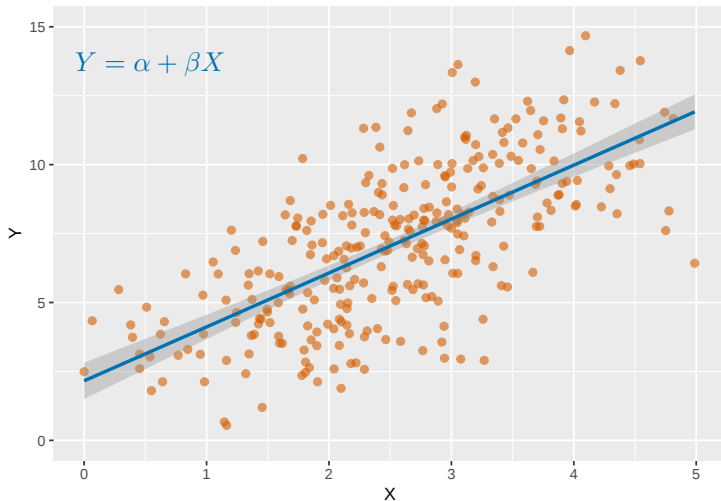
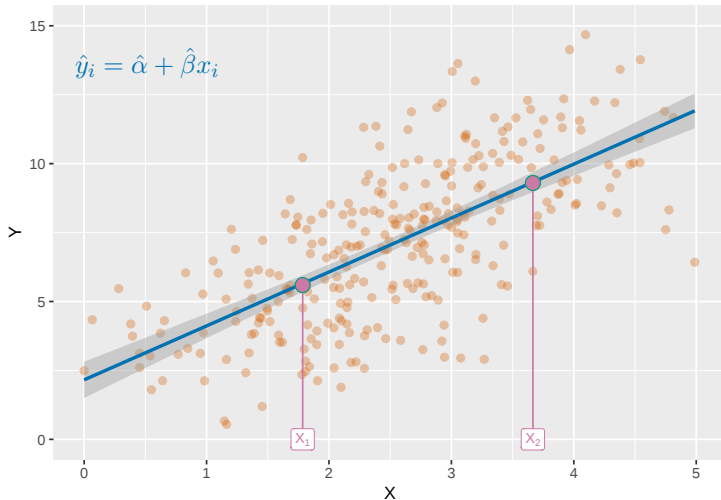


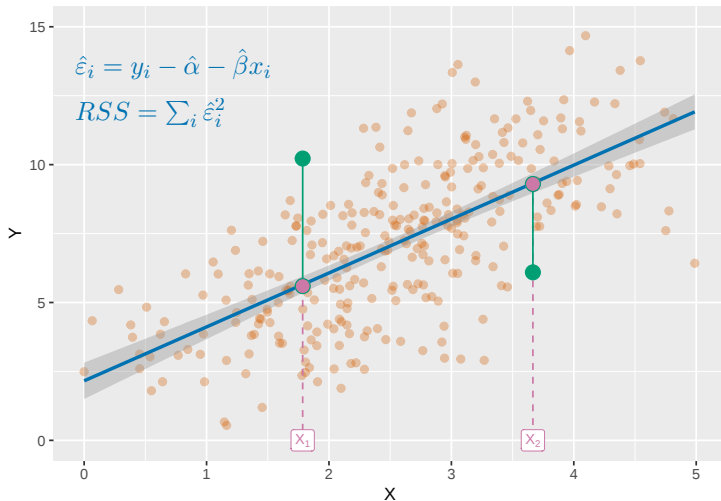
There's Nothing Ordinary About OLS



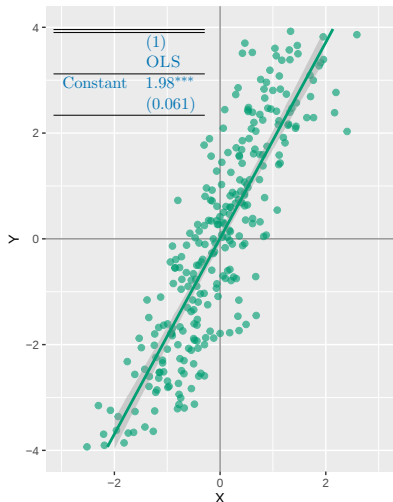
OLS Generates Predicted Values of Y



OLS Coefficients Minimize the Sum of Squared Residuals



$\hat{\beta}_{ols}$ Minimizes the Residual Sum of Squares (RSS)



