Design Optimization: HW2

Seyed Yousef Soltanian

you can find the file at https://github.com/YousefSoltanian/MAE598_Design_Optimization.git (https://github.com/YousefSoltanian/MAE598_Design_Optimization.git)

Problem 1:

$f(x_1, x_2) = 2x_1^2 - 4x_1x_2 + 1.5x_2^2 + x_2$

$\rightarrow \nabla f(x_1, x_2) = [4x_1 - 4x_2 \;\; 3x_2 - 4x_1 + 1]^T$

$$\nabla f(x_1, x_2) \;\; \rightarrow \;\; \begin{array}{r} 4x_1 - 4x_2 = 0 \\ 3x_2 - 4x_1 + 1 = 0 \end{array} \;\; \rightarrow \;\; x_1* = x_2* = 1 \quad \textbf{the Stationary point}$$

$$\rightarrow Hf(x_1, x_2) = \begin{pmatrix} 4 & -4 \\ -4 & 3 \end{pmatrix}$$

The eigenvalues for the Hessian matrix above are $\lambda_1 = \frac{-\sqrt{65}+7}{2} < 0 \;\;,\;\; \lambda_2 = \frac{\sqrt{65}+7}{2} > 0$ meaning that the Hessian matrix is indefinite and the stationary point is a saddle.

Although the we know that the direction of maximum downsolpe is in the dircetion of the eigenvector of the negative eigenvalue which is $v_1 = \begin{pmatrix} \frac{\sqrt{65}-1}{8} \\ 1 \end{pmatrix}$, but we use the method asked in the problem to double check that:

$$f(x_1, x_2) - f(1,1) = v_1^T Hf v_1 = [\delta x_1 \;\; \delta x_2] \begin{pmatrix} 4 & -4 \\ -4 & 3 \end{pmatrix} [\delta x_1 \;\; \delta x_2]^T < 0 \;\; AND \;\; \delta x_1^2 + \delta x_2^2 = 1$$

$$\rightarrow \begin{array}{r} 4\delta x_1^2 - 8\delta x_1 \delta x_2 + 3\delta x_2^2 < 0 \\ \delta x_1^2 + \delta x_2^2 = 1 \end{array}$$

all the values satisfying the two conditions above are the directions of downslopes.

Problem 2:

a:

This is a convex problem, as we know, a plane is a convex set and the distance function (norm-2 function) is a convex function, resulting in a convex problem. we can write:

$x_1 = 1 - 2x_2 - 3x_3$

so we remove the constraint, the objective function to minimize become:

$f(x_1, x_2, x_3) = (x_1 + 1)^2 + x_2^2 + (x3 - 1)^2 = (2 - 2x_2 - 3x_3)^2 + x_2^2 + (x_3 - 1)^2$

$$\nabla f = 0 \;\; \rightarrow \;\; \begin{array}{r} 5x_2 + 6x_3 - 4 = 0 \\ 10x_3 + 6x_2 - 7 = 0 \end{array}$$

$x_2* = \frac{-1}{7} = -0.143 \;\; x_3* = \frac{55}{70} = 0.786 \;\; \rightarrow \;\; x_1* = -1.072$

b:

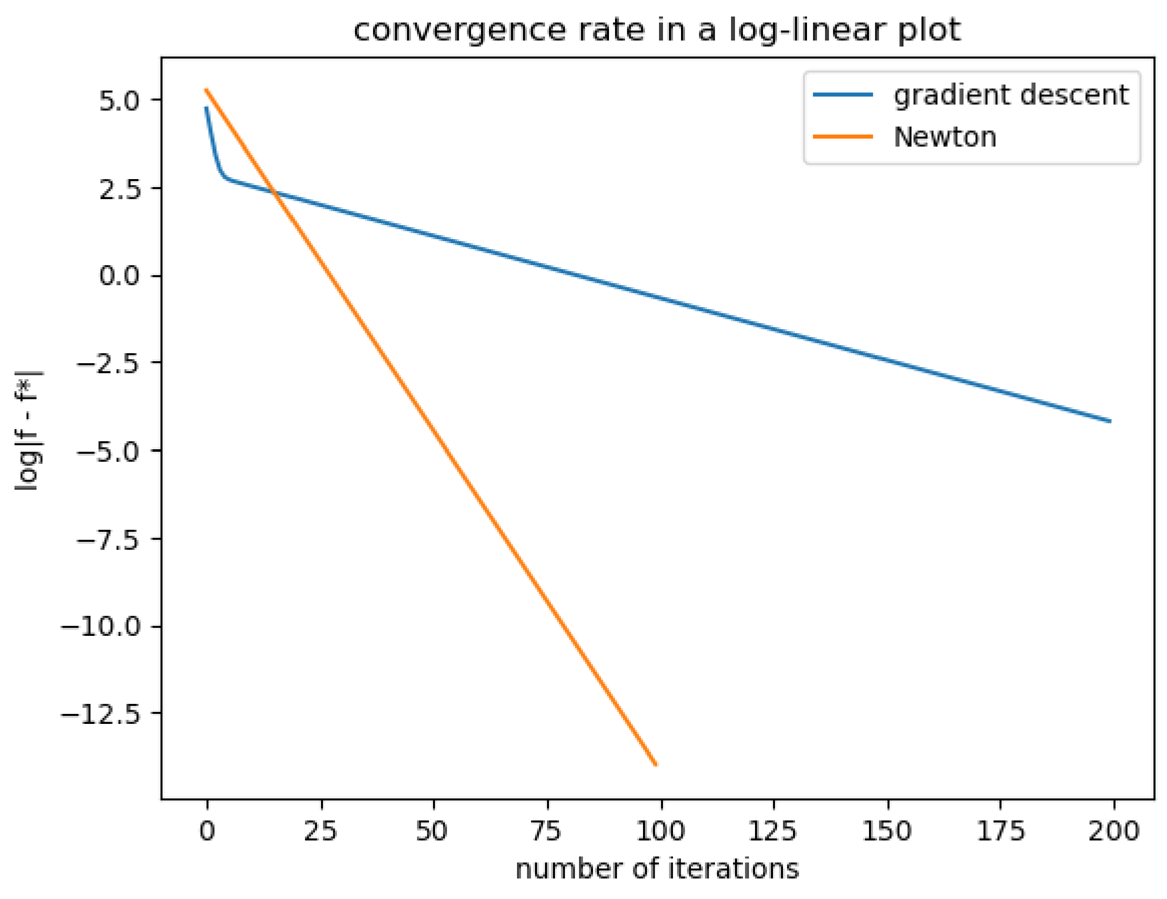
For this part, first we calculate the gradient and Hessian of the unconstraint objective function above:

$$\nabla f(x_2, x_3) = \begin{pmatrix} 10x_2 + 12x_3 - 8 \\ 20x_3 + 12x_2 - 14 \end{pmatrix} \qquad Hf(x1, x2) = \begin{pmatrix} 10 & 12 \\ 12 & 20 \end{pmatrix}$$

In [176]:
```
import numpy as np
import matplotlib.pyplot as plt

n_iter = 1000 # number of iteration

x0 = np.array([[100],[100]]) # initial guess

H = np.array([[10 , 12],[12 , 20]]) # Hessian

H_inv = np.linalg.inv(H)

x_gd = x0 # gradient descent kth x values
f_gd = 0 # gradient descent kth k values
f_gd_list = []

x_N = x0 # Newoton kth x values
f_N = 0 # Newoton descent kth k values
f_N_list = []

df_gd = np.array([[0],[0]]) # gradient for gd method
df_N = np.array([[0],[0]]) # gradient for Newoton method

a = 0.02 # learning rate



for i in range(n_iter):

    df_gd = np.array([[10,12],[12,20]])@(x_gd) - np.array([[8],[14]])

    df_N = np.array([[10,12],[12,20]])@(x_N) - np.array([[8],[14]])

    x_gd = x_gd - a*df_gd
    x_N = x_N - 10*a*np.dot(H_inv,df_N)

    f_gd = (2-2*x_gd[0][0]-3*x_gd[1][0])**2 + (x_gd[0][0])**2 + (x_gd[1][0]-1)**2
    f_gd_list.append(f_gd)

    f_N = (2-2*x_N[0][0]-3*x_N[1][0])**2 + (x_N[0][0])**2 + (x_N[1][0]-1)**2
    f_N_list.append(f_N)


print ("for gradient descent algorithm we have: x1 = "+str(1-2*x_gd[0][0]-3*x_gd[1][0])+" x2 = "+str(x_gd[0][0])+" x3

print ("for Newoton algorithm we have: x1 = "+str(1-2*x_N[0][0]-3*x_N[1][0])+" x2 = "+str(x_N[0][0])+" x3 = " + str(x_
```

```
for gradient descent algorithm we have: x1 = -1.0714285714285712 x2 = -0.1428571428571419 x3 = 0.785714285714285
for Newoton algorithm we have: x1 = -1.0714285714285718 x2 = -0.1428571428571429 x3 = 0.7857142857142859
```

In [177]:
```python
import math

f_gd_c = f_gd_list[-1] # converged values for objective function
f_N_c = f_N_list[-1]

gd_log = []
N_log = []

for i in range(200):
    c_gd = abs(f_gd_list[i]-f_gd_c)
    gd_log.append(math.log(c_gd,10))

for i in range(100):
    c_N = abs(f_N_list[i]-f_N_c)
    N_log.append(math.log(c_N,10))

plt.plot(gd_log)
plt.plot(N_log)
plt.title('convergence rate in a log-linear plot')
plt.ylabel('log|f - f*|')
plt.xlabel('number of iterations')
plt.legend(['gradient descent','Newton'])
plt.show()
```



as we can see, the newton method is faster, also the initial condition may affect the rate of convergence, in some conditions makeing the gradient descent more accurate and in some condition, close enough to the solution, making the newton more accurate. the initial condition may also affect the number of iterations we need to converge to the solution. but in general, the problem is a convex problem with one global solution, so no matter what the initial guess is, we will always converge to the answer using both methods.

Problem 3:

we assuem that 2 points of $x, y \in H$ and we have $a^T x = c \;\; a^T y = c$

we need to show tha the point $z = \lambda x + (1 - \lambda)y \in H$

$\rightarrow a^T z = \lambda a^T x + (1 - \lambda)a^T y = \lambda c + (1 - \lambda)c = c$

$\rightarrow a^T z = c$

$\rightarrow z \in H$

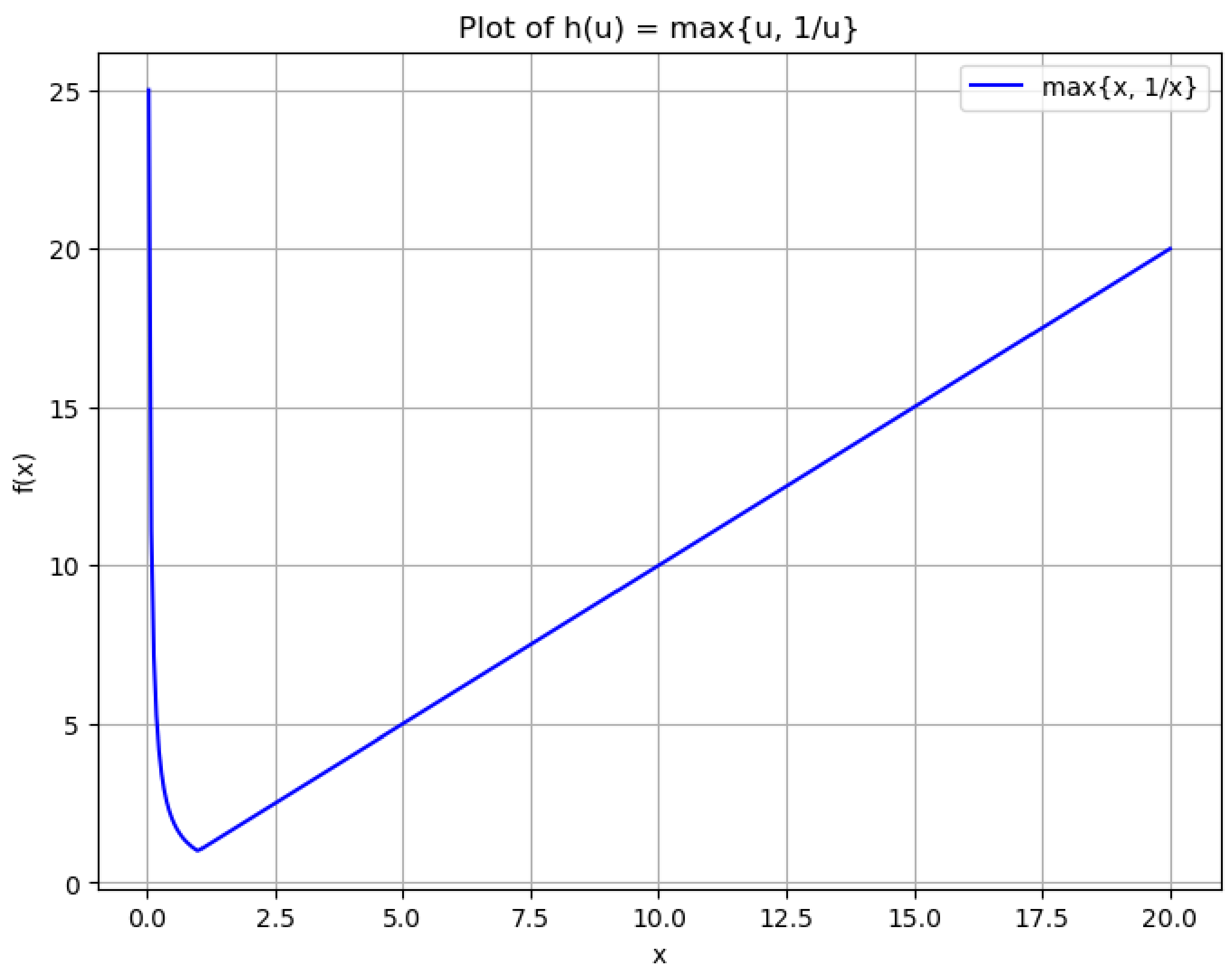
Problem 4:

a

We can write the the function h in a simpler way, it can basically be written as $h(u) = \max_k(\frac{1}{u}\ ,\ u)\ \ whereu = \frac{I}{I_t}$

This h function obviously has a minumum at $u = 1$, we can see that by plotting the function, $u = 1$ basically means $I = I_t$. as a results h is always a convex function,we also knwo that max of a set of convex function is also convex, all the constraints in this problem are also linear, so the problem is simply a convex problem. for a better demonstration and showing that the h function is convex, we have plotted the function using the code below.

In [178]:
```python
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0.04, 20, 400)  # Avoid x=0 for division by zero
y = np.maximum(x, 1/x)
# Create the plot
plt.figure(figsize=(8, 6))
plt.plot(x, y, label='max{x, 1/x}', color='blue')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Plot of h(u) = max{u, 1/u}')
plt.grid(True)
plt.legend()
plt.show()
```



b:

in this case we are going to add $\binom{n}{10}$ constraints in the form of $\Sigma p_j < p^*$ where p_js are from any 10 lamps to the optimization problem. it is basically adding some new linear constraints to the convex problem and the problem is still convex and easy to solve having a unique solution.

c:

This case cam be extremly difficult and not easy to solve, not necessarly having a unique solution. basically in this case we are limiting the degree of freedom of the problem to 10 lamps, limiting the space of possible intensities that we can make.

Problem 5:

it is obvious that $xy - c(x)$ is a line with respect to the y and as we know, a line is a convext set, the other thing that we have to show is that for any two points $(y_1, c^*(y_1)), (y_2, c^*(y_2))$ the line connecting these two points is always above the function, then we can conclude the convexity of the $c^*(y)$.

To define any possible point on the connecting line, we consider the variable $\lambda \in [0, 1]$.

$y_3 = \lambda y_1 + (1 - \lambda) y_2$

$\rightarrow \quad c^*(y_3) = \max_x (xy_3 - c(x)) = \max_x (x\lambda y_1 + x(1-\lambda)y_2 - c(x)) \quad (1)$

and for the connecting line at y_3 we have:

$value = \lambda c^*(y_1) + (1-\lambda)c^*(y_2) = \lambda \max_x (xy_1 - c(x)) + (1-\lambda) \max_x (xy_2 - c(x))$

it is obvious that always $\max_x (xy_i - c(x)) > (xy_i - c(x))$ so we have:

$\lambda \max_x (xy_1 - c(x)) + (1-\lambda) \max_x (xy_2 - c(x)) > \lambda(xy_1 - c(x)) + (1-\lambda)(xy_2 - c(x)) = (xy_3 - c(x))$

according to the above conclusion, $\lambda c^*(y_1) + (1-\lambda)c^*(y_2) > (xy_3 - c(x))$ for all values of the x, including the x that maximize that function, so it makes sense that we conclude:

$\lambda c^*(y_1) + (1-\lambda)c^*(y_2) > \max_x(xy_3 - c(x))$

as a result, $c^*(y)$ is a convex function and the proof is completed.