Gradient Descent-Momentum

In [1]:

```
from __future__ import division, print_function, unicode_literals
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
# import matplotlib.cm as cm
# import matplotlib.mlab as mlab
np.random.seed(2)

X = np.random.rand(1000, 1)
y = 4 + 3 * X + .2*np.random.randn(1000, 1)

# Building Xbar
one = np.ones((X.shape[0],1))
Xbar = np.concatenate((one, X), axis = 1)

A = np.dot(Xbar.T, Xbar)
b = np.dot(Xbar.T, y)
w_exact = np.dot(np.linalg.pinv(A), b)


def cost(w):
    return .5/Xbar.shape[0]*np.linalg.norm(y - Xbar.dot(w), 2)**2;

def grad(w):
    return 1/Xbar.shape[0] * Xbar.T.dot(Xbar.dot(w) - y)


def numerical_grad(w, cost):
    eps = 1e-4
    g = np.zeros_like(w)
    for i in range(len(w)):
        w_p = w.copy()
        w_n = w.copy()
        w_p[i] += eps
        w_n[i] -= eps
        g[i] = (cost(w_p) - cost(w_n))/(2*eps)
    return g

def check_grad(w, cost, grad):
    w = np.random.rand(w.shape[0], w.shape[1])
    grad1 = grad(w)
    grad2 = numerical_grad(w, cost)
    return True if np.linalg.norm(grad1 - grad2) < 1e-6 else False


print( 'Checking gradient...', check_grad(np.random.rand(2, 1), cost, grad))
```

```
Checking gradient... True
```

In [2]:

```
def GD_momentum(w_init, grad, eta, gamma):
    w = [w_init]
    v = [np.zeros_like(w_init)]
    for it in range(100):
        v_new = gamma*v[-1] + eta*grad(w[-1])
        w_new = w[-1] - v_new
#         print(np.linalg.norm(grad(w_new))/len(w_new))
        if np.linalg.norm(grad(w_new))/len(w_new) < 1e-3:
            break
        w.append(w_new)
        v.append(v_new)
    return (w, it)
w_init = np.array([[2], [1]])
(w_mm, it_mm) = GD_momentum(w_init, grad, .5, 0.9)
# print(it_mm, w_mm)
```

In [3]:

```
N = X.shape[0]
a1 = np.linalg.norm(y, 2)**2/N
b1 = 2*np.sum(X)/N
c1 = np.linalg.norm(X, 2)**2/N
d1 = -2*np.sum(y)/N
e1 = -2*X.T.dot(y)/N

matplotlib.rcParams['xtick.direction'] = 'out'
matplotlib.rcParams['ytick.direction'] = 'out'

delta = 0.025
xg = np.arange(1.5, 7.0, delta)
yg = np.arange(0.5, 4.5, delta)
Xg, Yg = np.meshgrid(xg, yg)
Z = a1 + Xg**2 +b1*Xg*Yg + c1*Yg**2 + d1*Xg + e1*Yg
```

In [4]:

```
import matplotlib.animation as animation
```

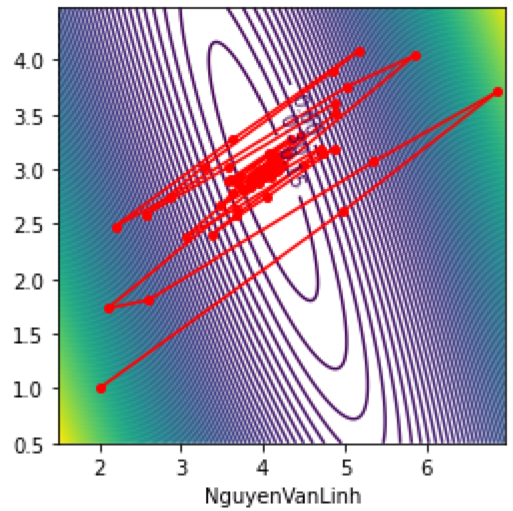
```
from matplotlib.animation import FuncAnimation
def save_gif2(eta, gamma):
    (w, it) = GD_momentum(w_init, grad, eta, gamma)
    fig, ax = plt.subplots(figsize=(4,4))
    plt.cla()
    plt.axis([1.5, 7, 0.5, 4.5])
#     x0 = np.linspace(0, 1, 2, endpoint=True)

    def update(ii):
        if ii == 0:
            plt.cla()
            CS = plt.contour(Xg, Yg, Z, 100)
            manual_locations = [(4.5, 3.5), (4.2, 3), (4.3, 3.3)]
            animlist = plt.clabel(CS, inline=.1, fontsize=10, manual=manual_locations)
#             animlist = plt.title('labels at selected locations')
            plt.plot(w_exact[0], w_exact[1], 'go')
        else:
            animlist = plt.plot([w[ii-1][0], w[ii][0]], [w[ii-1][1], w[ii][1]], 'r-')
        animlist = plt.plot(w[ii][0], w[ii][1], 'ro', markersize = 4)
        xlabel = "NguyenVanLinh"
        ax.set_xlabel(xlabel)
        return animlist, ax

    anim1 = FuncAnimation(fig, update, frames=np.arange(0, it), interval=200)
#     fn = 'img2_' + str(eta) + '.gif'
    fn = 'LR_momentum_contours.gif'
    anim1.save(fn, dpi=100, writer='imagemagick')

eta = 1
gamma = .9
save_gif2(eta, gamma)
# save_gif2(.1)
# save_gif2(2)
```

```
MovieWriter imagemagick unavailable; using Pillow instead.
```



Nesterov accelerated gradient (NAG)

In [5]:

```
from __future__ import division, print_function, unicode_literals
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
# import matplotlib.cm as cm
# import matplotlib.mlab as mlab
np.random.seed(2)
```

In [6]:

```
X = np.random.rand(1000, 1)
y = 4 + 3 * X + .2*np.random.randn(1000, 1)

# Building Xbar
one = np.ones((X.shape[0],1))
Xbar = np.concatenate((one, X), axis = 1)

A = np.dot(Xbar.T, Xbar)
b = np.dot(Xbar.T, y)
w_exact = np.dot(np.linalg.pinv(A), b)
```

In [7]:

```
def cost(w):
    return .5/Xbar.shape[0]*np.linalg.norm(y - Xbar.dot(w), 2)**2;

def grad(w):
    return 1/Xbar.shape[0] * Xbar.T.dot(Xbar.dot(w) - y)
```

In [8]:

```
def numerical_grad(w, cost):
    eps = 1e-4
    g = np.zeros_like(w)
    for i in range(len(w)):
        w_p = w.copy()
        w_n = w.copy()
        w_p[i] += eps
        w_n[i] -= eps
        g[i] = (cost(w_p) - cost(w_n))/(2*eps)
    return g

def check_grad(w, cost, grad):
    w = np.random.rand(w.shape[0], w.shape[1])
    grad1 = grad(w)
    grad2 = numerical_grad(w, cost)
    return True if np.linalg.norm(grad1 - grad2) < 1e-6 else False


print( 'Checking gradient...', check_grad(np.random.rand(2, 1), cost, grad))
print(grad(np.random.rand(2, 1)))
print(numerical_grad(np.random.rand(2, 1), cost))
```

```
Checking gradient... True
[[-5.11424113]
 [-2.71307444]]
[[-4.33336451]
 [-2.27599051]]
```

In [9]:

```
def GD_momentum(w_init, grad, eta, gamma):
    w = [w_init]
    v = [np.zeros_like(w_init)]
    for it in range(100):
        v_new = gamma*v[-1] + eta*grad(w[-1])
        w_new = w[-1] - v_new
#         print(np.linalg.norm(grad(w_new))/len(w_new))
        if np.linalg.norm(grad(w_new))/len(w_new) < 1e-3:
            break
        w.append(w_new)
        v.append(v_new)
    return (w, it)
w_init = np.array([[2], [1]])
(w_mm, it_mm) = GD_momentum(w_init, grad, .5, 0.9)
print(it_mm, w_mm)
```

```
87 [array([[2],
       [1]]), array([[3.4866689 ],
       [1.80445925]]), array([[5.37195597],
       [2.84266566]]), array([[6.42034152],
       [3.46665316]]), array([[6.0392551 ],
       [3.36309504]]), array([[4.58742458],
       [2.71409019]]), array([[3.05600426],
       [2.0312321 ]]), array([[2.38507806],
       [1.79973605]]), array([[2.88047602],
       [2.17479531]]), array([[4.08646356],
       [2.91536603]]), array([[5.14850841],
       [3.57319621]]), array([[5.38966703],
       [3.79309933]]), array([[4.71785761],
       [3.52511842]]), array([[3.62558949],
       [3.02439001]]), array([[2.82307179],
       [2.66002577]]), array([[2.77138508],
       [2.67192254]]), array([[3.41839033],
       [3.03315809]]), array([[4.28268143],
       [3.49366186]]), array([[4.79815794],
       [3.75962714]]), array([[4.67714645],
       [3.68257908]]), array([[4.06257837],
       [3.33856546]]), array([[3.39493066],
       [2.95877508]]), array([[3.10589131],
       [2.76990434]]), array([[3.34814758],
       [2.85334353]]), array([[3.92710738],
       [3.10955089]]), array([[4.45718186],
       [3.33940196]]), array([[4.62220685],
       [3.37979057]]), array([[4.36633197],
       [3.20302148]]), array([[3.90266385],
       [2.92128552]]), array([[3.55247587],
       [2.7028907 ]]), array([[3.53273754],
       [2.6615835 ]]), array([[3.83033977],
       [2.79103593]]), array([[4.2331994 ],
       [2.98105422]]), array([[4.48305278],
       [3.09717862]]), array([[4.44197395],
       [3.06743726]]), array([[4.16684362],
       [2.92115853]]), array([[3.85428071],
       [2.76031394]]), array([[3.70350829],
[2.68811718]]), array([[3.79132811],
[2.74393106]]), array([[4.03636902],
[2.8846746 ]]), array([[4.26608877],
[3.01975039]]), array([[4.3342412 ],
[3.07225614]]), array([[4.210111  ],
[3.02548814]]), array([[3.98638904],
[2.92706377]]), array([[3.80888183],
[2.85232617]]), array([[3.7799353 ],
[2.85405378]]), array([[3.89874562],
[2.93137964]]), array([[4.07228738],
[3.03548975]]), array([[4.18294457],
[3.10485286]]), array([[4.16477304],
[3.10494303]]), array([[4.03971945],
[3.04710166]]), array([[3.89509493],
[2.97679833]]), array([[3.82230199],
[2.9417074 ]]), array([[3.85910572],
[2.96162918]]), array([[3.97128958],
[3.0189244 ]]), array([[4.0812606 ],
[3.07340255]]), array([[4.12097804],
[3.08987956]]), array([[4.07359304],
[3.05985539]]), array([[3.97882533],
[3.00430378]]), array([[3.90233515],
[2.95770891]]), array([[3.89189517],
[2.94522755]]), array([[3.94916209],
[2.96797794]]), array([[4.03318743],
[3.00486201]]), array([[4.09029391],
[3.02812222]]), array([[4.08895157],
[3.0215041 ]]), array([[4.03728933],
[2.98937529]]), array([[3.9740002 ],
[2.95198764]]), array([[3.94100292],
[2.93123782]]), array([[3.95682375],
[2.93680412]]), array([[4.00731394],
[2.96131396]]), array([[4.05778832],
[2.98666972]]), array([[4.07683201],
[2.99645288]]), array([[4.05568204],
[2.98602354]]), array([[4.01147443],
[2.96421619]]), array([[3.97393327],
[2.94641093]]), array([[3.96550169],
[2.94418861]]), array([[3.98802612],
[2.95839925]]), array([[4.02368556],
[2.97964986]]), array([[4.04815794],
[2.99516594]]), array([[4.04654445],
[2.9970889 ]]), array([[4.02179181],
[2.98686539]]), array([[3.99108174],
[2.97336854]]), array([[3.97365425],
       [2.96655669]]), array([[3.9785549 ],
       [2.97109199]]), array([[3.99999528],
       [2.98392403]]), array([[4.02247402],
       [2.99695692]]), array([[4.03147176],
       [3.00261938]]), array([[4.02245776],
       [2.99855499]])]
```

In [10]:

```
def myGD(w_init, grad, eta):
    w = [w_init]
    for it in range(100):
        w_new = w[-1] - eta*grad(w[-1])
        if np.linalg.norm(grad(w_new))/len(w_new) < 1e-3:
            break
        w.append(w_new)
        # print('iter %d: ' % it, w[-1].T)
    return (w, it)

w_init = np.array([[2], [1]])
(w1, it1) = myGD(w_init, grad, 0.1)
(w2, it2) = myGD(w_init, grad, 1)
(w3, it3) = myGD(w_init, grad, 2)

print(it1, it2, it3)
```

```
99 49 99
```

In [11]:

```
N = X.shape[0]
a1 = np.linalg.norm(y, 2)**2/N
b1 = 2*np.sum(X)/N
c1 = np.linalg.norm(X, 2)**2/N
d1 = -2*np.sum(y)/N
e1 = -2*X.T.dot(y)/N

matplotlib.rcParams['xtick.direction'] = 'out'
matplotlib.rcParams['ytick.direction'] = 'out'

delta = 0.025
xg = np.arange(1.5, 6.0, delta)
yg = np.arange(0.5, 4.5, delta)
Xg, Yg = np.meshgrid(xg, yg)
Z = a1 + Xg**2 +b1*Xg*Yg + c1*Yg**2 + d1*Xg + e1*Yg
```

In [12]:

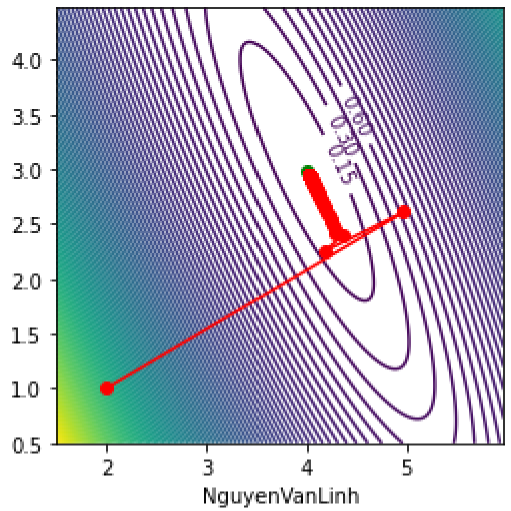
```
def save_gif2(eta):
    (w, it) = myGD(w_init, grad, eta)
    fig, ax = plt.subplots(figsize=(4,4))
    plt.cla()
    plt.axis([1.5, 6, 0.5, 4.5])
#     x0 = np.linspace(0, 1, 2, endpoint=True)

    def update(ii):
        if ii == 0:
            plt.cla()
            CS = plt.contour(Xg, Yg, Z, 100)
            manual_locations = [(4.5, 3.5), (4.2, 3), (4.3, 3.3)]
            animlist = plt.clabel(CS, inline=.1, fontsize=10, manual=manual_locations)
#             animlist = plt.title('labels at selected locations')
            plt.plot(w_exact[0], w_exact[1], 'go')
        else:
            animlist = plt.plot([w[ii-1][0], w[ii][0]], [w[ii-1][1], w[ii][1]], 'r-')
        animlist = plt.plot(w[ii][0], w[ii][1], 'ro')
        xlabel = "NguyenVanLinh"
        ax.set_xlabel(xlabel)
        return animlist, ax

    anim1 = FuncAnimation(fig, update, frames=np.arange(0, it), interval=200)
    fn = 'img2_' + str(eta) + '.gif'
    anim1.save(fn, dpi=100, writer='imagemagick')

save_gif2(1)
# save_gif2(.1)
# save_gif2(2)
```

```
MovieWriter imagemagick unavailable; using Pillow instead.
```



In [ ]: