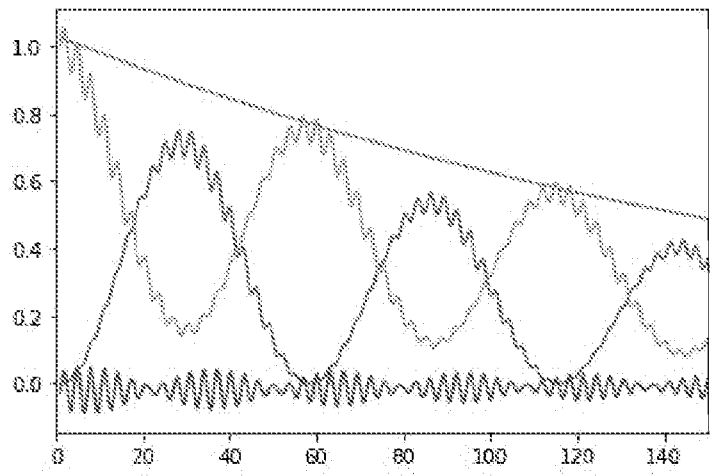
In [185]:
```
E_potential_x1 = 2*0.5*k1*df1.x1**2
E_kinetic_x1 = 2*0.5*m1*df1.v1**2
E_total_x1_abs = E_potential_x1 + E_kinetic_x1
#exp_x1 = 1*np.exp(-c*2*0.5*allt/m1)

E_potential_x2 = 2*0.5*k1*df1.x2**2
E_kinetic_x2 = 2*0.5*m1*df1.v2**2
E_total_x2_abs = E_potential_x2 + E_kinetic_x2

norm_factor = 1/(E_total_x1_abs[0]+E_total_x2_abs[0])

E_total_x1 = E_total_x1_abs*norm_factor
E_total_x2 = E_total_x2_abs*norm_factor

plt.plot(allt,E_total_x1)
plt.plot(allt,E_total_x2)

expo = 1.03*np.exp(-c*2*allt/(m1+m2))

E_coupling = (abs(E_total_x1)+abs(E_total_x2))-expo
plt.plot(allt,E_coupling,color="brown")
#plt.xlim(0,10)
plt.plot(allt,expo,color="grey")
plt.xlim(0,150)

graph = pd.DataFrame({'allt': allt, 'E_total_x1': E_total_x1, 'E_total_x2': E_total_x2,
graph.to_csv("graph2.csv", index=False)
```



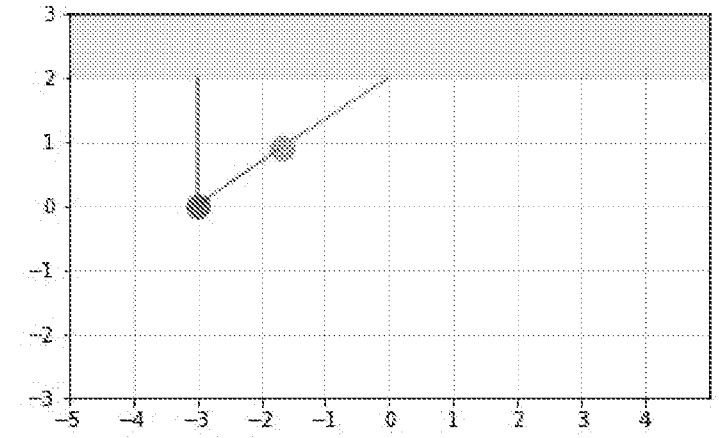
In [45]:
```
# create vector of position function up to respective point for the second animated gra
data6,data7,data8,data9 = ([],[],[],[])
for index,t in enumerate(allt): # step through every time step

    if index % 5 == 0: # take only every nth element
        e1_sofar = (E_total_x1)[:index+1]
        e2_sofar = (E_total_x2)[:index+1]
        ec_sofar = E_coupling[:index+1]
        exp_sofar = expo[:index+1]
        time_sofar = allt[:index+1]
        data6.append([time_sofar,e1_sofar])
        data7.append([time_sofar,e2_sofar])
        data8.append([time_sofar,ec_sofar])
        data9.append([time_sofar,exp_sofar])
```

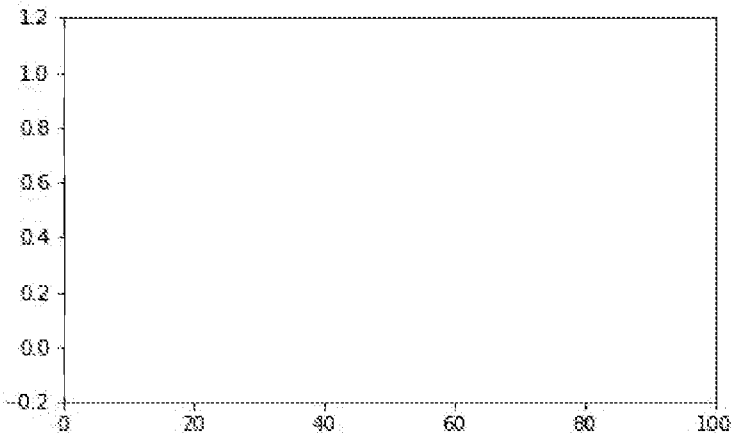
In [46]:
```
# call the animation function
```

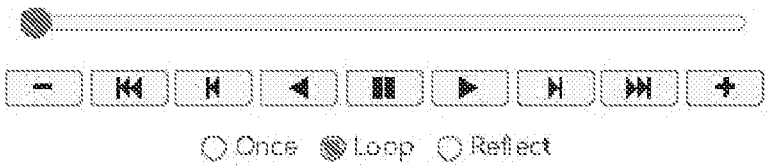
```
subplot1_data = [np.asarray(data1),np.asarray(data2),np.asarray(data3),np.asarray(data4
subplot2_data = [np.asarray(data6),np.asarray(data7),np.asarray(data8),np.asarray(data9

thisfig = create_fig2(ymin=-0.2)
add_pendulum_patches(thisfig)

lines2A[3].set_marker('') # first spring
lines2A[4].set_marker('') # second spring
lines2A[3].set_color('brown') # first spring
lines2A[4].set_color('brown') # second spring
lines2A[3].set_lw(1) # first spring
lines2A[4].set_lw(0) # second spring
lines2A[2].set_marker('') # third pendulum
lines2A[2].set_lw(0) # third pendulum

lines2B[2].set_color('brown') # coupling
lines2B[3].set_color('grey') # expo

ani = animation.FuncAnimation(thisfig,animate2,frames=200,interval=100,fargs=(subplot1_
rc('animation', html='jshtml')
ani.save('double_energy.gif', writer='imagemagick', fps=15) #ani.save('double_energy.mp
ani
```

```
C:\Users\X1\anaconda3\lib\site-packages\numpy\core\_asarray.py:83: VisibleDeprecationWar
ning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of list
s-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to
do this, you must specify 'dtype=object' when creating the ndarray
  return array(a, dtype, copy=False, order=order)
```

Out[46]:



