Stats 102B - Week 2, Lecture 2

Miles Chen, PhD

Department of Statistics

Week 2 Wednesday

Section 1

Gradient Descent

So far, we've been using calculus

So far, our loss function has been the mean squared error.

$$\mathcal{L} = \frac{1}{N} \sum (t_n - f(x_n))^2$$

To find the minimum, we calculated the derivative, set it to zero, and solved. This loss function is relatively simple and it has been relatively easy to find its derivative, even in matrix notation.

At other times, however, the loss function can be more complicated. Even if we can find the derivative, figuring out where the derivative is equal to zero might not be possible.

In these cases, we can rely on some computational methods to help us locate a local minimum.

Computational methods for multidimensional optimization

When fitting our model, our goal is to find the set of parameter estimates that minimize the loss function for our observed data.

We treat the data as fixed. The loss function is a function of the parameters we need to estimate. For linear regression, there are only two parameters to estimate (slope, intercept), but for more complex models there will be even more parameters.

In 102A, we covered coordinate descent to optimize a function of multiple variables.

In this class, we'll discuss gradient descent.

Gradient Descent

For a function of a single variable, the derivative represents the slope of the tangent line to the graph of the function at some point x.

For a function of several variables, the gradient is the direction of the greatest increase of the function at some point $\mathbf{x}$. The magnitude of the gradient is the rate of increase in that direction.

The idea behind gradient descent is that if we are at some location, we can use the gradient to find the direction of greatest increase. We then move in the direction opposite of the gradient. This will point us 'downhill'. If we repeat this process, we should reach the 'bottom' of the hill (or at least the bottom of some valley).

Gradient Descent Algorithm

The gradient descent algorithm is incredibly simple.

We have some multidimensional function $F(\mathbf{w})$. (Like our loss function)

Given that we are currently at some point $\mathbf{w}_n$, we choose our next value with

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma \nabla F(\mathbf{w}_n)$$

where gamma γ is some positive constant representing a scaling factor (sometimes lambda λ is used), and $\nabla F(\mathbf{w}_n)$ is the gradient of F at location $\mathbf{w}_n$.

Gradient Descent Example on an Arbitrary Function

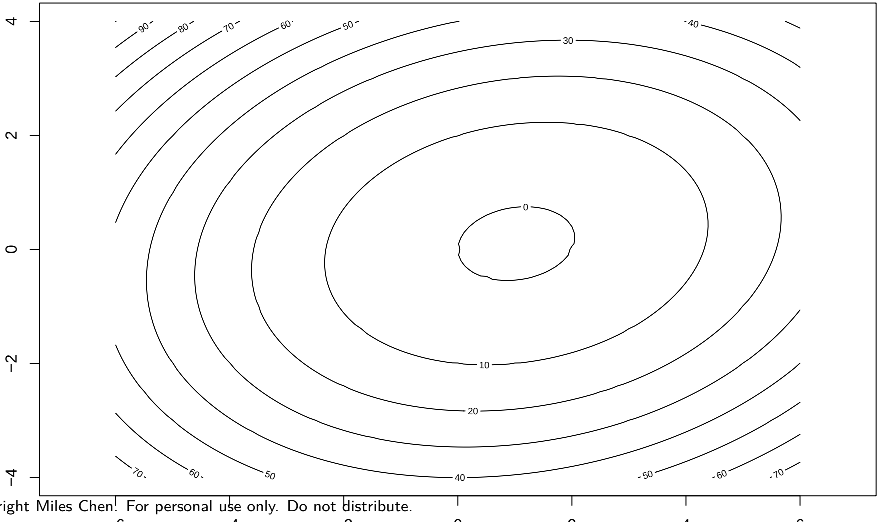
Let F be an arbitrary two-dimensional function:

$$F(\mathbf{x}) = F(x_1, x_2) = x_1^2 - 2x_1 - .5x_1x_2 + 2.5x_2^2$$

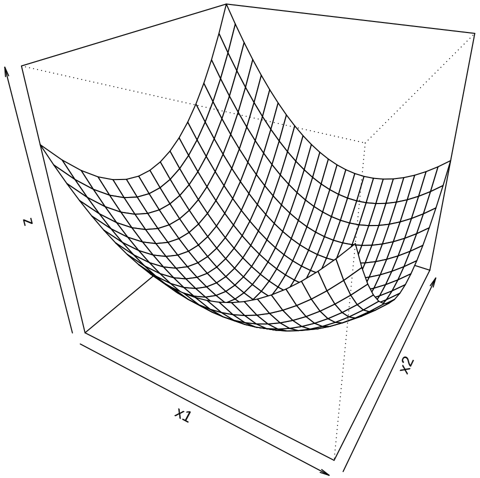
The gradient of F can be found analytically using calculus:

$$\nabla F = \left[\frac{\partial F}{\partial x_1}, \frac{\partial F}{\partial x_2}\right] = [2x_1 - 2 - 0.5x_2, -0.5x_1 + 5x_2]$$

Contour Plot $F = x_1^2 - 2x_1 - .5x_1x_2 + 2.5x_2^2$



3D Perspective Plot



Gradient Calculation for a Point

At time 0, let's say we begin at an arbitrary location: $(x_1, x_2)_{(0)} = (-5, 4)$.

The gradient at this location is:

$$\begin{aligned}
\nabla F &= [2x_1 - 2 - 0.5x_2, & -0.5x_1 + 5x_2] \\
\nabla F(-5,4) &= [2(-5) - 2 - 0.5(4), & -0.5(-5) + 5(4)] \\
\nabla F(-5,4) &= [-14, & 22.5]
\end{aligned}$$

If we use a step size of $\gamma = 0.1$, our next location will be:

$$\begin{aligned}
(x_1, x_2)_{(1)} &= (x_1, x_2)_{(0)} - \gamma \nabla F \\
(x_1, x_2)_{(1)} &= (-5, 4) - 0.1 \times (-14, 22.5) \\
(x_1, x_2)_{(1)} &= (-5, 4) + (1.4, -2.25) \\
(x_1, x_2)_{(1)} &= (-3.6, 1.75)
\end{aligned}$$

Code to represent the function and the gradient

$$F(\mathbf{x}) = F(x_1, x_2) = x_1^2 - 2x_1 - .5x_1x_2 + 2.5x_2^2$$

$$\nabla F = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}\right] = [2x_1 - 2 - 0.5x_2, -0.5x_1 + 5x_2]$$

```
f <- function(x1, x2){ x1 ^ 2 - 2 * x1 - .5 * x1 * x2 + 2.5 * x2 ^ 2 }

gradf <- function(x1, x2){
  dfdx1 <- 2 * x1 - 2 - 0.5 * x2
  dfdx2 <- -0.5 * x1 + 5 * x2
  return(c(dfdx1, dfdx2))
}
```

Calculations

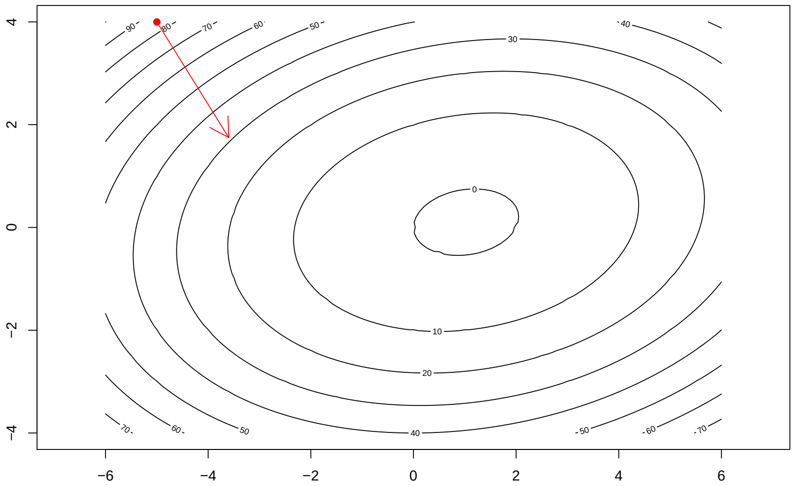
```r
start <- c(-5, 4)
gamma <- 0.1
gradf(start[1], start[2])
```

```
## [1] -14.0  22.5
```

```r
end <- start - gamma * gradf(start[1], start[2])
print(end)
```

```
## [1] -3.60  1.75
```

Plot of one iteration



Calculations at the next iteration

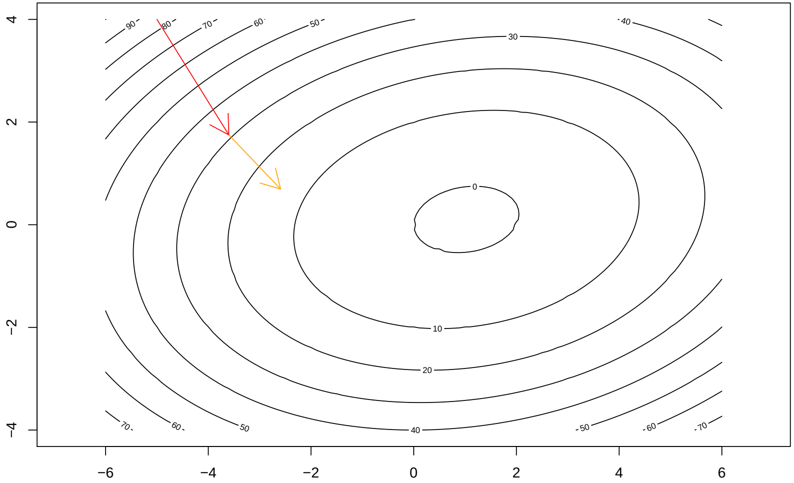
```r
start <- c(-3.6, 1.75); gamma <- 0.1
gradf(start[1], start[2])
```

```
## [1] -10.075  10.550
```

```r
end <- start - gamma * gradf(start[1], start[2])
print(end)
```

```
## [1] -2.5925  0.6950
```

Plot of the second iteration (orange)

