Optimization

Feng Li
feng.li@cufe.edu.cn

School of Statistics and Mathematics
Central University of Finance and Economics

Revised on August 15, 2018

Today we are going to learn...

Structure

- We cover many different topics. This week: Optimization
- For each topic we consider the following
  - Motivation
  - Intuition
  - Mathematics
  - Code

Optimization in Business

- Many problems in business require something to be minimized or maximized
  - Maximizing Revenue
  - Minimizing Costs
  - Minimizing Delivery Time
  - Maximizing Financial Returns

Input and output

- For many of these problems there is some control over the input
  - Maximizing Revenue - Price
  - Minimizing Costs - Number of Workers
  - Minimizing Delivery Time - Driving Route
  - Maximizing Financial Returns - Portfolio weights

Optimization in Statistics

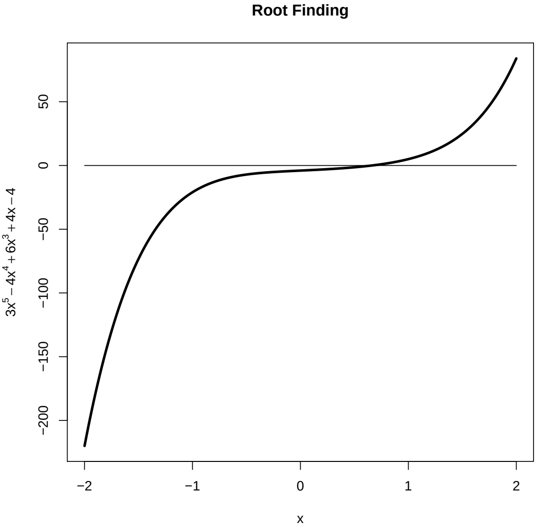
- In statistics, many estimators maximize or minimize a function
  - Maximum Likelihood
  - Least Squares
  - Method of Moments
  - Posterior Mode

- Suppose we want to find an minimum or maximum of a function $f(x)$
- Sometimes $f(x)$ will be very complicated
- Are there computer algorithms that can help?
- YES!
    - Newton's Method
    - Quasi-Newton
    - Nelder Mead

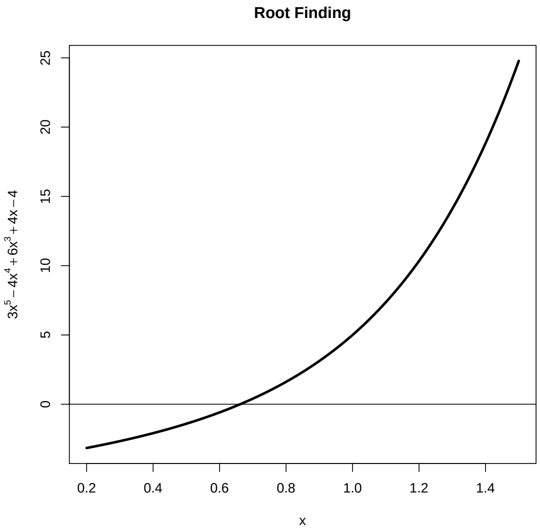
Root of a function

- Consider the problem of finding the **root** or **zero** a function.
- For the function $g(x)$ the **root** is the point x^* such that $g(x^*) = 0$
- An algorithm for solving this problem was proposed by Newton and Raphson nearly 500 years ago.
- We will use this algorithm to find the root of $g(x) = 3x^5 - 4x^4 + 6x^3 + 4x - 4$

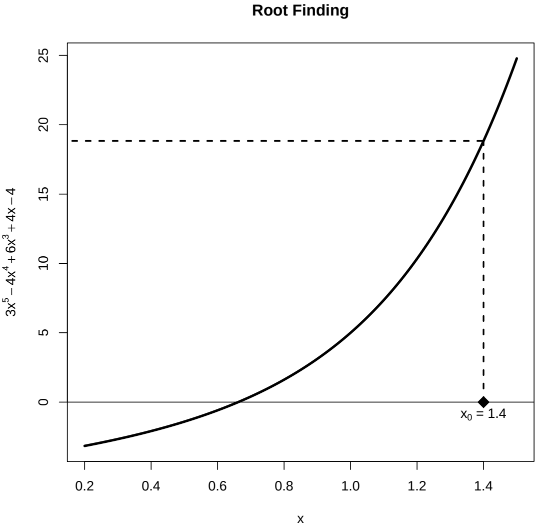
Root of a function



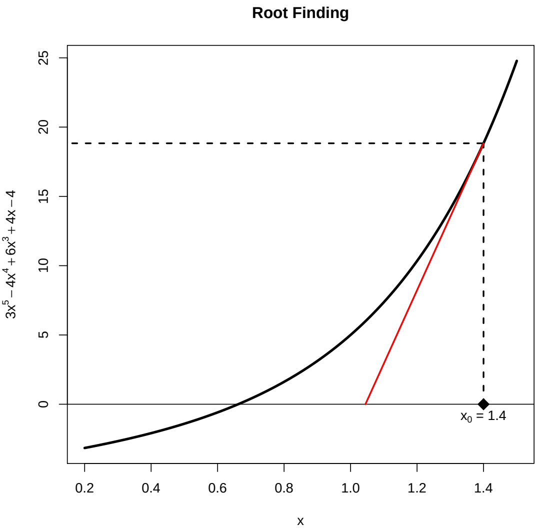
Root of a function



Initial Guess ($g(x_0) = 18.8$)

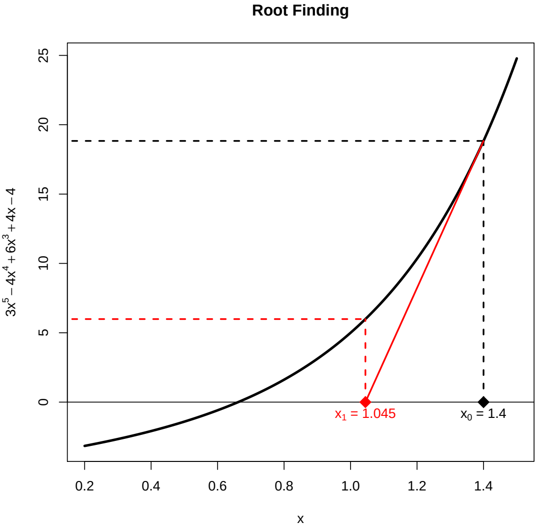


Tangent



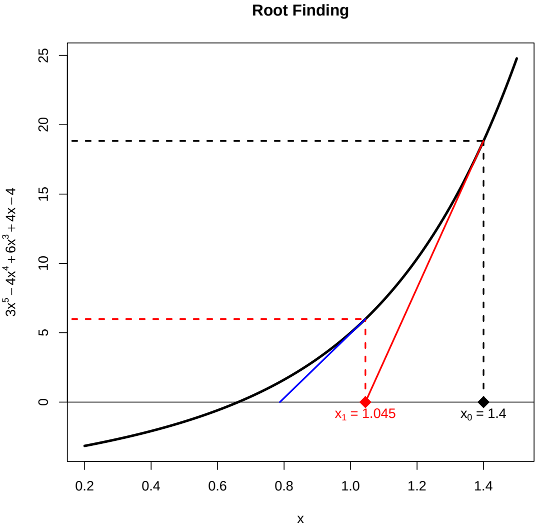
Root Finding
25
20
15
10
5
0
$3x^5 - 4x^4 + 6x^3 + 4x - 4$
$x_0 = 1.4$
0.2
0.4
0.6
0.8
1.0
1.2
1.4
x

Now $g(x_1) = 6.0$



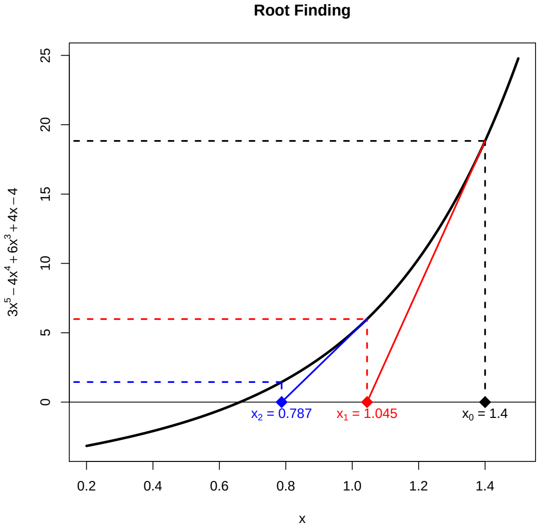
Root Finding
$3x^5 - 4x^4 + 6x^3 + 4x - 4$
x
0
5
10
15
20
25
0.2
0.4
0.6
0.8
1.0
1.2
1.4
$x_1 = 1.045$
$x_0 = 1.4$

Do it again...



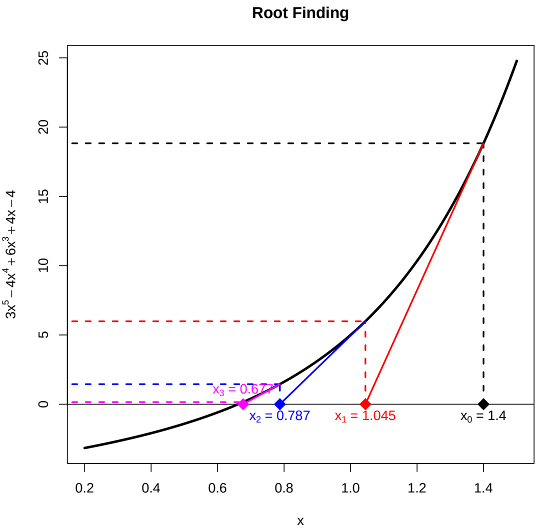
Root Finding
25
20
15
10
5
0
$3x^5 - 4x^4 + 6x^3 + 4x - 4$
x_1 = 1.045
x_0 = 1.4
0.2
0.4
0.6
0.8
1.0
1.2
1.4
x

Now $g(x_2) = 1.4$



Root Finding
25
20
15
10
5
0
$3x^5 - 4x^4 + 6x^3 + 4x - 4$
x_2 = 0.787
x_1 = 1.045
x_0 = 1.4
0.2
0.4
0.6
0.8
1.0
1.2
1.4
x

...and again: $g(x_3) = 0.2$



Finding the Tangent

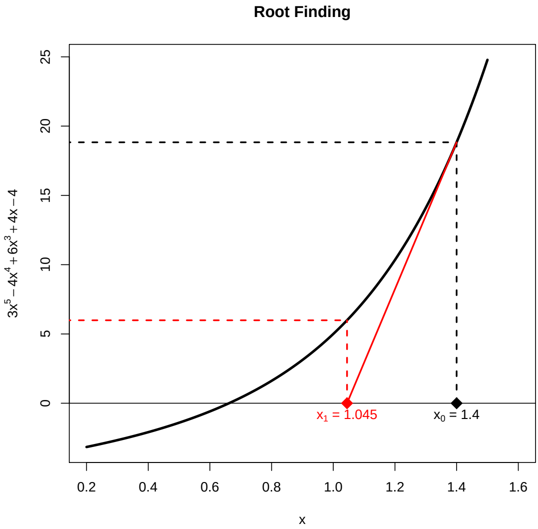
- To find the tangent evaluate the first derivative of $g(x)$.
- The function is

$$g(x) = 3x^5 - 4x^4 + 6x^3 + 4x - 4 \tag{1}$$

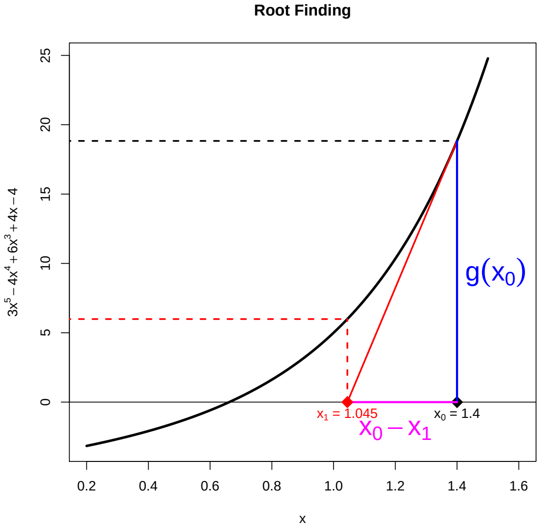
- The first derivative is

$$g'(x) = 15x^4 - 16x^3 + 18x^2 + 4 \tag{2}$$

Find the crossing point



Find the crossing point



Find the crossing point

From basic Geometry

$$g'(x_0) = \frac{g(x_0)}{x_0 - x_1} \tag{3}$$

Rearrange

$$x_0 - x_1 = \frac{g(x_0)}{g'(x_0)} \tag{4}$$

$$-x_1 = -x_0 + \frac{g(x_0)}{g'(x_0)} \tag{5}$$

$$x_1 = x_0 - \frac{g(x_0)}{g'(x_0)} \tag{6}$$

Stopping Rule

- With each step the algorithm should get closer to the root.
- However, it can run for a long time without reaching the **exact** root
- There must be a **stopping rule** otherwise the program could run forever.
- Let ϵ be an extremely small number e.g. 1×10^{-10} called the **tolerance level**
- If $|g(x^*)| < \epsilon$ then the solution is close enough and there is a root at x^*

Newton-Raphson Algorithm

1. Select initial value x_0 and set $n = 0$
2. Set $x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)}$
3. Evaluate $|g(x_{n+1})|$
    - If $|g(x_{n+1})| \leqslant \epsilon$ then stop.
    - Otherwise set $n = n + 1$ and go back to step 2.

Your task

Write R code to find the root of $g(x) = 3x^5 - 4x^4 + 6x^3 + 4x - 4$
Tips:

- Write functions for $g(x)$ and $g'(x)$ first.
- These can be inputs into a function that carries out the Newton Raphson method. Code should be flexible.
- Use loops!

Another Problem

- Now use your Newton-Raphson code to find the root of $g(x) = \sqrt{|x|}$
- The derivative has two parts

$$g'(x) = \begin{cases} 1/\sqrt{x} & \text{if } x > 0 \\ -1/\sqrt{-x} & \text{if } x < 0 \end{cases} \tag{7}$$

- Use 0.25 as the starting value

Learn from mistakes

- Newton-Raphson does not always converge
- Be careful using *while*. Avoid infinite loops.
- Don't always assume the answer given by code is correct. Check carefully!
- Print warning messages in code

Next Example

Next example:

$$g(x) = xe^{-x^2} - 0.4(e^x + 1)^{-1} + 0.2 \tag{8}$$

Try two different starting values

- Starting value $x_0 = 0.5$
- Starting value $x_0 = 0.6$

Next Example

Next example:

$$g(x) = x^3 - 2x^2 - 11x + 12 \tag{9}$$

Try two different starting values

- Starting value $x_0 = 2.35287527$
- Starting value $x_0 = 2.35284172$

Next Example

Next example:

$$g(x) = 2x^3 + 3x^2 + 5 \tag{10}$$

Try two different starting values

- Starting value $x_0 = 0.5$
- Starting value $x_0 = 0$

Learn from mistakes

- For some functions, using some certain starting values leads to a series that **converges**, while other starting values lead to a series that **diverges**
- For other functions different starting values converge to different roots.
- Be careful when choosing the initial value.
- Newton-Raphson doesn't work if the first derivative is zero.
- When can this happen?

