weightloss

February 2, 2018

```
In [2]: from scipy.integrate import ode
        from scipy.integrate import odeint
        import numpy as np
        import matplotlib.pyplot as plt
        from math import log
        %matplotlib inline
```

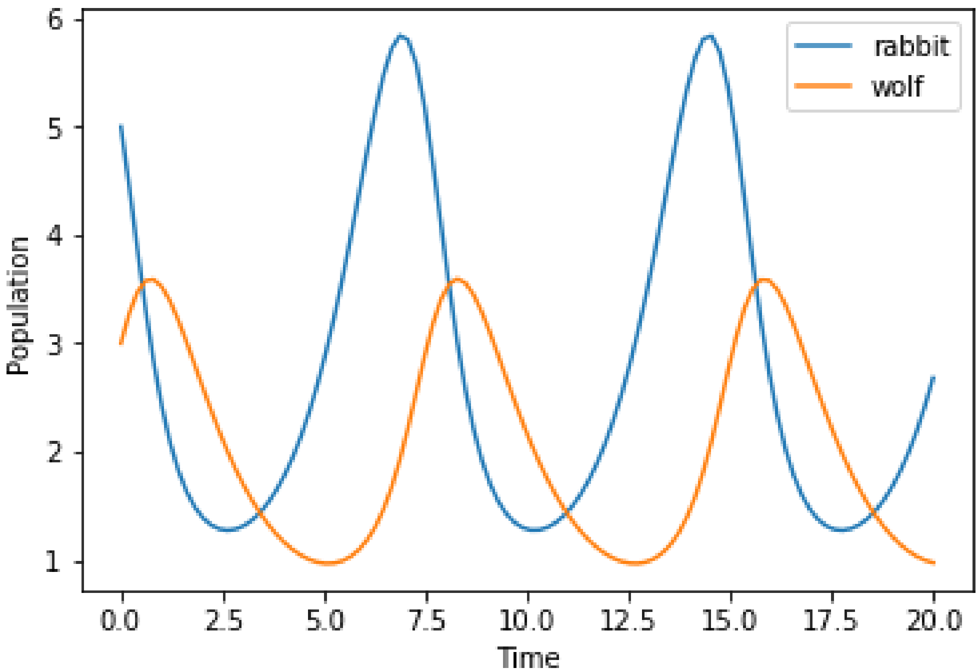
0.1 Modeling Systems of Differential Equations

Include SIR models and Lotka-Volterra

```
In [3]: def predator_prey(t, y, a, alpha, c, gamma):
            r0, w0=y
            dr=a*r0-alpha*r0*w0
            dw=-c*w0+gamma*r0*w0
            y=[dr,dw]
            return(y)
```

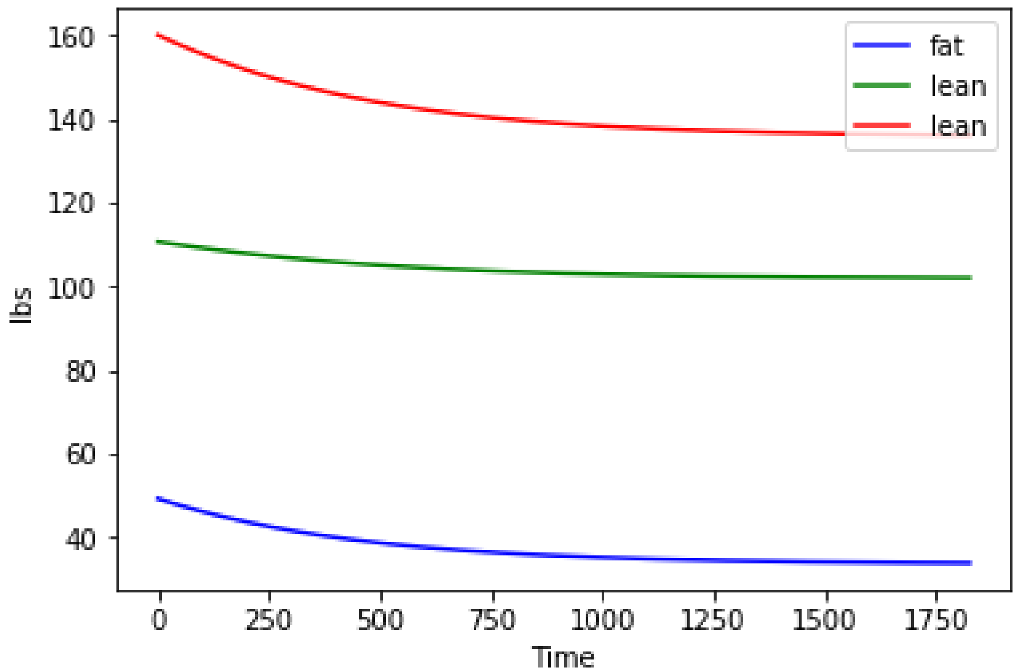
```
In [4]: #problem 1
        r0 = 5 # Initial rabbit population
        w0 = 3 # Initial wolf population
        # Define rabbit growth paramters
        a = 1.0
        alpha = 0.5
        # Define wolf growth parameters
        c = 0.75
        gamma = 0.25