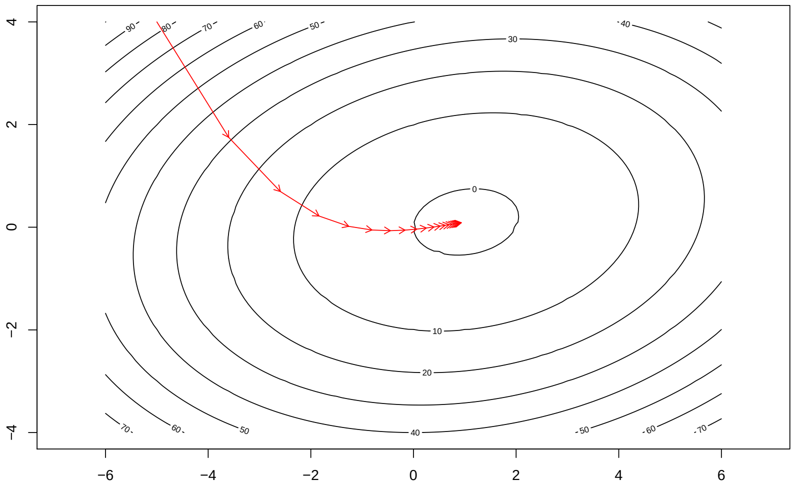


Code a loop of twenty iterations

```r
locations <- matrix(NA, nrow = 20, ncol = 2)
x_vec <- c(-5, 4)  # start location
for(i in 1:20){
  locations[i,] <- x_vec
  x_vec <- x_vec - gamma * gradf(x_vec[1], x_vec[2])
}
```

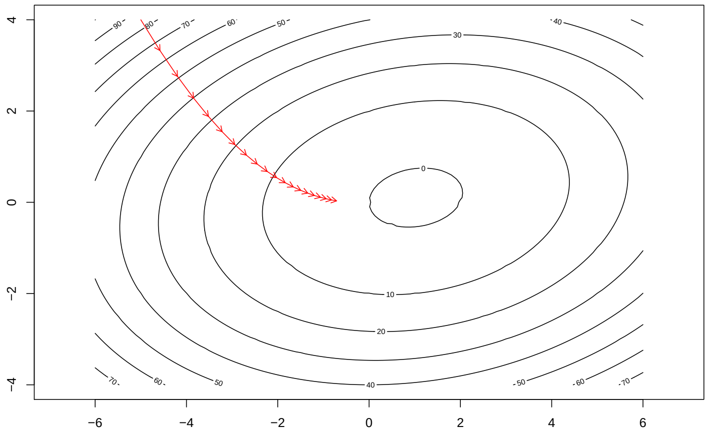
```r
contour(x1, x2, z = grid, asp = 1)
for(i in 1:19){
  arrows(locations[  i, 1], locations[  i, 2],
         locations[i+1, 1], locations[i+1, 2], col = 'red', length = 0.08)}
```

Plot the iterations



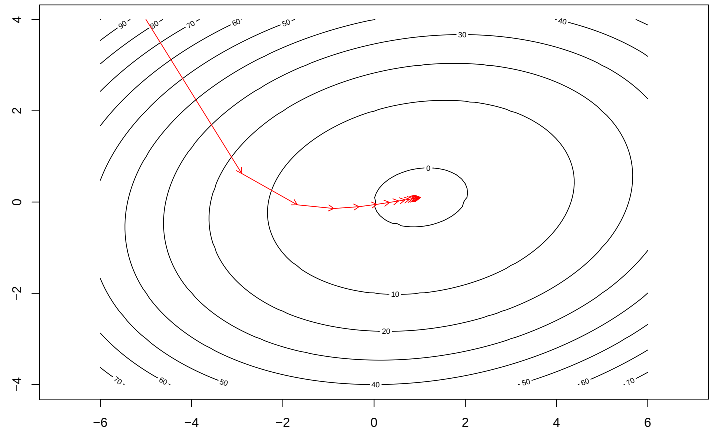
Choice of gamma

```
gamma = 0.03 # smaller steps takes more iterations
```



