In [1]:

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator
from mpl_toolkits.mplot3d import Axes3D
import gzip
from sklearn.preprocessing import OneHotEncoder
from scipy.special import expit
import celluloid
from celluloid import Camera
from matplotlib import animation
from IPython.display import HTML
from matplotlib.lines import Line2D

np.random.seed(2022)
```

In [2]:

```
def gradient_descent(xinit,steps,gradient):
    """Run gradient descent.
    Return an array with the rows as the iterates.
    """
    xs = [xinit]
    x = xinit
    for step in steps:
        x = x - step*gradient(x)
        xs.append(x)
    return np.array(xs)

def nagd(winit,gradient,eta=0.1,nsteps=100):
    """Run Nesterov's accelrated graident descent.
    Return an array with the rows as the iterates.
    """
    ws = [winit]
    u = v = w = winit
    for i in range(nsteps):
        etai = (i+1)*eta/2
        alphai = 2/(i+3)
        w = v - eta*gradient(v)
        u = u - etai*gradient(v)
        v = alphai*u + (1-alphai)*w
        ws.append(w)
    return np.array(ws)
```

In [3]:

```
def animated_lplot(Ys,labels=['1','2','3','4','5','6'],ylabel='Function value'):
    """Animated line plot of the Y values.
    Ys is a list where each element is an array of numbers to plot.
    """
    colors = ['blue','red','green','black','cyan','purple','pink']
    fig, ax = plt.subplots(figsize=(6,6))
    camera = Camera(fig)
    T = len(Ys[0])
    plt.yscale('log')
    for t in range(T):
        for j in range(len(Ys)):
            plt.plot(range(t),Ys[j][:t],color=colors[j],marker='o')
        camera.snap()
    handles = []
    for i in range(len(Ys)):
        handles.append(Line2D([0], [0], color=colors[i], label=labels[i]))
    plt.legend(handles = handles, loc = 'upper right')
    plt.xlabel('Step')
    plt.ylabel(ylabel)
    animation = camera.animate(interval=100,blit=False)
    plt.close()
    animation.save('animation.mp4');
    return animation
```

In [9]:

```
def sigmoid(x):
    return 1/(1 + np.exp(-x))

def generate_logistic(n,d):
    """Generate a dataset under the logistic model.
    """
    X = np.random.randn(n,d)
    wstar = np.random.randn(d,1)
    wstar = wstar/np.linalg.norm(wstar)

    preds = sigmoid(np.dot(X,wstar))
    y = np.ones((n,1))
    y[preds > np.random.rand(n,1)] = 0
    return (X,y,wstar)

def logits_loss(a,b):
    return -a*np.log(b) - (1-a)*np.log(1-b)

def logistic_cost(X,Y,w):
    n,d = X.shape
    b = sigmoid(X.dot(w))
    return np.average(logits_loss(Y,b))

def logistic_gradient(X,Y,w):
    n,d = X.shape
    return (1/n)*(X.T.dot(sigmoid(X.dot(w))-Y))
```

In [25]:

```
n,d = 1000, 50
X,Y,wstar = generate_logistic(n,d)
objective = lambda w: logistic_cost(X, Y, w)
gradient = lambda w: logistic_gradient(X, Y, w)

w0 = np.random.randn(d,1)
steps = [0.05, 0.1, 0.2]
wgd = [gradient_descent(w0, [step]*500, gradient) for step in steps]
wnagd = [nagd(w0,gradient,step,500) for step in steps]

HTML(animated_lplot([[objective(w) for w in wgdi] for wgdi in wgd] + [[objective(w) for w in wn
                    ['GD step ' + str(x) for x in steps]+['NAGD step '+ str(x) for x in steps])
```

Out[25]:

0:49 / 0:50

In [26]:
```
distance_gd = [[np.linalg.norm(w - wstar) for w in ws] for ws in wgd]
distance_nagd = [[np.linalg.norm(w - wstar) for w in wnagdi] for wnagdi in wnagd]
animl = animated_lplot(distance_gd+distance_nagd,
                       ['GD step ' + str(x) for x in steps]+['NAGD step '+ str(x) for x in step
HTML(animl.to_html5_video())
```

Out[26]:

0:49 / 0:50