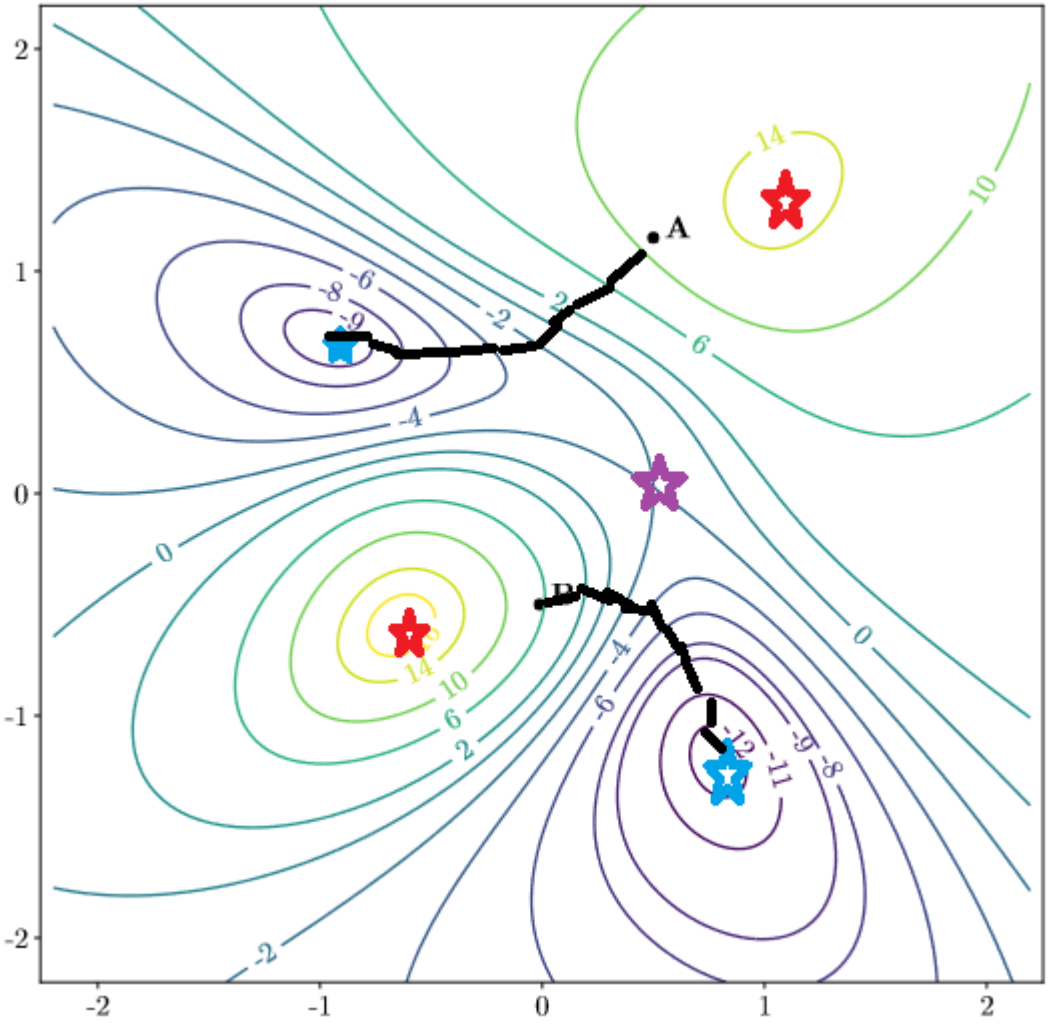
Linear Algebra HW 12

gjd9961

December 4th 2021

1 Problem 12.1

The following plot shows the contour lines of a function $f : \mathbb{R}^2 \to \mathbb{R}$.



a) Give (approximately) the coordinates of the global/local minimizers/maximizers, saddle points of f.

I marked the local minimums with blue stars, local maxima with red stars. The saddle point is marked with a purple star, and the path that gradient descent would take from either point A or point B is marked with a black line.

The coordiantes of the critical points are approximately as follows:

1. Global Minimum Approx. $(\frac{4}{5}, \frac{-5}{4})$
2. Local Minimum: Approx. $(\frac{4}{5}, \frac{-5}{4})$ and Approx. $(\frac{-7}{8}, \frac{2}{3})$
3. Global Maximum: Approx. $(\frac{-2}{3}, \frac{-2}{3})$
4. Local Maximum: Approx. $(\frac{-2}{3}, \frac{-2}{3})$ and Approx. $(\frac{5}{4}, \frac{5}{4})$
5. Saddle Point: Approx. $(\frac{1}{2}, 0)$

b) Assume that we run gradient descent to minimize f. Will gradient descent converge to the global minimizer of f when initialized at point A or at point B?

We can look at the counter plot and begin at either point either A or B. For point A, if we follow the gradient that is orthogonal to the counter lines, we can see that we get trapped in the local minimum at $(\frac{-7}{8}, \frac{2}{3})$ and will not reach the global minimum.

Conversely, we can see that if we start at point B, we will follow the orthogonal gradient to the counter lines and approach the global minimum at point $(\frac{-2}{3}, \frac{-2}{3})$.

2 Problem

Let $M \in \mathbb{R}^{d \times d}$ be a positive semidefinite matrix, $b \in \mathbb{R}^d$ and $c \in \mathbb{R}$. We aim at minimizing the quadratic function

$$f(x) = \frac{1}{2} xMx - \langle x, b \rangle + c$$

using gradient descent. We assume that M is positive definite (i.e. all its eigenvalues are positive). We let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d > 0$ be its eigenvalues and let $v_1, \ldots, v_d$ be an orthonormal basis of $\mathbb{R}^d$ consisting of associated eigenvectors ($Mv_i = \lambda_i v_i$ for all i). We write $L = \lambda_1$ and $\mu = \lambda_d$.

We consider standard gradient descent with constant step-size β:

$$x_{t+1} = x_t - \beta \nabla f(x_t).$$

a) Show that f is L-smooth, μ-strongly convex and that $x^* = M^{-1}b$ is the unique minimizer of f.

We can take advantage of a few facts given to us in the problem. Firstly, lets calculate the gradient and Hessian of our function $f(x)$:

$$\nabla f(x) = f'(x) = Mx - b$$

The Hessian can be computed as follows:

$$H_{f(x)} = M$$

Setting the gradient equal to 0 we can calculate a minimizer for our function:

$$\begin{aligned} \nabla f(x) &= 0 \\ Mx - b &= 0 \\ x &= M^{-1}b \quad \square \end{aligned} \tag{1}$$

We know this solution is unique as our Hessian is Positive Definite since its equal to the Positive Definite matrix M. Therefore, there is no null space, and the matrix M is invertible, thus M is a deterministic linear transformation.

As for the properties of f being L-smooth and μ-strongly convex, these are both true by definition of the problem statement, as $\mu = \lambda_d$ and $L = \lambda_1$. As we know $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d > 0$, we have shown that the function f satisfies the properties of being L-smooth and μ-strongly convex.

b) We now study the convergence of gradient descent to x^*. Show that for all $t \geq 0$,

$$x_{t+1} - x^* = \big(Id - \beta M\big)(x_t - x^*).$$

The answer follows from the definition of standard gradient descent defined in the problem statement, and our calculations from part a that showed that the gradient is: $\nabla f(x) = Mx - b$, and therefore $\nabla f(x) + b = Mx$

$$\begin{aligned} &x_{t+1} - x^* = \big(Id - \beta M\big)(x_t - x^*) \text{ definition} \\ &x_{t+1} - x^* = Id_n x_t - Id_n x^* - \beta M x_t + \beta M x^* \text{ foil function} \\ &x_{t+1} - x^* = x_t - x^* + \beta b - \beta \nabla f(x_t) - \beta b + \beta \nabla f(x^*) \text{ gradient plus b equals Mx} \\ &x_{t+1} - x^* = x_t - \beta \nabla f(x_t) - x^* \text{ definition of standard gradient descent} \\ &x_{t+1} - x^* = x_{t+1} - x^* \quad \square \end{aligned} \tag{2}$$

c)From now, we set $\beta = 1/L$. Deduce from the previous question that for all $t \geq 0$

$$\|x_t - x^*\| \leq \left(1 - \frac{\mu}{L}\right)^t \|x_0 - x^*\|.$$

We can generalize the result of our previous computation to hold for any arbitary t. We take the following:

$$x_{t+1} - x^* = \big(Id - \beta M\big)(x_t - x^*).$$

and re express the equation as:

$$x_t - x^* = \big(Id - \beta M\big)^t (x_0 - x^*).$$

Which holds as since:

$$x_1 - x^* = (Id - \beta M)(x_0 - x^*)$$

Solving for x_1 we get that $x_1 = x_0 - \beta M x_0 + \beta M x^*$ and therefore:

$$x_1 = x_0 - \beta M x_0 + \beta M x^* = (Id - \beta M)^1(x_0 - x^*)$$

Replicating the procedure for $t = 2$ we find that: $x_2 = (Id - \beta M)^2(x_0 - x^*)$ and finally generalizing for all t we have:

$$x_t - x^* = \big(Id - \beta M\big)^t(x_0 - x^*).$$

We can manipulate the above expression to yield our solution.

$$\begin{aligned} x_t - x^* &= \big(Id - \beta M\big)^t(x_0 - x^*) \\ ||x_t - x^*|| &= ||\big(Id - \beta M\big)^t(x_0 - x^*)|| \text{ take the norm of both sides} \end{aligned} \tag{3}$$

Note how the RHS resembles $||Ax||$, since we'll have a matrix multiplying a scalar. We can use an inequality regarding the $||\cdot||_{sp}$ norm we proved in the last homework, that

$$||Ax|| \leq ||A||_{sp}||x||$$

Using the above inequaltity we have:

$$\begin{aligned} &||x_t - x^*|| = ||\big(Id - \beta M\big)^t(x_0 - x^*)|| \leq ||(Id - \beta M)^t||_{sp}||x_0 - x^*|| \\ &||x_t - x^*|| \leq ||(Id - \frac{1}{L}M)^t||_{sp}||x_0 - x^*|| \\ &||x_t - x^*|| \leq (1 - \frac{\mu}{L})^t||x_0 - x^*|| \quad \square \end{aligned} \tag{4}$$

In the second to last step, we needed to choose the eigenvalues that would maximize the expression $||(Id - \frac{1}{L}M)^t||_{sp}$. We did so by taking the largest eigenvalue of Id which is 1 and subtracting the smallest eigenvalue of our matrix M, μ. We also substituted the identity $\frac{1}{L} = \beta$ into our expression to achieve the desired formula in the problem statement.

d) We would like now to have something more precise than the error bound of the previous question. We define $w_t = x_t - x^*$. Let

$$\alpha_1(t) = \langle v_1, w_t \rangle, \cdots, \alpha_d(t) = \langle v_d, w_t \rangle$$

be the coordinates of w_t in the orthonormal basis $(v_1, \ldots, v_d)$. For $i \in \{1, \ldots, d\}$, express $\alpha_i(t)$ in $\alpha_i(t)$ in terms of t, λ_i, L, and $\alpha_i(0)$

This is basically saying to express the position of the input vector of the loss function at time t (the vector $x_t - x^*$). The only thing we need to do is define the position as a vector of coordinates in the orthonormal basis of$(v_1, \ldots, v_d)$. So, we define

$$\alpha_1(t) = \langle v_1, w_t \rangle, \cdots, \alpha_d(t) = \langle v_d, w_t \rangle$$

Which means:

$$w_t = \begin{pmatrix} \alpha_i(t) \\ \vdots \\ \alpha_d(t) \end{pmatrix} = (Id - \beta M)^t \begin{pmatrix} \alpha_i(0) \\ \vdots \\ \alpha_d(0) \end{pmatrix} = (Id - \beta M)^t w_0$$

Known the identity we computed in the last problem, we can see that the matrix M inside the matrix summation term is really λ_i and when you substitue the identity of $\beta = \frac{1}{L}$ then it becomes:

$$\alpha_1(t)v_1 \cdots + \alpha_d(t)v_d = a_1(0)(Id - \frac{\lambda_1}{L})^t v_1 + \cdots + a_d(0)(Id - \frac{\lambda_d}{L})^t v_d$$

Which we can generalize to:

$$w_t = \begin{pmatrix} \alpha_1(t) \\ \vdots \\ \alpha_d(t) \end{pmatrix} = \begin{pmatrix} \alpha_1(0)(Id - \frac{\lambda_1}{L})^t \\ \vdots \\ \alpha_d(0)(Id - \frac{\lambda_d}{L})^t \end{pmatrix} \qquad \text{and} \qquad \alpha_i(t) = \alpha_i(0)(Id - \frac{\lambda_i}{L})^t \quad \square$$

e) Using the previous question, justify the following sentence:

Gradient descent converges towards the minimizer faster in directions given by the eigenvectors of the Hessian of f corresponding to large eigenvalues than in directions corresponding to eigenvectors with small eigenvalues.

We can clearly see this as each entry of w_t, $a_i(t)$ is equal to the original difference between the starting position of the input vector and the optimal input vector to our loss function, $(x_0 - x^*)$, multiplied by $(Id - \frac{\lambda_i}{L})^t$. We know by definition of the problem statement, that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d > 0$, therefore it follows that the term $(Id - \frac{\lambda_i}{L})^t$ in each $a_i(t)$ is sorted in ascending order. Like so:

$$0 \leq (Id - \frac{\lambda_1}{L})^t \leq \cdots \leq (Id - \frac{\lambda_d}{L})^t \leq 1$$

Since we're taking powers of t for some number $0 \leq (Id - \frac{\lambda_i}{L}) \leq 1$ it follows that this inequality holds when generalizing to any step taken at time t.
Which then implies:

$$|\frac{\partial}{\partial t}(1 - \frac{\lambda_1}{L})| \geq \cdots \geq |\frac{\partial}{\partial t}(1 - \frac{\lambda_d}{L})|$$

Therefore, we can see that when utilizing the gradient descent algorithm, the algorithm will converge towards the minimizer faster in directions given by the largest eigenvalues of the hessian, clearly illustrated as entry $a_1(t)$ gets updated the most dramatically as iterations are executed. Generalizing further using the inequalities demonstrated above, we can also see that the gradient will update in descending order i.e. $a_2(t)$ will update more significantly than $a_3(t)$ which updates more drastically than $a_3(t)$ and so on and so forth for all $a_i(t)$ through $a_d(t)$.

f) Show that for all $t \geq 0$

$$\|x_t - x^*\| = \sqrt{\sum_{i=1}^{d} \left(1 - \frac{\lambda_i}{L}\right)^{2t} \langle v_i, x_0 - x^* \rangle^2}.$$

The proof for this problem follows directly from the solution we computed in part c and d. Remember that:

$$x_t - x^* = (1 - \frac{\lambda_i}{L}) w_0 = (1 - \frac{\lambda_i}{L})^t \langle v_i, x_0 - x^* \rangle$$

Taking the L2 norm of each side we can see that:

$$\|x_t - x^*\| = ||(1 - \frac{\lambda_i}{L})^t \langle v_i, x_0 - x^* \rangle|| = \sqrt{\sum_{i=1}^{d} \left(1 - \frac{\lambda_i}{L}\right)^{2t} \langle v_i, x_0 - x^* \rangle^2} \quad \square$$

3 Problem 12.3

In this problem, you will implement and compare gradient descent with or without momentum to minimize the Ridge cost function:

$$f(x) = \frac{1}{2}\|Ax - y\|^2 + \frac{\lambda}{2}\|x\|^2.$$

All the instructions and questions are in the Jupyter notebook `gradient_descent.ipynb`.

It is intended that you code in Python and use the provided Jupyter Notebook. Please only submit a pdf version of your notebook (right-click → 'print' → 'Save as pdf').

In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import copy
plt.rc('font',family='serif')
```
executed in 301ms, finished 02:18:25 2021-12-05

In [2]:
```
d=1000 # d: dimension
n=2000 # n: number of points
A = np.random.normal(size=(n,d)) / np.sqrt(n) # matrix containing the data points
y = np.random.normal(size=n)
lambd= 1
```
executed in 78ms, finished 02:18:25 2021-12-05

We consider the Ridge cost function:

$$f(x) = \frac{1}{2}\|Ax - y\|^2 + \frac{\lambda}{2}\|x\|^2, \tag{1}$$

where $\lambda > 0$ is some regularization parameter that we take equal to 1. The matrix A and the vector y are defined in the cell above.

(a) Show that f is can be written in the format the function f of Problem 12.2, for some $M \in \mathbb{R}^{d\times d}$, $b \in \mathbb{R}^d$ and $c \in \mathbb{R}$. Compute numerically the values of L and μ. Plot the eigenvalues of $H_f(x)$ using an histogram.

We can manipulate the expression by expanding the terms with the squared norms, and then perform algebra to achieve the desired structure of $f(x) = x^T Mx - \langle x, b\rangle + c$

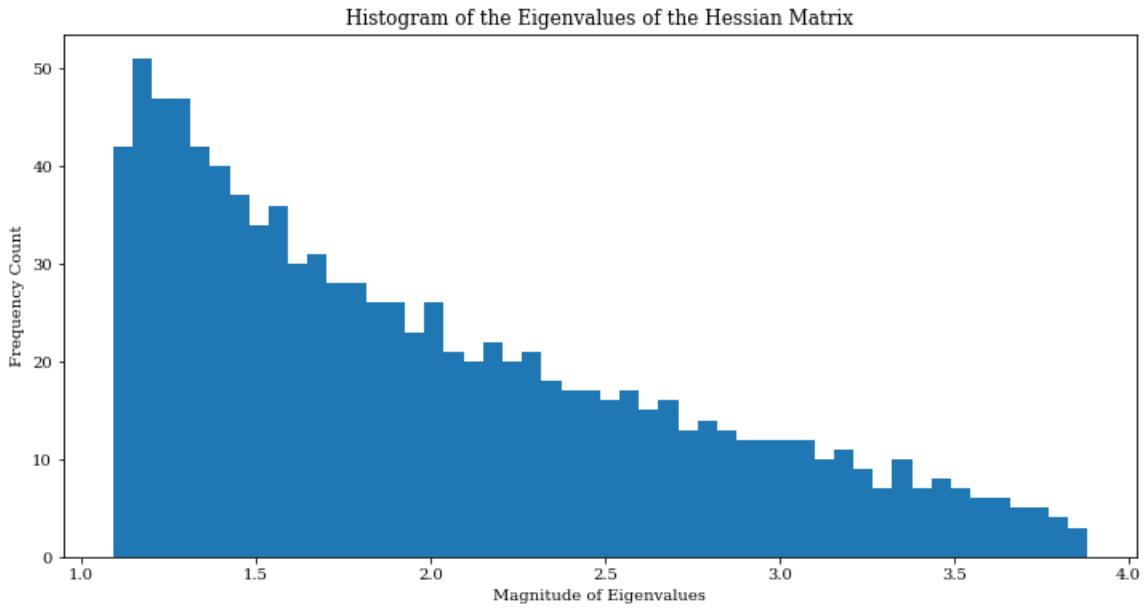
$$\begin{aligned} f(x) &= \frac{1}{2}\|Ax - y\|^2 + \frac{\lambda}{2}\|x\|^2 \\ f(x) &= \frac{1}{2}(Ax - y)^T(Ax - y) + \frac{\lambda}{2}x^T x \\ f(x) &= \frac{1}{2}(x^T A^T Ax - 2x^T A^T y + y^T y) + \frac{\lambda}{2}x^T x \\ f(x) &= \frac{1}{2}x^T(A^T A + \lambda Id)x - x^T A^T y + \frac{y^T y}{2} \end{aligned} \tag{2}$$

Now we have our ridge regression cost function in similar structure to that of a quadratic convex function, $f(x) = x^T Mx - \langle x, b\rangle + c$, where in our case, $M = A^T A + \lambda Id$, $b = A^T y$ and, $c = \frac{y^T y}{2}$

In [3]:
```
#Create the identity matrix and the hessian
I = np.identity(n=d)
hessian_matrix = A.T@A + lambd*I

#Get eigenvalues / vectors using numpy
eigen_values, eigen_vectors = np.linalg.eigh(hessian_matrix)

#Save min and max eigenvals for later
L = np.max(eigen_values)
mu = np.min(eigen_values)

#Plot the eigenvalues using a histogram
plt.figure(figsize=(12,6))
plt.hist(eigen_values, bins=50)[2]
plt.title('Histogram of the Eigenvalues of the Hessian Matrix')
plt.xlabel('Magnitude of Eigenvalues')
plt.ylabel('Frequency Count')
```

executed in 461ms, finished 02:18:25 2021-12-05

Out[3]: Text(0, 0.5, 'Frequency Count')



(b) Implement gradient descent with constant step-size $\beta = 1/L$ (as in Problem 12.2), with random initial position x_0. Plot the log-error $\log(\|x_t - x_*\|)$ as a function of t.

In [4]:
```
#Create a function to return log errors
def log_error_func(xt,xopt):
    difference = xt - xopt
    return np.log(np.linalg.norm(difference))
```

executed in 13ms, finished 02:18:25 2021-12-05

In [5]:
```
#Initialize a vairable to hold random position, create step size
random_position = np.random.normal(size=d)
step_size = 1/L
current_position = random_position

#Calculate optimal x* from part a
optimal_x = np.linalg.inv(hessian_matrix)@A.T@y
```

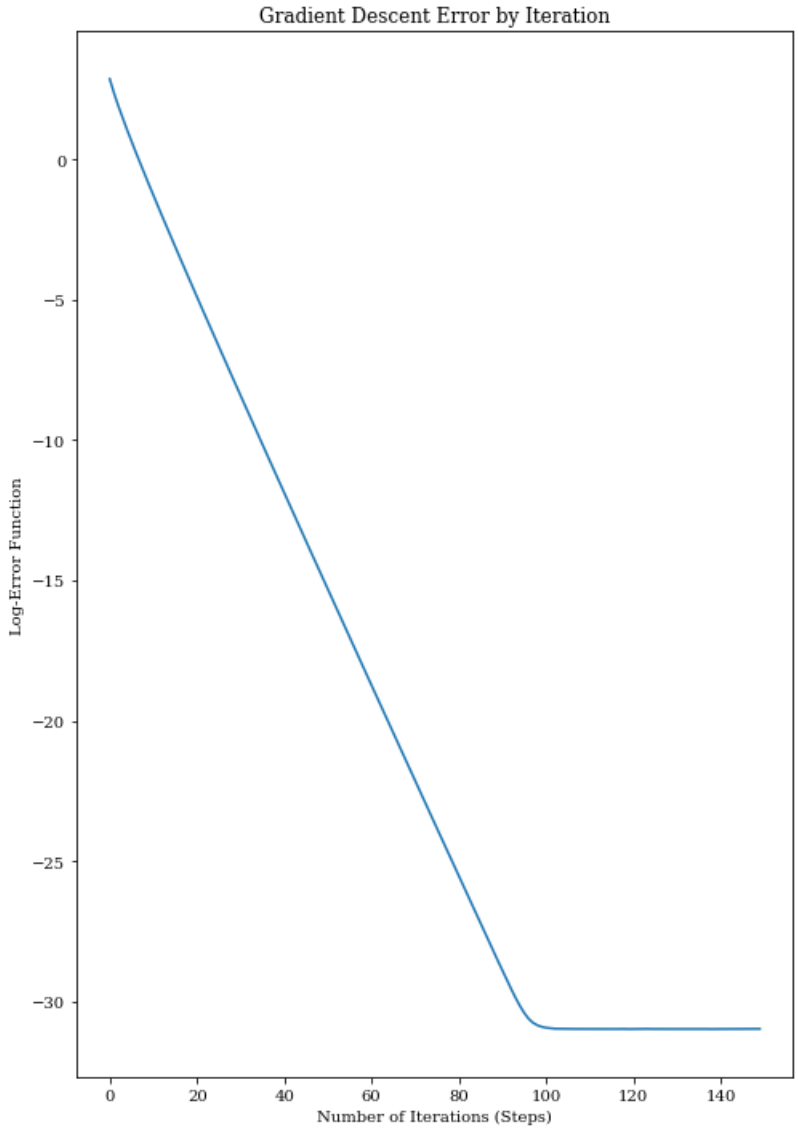
executed in 143ms, finished 02:18:25 2021-12-05

In [6]:
```
#Initialize variables for graph
iterations = []
log_error_arr = []

#Begin gradient descent
for i in range(150):
    current_position = current_position - ((1/L)*(hessian_matrix@current_position - A.T@y))
```

```
        iterations.append(i)
        log_error_arr.append(log_error_func(current_position, optimal_x))

    plt.figure(figsize=(8,12))
    plt.plot(iterations,log_error_arr)
    plt.xlabel('Number of Iterations (Steps)')
    plt.ylabel('Log-Error Function')
    plt.title('Gradient Descent Error by Iteration')
```

executed in 316ms, finished 02:18:26 2021-12-05

Out[6]: Text(0.5, 1.0, 'Gradient Descent Error by Iteration')



(c) Implement gradient descent with momentum, with the same parameters as in Problem 12.4. Plot the log-error $\log(\|x_t - x_*\|)$ as a function of t, on the same plot than the log-error of gradient descent without momentum. On the same plot, plot also the lines of equation

$$y = \log(1 - \mu/L) \times t \quad \text{and} \quad y = \log\left(\frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}}\right) \times t. \tag{3}$$

In [7]:
```
#Define gradient_descent_with_momentum:
def gradient_with_momentum(beta, gamma, current, old):

    #Calcualte new position
    new = current - beta*(hessian_matrix@current - A.T@y) + gamma*(current - old)

    #Return the new position, and the t-1 position
    return new, current
```
executed in 13ms, finished 02:18:26 2021-12-05

In [8]:
```
#Define the bounds of each gradient descent approach
def momentum_bound_gd(arr):
    messy_part = np.log(((L**0.5)-(mu**0.5))/((mu**0.5)+(L**0.5)))
    return messy_part*np.array(arr)

def standard_gd_bound(arr):
    messy_part = np.log(1-(mu/L))
    return messy_part * np.array(arr)
```
executed in 13ms, finished 02:18:26 2021-12-05

In [9]:
```
#Initialize random position, current position, and t-1 position variables
starting_position = np.random.normal(size=d)
current_position = old_position = starting_position