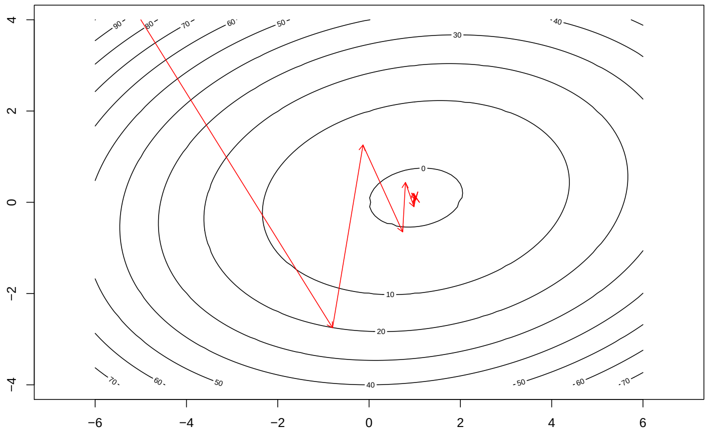
Choice of gamma

```r
gamma = 0.15
```



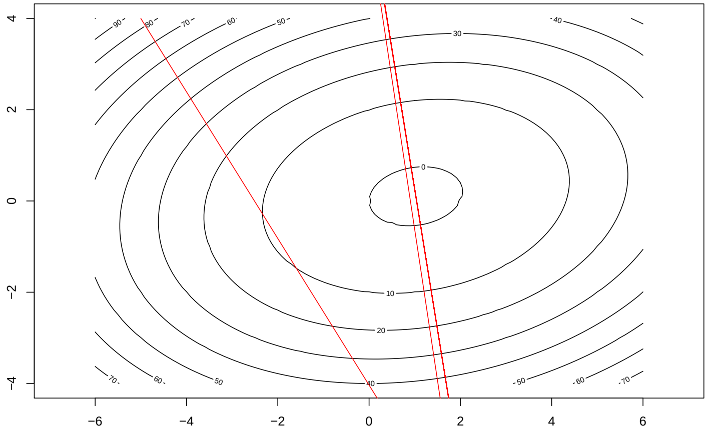
Choice of gamma

```r
gamma = 0.3 # steps that are too big can be inefficient
```



Choice of gamma

```r
gamma = 0.5 # steps that are too big can overshoot and fail to converge
```



Section 2

Gradient Descent for OLS Regression

Gradient Descent for OLS Regression

For OLS Regression, we wish to minimize the loss function, which is the mean squared residual.

The loss function can be written with a sum

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} (t_n - f(x_n))^2$$

It can also be written in matrix notation:

$$\mathcal{L} = \frac{1}{N} (\mathbf{t} - \mathbf{X}\mathbf{w})^T (\mathbf{t} - \mathbf{X}\mathbf{w})$$

Finding the Gradient Analytically (non Matrix notation)

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}(t_n - f(x_n))^2$$

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}(t_n - (w_0 + w_1 x_n))^2$$

$$\frac{\partial \mathcal{L}}{\partial w_0} = \frac{\partial}{\partial w_0}\frac{1}{N}\sum_{n=1}^{N}(t_n - w_0 - w_1 x_n)^2$$

$$= \frac{1}{N}\sum_{n=1}^{N} 2 \cdot (t_n - w_0 - w_1 x_n) \cdot -1$$

$$\frac{\partial \mathcal{L}}{\partial w_0} = \frac{-2}{N}\sum_{n=1}^{N}(t_n - w_0 - w_1 x_n)$$

Finding the Gradient Analytically (non Matrix notation)

$$
\begin{aligned}
\mathcal{L} &= \frac{1}{N}\sum_{n=1}^{N}(t_n - (w_0 + w_1 x_n))^2 \\
\frac{\partial \mathcal{L}}{\partial w_1} &= \frac{\partial}{\partial w_1}\frac{1}{N}\sum_{n=1}^{N}(t_n - w_0 - w_1 x_n)^2 \\
&= \frac{1}{N}\sum_{n=1}^{N} 2 \cdot (t_n - w_0 - w_1 x_n) \cdot -x_n \\
\frac{\partial \mathcal{L}}{\partial w_1} &= \frac{-2}{N}\sum_{n=1}^{N} x_n (t_n - w_0 - w_1 x_n)
\end{aligned}
$$

Finding the Gradient Analytically (Matrix notation)

$$\mathcal{L} = \frac{1}{N}(\mathbf{t} - \mathbf{X}\mathbf{w})^T(\mathbf{t} - \mathbf{X}\mathbf{w})$$

$$\mathcal{L} = \frac{1}{N}(\mathbf{t}^T\mathbf{t} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{t} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w})$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{-2}{N}\mathbf{X}^T\mathbf{t} + \frac{2}{N}\mathbf{X}^T\mathbf{X}\mathbf{w}$$

The Gradient Descent Algorithm

Non-matrix notation:

$$(w_0, w_1)_{(n+1)} = (w_0, w_1)_{(n)} - \gamma \cdot \left(\frac{\partial \mathcal{L}}{\partial w_0}, \frac{\partial \mathcal{L}}{\partial w_1} \right)$$

Matrix notation:

$$\mathbf{w}_{(n+1)} = \mathbf{w}_{(n)} - \gamma \nabla F(\mathbf{w}_{(n)})$$

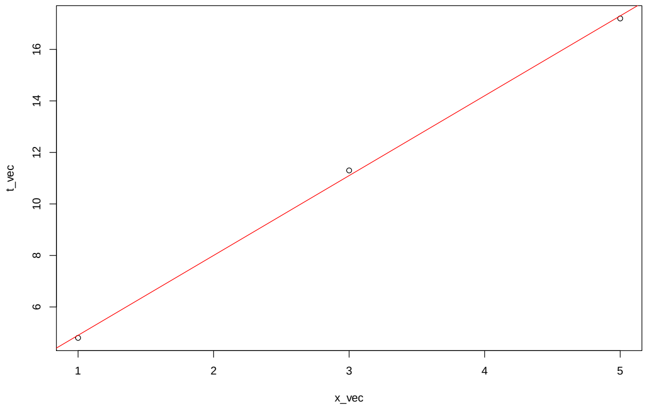
Example Data:

```r
x_vec <- c(1, 3, 5)
t_vec <- c(4.8, 11.3, 17.2)
X <- cbind(1, x_vec)
model <- lm(t_vec ~ x_vec)
model
```

```
##
## Call:
## lm(formula = t_vec ~ x_vec)
##
## Coefficients:
## (Intercept)        x_vec
##         1.8          3.1
```

