Assignment 8

Badreddine Itani

Problem 1

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp
```

In [2]:
```
def f(x):
    return x**2+1
def g(x):
    return (6*x/(2+2*x))**2 + 1 + x*(6*x/(2+2*x)-2)*(6*x/(2+2*x)-4)
```

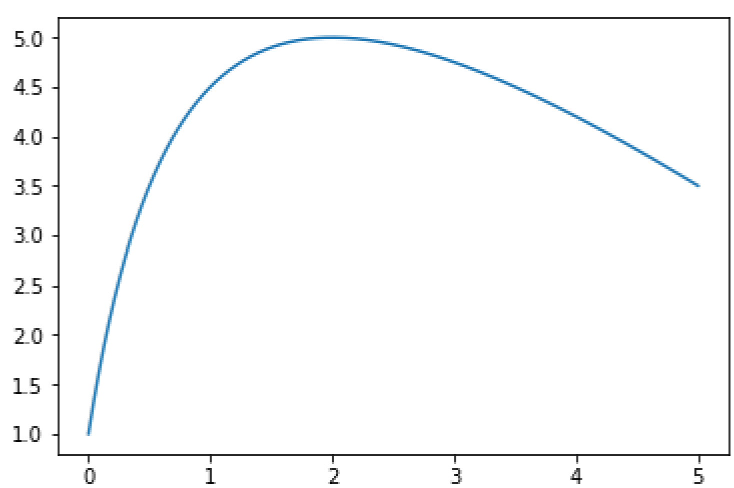
$$\min x^2 + 1$$
$$subject\ to\ (x-2)(x-4) \leq 0$$

Lagrangian:

$$L(x,\lambda) = x^2 + 1 + \lambda(x-2)(x-4)$$

$$\nabla L_x = 0$$
$$x = \frac{6\lambda}{2+2\lambda}$$
$$g(\lambda) = (\frac{6\lambda}{2+2\lambda})^2 + 1 + \lambda(\frac{6\lambda}{2+2\lambda} - 2)(\frac{6\lambda}{2+2\lambda} - 4)$$

In [3]:
```
x = np.linspace(0,5,10000)
plt.plot(x,g(x))
plt.show()
```



In [4]:
```
import cvxpy as cvx
```

In [5]:
```
x = cvx.Variable()
constraints = [
                -x<=-2,
                 x<= 4
              ]
obj = cvx.Minimize(f(x))
prob = cvx.Problem(obj, constraints)
prob.solve()
```

Out[5]:
```
5.000000000000002
```

In [6]:
```
x = sp.Symbol('x')

g_symb = [(6*x/(2+2*x))**2 + 1 + x*(6*x/(2+2*x)-2)*(6*x/(2+2*x)-4)]
g = sp.lambdify([x], g_symb, 'numpy')

derivative = sp.derive_by_array(g_symb, x)
g_div = sp.lambdify([x], derivative, 'numpy')
```

In [7]:
```
sp.solvers.solve(derivative)
```

Out[7]:
```
[{x: -4}, {x: 2}]
```

In [8]:
```
g(2)
```

Out[8]:
```
[5.0]
```

$$\lambda^* = 2$$
$$g^* = 5$$
$$f^* = 5$$

Problem 2:

$$\begin{aligned} &min\ p^t x \\ subject\ to \qquad & 0 \leq x \leq 1 \\ & 1^t x = b \end{aligned}$$

Lagragian:

$$L(x, \lambda, \nu) = p^t x - \lambda^t (x - 1) - \lambda_2^t (x) - \nu(1^t x - b)$$
$$L(x, \lambda, \nu) = (p^t - \lambda^t - \lambda_2^t - \nu 1^t)x + b\nu + \lambda^t 1$$

Dual Objective:

$$\begin{aligned} G(\lambda, \nu) &= b\nu + \lambda^t 1 \\ subject\ to \qquad p - \lambda - \lambda_2 - \nu &= 0 \\ \lambda &\geq 0 \\ G(\lambda, \nu) &= b\nu + \lambda^t 1 \\ subject\ to \qquad \lambda + \nu &\leq p \\ \lambda &\geq 0 \end{aligned}$$

In [9]:
```
from scipy.optimize import linprog
res = linprog([1], A_eq=[[1]], b_eq =[1],bounds = [(0,1)], options={"disp": True})
```

```
Optimization terminated successfully.
         Current function value: 1.000000
         Iterations: 1
```

In [10]:
```
res2 = linprog([-1,-1], A_ub=[[1,1]], b_ub = [1], options={"disp": True})
```

```
Optimization terminated successfully.
         Current function value: -1.000000
         Iterations: 1
```

We test out this equation in linprog with an example in which we have: $b = 1$ $p = 1$ Since we are mnimizing $-G(\lambda, \nu)$ we can say that $max\ G(\lambda, \nu) = 1$ thus, we have strong duality in this case since f - g = 0

Problem 3

$$
\begin{gathered}
minimize\ x^t x \\
subject\ to \qquad Ax = b \\
\lambda \geq 0
\end{gathered}
$$

Lagrangian

$$
\begin{gathered}
L(x, \lambda) = x^t x - \lambda^t (Ax - b) \\
\nabla L_x = 2x - A^t \lambda = \\
x = \frac{A^t \lambda}{2}
\end{gathered}
$$

Dual Problem

$$
\begin{gathered}
g(x, \lambda) = \frac{(A^t \lambda)^t (A^t \lambda)}{4} - \lambda^t (\frac{A A^t \lambda}{2} - b) \\
g(x, \lambda) = -\frac{\lambda^t A A^t \lambda}{4} + \lambda^t b \\
\lambda > 0
\end{gathered}
$$

Problem 4:

$$\min t$$
$$subject\ to\ \begin{bmatrix} A & -1 \end{bmatrix} \begin{bmatrix} x \\ t \end{bmatrix} \leq b$$

Lagrangian:

$$L(x, \lambda, \nu) = t + \lambda^t(Ax - 1t - b)$$
$$L(x, \lambda, \nu) = (1 - \lambda^t 1)t + \lambda^t Ax - \lambda^t b$$
$$\nabla L_x = 0$$
$$\frac{d}{dx}\lambda^t Ax = A^t\lambda = 0$$
$$\frac{d}{dt}(t - \lambda^t 1t) = 1 - 1^t\lambda = 0$$
$$1^t\lambda = 1$$
$$\lambda \geq 0$$

Dual Objective:

$$\max -b^t\lambda$$
$$subject\ to\ A^t\lambda = 0$$
$$1^t\lambda = 1$$
$$\lambda \geq 0$$

KKT Conditions

$$L(x_1, x_2, x_3, \lambda) = -b^t\lambda - x_1^t(A^t\lambda) - x_2^t(1^t\lambda - 1) - x_3^t(\lambda)$$
$$\nabla L_\lambda = -b - Ax_1 - 1x_2 - 1x_3 = 0$$
$$A^t\lambda = 0$$

$$1^t\lambda = 1$$
$$\lambda \geq 0$$
$$\lambda_i x_i = 0$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$
$$x_3 \geq 0$$

```
In [ ]:
```

Problem 5

In [61]:
```
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cvx
import numpy.linalg as la
```

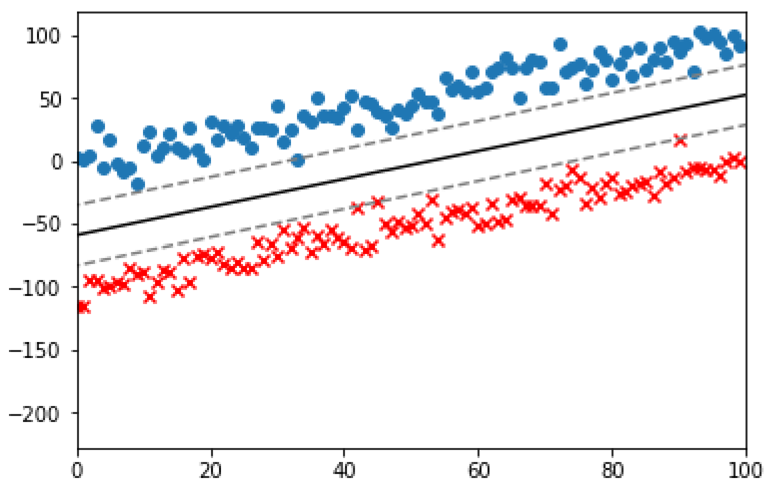
In [2]:
```
d = 2

size = 100
r = np.random.normal(size=size)
x = np.array(range(size))
r *= 10
r += x
r2 = np.random.normal(size=size)
x = np.array(range(size))
r2 *= 10
r2 -= x[::-1]
r = np.array(r)
r2 = np.array(r2)

y = np.array([r2,x])
x = np.array([r,x])
```

In [3]:
```
a = cvx.Variable(d)
b = cvx.Variable()

obj = cvx.Minimize(cvx.norm(a,2)/2)

x_constraints = [a.T * y.T[i] - b >=  1  for i in range(size)]
y_constraints = [a.T * x.T[i] - b <= -1 for i in range(size)]

constraints = x_constraints + y_constraints

prob = cvx.Problem(obj, constraints)

