Homework 9 Solutions:: MATH 504

Your homework submission must be a single pdf called "LASTNAME-hw9.pdf" with your solutions to all theory problem to receive full credit. All answers must be typed in Latex.

Consider "Rosenbrock" function

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2.$$

This function is known as the banana function because of the shape of its level sets.

a. Prove that $x^* = [1, 1]^\mathsf{T}$ is the unique global minimizer of f over $\mathbb{R}^2$.

b. Write a method with signature function xsol = GradDescent (f, grad, x0)

   The input f and grad are function handles. The function f: $\mathbb{R}^N \to \mathbb{R}$ is an arbitrary objective function, and grad: $\mathbb{R}^N \to \mathbb{R}^N$ is its gradient. The method should minimize f using gradient descent, and terminate when the gradient of f is small. I suggest stopping when

   $$\|\nabla f(x^k)\| < \|\nabla f(x^0)\| * 10^{-4}$$

   Test your algorithm on Rosenbrock function, and plot $\|x^k - x^*\|_2$ versus iteration numbers k for various fixed stepsize selection of $\alpha = 0.001, 0.05, 0.5$, and explain your observation.

c. Modify part b to create the new function xsol = GradDescentNesterov(f, grad, x0)
   This function should implement Nesterov's method, that is

   $$x^k = y^k - \alpha \nabla f(y^k)$$
   $$\delta^{k+1} = \tfrac{1+\sqrt{1+4(\delta^k)^2}}{2}$$
   $$y^{k+1} = x^k + \tfrac{\delta^k - 1}{\delta^{k+1}}(x^k - x^{k-1})$$

   The method is initialized with $x^0 = y^1$ and $\delta^1 = 1$, and the the first iteration has index $k = 1$. Test your algorithm on Rosenbrock function, and plot $\|x^k - x^*\|_2$ versus iteration numbers k for various fixed stepsize selection of $\alpha = 0.001, 0.05, 0.5$, and explain your observation.

Solution.

(A)
We want to show that (1,1) is a global minimizer of f. First, we want to show that (1) $\nabla f(x_1, x_2) = 0 \iff (x_1, x_2) = (1, 1)$. Next, we want to show that (2) $\nabla^2 f(1, 1)$ is positive definite.

Proof of (1):

Here, $\nabla f(x_1, x_2) = \begin{bmatrix} (100)(2)(x_2 - x_1^2)(-2x_1) + 2(1 - x_1)(-1) \\ 100(2)(x_2 - x_1^2) \end{bmatrix} = \begin{bmatrix} -400x_1(x_2 - x_1^2) - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{bmatrix}$.

$\nabla f(x_1, x_2) = 0 \iff \begin{bmatrix} -400x_1(x_2 - x_1^2) - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

We solve the above system of equations below:

$200(x_2 - x_1^2) = 0 \iff (x_2 - x_1^2) = 0 \iff x_1^2 = x_2 \iff x_1 = \sqrt{x_2}$

Plugging into the first equation gives us:
$-400\sqrt{x_2}(x_2 - \sqrt{x_2}^2) - 2(1 - \sqrt{x_2}) = 0 \iff 2\sqrt{x_2} - 2 = 0 \iff \sqrt{x_2} = 1 \iff x_2 = 1$

Therefore, we have $x_2 = 1$ and $x_1 = \sqrt{x_2} = 1$. This shows (1).

Proof of (2):

Here, $\nabla^2 f(x_1, x_2) = \begin{bmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix}$. Next, we show that $\nabla^2 f(1, 1)$ is positive definite.

$\nabla^2 f(1, 1) = \begin{bmatrix} 802 & -400 \\ -400 & 200 \end{bmatrix}$ and the eigenvalues of this matrix are $\lambda_1 = .39, \lambda_2 = 1001.6$. Since a matrix is positive definite if and only if its eigenvalues are positive, we can see that $\nabla^2 f(1, 1)$ is positive definite. This shows (2)

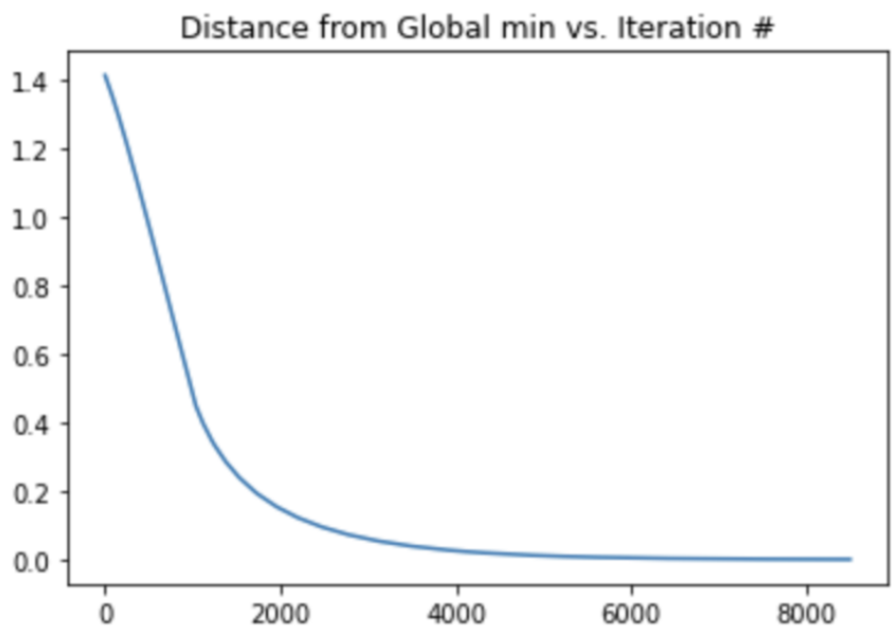
Since we showed (1) and (2), we know that the point $x^* = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is a global minimizer of f.

(B)

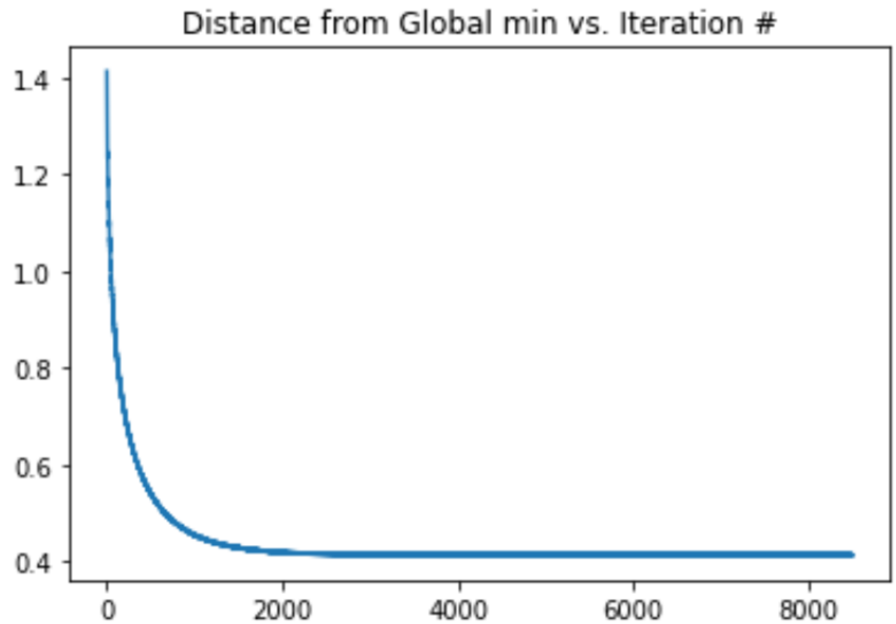
```
#Creating rosenbrock function and gradient of rosenbrock function
def rosenbrock(x,y):
  return 100*(y-x*x)**2+(1-x)**2

def grad(xvec):
  x=xvec[0][0]
  y=xvec[1][0]
  return np.array([[-400*x*(y-x**2)-2*(1-x)],[200*(y-x**2)]])

def GradDescent(f,grad,x0,alpha,max_iter=None):
  iter=0
  xk=x0
  xkarr=[np.linalg.norm(xk-np.array([[1],[1]]))]
  #print(np.linalg.norm(x0)*10**(-4))
  while np.linalg.norm(grad(xk))>=np.linalg.norm(grad(x0))*10**(-4):

    gradient=grad(xk)
    xk=xk-alpha*(gradient/np.linalg.norm(gradient))

    print(xk)
    xkarr.append(np.linalg.norm(xk-np.array([[1],[1]])))

    iter+=1
    if iter==max_iter:
      return xkarr

  return xkarr

#Running
x0=np.array([[0],[0]])
smalpha=.001
malpha=.05
laralpha=.5
smallalpha=GradDescent(rosenbrock,grad,x0,smalpha,max_iter=8500)
medalpha=GradDescent(rosenbrock,grad,x0,malpha,max_iter=8500)
```

```
largealpha=GradDescent(rosenbrock,grad,x0,laralpha,max_iter=8500)

#Plotting for alpha of .001
import matplotlib.pyplot as plt
plt.plot(np.arange(0,len(smallalpha),1),smallalpha)
plt.title('Distance from Global min vs. Iteration #')
plt.show()

#Plotting for alpha of .05
plt.plot(np.arange(0,len(medalpha),1),medalpha)
plt.title('Distance from Global min vs. Iteration #')
plt.show()

#Plotting for alpha of .5
plt.plot(np.arange(0,len(largealpha),1),largealpha)
plt.title('Distance from Global min vs. Iteration #')
plt.show()
```

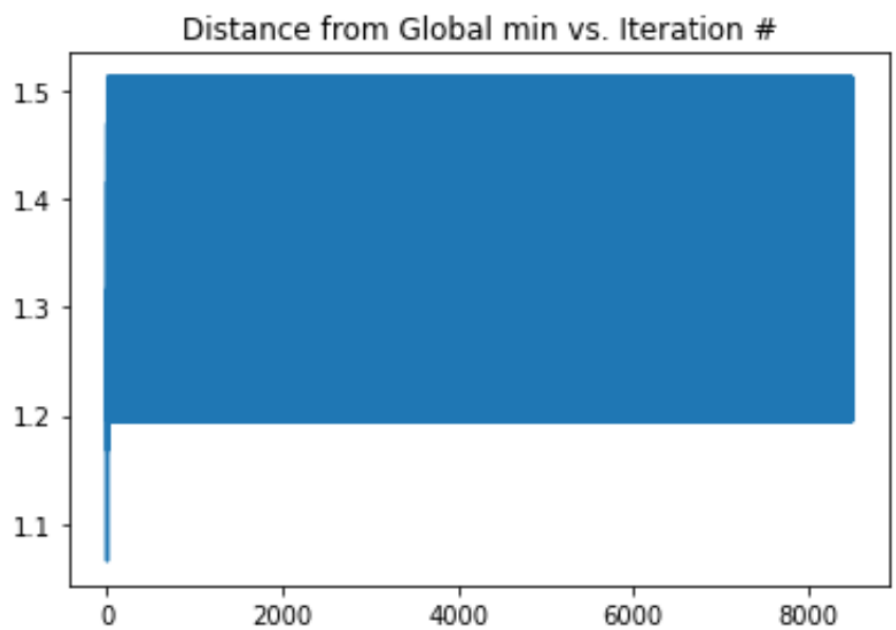
Plot for alpha=.001:



Plot for alpha=.05:



Plot for alpha=.5



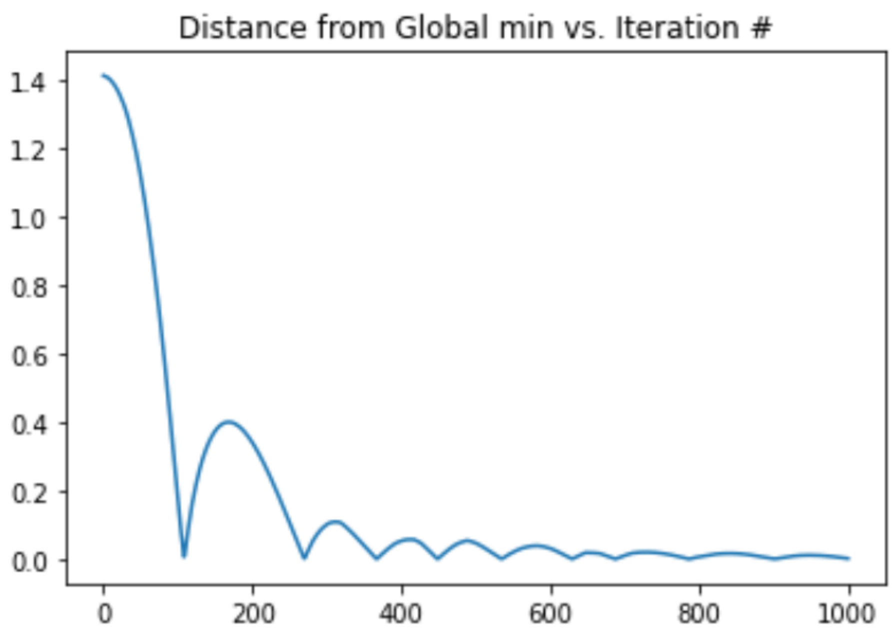
Here, we can see that the gradient descent algorithm does not converge in the case where the stepsize is .5. This is because our stepsize is too large and we actually take a large enough step past the minimum. This results in the algorithm oscillating relatively far away from the minimum.

(C)

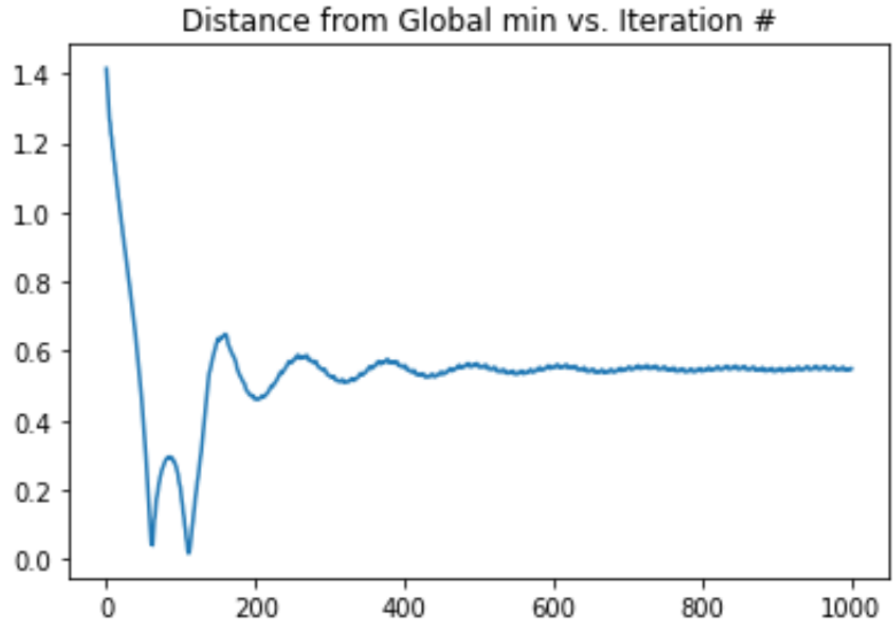
```python
def GradDescentNesterov(f,grad,x0,alpha,max_iter=8500):
  xkm1=x0
  yk=x0
  dk=1
  iter=0
  xkarr=[np.linalg.norm(xkm1-np.array([[1],[1]]))]
  while np.linalg.norm(grad(xkm1))>=np.linalg.norm(grad(x0))*10**(-4):
    #print(np.linalg.norm(grad(xkm1)))
    #Calculating gradient
    gradient=grad(yk)

    #Nudging xk in the oppposite direction of the gradient
    xk=yk-alpha*gradient/np.linalg.norm(gradient)

    #Calculating new delta
    dkp1=(1+np.sqrt(1+4*dk**2))/2

    #Calculating new y
    yk=xk+((dk-1)/dkp1)*(xk-xkm1)

    #Updating previous step's parameters
    xkm1=xk
    dk=dkp1
    xkarr.append(np.linalg.norm(xk-np.array([[1],[1]])))

    iter+=1
    if iter==max_iter:
      return xkarr

  return xkarr


x0=np.array([[0],[0]])
a1=GradDescentNesterov(rosenbrock,grad,x0,.001,1000)
a2=GradDescentNesterov(rosenbrock,grad,x0,.05,1000)
a3=GradDescentNesterov(rosenbrock,grad,x0,.5,1000)
```

```
#Plotting Nesterov for alpha=.001
plt.plot(np.arange(0,len(a1),1),a1)
plt.title('Distance from Global min vs. Iteration #')
plt.show()

#Plotting Nesterov for alpha=.05
plt.plot(np.arange(0,len(a2),1),a2)
plt.title('Distance from Global min vs. Iteration #')
plt.show()

#Plotting Nesterov for a=.5
plt.plot(np.arange(0,len(a3),1),a3)
plt.title('Distance from Global min vs. Iteration #')
plt.show()
```

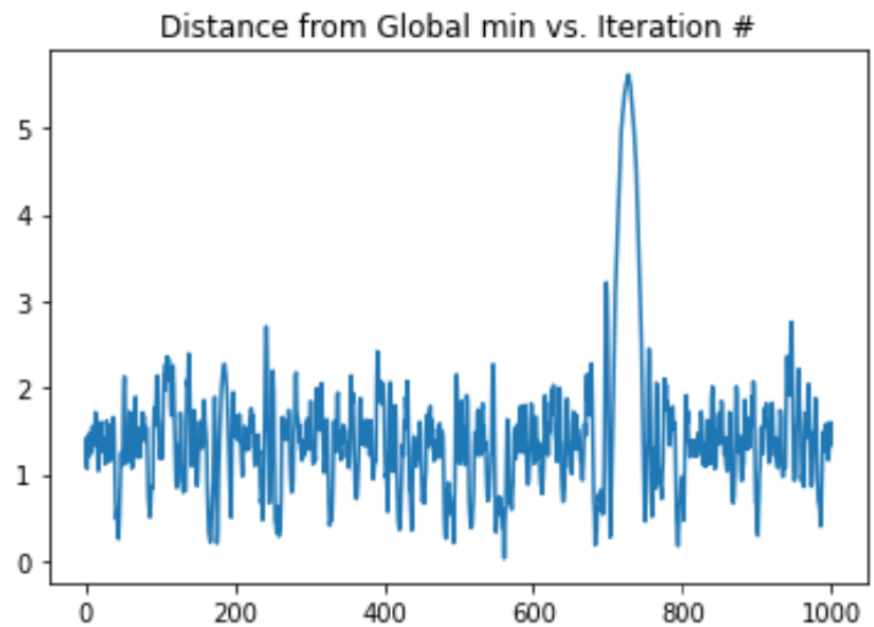
Plot for alpha=.001:



Plot for alpha=.05:



Plot for alpha=.5:



In this case, we can see that the Nesterov gradient descent algorithm converges much faster than simple gradient descent. In the alpha=.001 case, we can see that we converged to a similar value in about 1/8th the number of iterations. Additionally, we can see that Nesterov gradient descent is