Example Data:

```r
plot(x_vec, t_vec)
abline(model, col = "red")
```



Coding the Algorithm (matrix notation):

Algorithm:

$$\mathbf{w}_{(n+1)} = \mathbf{w}_{(n)} - \gamma \nabla F(\mathbf{w}_{(n)})$$

Gradient:

$$\nabla F(\mathbf{w}_{(n)}) = \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{-2}{N}\mathbf{X}^T\mathbf{t} + \frac{2}{N}\mathbf{X}^T\mathbf{X}\mathbf{w}$$

The value $\frac{2}{N}$ is a constant that can be accounted for in the gamma term, so I will drop it when writing my gradient function in R.

```r
# this function calculates the gradient of the loss function
# for some particular value of the w vector
grad_loss <- function(w_vec){
  - t(X) %*% t_vec + t(X) %*% X %*% w_vec
  }
```

Coding the Algorithm (matrix notation):

Algorithm:

$$\mathbf{w}_{(n+1)} = \mathbf{w}_{(n)} - \gamma \nabla F(\mathbf{w}_{(n)})$$

```r
# pick an arbitrary starting location for w vector
w_vec <- c(-1, 1.5)
# small gamma chosen, perhaps inefficient but should not overshoot
gamma <- 0.02
# let it run for many iterations
for(i in 1:1000){
  w_vec <- w_vec - gamma * grad_loss(w_vec)
}
w_vec # values at convergence
```

```
##              [,1]
##          1.799995
## x_vec 3.100001
```

Contour plot of the loss function

The following plot shows the contour plot of the loss function for different values of the slope and intercept.

The minimum occurs when the intercept is 1.8 and the slope is 3.1.

We can see when the line has a different value of slope and intercept, the loss function (mean squared error) takes on a higher value.

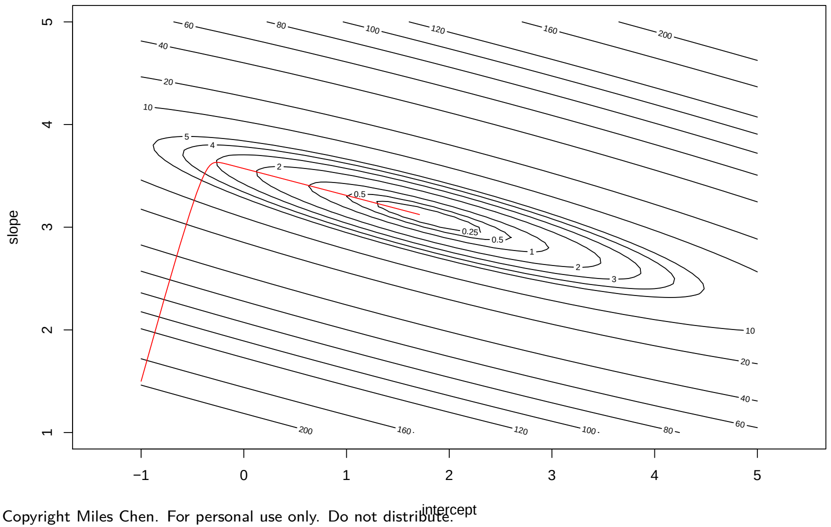
Plotting the Contours

```r
obs_data <- cbind(x_vec,t_vec)
loss_non_vec <- function(obs_data, w0, w1){
  loss <- 0; N <- dim(obs_data)[1]
  for(i in 1:N){
    loss <- loss + (obs_data[i,2] - (w0 + w1 * obs_data[i,1]))^2 }
  return(loss)
}
w0_range <- seq(-1, 5, by = .05); w1_range <- seq(1, 5, by = .05)
loss_contour_grid <- outer(w0_range, w1_range, FUN = "loss_non_vec",
                           obs_data = obs_data)
contour(w0_range, w1_range, z= loss_contour_grid,
  levels = c(0.25, 0.5, 1, 2, 3, 4, 5, 10, 20, 40,
             60, 80, 100, 120, 160, 200), asp = 1)
```