#Get the optimal solution for x
optimal_x = np.linalg.inv(hessian_matrix)@A.T@y

#Calculate beta and gamma parameters from 12.4
beta = 4/(((L**0.5)+(mu**0.5))**2)
gamma = (((L**0.5)-(mu**0.5))/((L**0.5)+(mu**0.5)))**2
```
executed in 160ms, finished 02:18:26 2021-12-05

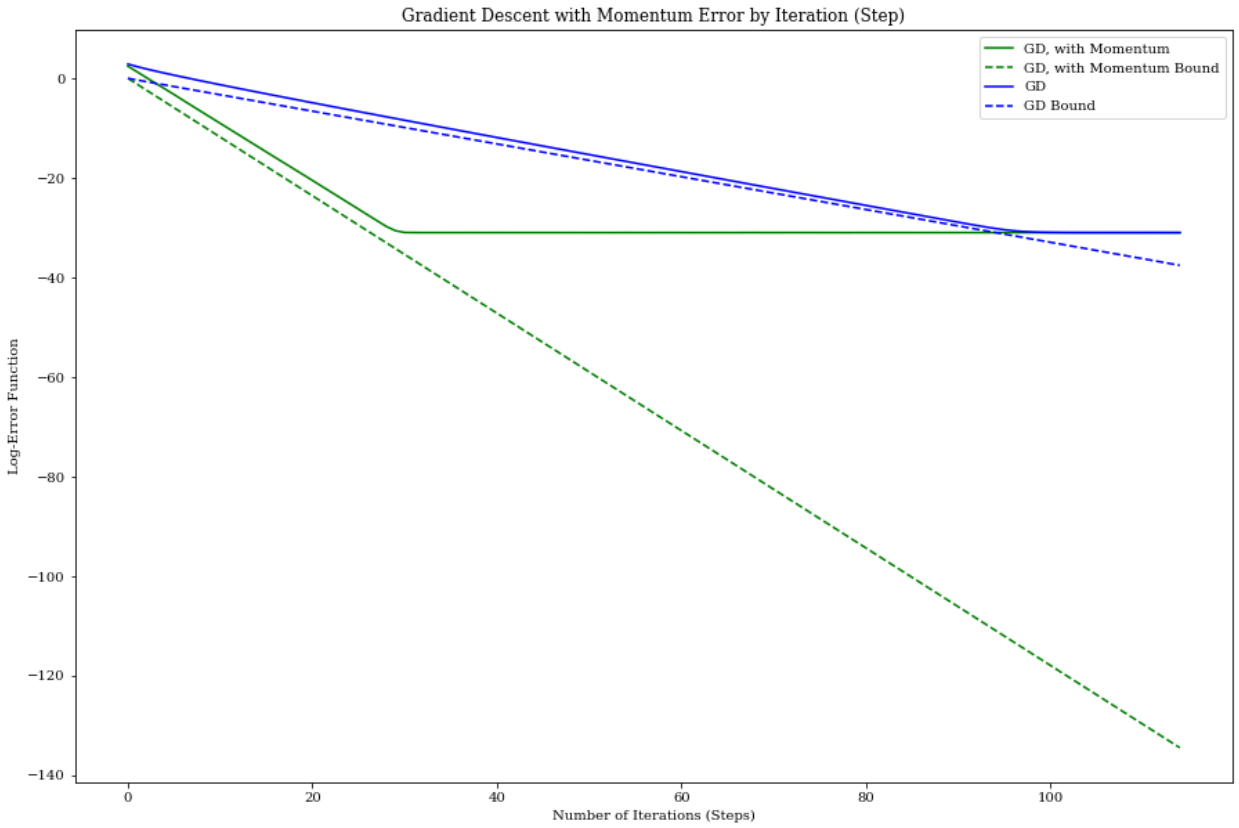
In [10]:
```
#Initialize variables to track iteration
iterations2 = []
log_error_arr2 = []
```
executed in 14ms, finished 02:18:26 2021-12-05

In [11]:
```
#Begin Iterating
for i in range(150):
    current_position, old_position = gradient_with_momentum(beta, gamma, current_position, ol
    iterations2.append(i)
    log_error_arr2.append(log_error_func(current_position, optimal_x))
```
executed in 190ms, finished 02:18:26 2021-12-05

In [12]:
```
#Plot Graphs
plt.figure(figsize=(15,10))
plt.plot(iterations2[:115],log_error_arr2[:115], c='g', label='GD, with Momentum')
plt.plot(iterations2[:115], momentum_bound_gd(iterations2[:115]),'g--',label='GD, with Moment
plt.plot(iterations[:115],log_error_arr[:115], c='b',label='GD')
plt.plot(iterations[:115], standard_gd_bound(iterations[:115]),'b--',label='GD Bound')

plt.xlabel('Number of Iterations (Steps)')
plt.title('Gradient Descent with Momentum Error by Iteration (Step)')
plt.ylabel('Log-Error Function')
plt.legend()
```

executed in 189ms, finished 02:18:26 2021-12-05

Out[12]: <matplotlib.legend.Legend at 0x21203a2d3c8>



Observation: Gradient Descent with Momentum approaches optimal solution nearly 3 times faster than normal gradient descent, which requires nearly 100 iterations to achieve optimal solution

4 Problem 12.4

We take exactly the same setting of Problem **??**, but we now consider gradient descent with momentum:

$$x_{t+1} = x_t - \beta \nabla f(x_t) + \gamma(x_t - x_{t-1}),$$

for $t \geq 1$, where we take

$$\beta = \frac{4}{(\sqrt{L} + \sqrt{\mu})^2} \qquad \text{and} \qquad \gamma = \left(\frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}} \right)^2 .$$

Show that the $\alpha_i(t)\langle v_i, x_t - x^* \rangle$ satisfy a second order linear recurrence relation (as a sequence indexed by t). Using this relation, show that for all $t \geq 0$

$$|\alpha_i(t)| \leq C_i \left(\frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}} \right)^t$$

where C_i is a constant that does not depend on t, but that may depend on x_0, x_1, μ and L (a precise expression of C_i is not expected). Deduce that for all $t \geq 0$

$$\|x_t - x^*\| \leq C \left(\frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}} \right)^t$$

where C is a constant that does not depend on t.

(Help: If you need to get a refresher about what is a linear recurrence, you can check this for instance: https://www.usna.edu/Users/cs/roche/courses/f19sm242/get.php?f=slides5_8.pdf. And you should not be afraid of seeing complex numbers showing up.)