Rough Proof of Quadratic Convergence

- Can we prove anything about the rate of convergence for the Newton Raphson Method?
- To do so requires the **Taylor Series**
- Let $f(x)$ have a root at α. The Taylor approximation states that

$$f(\alpha) \approx f(x_n) + f'(x_n)(\alpha - x_n) + \frac{1}{2}f''(x_n)(\alpha - x_n)^2 \tag{11}$$

- The quality of the approximation depends on the function and how close x_n is to α

Rough Proof of Convergence

- Since α is a root, $f(\alpha) = 0$ This implies

$$0 \approx f(x_n) + f'(x_n)(\alpha - x_n) + \frac{1}{2} f''(x_n)(\alpha - x_n)^2 \tag{12}$$

- Dividing by $f'(x_n)$ and rearranging gives:

$$\frac{f(x_n)}{f'(x_n)} + (\alpha - x_n) \approx \frac{-f''(x_n)}{2f'(x_n)} (\alpha - x_n)^2 \tag{13}$$

- More rearranging

$$\alpha - \left(x_n - \frac{f(x_n)}{f'(x_n)}\right) \approx \frac{-f''(x_n)}{2f'(x_n)}(\alpha - x_n)^2 \tag{14}$$

- The term in brackets on the left hand side is the formula used to update x in the Newton Raphson method

$$(\alpha - x_{n+1}) \approx \frac{-f''(x_n)}{2f'(x_n)}(\alpha - x_n)^2 \tag{15}$$

- This can be rewritten in terms of errors $e_{n+1} = \alpha - x_{n+1}$ and $e_n = \alpha - x_n$

$$e_{n+1} \approx \frac{-f''(x_n)}{2f'(x_n)}e_n^2 \tag{16}$$

Conclusion

- Why did we spend so much time on finding roots of an equation?
- Isn't this topic meant to be about optimization?
- Can we change this algorithm slightly so that it works for optimization?

Finding a maximum/minimum

- Suppose we want to find an minimum or maximum of a function $f(x)$
- First order condition: Find the derivative $f'(x)$ and find x^* such that $f'(x^*) = 0$
- This is the same as finding a root of the first derivative. We can use the Newton Raphson algorithm on the first derivative.

Newton's algorithm for finding local minima/maxima

1. Select initial value x_0 and set $n = 0$
2. Set $x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$
3. Evaluate $|f'(x_{n+1})|$
   - If $|f'(x_{n+1})| < \epsilon$ then stop.
   - Otherwise set $n = n + 1$ and go back to step 2.

Different Stopping Rules

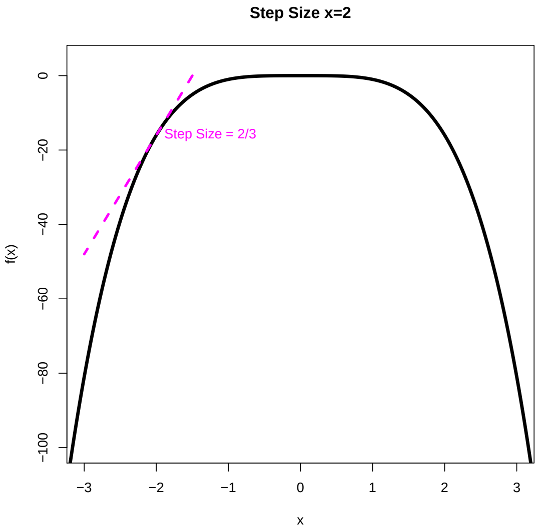
Three stopping rules can be used

- $|f'(x_n)| \leqslant \epsilon$
- $|x_n - x_{n-1}| \leqslant \epsilon$
- $|f(x_n) - f(x_{n-1})| \leqslant \epsilon$

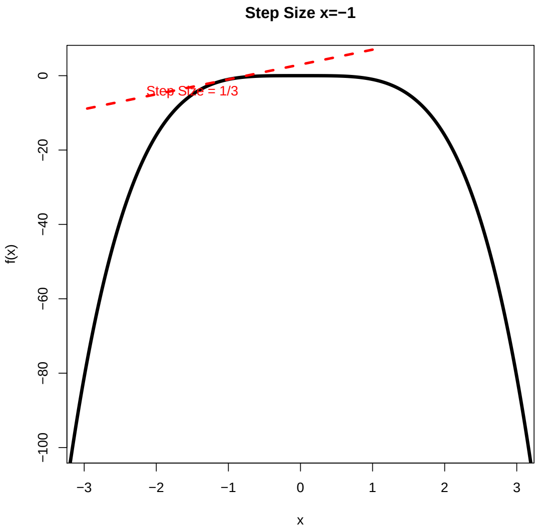
Intuition

- Focus the **step size** $-\frac{f'(x)}{f''(x)}$.
- The **signs** of the derivatives control the **direction** of the next step.
- The **size** of the derivatives control the **size** of the next step.
- Consider the concave function $f(x) = -x^4$ which has $f'(x) = -4x^3$ and $f''(x) = -12x^2$. There is a maximum at $x^* = 0$

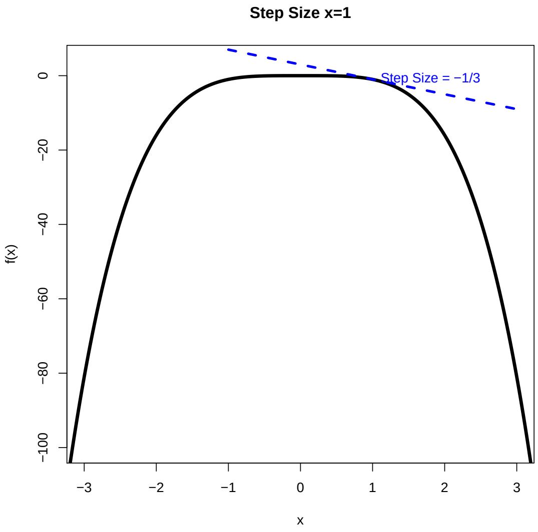
Role of first derivative



Role of first derivative



Role of first derivative



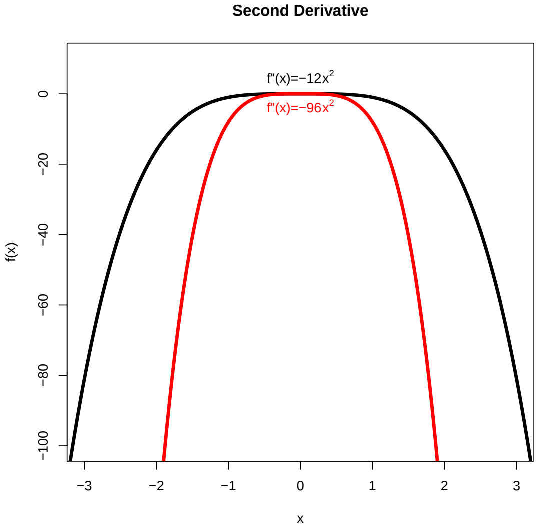
Role of first derivative

- If $f''(x)$ is negative the function is locally **concave**, and the search is for a local **maximum**
- To the left of this maximum $f'(x) > 0$
- Therefore $-\frac{f'(x)}{f''(x)} > 0$.
- The next step is to the right.
- The reverse holds if $f'(x) < 0$
- Large absolute values of $f'(x)$ imply a steep slope. A big step is needed to get close to the optimum. The reverse hold for small absolute value of $f'(x)$.

Role of first derivative

- If $f''(x)$ is positive the function is locally **convex**, and the search is for a local **minimum**
- To the left of this maximum $f'(x) < 0$
- Therefore $-\frac{f'(x)}{f''(x)} > 0$.
- The next step is to the right.
- The reverse holds if $f'(x) > 0$
- Large absolute values of $f'(x)$ imply a steep slope. A big step is needed to get close to the optimum. The reverse hold for small absolute value of $f'(x)$.

Role of second derivative



Role of second derivative

- Together with the sign of the first derivative, the sign of the second derivative controls the direction of the next step.
- A larger second derivative (in absolute value) implies a more curvature
- In this case smaller steps are need to stop the algorithm from overshooting.
- The opposite holds for a small second derivative.

Functions with more than one input

- Most interesting optimization problems involve **multiple** inputs.
  - In determining the most risk efficient portfolio the return is a function of many weights (one for each asset).
  - In least squares estimation for a linear regression model, the sum of squares is a function of many coefficients (one for each regressor).
- How do we optimize for functions $f(\boldsymbol{x})$ where $\boldsymbol{x}$ is a vector?

