BotInfluence2

August 9, 2021

1 Further Analysis of Bot+Strategic Network

1.0.1 Attempt at endogeneously computing r_{ss}

James Yu

8 August 2021

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

[2]:
```
def l_matrix(r_ss, A_tilde, B_tilde, w_0):
    A_tilde_prime = np.concatenate((np.concatenate((A_tilde, B_tilde), axis =␣
 ↪1), # A c
                    np.concatenate((np.zeros((1, 6)), np.array([1], ndmin =␣
 ↪2)), axis = 1)), # 0 1
                    axis = 0)
    w_0_prime = np.concatenate((w_0, np.array([r_ss], ndmin = 2)), axis = 0)
    res_mat = np.linalg.matrix_power(A_tilde_prime, 1000000000000)
    res_vec = res_mat @ w_0_prime
    return res_mat, res_vec
```

[3]:
```
def optimal_K_dynamic(A, c, B, x, z, delta = 0.8, tol = 10**(-18), R = 0):
    A_tilde = np.concatenate((np.concatenate((A, c), axis = 1), # A c
        np.concatenate((np.zeros((1, 5)), np.array([1], ndmin = 2)), axis =␣
 ↪1)), # 0 1
        axis = 0)
    B_tilde = np.concatenate((B, np.array([0], ndmin = 2)), axis = 0)
    w_0 = np.concatenate((x, np.array([z], ndmin = 2)), axis = 0)
    Q = 0.2 * np.identity(5)
    Q_tilde = 0.2 * np.identity(6)
    Q_tilde[5, :] = 0

    def L(K_entry):
        return -1 * np.linalg.inv(B_tilde.T @ K_entry @ B_tilde + np.array(R,␣
 ↪ndmin = 2)/delta) @ B_tilde.T @ K_entry @ A_tilde

    # first compute the sequence of optimal K_t matrices
    K = np.zeros((6, 6))
    K_t = [Q_tilde, K]
    K = Q_tilde
    current_difference = np.inf
    while abs(current_difference) > tol: # to avoid floating point error, don't␣
↪converge all the way to zero (this is standard)
        K_new = delta * (A_tilde.T @ (K
                - (K @ B_tilde @ np.linalg.inv(B_tilde.T @ K @ B_tilde + np.
↪array(R, ndmin = 2)/delta) @ B_tilde.T @ K))
                @ A_tilde) + Q_tilde
        K_t.insert(0, K_new)
        current_difference = np.max(np.abs(K - K_new))
        K = K_new

    # compute the Gamma matrix to use for later computations
    expr = A_tilde + B_tilde @ L(K_t[0])
    A_tilde_n = expr[:5, :5]
    c_nplus1 = np.array(expr[:5, 5], ndmin = 2).T
    x_t = x
    x_ts = [x]

    # compute the resulting sequence of x_t opinion vectors
    for K_ent in K_t:
        x_tp1 = A_tilde_n @ x_t + c_nplus1 * z
        x_ts.append(x_tp1)
        x_t = x_tp1

    # compute the sequence of r_t and cumulative costs
    payoff = 0
    payoffs = []
    r_ts = []
    i = 0
    for x_ent in x_ts:
        r = L(K_t[0]) @ np.concatenate((x_ent, np.array([z], ndmin = 2)), axis␣
↪= 0)
        r_ts.append(r)
        payoff += (-1 * delta**i * ((x_ent.T @ Q @ x_ent).item() + (r * R *␣
↪r))) # account for discounting
        payoffs.append(payoff)
        i += 1

    return r_ts, A_tilde, B_tilde, w_0, K_t, x_ts, payoffs
```

