## Gradient Descent

## Overview

In this notebook, I am trying to learn a simple linear regression problem by gradient descent. Gradient descent is an optimization algorithm that's used when training a machine learning model. It's based on a convex function and tweaks its parameters iteratively to minimize a given function to its local minimum. Gadient descent is the most popular optimization strategy used in machine learning and deep learning at the moment. It is used when training data models, can be combined with every algorithm and is easy to understand and implement.

# What is a Gradient?

In machine learning, a gradient is a derivative of a function that has more than one input variable. Known as the slope of a function in mathematical terms, the gradient simply measures the change in all weights with regard to the change in error.

In [1]:
```
# Preliminaries - packages to load
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

## Generate Data from a known distribution

$Y = b + \theta_1 X_1 + \theta_2 X_2 + \epsilon$

$X_1$ and $X_2$ have a uniform distribution on the interval $[0, 10]$, while `const` is a vector of ones (representing the intercept term).

We set actual values for $b$ ,$\theta_1$, and $\theta_2$

Here $b = 1.5$, $\theta_1 = 2$, and $\theta_2 = 5$

We then generate a vector of $y$-values according to the model and put the predictors together in a "feature matrix" `x_mat`

In [2]:
```
np.random.seed(1234)

num_obs = 100
x1 = np.random.uniform(0,10,num_obs)
x2 = np.random.uniform(0,10,num_obs)
const = np.ones(num_obs)
#Error with mean 0 and std deviation .5
eps = np.random.normal(0,.5,num_obs)

b = 1.5
theta_1 = 2
theta_2 = 5

y = b*const+ theta_1*x1 + theta_2*x2 + eps

x_mat = np.array([const,x1,x2]).T
```

In [5]:
```
x_mat[:5]
```

Out[5]:
```
array([[1.        , 1.9151945 , 7.67116628],
       [1.        , 6.22108771, 7.08115362],
       [1.        , 4.37727739, 7.96867184],
       [1.        , 7.85358584, 5.57760828],
       [1.        , 7.79975808, 9.65836532]])
```

In [6]:
```
y[:5]
```

Out[6]:
```
array([44.00060834, 49.44119069, 50.57415314, 45.58928189, 65.35503861])
```

## Solving Parameters using Scikit-learn Linear Regression model

In [7]:
```
from sklearn.linear_model import LinearRegression

lr_model = LinearRegression(fit_intercept=False)
lr_model.fit(x_mat, y)

lr_model.coef_
```

Out[7]:
```
array([1.49004618, 1.99675416, 5.01156315])
```

## Using matrix algebra directly via the formula $\theta = (X^T X)^{-1} X^T y$

In [8]:
```
np.linalg.inv(np.dot(x_mat.T,x_mat)).dot(x_mat.T).dot(y)
```

Out[8]:
```
array([1.49004618, 1.99675416, 5.01156315])
```

# Solving by Gradient Descent

In [9]:
```
learning_rate = 1e-3
num_iter = 10000
theta_initial = np.array([3,3,3])
```

In [10]:
```
def gradient_descent(learning_rate, num_iter, theta_initial):

    ## Initialization steps
    theta = theta_initial
    theta_path = np.zeros((num_iter+1,3))
    theta_path[0,:]= theta_initial

    loss_vec = np.zeros(num_iter)

    ## Main Gradient Descent loop (for a fixed number of iterations)
    for i in range(num_iter):
        y_pred = np.dot(theta.T,x_mat.T)
        loss_vec[i] = np.sum((y-y_pred)**2)
        grad_vec = (y-y_pred).dot(x_mat)/num_obs  #sum up the gradients acr
        grad_vec = grad_vec
        theta = theta + learning_rate*grad_vec
        theta_path[i+1,:]=theta
    return theta_path, loss_vec
```

In [11]:
```
theta_path, loss_vec = gradient_descent(learning_rate, num_iter, theta_init
```

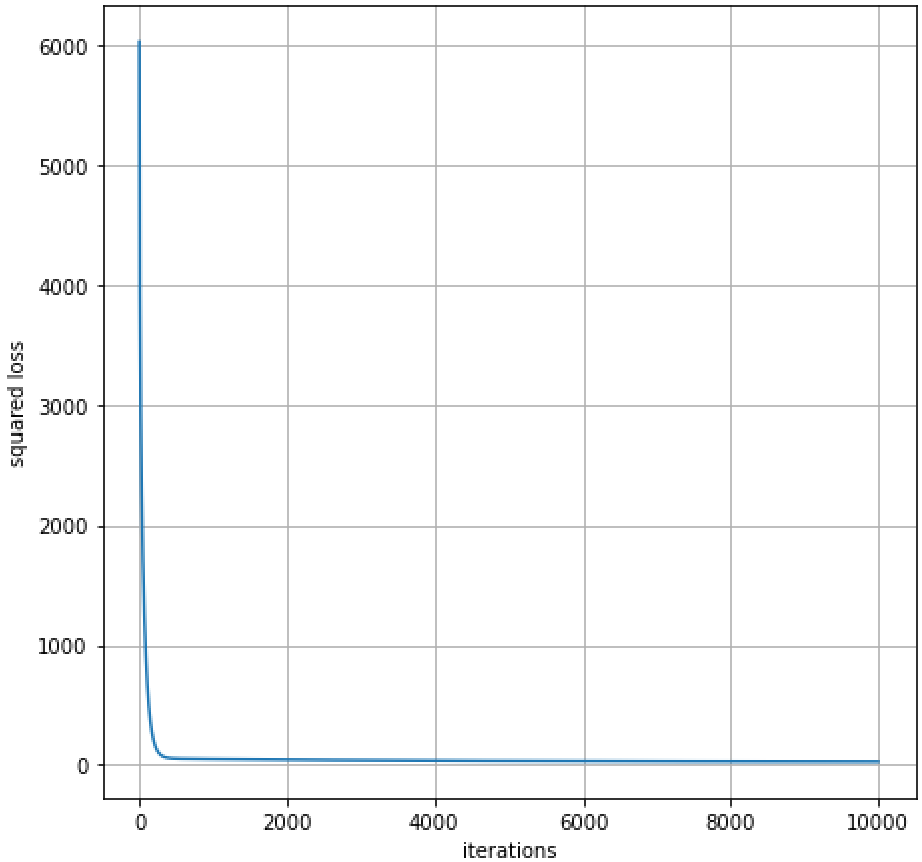
In [12]:
```
theta_path
```

Out[12]:
```
array([[3.        , 3.        , 3.        ],
       [3.0040861 , 3.01332901, 3.03957461],
       [3.00788661, 3.02507893, 3.0772721 ],
       ...,
       [1.93625202, 1.95792066, 4.97633957],
       [1.93619616, 1.95792552, 4.97634398],
       [1.93614031, 1.95793038, 4.97634839]])
```

In [13]:
```
loss_vec
```

Out[13]:
```
array([6030.84477131, 5688.39319737, 5382.14185518, ...,   26.89893211,
         26.89829943,   26.8976669 ])
```

In [16]:
```
fig = plt.figure(figsize=(16, 16))
ax = fig.add_subplot(2, 2, 1)
ax.plot(loss_vec)
ax.set(xlabel='iterations', ylabel='squared loss')
ax.grid(True)
```




In [ ]:
```
#dercreasing number of iterations
learning_rate = 1e-3
num_iter = 100
theta_initial = np.array([3,3,3])
```

In [17]:
```
theta_path, loss_vec = gradient_descent(learning_rate, num_iter, theta_init
```

In [18]:
```
theta_path
```

Out[18]:
```
array([[3.        , 3.        , 3.        ],
       [3.0040861 , 3.01332901, 3.03957461],
       [3.00788661, 3.02507893, 3.0772721 ],
       ...,
       [1.93625202, 1.95792066, 4.97633957],
       [1.93619616, 1.95792552, 4.97634398],
       [1.93614031, 1.95793038, 4.97634839]])
```

In [19]:
```
loss_vec
```

Out[19]:
```
array([6030.84477131, 5688.39319737, 5382.14185518, ...,   26.89893211,
