question6

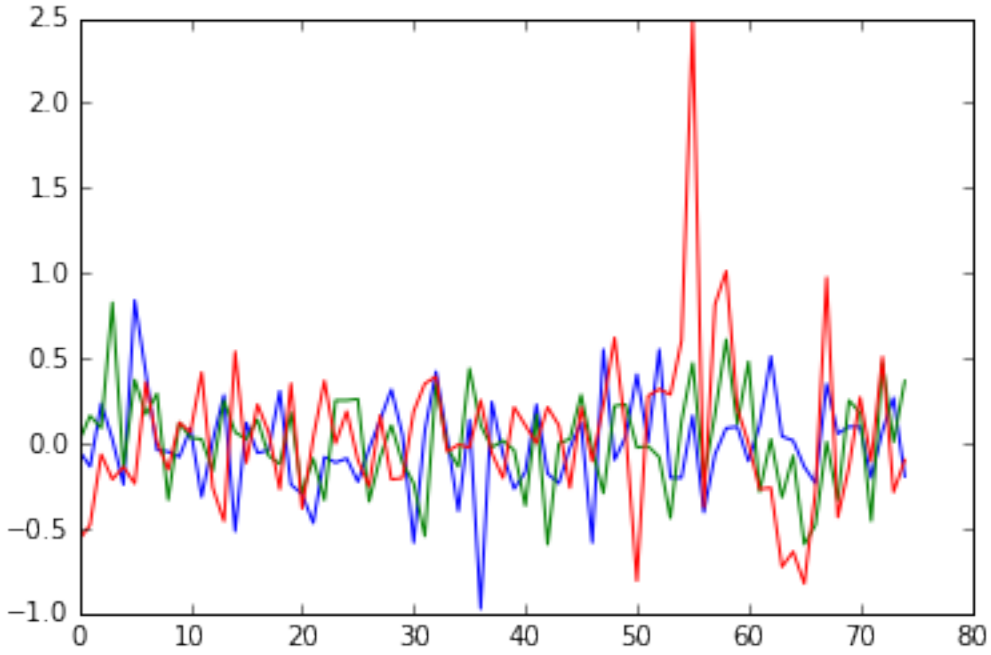
January 15, 2016

In [1]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

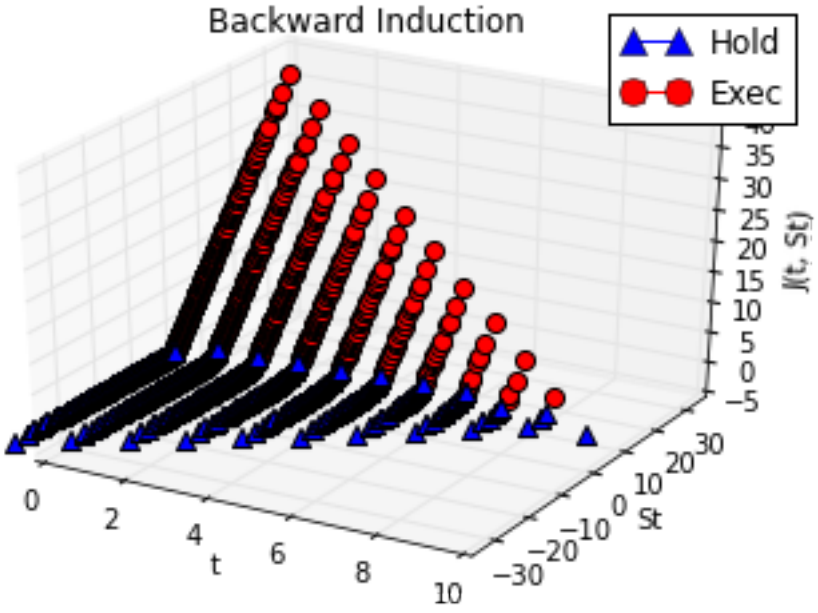
In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:
```
# Load microsoft data
msft = pd.read_csv('microsoft.csv')
trajectories = np.diff(msft['MSFT.Adjusted'])
length_short_path = 30 # days
length = len(trajectories)
num_paths = int(length / length_short_path)
trajectories = trajectories[:length_short_path * num_paths]
trajectories.shape = (length_short_path, num_paths)
for i in range(3):
    plt.plot(trajectories[i, :])
plt.show()

K = 0.
T = 5
```



```
In [4]: outcomes = np.sort(np.sum(trajectories[:5,:], axis=1))
        print(outcomes)
        num_outcomes = len(outcomes)

[-4.404346 -0.829189 -0.79187   1.067787  3.531883]

In [5]: def generate_slates(n):
            St = np.array([0])
            S = [St]
            for i in range(n):
                Stnext = np.array([])
                for ds in outcomes:
                    Stnext = np.concatenate([Stnext, (St + ds)])
                St = unique(Stnext)
                St = np.sort(St)
                S.append(St)
            return S[::-1]

        def backward_induction(St, Jnext):
            expected_J = np.zeros_like(St)
            offset = len(Jnext) - len(St)
            p = 1. / offset
            for i in range(offset):
                expected_J += p* (Jnext[i:-(offset - i)] if offset !=  i else Jnext[i:])
            buffer = np.array([expected_J, St-K])
            control = np.argmax(buffer, axis=0)
            J = np.maximum(St-K, expected_J)
            return (J, control)

        def price_tree(n):
            slates = generate_slates(n)
            J = np.zeros(len(slates[0])+num_outcomes-1)
            for t, St in enumerate(slates):
                J, control = backward_induction(St, J)
                yield np.array([t*np.ones_like(St), St, J, control])

In [7]: from mpl_toolkits.mplot3d import Axes3D
        import matplotlib.pyplot as plt
        import matplotlib.lines as mlines

        T = 10
        def plot(pricing_method, name, img):
            fig = plt.figure()
            ax = fig.add_subplot(111, projection='3d')
            c = np.array(('b', 'r'))
            m = np.array(('^', 'o'))
            for points in pricing_method(T):
                for u in (0, 1):
                    xs, ys, z_tree, _ = points[:,abs(points[3] - u) <= 0.01]
                    ax.scatter(xs,
                           ys, z_tree, s=50,
                           c=c[u], marker=m[u],
```

```
                    depthshade=False)

    ax.set_xlabel('t'); ax.set_xlim((0, T))
    ax.set_ylabel('St'); ax.set_ylim((-T * outcomes[-1], T * outcomes[-1]))
    ax.set_zlabel('J(t, St)')
    labels_str = ('Hold', 'Exec')
    labels = [
        mlines.Line2D([], [], color=c[u], marker=m[u],
                      markersize=10, label=labels_str[u])
        for u in (0, 1)]
    plt.title(name)
    plt.legend(handles=labels)
    plt.savefig('q%s.png' % img)
    plt.show()
plot(price_tree, 'Backward Induction', 'tree')
```




```
In [0]: #LP

        # Get the relevent matrices
        def get_A():
            row = np.zeros(T*T)
            for i in range(num_outcomes):
                row[T+i] = 1./num_oucomes
            A = [row]
            for _ in range(T*(T-1)-1):
                row = np.roll(row, 1)
                A.append(row)
            row = 0 * row
            for _ in range(T): A.append(row)
            return np.array(A)
```

```
def get_B():
    B = np.zeros( shape=(T*(T-1)/2, T*T) )
    i, j = 0, 0
    for St in generate_slates(T)[::-1]:
        Nt = len(St)
        j += Nt + 1
        for b in range(T-(Nt+1)):
            B[i, j] = 1
            i, j = i+1, j+1
    return B
def get_U():
    return np.array(itertools.chain(*generate_slate(T))) - K
```