Derviatives

- Newton's algorithm has a simple update rule based on first and second derivatives.
- What do these derivatives look like when the function is $y = f(\boldsymbol{x})$ where y is a scalar and $\boldsymbol{x}$ is a $d \times 1$ vector?

First derivative

Simply take the **partial derivatives** and put them in a vector

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_d} \end{pmatrix} \tag{17}$$

This is called the **gradient** vector.

An example

The function

$$y = x_1^2 - x_1 x_2 + x_2^2 + e^{x_2} \tag{18}$$

Has gradient vector

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{pmatrix} 2x_1 - x_2 \\ -x_1 + 2x_2 + e^{x_2} \end{pmatrix} \tag{19}$$

Second derivative

Simply take the second order **partial derivatives**. This will give a matrix

$$\frac{\partial y}{\partial \mathbf{x} \partial \mathbf{x}'} = \begin{pmatrix} \frac{\partial^2 y}{\partial x_1^2} & \frac{\partial^2 y}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 y}{\partial x_1 \partial x_d} \\ \frac{\partial^2 y}{\partial x_2 \partial x_1} & \frac{\partial^2 y}{\partial x_2^2} & \cdots & \frac{\partial^2 y}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 y}{\partial x_d \partial x_1} & \frac{\partial^2 y}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 y}{\partial x_d^2} \end{pmatrix} \tag{20}$$

This is called the **Hessian** matrix.

An example

The function

$$y = x_1^2 - x_1x_2 + x_2^2 + e^{x_2} \tag{21}$$

Has Hessian matrix

$$\frac{\partial y}{\partial \mathbf{x} \partial \mathbf{x}'} = \begin{pmatrix} 2 & -1 \\ -1 & 2 + e^{x_2} \end{pmatrix} \tag{22}$$

Preliminaries for matrix derivatives I

1. The derivative of a vector $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$, by a scalar x is written (in numerator layout notation) as

$$\frac{\partial \mathbf{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \frac{\partial y_2}{\partial x} \\ \vdots \\ \frac{\partial y_m}{\partial x} \end{bmatrix}.$$

   In vector calculus the derivative of a vector y with respect to a scalar x is known as the tangent vector of the vector y, $\frac{\partial \mathbf{y}}{\partial x}$

Preliminaries for matrix derivatives II

2. **The derivative of a scalar y by a vector $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$**, is written (in numerator layout notation) as

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} & \cdots & \frac{\partial y}{\partial x_n} \end{bmatrix}.$$

3. **The second order derivatives of a scalar y by a vector $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$** is written (in numerator layout notation) as

Preliminaries for matrix derivatives III

$$\frac{\partial^2 y}{\partial \mathbf{x} \partial \mathbf{x}'} = \frac{\partial}{\partial \mathbf{x}'}\left[\frac{\partial y}{\partial \mathbf{x}}\right] = \frac{\partial}{\partial \mathbf{x}'}\left[\frac{\partial y}{\partial x_1} \quad \frac{\partial y}{\partial x_2} \quad \cdots \quad \frac{\partial y}{\partial x_n}\right]$$

$$= \begin{bmatrix} \frac{\partial^2 y}{\partial x_1^2} & \frac{\partial^2 y}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 y}{\partial x_1 \partial x_n} \\ \frac{\partial^2 y}{\partial x_2 \partial x_1} & \frac{\partial^2 y}{\partial x_2^2} & \cdots & \frac{\partial^2 y}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 y}{\partial x_m \partial x_1} & \frac{\partial^2 y}{\partial x_m \partial x_2} & \cdots & \frac{\partial^2 y}{\partial x_m \partial x_m} \end{bmatrix}.$$

4. The derivative of a vector function (a vector whose components are functions) $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$, with respect to an input vector, $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, is written (in numerator layout notation) as

Preliminaries for matrix derivatives IV

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}.$$

5. The derivative of a matrix function Y by a scalar x is known as the tangent matrix and is given (in numerator layout notation) by

$$\frac{\partial \mathbf{Y}}{\partial x} = \begin{bmatrix} \frac{\partial y_{11}}{\partial x} & \frac{\partial y_{12}}{\partial x} & \cdots & \frac{\partial y_{1n}}{\partial x} \\ \frac{\partial y_{21}}{\partial x} & \frac{\partial y_{22}}{\partial x} & \cdots & \frac{\partial y_{2n}}{\partial x} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_{m1}}{\partial x} & \frac{\partial y_{m2}}{\partial x} & \cdots & \frac{\partial y_{mn}}{\partial x} \end{bmatrix}.$$

Preliminaries for matrix derivatives V

6. The derivative of a scalar y function of a matrix X of independent variables, with respect to the matrix X, is given (in numerator layout notation) by

$$\frac{\partial y}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{21}} & \cdots & \frac{\partial y}{\partial x_{p1}} \\ \frac{\partial y}{\partial x_{12}} & \frac{\partial y}{\partial x_{22}} & \cdots & \frac{\partial y}{\partial x_{p2}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial x_{1q}} & \frac{\partial y}{\partial x_{2q}} & \cdots & \frac{\partial y}{\partial x_{pq}} \end{bmatrix}.$$

Newton's algorithm for multidimensional optimization

We can now generalise the update step in Newton's method:

$$x_{n+1} = x_n - \left(\frac{\partial^2 f(\boldsymbol{x})}{\partial \boldsymbol{x} \partial \boldsymbol{x}'}\right)^{-1} \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \tag{23}$$

Now write code to minimise $y = x_1^2 - x_1 x_2 + x_2^2 + e^{x_2}$

The linear regression model, a revisit

- Consider the linear regression model with multiple covariates,

$$y_i = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p + \epsilon_i$$

  where $\epsilon_i \sim N(0, \sigma^2)$

- What is the gradient and Hessian matrix for the log likelihood ($\mathcal{L}$) with respect to the parameter vector $\boldsymbol{\beta} = (\beta_0, ..., \beta_p)$?

$$\frac{\partial \log \mathcal{L}}{\partial \boldsymbol{\beta}} = ?$$

$$\frac{\partial^2 \log \mathcal{L}}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}'} = ?$$

Maximum likelihood Estimate for linear models

- Assume you want to make a regression model

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

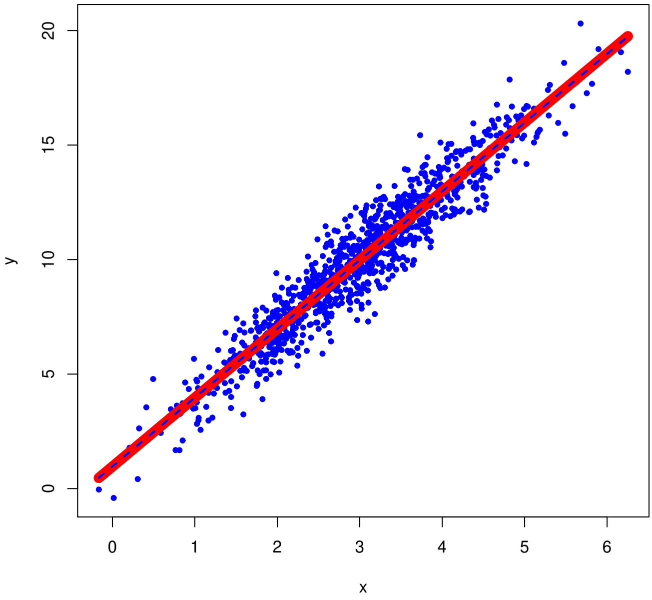
  where $\epsilon_i \sim N(0, \sigma^2)$
- What is the (log) likelihood function?
- What are the unknown parameters?
- How do we estimate the parameters? Let's consider three situations
  - When $\beta_0 = 1$ and $\sigma^2 = 1$ known.
  - When $\sigma^2 = 1$ known.
  - Neither β nor σ is known.
- Write down the likelihood function with respect to the unknown parameters.
- Write down the gradient for the likelihood function.
- Write down the Hessian for the likelihood function.
- Use Newton's method to obtain the best parameter estimate.