         26.89829943,   26.8976669 ])
```

In [20]:
```
#dercreasing number of iterations
learning_rate = 1e-1
num_iter = 100
theta_initial = np.array([3,3,3])
theta_path, loss_vec = gradient_descent(learning_rate, num_iter, theta_init
```

In [21]:
```
theta_path
```

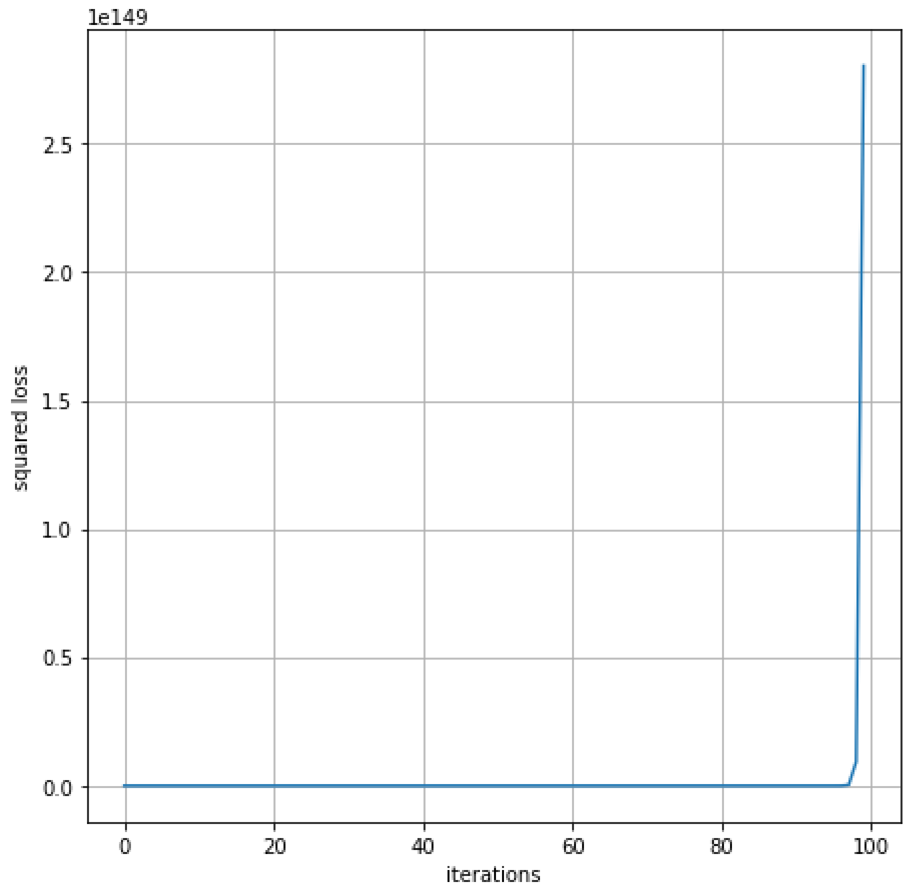
Out[21]:
```
array([[ 3.00000000e+00,  3.00000000e+00,  3.00000000e+00],
       [ 3.40860985e+00,  4.33290090e+00,  6.95746086e+00],
       [ 9.61375099e-01, -1.01251113e+01, -7.85627356e+00],
       [ 1.42042217e+01,  6.76944931e+01,  7.43511914e+01],
       [-5.83385868e+01, -3.58062714e+02, -3.74975004e+02],
       [ 3.38232987e+02,  1.97012227e+03,  2.08216823e+03],
       [-1.83050073e+03, -1.07613312e+04, -1.13544691e+04],
       [ 1.00288941e+04,  5.88594153e+04,  6.21225093e+04],
       [-5.48231908e+04, -3.21854978e+05, -3.39679320e+05],
       [ 2.99814126e+05,  1.76004533e+06,  1.85753562e+06],
       [-1.63948629e+06, -9.62462766e+06, -1.01577244e+07],
       [ 8.96539270e+06,  5.26313693e+07,  5.55465776e+07],
       [-4.90263753e+07, -2.87809604e+08, -3.03751123e+08],
       [ 2.68096069e+08,  1.57385934e+09,  1.66103404e+09],
       [-1.46605773e+09, -8.60649950e+09, -9.08320595e+09],
       [ 8.01699671e+09,  4.70638205e+10,  4.96706443e+10],
       [-4.38401808e+10, -2.57364008e+11, -2.71619174e+11],
       [ 2.39735842e+11,  1.40737050e+12,  1.48532351e+12],
       [-1.31097255e+12, -7.69607115e+12, -8.12234978e+12],
       [ 7.16892821e+12,  4.20852300e+13,  4.44162940e+13],
       [-3.92025992e+13, -2.30139061e+14, -2.42886262e+14],
       [ 2.14375670e+14,  1.25849348e+15,  1.32820033e+15],
       [-1.17229288e+15, -6.88195138e+15, -7.26313663e+15],
       [ 6.41057163e+15,  3.76332938e+16,  3.97177690e+16],
       [-3.50555987e+16, -2.05794073e+17, -2.17192826e+17],