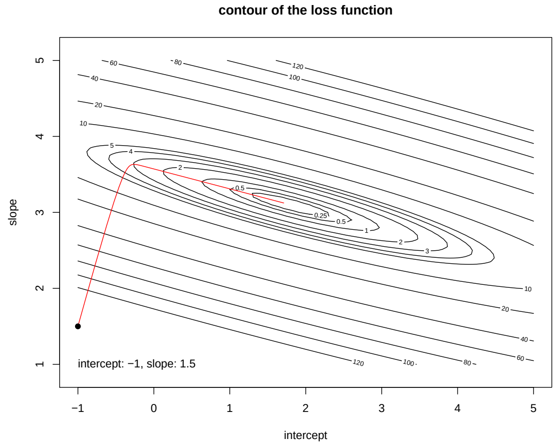
Plotting the Gradient Descent Path

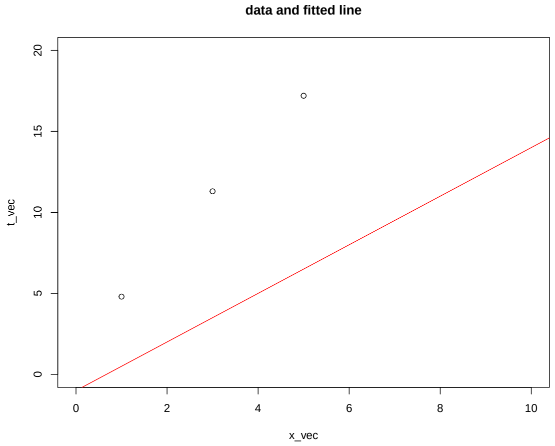
```r
gamma <- 0.01; iter = 500
w_record <- matrix(0, nrow = iter, ncol = 2) # a matrix to record w
w_vec <- c(-1, 1.5)                          # starting location

for(i in 1:iter){
  w_record[i,] <- w_vec                     # store the result
  w_vec <- w_vec - gamma * grad_loss(w_vec) # gradient descent algorithm
}

# plot the iterations
points(w_record[,1],w_record[,2], type="l", col = "red")
```

Plot of gradient descent; gamma = 0.01, start loc = (-1,1.5)

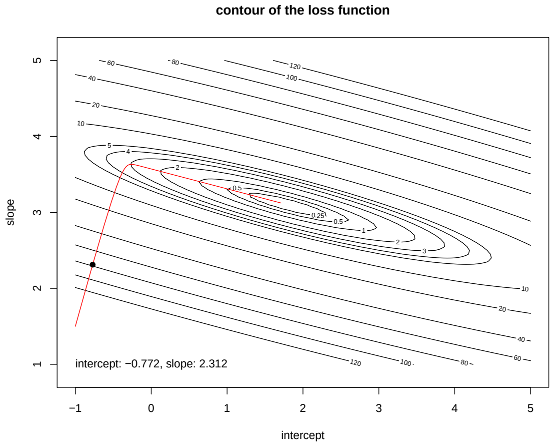


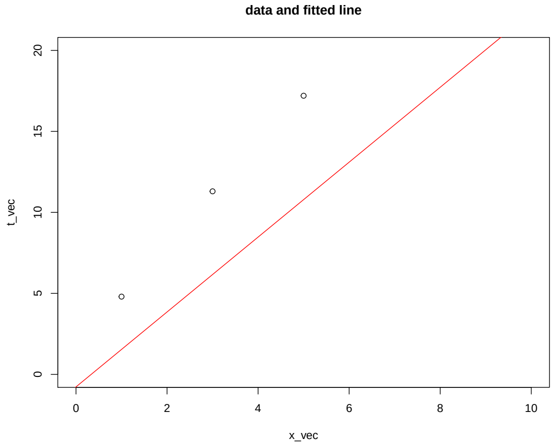
The Loss Function and the Fitted Line - iteration 1



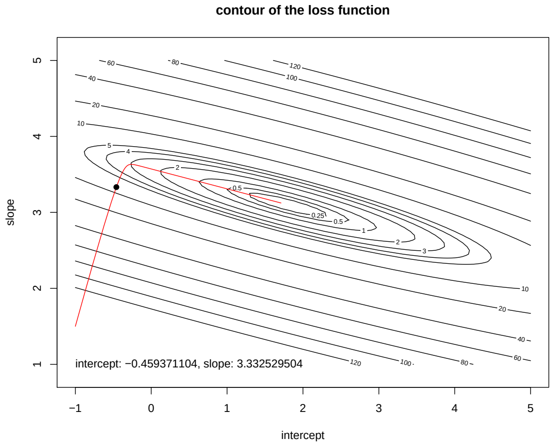


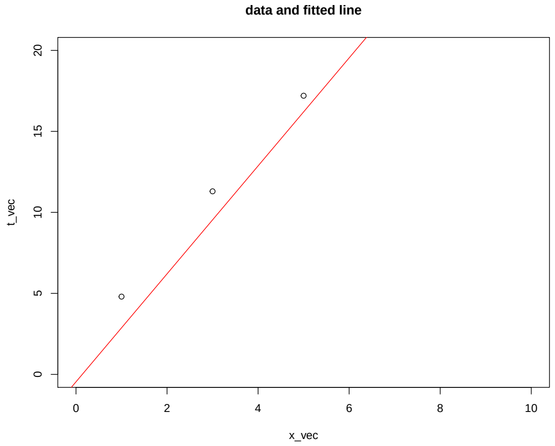
The Loss Function and the Fitted Line - iteration 2