$$RSS = \sum_i \varepsilon_i^2 = \sum_i (y_i - \hat{\beta}x_i)^2$$

$$\Leftrightarrow \frac{\partial RSS}{\partial \beta} = -\sum_i 2y_ix_i + \sum_i 2\hat{\beta}x_i^2$$

Solving the first-order condition for $\hat{\beta}$:

$$\frac{\partial RSS}{\partial \beta} = 0 \Leftrightarrow \hat{\beta}_{ols} = \sum_i y_i x_i / \sum_i x_i^2$$

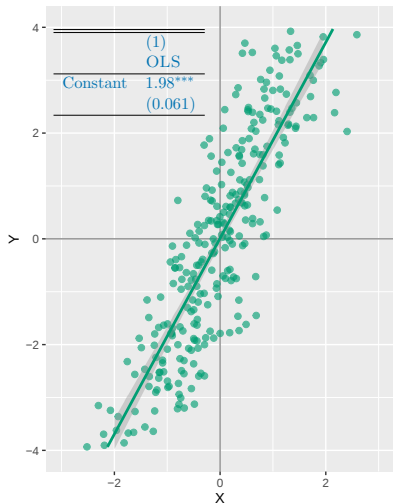
Generalizes to bivariate OLS with a constant:

$$\hat{\beta}_{ols} = \frac{COV(X,Y)}{VAR(X)} = \sum_i y_i (x_i - \bar{x}) / \sum_i (x_i - \bar{x})^2$$

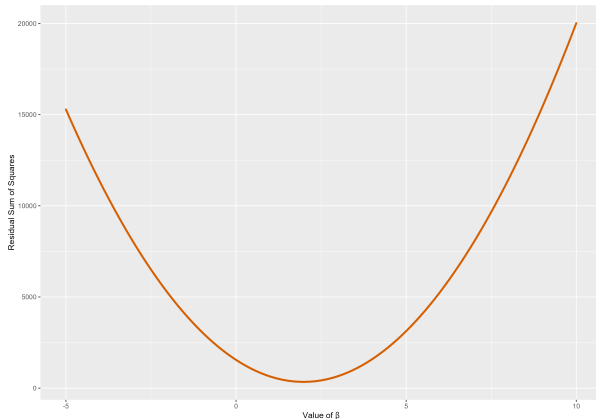
Generalizes to multivariate regression case:

$$\hat{\beta}_{ols} = (X'X)^{-1}X'Y$$

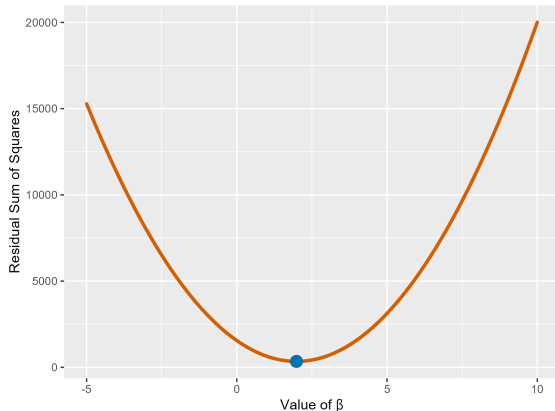
A Numerical Approach to OLS: Find the β that Minimizes RSS



