HW5-Q1

November 14, 2019

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.datasets import make_spd_matrix,make_sparse_spd_matrix
from scipy.stats import unitary_group
```

[2]:
```
class LossFunc:

    # initialization
    def __init__(self,Q,b,lam,eg):

        self.Q = Q
        self.b = b
        self.lam = lam
        self.eg = eg

    def g(self,x):
        return 0.5*np.dot(x.T.dot(self.Q),x)-np.dot(self.b,x)
    def g2(self,x):
        # print((0.5*(self.eg*(x**2))-self.b*x).shape)
        return 0.5*np.dot(self.eg,(x**2))-np.dot(self.b,x)

    def h(self,x):
        return self.lam*np.linalg.norm(x,1)

    def gGrad(self,x):
        return self.Q.dot(x) - self.b
    def g2Grad(self,x):
        # print((self.eg*x-self.b).shape)
        return self.eg*x-self.b

    def obj(self,x):
        return self.g2(x)+self.h(x)

    def Gt(self,x,t,idx):
        x_0 = x.copy()
        x[idx] = self.prox(x[idx]-t*self.g2Grad(x)[idx],t)
```

```
        return (x_0 - x)/t

    def soft_threshold(self,b,a):
        if (np.abs(a)<=b):
            return 0
        if(a<-b):
            return a+b
        if(a>b):
            return a-b

    def prox(self,x,t):
        return self.soft_threshold(x,self.lam*t)
```

1 Experiment Setup

- We would like to compare exact step coordinate proximal GD with fixed step coordinate proximal GD.
- For comparsion, we will run them on the specific convex problems for several times
- After prelinmary experiments, we found two factors which will affect how much exact step helps: dimension of variables X and the distribution of the eigen value of Q
- So we set 2 hyperparameters for our experiment: X's dimension and Q's condition number
- For other settings, we choose $\lambda = 0.1$ and fixed step size $t_{fixed} = 1/max(Q[i, i])$.

[40]:
```
def exact_vs_fixed(n,cond,trial=10):
    # n is the dimesion of x,Q,b
    # cond is the Q_eigen_value_max / Q_eigen_value_min
    # trial is the number of repeated tests


    plt.figure(figsize=(24,2*trial))
    epi = 1e-6
    flops_fx = []
    flops_ex = []

    for z in range(trial):
        x_0 = np.zeros(n)
        min_eg = np.abs(np.random.randn(1)[0])
        max_eg = cond*min_eg
        eg = np.append(np.array([min_eg,max_eg]),np.random.
 ↪uniform(low=min_eg,high=max_eg,size=n-2))
        # eg = np.random.uniform(low=0.001,high=1,size=n)
        lam = 0.1
        random.shuffle(eg)
        Q = np.diag(eg)
        W = unitary_group.rvs(n)
```

```python
    Q = np.dot(W.T.dot(Q),W)
    b = np.random.randn(n)

    # Fixed Step Coordinate Proximal GD
    # cot_fx = []
    obj_fx = []

    L = LossFunc(Q.copy(),b.copy(),lam,eg.copy())
    x = x_0.copy()
    obj_0 = L.obj(x)
    # obj_record = []
    # cot = 0
    obj_fx.append(obj_0)
    t = 1/max_eg

    while(True):
        for i in range(n):
            x[i] = L.prox(x[i]-t*L.g2Grad(x)[i],t)
            # cot = cot+1
            obj_1 = L.obj(x)
            obj_fx.append(obj_1)
        if (obj_0 - obj_1) < epi:
            break
        obj_0 = obj_1

    # cot_fx.append(cot)
    # obj_fx.append(obj_record)

    # Exact Step Coordinate Proximal GD
    # cot_ex = []
    obj_ex = []

    L = LossFunc(Q,b,lam,eg)
    x = x_0.copy()
    obj_0 = L.obj(x)
    # obj_record = []
    # cot = 0
    obj_ex.append(obj_0)

    while(True):
        for i in range(n):
            t = 1/eg[i]
            x[i] = L.prox(x[i]-t*L.g2Grad(x)[i],t)
            # cot = cot+1
            obj_1 = L.obj(x)
            obj_ex.append(obj_1)
        if (obj_0 - obj_1) < epi:
```

```
                break
            obj_0 = obj_1

        # cot_ex.append(cot)
        # obj_ex.append(obj_record)

        plt.subplot(trial/4,4,z+1)
        plt.xlabel('Flops')
        plt.ylabel('Criterion')
        fx = np.array(obj_fx)
        plt.plot(fx,label='Fixed')
        ex = np.array(obj_ex)
        plt.plot(ex,label='Exact')
        plt.legend()