       [ 1.91698193e+17,  1.12536523e+18,  1.18769823e+18],
       [-1.04828326e+18, -6.15395230e+18, -6.49481433e+18],
       [ 5.73243690e+18,  3.36523004e+19,  3.55162718e+19],
       [-3.13472837e+19, -1.84024391e+20, -1.94217341e+20],
       [ 1.71419627e+20,  1.00631981e+21,  1.06205898e+21],
       [-9.37391860e+20, -5.50296375e+21, -5.80776809e+21],
       [ 5.12603786e+21,  3.00924317e+22,  3.17592251e+22],
       [-2.80312485e+22, -1.64557589e+23, -1.73672289e+23],
       [ 1.53286205e+23,  8.99867458e+23,  9.49710327e+23],
       [-8.38230972e+23, -4.92083925e+24, -5.19340022e+24],
       [ 4.58378602e+24,  2.69091395e+25,  2.83996131e+25],
       [-2.50659960e+25, -1.47150059e+26, -1.55300572e+26],
       [ 1.37071005e+26,  8.04676045e+26,  8.49246345e+26],
       [-7.49559702e+26, -4.40029410e+27, -4.64402253e+27],
       [ 4.09889564e+27,  2.40625880e+28,  2.53953937e+28],
       [-2.24144194e+28, -1.31583965e+29, -1.38872285e+29],
       [ 1.22571112e+29,  7.19554343e+29,  7.59409827e+29],
       [-6.70268417e+29, -3.93481420e+30, -4.15276011e+30],
       [ 3.66529882e+30,  2.15171557e+31,  2.27089720e+31],
       [-2.00433366e+31, -1.17664511e+32, -1.24181845e+32],
       [ 1.09605072e+32,  6.43437139e+32,  6.79076558e+32],
       [-5.99364867e+32, -3.51857454e+33, -3.71346530e+33],
       [ 3.27756953e+33,  1.92409888e+34,  2.03067303e+34],
       [-1.79230759e+34, -1.05217510e+35, -1.11045415e+35],
       [ 9.80106290e+34,  5.75371904e+35,  6.07241249e+35],
       [-5.35961764e+35, -3.14636630e+36, -3.32064077e+36],
       [ 2.93085572e+36,  1.72056036e+37,  1.81586069e+37],
       [-1.60271046e+37, -9.40872002e+37, -9.92986077e+37],
       [ 8.76426908e+37,  5.14506869e+38,  5.43004953e+38],
       [-4.79265684e+38, -2.81353167e+39, -2.96937072e+39],
```

```
       [ 2.62081862e+39,  1.53855292e+40,  1.62377202e+40],
       [-1.43316963e+40, -8.41342972e+40, -8.87944222e+40],
       [ 7.83715126e+40,  4.60080369e+41,  4.85563817e+41],
       [-4.28567131e+41, -2.51590556e+42, -2.65525936e+42],
       [ 2.34357843e+42,  1.37579893e+43,  1.45200322e+43],
       [-1.28156348e+43, -7.52342503e+43, -7.94014094e+43],
       [ 7.00810749e+43,  4.11411312e+44,  4.34199024e+44],
       [-3.83231665e+44, -2.24976347e+45, -2.37437589e+45],
       [ 2.09566576e+45,  1.23026167e+46,  1.29840478e+46],
       [-1.14599481e+46, -6.72756842e+46, -7.10020254e+46],
       [ 6.26676315e+46,  3.67890654e+47,  3.88267795e+47],
       [-3.42691957e+47, -2.01177491e+48, -2.12320535e+48],
       [ 1.87397824e+48,  1.10011989e+49,  1.16105457e+49],
       [-1.02476710e+49, -6.01590057e+49, -6.34911602e+49],
       [ 5.60384103e+49,  3.28973777e+50,  3.47195346e+50],
       [-3.06440695e+50, -1.79896168e+51, -1.89860459e+51],
       [ 1.67574167e+51,  9.83745014e+51,  1.03823379e+52],
       [-9.16363329e+51, -5.37951565e+52, -5.67748231e+52],
       [ 5.01104535e+52,  2.94173676e+53,  3.10467698e+53],
       [-2.74024229e+53, -1.60866065e+54, -1.69776295e+54],
       [ 1.49847532e+54,  8.79680713e+54,  9.28405456e+54],
       [-8.19426920e+54, -4.81044996e+55, -5.07689657e+55],
       [ 4.48095785e+55,  2.63054862e+56,  2.77625240e+56],
       [-2.45036901e+56, -1.43849039e+57, -1.51816711e+57],
       [ 1.33996089e+57,  7.86624731e+57,  8.30195185e+57],
       [-7.32744814e+57, -4.30158221e+58, -4.53984308e+58],
       [ 4.00694503e+58,  2.35227915e+59,  2.48256983e+59],
       [-2.19115962e+59, -1.28632139e+60, -1.35756960e+60],
       [ 1.19821471e+60,  7.03412566e+60,  7.42373971e+60],
       [-6.55232272e+60, -3.84654444e+61, -4.05960116e+61],
       [ 3.58307510e+61,  2.10344609e+62,  2.21995412e+62],
       [-1.95937040e+62, -1.15024940e+63, -1.21396071e+63],
       [ 1.07146299e+63,  6.29002901e+63,  6.63842820e+63],
       [-5.85919302e+63, -3.43964229e+64, -3.63016106e+64],
       [ 3.20404375e+64,  1.88093554e+65,  1.98511890e+65],
       [-1.75210072e+65, -1.02857163e+66, -1.08554331e+66],
       [ 9.58119546e+65,  5.62464574e+66,  5.93618994e+66],
       [-5.23938523e+66, -3.07578380e+67, -3.24614877e+67],
       [ 2.86510777e+67,  1.68196300e+68,  1.77512545e+68],
       [-1.56675682e+68, -9.19765402e+68, -9.70710401e+68],
       [ 8.56766005e+68,  5.02964926e+69,  5.30823712e+69],
       [-4.68514307e+69, -2.75041566e+70, -2.90275877e+70],
       [ 2.56202574e+70,  1.50403853e+71,  1.58734591e+71],
       [-1.40101931e+71, -8.22469108e+71, -8.68024951e+71],
       [ 7.66134028e+71,  4.49759377e+72,  4.74671154e+72],
       [-4.18953076e+72, -2.45946620e+73, -2.59569387e+73]])
```

In [22]:
```
loss_vec
```

Out[22]:
```
array([6.03084477e+003, 6.73215975e+004, 2.00817650e+006, 6.00497435e+00
7,
       1.79569283e+009, 5.36974051e+010, 1.60573754e+012, 4.80170887e+01
3,
       1.43587650e+015, 4.29376578e+016, 1.28398401e+018, 3.83955486e+01
9,
       1.14815928e+021, 3.43339209e+022, 1.02670261e+024, 3.07019478e+02
5,
       9.18094090e+026, 2.74541786e+028, 8.20974595e+029, 2.45499709e+03
1,
       7.34128770e+032, 2.19529813e+034, 6.56469828e+035, 1.96307112e+03
7,
       5.87025945e+038, 1.75540996e+040, 5.24928101e+041, 1.56971600e+04
3,
       4.69399203e+044, 1.40366545e+046, 4.19744365e+047, 1.25518037e+04