$$RSS(\beta) = \sum_i [\varepsilon_i(\beta)]^2 = \sum_i [y_i - (\beta)x_i]^2$$



Grid Search

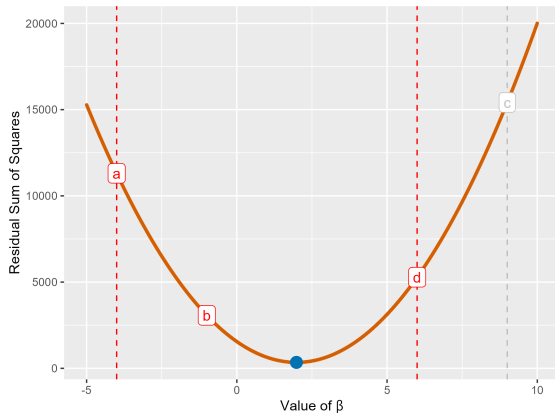


Simplest approach to choosing β :
calculate $RSS(\beta)$ for a range of β s
(sufficiently wide search window),
choose β that yields lowest RSS

Grid search is computationally costly,
not feasible in many dimensions

Grid search can still be useful,
especially when objective function
is not smooth and/or convex or
parameters are near the boundaries

Golden Search



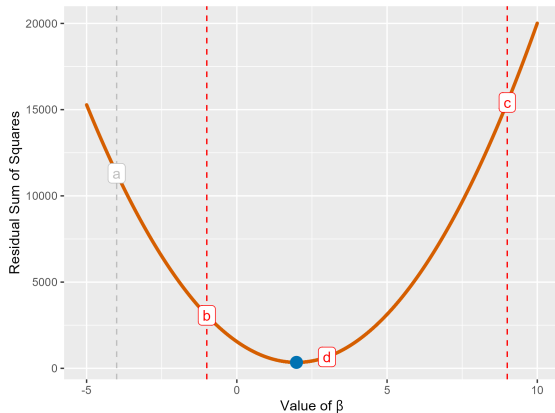
Suppose $a < b < c$, but
 $f(b) < f(a)$ and $f(b) < f(c)$

$\Rightarrow$ minimum is between a and c

Choose d between b and c

If $f(d) > f(b)$, then
the minimum is between a and d

Golden Search



Suppose $a < b < c$, but
 $f(b) < f(a)$ and $f(b) < f(c)$

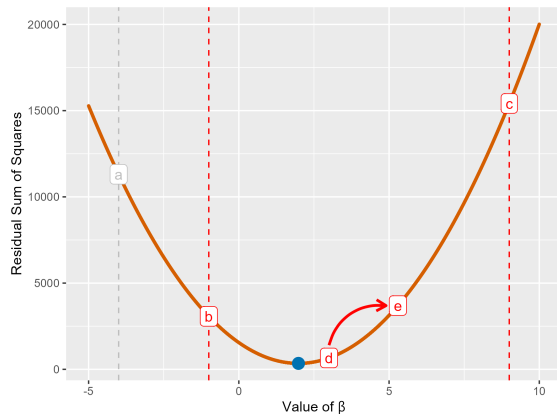
$\Rightarrow$ minimum is between a and c

Choose d between b and c

If $f(d) > f(b)$, then
the minimum is between a and d

If $f(d) < f(b)$, then
the minimum is between b and c

Golden Search

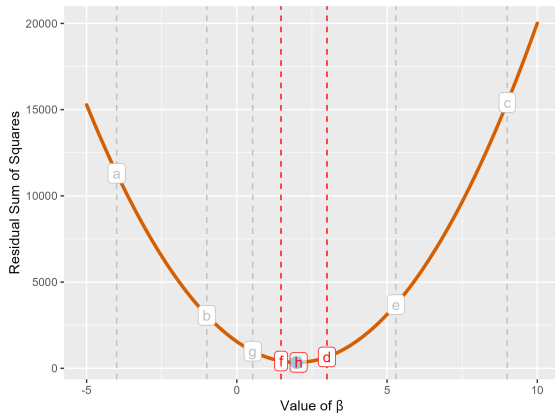


Why is it called Golden Search?

Optimal step size as a fraction of distance from midpoint to far end:

$$\frac{3-\sqrt{5}}{2} \approx 0.382 \text{ ("the golden mean")}$$

Golden Search



Why is it called Golden Search?