Optimizing the likelihood function by using optim()

```
## Generate some data
beta0 <- 1
beta1 <- 3
sigma <- 1
n <- 1000
x <- rnorm(n, 3, 1)
y <- beta0 +x*beta1 + rnorm(n, mean = 0, sd = sigma)
plot(x, y, col = "blue", pch = 20)

## The optimization
optimOut <- optim(c(0, -1, 0.1), logNormLikelihood,
                  control = list(fnscale = -1),
                  x = x, y = y)
beta0Hat <- optimOut$par[1]
beta1Hat <- optimOut$par[2]
sigmaHat <- optimOut$par[3]
yHat <- beta0Hat + beta1Hat*x

plot(x, y, pch = 20, col = "blue")
points(sort(x), yHat[order(x)], type = "l", col = "red", lwd = 2)
```

Comparison with OLS

```
myLM <- lm(y~x)
myLMCoef <- myLM$coefficients
yHatOLS <- myLMCoef[1] + myLMCoef[2]*x

plot(x, y, pch = 20, col = "blue")
points(sort(x), yHat[order(x)], type = "l", col = "red", lwd = 10)
points(sort(x), yHatOLS[order(x)], type = "l",
       col = "blue", lty="dashed", lwd = 2, pch = 20)
```



0
5
10
15
20
y
0
1
2
3
4
5
6
x

Vector Newton-Raphson Algorithm: The logit model

↬ Estimate logit model with ungrouped (individual) data

- **The idea**: using maximum likelihood method with binomial distribution.
- One owns a house ($Y = 1$) or do not own a house ($Y = 0$) can be represented with **Bernoulli distribution**

$$\Pr(y; p) = p^y (1-p)^{1-y} \quad \text{for } y \in \{0, 1\}.$$

- The log likelihood function is as follows

$$l(\beta) = \sum_{n=1}^{N} \{y_i \log P_i + (1 - y_i) \log(1 - P_i)\}$$

where

$$P_i = \frac{1}{1 + \exp(-(\beta_1 + \beta_2 X_{2i} + ... + \beta_p X_{pi}))}$$

- Note that the sum of n Bernoulli samples will be **binomial** distributed.
- To obtain $\hat{\beta}$, use Newton-Raphson algorithm

$$\beta^{new} = \beta^{old} - \left(\frac{\partial^2 l(\beta)}{\partial\beta\partial\beta'}\right)^{-1} \frac{\partial l(\beta)}{\partial\beta}|_{\beta=\beta^{old}}$$

A harder example

- Use Newton's method to find the maximum likelihood estimate for the coefficients in a logistic regression. The steps are:
  - Write down likelihood function
  - Find the gradient and Hessian matrix
  - Code these up in R
  - Simulate some data from a logistic regression model.
  - Test your code.

Quasi-Newton Methods

- One of the most difficult parts of the Newton method is working out the derivatives especially the Hessian.
- However methods can be used to approximate the Hessian and also the gradient.
- These are known as Quasi-Newton Methods
- In general they will converge slower than pure Newton methods.

The BFGS algorithm

- The BFGS algorithm was introduced over several papers by Broyden, Fletcher, Goldfarb and Shanno.
- It is the most popular Quasi-Newton algorithm.
- The R function 'optim' also has a variation called L-BFGS-B.
- The L-BFGS-B uses less computer memory than BFGS and allows for box constraints

Box Constraints

- Box constraints have the form

$$l_i \leqslant x_i \leqslant u_i \quad \forall i \tag{24}$$

- In statistics this can be very useful. Often parameters are constrained
  - Variance must be greater than 0
  - For a stationary AR(1), coefficient must be between -1 and 1
  - Weights in a portfolio must be between 0 and 1 if short selling is prohibited.

Optim function in R

- The optim function in R requires at least two inputs
  - Initial values
  - The function that needs to be optimized
- By default it **minimises** a function.
- A function that computes the gradient vector can also be provided.
- The optimization method can be set (choices include BFGS, L-BFGS-B and Nelder-Mead)
- Lower and upper bounds can be set through the arguments *lower* and *upper* if the L-BFGS-B method is used.

Optim function in R

- Further arguments can be passed in an argument called *control*.
- Some things that can be included in this list are
    - Maximum number of iterations (*maxit*)
    - Information about the algorithm (*trace*)
    - How often to display information about the algorithm (*REPORT*)

Optim function in R

- The result of *optim* can be saved in an object that is a list containing
  - The value of the function at the turning point (*value*)
  - The optimal parameters (*par*)
  - Useful information about whether the algorithm has converged (*convergence*)
- For all algorithms *convergence*=0 if the algorithm has converged (slightly confusing)

Homework

Use *optim* to carry out maximum likelihood for the

- Logistic regression model

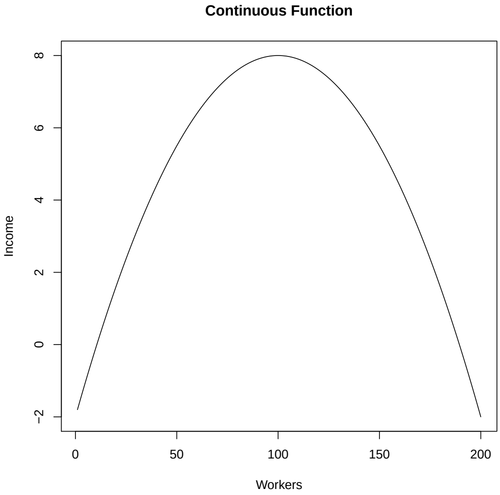
Discontinuous Functions

- The *Newton Method* requires first and second derivatives.
- If derivatives are not available the they can be approximated by *Quasi-Newton methods*
- What if the derivatives do not exist?
- This may occur if there are **discontinuities** in the function.

Business Example

- Suppose the aim is to optimize income of the business by selecting the number of workers.
- In the beginning adding more workers leads to more income for the business.
- If too many workers are employed, they may be less efficient and the income of the company goes down

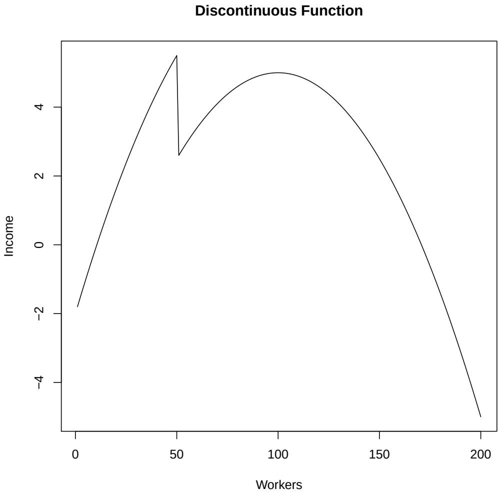
Business Example



Business Example

- Now suppose that there is a tax that the company must pay.
- Companies with less than 50 workers do not pay the tax
- Companies with more than 50 workers do pay the tax
- How does this change the problem?

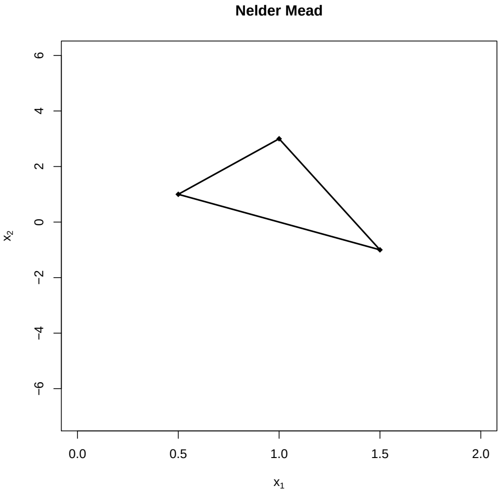
Business Example



The Nelder Mead Algorithm

- The Nelder Mead algorithm is robust even when the functions are discontinuous.
- The idea is based on evaluating the function at the vertices of an n-*dimensional simplex* where n is the number of input variables into the function.
- For two dimensional problems the n-*dimensional simplex* is simply a triangle, and each corner is one vertex
- In general there are $n + 1$ vertices.

A 2-dimensional simplex



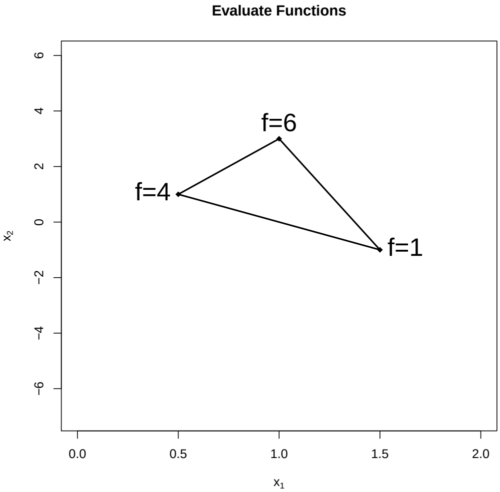
Step 1: Evaluate Function

- For each vertex x_j evaluate the function $f(x_j)$
- Order the vertices so that

$$f(x_1) \leqslant f(x_2) \leqslant \ldots \leqslant f(x_{n+1}) \tag{25}$$

- Suppose that the aim is to **minimize** the function, then $f(x_{n+1})$ is the worst point.
- The aim is to replace $f(x_{n+1})$ with a better point

A 2-dimensional simplex



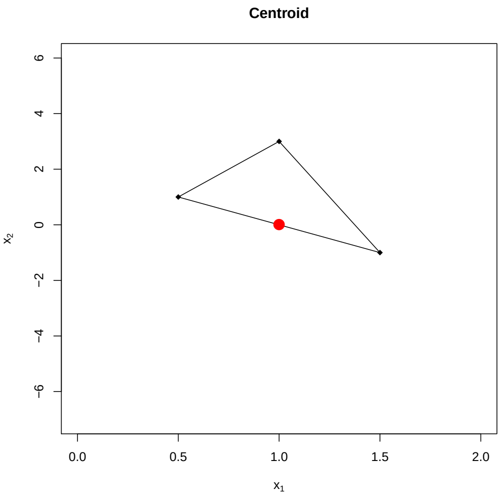
Step 2: Find Centroid

- After eliminating the worst point x_{n+1}, compute the **centroid** of the remaining n points

$$x_0 = \frac{1}{n}\sum_{j=1}^{n} x_j \tag{26}$$

- For the 2-dimensional example the centroid will be in the middle of a line.

Find Centroid



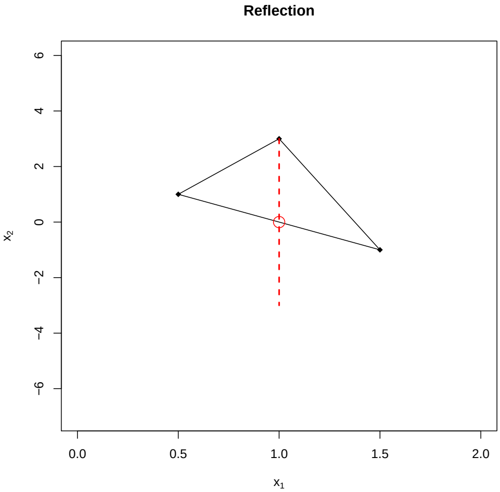
Step 3: Find reflected point

- Reflect the worst point around the centroid to get the **reflected point**.
- The formula is:

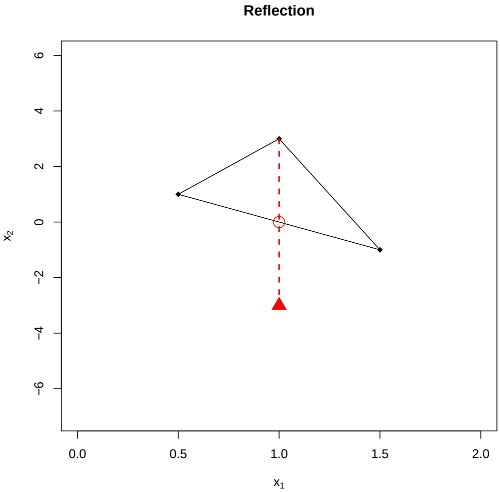
$$\mathbf{x}_r = \mathbf{x}_0 + \alpha(\mathbf{x}_0 - \mathbf{x}_{n+1}) \tag{27}$$

- A common choice is $\alpha = 1$.
- In this case the reflected point is the same distance from the centroid as the worst point.

Find Reflected point



Find Reflected point



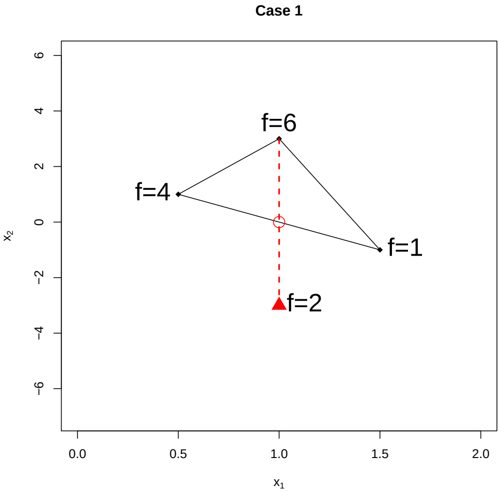
Three cases

1. $f(x_1) \leqslant f(x_r) < f(x_n)$
   - x_r is neither best nor worst point
2. $f(x_r) < f(x_1)$
   - x_r is the best point
3. $f(x_r) \geqslant f(x_n)$
   - x_r is the worst point

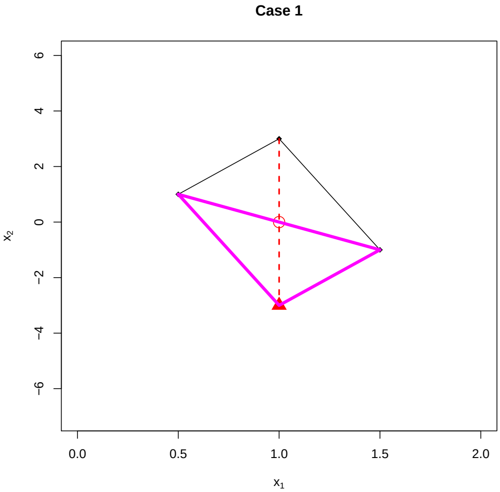
Case 1

In Case 1 a new simplex is formed with $\boldsymbol{x}_{n+1}$ replaced by the reflected point $\boldsymbol{x}_r$. Then go back to step 1.

Case 1



Case 1



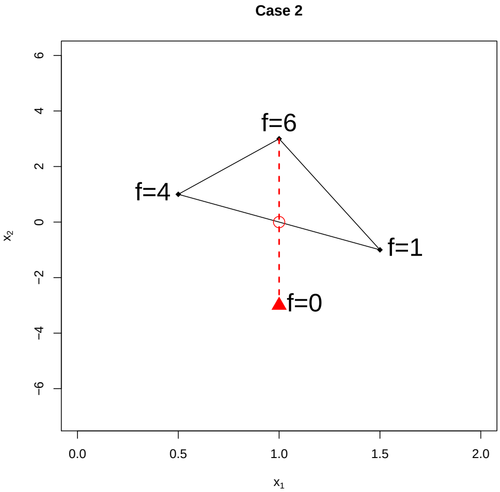
Case 2

In Case 2, $x_r < x_1$. A good direction has been found so we **expand** along that direction

