```
import numpy as np
import matplotlib.pyplot as plt
from math import sqrt

def do_IC(k1, kn1, k2, kn2, k3, x0, y0, dt=0.01, numsteps=256):
    def xprime(x, y):
        return k1 * (1 - x - y) - kn1 * x - k3 * x * y
    def yprime(x, y):
        return k2 * (1 - x - y) - kn2 * y ** 2 - k3 * x * y
    def pprime(x, y):
        return k3 * x * y

    xold = x0
    yold = y0
    tlist = []
    xlist = []
    ylist = []
    plist = []
    pold = 0
    told = 0
    tlist.append(told)
    xlist.append(xold)
    ylist.append(yold)
    plist.append(pold)
    for j in range(numsteps):
        x = xold + dt * xprime(xold, yold)
        y = yold + dt * yprime(xold, yold)
        p = pold + dt * pprime(xold, yold)
        t = told + dt
        xlist.append(x)
        xold = x
        ylist.append(y)
        yold = y
        plist.append(p)
        pold = p
        tlist.append(t)
        told = t
    return (tlist, xlist, ylist, plist)

def make_ICs(resolution=32):
    ICs = []
    for x0 in np.arange(0, 1. + 1./resolution, 1./resolution):
        ICs.append((x0, 0))
        ICs.append((x0, 1))
        ICs.append((0, x0))
        ICs.append((1, x0))
    return ICs

# Problem 2 ###########################################################################
k2 = 1
k3 = 10
fk1 = lambda y: k2*k3*(1-y)/k3/y
fx = lambda y: fk1(y)*(1-y)/(fk1(y)+k3*y)
fr = lambda x,y: k3*x*y
l1 = list(np.arange(0.001,        1,   .001))
l2 = list(np.arange(0.0001,    .001,  .0001))
l3 = list(np.arange(0.00001,  .0001, .00001))
l4 = list(np.arange(0.000001,.00001,.000001))
ylist = l1
ylist.extend(l2)  # We need extra resolution at low y
ylist.extend(l3)  # to properly capture the behavior.
ylist.extend(l4)  # Only a couple dozen extra points.
ylist.sort()
xlist = []
k1list= []
rlist = []

kn1 = 0.05
fk1n = lambda y: (kn1+k3*y)*(sqrt(1+4*k2*(1-y)/k3/y)-1)/2
fxn = lambda y: fk1n(y)*(1-y)/(fk1n(y)-kn1+k3*y)
```

```
 71 frn = fr
 72 ylistn = []
 73 xlistn = []
 74 k1listn= []
 75 rlistn = []
 76 for y in ylist:
 77     x = fx(y)
 78     xlist.append(x)
 79     k1list.append(fk1(y))
 80     rlist.append(fr(x,y))
 81 
 82     ylistn.append(y)
 83     xn = fxn(y)
 84     xlistn.append(xn)
 85     k1listn.append(fk1n(y))
 86     rlistn.append(frn(xn,y))
 87 
 88 # Make a plot comparing reversible and reversible
 89 fig2b = plt.figure(22, figsize=(12, 8))
 90 
 91 ax2O = fig2b.add_subplot(3,1,1)
 92 ax2O.set_xlim((0,1))
 93 ax2O.plot(k1list, ylist, 'k--')
 94 ax2O.plot(k1listn, ylistn, 'k-')
 95 ax2O.set_ylabel(r'Adsorbed $O_1$')
 96 
 97 ax2CO = fig2b.add_subplot(3,1,2)
 98 ax2CO.set_xlim((0,1))
 99 ax2CO.plot(k1list,xlist, 'k--')
100 ax2CO.plot(k1listn,xlistn, 'k-')
101 ax2CO.set_ylabel(r'Adsorbed $CO_1$')
102 
103 ax2r = fig2b.add_subplot(3,1,3)
104 ax2r.set_xlim((0,1))
105 ax2r.set_ylim((0,.5))
106 ax2r.plot(k1list, rlist, 'k--')
107 ax2r.plot(k1listn, rlistn, 'k-')
108 ax2r.set_ylabel(r'Rate of Product Formation, ${d[CO_2]}/{dt}$')
109 ax2r.set_xlabel(r'Partial Pressure of Unadsorbed $CO_2$, $k_1$')
110 
111 ax2r.legend([r'$k_{-1}=0$, Irreversible', r'$k_{-1}=%.2f$, Reversible'%kn1])
112 plt.suptitle('Reversible vs. Irreversible Adsorption and Reaction\n'\
113              r'$k_2=%.2f$, $k_3=%.2f$, $k_{-2}=0$' % (k2, k3))
114 plt.savefig('hw4_2b.pdf')
115 #plt.show()
116 
117 # Make a plot showing irreversible only, since it has
118 # qualitatively interesting large-k1 behavior that won't
119 # fit on the previous plot
120 fig2c = plt.figure(23, figsize=(12, 8))
121 
122 ax2O = fig2c.add_subplot(3,1,1)
123 ax2O.set_xlim((0,100))
124 ax2O.plot(k1list, ylist, 'k--')
125 ax2O.set_ylabel(r'Adsorbed $O_1$')
126 
127 ax2CO = fig2c.add_subplot(3,1,2)
128 ax2CO.set_xlim((0,100))
129 ax2CO.plot(k1list,xlist, 'k--')
130 ax2CO.set_ylabel(r'Adsorbed $CO_1$')
131 
132 ax2r = fig2c.add_subplot(3,1,3)
133 ax2r.set_xlim((0,100))
134 ax2r.plot(k1list, rlist, 'k--')
135 ax2r.set_ylabel(r'Rate of Product Formation, ${d[CO_2]}/{dt}$')
136 ax2r.set_xlabel(r'Partial Pressure of Unadsorbed $CO_2$, $k_1$')
137 
138 ax2r.legend([r'$k_{-1}=0$, Irreversible'])
139 plt.suptitle('Irreversible Adsorption and Reaction\n'\
140              r'$k_2=%.2f$, $k_3=%.2f$, $k_{-2}=0$' % (k2, k3))
```

```
141 plt.savefig('hw4_2c.pdf')
142 #plt.show()
143
144
145 # Problem 3 ###################################################################
146 k1=.5
147 kn1=.05
148 k2=1
149 kn2=0
150 k3=10
151 # roots of the cubic polynomial in y that results from setting
152 #    dx/dt = 0 = dy/dt
153 # coefficients of said polynomial:
154 coeffs = [
155           -k2*k3**2,
156           -2*k2*kn1*k3 + k2*k3**2 - k1*k3**2,
157           -k1**2*k3 - k1*k3*kn1 + 2*k2*k3*kn1 - k2*kn1**2,
158           k2*kn1**2
159          ]
160
161 yroots = np.roots(coeffs)
162 print "The roots are y =", yroots
163 fx = lambda y: k1*(1-y)/(k1+kn1+k3*y)
164 xroots = map(fx, yroots)
165 print "Similarly,    x =", xroots
166
167 fig3 = plt.figure(3, figsize=(12, 8))
168 ax3 = fig3.add_subplot(1,1,1)
169 ICs_ss = zip(xroots, yroots)
170 ICs_ss.append((0, 1))
171
172 for (x0,y0) in ICs_ss:
173     x0 = x0 + np.random.rand()*.25
174     y0 = y0 + np.random.rand()*.25
175     (tlist, xlist, ylist, plist) = do_IC(k1, kn1, k2, kn2, k3, x0, y0)
176     ax3.plot(xlist, ylist, 'k')
177     ax3.scatter([x0],[y0], color='k')
178 ax3.set_title('Small random perturbations from steady-states')
179 ax3.set_xlabel(r'x, adsorbed $CO_1$')
180 ax3.set_ylabel(r'y, adsorbed $O_1$')
181 ax3.set_xbound(lower=-.08)
182 ax3.set_ybound(lower=0)
183 plt.savefig('hw4_3.pdf')
184
185 fig3b = plt.figure(32, figsize=(12, 8))
186 ax3b = fig3b.add_subplot(1,1,1)
187 ICs = make_ICs()
188
189 for (x0,y0) in ICs:
190     (tlist, xlist, ylist, plist) = do_IC(k1, kn1, k2, kn2, k3, x0, y0)
191     ax3b.plot(xlist, ylist, 'k')
192     ax3b.set_xlim((0,1))
193     ax3b.set_ylim((0,1))
194 ax3b.set_xlim((0, 1))
195 ax3b.set_ylim((0, 1))
196 ax3b.set_title('Phase plot for many ICs.')
197 ax3b.set_xlabel(r'x, adsorbed $CO_1$')
198 ax3b.set_ylabel(r'y, adsorbed $O_1$')
199 plt.savefig('hw4_3b.pdf')
200
201 # Problem 3, Linear stability analysis ########################################
202 f1x = lambda x,y: -k1 -kn1 - k3*y
203 f2x = lambda x,y: -2*k2+2*k2*x+2*k2*y-k3*y
204 f1y = lambda x,y: -k1-k3*x
205 f2y = lambda x,y: -2*k2+2*k2*x+2*k2*y-k3*x
206 print '#### Problem 3, Linear stability analysis ####'
207 for (x,y) in ICs_ss:
208     jacobian = np.array([[f1x(x,y), f1y(x,y)],
209                          [f2x(x,y), f2y(x,y)]])
210     print ''
```

```
211     print 'At (', x, ',', y, '), the Jacobian is:'
212     print jacobian
213     [a,b,c,d] = jacobian.flatten().tolist()
214     coeffs = [1, -d-a, a*d-c*b]
215     eigv = np.roots(coeffs)
216     print 'Eigenvalues are:', eigv
217
218
219
```