        t_f = 20 # How long we want to run the model
        y0 = [r0, w0]
        # Initialize time and output arrays needed for the ode solver
        t = np.linspace(0, t_f, 5*t_f)
        y = np.zeros((len(t), len(y0)))
        y[0,:] = y0

        predator_prey_ode=lambda t, y:predator_prey(t, y, a, alpha, c, gamma)
        p_p_solver = ode(predator_prey_ode).set_integrator('dopri5')
        p_p_solver.set_initial_value(y0, 0)
        for j in range(1, len(t)):
```

```
        y[j,:] = p_p_solver.integrate((t[j]))

    plt.plot(t, y[:,0], label='rabbit')
    plt.plot(t, y[:,1], label='wolf')
    plt.legend()
    plt.xlabel('Time')
    plt.ylabel('Population')
    plt.show()
```



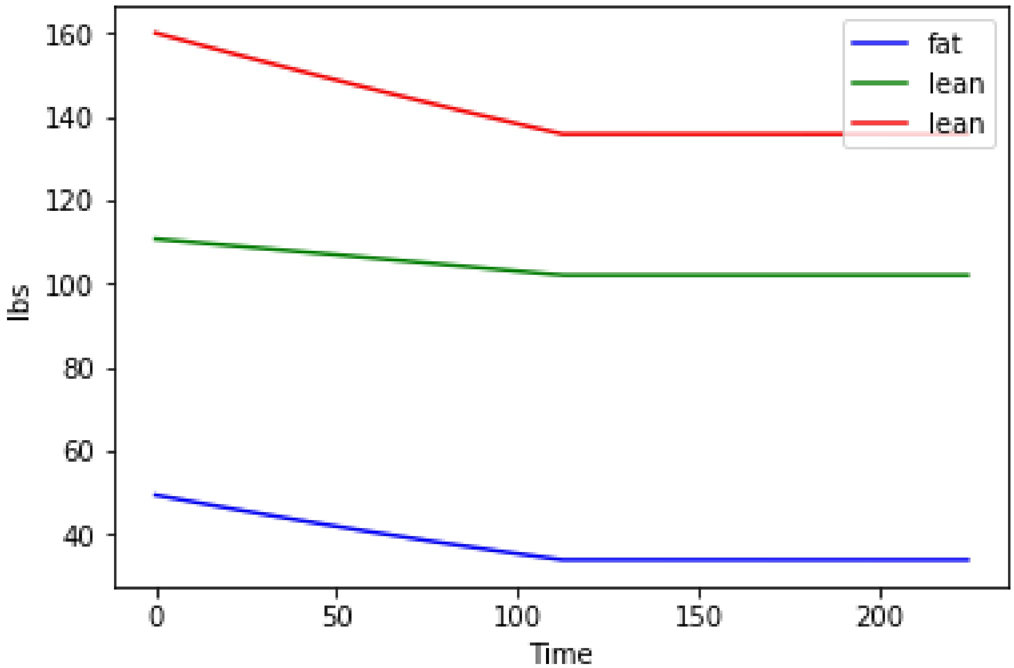

```
In [5]: #problem 2
        rho_F = 9400.
        rho_L = 1800.
        gamma_F = 3.2
        gamma_L = 22.
        eta_F = 180.
        eta_L = 230.