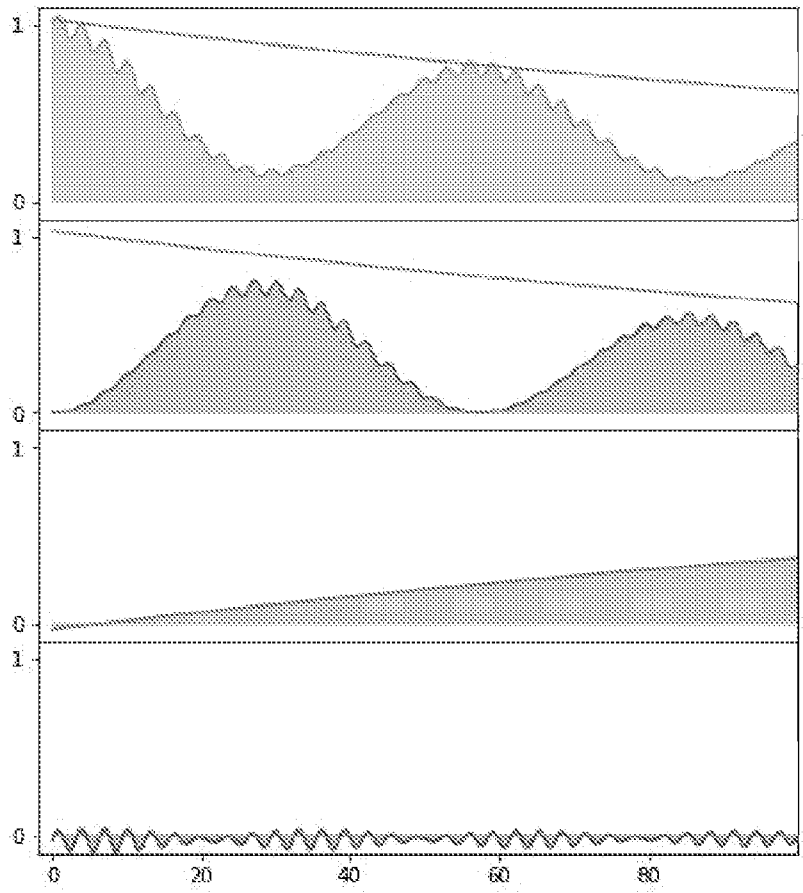




Once Loop Reflect

In [47]:
```
f = plt.figure(figsize=(7,8))

ax1 = plt.subplot(411)
ax2 = plt.subplot(412, sharex = ax1)
ax3 = plt.subplot(413, sharex = ax1)
ax4 = plt.subplot(414, sharex = ax1)

ax1.plot(allt,E_total_x2[:2500],color="C{}".format(1))
ax1.fill_between(allt, E_total_x2[:2500],color="C{}".format(1), alpha=0.5)
ax1.plot(allt,expo,color="grey")
ax1.set_ylim([-.1, 1.1])
ax1.set_xlim([-2,100])
ax1.set_yticks([0,1])

ax2.plot(allt,E_total_x1[:2500],color="C{}".format(0))
ax2.fill_between(allt, E_total_x1[:2500],color="C{}".format(0), alpha=0.5)
ax2.plot(allt,expo,color="grey")
ax2.set_ylim([-.1, 1.1])
```

```
ax2.set_xlim([-2,100])
ax2.set_yticks([0,1])

ax3.plot(allt,1-expo,color="grey")
ax3.fill_between(allt,1-expo,color="grey", alpha=0.5)
ax3.set_ylim([-.1, 1.1])
ax3.set_xticks([0,20,40,60,80])
ax3.set_yticks([0,1])

ax4.plot(allt,E_coupling[:2500],color="brown")
ax4.fill_between(allt, E_coupling[:2500],color="brown", alpha=0.5)
ax4.set_ylim([-.1, 1.1])
ax4.set_xlim([-2,100])
ax4.set_yticks([0,1])

plt.subplots_adjust(wspace=0, hspace=0)
```



4.4.2. Strong coupling induced midway

4.4.2.1. Part 1: Start with one excited pendulum and weak coupling

This example illustrates a system where coupling from an excited subsystem to other subsystems is initially weak but is enhanced after some time through external intervention. In the case of nuclei in a lattice with some nuclear excited states, such intervention can be stimulation of the systems with coherent photons or coherent phonons. Such coherent photons or phonons can lead to strong, superradiantly enhanced couplings between nearby nuclei which then lead to changes in the energy transfer pathways and the de-excitation and excitation parameters of the subsystems as illustrated.

In [48]:
```
offset2 = 10000
len(allt)
```

Out[48]: 2500

In [49]:
```
k1 = k2 = k3 = 1
m1 = 1
m2 = 1
m3 = 1

L1 = 2
L2 = 2 # this is only for looks
L3 = 2

c = 0.005 # DAMPING PARAMETER
kcA = 0.001 # COUPLING A
kcB = 0.000 # COUPLING B

x1i = -1 #initial displacement m1
v1i = 0 #initial velocity m1
x2i = 0 #initial displacement m2
v2i = 0 #initial velocity m2
x3i = 0 #initial displacement m3
v3i = 0 #initial velocity m3
```

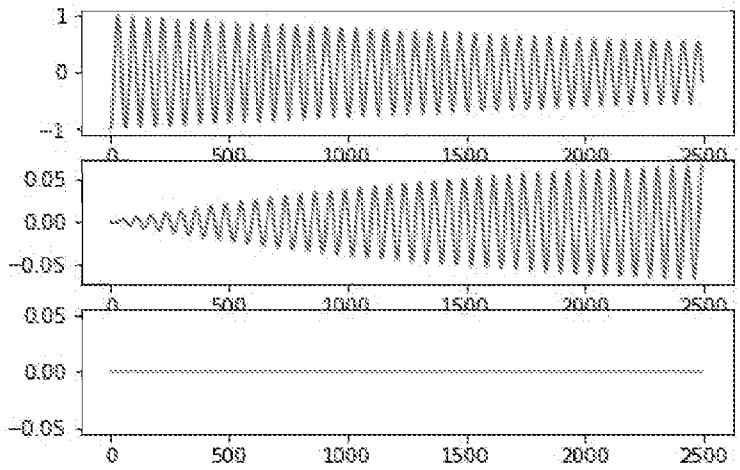
In [50]:
```
thisstate = np.array([x1i,v1i,x2i,v2i,x3i,v3i])
thisstate
```

Out[50]: array([-1, 0, 0, 0, 0, 0])

In [51]:
```
allstates = []
for index,t in enumerate(allt): # step through every time step

    allstates.append(thisstate)
    nextstate = get_next_state_3CD(thisstate, dt)
    thisstate = nextstate
```

In [52]:
```
df1 = pd.DataFrame(np.row_stack(allstates), columns = ['x1', 'v1', 'x2', 'v2', 'x3', 'v
df1.insert(0, "time", allt, True)
df1 = add_polar_coordinates(df1)

plt.subplot(311)
plt.plot(df1.x1)
plt.subplot(312)
plt.plot(df1.x2)
plt.subplot(313)
plt.plot(df1.x3)
```

Out[52]: [<matplotlib.lines.Line2D at 0x29b7b8f7d00>]



4.4.2.2. Part 2: Switch to stronger coupling

In [53]:
```
offset1 = 430 # at this index of the first part, we switch to the second
```

In [54]:
```
df1.iloc[offset1]
```

Out[54]:
```
time    43.000000
x1      -0.493215
v1      -0.725141
x2       0.016278
v2      -0.010921
x3       0.000000
v3       0.000000
sin1    -0.946920
cos1     1.761630
sin2     0.032555
cos2     1.999735
sin3     0.000000
cos3     2.000000
Name: 430, dtype: float64
```

In [55]:
```
k1 = k2 = k3 = 1
m1 = 1
m2 = 1
m3 = 1

L1 = 2
L2 = 2 # this is only for looks
L3 = 2

c = 0.005 # DAMPING PARAMETER
kcA = 0.1 # COUPLING A
kcB = 0.0 # COUPLING B

x1i = df1.iloc[offset1].x1 #initial displacement m1
v1i = df1.iloc[offset1].v1 #initial velocity m1
x2i = 0 #initial displacement m2
v2i = 0 #initial velocity m2
x3i = 0 #initial displacement m3
v3i = 0 #initial velocity m3
```

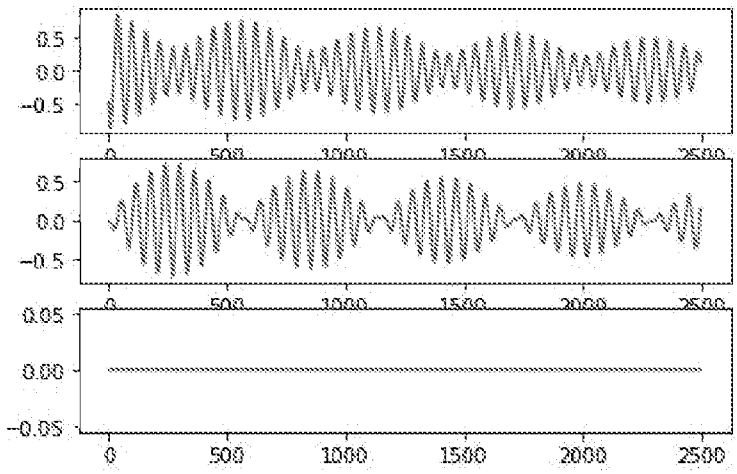
In [56]:
```
thisstate = np.array([x1i,v1i,x2i,v2i,x3i,v3i])
thisstate
```

Out[53]:
```
array([-0.49321499, -0.72514064,  0.        ,  0.        ,  0.        ,
        0.        ])
```

In [57]:
```
allstates = []
for index,t in enumerate(allt): # step through every time step

    allstates.append(thisstate)
    nextstate = get_next_state_3CD(thisstate, dt)
    thisstate = nextstate
```

In [58]:
```
df2 = pd.DataFrame(np.row_stack(allstates), columns = ['x1', 'v1', 'x2', 'v2', 'x3', 'v
df2.insert(0, "time", allt, True)
df2 = add_polar_coordinates(df2)

plt.subplot(311)
plt.plot(df2.x1)
plt.subplot(312)
plt.plot(df2.x2)
plt.subplot(313)
plt.plot(df2.x3)
```

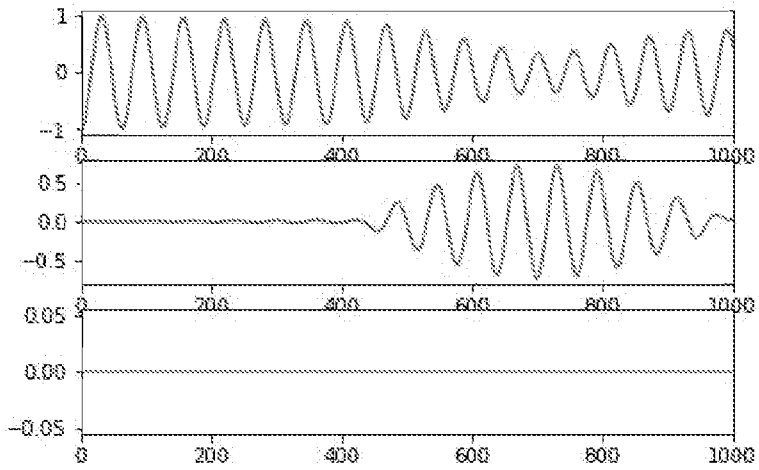
Out[58]:
```
[<matplotlib.lines.Line2D at 0x29b7b9f23d0>]
```



4.4.2.3. Concat the first two parts

In [59]:
```
dfA = pd.concat([df1[:offset1].copy(), df2.copy()], ignore_index=True, sort=False)
allt = np.linspace(1, len(dfA)*0.1, num=len(dfA))

plt.subplot(311)
plt.plot(dfA.x1)
plt.xlim(0,1000)
plt.subplot(312)
plt.plot(dfA.x2)
plt.xlim(0,1000)
plt.subplot(313)
plt.plot(dfA.x3)
plt.xlim(0,1000)
```

Out[59]:
```
(0.0, 1000.0)
```



4.4.2.5. Part 3: Switch back to weak coupling

In [60]:
```
offset2 = 680 # check in the plot above when the switchover should happen
```

In [61]:
```
dfA.iloc[offset2]
```

Out[61]:
```
time    25.000000
x1      -0.295782
v1       0.281635
x2       0.359753
v2      -0.669400
x3       0.000000
v3       0.000000
sin1    -0.582880
cos1     1.913178
sin2     0.704086
cos2     1.871968
sin3     0.000000
cos3     2.000000
Name: 680, dtype: float64
```

In [62]:
```
k1 = k2 = k3 = 1
m1 = 1
m2 = 1
m3 = 1

L1 = 2
L2 = 2 # this is only for looks
L3 = 2

c = 0.005 # DAMPING PARAMETER
kcA = 0.001 # COUPLING A
kcB = 0.000 # COUPLING B

x1i = dfA.iloc[offset2].x1 #initial displacement m1
v1i = dfA.iloc[offset2].v1 #initial velocity m1
x2i = dfA.iloc[offset2].x2 #initial displacement m2
v2i = dfA.iloc[offset2].v2 #initial velocity m2
x3i = 0 #initial displacement m3
v3i = 0 #initial velocity m3
```

In [63]:
```
cfirst = c # for plotting energy plots later
```

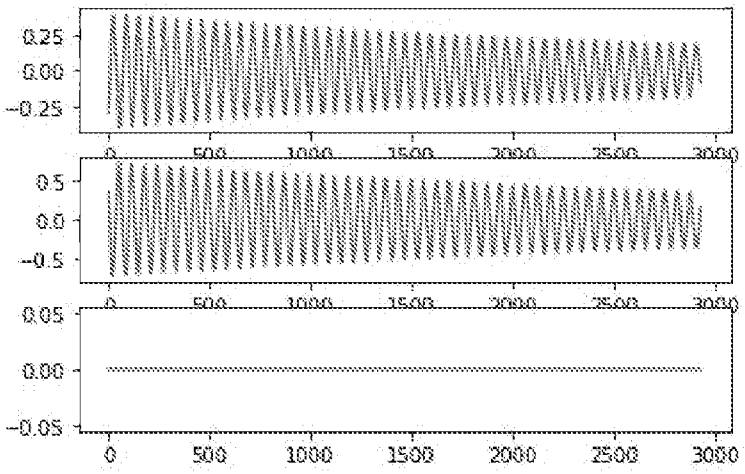
In [64]:
```
thisstate = np.array([x1i,v1i,x2i,v2i,x3i,v3i])
thisstate
```

Out[64]:
```
array([-0.29573197,  0.28163486,  0.35975273, -0.66940019,  0.        ,
        0.        ])
```

In [65]:
```
allstates = []
for index,t in enumerate(allt): # step through every time step

    allstates.append(thisstate)
    nextstate = get_next_state_3CD(thisstate, dt)
    thisstate = nextstate
```

In [66]:
```
df3 = pd.DataFrame(np.row_stack(allstates), columns = ['x1', 'v1', 'x2', 'v2', 'x3', 'v
df3.insert(0, "time", allt, True)
df3 = add_polar_coordinates(df3)

plt.subplot(311)
plt.plot(df3.x1)
plt.subplot(312)
plt.plot(df3.x2)
plt.subplot(313)
plt.plot(df3.x3)
```

Out[66]:
```
[<matplotlib.lines.Line2D at 0x29b7bbc18e0>]
```



In [67]:
```
k1 = k2 = k3 = 1
m1 = 1
m2 = 1
m3 = 1

L1 = 2
L2 = 2 # this is only for looks
L3 = 2

c = 0.25 # DAMPING PARAMETER
kcA = 0.001 # COUPLING A
kcB = 0.000 # COUPLING B

x1i = dfA.iloc[offset2].x1 #initial displacement m1
v1i = dfA.iloc[offset2].v1 #initial velocity m1
x2i = dfA.iloc[offset2].x2 #initial displacement m2
v2i = dfA.iloc[offset2].v2 #initial velocity m2
```

```
x3i = 0 #initial displacement m3
v3i = 0 #initial velocity m3
```

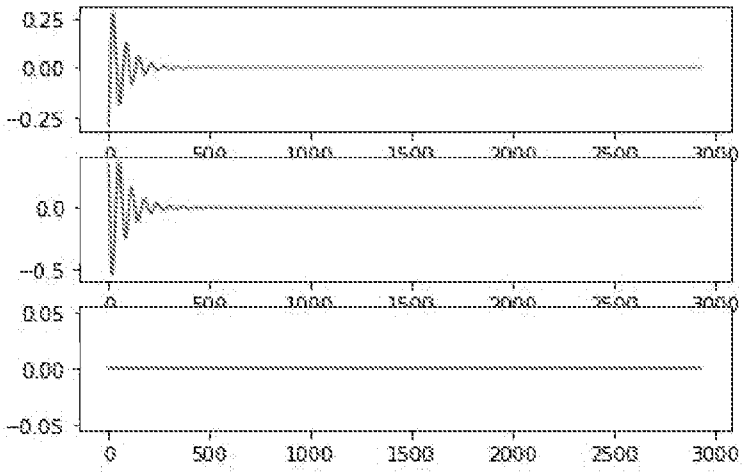
In [68]:
```
thisstate = np.array([x1i,v1i,x2i,v2i,x3i,v3i])
thisstate
```

Out[68]:
```
array([-0.29573197,  0.28163486,  0.35975273, -0.66940019,  0.        ,
        0.        ])
```

In [69]:
```
allstates = []
for index,t in enumerate(allt): # step through every time step

    allstates.append(thisstate)
    nextstate = get_next_state_3CD(thisstate, dt)
    thisstate = nextstate
```

In [70]:
```
df4 = pd.DataFrame(np.row_stack(allstates), columns = ['x1', 'v1', 'x2', 'v2', 'x3', 'v
df4.insert(0, "time", allt, True)
df4 = add_polar_coordinates(df4)

plt.subplot(311)
plt.plot(df4.x1)
plt.subplot(312)
plt.plot(df4.x2)
plt.subplot(313)
plt.plot(df4.x3)
```

Out[70]:
```
[<matplotlib.lines.Line2D at 0x29b7ba99a60>]
```



In [71]:
```
df
```

Out[71]:

	time	x1	v1	x2	v2	x3	v3	sin1	cos1	sin2	cos
0	0.0	0.000000	0.000000	-1.000000	0.000000	0.0	0.0	0.000000	2.000000	-1.682942	1.08060
1	0.1	0.000000	-0.010000	-1.000000	0.100000	0.0	0.0	0.000000	2.000000	-1.682942	1.08060
2	0.2	-0.001000	-0.019765	-0.990000	0.198940	0.0	0.0	-0.002000	1.999999	-1.672052	1.09738
3	0.3	-0.002979	-0.029149	-0.970106	0.295821	0.0	0.0	-0.005957	1.999991	-1.649891	1.13042
4	0.4	-0.005893	-0.037891	-0.940524	0.389667	0.0	0.0	-0.011787	1.999965	-1.615734	1.17873
...	...	...	...	...	...	...	...	...	...	...	...

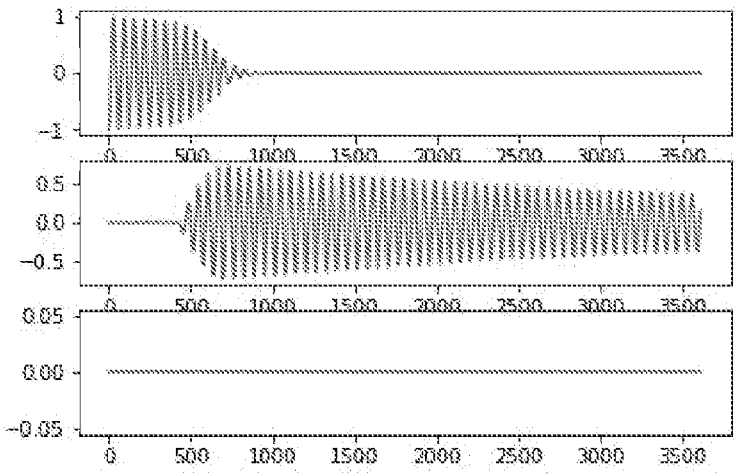
	time	x1	v1	x2	v2	x3	v3	sin1	cos1	sin2	cos
2495	249.5	0.336716	0.254664	0.321369	-0.099541	0.0	0.0	0.660778	1.887690	0.631731	1.89760
2496	249.6	0.362182	0.217811	0.311415	-0.127011	0.0	0.0	0.708631	1.870252	0.612811	1.90380
2497	249.7	0.383963	0.178453	0.298714	-0.152979	0.0	0.0	0.749196	1.854375	0.588582	1.91143
2498	249.8	0.401808	0.136999	0.283416	-0.177226	0.0	0.0	0.782167	1.840711	0.559274	1.92021
2499	249.9	0.415508	0.093882	0.265693	-0.199552	0.0	0.0	0.807310	1.829822	0.525156	1.92982

2500 rows × 13 columns

4.5.2.6. Add the last part

In [72]:
```
df_all = pd.concat([dfA[:(offset2)],df3], ignore_index=True, sort=False)
allt = np.linspace(1, len(df_all)*0.1, num=len(df_all))

df3.x1 = df4.x1
df3.sin1 = df4.sin1
df3.cos1 = df4.cos1
df3.v1 = df4.v1

df_all_ani = pd.concat([dfA[:(offset2)],df3], ignore_index=True, sort=False)
allt = np.linspace(1, len(df_all_ani)*0.1, num=len(df_all_ani))

plt.subplot(311)
plt.plot(df_all_ani.x1)
plt.subplot(312)
plt.plot(df_all_ani.x2)
plt.subplot(313)
plt.plot(df_all_ani.x3)
```

Out[72]: [<matplotlib.lines.Line2D at 0x29b797a4ee0>]



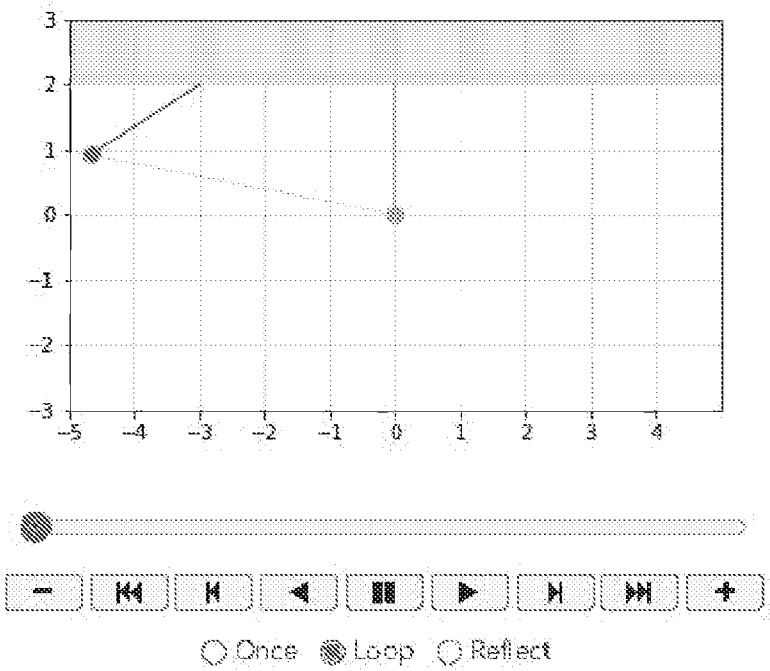
In [73]:
```
df = df_all_ani

# create position vectors for the first animated graph
data1,data2,data3,data4,data5 = ([],[],[],[],[])
for index,t in enumerate(allt): # step through every time step
```

```
    if index % 5 == 0: # take only every nth element
        data1.append([[-3*df.sin1[index],-3],[2-df.cos1[index],2]]) # first number: x o
        data2.append([[0*df.sin2[index],0],[2-df.cos2[index],2]]) # first number: x anc
        data3.append([[3*df.sin3[index],3],[2-df.cos3[index],2]]) # first number: x anc
        data4.append([[0*df.sin2[index],-3*df.sin1[index]],[2-df.cos2[index],2-df.cos1[
        data5.append([[3*df.sin3[index],0*df.sin2[index]],[2-df.cos3[index],2-df.cos2[i
```

In [74]:
```
offset3 = 681 #800
offset2 = 10000
```

In [75]:
```
# call the animation function
subplot1_data = [[np.asarray(data1),np.asarray(data2),np.asarray(data3),np.asarray(data

thisfig = create_fig1()
add_pendulum_patches(thisfig)

lines1A[0].set_markersize(8)
lines1A[1].set_markersize(8)
lines1A[2].set_markersize(8)
lines1A[3].set_marker('') # first spring
lines1A[4].set_marker('') # second spring
lines1A[3].set_color('brown') # first spring
lines1A[4].set_color('brown') # second spring
lines1A[3].set_lw(0.2) # first spring
lines1A[4].set_lw(0.2) # second spring
lines1A[2].set_marker('') # third pendulum
lines1A[2].set_lw(0) # third pendulum
lines1A[4].set_alpha(0) # second spring

ani = animation.FuncAnimation(thisfig,animate1special,frames=250,interval=100,fargs=(su
rc('animation', html='jshtml')
ani
```

Out[75]:



4.5.1.8. Add position information

In [76]:
```
# create vector of position function up to respective point for the second animated gra
```

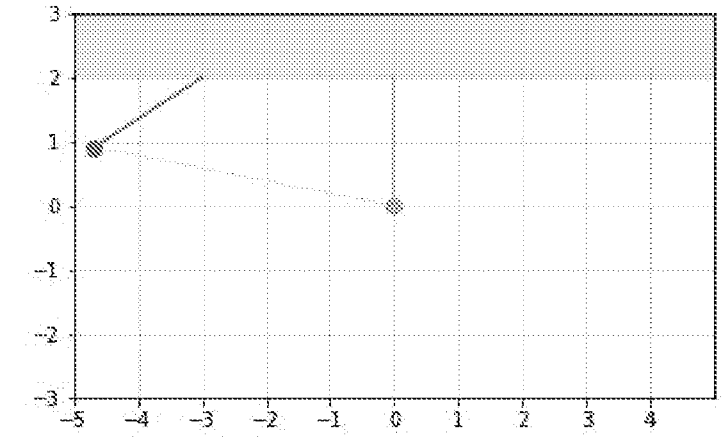
```
data6,data7,data8 = ([],[],[])
for index,t in enumerate(allt): # step through every time step

    if index % 5 == 0: # take only every nth element
        position1_sofar = df.x1[:index+1]
        position2_sofar = df.x2[:index+1]
        time_sofar = allt[:index+1]
        data6.append([time_sofar,position1_sofar])
        data7.append([time_sofar,position2_sofar])
```

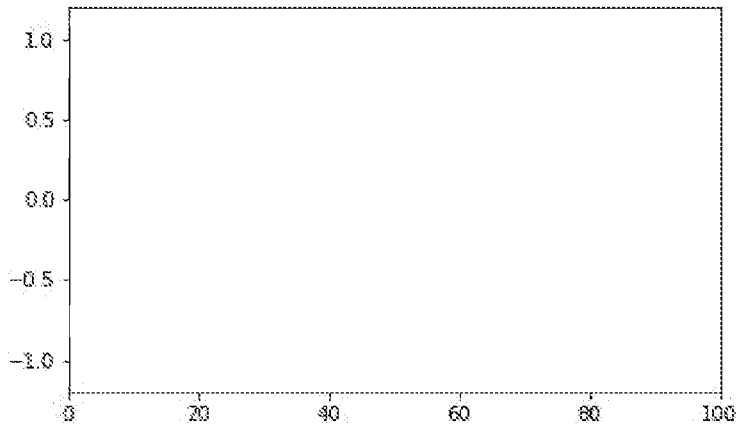
In [77]:

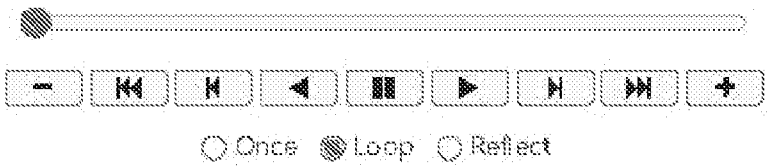
```
# call the animation function
subplot1_data = [np.asarray(data1),np.asarray(data2),np.asarray(data3),np.asarray(data4
subplot2_data = [np.asarray(data6),np.asarray(data7)]

thisfig = create_fig2()
add_pendulum_patches(thisfig)

lines2A[0].set_markersize(0)
lines2A[1].set_markersize(0)
lines2A[2].set_markersize(0)
lines2A[3].set_marker('') # first spring
lines2A[4].set_marker('') # second spring
lines2A[3].set_color('brown') # first spring
lines2A[4].set_color('brown') # second spring
lines2A[3].set_lw(0.2) # first spring
lines2A[4].set_lw(0.2) # second spring
lines2A[2].set_marker('') # third pendulum
lines2A[2].set_lw(0) # third pendulum
lines2A[4].set_alpha(0) # second spring

ani = animation.FuncAnimation(thisfig,animate2special,frames=200,interval=100,fargs=(su
rc('animation', html='jshtml')
ani.save('qubit_pos.gif', writer='imagemagick', fps=15) #ani.save('qubit_pos.mp4', writ
ani
```

```
C:\Users\X1\anaconda3\lib\site-packages\numpy\core\_asarray.py:83: VisibleDeprecationWar
ning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of list
s-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to
do this, you must specify 'dtype=object' when creating the ndarray
  return array(a, dtype, copy=False, order=order)
```

Out[77]:





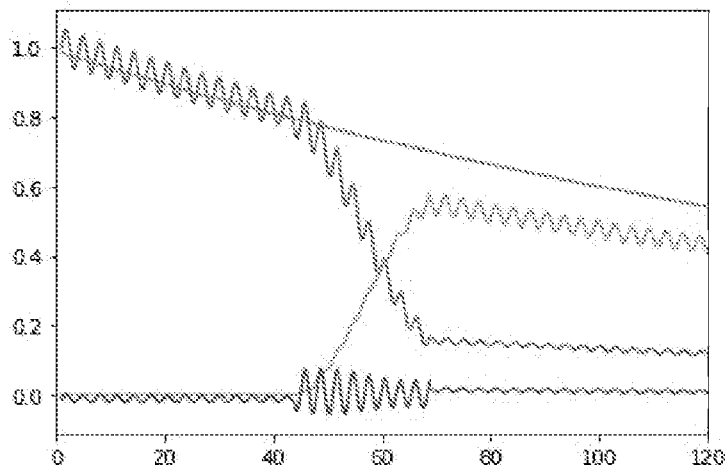


Once Loop Reflect

In [76]:
```
df = df_all

E_potential_x1 = 2*0.5*k1*df.x1**2
E_kinetic_x1 = 2*0.5*m1*df.v1**2
E_total_x1_abs = E_potential_x1 + E_kinetic_x1

E_potential_x2 = 2*0.5*k1*df.x2**2
E_kinetic_x2 = 2*0.5*m1*df.v2**2
E_total_x2_abs = E_potential_x2 + E_kinetic_x2

norm_factor = 1/(E_total_x1_abs[0]+E_total_x2_abs[0])

E_total_x1 = E_total_x1_abs*norm_factor
E_total_x2 = E_total_x2_abs*norm_factor

plt.plot(allt,E_total_x1)
plt.plot(allt,E_total_x2)
plt.xlim(0,120)
```

```
expo = 0.93*np.exp(-cfirst*2*allt/(m1+m2))
plt.plot(allt,expo,color="grey")
#expo2 = 0.8*np.exp(-cfirst*20*allt/(m1+m2))
#plt.plot(allt,expo2,color="grey")
expo3 = 0.8*np.exp(-cfirst*2*allt/(m1+m2))

E_coupling = (abs(E_total_x1)+abs(E_total_x2))-0.07
E_coupling[:offset1] = (E_coupling[:offset1]-expo[:offset1])/5
E_coupling[offset1:offset3] = (E_coupling[offset1:offset3]-(expo3[offset1:offset3]+expo
E_coupling[offset3:] = (E_coupling[offset3:]-expo3[offset3:])/5
plt.plot(allt,E_coupling,color="brown")

#expo2 = 0.80*np.exp(-cfirst*25*(allt-48)/m2)-0.030
#plt.plot(allt[(offset1+40):offset2],expo2[(offset1+40):offset2],color="grey")
```

Out[78]: [<matplotlib.lines.Line2D at 0x29b7c99b850>]



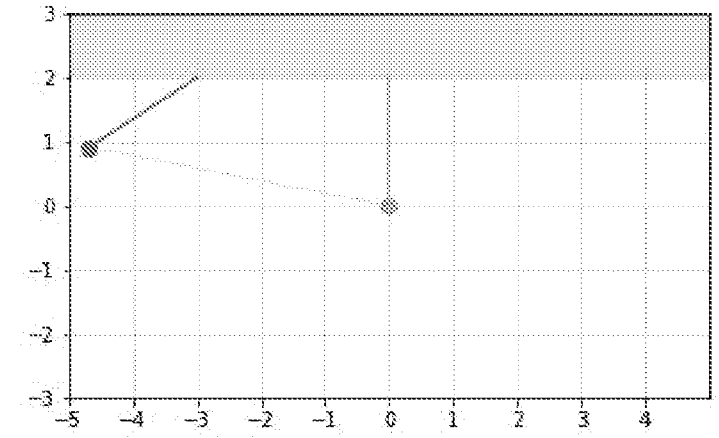
In [79]:
```
# create vector of position function up to respective point for the second animated gra
data6,data7,data8,data9,data10 = ([],[],[],[],[])
for index,t in enumerate(allt): # step through every time step

    if index % 5 == 0: # take only every nth element
        e1_sofar = (E_total_x1)[:index+1]
        e2_sofar = (E_total_x2)[:index+1]
        ec_sofar = E_coupling[:index+1]
        exp_sofar = expo[:index+1]
        time_sofar = allt[:index+1]
        data6.append([time_sofar,e1_sofar])
        data7.append([time_sofar,e2_sofar])
        data8.append([time_sofar,ec_sofar])
        data9.append([time_sofar,exp_sofar])
```

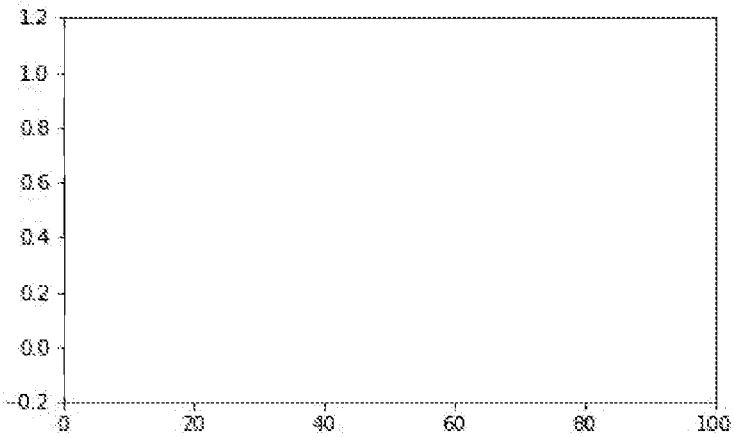
In [80]:
```
# call the animation function
subplot1_data = [np.asarray(data1),np.asarray(data2),np.asarray(data3),np.asarray(data4
subplot2_data = [np.asarray(data6),np.asarray(data7),np.asarray(data8),np.asarray(data9

thisfig = create_fig2(ymin=-0.2)
add_pendulum_patches(thisfig)

lines2A[0].set_markersize(8)
lines2A[1].set_markersize(8)
lines2A[2].set_markersize(8)
```

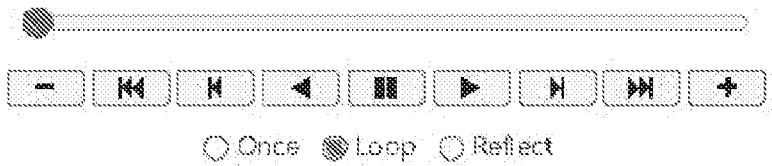
```
lines2A[3].set_marker('') # first spring
lines2A[4].set_marker('') # second spring
lines2A[3].set_color('brown') # first spring
lines2A[4].set_color('brown') # second spring
lines2A[3].set_lw(0.2) # first spring
lines2A[4].set_lw(0.2) # second spring
lines2A[2].set_marker('') # third pendulum
lines2A[2].set_lw(0) # third pendulum
lines2A[4].set_alpha(0) # second spring

lines2B[2].set_color('brown') # coupling
lines2B[3].set_color('grey') # expo

ani = animation.FuncAnimation(thisfig,animate2special,frames=200,interval=100,fargs=(su
rc('animation', html='jshtml')
ani.save('qubit_energy.gif', writer='imagemagick', fps=15) #ani.save('qubit_energy.mp4'
ani
```

```
C:\Users\X1\anaconda3\lib\site-packages\numpy\core\_asarray.py:83: VisibleDeprecationWar
ning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of list
s-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to
do this, you must specify 'dtype=object' when creating the ndarray
  return array(a, dtype, copy=False, order=order)
```

Out[88]:



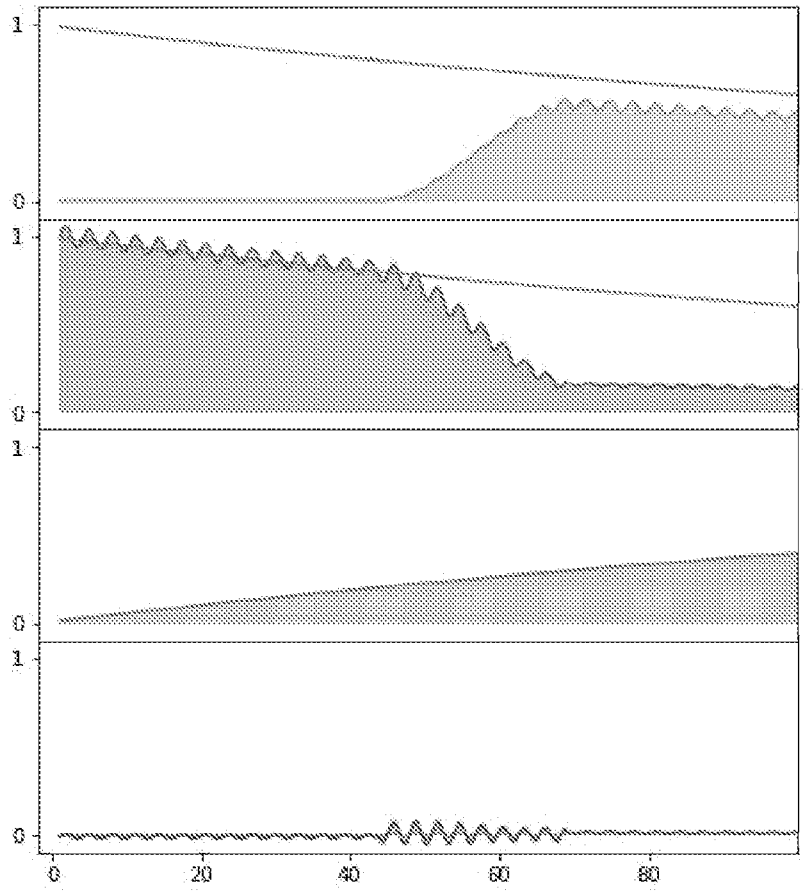




Once Loop Reflect

In [61]:
```
f = plt.figure(figsize=(7,8))

ax1 = plt.subplot(411)
ax2 = plt.subplot(412, sharex = ax1)
ax3 = plt.subplot(413, sharex = ax1)
ax4 = plt.subplot(414, sharex = ax1)

ax1.plot(allt,E_total_x2,color="C{}".format(1))
ax1.fill_between(allt, E_total_x2,color="C{}".format(1), alpha=0.5)
ax1.plot(allt,expo,color="grey")
#ax1.plot(allt[offset1+40:],expo2[offset1+40:],color="grey")
ax1.set_ylim([-.1, 1.1])
ax1.set_xlim([-2,100])
ax1.set_yticks([0,1])

ax2.plot(allt,E_total_x1,color="C{}".format(0))
ax2.fill_between(allt, E_total_x1,color="C{}".format(0), alpha=0.5)
ax2.plot(allt,expo,color="grey")
```

```
ax2.set_ylim([-.1, 1.1])
ax2.set_xlim([-2,100])
ax2.set_yticks([0,1])

expo_final = 1-expo.copy()
#expo_final[offset2:] = np.ones(np.size(expo_final)-(offset2))
ax3.plot(allt,expo_final,color="grey")
ax3.fill_between(allt,expo_final,color="grey", alpha=0.5)
ax3.set_ylim([-.1, 1.1])
ax3.set_xticks([0,20,40,60,80])
ax3.set_yticks([0,1])

ax4.plot(allt,E_coupling,color="brown")
ax4.fill_between(allt, E_coupling,color="brown", alpha=0.5)
ax4.set_ylim([-.1, 1.1])
ax4.set_xlim([-2,100])
ax4.set_yticks([0,1])

plt.subplots_adjust(wspace=0, hspace=0)
```



4.5. Coupled triple of damped oscillators

4.5.1 Upconversion

4.5.1.1. Part 1: Start with two excited pendula and weak coupling

This example illustrates how the de-excitation of two initially excited states can be accelerated when comparatively strongly coupled to an additional quantum state that can readily accomodate the

energy of the initially excited states (i.e. that is resonant). This can be thought of as induced upconversion.

In this example, as in others above and below, the coupling strength is initially weak and is then increased by stimulation e.g. via coherent photons or phonons. The increase in coupling strength then changes the dynamics and faciliates the transfer of excitation.

This model describes processes where multiple de-excitations of smaller quanta collectively drive the upconversion of a smaller number of larger quanta. An example includes multiple fusion reactions in a metal hydride such as D2 --> He-4 or p+d --> He-3 that transfer their excitation to host lattice nuclei such as Pd or Ag to high-energy excited states -- which eventually leads to disintegration/fission of such receiver nuclei, leaving behind fission products.

In [82]:
```
k1 = k2 = k3 = 1
m1 = 1
m2 = 1
m3 = 1

L1 = 2
L2 = 2 # this is only for looks
L3 = 2

c = 0.005 # DAMPING PARAMETER
kcA = 0.001 # COUPLING A
kcB = 0.001 # COUPLING B

x1i = -1 #initial displacement m1
v1i = 0 #initial velocity m1
x2i = 0 #initial displacement m2
v2i = 0 #initial velocity m2
x3i = -0.578495 #0.3 #initial displacement m3
v3i = 0.787039 #0.3 #initial velocity m3
```

In [83]:
```
cfirst = c # for plotting energy plots later
```

In [84]:
```
thisstate = np.array([x1i,v1i,x2i,v2i,x3i,v3i])
thisstate
```

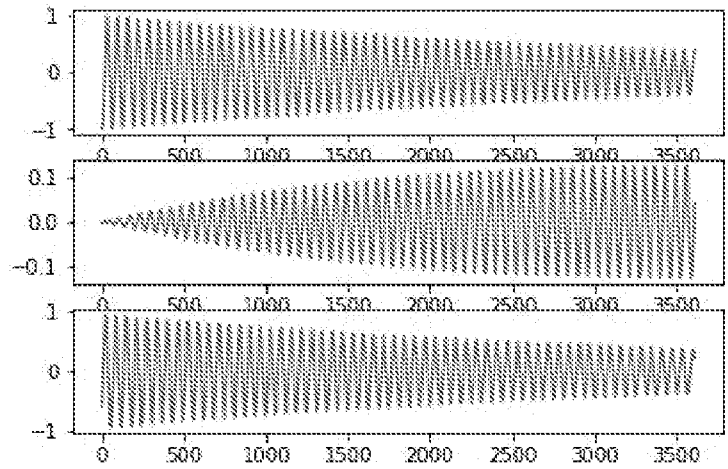
Out[84]: array([-1. , 0. , 0. , 0. , -0.578495, 0.787039])

In [85]:
```
allstates = []
for index,t in enumerate(allt): # step through every time step

    allstates.append(thisstate)
    nextstate = get_next_state_3CD(thisstate, dt)
    thisstate = nextstate
```

In [86]:
```
df1 = pd.DataFrame(np.row_stack(allstates), columns = ['x1', 'v1', 'x2', 'v2', 'x3', 'v
df1.insert(0, "time", allt, True)
df1 = add_polar_coordinates(df1)

plt.subplot(311)
plt.plot(df1.x1)
plt.subplot(312)
plt.plot(df1.x2)
```

```
plt.subplot(313)
plt.plot(df1.x3)
```

Out[86]: [<matplotlib.lines.Line2D at 0x29b00555520>]



4.5.1.2. Part 2: Switch to stronger coupling and phase coherence

In [87]:
```
k1 = k2 = k3 = 1
m1 = 1
m2 = 1
m3 = 1

L1 = 2
L2 = 2 # this is only for locks
L3 = 2

c = 0.005 # DAMPING PARAMETER
kcA = 0.1 # COUPLING A
kcB = 0.1 # COUPLING B

x1i = 1 #initial displacement m1
v1i = 0 #initial velocity m1
x2i = 0 #initial displacement m2
v2i = 0 #initial velocity m2
x3i = 1 #initial displacement m3
v3i = 0 #0.3 #initial velocity m3
```

In [88]:
```
thisstate = np.array([x1i,v1i,x2i,v2i,x3i,v3i])
thisstate
```

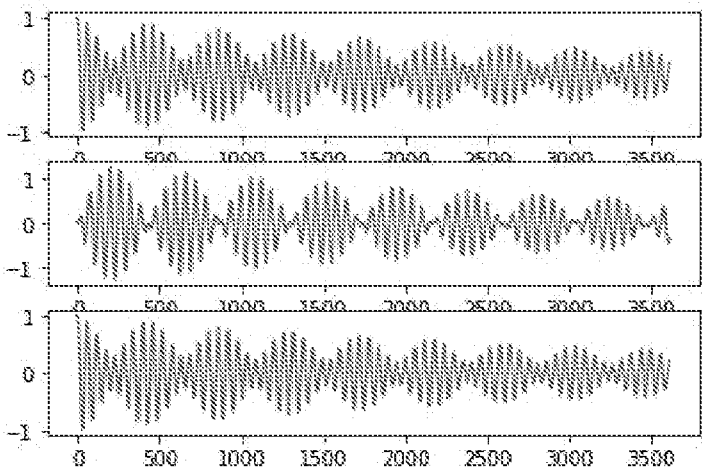
Out[88]: array([1, 0, 0, 0, 1, 0])

In [89]:
```
allstates = []
for index,t in enumerate(allt): # step through every time step

    allstates.append(thisstate)
    nextstate = get_next_state_3CD(thisstate, dt)
    thisstate = nextstate
```

In [90]:
```
df2 = pd.DataFrame(np.row_stack(allstates), columns = ['x1', 'v1', 'x2', 'v2', 'x3', 'v
df2.insert(0, "time", allt, True)
df2 = add_polar_coordinates(df2)
```

```
plt.subplot(311)
plt.plot(df2.x1)
plt.subplot(312)
plt.plot(df2.x2)
plt.subplot(313)
plt.plot(df2.x3)
```

Out[88]: [<matplotlib.lines.Line2D at 0x29b0063dd00>]

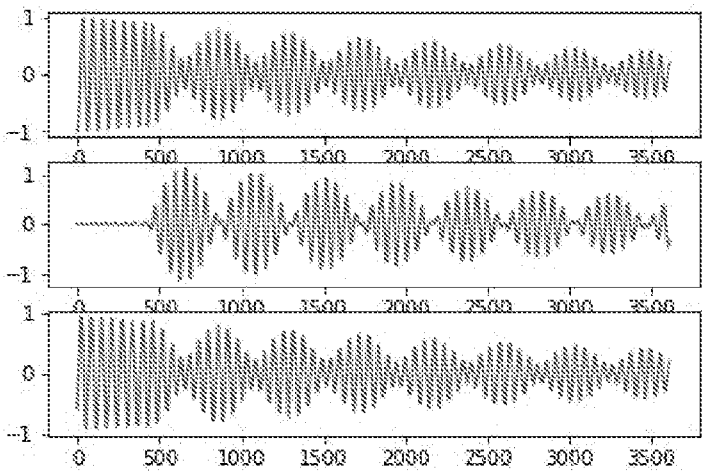


4.5.1.3. Concat the first two parts

In [91]:
```
offset1 = 430 # at this index of the first part, we switch to the second
offset3 = 10000
```

In [92]:
```
dfA = pd.concat([df1[:offset1].copy(), df2[(offset1+1):].copy()], ignore_index=True, so
allt = np.linspace(1, len(dfA)*0.1, num=len(dfA))

plt.subplot(311)
plt.plot(dfA.x1)
plt.subplot(312)
plt.plot(dfA.x2)
plt.subplot(313)
plt.plot(dfA.x3)
```

Out[92]: [<matplotlib.lines.Line2D at 0x29b00de41f0>]



4.5.1.5. Part 3: Let the remaining oscillators idle out

In [93]:
```
offset2 = 650 # check in the plot above when the switchover should happen
```

In [94]:
```
dfA.iloc[offset2]
```

Out[94]:
```
time    65.937656
x1      -0.172134
v1      -0.229769
x2       0.747975
v2       0.847724
x3      -0.172134
v3      -0.229769
sin1    -0.342570
cos1     1.970443
sin2     1.360311
cos2     1.466136
sin3    -0.342570
cos3     1.970443
Name: 650, dtype: float64
```

In [95]:
```
k1 = k2 = k3 = 1
m1 = 1
m2 = 1
m3 = 1

L1 = 2
L2 = 2 # this is only for looks
L3 = 2

c = 0.15 # DAMPING PARAMETER
kcA = 0.0 # COUPLING A
kcB = 0.0 # COUPLING B

x1i = dfA.iloc[offset2].x1 #initial displacement m1
v1i = dfA.iloc[offset2].v1 #initial velocity m1
x2i = 0 #initial displacement m2
v2i = 0 #initial velocity m2
x3i = dfA.iloc[offset2].x3 #initial displacement m3
v3i = dfA.iloc[offset2].v3 #initial velocity m3
```

In [96]:
```
thisstate = np.array([x1i,v1i,x2i,v2i,x3i,v3i])
thisstate
```

Out[96]:
```
array([-0.17213404, -0.22976942,  0.        ,  0.        , -0.17213404,
       -0.22976942])
```

In [97]:
```
allstates = []
for index,t in enumerate(allt): # step through every time step

    allstates.append(thisstate)
    nextstate = get_next_state_3CD(thisstate, dt)
    thisstate = nextstate
```

In [98]:
```
df3 = pd.DataFrame(np.row_stack(allstates), columns = ['x1', 'v1', 'x2', 'v2', 'x3', 'v
df3.insert(0, "time", allt, True)
df3 = add_polar_coordinates(df3)

plt.subplot(311)
plt.plot(df3.x1)
plt.subplot(312)
plt.plot(df3.x2)
```