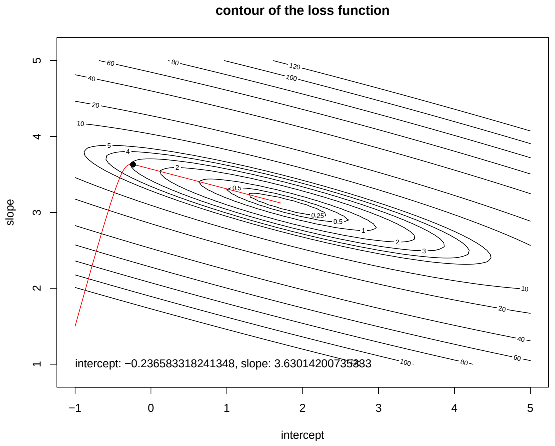


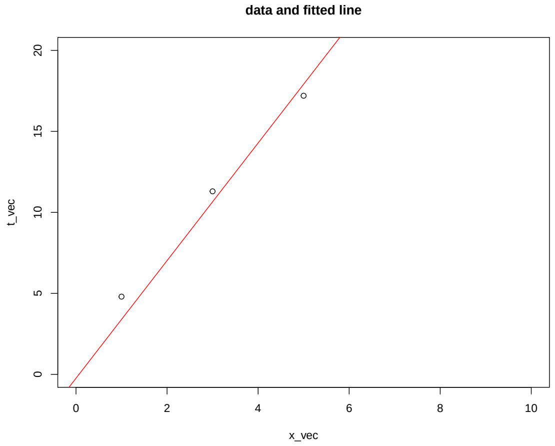
The Loss Function and the Fitted Line - iteration 5



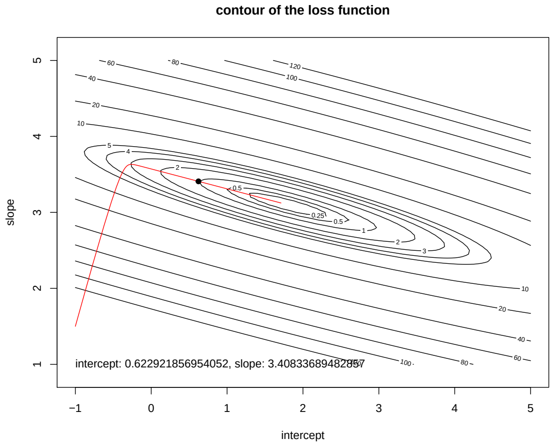


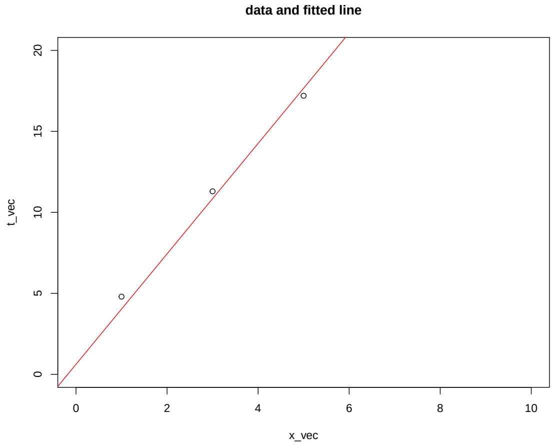
The Loss Function and the Fitted Line - itartion 15



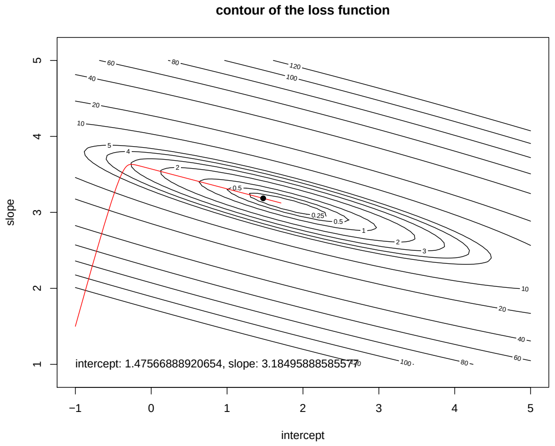


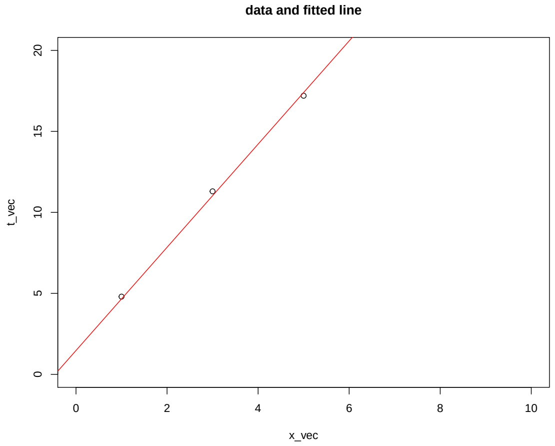
The Loss Function and the Fitted Line - iteration 100



