}
Browse[1]> where
where 1: mean.default(1:10)
where 2: mean(1:10)

Browse[1]>
debug: if (na.rm) x <- x[!is.na(x)]
Browse[1]>
debug: trim <- trim[1]
Browse[1]> trim
[1] 0
Browse[1]>
debug: n <- length(c(x, recursive = TRUE))
```

http://www.stats.uwo.ca/faculty/murdoch/software/debuggingR/debug.shtml

Multi-Variable Unconstrained Optimization

- Because the objective function $f(\mathbf{x})$ is assumed to be differentiable, it possesses a gradient, denoted by $f(\mathbf{x})$, at each point $\mathbf{x}$. In particular, the **gradient** at a specific point $\mathbf{x} = \mathbf{x'}$ is the *vector* whose elements are the respective *partial derivatives* evaluated at $\mathbf{x} = \mathbf{x'}$, so that

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right).$$

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

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

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

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

Appendix: Cheat Sheets

- **Notation**
- **Calculus**
- **Algebra**
- **Matrices**

Notation

X or ***X***	Uppercase/Bold letters denotes a vector
X^T	Transpose of vector *X*
x	Lowercase letters denotes a variable
x_i	A lowercase subscripted letter denotes a variable component of a vector
y	Output or dependent variable
X or ***X***	Input vector
n	The dimension or number of input features/variables
L or *m*	The number of training examples
W	The weight vector component of a hyperplane
b	Bias or threshold component of a hyperplane
(W, b)	A hyperplane with weight vector *W* and bias component *b*
S	Training sample
γ	Margin
ξ	Slack variable
η	Learning rate
$\Phi(.)$	Input feature transformation/remapping function
α	Dual variable or Lagrange multiplier
d	VC dimension
h	A hypothesis or model (e.g., a hyperplane *(W, b)*)
$\sum_{i=1}^{n} x_i$	The sum $x_1 + x_2 + \ldots + x_n$

Notation

$< X, Z >= \sum_{i=1}^{n} x_i z_i$	The inner product (or dot product) between two vector *X* and *Z*
$K(X,Z) =< \Phi(X), \Phi(Z) >$	A kernel function whose effect is the dot product of two vectors that have been transformed into a new feature space induced by Φ.
$\prod_{i=1}^{n} x_i$	The product $x_1 \times x_2 \times \ldots \times x_n$
$\arg\max_{x \in \Omega_X} f(x)$	The value of *x* that maximizes *f(x)*. For example, $\arg\max_{x \in \{1,2,-3\}} f(x^2) = -3$
$\arg\min_{x \in \Omega_X} f(x)$	The value of *x* that minimizes *f(x)*. For example, $\arg\min_{x \in \{1,2,-3\}} f(x^2) = 1$
$\lVert W \rVert_2$ or $\lVert W \rVert$	$\sqrt[2]{\sum_{i=1}^{n} (w_i)^2}$ where W is a vector and w_i is a component of W Often referred to as the Euclidean Norm
$\lVert W \rVert_1$	$\sum_{i=1}^{n} abs(w_i)$ where W is a vector and w_i is a component of W and abs(.) denotes the absolute value
$\varnothing$	Null set or empty set

Notation

Α α Alpha
Β β Beta
Γ γ Gamma
Δ δ Delta
Ε ε Epsilon
Ϝ ϝ Digamma
Ζ ζ Zeta
Η η Eta
Θ θ Theta
Ι ι Iota
Κ κ Kappa
Λ λ Lambda
Μ μ Mu
Ν ν Nu
Ξ ξ Xi (zai)
Ο ο Omicron
Π π Pi
□ □ San
Ϙ ϙ Qoppa
Ρ ρ Rho
Σ σ Sigma
Τ τ Tau
Υ υ Upsilon
Φ φ Phi
Χ χ Chi
Ψ ψ Psi
Ω ω Omega
Ϡ ϡ Sampi

Greek alphabet

{x | P(x)}	Set determined by the property *P*. "|" is read as "such that".
$<x_1, x_2, \ldots, x_n>$	n-tuple
$\forall$	Universal quantifier denoting *for all*
$\exists$	Existential quantifier denoting *there exists*
|A|	Cardinality of a set A
{a, …, b}	{a, …, b} denotes a discrete interval, such that $a \le x \le b$ $\forall x \in$ {a, …, b}. For example, {1, …, 6} denotes {1, 2, 3, 4, 5, 6}
(a, b)	A continuous interval denoting any value *x* that satisfies the following condition: $a < x < b$
[a, b]	A continuous interval denoting any value *x* that satisfies the following condition: $a \le x \le b$
$\Re$	Set of all real numbers

Cheat Sheets & Tables

http://tutorial.math.lamar.edu/

- **Algebra Cheat Sheet - This is as many common algebra facts, properties, formulas, and functions that I could think of. There is also a page of common algebra errors included. Currently the cheat sheet is four pages long.**
  Algebra Cheat Sheet (Reduced) - This is the same cheat sheet as above except it has been reduced so that it will fit onto the front and back of a single piece of paper. It contains all the information that the normal sized cheat sheet does.
  Trig Cheat Sheet - Here is a set of common trig facts, properties and formulas. A unit circle (completely filled out) is also included. Currently this cheat sheet is four pages long.
  Trig Cheat Sheet (Reduced) - My standard trig cheat sheet reduced to fit onto the front and back of a single piece of paper. It contains all the information that the normal sized cheat sheet does.
  Calculus Cheat Sheets - These are a series of Calculus Cheat Sheets that covers most of a standard Calculus I course and a few topics from a Calculus II course.
  Common Derivatives and Integrals - Here is a set of common derivatives and integrals that are used somewhat regularly in a Calculus I or Calculus II class. Also included are *reminders* on several integration techniques. Currently this cheat sheet is four pages long.
  Common Derivatives and Integrals (Reduced) - My common derivatives and integrals table reduced to fit onto the front and back of a single piece of paper. It contains all the information that the normal sized table does.
  Table of Laplace Transforms - Here is a list of Laplace transforms for a differential equations class. This table gives many of the commonly used Laplace transforms and formulas.

. Product Rule	$\frac{d}{dx}[uv] = u'v + uv'$	12. Simple Exponential Derivative Rule	$\frac{d}{dx}[e^x] = e^x$
. Quotient Rule	$\frac{d}{dx}\left[\frac{u}{v}\right] = \frac{u'v - uv'}{v^2}$	13. General Exponential Derivative Rule	$\frac{d}{dx}[e^u] = e^u u'$
. Very Simple Power Rule	$\frac{d}{dx}[x] = 1$	14. Simple Different Exponential Base Derivative Rule	$\frac{d}{dx}[a^x] = (\ln a)a^x$
. Simple Power Rule	$\frac{d}{dx}[x^n] = nx^{n-1}$	15. General Different Exponential Base Derivative Rule	$\frac{d}{dx}[a^u] = (\ln a)a^u u'$
. General Power Rule	$\frac{d}{dx}[u^n] = nu^{n-1}u'$	16. Simple Logarithm Derivative Rule	$\frac{d}{dx}[\ln x] = \frac{1}{x}$
. Chain Rule	$\frac{d}{dx}[f(u)] = f'(u)u'$	17. General Logarithm Derivative Rule	$\frac{d}{dx}[\ln u] = \frac{u'}{u}$
0. Simple Absolute Value Derivative Rule	$\frac{d}{dx}[\lvert x \rvert] = \frac{x}{\lvert x \rvert}$	18. Simple Different Base Logarithm Derivative Rule	$\frac{d}{dx}[\ln_a x] = \frac{1}{(\ln a)x}$
1. General Absolute Value Derivative Rule	$\frac{d}{dx}[\lvert u \rvert] = \frac{u}{\lvert u \rvert}u'$	19. General Different Base Logarithm Derivative Rule	$\frac{d}{dx}[\ln_a u] = \frac{u'}{(\ln a)u}$
2. Simple Exponential Derivative Rule	$\frac{d}{dx}[e^x] = e^x$		

http://www.freehomeworkmathhelp.com/Calculus_I/Calculus_I_Rules/calculus_I_rules_for_derivatives_integrals.html

Differentiation Rules

$$\frac{d}{dx}\exp(x) = \exp(x)$$

Example 1

Find the derivative of $f(x) = \exp(x^2)$.

We use the chain rule.

$$\frac{d}{dx}\exp(x^2) = \frac{d}{dx}\exp(y)$$

(where $y = x^2$)

$$= \frac{d}{dy}\exp(y).\frac{dy}{dx}$$
$$= \exp(y).\frac{d}{dx}x^2$$
$$= 2x\exp(x^2)$$

Derivative

how much a function changes as its input changes

- **The derivative is a measure of how a function changes as its input changes.**
  - Loosely speaking, a derivative can be thought of as how much one quantity is changing in response to changes in some other quantity;
- **The derivative of a function at a chosen input value describes the best linear approximation of the function near that input value.**
  - For a real-valued function of a single real variable, the derivative at a point equals the slope of the tangent line to the graph of the function at that point.
  - In higher dimensions, the derivative of a function at a point is a linear transformation called the linearization.

http://en.wikipedia.org/wiki/Derivative

Derivative Notation

- **Leibniz's Notation**

$$\left.\frac{dy}{dx}\right|_{x=a} = \frac{dy}{dx}(a).$$

- **Lagrange's Notation**
  - f'(a)
- **Newton's Notation**
  - *Assume y = f(t)* the first derivative and second derivative are denoted as: $\dot{y}$ $\ddot{y}$
- **Laplacian**
  - $\nabla^2\varphi$

Unconstrained optimization

For noncovex/non-concave

$$\begin{array}{ll} \text{max/ min} & f(x) = f(x_1,\dots,x_n) \\ \text{subject to} & x \in R^n \end{array}$$

- **Assume f'(x) and *H(x)* exists for all $x \in S$**
- **Locate candidate extrema using f'(x) = 0 and boundary points**
- **Then candidate x is**
  - *f(x)* is a convex function on *S* if and only if all principal minors of *H(x)* are nonnegative for all $x \in S$
  - *f(x)* is a concave function on *S* if and only if the principal minors of *H(x)* of order *k* have the same sign as $(-1)^k$ for all $x \in S$ and all *k*