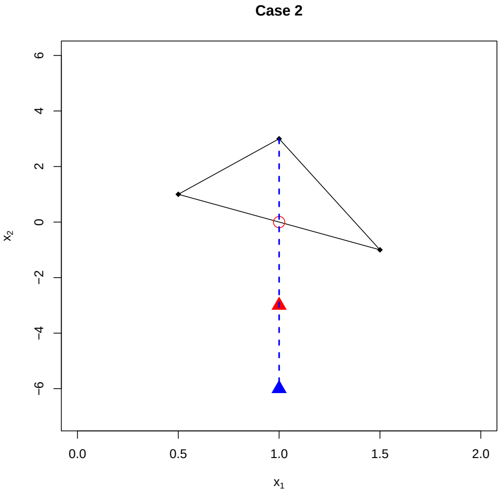
$$x_e = x_0 + \gamma(x_r - x_0) \tag{28}$$

A common choice is $\gamma = 2$

Case 2



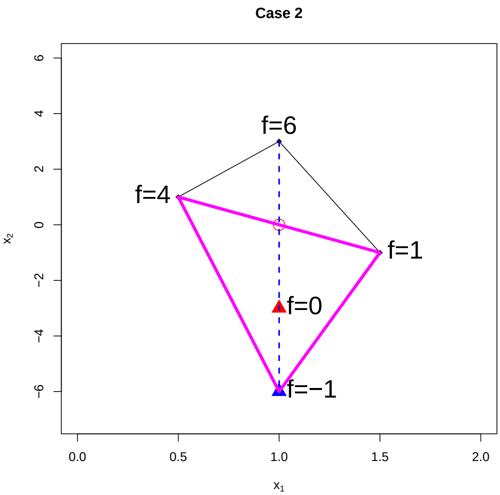
Case 2



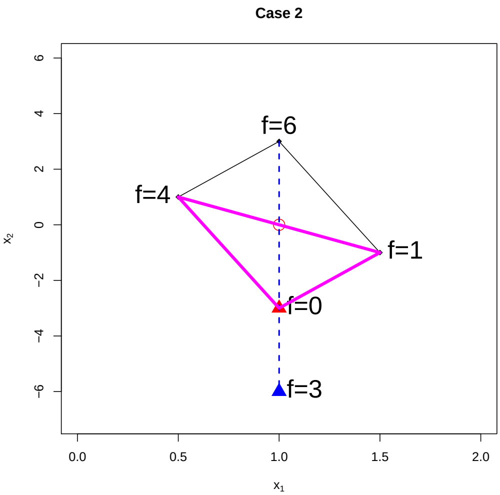
Choosing the expansion point

- Evaluate $f(\boldsymbol{x}_e)$.
- If $f(\boldsymbol{x}_e) < f(\boldsymbol{x}_r)$:
  - The expansion point is better than the reflection point. Form a new simplex with the expansion point
- If $f(\boldsymbol{x}_r) \leqslant f(\boldsymbol{x}_e)$:
  - The expansion point is not better than the reflection point. Form a new simplex with the reflection point.

Keep expansion point



Keep reflection point



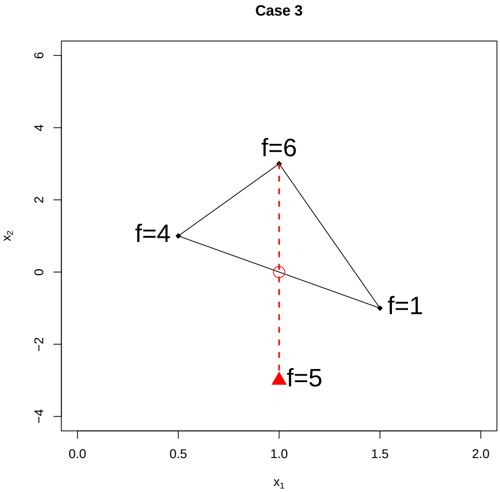
Case 3

Case 3 implies that there may be a valley between x_{n+1} and x_r so find the **contracted** point. A new simplex is formed with the contraction point if it is better than x_{n+1}

$$x_c = x_0 + \rho(x_{n+1} - x_0) \tag{29}$$

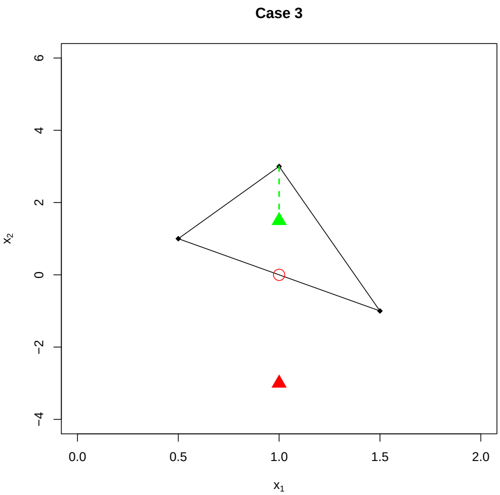
A common choice is $\rho = 0.5$

Case 3

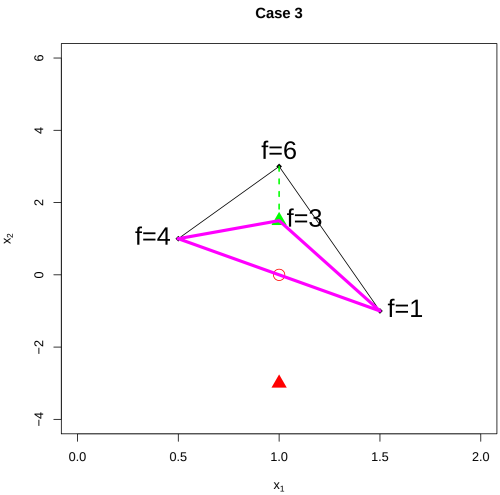


‘Valley’

Find Contraction point



New Simplex



Shrink

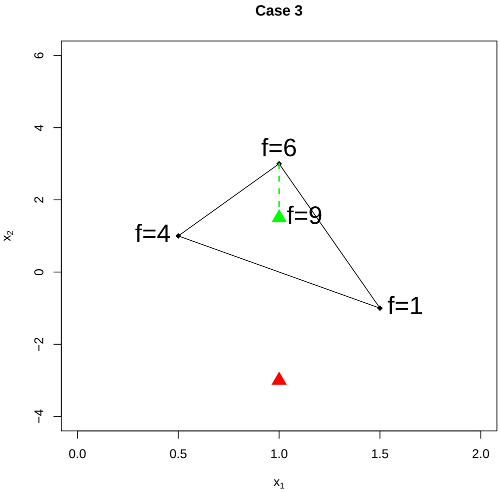
If $f(x_{n+1}) \leqslant f(x_c)$ then contracting away from the worst point does not lead to a better point. In this case the function is too irregular a smaller simplex should be used. Shrink the simplex

$$x_i = x_1 + \sigma(x_i - x_1) \tag{30}$$

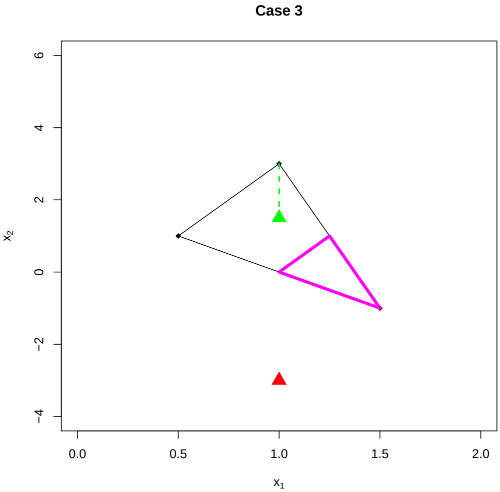
A popular choice is $\sigma = 0.5$

'Egg Carton'

Contraction Point is worst



New Simplex



Summary

- Order points
- Find centroid
- Find reflected point
- Three cases:
  1. Case 1 ($f(x_1) \leqslant f(x_r) < f(x_n)$): Keep x_r
  2. Case 2 ($f(x_r) < f(x_1)$): Find x_e.
     - If $f(x_e) < f(x_r)$ then keep $f(x_e)$
     - Otherwise keep $f(x_r)$
  3. Case 3 ($f(x_r) \geqslant f(x_n)$): Find $f(x_c)$
     - If $f(x_c) < f(x_{n+1})$ then keep $f(x_c)$
     - Otherwise Shrink