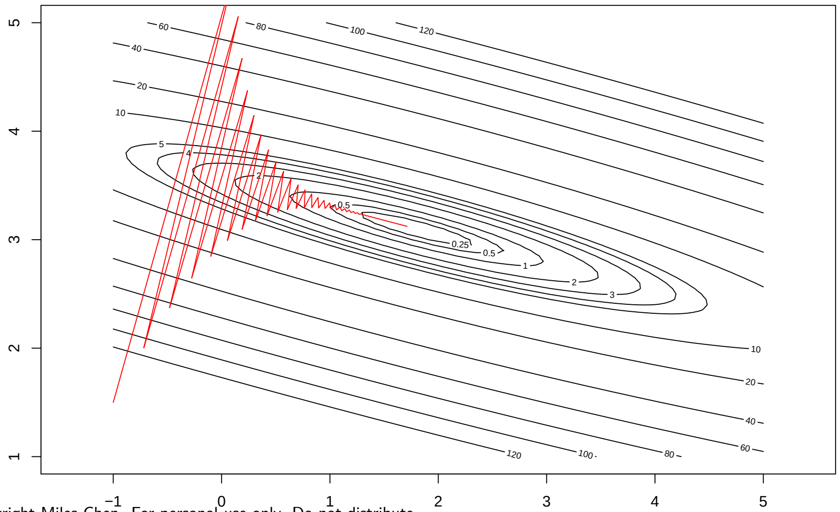


The Loss Function and the Fitted Line - iteration 300

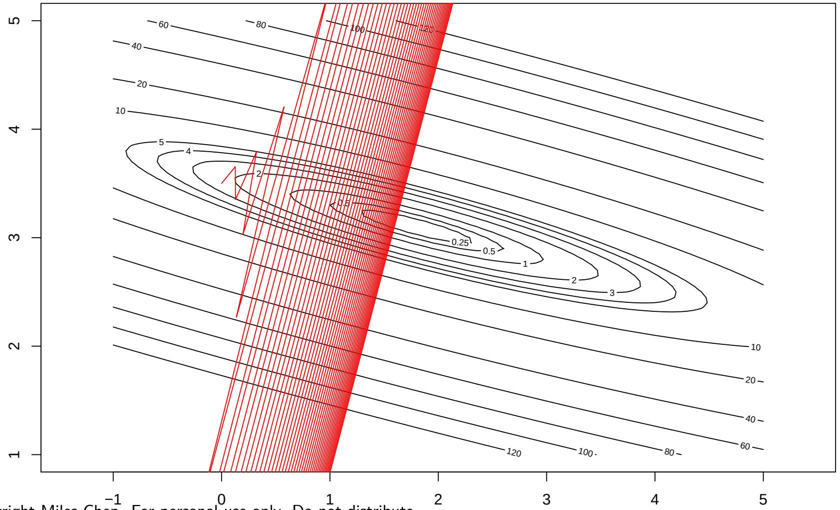




Using a gamma that is a little too big; gamma = 0.05



Using a gamma that is much too big; gamma = 0.07, start loc = (0,3.5)



Section 3

Numeric Gradient Calculation / Checking

Estimating the Gradient Numerically (No Calculus)

Sometimes it can be difficult to find the gradient via calculus.

Sometimes you found the gradient via calculus but you aren't 100% confident that you did your math correctly.

Numeric Gradient calculation is a simple (albeit inefficient) tool to find the gradient and verify your work.

Because of this, we almost always recommend finding the gradient analytically via calculus and implementing the analytic approach.

Numeric Gradient Checking

For a unidimensional function, the derivative is:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

And we can estimate the derivative of f at x by using a small value of h, e.g. $h = 10^{-5}$

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

Numeric Gradient Checking

The idea is similar for a multidimensional function.

Let's say we have a function $F(x_1, x_2)$. We can estimate the gradient of F at (x_1, x_2) by choosing a small value of h.

$$\nabla F = \left[\frac{\partial F}{\partial x_1}, \frac{\partial F}{\partial x_2}\right] \approx \left[\frac{F(x_1 + h, x_2) - F(x_1, x_2)}{h}, \frac{F(x_1, x_2 + h) - F(x_1, x_2)}{h}\right]$$

Numeric Gradient Example

We will revisit this arbitrary function from earlier:

$$F(\mathbf{x}) = F(x_1, x_2) = x_1^2 - 2x_1 - .5x_1x_2 + 2.5x_2^2$$

We'll find the gradient numerically at the point (-5, 4) on the next few slides.

First, we evaluate the value of the function itself at the point (-5, 4):

$$\begin{aligned}
F(\mathbf{x}) &= x_1^2 && - 2x_1 && - .5x_1x_2 && + 2.5x_2^2 \\
F(-5, 4) &= (-5)^2 && - 2(-5) && - .5(-5)(4) && + 2.5(4)^2 \\
F(-5, 4) &= 25 && + 10 && + 10 && + 40 \\
F(-5, 4) &= 85
\end{aligned}$$

Numeric Gradient Example

I'll use an unusually large $h = 10^{-2}$ to illustrate (normally, we would add something small like 10^{-5}):

$$\nabla F \approx [\frac{F(-5+.01,4)-F(-5,4)}{h} \quad , \frac{F(-5,4+.01)-F(-5,4)}{h}]$$
$$\nabla F \approx [\frac{F(-4.99,4)-85}{.01} \quad , \frac{F(-5,4.01)-85}{.01}]$$
$$\nabla F \approx [\frac{84.8601-85}{.01} \quad , \frac{85.22525-85}{.01}]$$
$$\nabla F \approx [\frac{-.1399}{.01} \quad , \frac{.22525}{.01}]$$
$$\nabla F \approx [-13.99 \quad , 22.525]$$

Our numeric gradient estimate is [-13.99, 22.525] which is pretty close to the analytic results of [-14, 22.5]. (slide 10)