9,
       3.75342204e+050, 1.12240260e+052, 3.35637074e+053, 1.00367057e+05
5,
       3.00132104e+056, 8.97498467e+057, 2.68382985e+059, 8.02557655e+06
0,
       2.39992409e+062, 7.17660044e+063, 2.14605096e+065, 6.41743227e+06
6,
       1.91903350e+068, 5.73857180e+069, 1.71603082e+071, 5.13152378e+07
2,
       1.53450253e+074, 4.58869162e+075, 1.37217700e+077, 4.10328231e+07
8,
       1.22702288e+080, 3.66922145e+081, 1.09722372e+083, 3.28107721e+08
4,
       9.81155207e+085, 2.93399234e+087, 8.77364866e+088, 2.62362344e+09
0,
       7.84553863e+091, 2.34608654e+093, 7.01560762e+094, 2.09790856e+09
6,
       6.27346988e+097, 1.87598378e+099, 5.60983830e+100, 1.67753506e+10
2,
       5.01640820e+103, 1.50007900e+105, 4.48575340e+106, 1.34139492e+10
8,
       4.01123329e+109, 1.19949705e+111, 3.58690973e+112, 1.07260968e+11
4,
       3.20747274e+115, 9.59144932e+116, 2.86817403e+118, 8.57682922e+11
9,
       2.56476764e+121, 7.66953950e+122, 2.29345673e+124, 6.85822635e+12
5,
       2.05084613e+127, 6.13273700e+128, 1.83389980e+130, 5.48399268e+13
1,
       1.63990288e+133, 4.90387502e+134, 1.46642770e+136, 4.38512441e+13
7,
       1.31130338e+139, 3.92124922e+140, 1.17258871e+142, 3.50644451e+14
3,
       1.04854780e+145, 3.13551944e+146, 9.37628415e+147, 2.80383223e+14
9])
```

In [23]:
```
fig = plt.figure(figsize=(16, 16))
ax = fig.add_subplot(2, 2, 1)
ax.plot(loss_vec)
ax.set(xlabel='iterations', ylabel='squared loss')
ax.grid(True)
```




# The Learning rate

This size of steps taken to reach the minimum or bottom is called Learning Rate. We can cover more area with larger steps/higher learning rate but are at the risk of overshooting the minima. On the other hand, small steps/smaller learning rates will consume a lot of time to reach the lowest point.A learning rate that is too small leads to painfully slow convergence, while a learning rate that is too large can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge.

For gradient descent to reach the local minimum we must set the learning rate to an appropriate value, which is neither too low nor too high. This is important because if the steps it takes are too big, it may not reach the local minimum because it bounces back and forth between the convex function of gradient descent (see left image below). If we set the learning rate to a very small value, gradient descent will eventually reach the local minimum but that may take a while (see the right image).

# Stochastic Gradient Descent

Rather than average the gradients across the whole dataset before taking a step, we will now take a step for every datapoint. Each step will be somewhat of an "overreaction" but they should average out.

Stochastic gradient descent (SGD) does this for each training example within the dataset, meaning it updates the parameters for each training example one by one. Depending on the problem, this can make SGD faster than batch gradient descent. One advantage is the frequent updates allow us to have a pretty detailed rate of improvement.

The frequent updates, however, are more computationally expensive than the batch gradient descent approach. Additionally, the frequency of those updates can result in noisy gradients, which may cause the error rate to jump around instead of slowly decreasing.

In [28]:
```
def stochastic_gradient_descent(learning_rate, num_iter, theta_initial):

    ## Initialization steps
    theta = theta_initial
    # below are different in STOCHASTIC gradient descent
    theta_path = np.zeros(((num_iter*num_obs)+1,3))
    theta_path[0,:] = theta_initial
    loss_vec = np.zeros(num_iter*num_obs)

    ## Main SGD loop
    count = 0
    for i in range(num_iter):
        for j in range(num_obs):
            count+=1
            y_pred = np.dot(theta.T,x_mat.T)
            loss_vec[count-1] = np.sum((y-y_pred)**2)
            grad_vec = (y[j]-y_pred[j])*(x_mat[j,:])
            theta = theta + learning_rate*grad_vec
            theta_path[count,:]=theta
    return theta_path, loss_vec
```

In [29]:
```
## Parameters to play with
learning_rate = 1e-4
num_iter = 100
theta_initial = np.array([3, 3, 3])


theta_path, loss_vec = stochastic_gradient_descent(learning_rate,
                                                   num_iter,
                                                   theta_initial)
```

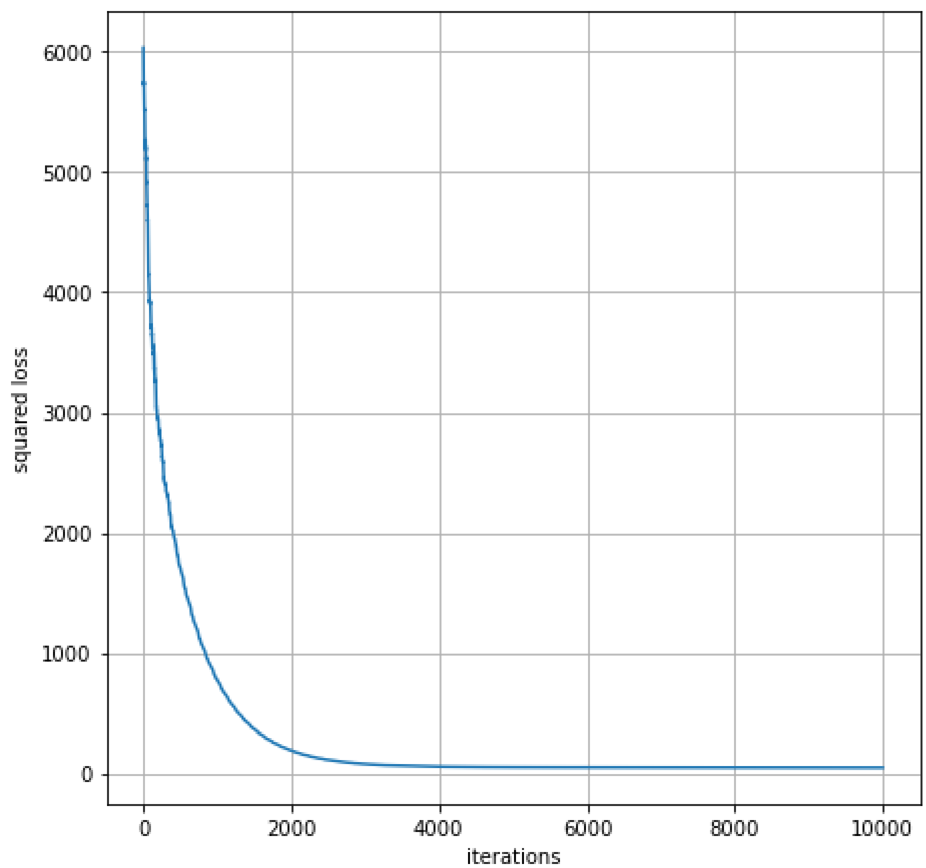
In [30]:
```
theta_path
```

Out[30]:
```
array([[3.        , 3.        , 3.        ],
       [3.00122415, 3.00234449, 3.00939068],
       [3.00186937, 3.00635844, 3.01395955],
       ...,
       [2.86596059, 1.8779056 , 4.90165229],
       [2.86592484, 1.87788517, 4.90136538],
       [2.86601708, 1.87850268, 4.90223761]])
```

In [31]:
```
loss_vec
```

Out[31]:
```
array([6030.84477131, 5949.75553112, 5903.28213363, ...,   48.40016985,
         48.39997699,   48.40134023])
```

In [32]:
```
fig = plt.figure(figsize=(16, 16))
ax = fig.add_subplot(2, 2, 1)
ax.plot(loss_vec)
ax.set(xlabel='iterations', ylabel='squared loss')
ax.grid(True)
```




In [ ]:
```
```