Your task

- Find the minimum of the function $f(\boldsymbol{x}) = x_1^2 + x_2^2$
- Use a triangle with vertices $(1, 1)$, $(1, 2)$, $(2, 2)$ as the starting simplex
- Don't worry about using a loop just yet. Try to get code that just does the first iteration.
- Don't worry about the stopping rule yet either

Use pseudo-code

Algorithm 1 Nelder Mead

Set initial simplex and evaluate function
Sort $f(x_1) \leqslant \ldots \leqslant f(x_n)$
Compute **reflected** point
if $f(x_1) \leqslant f(x_r) < f(x_n)$ **then**
    return $x_{n+1} \leftarrow x_r$
else if $f(x_r) < f(x_1)$ **then**
    Compute **expanded** point
    if $f(x_e) < f(x_r)$ **then**
        return $x_{n+1} \leftarrow x_e$
    else if $f(x_r) \leqslant f(x_e)$ **then**
        return $x_{n+1} \leftarrow x_r$
    end if
else if $f(x_n) \leqslant f(x_r)$ **then**
    Compute **contracted** point
end if

Lessons

- Break down a difficult problem into smaller problems.
- Use pseudo code in planning
- Use comments
- Use indents

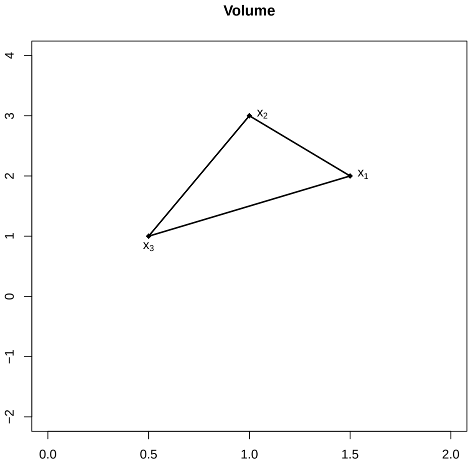
Stopping Rule for Nelder Mead

- As Nelder Mead gets close to (or reaches) the minimum, the simplex gets smaller and smaller.
- One way to know that Nelder Mead has converged is by looking at the volume of the simplex.
- To work out the volume requires some understanding between the relationship between matrix algebra and geometry.

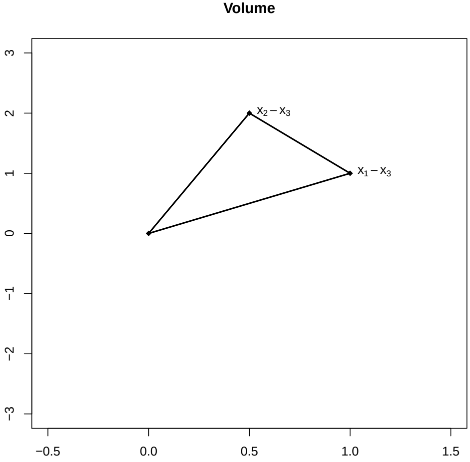
Stopping Rule for Nelder Mead

- Choose the vertex $\mathbf{x}_{n+1}$ (although choosing any other vertex will also work)
- Build the matrix $\tilde{X} = (\mathbf{x}_1 - \mathbf{x}_{n+1}, \mathbf{x}_2 - \mathbf{x}_{n+1}, \ldots, \mathbf{x}_n - \mathbf{x}_{n+1})$
- The volume of the simplex is $\frac{1}{2}|det(\tilde{X})|$

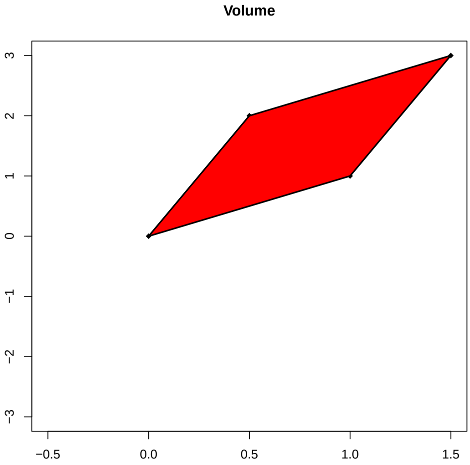
Why?



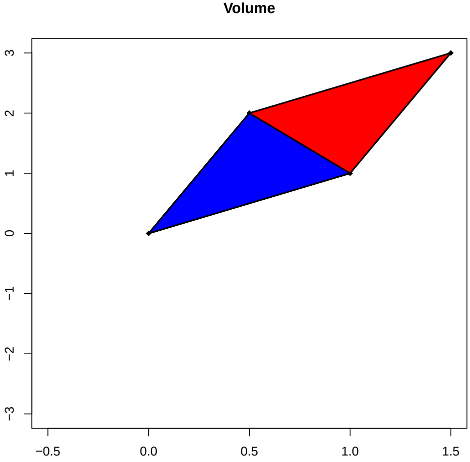
Translate



Determinant=Area of Trapezoid



Triangle=Half Trapezoid



Alternative formula

Some of you may have learnt the formula for the area of a triangle as:

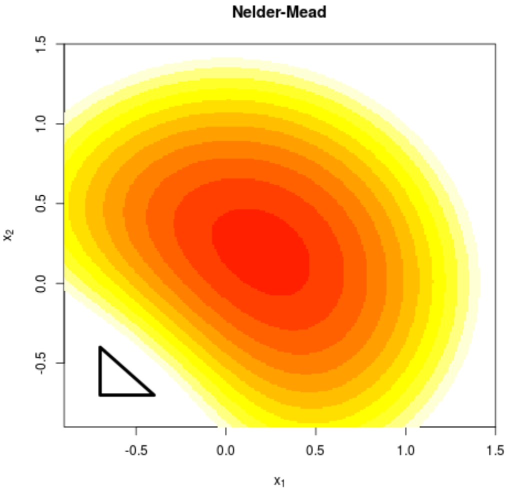
$$\frac{1}{2}\left|\det\begin{pmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \boldsymbol{x}_3 \\ 1 & 1 & 1 \end{pmatrix}\right| \tag{31}$$

The two approaches are equivalent.

Questions

- In my code, where should I start the loop?
- Should it be a *for* loop or a *while* loop?
- What should the loop look like?

Visualizing Nelder Mead



Nelder Mead in ‘optim’

- Nelder Mead is the **default** algorithm in the R function *optim*
- It is generally slower than Newton and Quasi-Newton methods but is more stable for functions that are not smooth.
- Including the argument control=list(trace, REPORT=1) will print out details about each step of the algorithm.
- Slight different terminology is used for example ‘expansion’ is called ‘extension’

Box constraints in Nelder Mead

- It is not possible to impose box constraints in Nelder Mead.
- However it is possible to trick R. How?
- Suppose the problem is a minimization. We can use an *if* statement to force the function to be extremely large outside the box.
- This is not an option in BFGS since this induces a discontinuity in the function.

Some test functions

Use both Nelder Mead and L-BFGS-B to minimize the following

- Booth's Function:

$$f(\boldsymbol{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 \quad -10 \leqslant x_1, x_2 \leqslant 10$$

- Bukin Function N.6

$$f(\boldsymbol{x}) = 100\sqrt{\left|x_2 - \frac{x_1^2}{100}\right|} + \frac{|x_1 + 10|}{100} \qquad \begin{array}{c} -15 \leqslant x_1 \leqslant 5 \\ -3 \leqslant x_2 \leqslant 3 \end{array}$$

Summary

- This is the end of the optimization topic.
- You should now be familiar with
  - Newton's Method
  - Quasi Newton Method
  - Nelder Mead
- Hopefully you also improved your coding skills!

Summary

- Some important lessons:
  - If you can evaluate derivatives and Hessians then do so when implementing Newton and Quasi-Newton methods.
  - If there are discontinuities in the function then Nelder Mead may work better.
  - In any case the best strategy is to optimize using **more than one method** to check that results are robust.
  - Also pay special attention to **starting values**. A good strategy is to check that results are robust to a few different choices of starting values.