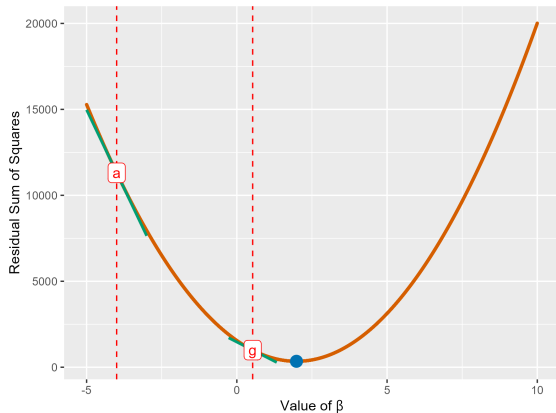
Optimal step size as a fraction of distance from midpoint to far end:

$$\frac{3-\sqrt{5}}{2} \approx 0.382 \text{ ("the golden mean")}$$

Stopping rule: when $f(x)$ converges

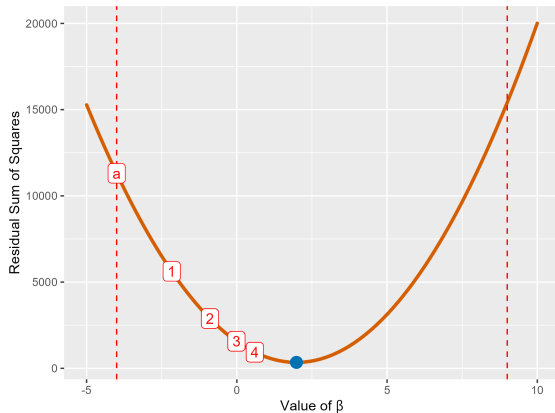
Here, **convergence** means that the optimal β is bounded in a sufficiently narrow window

Gradient Descent



Use the slope (i.e. the **gradient**) to decide direction to go and how far

Gradient Descent



Use the slope (i.e. the **gradient**) to decide direction to go and how far

$$x_1 = x_0 - \alpha [f'(x_0)]$$

Steps get smaller and smaller as you approach the minimum of $RSS(\beta)$

Importantly: generalizes to vector β s

Numerical Minimization in Multiple Dimensions

In multiple dimensions, algorithms define direction and step size, and convergence occurs when subsequent values of the function being optimized are sufficiently close together (tolerance)

- **Gradient descent:** from an arbitrary starting point, steps in parameter space are proportional to the negative of the gradient (i.e. the vector of partial derivatives)
 - ▶ Can be slow because it sometimes requires a large number of steps to reach convergence
- **Newton's method:** from an arbitrary starting point, steps are minus one times the ratio of the gradient to the Hessian (i.e. the matrix of second derivatives including cross partials)
 - ▶ Can be slow because of the time required to calculate the Hessian
- **Broyden-Fletcher-Goldfarb-Shanno (BFGS):** a quasi-Newton method, similar to Newton's method but using a less computationally-intensive approximation of the Hessian

Numerical Approaches to OLS

$$Y = \beta_0 \cdot \mathbf{1} + \beta_1 X_1 + \beta_2 X_2$$

$$\begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ \vdots & \vdots & \vdots \\ 1 & x_{N1} & x_{N2} \end{bmatrix} = X$$

Numerical Approaches to OLS

$$Y = \beta_0 \cdot \mathbf{1} + \beta_1 X_1 + \beta_2 X_2$$

$$\begin{aligned}\hat{\beta}_{ols} &= (\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2) \\ &= (X'X)^{-1}X'Y\end{aligned}$$

$$\begin{aligned}RSS(\beta_0, \beta_1, \beta_2|X) &= \sum_i (y_i - \hat{y}_i(\beta_0, \beta_1, \beta_2|X))^2 \\ &= \sum_i (y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2})^2\end{aligned}$$

1. Choose starting values of β : $(0, 0, 0)$
 2. Update β in the direction that lowers RSS
 3. Stop when process reaches convergence
- Chosen values of β should match OLS

Summary

- OLS coefficients are chosen to minimize the RSS
- OLS can be framed as a numerical minimization problem
- We can find OLS coefficients using standard tools for numerical optimization
- Aside: OLS also solves the maximum likelihood estimation problem with normal residuals