1.0.2 First: a quick check of the Part 2 of Corollary 2 given in the note:

[4]:
```
A = np.array([
      [0.1, 0.2, 0.2, 0.2, 0.2],
      [0.2, 0.1, 0.2, 0.2, 0.2],
      [0.2, 0.2, 0.1, 0.2, 0.2],
      [0.2, 0.2, 0.2, 0.1, 0.2],
      [0.2, 0.2, 0.2, 0.2, 0.1],
    ], ndmin = 2)
c = np.array([0.04, 0.04, 0.04, 0.04, 0.04,], ndmin = 2).T
B = np.array([0.06, 0.06, 0.06, 0.06, 0.06], ndmin = 2).T
x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, x, 1)
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
print(np.round(mat, 7))
```

```
[[0.  0.  0.  0.  0.  0.4 0.6]
 [0.  0.  0.  0.  0.  0.4 0.6]
 [0.  0.  0.  0.  0.  0.4 0.6]
 [0.  0.  0.  0.  0.  0.4 0.6]
 [0.  0.  0.  0.  0.  0.4 0.6]
 [0.  0.  0.  0.  0.  1.  0. ]
 [0.  0.  0.  0.  0.  0.  1. ]]
```

[5]:
```
A = np.array([
      [0.1, 0.2, 0.2, 0.2, 0.2],
      [0.1, 0.2, 0.2, 0.2, 0.2],
      [0.1, 0.2, 0.2, 0.2, 0.2],
      [0.1, 0.2, 0.2, 0.2, 0.2],
      [0.1, 0.2, 0.2, 0.2, 0.2],
    ], ndmin = 2)
c = np.array([0.04, 0.04, 0.04, 0.04, 0.04,], ndmin = 2).T
B = np.array([0.06, 0.06, 0.06, 0.06, 0.06], ndmin = 2).T
x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, x, 1)
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
print(np.round(mat, 7))
```

```
[[0.  0.  0.  0.  0.  0.4 0.6]
 [0.  0.  0.  0.  0.  0.4 0.6]
 [0.  0.  0.  0.  0.  0.4 0.6]
 [0.  0.  0.  0.  0.  0.4 0.6]
 [0.  0.  0.  0.  0.  0.4 0.6]
 [0.  0.  0.  0.  0.  1.  0. ]
 [0.  0.  0.  0.  0.  0.  1. ]]
```

[6]:
```
A = np.array([
      [0.1, 0.2, 0.2, 0.2, 0.2],
      [0.1, 0.1, 0.3, 0.2, 0.2],
      [0.1, 0, 0.2, 0.4, 0.2],
      [0.1, 0.2, 0.2, 0.2, 0.2],
      [0.1, 0.2, 0.2, 0.2, 0.2],
   ], ndmin = 2)
c = np.array([0.04, 0.04, 0.04, 0.04, 0.04,], ndmin = 2).T
B = np.array([0.06, 0.06, 0.06, 0.06, 0.06], ndmin = 2).T
x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, x, 1, tol␣
 ↪= 10**(-10))
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
print(np.round(mat, 7))
```

```
[[0.  0.  0.  0.  0.  0.4 0.6]
 [0.  0.  0.  0.  0.  0.4 0.6]
 [0.  0.  0.  0.  0.  0.4 0.6]
 [0.  0.  0.  0.  0.  0.4 0.6]
 [0.  0.  0.  0.  0.  0.4 0.6]
 [0.  0.  0.  0.  0.  1.  0. ]
 [0.  0.  0.  0.  0.  0.  1. ]]
```

The statement is true on its own, and perhaps can be expanded to say that any matrix with identical row sums is sufficient.

[7]:
```
A
```

[7]:
```
array([[0.1, 0.2, 0.2, 0.2, 0.2],
       [0.1, 0.1, 0.3, 0.2, 0.2],
       [0.1, 0. , 0.2, 0.4, 0.2],
       [0.1, 0.2, 0.2, 0.2, 0.2],
       [0.1, 0.2, 0.2, 0.2, 0.2]])
```

[8]:
```
i = sum(A[0])
i
```

[8]:
```
0.8999999999999999
```

[9]:
```
A @ A @ A
```

[9]:
```
array([[0.081, 0.11 , 0.174, 0.202, 0.162],
       [0.081, 0.113, 0.173, 0.2  , 0.162],
       [0.081, 0.112, 0.176, 0.198, 0.162],
       [0.081, 0.11 , 0.174, 0.202, 0.162],
       [0.081, 0.11 , 0.174, 0.202, 0.162]])
```

