Lab 1 : Jesse Livezey

Worked with: Joe Thurakal

In [1]:
```
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
```

1. Membrane Model

(a)

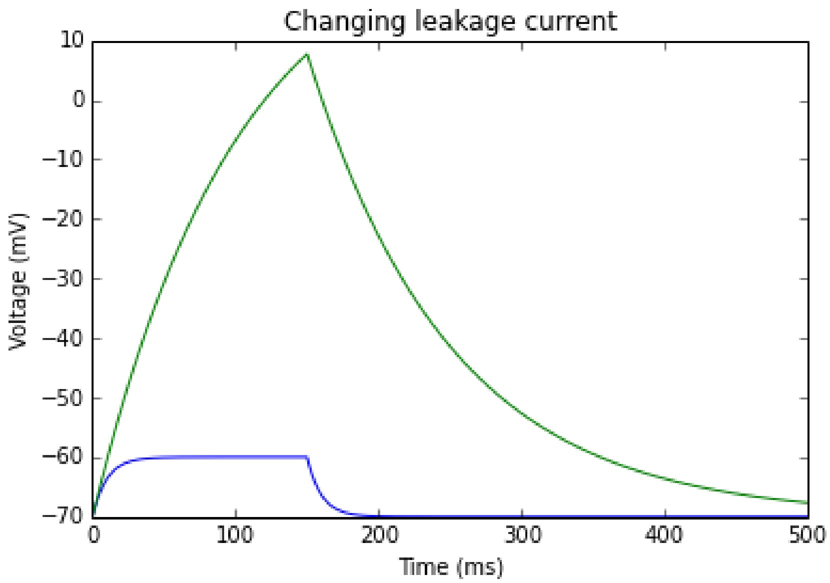
In [2]:
```
t = 500.
N = 5000
dt = t/N
v_r = -70.
c_m = 100
i_in = lambda tt: 100. if tt<150. else 0.
v_t = np.zeros(shape=(2,N))
v_t[:,0] = v_r
time = np.linspace(0,t, N)
```

In [3]:
```
g_l = 10.
for tt in xrange(1,N):
    v_t[0,tt] = v_t[0,tt-1]+dt*((v_r-v_t[0,tt-1])*g_l
                                +i_in(dt*tt))/c_m
```

In [4]:
```
g_l = 1.
for tt in xrange(1,N):
    v_t[1,tt] = v_t[1,tt-1]+dt*((v_r-v_t[1,tt-1])*g_l
                                +i_in(dt*tt))/c_m
```

In [5]:
```
plt.plot(time, v_t[0], time, v_t[1])
plt.title('Changing leakage current')
plt.ylabel('Voltage (mV)')
plt.xlabel('Time (ms)')
```

Out[5]: <matplotlib.text.Text at 0x10c671790>

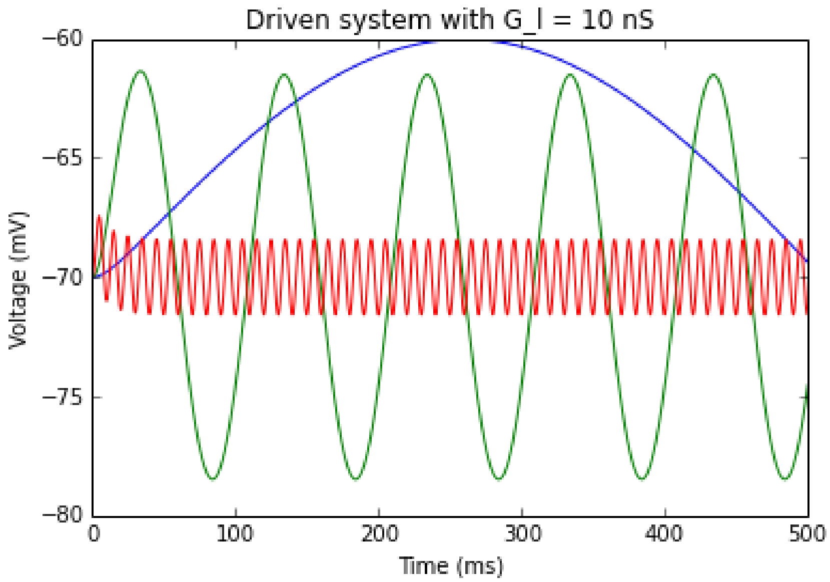


(b)