Lab #4

The objective of this lab is to familiarize ourselves with: simulating data-generating processes, the mechanics of linear regression (again!), defining functions, and numerical optimization

1. Simulate a data set
2. Run OLS
3. Calculate OLS coefficients “by hand” (using the formula) in the bivariate, no-constant case
4. Write a function to calculate the RSS and find $\hat{\beta}_{ols}$ through grid search
5. Find $\hat{\beta}_{ols}$ through numerical optimization by finding the minimum of $RSS(\beta)$

Numerical Minimization in Practice: Pseudo-Random Numbers

R:

```
set.seed(8675309)  
A <- matrix(rnorm(4*3), 4, 3)
```

$$A = \begin{bmatrix} -0.996 & 1.065 & 0.572 \\ 0.721 & 0.987 & 0.903 \\ -0.617 & 0.027 & -1.549 \\ 2.029 & 0.672 & 1.022 \end{bmatrix}$$

Python:

```
import numpy as np  
np.random.seed(8675309)  
B = np.random.randn(4, 3)
```

$$B = \begin{bmatrix} 0.589 & 0.733 & -1.162 \\ -0.556 & -0.772 & -0.168 \\ -0.416 & -1.378 & 0.749 \\ 0.178 & 0.694 & -1.978 \end{bmatrix}$$

Numerical Minimization in Practice: Functions (RSS)

R:

```
RSS <- function(beta){  
  yhat <- X %*% beta  
  return(sum((Y - yhat)^2))  
}
```

Python:

```
def RSS(beta):  
    yhat = X @ beta  
    return np.sum((Y - yhat) ** 2)
```

%*% and **@** indicate matrix multiplication:

$$\begin{matrix} \begin{bmatrix} a & b & c \end{bmatrix} \\ 1 \times 3 \end{matrix} \begin{matrix} \begin{bmatrix} d \\ e \\ f \end{bmatrix} \\ 3 \times 1 \end{matrix} = \begin{matrix} \begin{bmatrix} ad + be + cf \end{bmatrix} \\ 1 \times 1 \end{matrix}$$

$$\begin{matrix} \begin{bmatrix} a & b & c \\ A & B & C \end{bmatrix} \\ 2 \times 3 \end{matrix} \begin{matrix} \begin{bmatrix} d \\ e \\ f \end{bmatrix} \\ 3 \times 1 \end{matrix} = \begin{matrix} \begin{bmatrix} ad + be + cf \\ Ad + Be + Cf \end{bmatrix} \\ 2 \times 1 \end{matrix}$$

Numerical Minimization in Practice: Functions (RSS)

R:

```
RSS <- function(beta){  
  yhat <- X %*% beta  
  return(sum((Y - yhat)^2))  
}
```

Python:

```
def RSS(beta):  
  yhat = X @ beta  
  return np.sum((Y - yhat) ** 2)
```

%*% and **@** indicate matrix multiplication:

$$\begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ 1 & x_{31} & x_{32} \\ 1 & x_{41} & x_{42} \end{bmatrix}_{4 \times 3} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}_{3 \times 1} = \begin{bmatrix} \beta_0 + \beta_1 x_{11} + \beta_2 x_{12} \\ \beta_0 + \beta_1 x_{21} + \beta_2 x_{22} \\ \beta_0 + \beta_1 x_{31} + \beta_2 x_{32} \\ \beta_0 + \beta_1 x_{41} + \beta_2 x_{42} \end{bmatrix} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \hat{y}_4 \end{bmatrix}_{4 \times 1}$$

Numerical Minimization in Practice: Numerical Optimization

R:

```
b0 <- rep(0, numvars + 1)
result <- optim( b0, RSS, method = "BFGS", control = list(maxit = 1e5) )
```

numerical optimization function

function to minimize

starting values of parameters

method

maximum number of iterations

Python:

```
from scipy.optimize import minimize
b0 = np.zeros(numvars + 1)
result = minimize( RSS, b0, method='BFGS', options={'maxiter': int(1e5)} )
```