        C = 10.4 # Forbes constant
        beta_AT = 0.14 # Adaptive Thermogenesis
        beta_TEF = 0.1 # Thermic Effect of Feeding
        K = 0

        def forbes(F):
            C1 = C * rho_L / rho_F
            return C1 / (C1 + F)
```

```python
def energy_balance(F, L, EI, PAL):
    p = forbes(F)
    a1 = (1. / PAL - beta_AT) * EI - K - gamma_F * F - gamma_L * L
    a2 = (1 - p) * eta_F / rho_F + p * eta_L / rho_L + 1. / PAL
    return a1 / a2

def weight_odesystem(t, y, EI, PAL):
    F, L = y[0], y[1]
    p, EB = forbes(F), energy_balance(F, L, EI, PAL)
    return np.array([(1 - p) * EB / rho_F , p * EB / rho_L])

def fat_mass(BW, age, H, sex):
    BMI = BW / H**2.
    if sex == 'male':
        return BW * (-103.91 + 37.31 * log(BMI) + 0.14 * age) / 100
    else:
        return BW * (-102.01 + 39.96 * log(BMI) + 0.14 * age) / 100



F0=fat_mass(72.5748, 38, 1.7272, 'f')
L0=72.5748-F0

t = np.linspace(0, 365*5, 5*365*5)
y = np.zeros((len(t), 2))
y[0:,]=[F0,L0]
weight_ode=lambda t, y: weight_odesystem(t, y, 2025, 1.5)
solver=ode(weight_ode).set_integrator('dopri5')
solver.set_initial_value(y[0], 0)

for j in range(1, len(t)):
    y[j,:] = solver.integrate(t[j])

plt.plot(t, 2.20462262*y[:,0], label='fat', color='b')
plt.plot(t, 2.20462262*y[:,1], label='lean', color='g')