In [6]:
```
t = 500.
N = 5000
dt = t/N
i_in2 = lambda tt, f: 100.*np.sin(2.*np.pi*f*tt/1000.)
v_t2 = np.zeros(shape=(2,3,N))
v_t2[...,0] = v_r
time = np.linspace(0,t, N)
fs = [1,10,100]
```

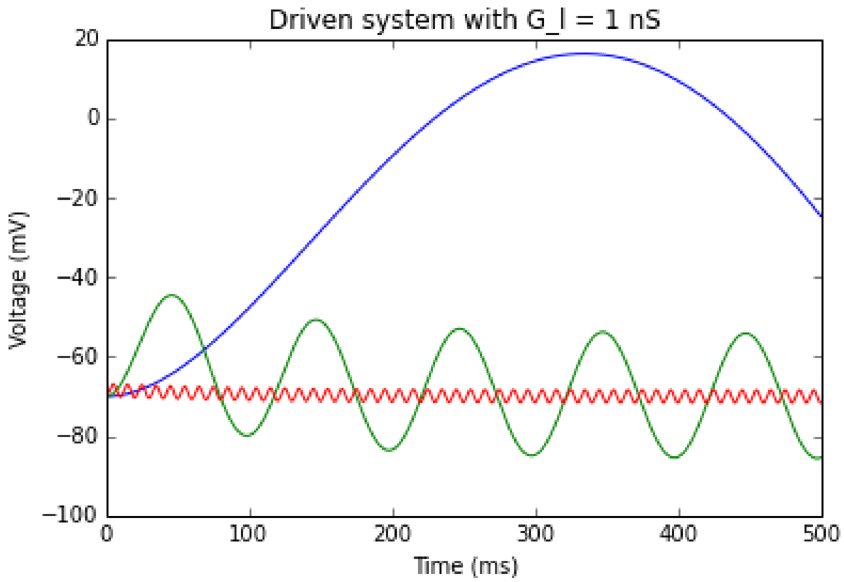
In [7]:
```
g_l = 10.
for ii, f in enumerate(fs):
    for tt in xrange(1,N):
        v_t2[0,ii,tt] = v_t2[0,ii,tt-1]+dt*((v_r-v_t2[0,ii,tt-1])*g_l
                                             +i_in2(dt*tt, f))/c_m
plt.plot(time, v_t2[0,0], time, v_t2[0,1], time, v_t2[0,2])
plt.title('Driven system with G_l = 10 nS')
plt.ylabel('Voltage (mV)')
plt.xlabel('Time (ms)')
```

Out[7]: <matplotlib.text.Text at 0x10c660f50>



In [8]:
```
g_l = 1.
for ii, f in enumerate(fs):
    for tt in xrange(1,N):
        v_t2[0,ii,tt] = v_t2[0,ii,tt-1]+dt*((v_r-v_t2[0,ii,tt-1])*g_l
                                            +i_in2(dt*tt, f))/c_m
plt.plot(time, v_t2[0,0], time, v_t2[0,1], time, v_t2[0,2])
plt.title('Driven system with G_l = 1 nS')
plt.ylabel('Voltage (mV)')
plt.xlabel('Time (ms)')
```

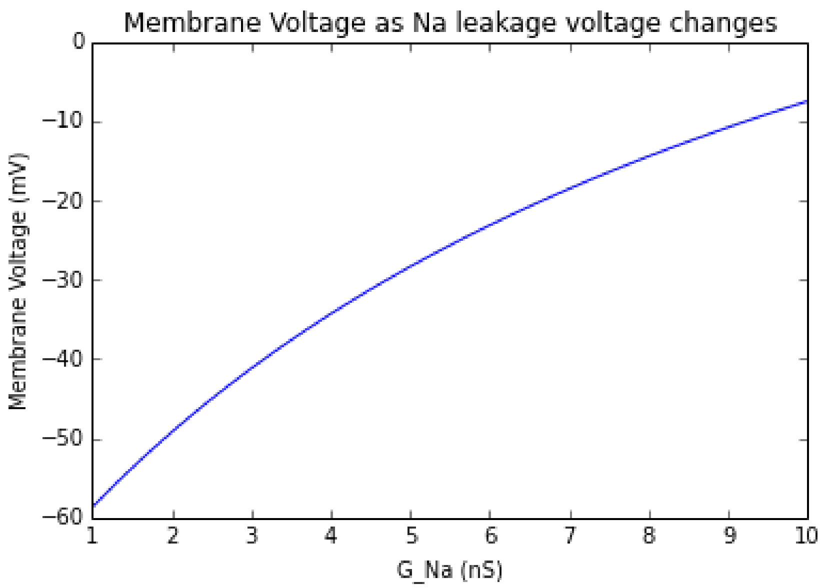
Out[8]: <matplotlib.text.Text at 0x10c6b9890>



2. Shunting Inhibition

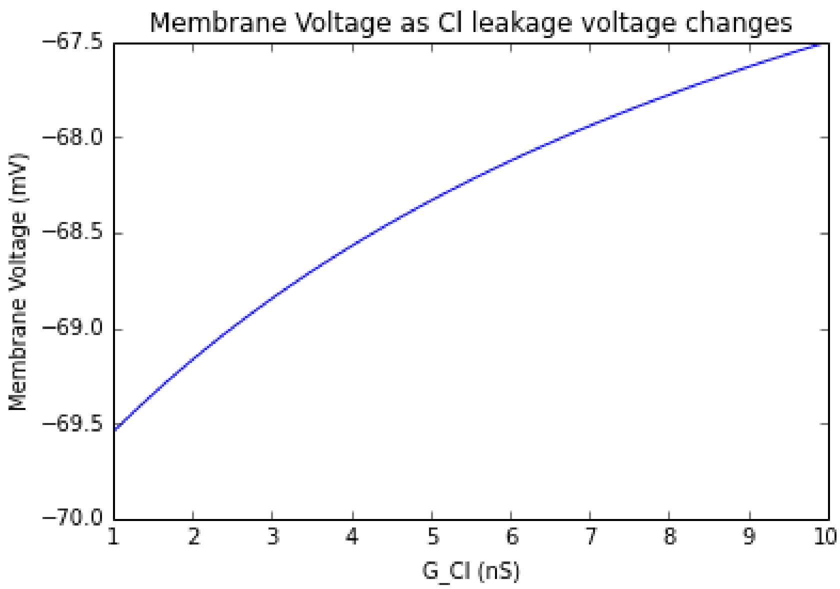
In [9]:
```
N = 100
def V_no_Cl(G):
    return (-70.+55.*(G/10.))/(1.+(G/10.))
g = np.linspace(1.,10.,100)
v_na = np.zeros_like(g)
for ii in xrange(N):
    v_na[ii] = V_no_Cl(g[ii])
plt.plot(g,v_na)
plt.xlabel('G_Na (nS)')
plt.ylabel('Membrane Voltage (mV)')
plt.title('Membrane Voltage as Na leakage voltage changes')
```

Out[9]:
```
<matplotlib.text.Text at 0x10d118890>
```



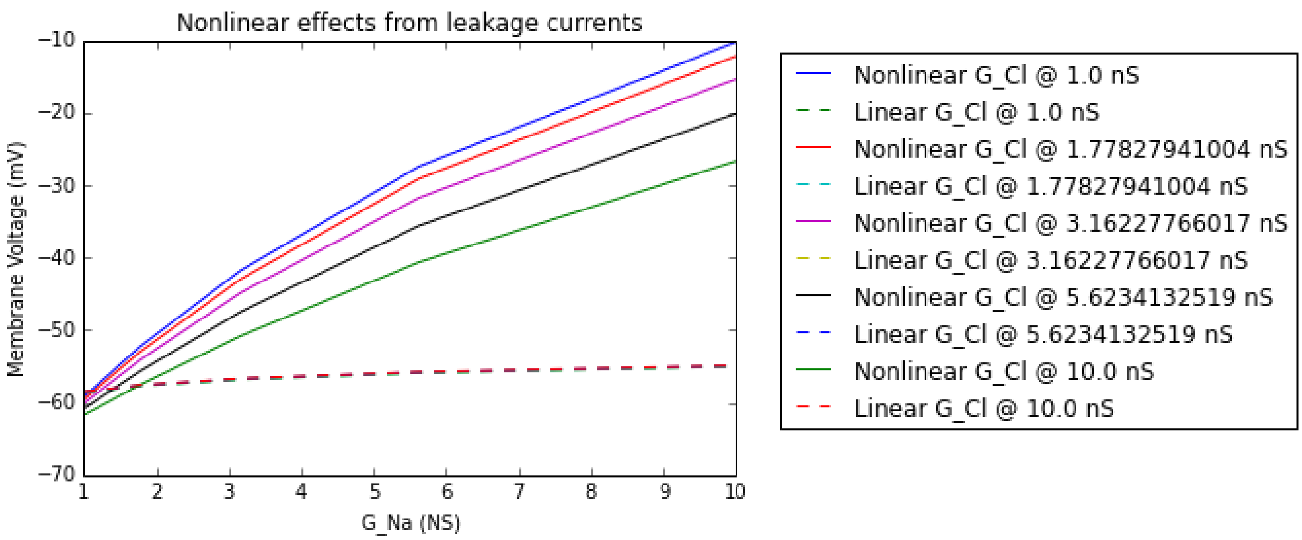
In [10]:
```
def V_no_Na(G):
    return (-70.-65.*(G/10.))/(1.+(G/10.))
g = np.linspace(1.,10.,N)
v_cl = np.zeros_like(g)
for ii in xrange(N):
    v_cl[ii] = V_no_Na(g[ii])
plt.plot(g,v_cl)
plt.xlabel('G_Cl (nS)')
plt.ylabel('Membrane Voltage (mV)')
plt.title('Membrane Voltage as Cl leakage voltage changes')
```

Out[10]: <matplotlib.text.Text at 0x10d37f190>



In [11]:
```
N=5
def V_all(G_Na, G_Cl):
    return (-70.*10.-65.*G_Cl+55.*G_Na)/(10.+G_Cl+G_Na)
g_na = np.logspace(0,1,N)
g_cl = np.logspace(0,1,N)
v = np.zeros(N)
v2 = np.zeros(N)
for jj in xrange(N):
    for ii in xrange(N):
        v[ii] = V_all(g_na[ii],g_cl[jj])
        v2[ii] = v_na[ii]
    plt.plot(g_na, v, label='Nonlinear G_Cl @ '+str(g_cl[jj])+' nS')
    plt.plot(g_na,v2+v_cl[jj]-v_cl[0],'--',
             label='Linear G_Cl @ '+str(g_cl[jj])+' nS')
plt.legend(bbox_to_anchor=(1.05,1),loc=2)
plt.xlabel('G_Na (NS)')
plt.ylabel('Membrane Voltage (mV)')
plt.title('Nonlinear effects from leakage currents')
```

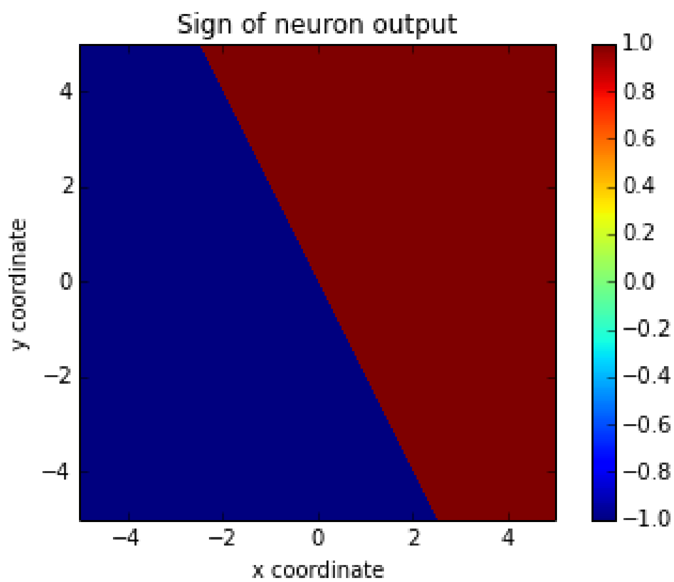
Out[11]: <matplotlib.text.Text at 0x10d4ae950>



3. Linear Neuron

In [15]:
```
w = np.array([-.5, 1.])
N = 1000
image = np.zeros(shape=(N,N))
x = np.linspace(-5,5,N)
for ii in xrange(N):
    for jj in xrange(N):
        image[ii,jj] = np.sign(w.dot([x[ii],x[jj]]))
plt.imshow(image, interpolation='nearest', extent=(-5,5,-5,5))
plt.colorbar()
plt.xlim(-5,5)
plt.ylim(-5,5)
plt.xlabel('x coordinate')
plt.ylabel('y coordinate')
plt.title('Sign of neuron output')
```

Out[15]: <matplotlib.text.Text at 0x11563ca10>

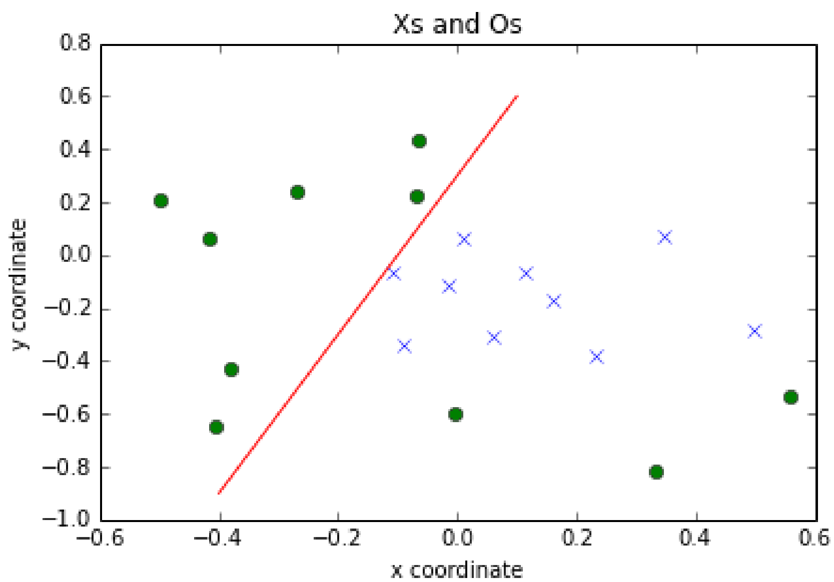


4. Pattern Discrimination

In [13]:
```
from scipy.io import loadmat
data = loadmat('data.mat')
X = data['X'].T
O = data['O'].T
```

In [14]:
```
plt.plot(X[0],X[1],'x',O[0],O[1],'o')
plt.title('Xs and Os')
x = np.linspace(-.4, .1,100)
y = 3.*x+.3
plt.plot(x,y)
plt.xlabel('x coordinate')
plt.ylabel('y coordinate')
```

Out[14]: <matplotlib.text.Text at 0x115534f50>



A hand-tuned kernel that could be used would be the distance from the center of the x's.

More generally, one could take combinations of \(x^ny^m\).