prob.solve()
print("Problem Status: %s"%prob.status)
```

```
Problem Status: optimal
```

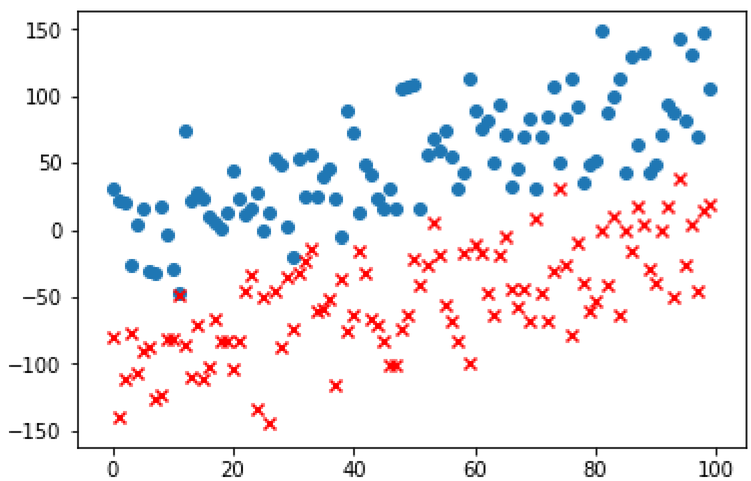
In [4]:
```python
a = a.value
b = b.value

plt.scatter(range(size), r)
plt.scatter(range(size), r2, color='red', marker='x')

d1_min = np.min([x.T[:,0],y.T[:,0]])
d1_max = np.max([x.T[:,0],y.T[:,0]])

d2_atD1min = (-a[1]*d1_min + b ) / a[0]
d2_atD1max = (-a[1]*d1_max + b ) / a[0]

plt.plot([d1_min,d1_max],[d2_atD1min,d2_atD1max],color='black')

sup_up_atD1min = (-a[1]*d1_min + b + 1 ) / a[0]
sup_up_atD1max = (-a[1]*d1_max + b + 1 ) / a[0]
sup_dn_atD1min = (-a[1]*d1_min + b - 1 ) / a[0]
sup_dn_atD1max = (-a[1]*d1_max + b - 1 ) / a[0]

plt.plot([d1_min,d1_max],[sup_up_atD1min,sup_up_atD1max],'--',color='gray')
plt.plot([d1_min,d1_max],[sup_dn_atD1min,sup_dn_atD1max],'--',color='gray')

plt.xlim([0, 100])
```

Out[4]: (0, 100)



In [56]:
```
size = 100
mult = 30
r = np.random.normal(size=size)
x = np.array(range(size))
r *= mult
r += x
r2 = np.random.normal(size=size)
x = np.array(range(size))
r2 *= mult
r2 -= x[::-1]
r = np.array(r)
r2 = np.array(r2)

y = np.array([r2,x])
x = np.array([r,x])

plt.scatter(range(size), r)
plt.scatter(range(size), r2, color='red', marker='x')
```

Out[56]: <matplotlib.collections.PathCollection at 0x24ef6d08ef0>

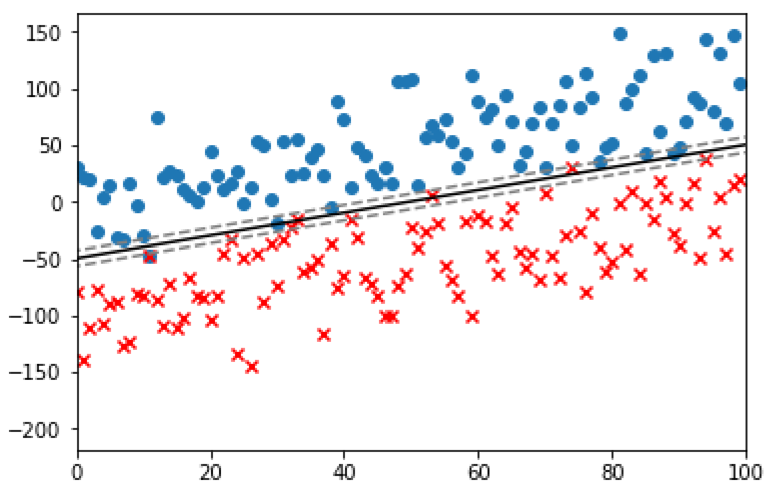


In [57]:
```
a = cvx.Variable(2)
b = cvx.Variable()
u = cvx.Variable(200)

alpha = 0.5
obj = cvx.Minimize(cvx.norm(a,2)/2 + alpha*np.ones(200)@u)

x_constraints = [a.T * y.T[i] - b + u[i]>=  1 for i in range(size)]
y_constraints = [a.T * x.T[i] - b - u[i + 100]<= -1 for i in range(size)]
u_constraints = [u[i] >= 0 for i in range(200)]

constraints = x_constraints + y_constraints + u_constraints
prob = cvx.Problem(obj, constraints)
prob.solve()
```

Out[57]: 4.342371467875232

In [58]:
```
a = a.value
b = b.value

plt.scatter(range(size), r)
plt.scatter(range(size), r2, color='red', marker='x')

d1_min = np.min([x.T[:,0],y.T[:,0]])
d1_max = np.max([x.T[:,0],y.T[:,0]])

d2_atD1min = (-a[1]*d1_min + b ) / a[0]
d2_atD1max = (-a[1]*d1_max + b ) / a[0]

plt.plot([d1_min,d1_max],[d2_atD1min,d2_atD1max],color='black')

sup_up_atD1min = (-a[1]*d1_min + b + 1 ) / a[0]
sup_up_atD1max = (-a[1]*d1_max + b + 1 ) / a[0]
sup_dn_atD1min = (-a[1]*d1_min + b - 1 ) / a[0]
sup_dn_atD1max = (-a[1]*d1_max + b - 1 ) / a[0]

plt.plot([d1_min,d1_max],[sup_up_atD1min,sup_up_atD1max],'--',color='gray')
plt.plot([d1_min,d1_max],[sup_dn_atD1min,sup_dn_atD1max],'--',color='gray')

plt.xlim([0, 100])
```

Out[58]: (0, 100)



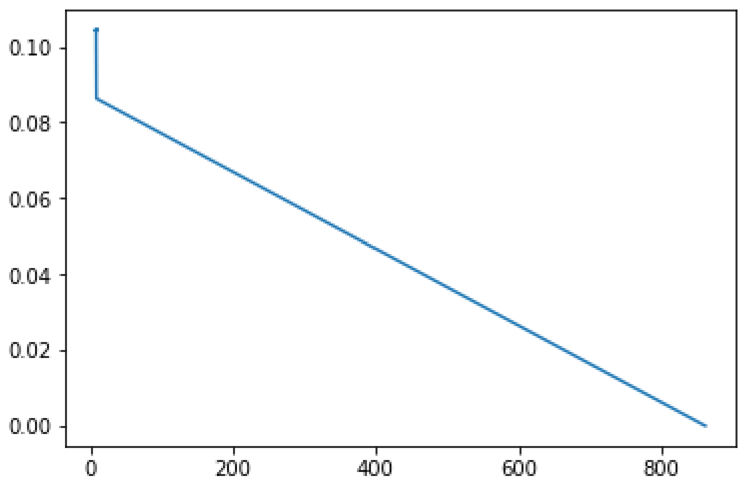
Pareto Optimal Graph

In [59]:
```
a = cvx.Variable(2)
b = cvx.Variable()
u = cvx.Variable(200)

x_constraints = [a.T * y.T[i] - b + u[i]       >=  1 for i in range(size)]
y_constraints = [a.T * x.T[i] - b - u[i + 100]<= -1 for i in range(size)]
u_constraints = [u[i] >= 0 for i in range(200)]

constraints = x_constraints + y_constraints + u_constraints
```

In [69]:
```
l = np.linspace(0,2,99)

term1 = []
term2 = []

for i, alpha in enumerate(l):
    obj = cvx.Minimize(cvx.norm(a,2)/2 + alpha*np.ones(200)@u)
    prob = cvx.Problem(obj, constraints)
    prob.solve()
    term1.append(0.5*la.norm(a.value))
    term2.append(np.ones(200)@u.value)
```

In [70]:
```
term1 = np.array(term1)
term2 = np.array(term2)
plt.plot(term2, term1)
```

Out[70]:
```
[<matplotlib.lines.Line2D at 0x24ef8843cc0>]
```



Problem 6

$$\min \ \frac{1}{2}a^t a + \alpha 1^t u$$
$$subject\ to\ D(Xa - b1) \geq 1 - u$$
$$L(a,b,u,\lambda,\sigma) = \frac{1}{2}a^t a + \alpha 1^t u - \lambda^t(D(Xa - b1) - 1 + u) - \sigma^t u$$
$$L(a,b,u,\lambda,\sigma) = \frac{1}{2}a^t a + \alpha 1^t u - \lambda^t DXa + \lambda^t Db1 + \lambda^t D1 - \lambda^t Du - \sigma^t u$$
$$L(a,b,u,\lambda,\sigma) = \frac{1}{2}a^t a - \lambda^t DXa + \lambda^t Db1 + (\alpha 1^t - \lambda^t - \sigma^t)u + \lambda^t 1$$
$$\nabla L_a = 0$$
$$a - X^t D\lambda = 0$$
$$a = X^t D\lambda$$

We replace this value of a in the Lagrangian

$$\nabla L_b = 0$$
$$1^t D\lambda = 0$$
$$\nabla L_u = 0$$
$$\alpha 1^t - \lambda^t - \sigma^t = 0$$
$$\lambda^t + \sigma^t = \alpha 1^t$$

Since sigma is always postive we can remove it and we get

$$\lambda^t \leq \alpha 1^t$$

Finally:

$$\max \ -\frac{1}{2}\lambda^t DXX^t D\lambda$$
$$1^t D\lambda = 0$$
$$0 \leq \lambda \leq \alpha 1$$

In [75]:
```
z = np.ones(200)
z[100:] = -1*z[100:]
D = np.diag(z)
```

In [76]:
```
temp = np.ones([200,2])
temp[:100] = x.T
temp[100:] = y.T
```

In [79]:
```
G = -(np.dot(D,temp)@np.dot(temp.T,D))-0.000000001*np.identity(200)
```

In [80]:
```
x = cvx.Variable(200)

temp2 = np.ones(200).T@D

I = np.identity(200)

constraints_1 = [temp2*x == 0]

constraints_2 = [I*x <= 0.5]

constraints_3 = [I*x >= 0]

constraints = constraints_1 + constraints_2 + constraints_3

obj = cvx.Maximize(cvx.quad_form(x,G)+np.ones([1,200])*x)

prob = cvx.Problem(obj,constraints)
prob.is_qp()
prob.solve(verbose = True)
```

```
-----------------------------------------------------------------
           OSQP v0.5.0  -  Operator Splitting QP Solver
              (c) Bartolomeo Stellato,  Goran Banjac
        University of Oxford  -  Stanford University 2018
-----------------------------------------------------------------
problem:  variables n = 200, constraints m = 401
          nnz(P) + nnz(A) = 20700
settings: linear system solver = qdldl,
          eps_abs = 1.0e-04, eps_rel = 1.0e-04,
          eps_prim_inf = 1.0e-04, eps_dual_inf = 1.0e-04,
          rho = 1.00e-01 (adaptive),
          sigma = 1.00e-06, alpha = 1.60, max_iter = 10000
          check_termination: on (interval 25),
          scaling: on, scaled_termination: off
          warm start: on, polish: on

objective    pri res    dua res    rho        time
   1  -6.7160e-02   2.87e-03   1.43e+00   1.00e-01   6.31e-03s
 200  -4.1882e+00   8.31e-04   5.42e-01   1.52e-02   2.11e-02s
 400  -4.0245e+00   3.97e-03   1.69e-02   1.48e-03   2.13e-02s
 600  -4.4362e+00   3.19e-03   1.63e-02   1.48e-03   3.60e-02s
 800  -4.1654e+00   9.80e-04   1.35e-02   1.48e-03   4.93e-02s
1000  -4.2066e+00   1.50e-03   1.57e-02   1.48e-03   6.26e-02s
1200  -4.3695e+00   2.22e-03   9.46e-03   1.48e-03   7.57e-02s
1400  -4.1295e+00   2.76e-03   4.21e-03   1.48e-03   8.99e-02s
1600  -4.3333e+00   1.65e-03   1.21e-02   1.48e-03   1.04e-01s
1800  -4.2330e+00   9.90e-04   9.46e-03   1.48e-03   1.18e-01s
2000  -4.2013e+00   9.82e-04   5.48e-03   1.48e-03   1.31e-01s
2200  -4.3053e+00   1.13e-03   3.22e-03   1.48e-03   1.46e-01s
2400  -4.2071e+00   1.19e-03   1.66e-03   1.48e-03   1.59e-01s
2600  -4.2829e+00   1.01e-03   4.40e-03   1.48e-03   1.72e-01s
2800  -4.2295e+00   5.13e-04   3.10e-03   1.48e-03   1.85e-01s
2850  -4.2413e+00   1.40e-04   1.72e-03   1.48e-03   1.88e-01s

status:               solved
solution polish:      unsuccessful
number of iterations: 2850
optimal objective:    -4.2413
run time:             1.91e-01s
optimal rho estimate: 5.81e-04
```

Out[80]: 4.241262152790743

The diffrence between the dual function and the original one is about 0.1. The solution of the dual problem is supposed to act as a lower bound for the original problem. This is why we don't have equality.

$$g^* \leq f^*$$