plt.plot(t, 2.20462262*(y[:,1]+y[:,0]), label='lean', color='r')
plt.legend()
plt.xlabel('Time')
plt.ylabel('lbs')
plt.show()
```



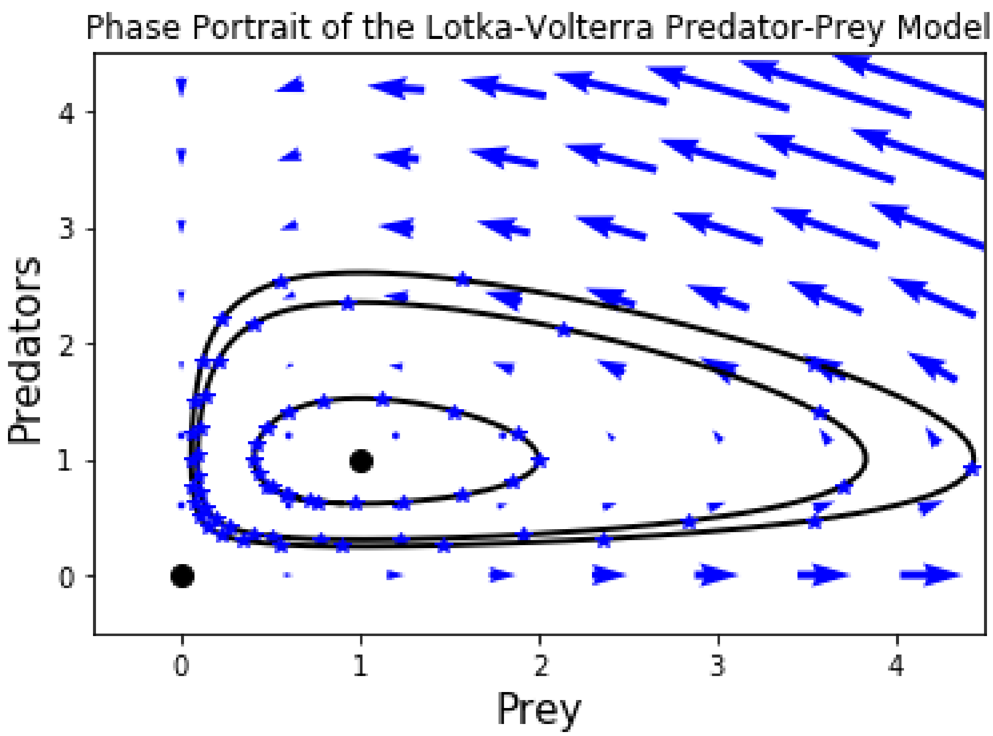
In [6]:
```
#problem 4
F0=fat_mass(72.5748, 38, 1.7272, 'f')
L0=72.5748-F0

t = np.linspace(0, 32*7, 32*7*4+1)
y = np.zeros((len(t), 2))
y[0:,]=[F0,L0]
weight_ode=lambda t, y: weight_odesystem(t, y, 1600, 1.7)
second_ode=lambda t, y: weight_odesystem(t, y, 2025, 1.5)
solver=ode(weight_ode).set_integrator('dopri5')
solver2=ode(second_ode).set_integrator('dopri5')
solver.set_initial_value(y[0], 0)

for j in range(1, len(t)):
    if t[j] <16*7:
        y[j,:] = solver.integrate(t[j])
    elif t[j] == 16*7:
        y[j,:] = solver.integrate(t[j])
        solver2.set_initial_value(y[j], 0)
    else:
        y[j,:] = solver2.integrate(t[j])

plt.plot(t, 2.20462262*y[:,0], label='fat', color='b')
plt.plot(t, 2.20462262*y[:,1], label='lean', color='g')
```

```
plt.plot(t, 2.20462262*(y[:,1]+y[:,0]), label='lean', color='r')
plt.legend()
plt.xlabel('Time')
plt.ylabel('lbs')
plt.show()
```



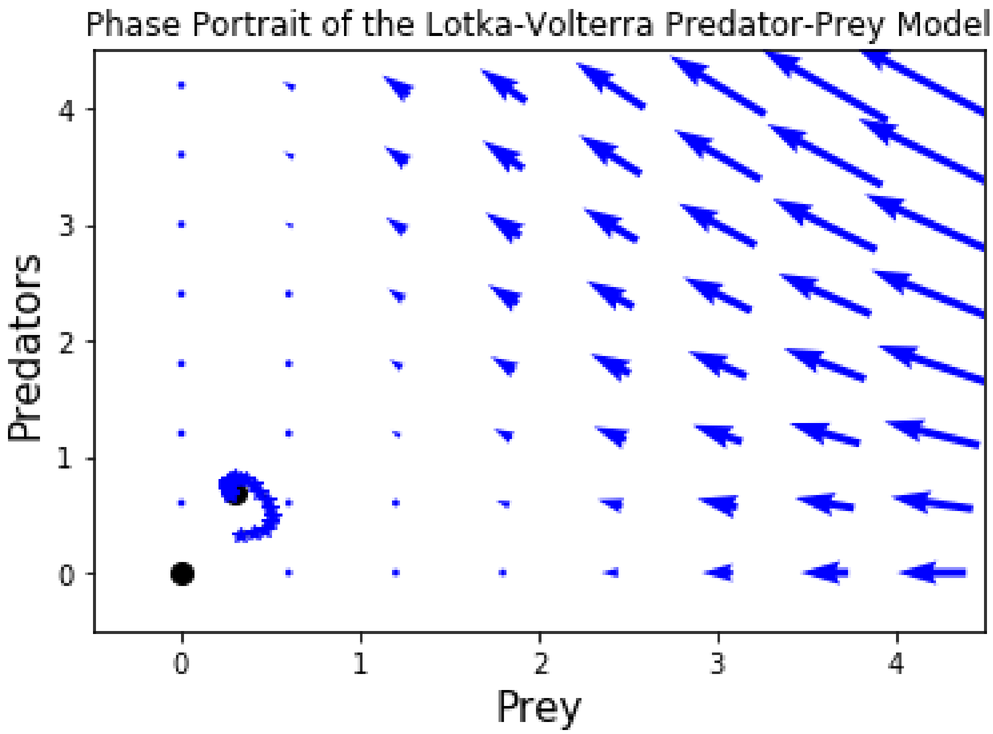
In [7]:
```
#problem 5
a, b = 0., 13.
alpha = 1. / 3 # Nondimensional parameter
dim = 2 # dimension of the system
y0 = np.array([1 / 2., 1 / 3.]) # initial conditions
y1 = np.array([1 / 2., 3 / 4.])
y2 = np.array([1 / 16., 3 / 4.])

# Note: swapping order of arguments to match the calling convention
# used in the built in IVP solver.

def Lotka_Volterra(y, x):
    return np.array([y[0] * (1. - y[1]), alpha * y[1] * (y[0] - 1.)])
subintervals = 200

# Using the built in ode solver
X = odeint(Lotka_Volterra, y1, np.linspace(a, b, subintervals))
Y = odeint(Lotka_Volterra, y0, np.linspace(a, b, subintervals))
Z = odeint(Lotka_Volterra, y2, np.linspace(a, b, subintervals))
```

```
# Plot the direction field
Y1, Y2 = np.meshgrid(np.arange(0, 4.5, .2), np.arange(0, 4.5, .2), sparse=True, copy=Fal
U, V = Lotka_Volterra((Y1, Y2), 0)
Q = plt.quiver(Y1[::3, ::3], Y2[::3, ::3], U[::3, ::3], V[::3, ::3], pivot='mid', color=

# Plot the 2 Equilibrium points
plt.plot(1, 1, 'ok', markersize=8)
plt.plot(0, 0, 'ok', markersize=8)
# Plot the solution in phase space
plt.plot(Y[:,0], Y[:,1], '-k', linewidth=2.0)
plt.plot(Y[::10,0], Y[::10,1], '*b')
plt.plot(X[:,0], X[:,1], '-k', linewidth=2.0)
plt.plot(X[::10,0], X[::10,1], '*b')
plt.plot(Z[:,0], Z[:,1], '-k', linewidth=2.0)
plt.plot(Z[::10,0], Z[::10,1], '*b')
plt.axis([-.5, 4.5, -.5, 4.5])
plt.title("Phase Portrait of the Lotka-Volterra Predator-Prey Model")
plt.xlabel('Prey',fontsize=15)
plt.ylabel('Predators',fontsize=15)
plt.show()
```



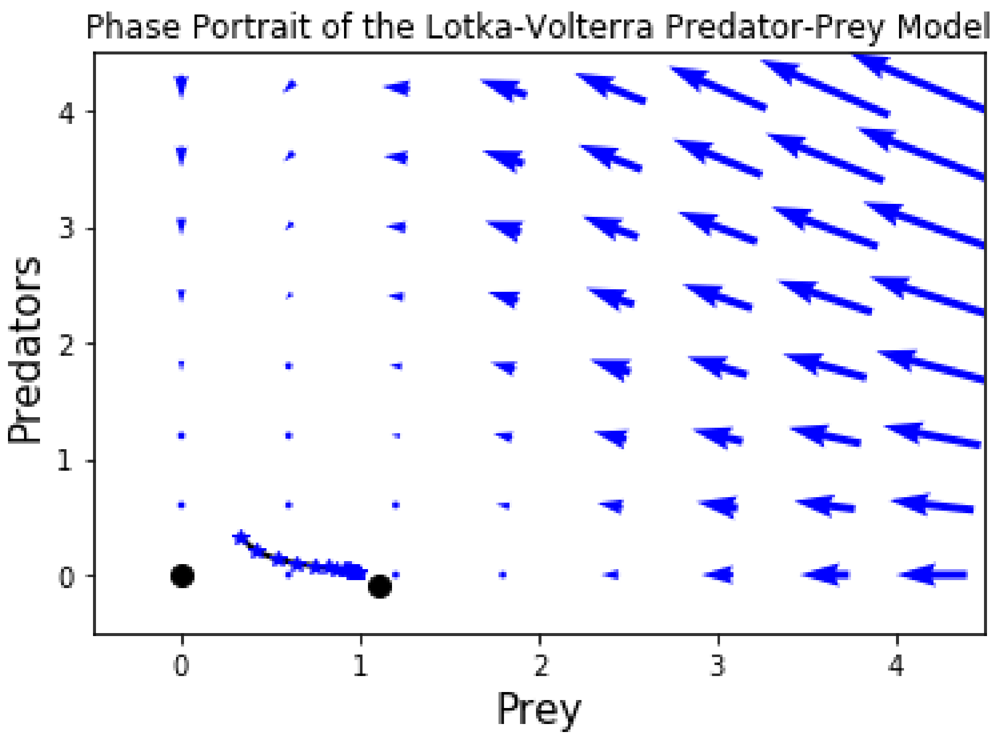

```
In [15]: #problem 6
```

```
a, b = 0., 13.
alpha = 1. # Nondimensional parameter
beta = .3
dim = 2 # dimension of the system
y0 = np.array([1 / 3., 1 / 3.]) # initial conditions

# Note: swapping order of arguments to match the calling convention
# used in the built in IVP solver.

def Lotka_Volterra(y, x):
    return np.array([y[0] * (1. - y[0] - y[1]), alpha * y[1] * (y[0] - beta)])
subintervals = 200

# Using the built in ode solver
Y = odeint(Lotka_Volterra, y0, np.linspace(a, b, subintervals))

# Plot the direction field
Y1, Y2 = np.meshgrid(np.arange(0, 4.5, .2), np.arange(0, 4.5, .2), sparse=True, copy=Fa
U, V = Lotka_Volterra((Y1, Y2), 0)
Q = plt.quiver(Y1[::3, ::3], Y2[::3, ::3], U[::3, ::3], V[::3, ::3], pivot='mid', color

# Plot the 2 Equilibrium points
plt.plot(beta, 1-beta, 'ok', markersize=8)
plt.plot(0, 0, 'ok', markersize=8)
# Plot the solution in phase space
plt.plot(Y[:,0], Y[:,1], '-k', linewidth=2.0)
plt.plot(Y[::10,0], Y[::10,1], '*b')
plt.axis([-.5, 4.5, -.5, 4.5])
plt.title("Phase Portrait of the Lotka-Volterra Predator-Prey Model")
plt.xlabel('Prey',fontsize=15)
plt.ylabel('Predators',fontsize=15)
plt.show()
```




```
In [16]: #problem 6
         a, b = 0., 13.
         alpha = 1. # Nondimensional parameter
         beta = 1.1
         dim = 2 # dimension of the system
         y0 = np.array([1 / 3., 1 / 3.]) # initial conditions

         # Note: swapping order of arguments to match the calling convention
         # used in the built in IVP solver.

         def Lotka_Volterra(y, x):
             return np.array([y[0] * (1. - y[0] - y[1]), alpha * y[1] * (y[0] - beta)])
         subintervals = 200

         # Using the built in ode solver
         Y = odeint(Lotka_Volterra, y0, np.linspace(a, b, subintervals))

         # Plot the direction field
         Y1, Y2 = np.meshgrid(np.arange(0, 4.5, .2), np.arange(0, 4.5, .2), sparse=True, copy=Fa
         U, V = Lotka_Volterra((Y1, Y2), 0)
         Q = plt.quiver(Y1[::3, ::3], Y2[::3, ::3], U[::3, ::3], V[::3, ::3], pivot='mid', color

         # Plot the 2 Equilibrium points
```

```
plt.plot(beta, 1-beta, 'ok', markersize=8)
plt.plot(0, 0, 'ok', markersize=8)
# Plot the solution in phase space
plt.plot(Y[:,0], Y[:,1], '-k', linewidth=2.0)
plt.plot(Y[::10,0], Y[::10,1], '*b')
plt.axis([-.5, 4.5, -.5, 4.5])
plt.title("Phase Portrait of the Lotka-Volterra Predator-Prey Model")
plt.xlabel('Prey',fontsize=15)
plt.ylabel('Predators',fontsize=15)
plt.show()
```



In [ ]: