In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import cvxpy as cp
import warnings

from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import LogNorm
from matplotlib import animation

from scipy.optimize import minimize, OptimizeResult
from scipy.special import expit
#from scipy.optimize import line_search
from collections import defaultdict
from itertools import zip_longest
from functools import partial

from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.inspection import DecisionBoundaryDisplay

from copy import deepcopy
```

Implementation (Students do)

Methods

You will implement five optimization algorithms (descriptions available here).

- Gradient descent (`gd`)
- MirrorDescent (`mirrD`)
- Accelerated gradient method (`acc`)
- Adaptive gradient method (`adagrad`)
- Adaptive moment estimation (`adam`)

The last is a very common optimizer used in practical applications -- possibly the most common in the world.

In addition, you will also implement a method for mirrorstep for the entropy regularizer and the ell_2 regularizer `mirrorstep_entropy/ell_2` . The latter of which you will use for the mirror descent algorithm and may also use for the accelerated method implementation. Make note of the function headers: `def gd(func, x, lr, num_iters, grad):` .

- `func` : [type: function] The loss function. Takes in a point of type np.ndarray (n,) for some n and returns a float representing the value of the function at that point.
- `x` : [type: np.ndarray (n,)] The starting point of the optimization.
- `lr` : [type: real] Learning rate.
- `num_iters` : [type: int] The number of iterations of the optimization method to run.
- `grad` : [type: function] The gradient of the loss function. Takes in a point of type np.ndarray (n,) and returns an np.ndarray (n,) representing the gradient of the function at that point.

Each function will need to return a `np.ndarray` containing all the iterates over the course of the optimization.

In [2]:

```python
#from os import XATTR_CREATE
#takes as input an x which is on the n-dimensional unit simplex, lr is a FIXED SCALAR Learning Rate (not a schedule function like th
def mirrorstep_entropy(x, lr, grad):
  # total = sum([x[j] * np.exp(-lr * grad[j]) for j in range(len(x))])
  # return np.array([(x[i] * np.exp(-lr * grad[i])) / total for i in range(len(x))])

  # return x - lr * np.log(1+np.exp(-x)) * grad

  grad_exp = np.exp(-lr * grad)
  return x * grad_exp / np.sum(x*grad_exp)

#same as above except x lies in R^n inside of the unit simplex.
def mirrorstep_ell_2(x, lr, grad):
    return x - lr * grad

def gd(func, x, lr, num_iters, grad):
    iterates = []
    iterates.append(deepcopy(x))
    for itr in range(num_iters):
        update = -lr * grad(x) #change the update function
        x += update
        iterates.append(deepcopy(x))

    return np.array(iterates)

def mirrorD(func, x, lr, num_iters, grad):
    iterates = []
    iterates.append(deepcopy(x))
    for k in range(num_iters):
      x = mirrorstep_entropy(x, lr, grad(x))
      iterates.append(deepcopy(x))
    return np.array(iterates)

#we will not use a learning rate but rather use a parameter L which denotes the Lipshitz constant of the function to be optimized. L
def acc(func, x, L, num_iters, grad):
    y, z = deepcopy(x), deepcopy(x)
    iterates = []
    iterates.append(deepcopy(y))
    for k in range(num_iters):
      tau_k = 2/(k+2)
      eta_k_plus_1 = 1/(tau_k * L)
      x = tau_k * z + (1 - tau_k) * y
      y = x - (1/L) * grad(x)
      z = mirrorstep_entropy(z, eta_k_plus_1, grad(x))
      iterates.append(deepcopy(y))
    return iterates

def adagrad(func, x, lr, num_iters, grad, eps=1e-5):
    iterates = []
    iterates.append(deepcopy(x))
    g_total = np.zeros(x.shape)
    for k in range(num_iters):
      g_total += grad(x) ** 2
      x = x - lr * grad(x) / (np.sqrt(g_total) + eps)
      iterates.append(deepcopy(x))

    return iterates

def adam(func, x, lr, num_iters, grad, beta1=0.9, beta2=0.999, eps=1e-5):
    iterates = []
    iterates.append(deepcopy(x))

    m = np.zeros(x.shape)
    v = np.zeros(x.shape)

    for k in range(num_iters):
      m = beta1 * m + (1-beta1) * grad(x)
      v = beta2 * v + (1-beta2) * grad(x) ** 2
      m_hat = m / (1 - beta1 ** (k+1))
      v_hat = v / (1 - beta2 ** (k+1))
      x = x - lr * m_hat / (np.sqrt(v_hat) + eps)
      iterates.append(deepcopy(x))

    return iterates
```

Testing your code

We are not providing much structure here, but now is a good time to make sure your optimization methods are working well. The cell below tests your gradient descent method on the function $f(x) = x^2$.

In [3]:

```
x_squared_fval = lambda x: x**2
x_squared_grad = lambda x: 2*x
iterates = acc(x_squared_fval,2,1.9,100,x_squared_grad)
```

```
res=[(i,x,x_squared_fval(x)) for (i,x) in enumerate(iterates)]
print([x[2] for x in res])
```

```
[4, 0.011080332409972275, 0.001104963896839344, 0.0006472458237098347, 0.00041003134578937873, 0.00028336201982169084, 0.00020749713
629946435, 0.00015849011580443257, 0.00012500482249419955, 0.00010111419753225267, 8.347314049581352e-05, 7.007719237833092e-05, 5.9
66572156475995e-05, 5.1413627581209796e-05, 4.476247165532881e-05, 3.932335069444449e-05, 3.481867701124556e-05, 3.104601862531492e-
05, 2.785493827160486e-05, 2.513175207756221e-05, 2.2789257369614415e-05, 2.0759660613556658e-05, 1.8989614741676902e-05, 1.74366762
49737368e-05, 1.606673611111113e-05, 1.485213017751478e-05, 1.3770231166954763e-05, 1.2802386901262498e-05, 1.1933110788905359e-05,
1.1149458316818538e-05, 1.044054225922066e-05, 9.797152420980739e-06, 9.211454925677192e-06, 8.676752584034745e-06, 8.18729256408442
6e-06, 7.73811098513479e-06, 7.3249065506948146e-06, 6.9439371804236345e-06, 6.591934978627919e-06, 6.266035913872454e-06, 5.9637213
712075916e-06, 5.682769340137379e-06, 5.421213464653934e-06, 5.1773085406898385e-06, 4.949501326395243e-06, 4.736405750425896e-06,
4.5367817769033686e-06, 4.349517323386633e-06, 4.173612738037383e-06, 4.008167430237402e-06, 3.85236831987695e-06, 3.70547982796784e
-06, 3.566835177925558e-06, 3.4358288149837685e-06, 3.3119097824484986e-06, 3.194575919210665e-06, 3.083368764172328e-06, 2.97786907
08455266e-06, 2.8776928500256648e-06, 2.7824878706629565e-06, 2.691930559288108e-06, 2.6057232469424933e-06, 2.523591719803487e-06,
2.4452830358126897e-06, 2.370563574796594e-06, 2.2992172939725297e-06, 2.2310441644824594e-06, 2.1658587677991717e-06, 2.10348903359
11165e-06, 2.043775102983844e-06, 1.986568303179234e-06, 1.931730221137288e-06, 1.8791318655319407e-06, 1.8286529074961018e-06, 1.78
01809918026096e-06, 1.7336111111111038e-06, 1.6888450367670209e-06, 1.6457908003861992e-06, 1.6043622211118718e-06, 1.56447847400256
2e-06, 1.5260636955113519e-06, 1.489046622457926e-06, 1.4533602612826507e-06, 1.4189415847143954e-06, 1.3857312532856228e-06, 1.3536
733593956083e-06, 1.322715191859194e-06, 1.2928070190882988e-06, 1.2639018892399204e-06, 1.235955445830041e-06, 1.2089257574605516e-
06, 1.1827731604380981e-06, 1.1574601131811293e-06, 1.1329510614168838e-06, 1.1092123132639208e-06, 1.0862119233802714e-06, 1.063919
5854331266e-06, 1.0423065322137635e-06, 1.0213454427828486e-06, 1.0010103560861133e-06, 9.812765905303377e-07]
```

Submission: Challenge

In this part, you will implement the lambda functions for the functional value and the gradient function for logistic regression function given a data matrix X and output vector y. Finally you will run the above algorithms that you implemented for a classification dataset.

In [4]:

```
#1/n \sum_i y_i log(1/(1+e^(-x_i^t w)) + (1-y_i) log(1-1/(1+e^(-x_i^t w))
def logistic_regression_fval(X,y,w):
    # res = 0
    # for i in range(X.shape[0]):
    #    res += y[i] * np.log(expit(np.dot(X[i,:], w)) + 1**-10) + (1-y[i]) * np.log(1-expit(-np.dot(X[i, :], w)) + 1**-10)
    #    #res += y[i] * np.log(1/(1+np.exp(-np.dot(X[i,:], w))) + 1**-10) + (1-y[i]) * np.log(1 - 1/(1+np.exp(-np.dot(X[i,:], w))) +
    # return -res / X.shape[0]

    #p = 1 / (1 + np.exp(-X.dot(w)))
    p = expit(X.dot(w))

    # Compute the logistic regression function
    f = np.mean(y * np.log(p + 1**-10) + (1 - y) * np.log(1 - p + 1**-10))

    return f

def logistic_regression_grad(X,y,w):
    #print((1/(1+np.exp(np.dot(-X, w))) - y).shape)
    #s = expit(X.dot(w))
    s = expit(X.dot(w))
    return X.T.dot(y-s)
```

In [6]:

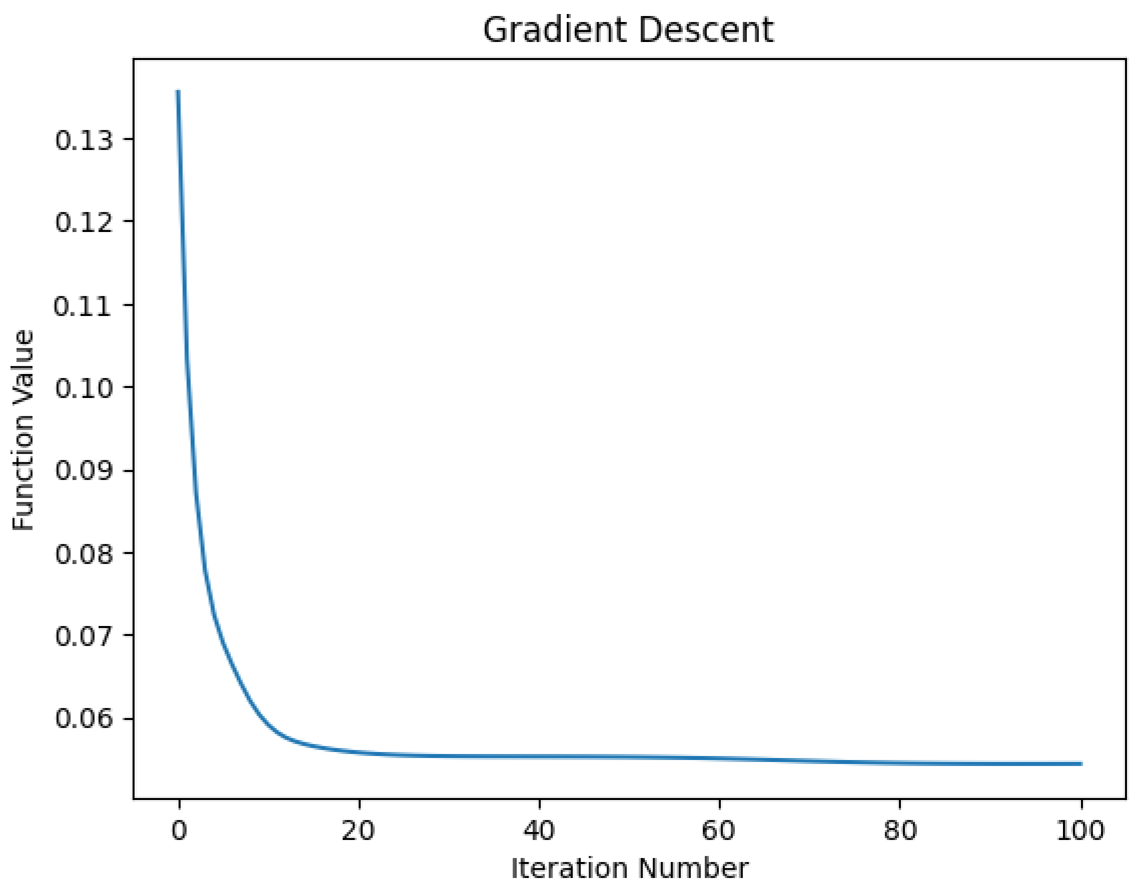
```python
#Logistic Regression Dataset
data=datasets.load_breast_cancer()
X=data.data
scaler=StandardScaler()
scaler.fit(X)
X=scaler.transform(X)
y=data.target
```

In [11]:

```python
iterates = gd(lambda w:logistic_regression_fval(X,y,w),np.random.randn(X.shape[1]),0.0005,100,lambda w:logistic_regression_grad(X,y
res=[(i,w,logistic_regression_fval(X,y,w)) for (i,w) in enumerate(iterates)]
print([x[2] for x in res])

#Plot function value decrease over all iterations
plt.plot([x[0] for x in res],[x[2] for x in res])
plt.xlabel('Iteration Number')
plt.ylabel('Function Value')
plt.title('Gradient Descent')
plt.show()
```