        flops_fx.append(len(obj_fx))
        flops_ex.append(len(obj_ex))
    return np.array(flops_fx).mean(),np.array(flops_ex).mean()
```

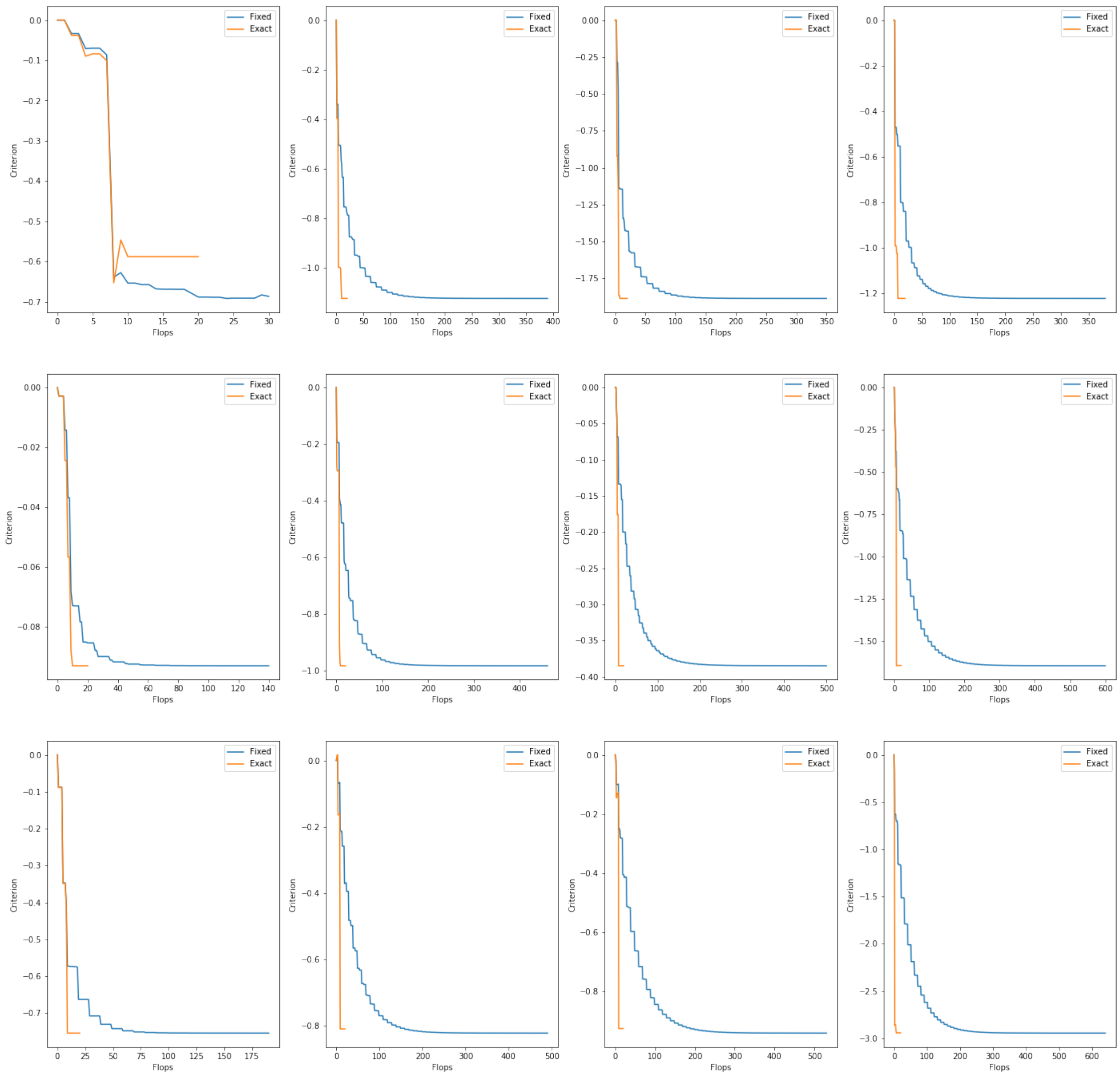
2 Result

2.1 How dimension affects

2.1.1 n = 10

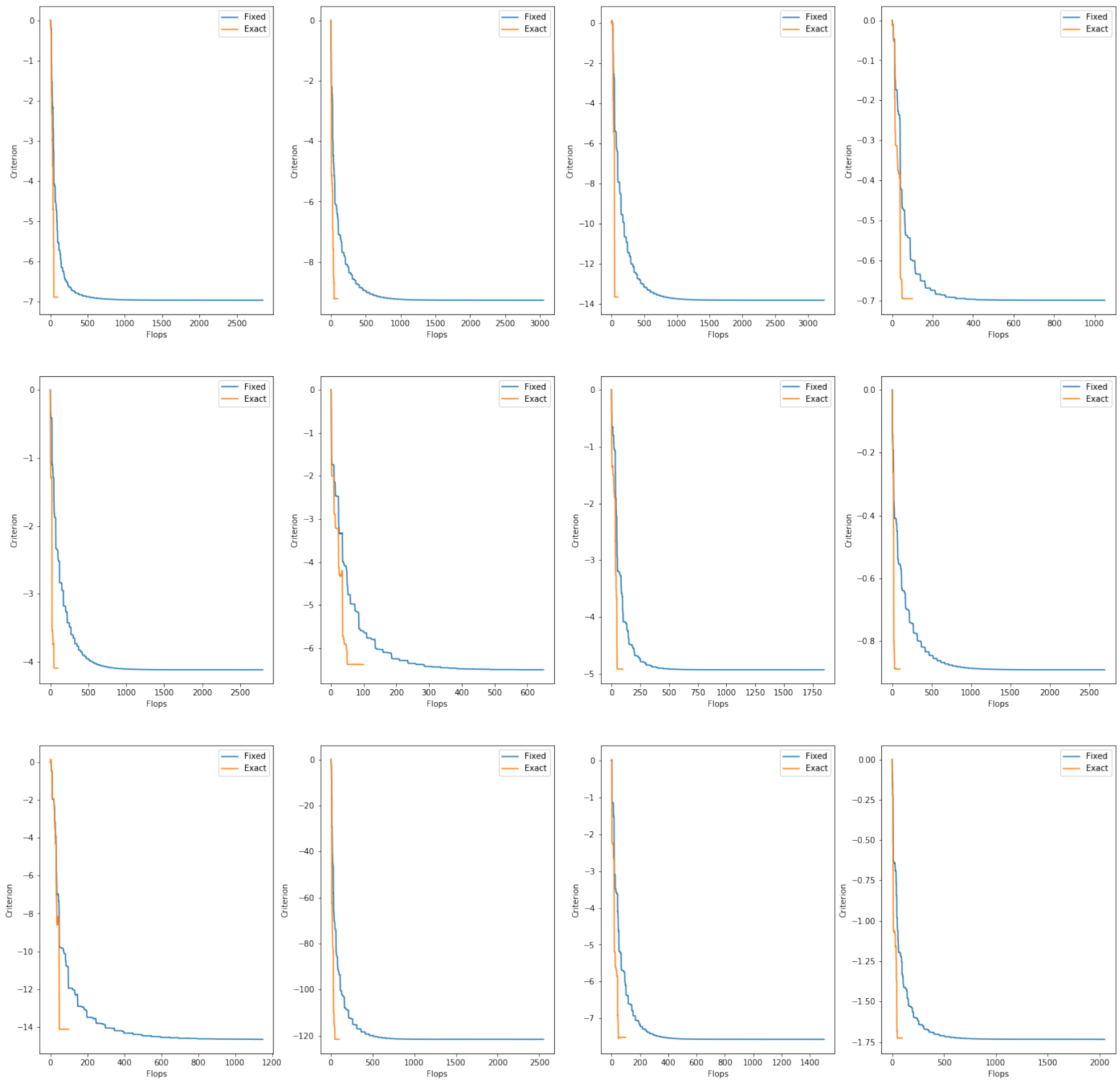
[60]:
```
flop_until_convergence_fx = []
flop_until_convergence_ex = []
```

[61]:
```
nf,ne = exact_vs_fixed(10,10,trial=12)
flop_until_convergence_fx.append(nf)
flop_until_convergence_ex.append(ne)
```



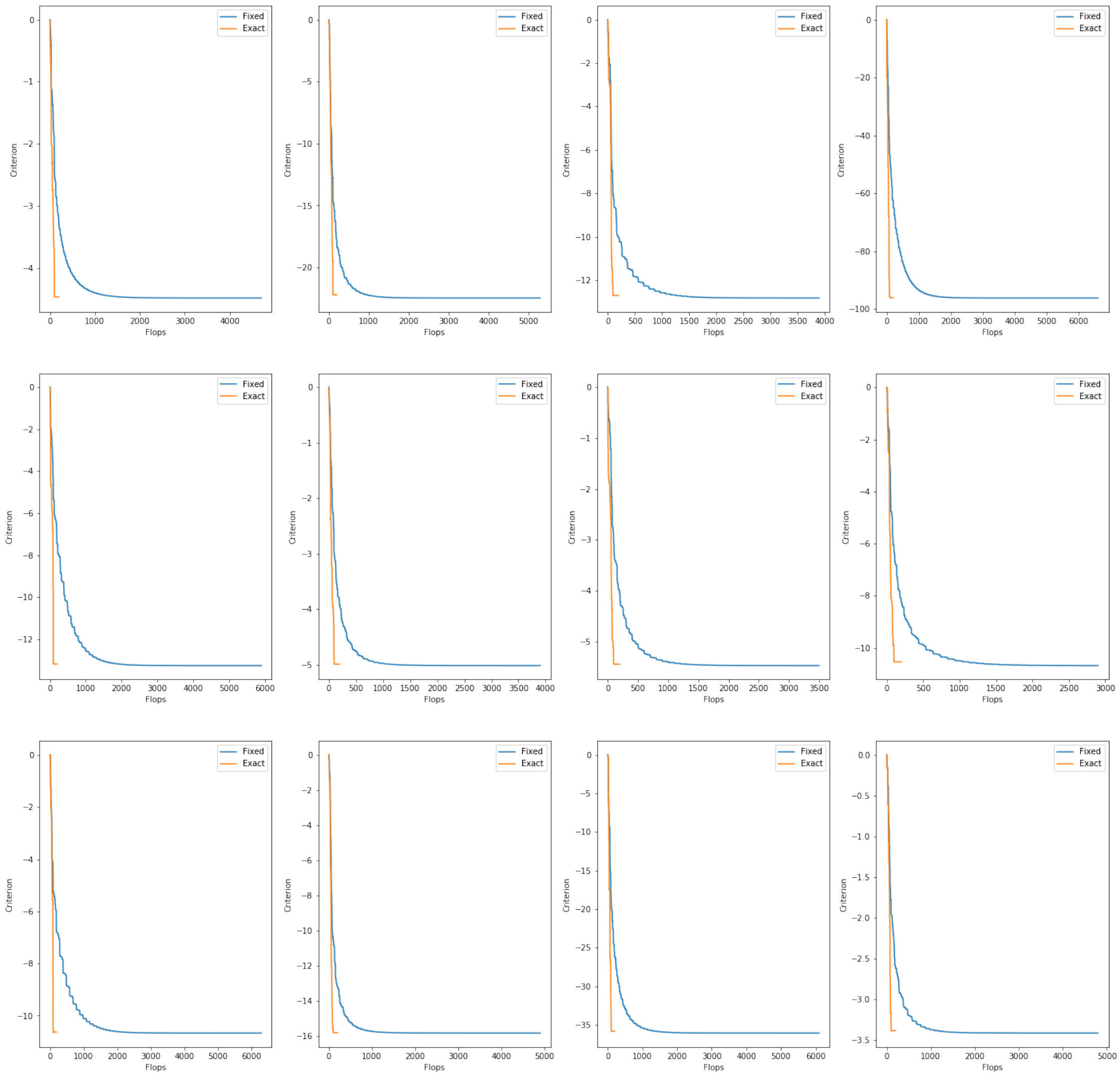
2.1.2 n = 50

[62]:
```
nf,ne = exact_vs_fixed(50,10,trial=12)
flop_until_convergence_fx.append(nf)
flop_until_convergence_ex.append(ne)
```



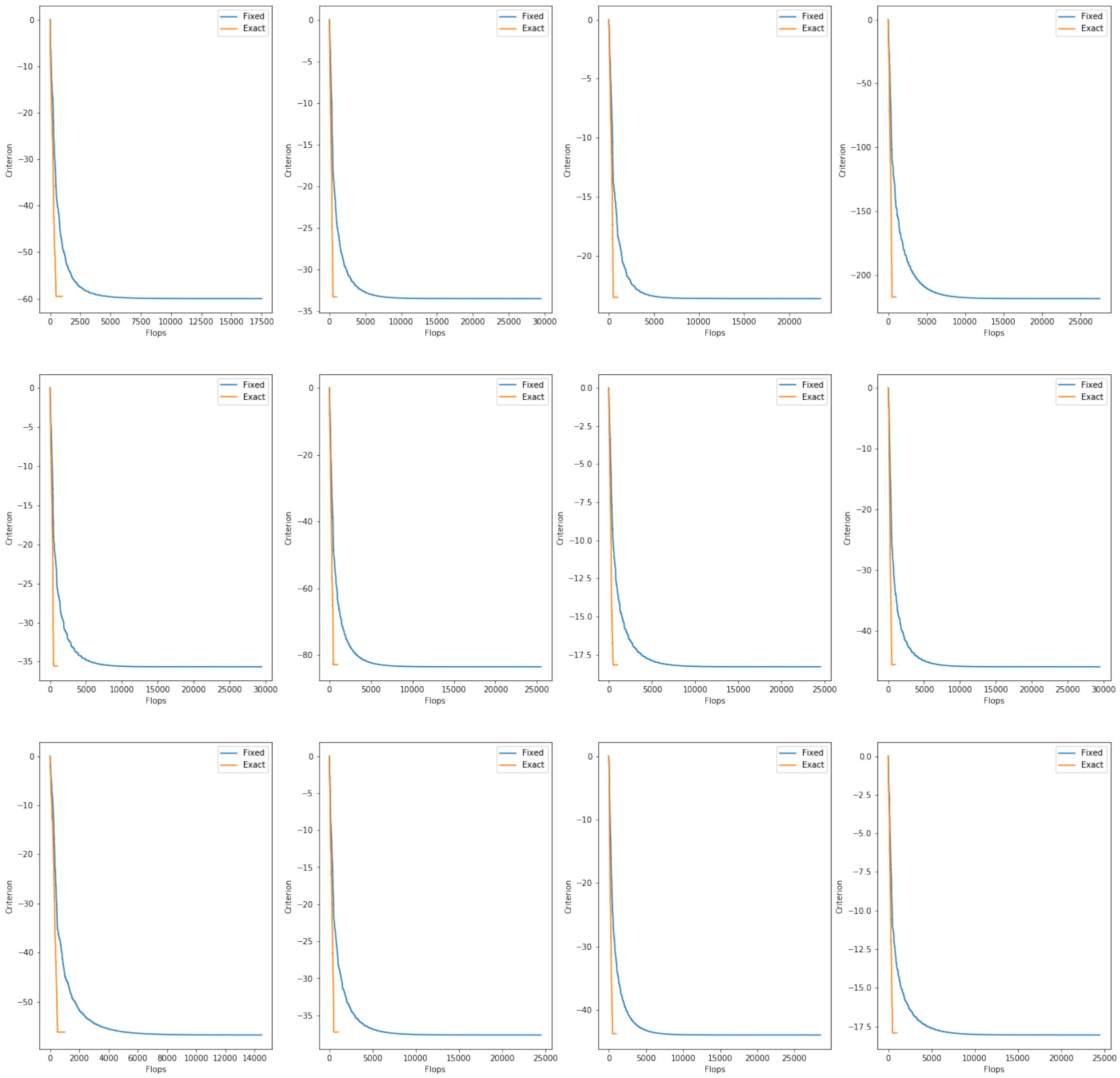
2.1.3 n = 100

[63]:
```
nf,ne = exact_vs_fixed(100,10,trial=12)
flop_until_convergence_fx.append(nf)
flop_until_convergence_ex.append(ne)
```



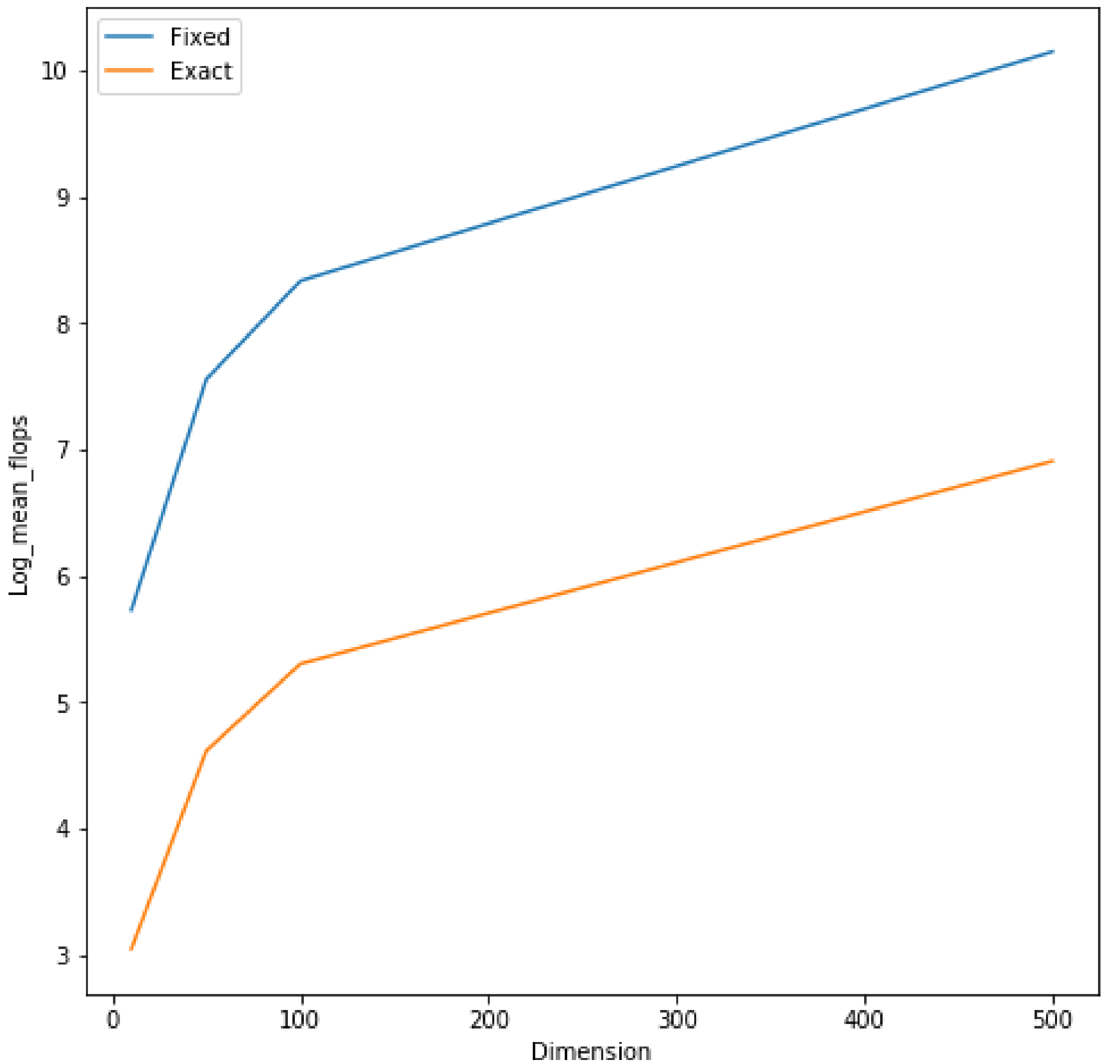
2.1.4 n = 500

[64]:
```
nf,ne = exact_vs_fixed(500,10,trial=12)
flop_until_convergence_fx.append(nf)
flop_until_convergence_ex.append(ne)
```



2.2 Dimension VS Flops Until Convergence

[53]:
```
plt.figure(figsize=(8,8))
plt.plot([10,50,100,500],np.log(np.
 ↪array(flop_until_convergence_fx)),label='Fixed')
plt.plot([10,50,100,500],np.log(np.
 ↪array(flop_until_convergence_ex)),label='Exact')
plt.xlabel('Dimension')
plt.ylabel('Log_mean_flops')
plt.legend()
plt.show()
```

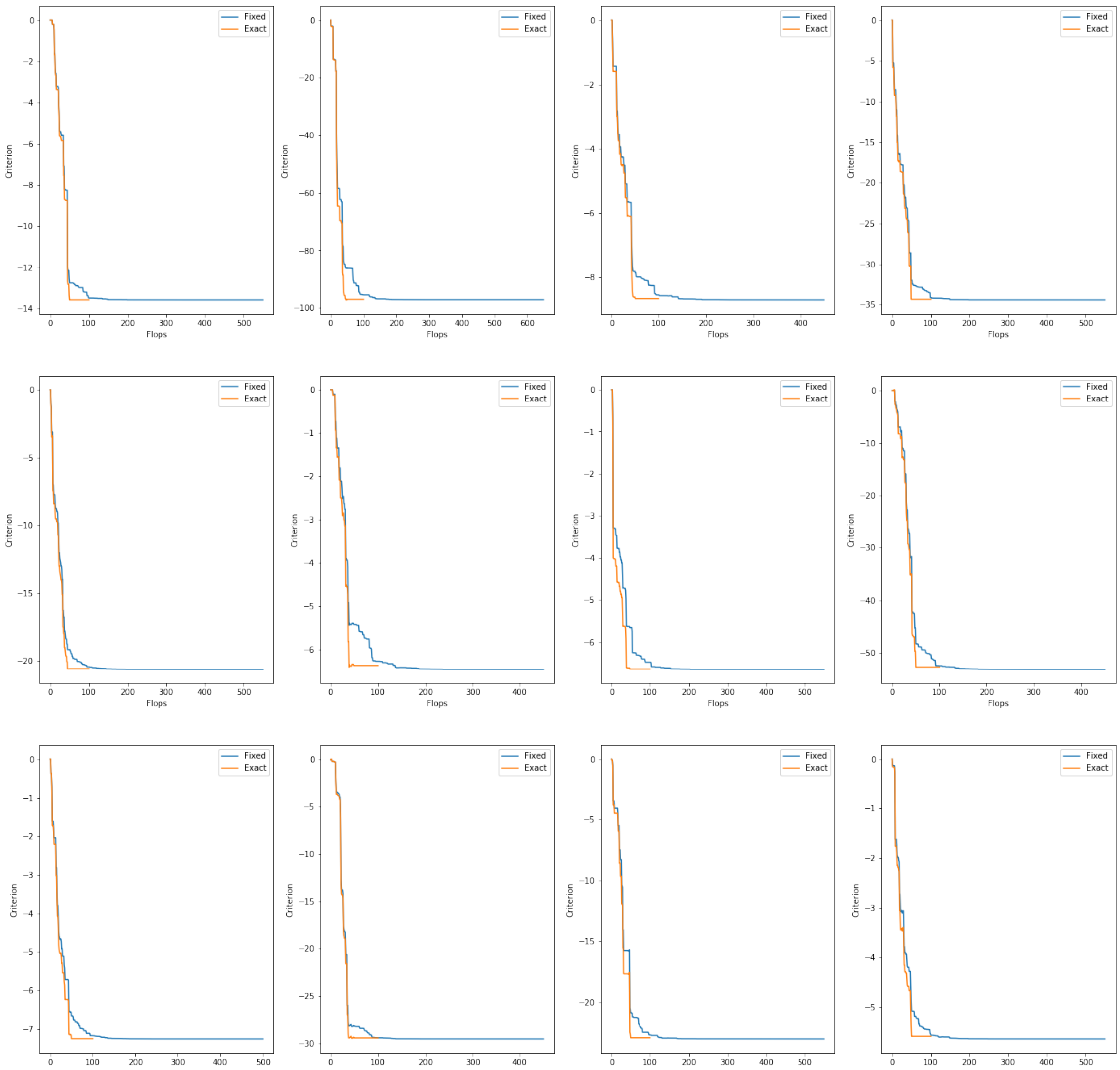


2.3 How condition number of Q affects

2.3.1 cond = 2

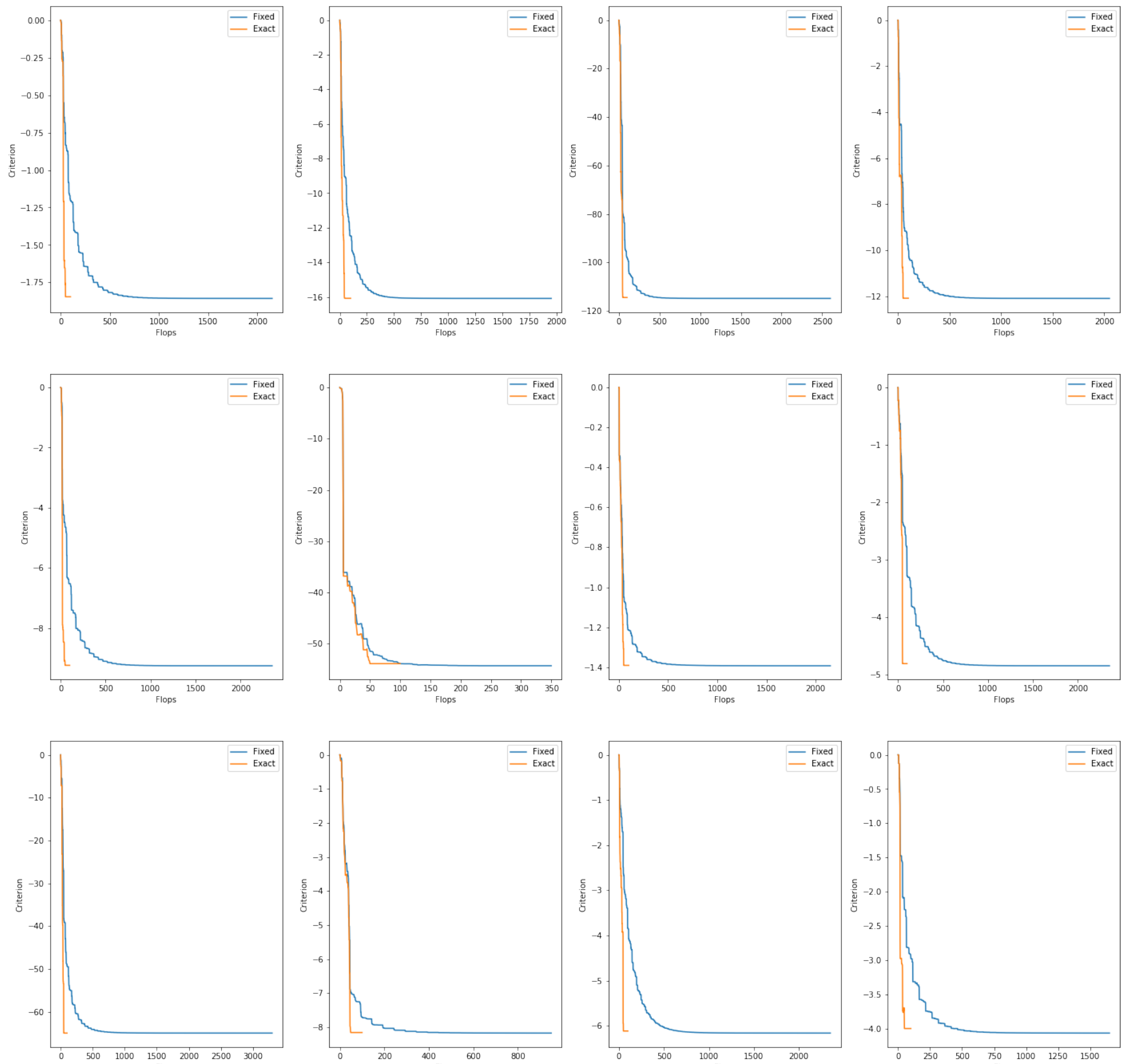
[66]:
```
flop_until_convergence_fx = []
flop_until_convergence_ex = []
```

[67]:
```
nf,ne = exact_vs_fixed(50,2,trial=12)
flop_until_convergence_fx.append(nf)
flop_until_convergence_ex.append(ne)
```



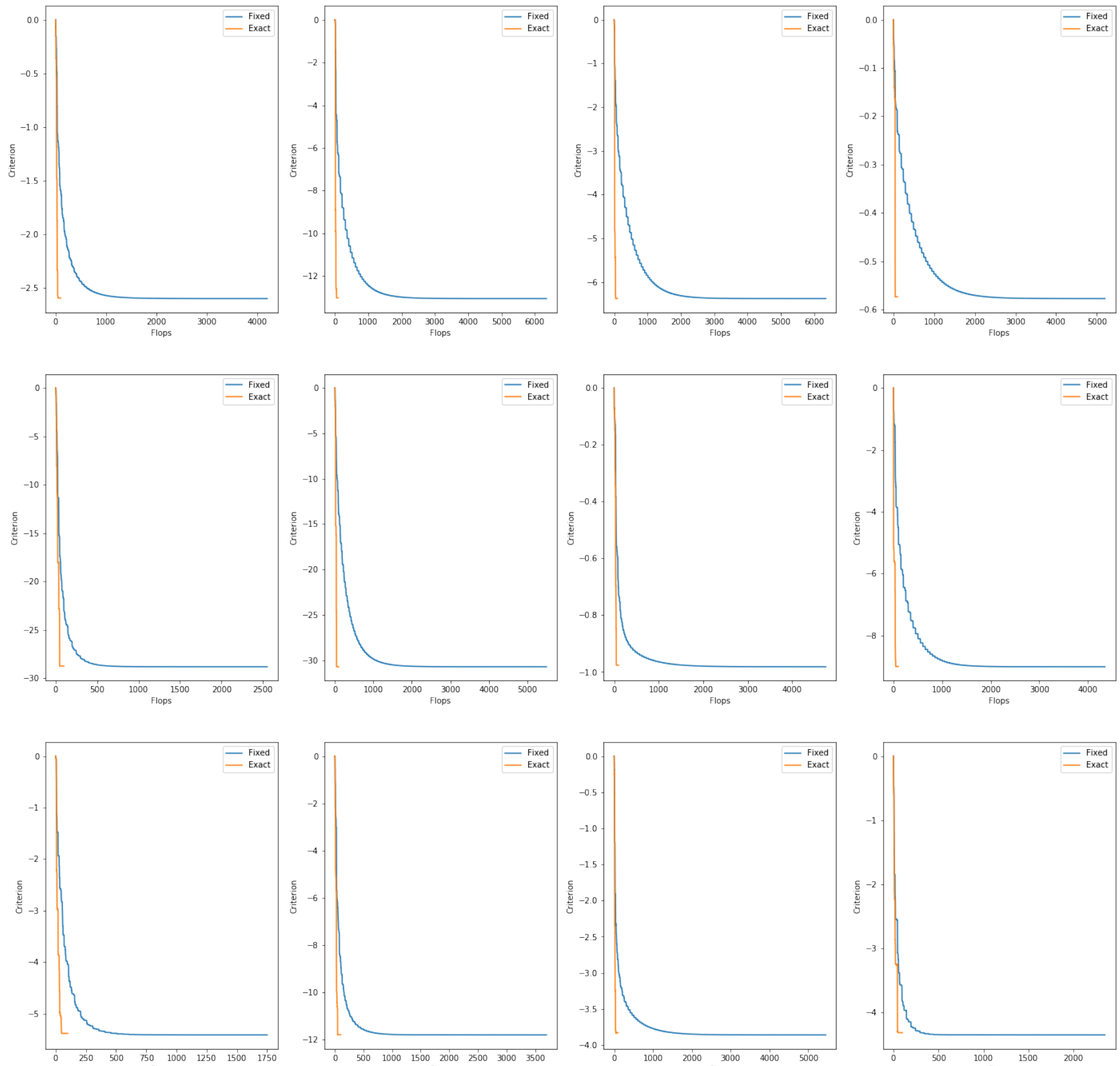
2.3.2 cond = 10

[68]:
```
nf,ne = exact_vs_fixed(50,10,trial=12)
flop_until_convergence_fx.append(nf)
flop_until_convergence_ex.append(ne)
```



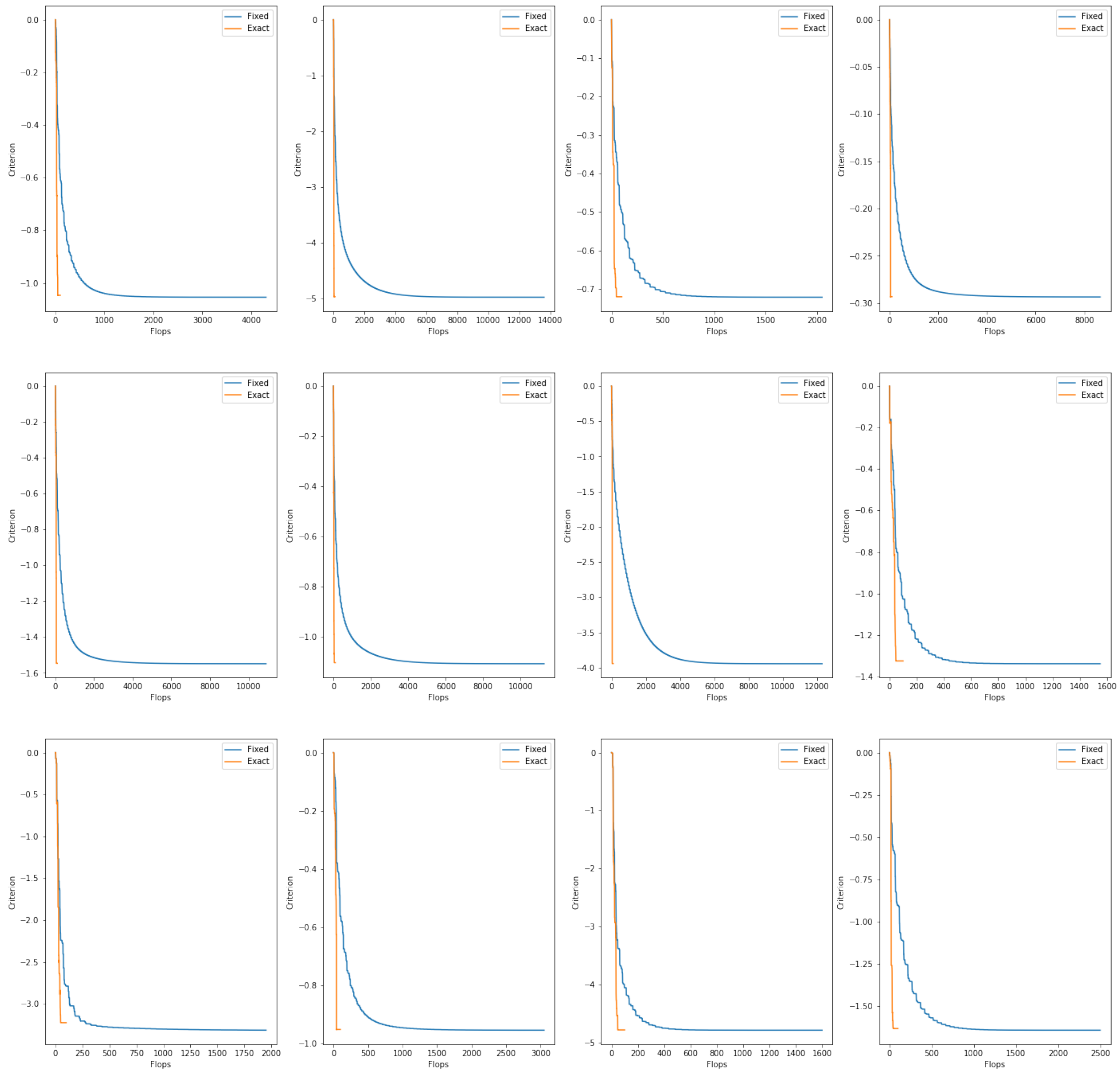
2.3.3 cond = 20

[69]:
```
nf,ne = exact_vs_fixed(50,20,trial=12)
flop_until_convergence_fx.append(nf)
flop_until_convergence_ex.append(ne)
```



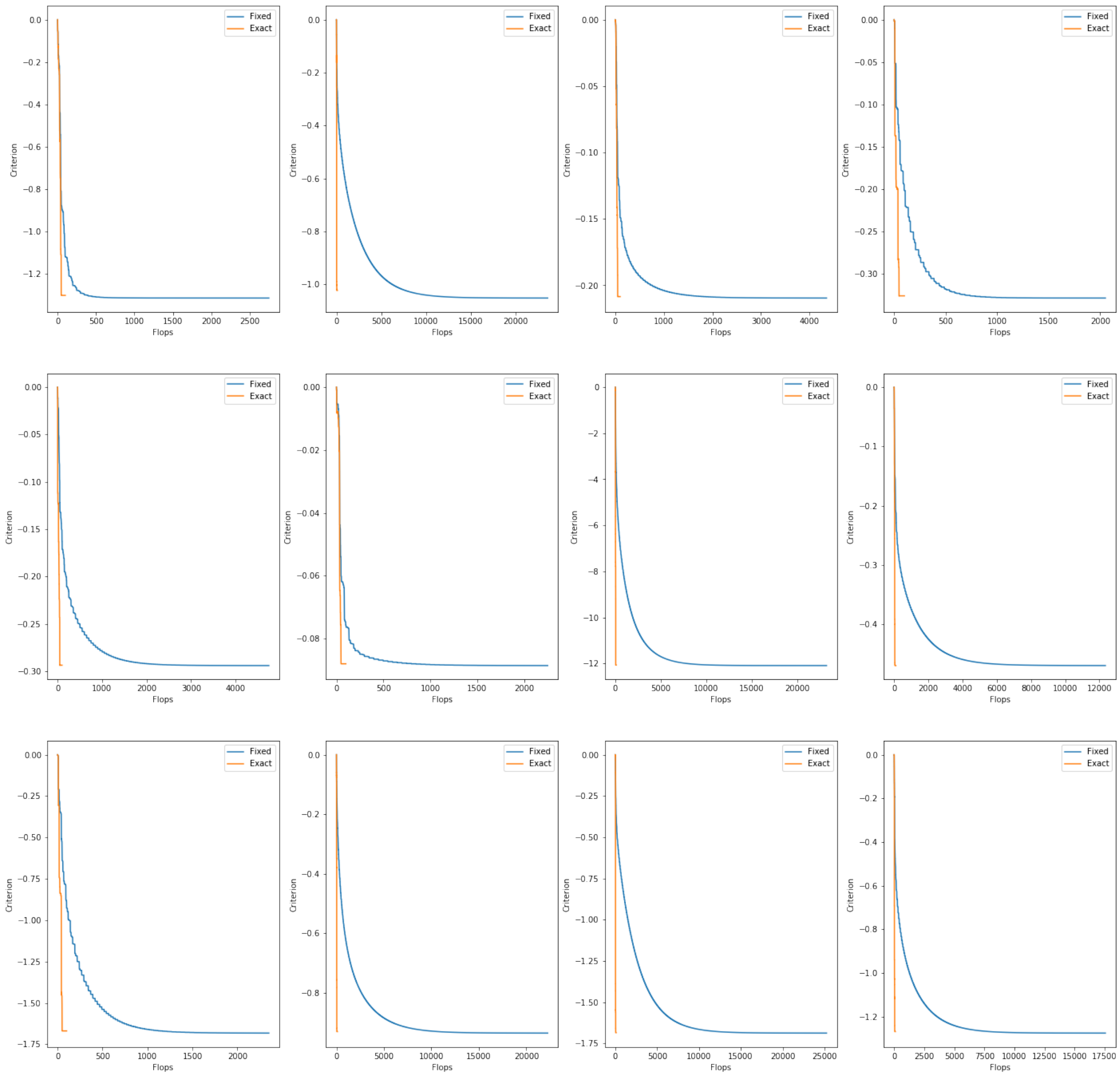
2.3.4 cond = 50

[70]:
```
nf,ne = exact_vs_fixed(50,50,trial=12)
flop_until_convergence_fx.append(nf)
flop_until_convergence_ex.append(ne)
```



2.3.5 cond = 100

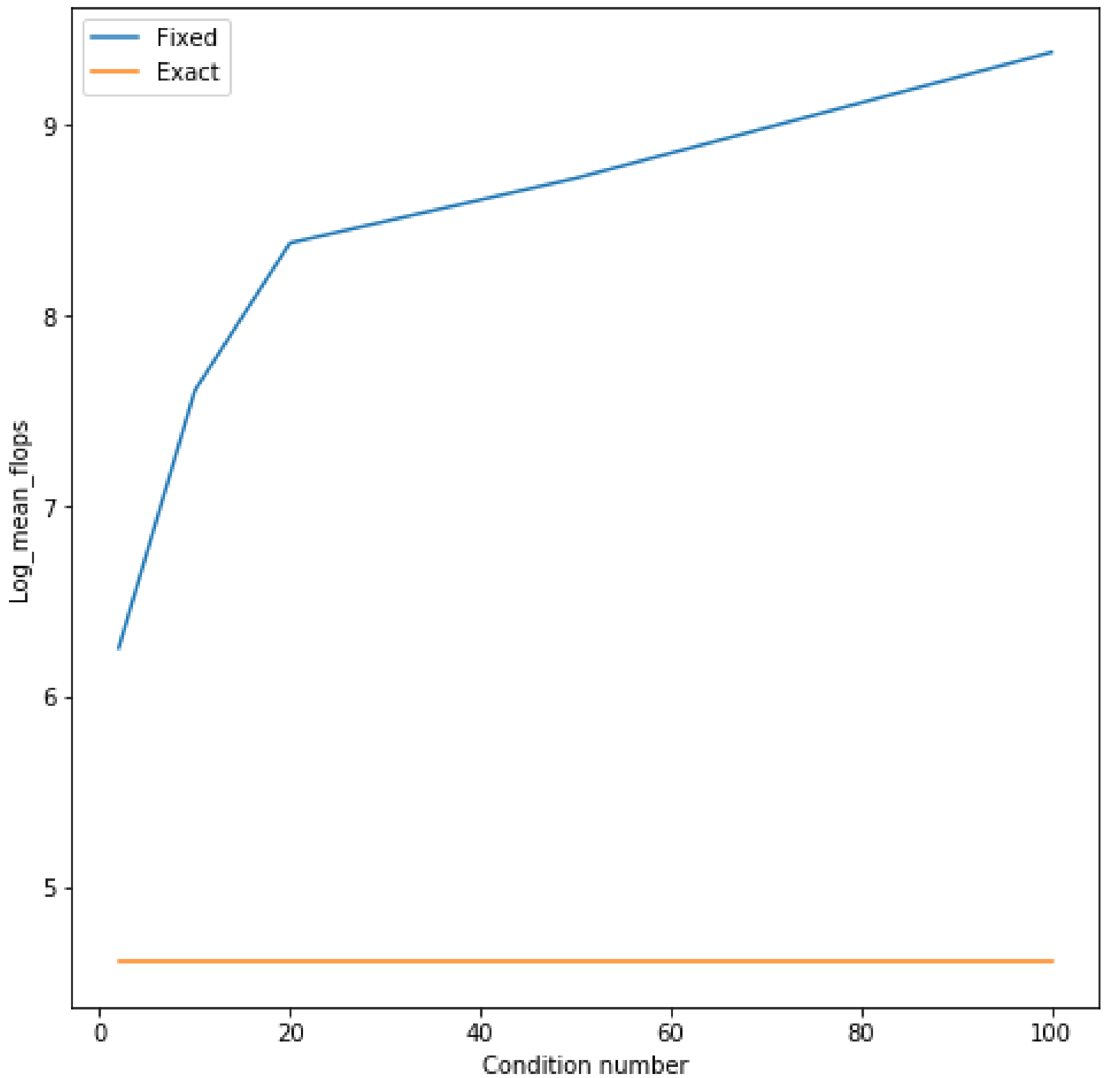
[71]:
```
nf,ne = exact_vs_fixed(50,100,trial=12)
flop_until_convergence_fx.append(nf)
flop_until_convergence_ex.append(ne)
```



2.4 Condition Number VS Flops Until Convergence

[72]:

```
plt.figure(figsize=(8,8))
plt.plot([2,10,20,50,100],np.log(np.
 ↪array(flop_until_convergence_fx)),label='Fixed')
plt.plot([2,10,20,50,100],np.log(np.
 ↪array(flop_until_convergence_ex)),label='Exact')
plt.xlabel('Condition number')
plt.ylabel('Log_mean_flops')
plt.legend()
plt.show()
```



3 Conclusion

- Exact step size coordinate proximal gradient descent (CPGD) always converge faster than fixed step size coordinate proxiaml gradient descent.
- When variable dimension grows, both of them take more flops to converge. Meanwhile, the increase rate of the number of flops they take to converge is almost log-linear as the dimension increase.
- When distribution of eigenvalues of Q changes, as the condition number of Q grows, Fixed step size CPGD take more flops to converge but the number of flops that exact step size CPGD takes to converge almost remains.