# Gradient descent in Matlab/Octave

Shaun Enslin Follow
Jun 15 · 5 min read

So, you have read a little on linear regression. In the world of machine learning it is one of the most used equations and for good reason.

**Linear Regression** is a **machine learning** algorithm based on supervised **learning**. It performs a **regression** task. **Regression** models a target prediction value based on independent variables. … **Linear regression** performs the task to predict a dependent variable value (y) based on a given independent variable (x).

So, how about a quick tutorial on running gradient descent in Matlab / octave to predict house prices?

Sound good?

If you need some pre-reading, here are some good articles:

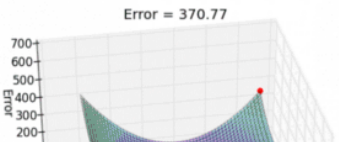
1. Site for great datasets on linear regression
2. Matrix arithmetic
3. Great article that helped me alot

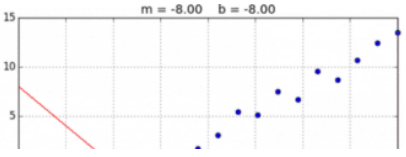
If you prefer a video, then follow my 5 part series on Youtube.
https://www.youtube.com/playlist?list=PLOEB0iByIwznDBU8aF9hVXALjwj5Odrnm

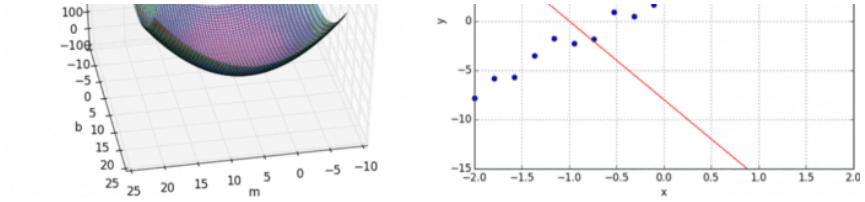
You can also download the source code here:
https://github.com/shaunenslin/gradientdescentmatlab

Gradient descent allows to run through a few thousand thetas till we get to the lowest cost and thus the best theta to make a predition.

Source






## Formulas

Besides gradient descent, we will be using the following formula's. Our hypothesis function is used to predict results in linear regression. In our data below, we have 3 features and as such our hypothesis will be:

```
hø(x) = ø0 + ø1x + ø2x + ø3x
```

If you have not had any exposure to hypothesis equation, have a look here.

## About the data

We will be working with a curated dataset. It will have 3 independent variables (x) which are all identical and the dependent variable (y) is always 1.5 * x. This allows us to check our values as we go and they are easy to calculate.

muiltivariables4 > dataset.txt

```
1,1,1,1.5
2,2,2,3
3,3,3,4.5
4,4,4,6
5,5,5,7.5
```

## Step 1: load the dataset

We first need to load the dataset and split it into our X/Y axis. Lets normalise our X values so the data ranges between -1 and 0. This will assist a-lot with gradient descent and allow for a bigger learning rate and getting our lowest cost theta quicker. Finally, we add a column of ones to assist with our hypothesis calculation and make calculating the cost of each theta a simple matrix arithmetic calculation.

```
% load dataset
ds = load("realestat3.txt")


% split x/y
n = size(ds,2)-1;
x = ds(:,1:n);
y = ds(:,n+1);
m = length(y);


% normalise
[x, maxs, mins] = normalize(x, n);


% add column with ones — help hypothesis
xo = [ones(m,1),x];
```

## Step 2: Normalize

Normalising is easily accomplished with some simple matrix arithmetic. Note the formulae in comments in line 2. This function normalises all our features (independent variables) to somewhere between -1 and 0. We also return a vector of the max's and mins for each feature for later.

```
function [x, maxs, mins] = normalize(x, n)
 % n = (x-max) / (max-min)
 maxs = max(x);
 mins = min(x);
 x = (x-max(x)) ./ (max(x)-min(x));
end
```

## Step 3: Gradient Descent

To start off, gradient descent needs 3 things

1. Learning rate — we will guess at 0.01
2. Number of repititions — we will guess to 1500
3. We need the following theta's(ø), which we will start all at zero
   - ø0 — the intercept
   - ø1 — theta 1 for 1st feature
   - ø2 — theta 2 for 2nd feature
   - ø3 — theta 3 for 3rd feature

Gradient descent now applies the learning rate to our cost derivative function for each of the features. See the formula below. If this is foreign to you, just read some of the articles above or watch my youtube series.

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left( h_\theta(x^{(i)}) - y^{(i)} \right) \\
&\quad \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left( h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)} \\
&\}
\end{aligned}
$$

So, lets create the code below and next we will look at the actual function. Once we have run gradient descent, we will get back our best theta as well as the cost of each theta as we made our way through gradient descent.

```
% gradient descent
repeat = 1500;
lrate = 0.1;
thetas = zeros(n+1, 1);
[best, costs] = gradientdescent(repeat, lrate, thetas, xo, y, m, n);
```

Using matrix arithmetic, we can easily perform gradient descent. Again, if below is looking greek to you, then watch my YouTube series for a deeper understanding.

```
function [thetas, costs] = gradientdescent(repeat, lrate, thetas, xo,
y, m, n)
 costs = zeros(repeat,1);
 for r = 1:repeat
   hc = xo * thetas — y;
   temp = sum(hc .* xo);
   thetas = thetas — (lrate * (1/m)) * temp';
   costs(r) = cost(thetas, xo, y);
 end
end
```

## Step 4: Cost function

You want gradient descent to call your cost function so for each repetition, you can calculate the cost and make sure the cost comes down. See below the equation which we will put into action below.

$$\text{Cost Function:} \quad J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Again, watch my youtube series to see playing around with the learning rate and how it impacts how quickly gradient descent gets to the lowest cost.

```
function j = cost(thetas, xo, y)
  hc = xo * thetas — y;
  m = length(y);
  j = (hc' * hc) / (2 * m);
end
```

## Step 5: Predict our result

Finally, back in our main program, lets:

1. Plot our cost so we can get a visual queue on the costs.

2. Get a matrix of the features (x's) that we want to predict.
3. Normalise the prediction using the mass/mins from earlier.
4. Perform our hypothesis: hø = ø0 + ø1*x1 + ø2*x2 + ø3*x3.
   (We add a "1" column so simple matrix multiplication will do above equation)

```
% plot costs
plot(costs, 1:repeat);

% predict a value
p = [6;6;6];
% normalise the dependent variables
pn = (p-maxs').(maxs'-mins')
% add a 1 for easy hypothesis calc
pn = [1;pn];
r = pn' * best
```

Running above, you should get a result of "9"

If you would like to get talked through all this lot, go to

> *https://www.youtube.com/playlist?list=PLOEB0iByIwznDBU8aF9hVXALjwj5Odrnm*



Matlab Gradient Descent Linear Regression Octave Machine Learning