```
[0.135590456170038, 0.10331278192442463, 0.08710466381977733, 0.07778123154974347, 0.07238830969057754, 0.0689718140632538７, 0.06636
620451423778, 0.06405057497533354, 0.06198217300937347, 0.06034041623319575, 0.05909634811099911, 0.05817411800562194, 0.05753074648
5465606, 0.05708713895233629, 0.05676359263242513５, 0.05651057080827912, 0.05630270827746161６, 0.05612744687284718, 0.05597818979311
955, 0.05585098950132290４, 0.05574304949137620６, 0.05565207517743034３, 0.05557601166416204, 0.05551295301011011, 0.0554611232335094
2, 0.05541888323463083, 0.05538474296445297, 0.05535737006797888, 0.05533559203961825６, 0.05531839181407891, 0.05530489805705499, 0.
05529437187508069, 0.05528619161641774, 0.05527983713926561, 0.05527487454747342, 0.05527094203572565５, 0.05526773719254972, 0.05526
500589327814, 0.05526253277046824, 0.05526013316251063, 0.05525764639704123４, 0.05525493025082397, 0.05525185643136149, 0.0552483069
3988096, 0.05524417119523796６, 0.05523934382024728, 0.05523372301385664, 0.05522720945314389, 0.05521970568755412, 0.055211116003548
91, 0.05520134675041308４, 0.05519030712678024, 0.05517791043177168, 0.05516407578359803, 0.05514873030102246４, 0.05513181172817631,
0.05511327145998292５, 0.05509307789349128, 0.05507121999023567, 0.05504771088814646, 0.05502259135214400４, 0.05499593280603301５, 0.0
5496783965239121５, 0.05493845057100888６, 0.05490793849952603, 0.05487650905001588４, 0.05484439720615425６, 0.05481186227476985, 0.054
77918122267108, 0.05474664069665182６, 0.05471452817787988, 0.05468312283644594, 0.05465268670661864, 0.05462345678632706, 0.05459563
85760179, 0.0545694014255552, 0.05454487587644816, 0.05452215299833962４, 0.05450128554997671, 0.05448229066598809４, 0.05446515369247
6155, 0.05444983276769223, 0.05443626376192431４, 0.05442436524128945, 0.05441404318981904５, 0.05440519530048923４, 0.0543977147187987
15, 0.05439149318560263７, 0.05438642357585378５, 0.05438240186595580４, 0.05437932858565926, 0.05437710982289918, 0.05437565785413720
6, 0.05437489147103823, 0.05437473606877959, 0.05437512355364818, 0.05437599211907987６, 0.05437728593080587, 0.05437895475383664６,
0.0543809535469437, 0.054383242044216754]
```

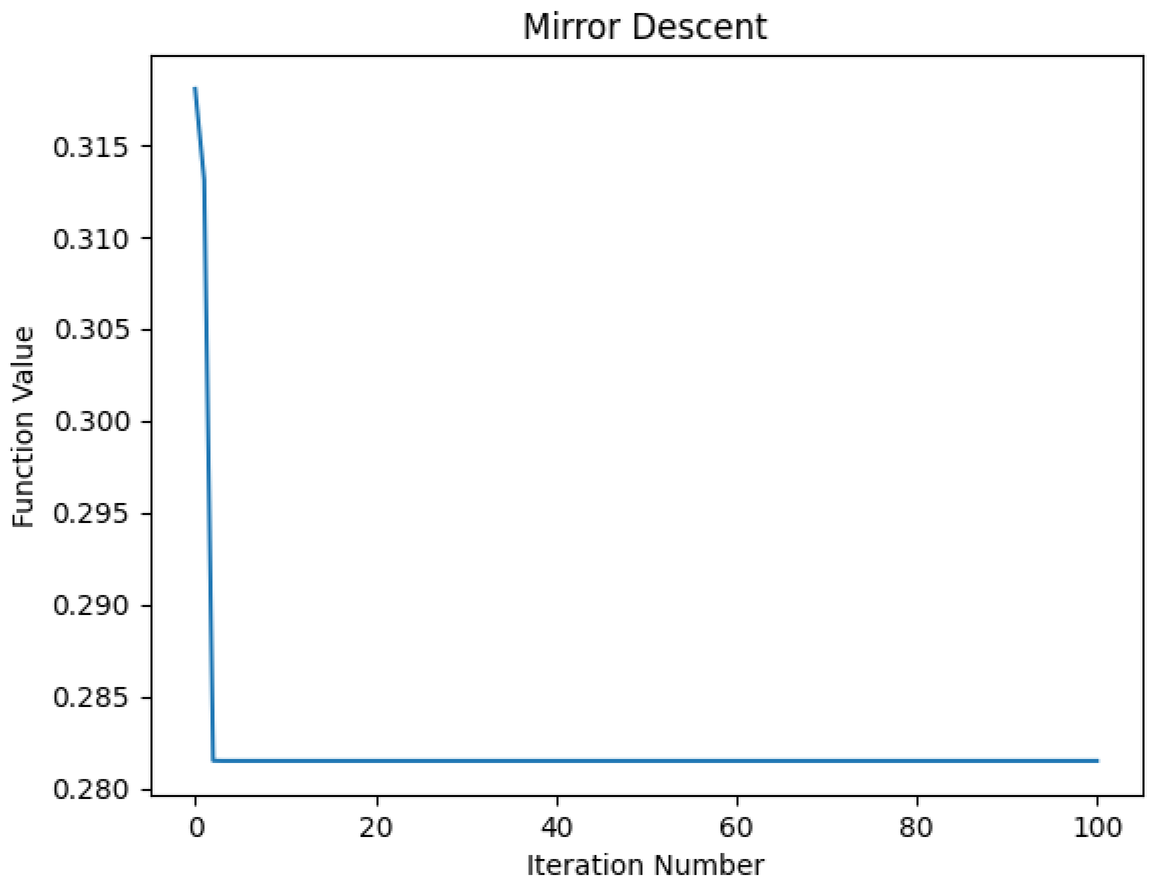


In [21]:

```
iterates = mirrorD(lambda w:logistic_regression_fval(X,y,w),np.random.randn(X.shape[1]),1,100,lambda w:logistic_regression_grad(X,y
res=[(i,w,logistic_regression_fval(X,y,w)) for (i,w) in enumerate(iterates)]
print([x[2] for x in res])

#Plot function value decrease over all iterations
plt.plot([x[0] for x in res],[x[2] for x in res])
plt.xlabel('Iteration Number')
plt.ylabel('Function Value')
plt.title('Mirror Descent')
plt.show()
```

```
[0.31803325136819255, 0.3130988506188747, 0.2815080387614786, 0.2814976858910953, 0.28149768588877655, 0.28149768588877655, 0.281497
68588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.281497685888
77655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655,
0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.2814
9768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.2814976858
8877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.2814976858887765
5, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.2
8149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.2814976
8588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.2814976858887
7655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655,
0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.2814
9768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.2814976858
8877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.281497685888765
5, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.2
8149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.2814976
8588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.28149768588877655, 0.2814976858887
7655]
```

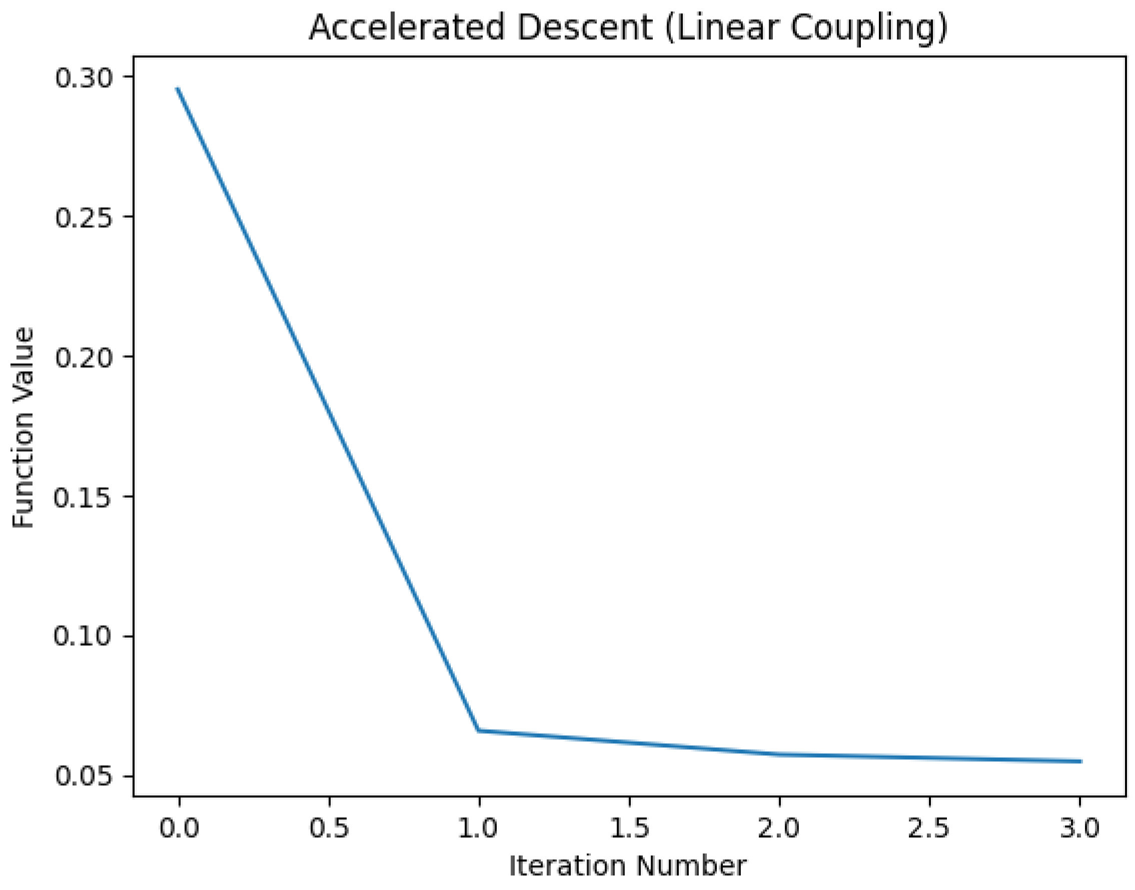


In [22]:

```python
iterates = acc(lambda w:logistic_regression_fval(X,y,w),np.random.randn(X.shape[1]),1,100,lambda w:logistic_regression_grad(X,y,w))
res=[(i,w,logistic_regression_fval(X,y,w)) for (i,w) in enumerate(iterates)]
print([x[2] for x in res])

#Plot function value decrease over all iterations (acc only required 3 iterations to converge)
plt.plot([x[0] for x in res],[x[2] for x in res])
plt.xlabel('Iteration Number')
plt.ylabel('Function Value')
plt.title('Accelerated Descent (Linear Coupling)')
plt.show()
```

```
<ipython-input-2-bdf4a8881cd9>:9: RuntimeWarning: overflow encountered in exp
  grad_exp = np.exp(-lr * grad)
/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py:86: RuntimeWarning: invalid value encountered in reduce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
[0.29524254564822394, 0.06578198198644614, 0.05725468802516243, 0.05481831832196403, nan, nan, nan, nan, nan, nan, nan, nan, nan, na
n, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, na
n, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
nan, nan, nan, nan, nan, nan, nan, nan]
```

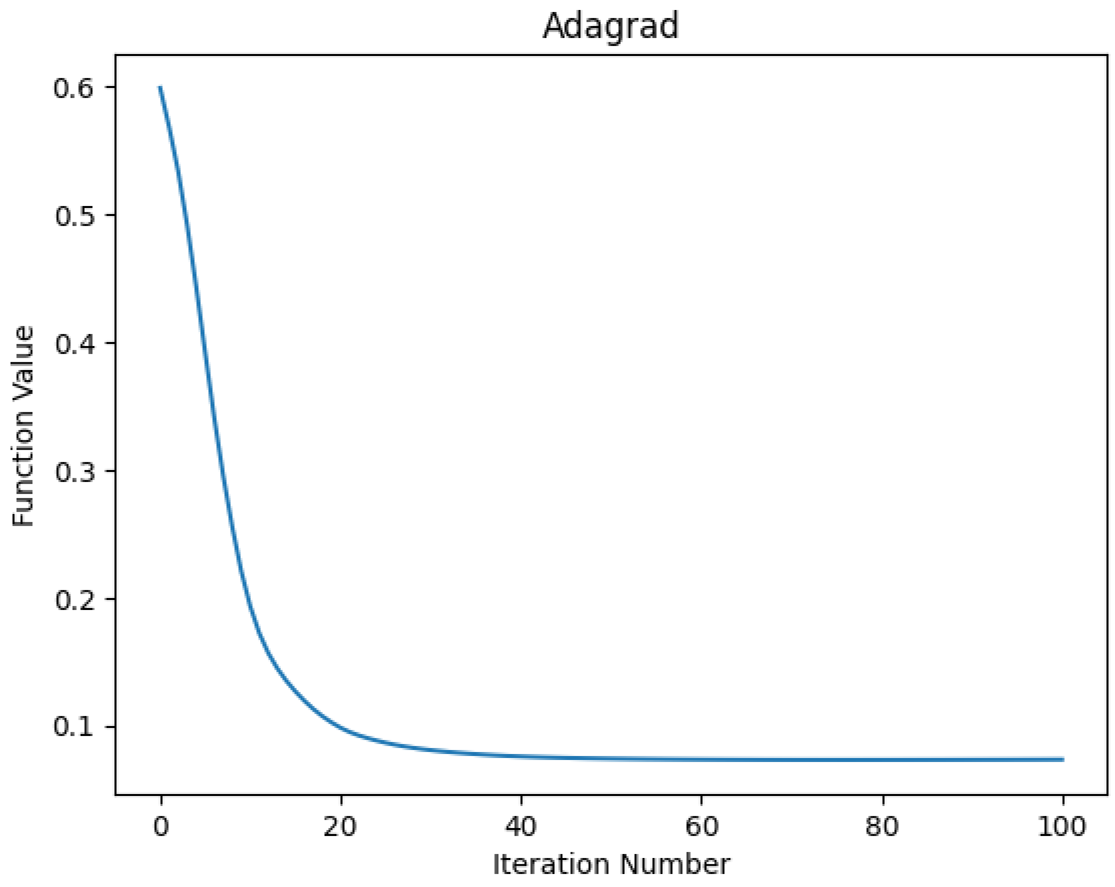


In [23]:

```
iterates = adagrad(lambda w:logistic_regression_fval(X,y,w),np.random.randn(X.shape[1]),0.1,100,lambda w:logistic_regression_grad(X
res=[(i,w,logistic_regression_fval(X,y,w)) for (i,w) in enumerate(iterates)]
print([x[2] for x in res])

#Plot function value decrease over all iterations
#changed step size to 0.1
plt.plot([x[0] for x in res],[x[2] for x in res])
plt.xlabel('Iteration Number')
plt.ylabel('Function Value')
plt.title('Adagrad')
plt.show()
```

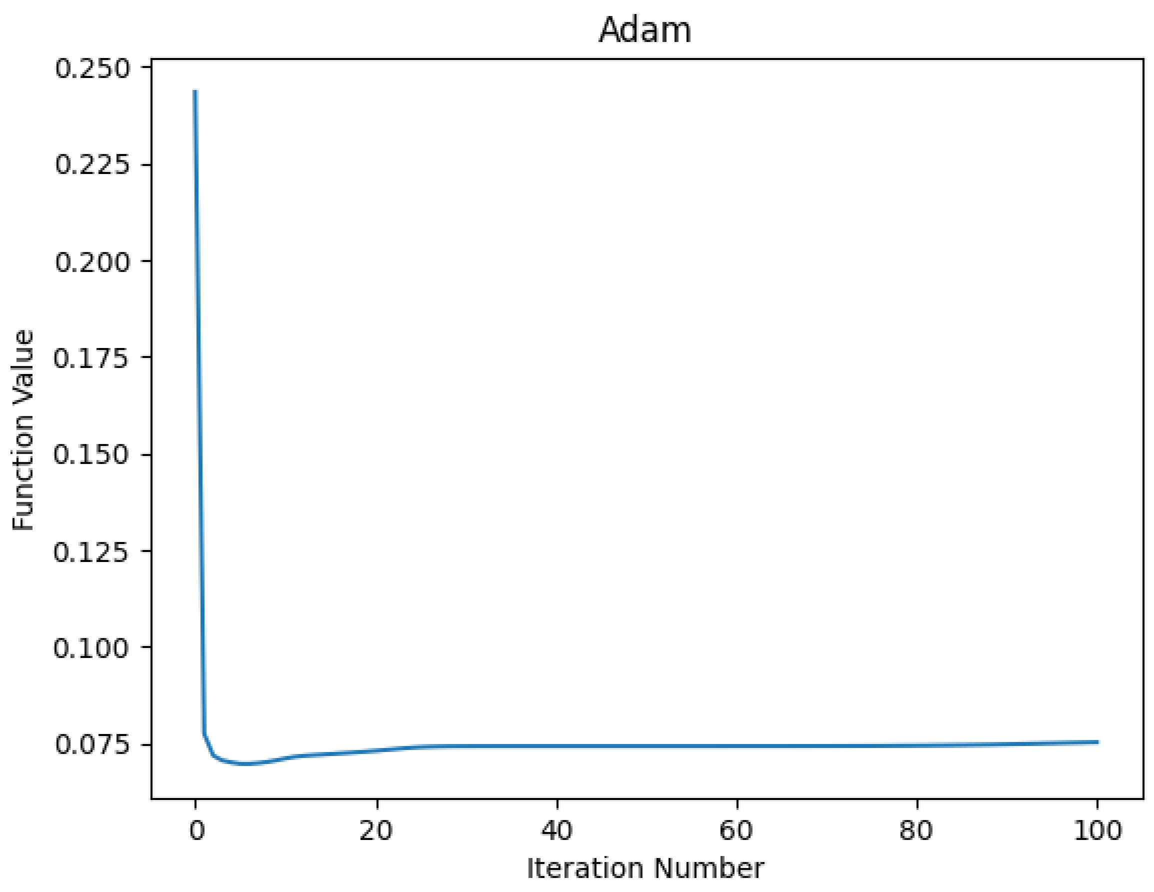
```
[0.5991413808666929, 0.5683925391418021, 0.5338264408024637, 0.4915743297822969, 0.44341716670560233, 0.39172144837428846, 0.3408965
143596152, 0.29526237768128993, 0.2553519536663352, 0.22062864404791974, 0.1930340195646278, 0.17212923363105545, 0.1565179895285541
4, 0.14456067652800794, 0.13484354418136618, 0.12657059564450163, 0.11923719015427942, 0.11272300365343632, 0.10713407979628145, 0.1
0233991288985668, 0.09816936951614699, 0.0948246426588886, 0.09219822729486789, 0.09000698726811009, 0.08809219674363859, 0.08639425
824657265, 0.08489715480297721, 0.083592918642366, 0.08246440816904453, 0.08148556586257254, 0.0806284502689026, 0.0798687144705183,
0.07918771850294885, 0.07857241776462279, 0.07801426900649752, 0.07750784591455984, 0.07704951475142321, 0.07663637483542014, 0.0762
6556728522639, 0.07593395106616965, 0.07563806385984785, 0.0753742514781915, 0.07513885936804379, 0.07492841219620179, 0.07473974300
943989, 0.07457006148930735, 0.07441696844679153, 0.07427843197812817, 0.07415274213181114, 0.07403845824733683, 0.0739343586878532,
0.07383939820800374, 0.07375267463621012, 0.07367340423372687, 0.07360090392748175, 0.0735345783066385, 0.07347390949298267, 0.07341
844846339447, 0.07336780692302537, 0.07332164928530358, 0.0732796846576857, 0.07324165895072736, 0.0732073473375073, 0.0731765473164
674, 0.07314907260094634, 0.07312474799748389, 0.07310340536130551, 0.07308488064444656, 0.07306901198788043, 0.0730556387580302, 0.
07304460139168557, 0.07303574189137575, 0.07302890480435714, 0.07302393852066578, 0.0730206967369588, 0.07301903995082813, 0.0730188
368726346, 0.07301996566654546, 0.07302231495746234, 0.07302578456430267, 0.07303028594142495, 0.07303574232802963, 0.07304208861968
016, 0.07304927098657281, 0.0730572462700403, 0.07306598119241463, 0.07307545141635766, 0.07308564048871767, 0.07309653870151306, 0.
07310814189935914, 0.07312045025904364, 0.073133467063411, 0.07314719748852894, 0.07316164742045748, 0.07317682231590743, 0.07319272
611965384, 0.07320936025069463, 0.073226722668686, 0.07324480703198777, 0.07326360195853458, 0.07328309040052995]
```



In [24]:
```python
iterates = adam(lambda w:logistic_regression_fval(X,y,w),np.random.randn(X.shape[1]),1,100,lambda w:logistic_regression_grad(X,y,w)
res=[(i,w,logistic_regression_fval(X,y,w)) for (i,w) in enumerate(iterates)]
print([x[2] for x in res])
```

```python
#Plot function value decrease over all iterations
plt.plot([x[0] for x in res],[x[2] for x in res])
plt.xlabel('Iteration Number')
plt.ylabel('Function Value')
plt.title('Adam')
plt.show()
```

```
[0.2435267150276437, 0.07753646636708368, 0.07188501168598076, 0.07062600355744889, 0.0701135602415155, 0.06975104643095457, 0.06974
058216585761, 0.06993751504582663, 0.07024139623817811, 0.07065415193956653, 0.07113965101486187, 0.07154758370074743, 0.07181006933
139691, 0.07198562411476737, 0.07213242675604933, 0.07227667799911958, 0.07242666976959447, 0.07258372018429633, 0.0727479745037643
6, 0.07292164252983893, 0.0731093034416035, 0.07331397379631588, 0.07353024688812589, 0.07374097962053454, 0.07392404278691227, 0.07
406474323822212, 0.07416179169084193, 0.0742233886618196, 0.07426030565493964, 0.0742816472995106, 0.07429373474007427, 0.0743005151
4894015, 0.07430431033741358, 0.0743064410028895, 0.07430764515427606, 0.07430833202609707, 0.07430872835703807, 0.0743089603093183
6, 0.07430909867257686, 0.07430918365609397, 0.07430923848833092, 0.07430927691608973, 0.07430930738120839, 0.07430933538230285, 0.0
7430936483989752, 0.07430939891137257, 0.0743094405015048, 0.07430949260571498, 0.07430955856386526, 0.07430964226979535, 0.07430974
836357054, 0.07430988242309745, 0.07431005116581303, 0.07431026266763105, 0.07431052660416419, 0.07431085451782052, 0.07431126011334
502, 0.0743117595835318, 0.07431237196603645, 0.07431311953139151, 0.0743140282014145, 0.07431512799616549, 0.07431645350643967, 0.0
7431804438746596, 0.07431994586803319, 0.0743222092677039, 0.07432489251313837, 0.07432806064288837, 0.07433178628838699, 0.07433615
011731673, 0.07434124122412285, 0.07434715745116213, 0.0743540056227749, 0.07436190167327769, 0.07437097064817458, 0.074381346555252
75, 0.07439317203787797, 0.07440659783565082, 0.07442178198621487, 0.07443888870471484, 0.07445808685228486, 0.07447954787014241, 0.
07450344300998492, 0.07452993963426699, 0.07455919629384639, 0.07459135622198673, 0.07462653882635076, 0.07466482873852606, 0.074706
26203254711, 0.07475080940643619, 0.07479835650775703, 0.07484868225600907, 0.07490143702957168, 0.07495612392218742, 0.075012087748
5926, 0.07506851764962463, 0.07512446928571254, 0.07517891084237906, 0.0752307928071776, 0.0752791350396039, 0.07532311779378216]
```



Run the above code for all the different algorithms and report the convergence behavior of all the algorithms. Add the plots of the function value decrease over all iterates to a PDF file (or add different cells for the run of each algorithm and the plotting and you can just use the PDF of the jupyter notebook.)

Extension: Geometric Descent Implementation

In [25]:

```
def min_enclosing_ball(center_a, radius_a, center_b, radius_b):
    if np.linalg.norm(center_a - center_b) ** 2 >= np.linalg.norm(radius_a ** 2 - radius_b ** 2):
      c = .5 * (center_a + center_b) - (radius_a ** 2 - radius_b**2) * (center_a - center_b) / (2 * np.linalg.norm(center_a - cente
      r = np.sqrt(radius_b ** 2 - ((np.linalg.norm(center_a - center_b) ** 2 + (radius_b ** 2 - radius_a**2)) ** 2) / (4 * np.linal
    elif np.linalg.norm(center_a - center_b) ** 2 < radius_a ** 2 - radius_b ** 2:
      c = center_b
      r = radius_b
    else:
      c = center_a
      r = radius_a

    return c, r
```

In [26]:

```
#def line_search(func, x, y, X):
  #t = cp.Variable()
  #objective = cp.Minimize(func(x + t*(y-x)))
  #prob = cp.Problem(objective)
  #result = prob.solve()
  #print(t.value)
  #print(x + t.value * (y - x))
  #return x + t.value * (y - x)
def line_search(f, x, y):
    """
    Find the value of t that minimizes f(x + t(y - x)) using line search.

    Args:
    x (numpy.ndarray): The starting point of the line search.
    y (numpy.ndarray): The end point of the line search.
    f (callable): The function to be minimized.

    Returns:
    t (float): The value of t that minimizes f(x + t(y - x)).
    """
    def objective(t):
        return f(x + t * (y - x))

    # Define the interval to search over.
    a, b = 0, 1

    # Set the tolerance for the search.
    tol = 1

    # Use the golden section search algorithm to find the minimum.
    rho = (3 - np.sqrt(5)) / 2
    c = 1 - rho
    d = rho
    while abs(b - a) > tol:
        x1 = a + c * (b - a)
        x2 = a + d * (b - a)
        if objective(x1) < objective(x2):
            b = x2
        else:
            a = x1
        c = 1 - (b - a) / (b + a)
        d = (b - a) / (b + a)

    # Return the value of t that minimizes f(x + t(y - x)).
    return x + ((a + b) / 2) * (y - x)
```

In [27]:
```
def geometric_descent(func, x, alpha, num_iters, grad):
  iterates = []
  iterates.append(deepcopy(x))

  x_plus = line_search(func, x, x - grad(x))

  c = x - (1/alpha) * grad(x)
  r = np.sqrt(np.linalg.norm(grad(x))**2 / (alpha**2) - (2/alpha) * (func(x) - func(x_plus)))

  for k in range(num_iters):
    x = line_search(func, x_plus, c)
    x_plus_prev = x_plus
    x_plus = line_search(func, x, x - grad(x))

    center_A = x - (1/alpha) * grad(x)
    radius_A = np.sqrt(np.linalg.norm(grad(x))**2 / (alpha**2) - (2/alpha) * (func(x) - func(x_plus)))

    center_B = c
    radius_B = np.sqrt(r**2 - (2/alpha) * (func(x_plus_prev) - func(x_plus)))

    c, r = min_enclosing_ball(center_A, radius_A, center_B, radius_B)

    iterates.append(deepcopy(x))

  return iterates
```

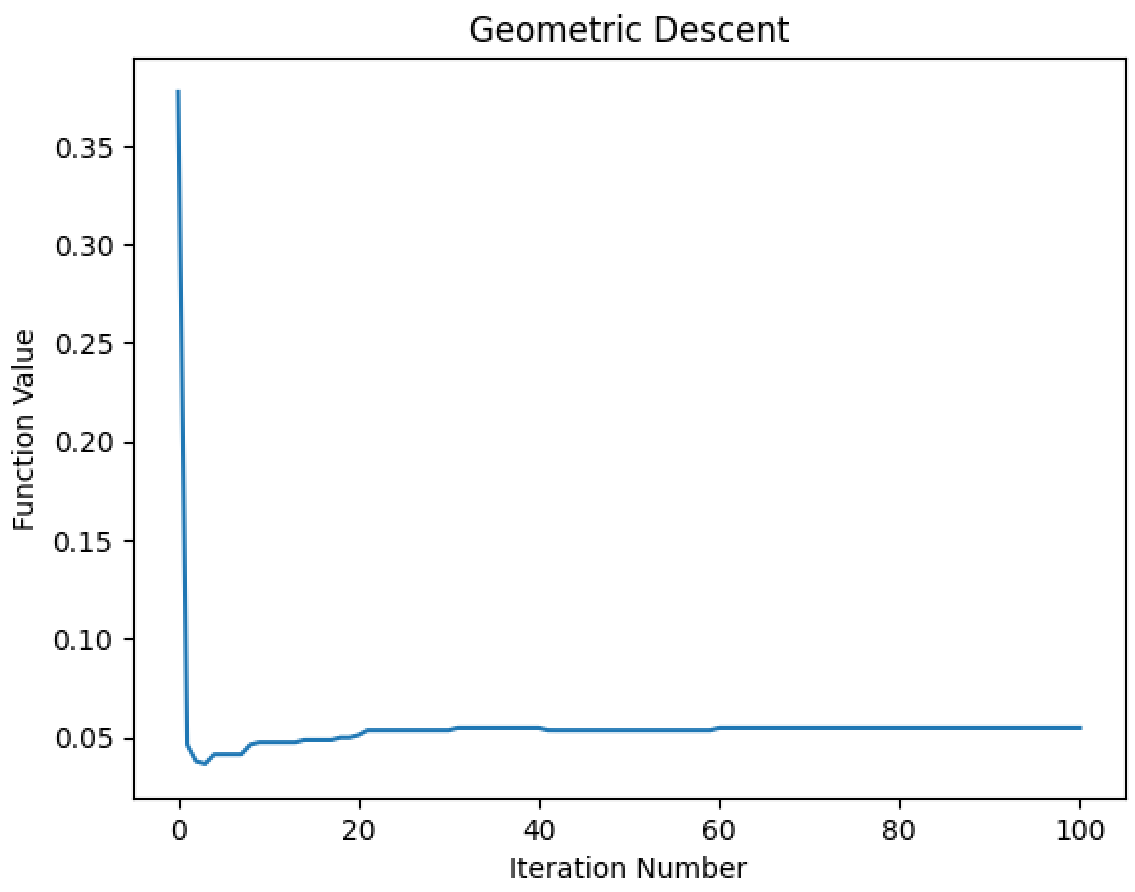
In [28]:
```
iterates = geometric_descent(lambda w:logistic_regression_fval(X,y,w),np.random.randn(X.shape[1]),1,100,lambda w:logistic_regressio
res=[(i,w,logistic_regression_fval(X,y,w)) for (i,w) in enumerate(iterates)]
print([x[2] for x in res])
```

```
[0.37728794832221707, 0.04629103390449065, 0.03786222783688551, 0.03658689916440948, 0.04139670057591397, 0.041418284954372823, 0.04
1418284954372823, 0.041418381013932526, 0.04629102436076963, 0.04750920921236883, 0.04750920921236883, 0.04750920921236883, 0.047509
20921236883, 0.04750920921236883, 0.04872738782656667, 0.04872739406396803, 0.04872739406396803, 0.04872739406397085, 0.049945577046
02576, 0.04994557891572489, 0.05116376376716643, 0.053600133453557436, 0.05360013347036483, 0.05360013347036483, 0.0536001334703648
3, 0.05360013347036483, 0.05360013347036483, 0.05360013347036483, 0.05360013347036483, 0.05360013347036483, 0.05360013360510189, 0.0
5481831832196402, 0.05481831832196402, 0.05481831832196402, 0.05481831832196402, 0.05481831832196402, 0.05481831832196402, 0.0548183
1832196402, 0.05481831832196402, 0.054818318319386536, 0.05481775547651418, 0.05362506118878303, 0.05360013349257797, 0.053600133470
36488, 0.053600133470364825, 0.053600133470364825, 0.053600133470364825, 0.053600133470364825, 0.053600133470364825, 0.0536001334703
64825, 0.053600133470364825, 0.053600133470364825, 0.053600133470364825, 0.053600133470364825, 0.053600133470364825, 0.0536001334703
64825, 0.053600133470364825, 0.053600133470364825, 0.053600133470364825, 0.05360015620823326, 0.054818289789101644, 0.05481831832196
403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403,
0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.0548
1831832196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.0548183183
2196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.0548183183219640
3, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.0
5481831832196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.05481831832196403, 0.0548183
1832196403, 0.05481831832196403]
```

```
<ipython-input-25-4ffa7e68274e>:4: RuntimeWarning: invalid value encountered in sqrt
  r = np.sqrt(radius_b ** 2 - ((np.linalg.norm(center_a - center_b) ** 2 + (radius_b ** 2 - radius_a**2)) ** 2) / (4 * np.linalg.nor
m(center_a - center_b) ** 2))
```

In [29]:

```python
#Plot function value decrease over all iterations
plt.plot([x[0] for x in res],[x[2] for x in res])
plt.xlabel('Iteration Number')
plt.ylabel('Function Value')
plt.title('Geometric Descent')
plt.show()
```

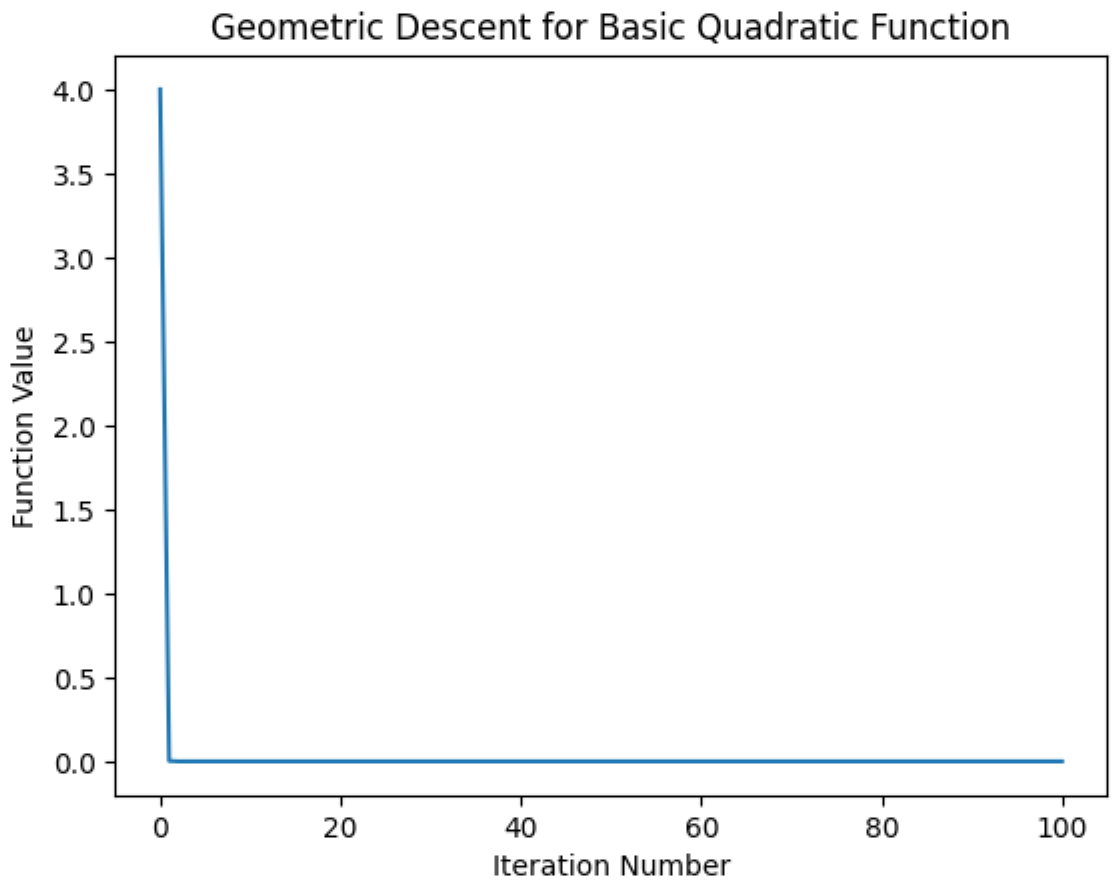


In [30]:
```python
#Geometric Descent with basic parabola
x_squared_fval = lambda x: x**2
x_squared_grad = lambda x: 2*x
iterates = geometric_descent(x_squared_fval,2,1.9,100,x_squared_grad)
res=[(i,x,x_squared_fval(x)) for (i,x) in enumerate(iterates)]
print([x[2] for x in res])
```

```
[4, 0.0027700831024930687, 1.9183400986794067e-06, 1.3284903730466785e-09, 9.200071835503293e-13, 6.371240883312519e-16, 4.412216678
194256e-19, 3.055551716201009e-22, 2.1160330444605283e-25, 1.4653968451942702e-28, 1.0148177598298248e-31, 7.02782382153617e-35, 4.8
66914003833902e-38, 3.3704390608267876e-41, 2.3340990725947245e-44, 1.6164121001348473e-47, 1.1193989613122214e-50, 7.7520703691982e
-54, 5.368469784763286e-57, 3.717776859254341e-60, 2.574637714165048e-63, 1.7829901067624916e-66, 1.234757691663773e-69, 8.550953543
377912e-73, 5.921712980178603e-76, 4.1009092660516466e-79, 2.839964865686735e-82, 1.96673467152821e-85, 1.3620046201718906e-88, 9.43
216495963910 7e-92, 6.531970193655889e-95, 4.523525064858635e-98, 3.132635086467189e-101, 2.16941487982491e-104, 1.502364875225e-107,
1.0404188886599714e-110, 7.205116957479029e-114, 4.9896931838497434e-117, 3.4554661938017496e-120, 2.3929821286715647e-123, 1.657189
8398002444e-126, 1.1476383932134632e-129, 7.947634302032286e-133, 5.5039018712134835e-136, 3.811566392807109e-139, 2.639588914686359
3e-142, 1.8279701625251764e-145, 1.2659073147681252e-148, 8.76667115490388e-152, 6.071101907828162e-155, 4.20436420209705 2e-158, 2.9
116095582389458e-161, 2.016350109583759e-164, 1.3963643418170048e-167, 1.1947642399671836e-165, 1.722633374154293e-166, 1.5115913078
615098e-167, 1.0939902793151105e-165, 1.4913890405364938e-166, 2.614392096103336e-167, 5.801091136550932e-166, 3.893068833310067e-16
7, 3.506652104996368e-166, 3.910075585203635e-168, 4.600881518338293e-165, 9.906283122907853e-166, 1.2572225929919686e-166, 4.355726
424458385e-167, 3.0128644289609236e-166, 5.316250916613461e-169, 3.4685917369179287e-164, 8.316780987338814e-165, 1.894138360969125e
-165, 3.369317128097715e-166, 2.7158737931627355e-168, 6.682292099490255e-165, 1.4964708917957485e-165, 2.4264467690733676e-166, 1.09
35101552277193e-168, 1.6794227873383314e-164, 3.958382292797412e-165, 8.347487438144936e-166, 9.123464938862097e-167, 9.081289035600
542e-167, 9.165244417258366e-167, 8.998977835748114e-167, 9.331637154446782e-167, 8.679379064906718e-167, 1.0011231360363336e-166,
7.498345464186302e-167, 1.3082755253682718e-166, 3.9055346875540324e-167, 3.4917788463239415e-166, 3.7725719609272793e-168, 4.773402
776237081e-165, 1.0325143954854319e-165, 1.351689498026748e-166, 3.556851641029125e-167, 3.947927560376874e-166, 8.76047999869492e-1
68, 1.9808619287214002e-165]
```

In [31]:

```python
#Plot function value decrease over all iterations
plt.plot([x[0] for x in res],[x[2] for x in res])
plt.xlabel('Iteration Number')
plt.ylabel('Function Value')
plt.title('Geometric Descent for Basic Quadratic Function')
plt.show()
```



EECS 127 Project: Speeding Up Gradient Descent

Aarush Aitha, Ethan Preston, Henry Tsai

May 3rd 2023

1 Introduction

2 Problems

1. Overview of Relevant Literature

   a) i) The main assumptions made on the function f to optimize are that the function is convex, continuous, L-smooth, and twice-differentiable. Later in the paper, the authors additionally assume that f is m-strongly convex.

   ii) The update rule at the k^{th} step is as follows:

$$\begin{gathered}\text{Calculate the smallest index } i \geq 0 \text{ for which:}\\ f(y_k) - f(y_k - 2^{-i}\alpha_{k-1}f'(y_k)) \geq 2^{-i-1}\alpha_{k-1}\left\|f'(y_k)\right\|_2^2\\ \alpha_k = 2^{-i}\alpha_{k-1},\ x_k = y_k - \alpha_k f'(y_k)\\ a_{k+1} = (1+\sqrt{4a_k^2+1})/2\\ y_{k+1} = x_k + (a_k - 1)(x_k - x_{k-1})/a_{k+1}\end{gathered}$$

   iii) After T iterations, we have the following upper bound:

$$\begin{gathered} f(\vec{x}_T) - \min_{\vec{x}\in R^n} f(\vec{x}) \leq \frac{C}{(T+2)^2}\\ \text{Where: } C = 4L\left\|\vec{y}_0 - \vec{x}^*\right\|_2^2\end{gathered}$$

   b) i) The authors claim that primal progress is made by gradient descent, and that dual progress is made by mirror descent.

   ii) The authors assume that f is L-smooth, convex, and differentiable, and that the regularization function w is 1-strongly convex.

   iii) After T iterations of AGM, we have the following upper bound:

$$f(\vec{y}_T) - \min_{\vec{x}\in Q} f(\vec{x}) \leq \frac{4\theta L}{(T+1)^2}$$

   iv) This paper proved the same bound on the number of steps required to reach the optimum as Nesterov's paper, but did so with linear coupling with Gradient Descent and Mirror Descent.

2. Mirror Descent

a) From First-Order Condition for strongly-convex functions:

$h(\vec{y}) \geq h(\vec{x}) + < \nabla_{\vec{x}} h(\vec{x}), \vec{y} - \vec{x} > + \frac{\alpha}{2} \|\vec{y} - \vec{x}\|_2^2, \forall \vec{x}, \vec{y} \in X$

$\nabla_{\vec{y}} D_h(\vec{y}, \vec{x}) = \nabla_{\vec{y}}(h(\vec{y}) - h(\vec{x}) - \nabla_{\vec{x}} h(\vec{x})^T \vec{y} - \nabla_{\vec{x}} h(\vec{x})^T \vec{x}$

$\nabla_{\vec{y}} D_h(\vec{y}, \vec{x}) = \nabla_{\vec{y}} h(\vec{y}) - \nabla_{\vec{x}} h(\vec{x})$

To prove that $D_h(\vec{y}, \vec{x})$ is α-strongly convex with respect to $\vec{y}$, we show that $\forall \vec{a}, \vec{b} \in X, D_h(\vec{b}, \vec{x}) \geq D_h(\vec{y}, \vec{x}) + [\nabla_{\vec{a}} D_h(\vec{a}, \vec{x})]^T (\vec{b} - \vec{a}) + \frac{\alpha}{2} \left\|\vec{b} - \vec{a}\right\|_2^2$

$h(\vec{b}) - h(\vec{x}) - \nabla_{\vec{x}} h(\vec{x})^T (\vec{b} - \vec{x}) \geq h(\vec{a}) - h(\vec{x}) - \nabla_{\vec{x}} h(\vec{x})^T (\vec{a} - \vec{x}) + [\nabla_{\vec{a}} h(\vec{a}) - \nabla_{\vec{x}} h(\vec{x})]^T (\vec{b} - \vec{a}) + \frac{\alpha}{2} \left\|\vec{b} - \vec{a}\right\|_2^2$

$h(\vec{b}) - h(\vec{x}) - \nabla_{\vec{x}} h(\vec{x})^T \vec{b} + \nabla_{\vec{x}} h(\vec{x})^T \vec{x} \geq h(\vec{a}) - h(\vec{x}) - \nabla_{\vec{x}} h(\vec{x})^T \vec{a} + \nabla_{\vec{a}} h(\vec{x})^T \vec{x} + \nabla_{\vec{a}} h(\vec{a})^T \vec{b} - \nabla_{\vec{a}} h(\vec{a})^T \vec{a} - \nabla_{\vec{x}} h(\vec{x})^T \vec{b} + \nabla_{\vec{x}} h(\vec{x})^T \vec{a} + \frac{\alpha}{2} \left\|\vec{b} - \vec{a}\right\|_2^2$

$h(\vec{b}) \geq h(\vec{a}) + \nabla_{\vec{a}} h(\vec{a})^T \vec{b} - \nabla_{\vec{a}} h(\vec{a})^T \vec{a} + \frac{\alpha}{2} \left\|\vec{b} - \vec{a}\right\|_2^2$

$h(\vec{b}) \geq h(\vec{a}) + < \nabla_{\vec{a}} h(\vec{a}), \vec{b} - \vec{a} > + \frac{\alpha}{2} \left\|\vec{b} - \vec{a}\right\|_2^2$, which is true by the first-order condition for $h(\vec{x})$, which is α-strongly convex. Holds for all $\vec{a}, \vec{b} \in X$

Therefore, $D_h(\vec{y}, \vec{x})$ is a α-strongly convex function in $\vec{y}$.

b) $D_h(\vec{u}, \vec{x}) - D_h(\vec{u}, \vec{y}) - D_h(\vec{y}, \vec{x})$

$(h(\vec{u}) - h(\vec{x}) - < \nabla h(\vec{x}), \vec{x} - \vec{u} >) - (h(\vec{u}) - h(\vec{y}) - < \nabla h(\vec{y}), \vec{u} - \vec{y} >) - (h(\vec{y}) - h(\vec{x}) - < \nabla h(\vec{x}), \vec{y} - \vec{x} >)$

$h(\vec{u}) - h(\vec{u}) - h(\vec{x}) + h(\vec{x}) + h(\vec{y}) - h(\vec{y}) - < \nabla h(\vec{x}), \vec{u} - \vec{x} > + < \nabla h(\vec{y}), \vec{u} - \vec{y} > + < \nabla h(\vec{x}), \vec{y} - \vec{x} >$

$-\nabla h(\vec{x})^T \vec{u} + \nabla h(\vec{x})^T \vec{x} + \nabla h(\vec{y})^T \vec{u} - \nabla h(\vec{y})^T \vec{y} + \nabla h(\vec{x})^T \vec{y} - \nabla h(\vec{X})^T \vec{x}$

$\nabla h(\vec{x})^T \vec{y} - \nabla h(\vec{x})^T \vec{u} - \nabla h(\vec{y})^T \vec{y} + \nabla h(\vec{y})^T \vec{u}$

$< \nabla h(\vec{x}) - \nabla h(\vec{y}), \vec{y} - \vec{u} >$

c) $Mirr(\eta \vec{g}, \vec{x}) = argmin_{\vec{z}} \;\; \eta \vec{g}^T \vec{z} + h(\vec{z}) - h(\vec{x}) - \nabla h(\vec{x})^T (\vec{z} - \vec{x})$

s.t. $\forall i, z_i \geq 0, \sum_{i=1}^n z_i = 1$

$\frac{\partial}{\partial x_i} h(\vec{x}) = \frac{\partial}{\partial x_i} (x_i log(x_i) - x_i) = log(x_i) + 1 - 1 = log(x_i)$

Therefore, $\nabla h(\vec{x}) = log(\vec{x})$

$argmin_{\vec{z}} \;\; \eta \vec{g}^T \vec{z} + \sum_{i=1}^n z_i log(z_i) - z_i - h(\vec{x}) - log(\vec{x})^T \vec{z} - log(\vec{x})^T \vec{x}$ [terms with only $\vec{x}$ will not be relevant to the argmin in this case]

s.t. $\forall i, -z_i \leq 0, \sum_{i=1}^n z_i - 1 = 0$

- Let some arbitrary vector $\vec{p} = [\frac{1}{n} \frac{1}{n} \cdots \frac{1}{n}]^T, \vec{p} \in \mathbb{R}^n$. Then, we have $\forall i, -p_i < 0$ and $\sum_{i=1}^n p_i - 1 = 0$, so by Slater's condition strong duality holds.
- The objective of the original optimization problem is the sum of an affine function in $\vec{z}, \vec{g}^T \vec{z}$ and a α-strongly convex function in $\vec{z}, D_h(\vec{z}, \vec{x})$ (as proven in part A), so the objective is convex and we have linear constraints.
- Since we have a convex optimization problem and strong duality holds, the KKT conditions are

necessary and sufficient for optimality.

i. Primal feasibility: $\forall i, z_i^* \geq 0$
$\sum_{i=1}^n z_i = 1$

ii. Dual feasibility: $\forall i, \lambda_i^* \geq 0$

iii. Complementary slackness: $\forall i, \lambda_i^* z_i^* = 0$

iv. Stationarity: $\nabla_{\vec{z}} L(\vec{z}^*, \vec{\lambda}^*, v^*$
$\frac{\partial}{\partial z_i}(z_i log(z_i) - z_i log(x_i) - z_i) = log(z_i) - log(x_i)$
We can ensure that complementary slackness holds by setting $\lambda^* = \vec{0}$. v^* is unconstrained so we can set it to 0. Then,
$\eta\vec{g}_i + log(\vec{z}_i^*) - log(\vec{x}_i) - \lambda_i + v = \vec{0}$
$log(\vec{z}_i^*) = log(\vec{x}_i) - \eta\vec{g}_i + \lambda_i - v$
$\vec{z}_i^* = x_i * exp(-v + \lambda_i - \eta\vec{g}_i)$
$\sum_{i=1}^n z_i = 1$
$\sum_{i=1}^n x_i * exp(-n * g_i - v) = 1$
$\sum_{i=1}^n x_i * \frac{e^{-n*g_i}}{e*v} = 1$
$v = log(\sum_{i=1}^n x_i * exp(-n * g_i) \rightarrow z_i = x_i * exp(-n * g_i - log(\sum_{i=1}^n x_i * exp(-n * g_i))$
$z*_i = \frac{x_i * exp(-n*g_i)}{\sum_{j=1}^n x_j * exp(-n*g_j)}$
$\vec{z*} = [z_1 ... z_n]^T$

d) $Mirr(\eta\vec{g}), \vec{x}) = argmin_{\vec{z}} \ \ \eta\vec{g}^T\vec{z} + h(\vec{z}) - h(\vec{x}) - \nabla h(\vec{x})^T(\vec{z} - \vec{x})$
$\nabla h(\vec{x}) = \nabla(\frac{1}{2}\|\vec{x}\|_2^2) = \vec{x}$
$= argmin_z \ \ \eta\vec{g}^T\vec{z} + \frac{1}{2}\|\vec{x}\|_2^2 - h(\vec{x}) - \vec{x}^T\vec{z} + \nabla h(\vec{x})^T\vec{x}$ [the terms only containing $\vec{x}$ will not be relevant to the argmin]
Again, the objective function is convex as shown above, so we can take the gradient and set it to 0 to find the minimizer.
$\nabla_{\vec{z}}(\eta\vec{g}^T\vec{z} + \frac{1}{2}\|\vec{z}\|_2^2 - \vec{x}^T\vec{z}) = \vec{0}$
$\eta\vec{g} + \vec{z} - \vec{x} = \vec{0}$
$\vec{z}^* = \vec{x} - \eta\vec{g}$ [This is the gradient descent update step, if $\vec{g} = \nabla_{\vec{x}} f(\vec{x})$ and η is the step size]
$D_h(\vec{y}, \vec{x}) = h(\vec{y}) - h(\vec{x}) - \nabla h(\vec{x})^T(\vec{y} - \vec{x})$
$= \frac{1}{2}\|\vec{y}\|_2^2 - \frac{1}{2}\|\vec{x}\|_2^2 - \vec{x}^T\vec{y} + \vec{x}^T\vec{x}$
$= \frac{1}{2}\|\vec{y}\|_2^2 + \frac{1}{2}\|\vec{x}\|_2^2 - \vec{x}^T\vec{y}$
$= \frac{1}{2}\|\vec{x} - \vec{y}\|_2^2$ [This is the squared error between $\vec{x}$ and $\vec{y}$.]

e) $< \eta_k\vec{g}_k, \vec{x}_k - \vec{x}_{k+1} > + D_h(\vec{u}, \vec{x}) - D_h(\vec{u}, \vec{x}_{k+1}) - D_h(\vec{x}_{k+1}, \vec{x}_k)$
$=< \eta_k\vec{g}_k, \vec{x}_k - \vec{x}_{k+1} > + < \nabla h(\vec{x}_k) - \nabla h(\vec{x}_{k+1}), \vec{x}_{k+1} - \vec{u} >$ [Bregman 3-point equality]
$= \eta_k\vec{g}_k^T\vec{x}_k - \eta_k\vec{g}_k^T\vec{x}_{k+1} + \nabla h(\vec{x}_k)^T\vec{x}_{k+1} - \nabla h(\vec{x}_k)^T\vec{u} - \nabla h(\vec{x}_{k+1})^T\vec{x}_{k+1} + \nabla h(\vec{x}_{k+1})^T\vec{u}$
$= \eta_k\vec{g}_k^T\vec{x}_k - \eta_k\vec{g}_k^T\vec{x}_{k+1} + \vec{x}_k^T\vec{x}_{k+1} - \vec{x}_k^T\vec{u} - \|\vec{x}_{k+1}\|_2^2 + \vec{x}_{k+1}^T\vec{u}$

$= \eta_k \vec{g}_k^T \vec{x}_k - \eta_k \vec{g}_k^T(\vec{x}_k - \eta_k \vec{g}_k) + \vec{x}_k^T(\vec{x}_k - \eta_k \vec{g}_k) - \vec{x}_k^T \vec{u} - \|\vec{x}_k - \eta_k \vec{g}_k\|_2^2 + (\vec{x}_k - \eta_k \vec{g}_k)^T \vec{u}$

$= \eta k^2 \|\vec{g}_k\|_2^2 + \|\vec{x}\|_2^2 - \eta_k \vec{x}_k^T \vec{g}_k - \|\vec{x}_k\|_2^2 + \eta_k \vec{x}_k^T \vec{g}_k + \eta_k \vec{g}_k^T \vec{x}_k - \eta_k^2 \|\vec{g}_k\|_2^2 - \eta_k \vec{g}^T \vec{u}$

$= \eta_k \vec{g}_k^T \vec{x}_k - \eta_k \vec{g}_k^T \vec{u} =< \eta_k \vec{g}_k, \vec{x}_k - \vec{u} >$ [First equality proven]

$< \eta_k \vec{g}_k, \vec{x}_k - \vec{x}_{k+1} > -D_h(\vec{x}_{k+1}, \vec{x})$

$=< \eta_k \vec{g}_k > -(\frac{1}{2}\|\vec{x}_{k+1}\|_2^2 - \frac{1}{2}\|\vec{x}_k\|_2^2 - < \vec{x}_k, \vec{x}_{k+1} - \vec{x}_k >)$

$= \eta_k^2 \|\vec{g}_k\|_2^2 - (\frac{1}{2}\|\vec{x}_{k+1}\|_2^2 - \vec{x}_k^T \vec{x}_{k+1})$

$= \eta_k^2 \|\vec{g}_k\|_2^2 - \frac{1}{2}\|vecx_k - \vec{x}_{k+1}\|_2^2 = \eta_k^2 \|\vec{g}_k\|_2^2 - \frac{1}{2}\|\eta_k \vec{g}_k\|_2^2 = \frac{\eta_k^2 \|\vec{g}_k\|_2^2}{2}$ [Second equality proven]

f) $TotalReg_T(\vec{u}) = \sum_{k=0}^T < \eta_k \vec{g}_k, \vec{x}_k - \vec{u} >= \sum_{k=0}^T (\frac{\eta_k^2 \|\vec{g}_k\|_2^2}{2} + D_h(\vec{u}, \vec{x}_k) - D_h(\vec{u}, \vec{x}_{k+1}))$

$= \sum_{k=0}^T (\frac{\eta_k^2 \|\vec{g}_k\|_2^2}{2}) + ((D_h(\vec{u}, \vec{x}_0) - D_h(\vec{u}, \vec{x}_1)) + ((D_h(\vec{u}, \vec{x}_1) - D_h(\vec{u}, \vec{x}_2)) + ... + (D_h(\vec{u}, \vec{x}_T) - D_h(\vec{u}, \vec{x}_{T+1})))$

[Intermediate terms cancel out]

$= \sum_{k=0}^T (\frac{\eta_k^2 \|\vec{g}_k\|_2^2}{2}) + D_h(\vec{u}, \vec{x}_0) - D_h(\vec{u}, \vec{x}_{T+1})$

$\leq \sum_{k=0}^T \eta_k^2 \|\vec{g}_k\|_2^2 + D_h(\vec{u}, \vec{x}_0)$

g) $TotalReg_T(\vec{u}) = \sum_{k=0}^T < \eta_k \vec{g}_k, \vec{x}_k - \vec{u} >= \eta \sum_{k=0}^T < \vec{g}_k, \vec{x}_k - \vec{u} >$

Using the convexity of f, by the first-order condition we have $\forall \vec{u}, f(\vec{u}) \geq f(\vec{x}) + < \nabla f(\vec{x}), \vec{u} - \vec{x} >$

$\Rightarrow f(\vec{u}) - f(\vec{x}) \geq \ < \nabla f(\vec{x}), \vec{u} - \vec{x} >$

$\Rightarrow f(\vec{x}) - f(\vec{u}) \leq \ < \nabla f(\vec{x}), \vec{x} - \vec{u} >$

$\Rightarrow < \vec{g}_k, \vec{x}_k - \vec{u} > \ \geq f(\vec{x}_k) - f(\vec{u})$

$\eta \sum_{k=0}^T < \vec{g}_k, \vec{x}_k - \vec{u} > \ \geq \ \ \eta \sum_{k=0}^T f(\vec{x}_k) - f(\vec{u}) = \eta \sum_{k=0}^T f(\vec{x}_k) - T\eta f(\vec{u})$

Because f is convex, we have that $f(\vec{x}) \leq \frac{1}{T} \sum_{k=0}^T f(\vec{x}_k)$ [By Jensen's Inequality]

$\Rightarrow T\eta f(\vec{x}_T) \leq \eta \sum_{k=0}^T f(\vec{x}_k)$

$\eta \sum_{k=0}^T f(\vec{x}_k) - T\eta f(\vec{u}) \geq T\eta f(\vec{x}_T) - T\eta f(\vec{u}) = T\eta(f(\vec{x}_T) - f(\vec{u}))$

Result from part (f): $TotalReg_T(\vec{u}) \leq \sum_{k=0}^T \eta_k^2 \|\vec{g}_k\|_2^2 + D_h(\vec{u}, \vec{x}_0)$

$T\eta(f(\vec{x}_T) - f(\vec{u})) \leq TotalReg_T(\vec{u}) \leq \sum_{k=0}^T \eta_k^2 \|\vec{g}\|_2^2 + D_h(\vec{u}, \vec{x}_0)$

$f(\vec{x}_T) - f(\vec{u}) \leq \frac{1}{T\eta}[\sum_{k=0}^T \eta_k^2 \|\vec{g}_k\|_2^2 + D_h(\vec{u}, \vec{x}_0)]$ [Assume $\eta_k = n \forall k$] $\leq \frac{1}{T}[\eta \sum_{k=0}^T \|\vec{g}_k\|_2^2 + D_h(\vec{u}, \frac{\vec{x}_0}{\eta}]$

Recall that $\vec{u}$ is any feasible vector $\in X$, so we can set $\vec{u} = \vec{x}^*$ to achieve the desired result:

$f(\vec{x}_T - f(\vec{x}^*) \leq \frac{1}{T}[\eta \sum_{k=0}^1 \|\vec{g}_k\|_2^2 + \frac{D_h(\vec{x}^*, \vec{x}_0)}{\eta}]$

$\Rightarrow f(\vec{x}_T) \leq f(\vec{x}^*) = \frac{1}{T}[\eta \sum_{k=0}^1 \|\vec{g}_k\|_2^2 + \frac{D_h(\vec{x}^*, \vec{x}_0)}{\eta}]$

h) L-Lipschitz implies that:

$\|f(\vec{x}) - f(\vec{y})\| \leq L\|\vec{x} - \vec{y}\|_2$

$\Rightarrow (f(\vec{x} - f(\vec{y}))^2 \leq L^2 \|\vec{x} - \vec{y}\|_2^2 \forall \vec{x}, \vec{y} \in X$

$\|\nabla f(\vec{x})\|_2 \leq L \forall \vec{x} \in X$

$\Rightarrow \|\nabla f(\vec{x})\|_2^2 \leq L^2 \forall \vec{x} \in X$

$f(\vec{x}_T) \leq f(\vec{x}^*) + \frac{1}{T}[\eta \sum_{k=0}^T \|\vec{g}_k\|_2^2 + \frac{D_h(\vec{x}^*, \vec{x}_0)}{\eta}]$

$\leq f(\vec{x}^*) + \frac{1}{T}[\eta \sum_{k=0}^T L^2 + \frac{D_h(\vec{x}^*, \vec{x}_0}{\eta}]$

$= f(\vec{x}^*) + \eta L^2 + \frac{D_h(\vec{x}^*, \vec{x}_0}{\eta}$

From part (d), $D_h(\vec{x}^*, \vec{x}_0) = \frac{1}{2}\|\vec{x}_0 - \vec{x}^*\|_2^2$

$= f(\vec{x}^*) + \eta L^2 + \frac{1}{2\eta T}\|\vec{x}_0 - \vec{x}^*\|_2^2$

$\nabla_\eta(f(\vec{x}^*) + \eta L^2 + \frac{1}{2\eta T}|\vec{x}_0 - \vec{x}^*\|_2^2) = 0$
$L^2 - \frac{1}{2T\eta^2}|\vec{x}_0 - \vec{x}^*\|_2^2 = 0$
$2TL^2\eta^2 = |\vec{x}_0 - \vec{x}^*\|_2^2$
$\eta^2 = \frac{|\vec{x}_0 - \vec{x}^*\|_2^2}{2TL^2}$
$\eta^* = \frac{|\vec{x}_0 - \vec{x}^*\|_2}{L\sqrt{2T}}$
Substituting η^*, we get
$= f(\vec{x}^*) + \frac{|\vec{x}_0 - \vec{x}^*\|_2}{L\sqrt{2T}} \cdot L^2 + \frac{|\vec{x}_0 - \vec{x}^*\|_2}{2T} \cdot \frac{L\sqrt{2T}}{|\vec{x}_0 - \vec{x}^*\|_2}$
$= f(\vec{x}^*) + \frac{L|\vec{x}_0 - \vec{x}^*\|_2}{\sqrt{2T}} + \frac{L|\vec{x}_0 - \vec{x}^*\|_2}{\sqrt{2T}} = f(\vec{x}^*) + \frac{\sqrt{2}L|\vec{x}_0 - \vec{x}^*\|_2}{\sqrt{T}}$

3. Accelerated Gradient Descent

a) $h(\vec{x}) = \frac{1}{2}\|\vec{x}\|_2^2$
$\vec{z}_{k+1} = \vec{z}_k - \eta_{k+1}\nabla f(\vec{x}_{k+1})$
Bregman divergence 3-point equality: $< \nabla h(\vec{x}) - \nabla h(\vec{y}), \vec{y} - \vec{u} >= D_h(\vec{u}, \vec{x}) - D_h(\vec{u}, \vec{y}) - D_h(\vec{y}, \vec{x})$
We can apply the same equality (1) from question (2)(e), replacing $\vec{g}_k$ with $\nabla f(\vec{x}_{k+1})$
$< \eta_{k+1}\nabla f(\vec{x}_{k+1}), \vec{z}_k - \vec{u} >=< \eta_{k+1}\nabla f(\vec{x}_{k+1}), \vec{z}_k - \vec{z}_{k+1} > +D_h(\vec{u}, \vec{z}_k) - D_h(\vec{u}, \vec{z}_{k+1}) - D_h(\vec{z}_{k+1}, \vec{z}_k)$
From here, we want to prove that $< \eta k + 1\nabla f(\vec{x}_{k+1}), \vec{z}_k - \vec{z}_{k+1} > -D_h(\vec{z}_{k+1}) \leq \frac{\eta_{k+1}^2}{2}\|\nabla f(\vec{x}_{k+1}\|_2^2$
From part (2d), we proved that with $h(\vec{x}) = \frac{1}{2}\|\vec{x}\|_2^2, \vec{z}_{k+1} = \vec{z}_k - \eta_{k+1}\nabla f(\vec{x}_{k+1})$ and $D_h(\vec{z}_{k+1}, \vec{z}_k) = \frac{1}{2}\|\vec{z}_k - \vec{z}_{k+1}\|_2^2$
Expanding the lefthand-side, we get:
$= \eta_{k+1}\nabla f(\vec{x}_{k+1})^T\vec{z}_k - \eta_{k+1}\nabla f(\vec{x}_{k+1})^T\vec{z}_{k+1} - \frac{1}{2}\|\vec{z}_k - \vec{z}_{k+1}\|_2^2$
$= \eta_{k+1}^2\|\nabla f(\vec{x}_{k+1})\|_2^2 - \frac{\eta_{k+1}^2}{2}\|\nabla f(\vec{x}_{k+1})\|_2^2 = \frac{\eta_{k+1}^2}{2}\|\nabla f(\vec{x}_{k+1})\|_2^2$

b) $< \eta_{k+1}\nabla f(\vec{x}_{k+1}), \vec{x}_{k+1} - \vec{u} >=< \eta_{k+1}\nabla f(\vec{x}_{k+1}), \vec{x}_{k+1} - \vec{z}_k >$
$=< \eta_{k+1}\nabla f(\vec{x}_{k+1}), \vec{z}_k - \vec{u} > + < \eta_{k+1}\nabla f(\vec{x}_{k+1}), \vec{x}_{k+1} - \vec{z}_k >$
$= \frac{\eta_{k+1}^2}{2}\|\nabla f(\vec{x}_{k+1})\|_2^2 + D(\vec{u}, \vec{z}_k) - D(\vec{u}, \vec{z}_{k+1}) + < \eta_{k+1}\nabla f(\vec{x}_{k+1}), \vec{x}_{k+1} - \vec{z}_k >$
$\vec{x}_{k+1} = \tau_k\vec{z}_k + (1 - \tau_k)\vec{y}_k \Rightarrow \vec{z}_k = \frac{\vec{x}_{k+1} - (1 - \tau_k\vec{y}_k}{\tau_k}$
$\vec{x}_{k+1} - \vec{z}_k = -(\vec{z}_k - \vec{x}_{k+1}) = -(\frac{\vec{x}_{k+1} - \tau_k\vec{x}_{k+1} - (1-\tau_k)\vec{y}_k}{\tau_k}) = \frac{-(1-\tau_k)(\vec{x}_{k+1} - \vec{y}_k)}{\tau_k} = \frac{(1-\tau_k)(\vec{y}_k - \vec{x}_{k+1})}{\tau_k}$
$\frac{\eta_{k+1}^2}{2}\|\nabla f(\vec{x}_{k+1})\|_2^2 + D(\vec{u}, \vec{z}_k) - D(\vec{u}, \vec{z}_{k+1}) + \frac{(1-\tau_k)\eta_{k+1}}{\tau_k}\nabla f(\vec{x}_{k+1}), \vec{y}_k - \vec{x}_{k+1} >$
Then, by the convexity of f, we have $\forall \vec{u}, f(\vec{u}) \geq f(\vec{x}) + < \nabla f(\vec{x}), \vec{u} - \vec{x} >$
Setting $\vec{u} = \vec{y}_k$ and $\vec{x} = \vec{x}_{k+1}$, we have that:
$< \nabla f(\vec{x}_{k+1}), \vec{y}_k - \vec{x}_{k+1} > \ \leq f(\vec{y}_k) - f(\vec{x}_{k+1})$ (first-order condition for f)
By L-smoothness of f, we have that $f(\vec{y}_{k+1}) \leq f(\vec{x}_{k+1}) - \frac{1}{2L}\|\nabla f(\vec{x}_{k+1})\|_2^2$
$\Rightarrow \frac{1}{2L}\|\nabla f(\vec{x}_{k+1})\|_2^2$
$\Rightarrow \|\nabla f(\vec{x}_{k+1})\|_2^2 \leq 2L(f(\vec{x}_{k+1}) - f(\vec{y}_{k+1}))$
Substituting into the righthand-side of the inequality above, we get:
$\leq \eta_{k+1}^2 L(f(\vec{x}_{k+1}) - f(\vec{y}_{k+1})) + D(\vec{u}, \vec{z}_k) - D(\vec{u}, \vec{z}_{k+1}) + \frac{(1-\tau_k)\eta_{k+1}}{\tau_k}(f(\vec{y}_k) - f(\vec{x}_{k+1}))$ [Completing the upper-bound]

c) We have the upper bound from the previous problem.

Negating the inequality from the previous problem, we get:

$-(\frac{(1-\tau_k)\frac{1}{\tau_k L}}{\tau_k}(f(\vec{y_k}) - \eta_{k+1}^2 L(f(\vec{x}_{k+1}) - f(\vec{y}_{k+1})) - D(\vec{u}, \vec{z}_k) + D(\vec{u}, \vec{z}_{k+1})))$

$\leq - < \eta_{k+1} \nabla f(\vec{x}_{k+1}), vecx_{k+1} - \vec{u} >$

$\frac{1-\tau_k}{\tau_k^2 L}(f(\vec{x}_{k+1}) - f(\vec{y}_k)) - \eta_{k+1}^2 L(f(\vec{x}_{k+1}) - f(\vec{y}_{k+1})) - D(\vec{u}, \vec{z}_k) + D(\vec{u}, \vec{z}_{k+1})$

$\leq - < \eta_{k+1} \nabla f(\vec{x}_{k+1}), \vec{x}_{k+1} - \vec{u} >$

$\frac{1}{\tau_k^2 L} f(\vec{x}_{k+1}) - \frac{1}{\tau_k L} f(\vec{x}_{k+1}) - \frac{1}{\tau_k^2 L} f(\vec{y}_k) + \frac{1}{\tau_k L} f(\vec{y}_k) - \frac{1}{\tau_k^2 L} f(\vec{x}_{k+1}) + \frac{1}{\tau_k^2 L} f(\vec{y}_{k+1}) - D(\vec{u}, \vec{z}_L) + D(\vec{u}, \vec{z}_{k+1}) \leq$

$< \eta_{k+1} \nabla f(\vec{x}_{k+1}), \vec{x}_{k+1} - \vec{u} >$

$\eta k + 1^2 L f(\vec{y}_{k+1}) - (\eta_{k+1}^2 L - \eta_{k+1}) f(\vec{y}_k) - \eta_{k+1} f(\vec{x}_{k+1}) - D(\vec{u}, \vec{z}_k) + D(\vec{u}, \vec{z}_{k+1})$

$\leq - < \eta_{k+1} \nabla f(\vec{x}_{k+1}), \vec{x}_{k+1} - \vec{u} >$

$\eta_{k+1}^2 L f(\vec{y}_{k+1}) - (\eta_{k+1}^2 L - \eta_{k+1}) f(\vec{y}_k) - \eta_{k+1} f(\vec{x}_{k+1}) - D(\vec{u}_k, \vec{z}_k) + D(\vec{u}, \vec{z}_{k+1})$

$\leq \eta_{k+1}(f(\vec{x}_{k+1}) - < \nabla f(\vec{x}_{k+1}), \vec{x}_{k+1} - \vec{u} >$

By convexity of f (using the first-order condition), we have:

$f(\vec{u}) \geq f(\vec{x}_{k+1}) + < \nabla f(\vec{x}_{k+1}), \vec{u} - \vec{x}_{k+1} > \Rightarrow f(\vec{u}) \geq f(\vec{x}_{k+1}) - < \nabla f(\vec{x}_{k+1}), \vec{x}_{k+1} - \vec{u} >$

$\eta_{k+1}^2 L f(\vec{y}_{k+1}) - (\eta_{k+1} L - \eta_{k+1}) f(\vec{y}_k) - D(\vec{u}, \vec{z}_k) + D(\vec{u}, \vec{z}_{k+1})$

$\leq \eta_{k+1} f(\vec{u})$

d) $\eta_{k+1}^2 L f(\vec{y}_{k+1}) - (\eta_{k+1}^2 * L - \eta_{k+1}) f(\vec{y}_k) - D(\vec{u}, z_k) + D(\vec{u}, z_{k+1}) \leq \eta_{k+1} f(\vec{u}) \; \forall k \in [0, T]$

$\sum_{k=0}^{T-1} \eta_{k+1} = \sum_{k=0}^{T-1} \frac{k+2}{2L} = \frac{1}{2L} \sum_{k=0}^{T-1} k + 2 = \frac{1}{2L}(2 + 3 + ... + (T+1)) = \frac{(T+1)(T+2)}{4L} - \frac{1}{2L} = \frac{T*(T+3)}{4L}$

$\eta_T^2 = \frac{(T+1)^2}{(2L)^2}$

$\frac{(T+1)^2}{4L} f(\vec{y}_t) + \frac{(T-1)^2}{4L} f(\vec{x*}) - \frac{1}{2}||\vec{x*} - \vec{x}_0||_2^2 \leq \frac{(T*(T+3))}{4L} * f(\vec{x*})$

$(T+1)^2 f(\vec{y}_T) \leq (T^2 + 2T + 1) f(\vec{x*}) + 2L||\vec{x*} - \vec{x}_0||_2^2$

$f(\vec{(}y_T) \leq f(\vec{x*}) + \frac{2L||\vec{x*} - \vec{x}_0||_2^2}{(T+1)^2}$

3 Extension

Please see attached PDF below

4 Code Appendix

Please refer to the Jupyter Notebook attached below.

5 Sources

EECS 127 Project: Accelerating Gradient Descent

Introduction

Our project focused on accelerating gradient descent. Specifically, we focused on algorithms similar to Nesterov's accelerated gradient descent method but with derivations considered either algebraically or geometrically more intuitive.

We considered existing work from three papers: “"A Method of Solving a Convex Programming Problem with Convergence Rate O($1/k^2$)", Nesterov; "Linear Coupling: An Ultimate Unification of Gradient and Mirror Descent", Zhu et al.; and “A geometric alternative to Nesterov's accelerated gradient descent”; Bubeck et al. In his seminal paper, Nesterov described a method that converged faster than “vanilla” gradient descent for functions that met the following criterion: convex, differentiable (and thus continuous), and L-smooth. Zhu et al. described how gradient descent and mirror descent are complementary and therefore can be linearly coupled.

The primary open problem researchers contend with today is understanding the intuition behind exactly why Nesterov's accelerated gradient descent (AGD) works. One group led by Weijie Su has proposed that AGD can be viewed as a discrete solution to an ordinary differential equation. Other approaches have included treating the search space of the problem as an ellipsoid, such as that by Yin-Tat Lee. In our project we focused on the ellipsoid approach, specifically by assuming the ellipsoids are n-dimensional spheres as proposed by Bubeck et al.

In this paper, the authors propose a new method for unconstrained optimization of a smooth and strongly convex function. This new method that they propose can be viewed as a combination of gradient and the ellipsoid method, where the squared distance to the optimal point decreases at a rate of 1. Since the intuition behind Nesterov's algorithm can be extremely difficult to grasp, the authors propose this new method that has much clearer intuition and also achieves acceleration. We know that the function at hand is strongly convex, so the gradient will give a ball containing the optimal solution, which we can call Ball A. We also have the Ball B, which contains the optimum, which it obtained from previous iterations. The algorithm that achieves acceleration with the gradient can be used to shrink ball B, so the ball containing the intersection of the two balls shrinks faster.

Methodology

For our project, we implemented several common variants of gradient descent in Python using the cvxpy and numpy libraries. The variants we implemented were “vanilla” gradient descent, MirrorDescent, accelerated gradient method, adaptive gradient method, and adaptive moment estimation.

Our work primarily took the form of multiparameter methods representing smaller discrete mathematical methods. We first implemented a method for mirrorstep for an entropy regularizer, which used the L2-norm. This function took in values for starting parameters, loss function, learning rate (η), and the gradient of the loss function. Next, we used this function in our implementation of vanilla gradient descent, which took in the same parameters in addition to the loss function and the number of training iterations.

Our remaining implementations were simply derived from existing literature in the same manner as vanilla gradient descent, with the exception of two changes. We substituted in the Lipshitz constant for the learning rate (η) in accelerated gradient descent, and we added parameters for tolerances ($/epsilon$) in adaptive gradient method to ensure convergence.

Next, we called each function on a provided dataset to compare their performance relative to each other. To visualize the results we used the plotting library matplotlib to generate line charts for each function’s value over 100 training iterations.

For geometric descent, we followed the algorithm as described in Bubeck et al. The algorithm essentially works as follows: We start with our initial point, which could be randomly generated. From our initial point, we move a certain length (dependent on step size) in the direction of the negative gradient and create our first “ball” (the aggregate ball) centered at that destination. The radius of the ball can be shrunk based on the properties of f. From there we maintain one ball that represents the aggregation of the previous gradients, as well as the current gradient. We find the minimum enclosing ball of the intersection of these two balls, and step towards the center of it to try to find the optimum. Each iteration will update the radii of the two balls, but the aggregate ball will remain in the same place where it was originally created.

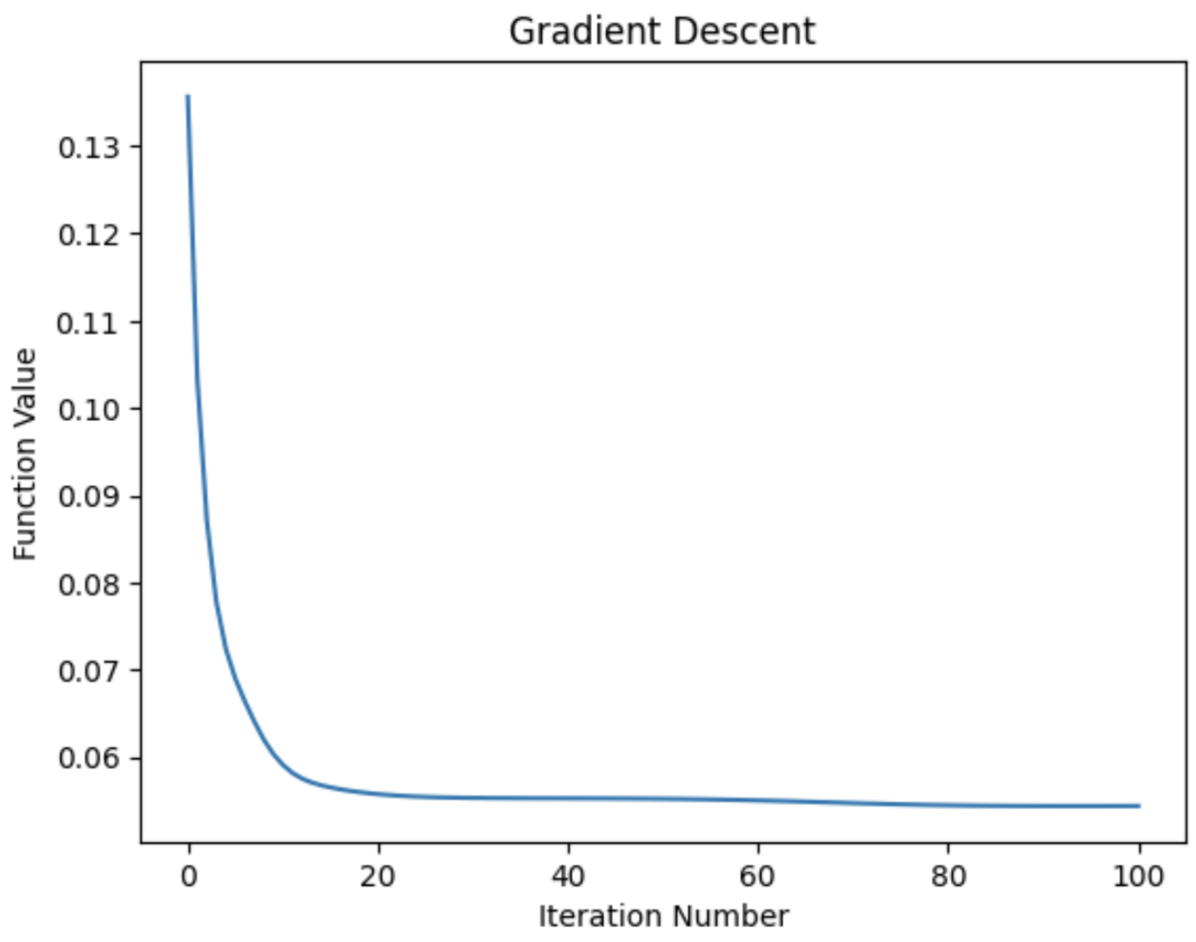
We created a helper function called min_enclosing_ball to calculate the center and radius of the smallest ball that encloses the intersection of Ball A and B as defined above (refer to Bubeck et al). This helper function was used in our implementation of geometric descent. We wrote the geometric descent function based on provided literature as we did above and also generated a plot of function value versus training iterations to compare it with the previous methods.

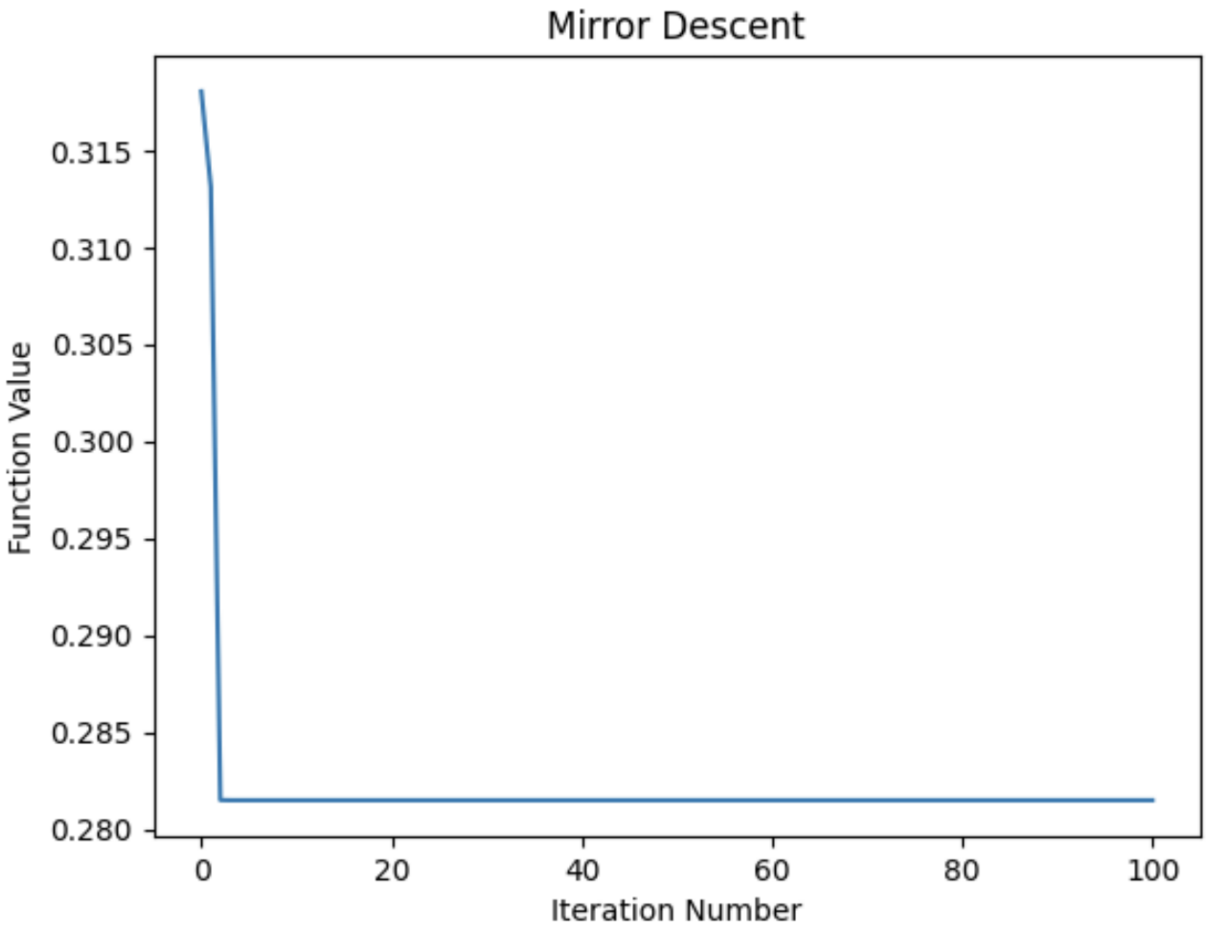
We additionally implemented a basic line search algorithm. Line search works by searching alone the line connecting the two input points, $\vec{x}$ and $\vec{y}$, and evaluates the function at various points along the line in an attempt to find the point that minimizes the objective function $f(\vec{x} + t(\vec{y} - \vec{x}))$. One might think that the general line search algorithm is simply searching along all possible convex combinations of the inputs, since $f(\vec{x} + t(\vec{y} - \vec{x}))$ simplifies to $f(t\vec{y} + (1-t)\vec{x})$. However, because t is an unconstrained real number as specified in the optimization problem from the paper, we are actually searching along the entire line extending past the two points, not just the line segment between them. We cannot evaluate all of the points along the line because there is an infinite number of points, so we must settle for some kind of approximation of the true algorithm.

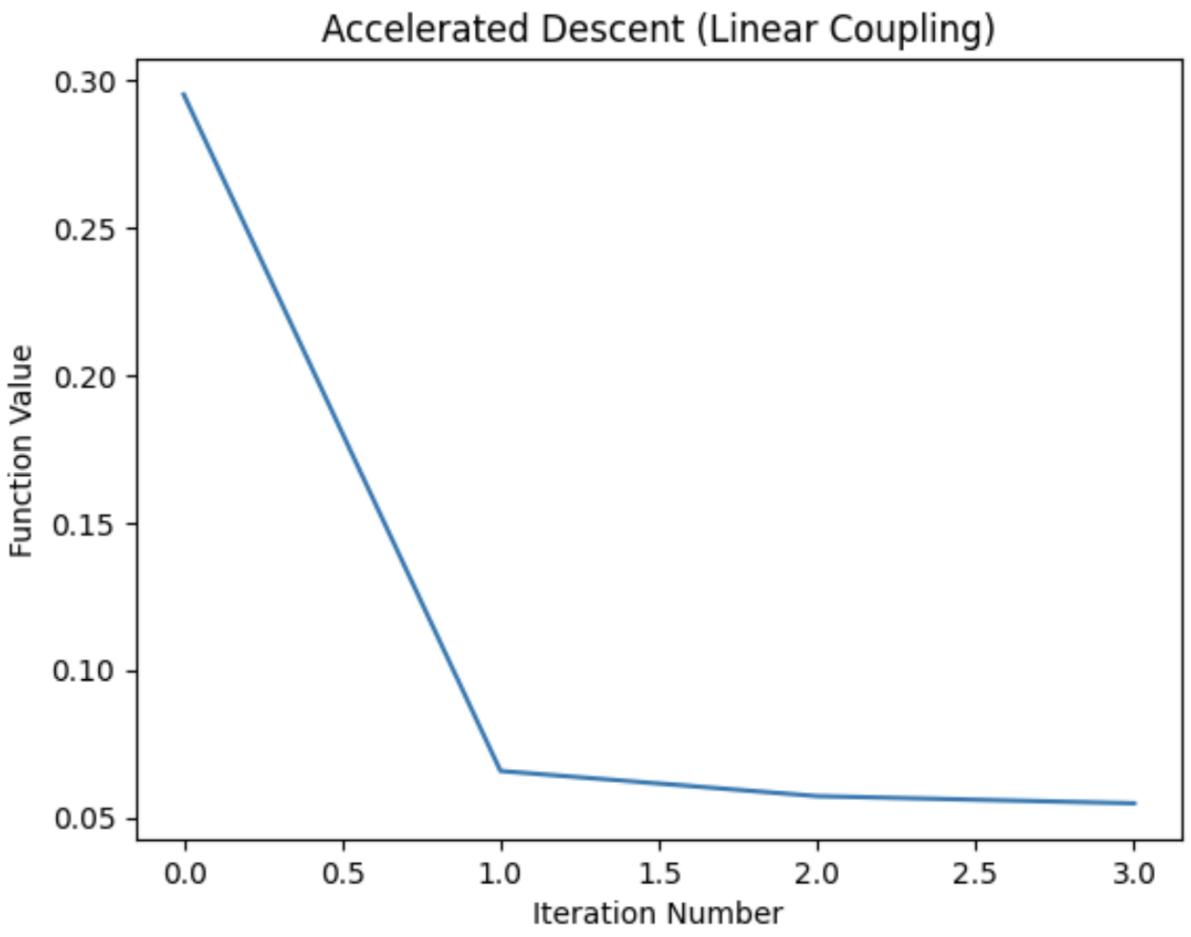
We explored using scipy.optimize.line_search, but it was unable to converge with the given inputs. We also tried using cvxpy to directly implement the optimization problem for line search, but we encountered some technical errors that prevented us from doing so. Instead we decided to create our own, custom line search algorithm so that we could adjust the tolerance rate for our needs. Our custom algorithm uses the golden section search method, which is a way of finding the minimum of a function inside a specified interval. Our specific line search algorithm actually specified that t be in the range of 0 to 1, so we don’t spend an inordinate amount of time searching for spurious solutions.

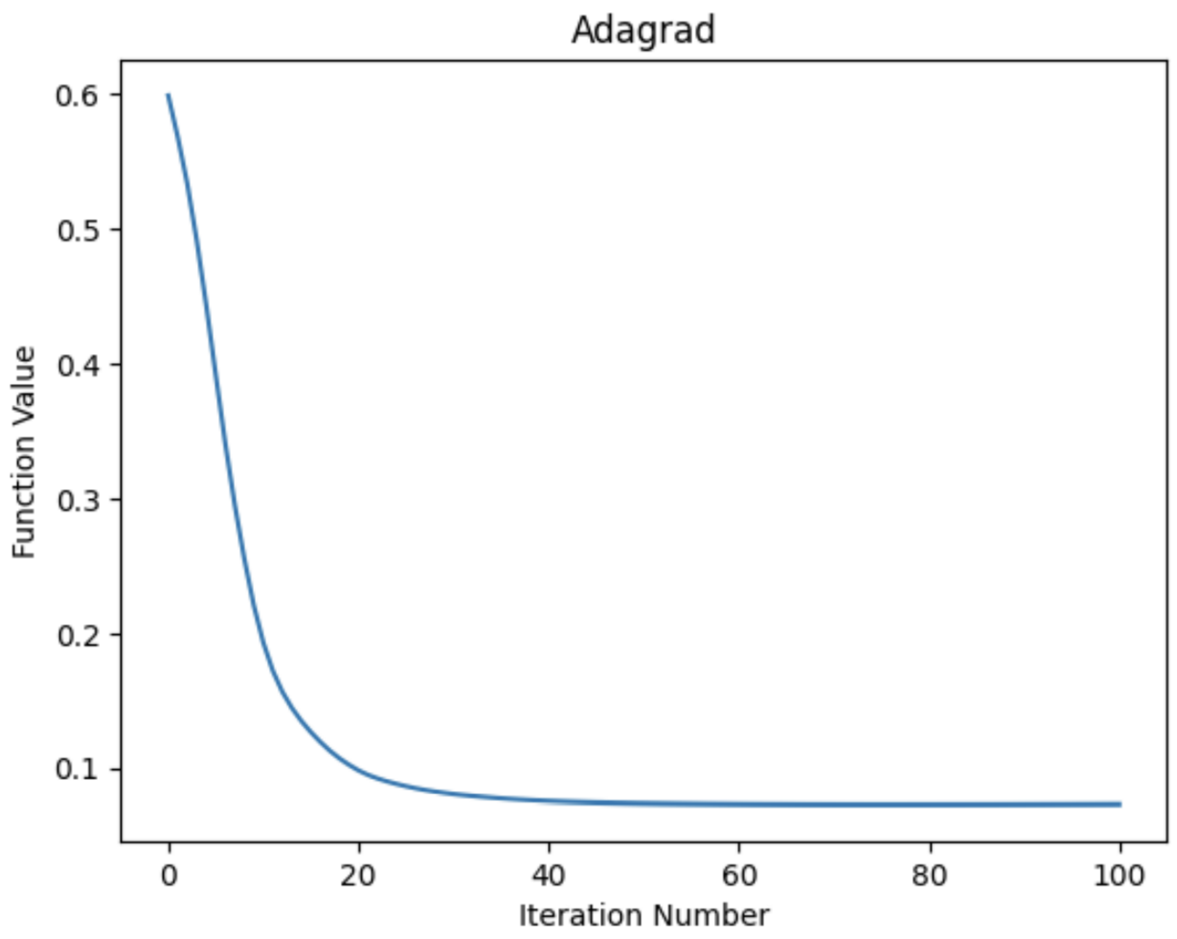
Results

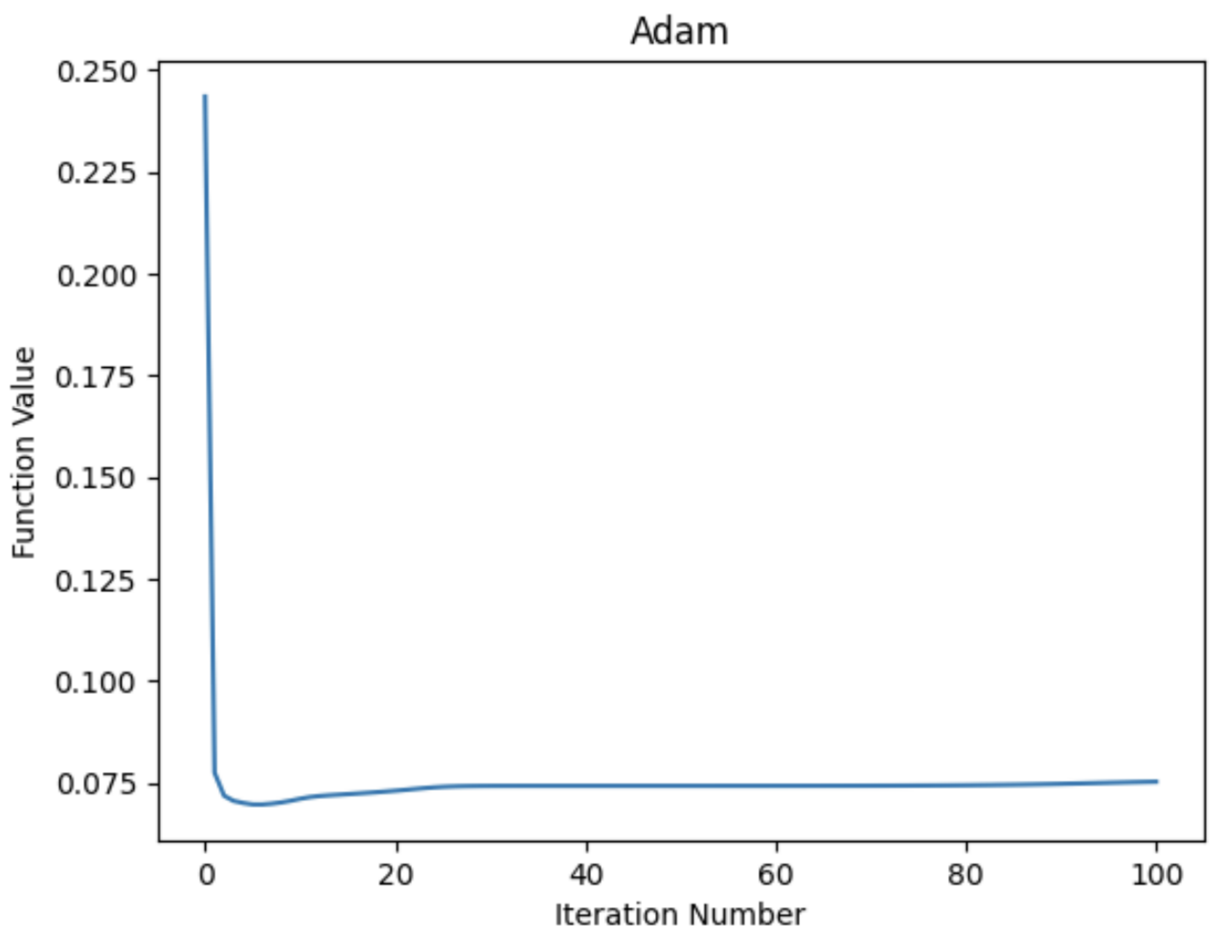
We present the following graphs showing the different rates of convergence for the algorithms used above:

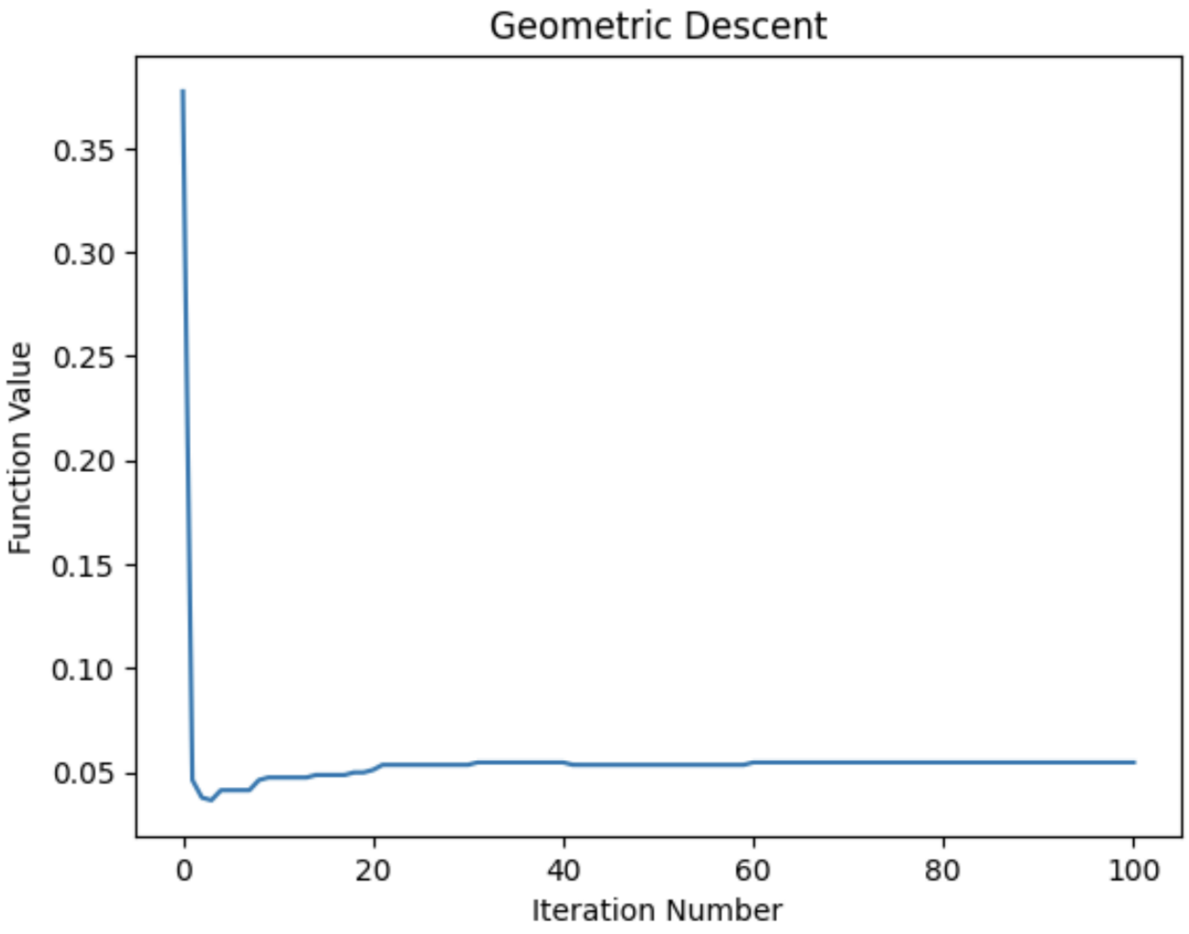


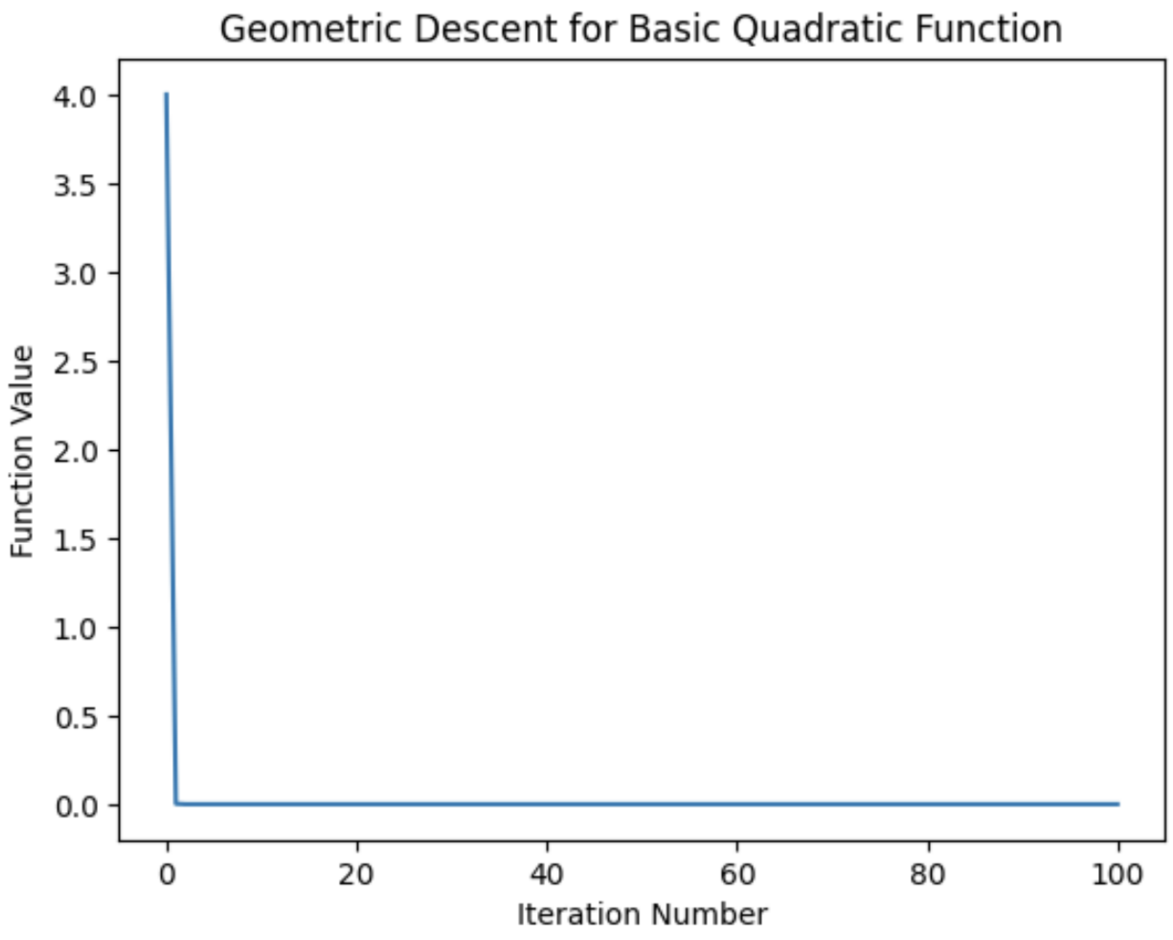

Mirror Descent
Function Value
0.315
0.310
0.305
0.300
0.295
0.290
0.285
0.280
Iteration Number
0
20
40
60
80
100


Accelerated Descent (Linear Coupling)
Function Value
Iteration Number
0.30
0.25
0.20
0.15
0.10
0.05
0.0
0.5
1.0
1.5
2.0
2.5
3.0


Adagrad
Function Value
Iteration Number
0.6
0.5
0.4
0.3
0.2
0.1
0
20
40
60
80
100


Adam
Function Value
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0
20
40
60
80
100
Iteration Number


Geometric Descent
Function Value
Iteration Number
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
20
40
60
80
100



As you can see, several of the variants were able to converge faster than standard gradient descent. In particular, mirror descent, adam, and geometric descent were all able to converge in under 5 iterations, in comparison to the 20 iterations necessary for standard gradient descent. We were able to achieve these results by changing the hyperparameters involved in the training process, such as epsilon and stepsize, in order to optimize the convergence rate. In addition, our function exhibited similar runtimes to Nesterov, which was proven in the provided paper.