[10]:
```
round(sum((A @ A @ A)[0]), 14)
```

[10]:
```
0.729
```

[11]:
```
print(round(i**3, 14))
```

```
0.729
```

(there should exist a theoretical proof that if the row sums of A are all the same, regardless of structure, the sums of A^2 are also the same)

1.0.3 Statics for r_{ss}

Experiment 0: Vary δ in the original model to see what happens Baseline: $\delta = 0.8$:

[12]:
```
A = np.array([
      [0.217,    0.2022,   0.2358,   0.1256,   0.1403],
      [0.8988*0.2497,   0.8988*0.0107,   0.8988*0.2334,   0.8988*0.1282,   0.
 ↪8988*0.378],
      [0.1285,   0.0907,   0.3185,   0.2507,   0.2116],
      [0.1975,   0.0629,   0.2863,   0.2396,   0.2137],
      [0.1256,   0.0711,   0.0253,   0.2244,   0.5536],
    ], ndmin = 2)
c = np.array([0, 0.1012, 0, 0, 0,], ndmin = 2).T
B = np.array([0.0791, 0, 0, 0, 0,], ndmin = 2).T
x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, x, 10)
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
print(np.round(mat, 7))
print(rs[-1].item())
```

```
[[0.         0.         0.         0.         0.         0.381708   0.618292 ]
 [0.         0.         0.         0.         0.         0.4632063  0.5367937]
 [0.         0.         0.         0.         0.         0.4102641  0.5897359]
 [0.         0.         0.         0.         0.         0.406857   0.593143 ]
 [0.         0.         0.         0.         0.         0.4089491  0.5910509]
 [0.         0.         0.         0.         0.         1.         0.       ]
 [0.         0.         0.         0.         0.         0.         1.       ]]
-6.838450103205228
```

Alternate: $\delta = 0.5$:

[13]:
```
A = np.array([
      [0.217,    0.2022,   0.2358,   0.1256,   0.1403],
      [0.8988*0.2497,   0.8988*0.0107,   0.8988*0.2334,   0.8988*0.1282,   0.
 ↪8988*0.378],
      [0.1285,   0.0907,   0.3185,   0.2507,   0.2116],
      [0.1975,   0.0629,   0.2863,   0.2396,   0.2137],
      [0.1256,   0.0711,   0.0253,   0.2244,   0.5536],
    ], ndmin = 2)
c = np.array([0, 0.1012, 0, 0, 0,], ndmin = 2).T
B = np.array([0.0791, 0, 0, 0, 0,], ndmin = 2).T
x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, x, 10,␣
 ↪delta = 0.5, tol = 10**(-14))
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
```

```
print(np.round(mat, 7))
print(rs[-1].item())
```

```
[[0.         0.         0.         0.         0.         0.381708   0.618292 ]
 [0.         0.         0.         0.         0.         0.4632063 0.5367937]
 [0.         0.         0.         0.         0.         0.4102641 0.5897359]
 [0.         0.         0.         0.         0.         0.406857   0.593143 ]
 [0.         0.         0.         0.         0.         0.4089491 0.5910509]
 [0.         0.         0.         0.         0.         1.         0.        ]
 [0.         0.         0.         0.         0.         0.         1.        ]]
-6.576584701504723
```

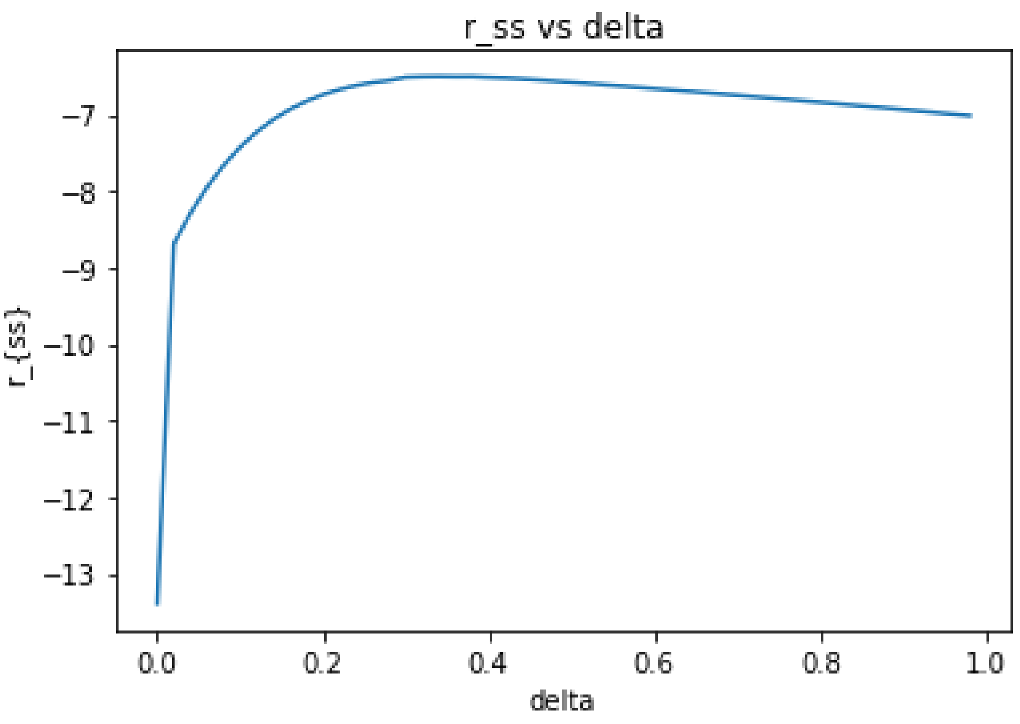
Alternate: $\delta = 0.1$:

[14]:
```
A = np.array([
      [0.217,    0.2022,    0.2358,    0.1256,    0.1403],
      [0.8988*0.2497,    0.8988*0.0107,    0.8988*0.2334,    0.8988*0.1282,    0.
 ↪8988*0.378],
      [0.1285,    0.0907,    0.3185,    0.2507,    0.2116],
      [0.1975,    0.0629,    0.2863,    0.2396,    0.2137],
      [0.1256,    0.0711,    0.0253,    0.2244,    0.5536],
    ], ndmin = 2)
c = np.array([0, 0.1012, 0, 0, 0,], ndmin = 2).T
B = np.array([0.0791, 0, 0, 0, 0,], ndmin = 2).T
x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, x, 10,␣
 ↪delta = 0.1, tol = 10**(-14))
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
print(np.round(mat, 7))
print(rs[-1].item())
```

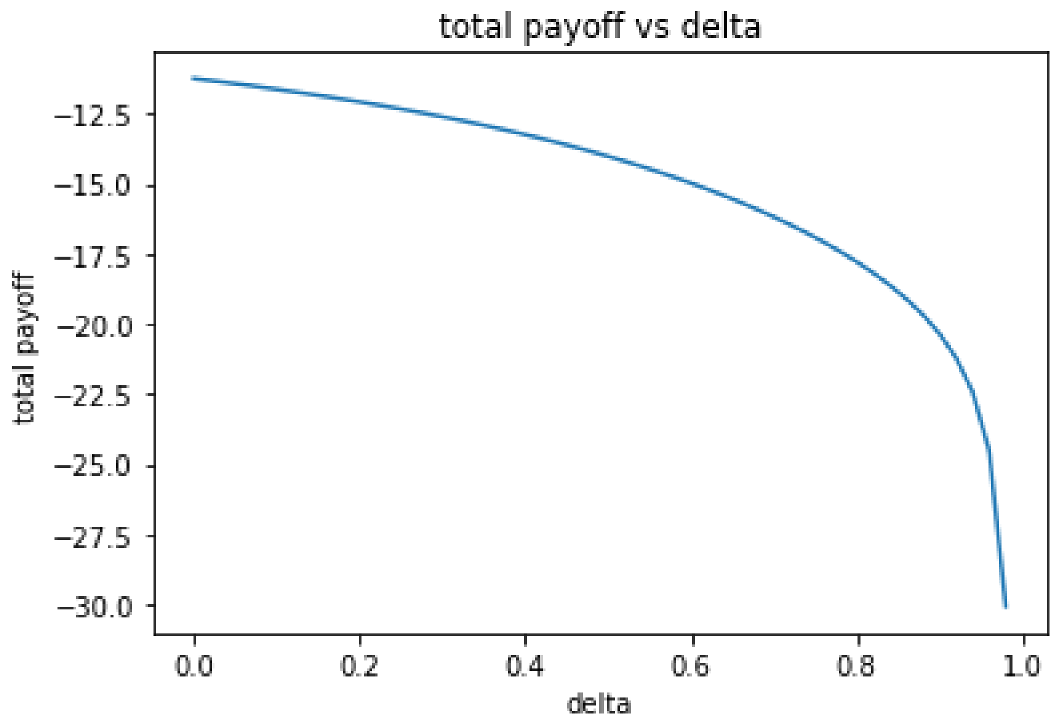
```
[[0.         0.         0.         0.         0.         0.381708   0.618292 ]
 [0.         0.         0.         0.         0.         0.4632063 0.5367937]
 [0.         0.         0.         0.         0.         0.4102641 0.5897359]
 [0.         0.         0.         0.         0.         0.406857   0.593143 ]
 [0.         0.         0.         0.         0.         0.4089491 0.5910509]
 [0.         0.         0.         0.         0.         1.         0.        ]
 [0.         0.         0.         0.         0.         0.         1.        ]]
-7.424942636530235
```

Plot of r_{ss} with respect to δ:

[15]:
```
rsx = []
tsx = []
for delta in np.linspace(0.0000001, 0.98, 50):
    A = np.array([
          [0.217,    0.2022,    0.2358,    0.1256,    0.1403],
          [0.8988*0.2497,    0.8988*0.0107,    0.8988*0.2334,    0.8988*0.1282,  ␣
 ↪0.8988*0.378],
            [0.1285,   0.0907,   0.3185,   0.2507,   0.2116],
            [0.1975,   0.0629,   0.2863,   0.2396,   0.2137],
            [0.1256,   0.0711,   0.0253,   0.2244,   0.5536],
        ], ndmin = 2)
    c = np.array([0, 0.1012, 0, 0, 0,], ndmin = 2).T
    B = np.array([0.0791, 0, 0, 0, 0,], ndmin = 2).T
    x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
    rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, x,␣
↪10, delta = delta, tol = 10**(-14))
    rsx.append(rs[-1].item())
    tsx.append(ps[-1].item())

plt.plot(np.linspace(0.0000001, 0.98, 50), rsx)
plt.title("r_ss vs delta")
plt.xlabel("delta")
plt.ylabel("r_{ss}")
plt.show()
```



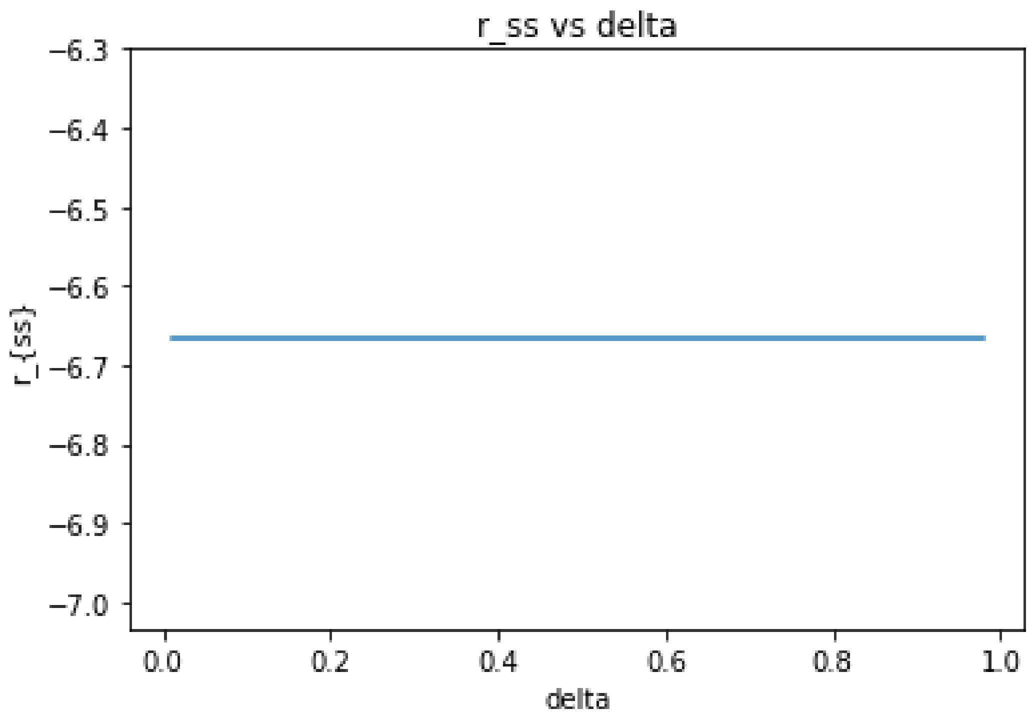
[16]:
```
plt.plot(np.linspace(0.0000001, 0.98, 50), tsx)
plt.title("total payoff vs delta")
plt.xlabel("delta")
plt.ylabel("total payoff")
plt.show()
```



Conclusion: there is a sort of log-like relationship with a diminishing factor. More analysis is needed.

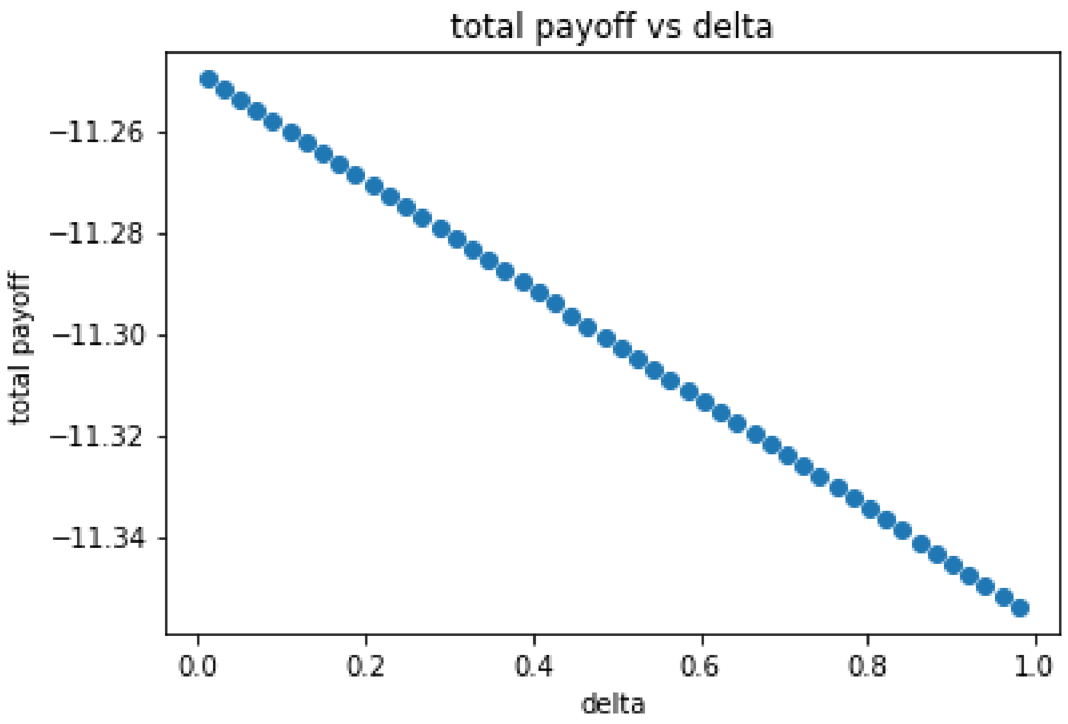
1.0.4 Experiment 1: Fresh model, symmetric, vary δ

[17]:
```
rsx = []
tsx = []
for delta in np.linspace(0.01, 0.98, 50):
    A = np.array([
      [0.1, 0.2, 0.2, 0.2, 0.2],
      [0.2, 0.1, 0.2, 0.2, 0.2],
      [0.2, 0.2, 0.1, 0.2, 0.2],
      [0.2, 0.2, 0.2, 0.1, 0.2],
      [0.2, 0.2, 0.2, 0.2, 0.1],
    ], ndmin = 2)
    c = np.array([0.04, 0.04, 0.04, 0.04, 0.04,], ndmin = 2).T
    B = np.array([0.06, 0.06, 0.06, 0.06, 0.06], ndmin = 2).T
    x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
    rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, x,␣
 ↪10, delta = delta, tol = 10**(-14))
    rsx.append(rs[-1].item())
    tsx.append(ps[-1].item())

plt.plot(np.linspace(0.01, 0.98, 50), rsx)
plt.title("r_ss vs delta")
plt.xlabel("delta")
plt.ylabel("r_{ss}")
plt.show()
```



Now δ has no impact, meaning this is model-dependent.

[18]:
```
plt.scatter(np.linspace(0.01, 0.98, 50), tsx)
plt.title("total payoff vs delta")
plt.xlabel("delta")
plt.ylabel("total payoff")
plt.show()
```



Payoff should vary because it factors directly into the payoff equation, and it does vary to some extent here.

A possible question: does this network allow for a $\delta = 1$ solution?

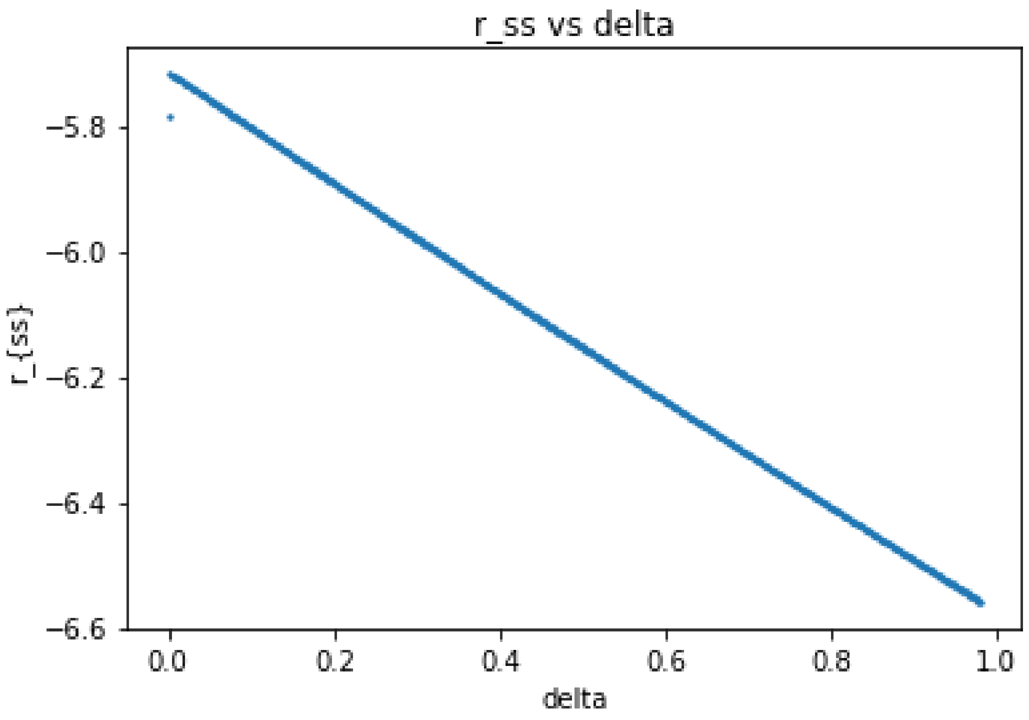
[19]:
```
A = np.array([
    [0.1, 0.2, 0.2, 0.2, 0.2],
    [0.2, 0.1, 0.2, 0.2, 0.2],
    [0.2, 0.2, 0.1, 0.2, 0.2],
    [0.2, 0.2, 0.2, 0.1, 0.2],
    [0.2, 0.2, 0.2, 0.2, 0.1],
], ndmin = 2)
c = np.array([0.04, 0.04, 0.04, 0.04, 0.04,], ndmin = 2).T
B = np.array([0.06, 0.06, 0.06, 0.06, 0.06], ndmin = 2).T
x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, x, 10,␣
 ↪delta = 1, tol = 10**(-14))
rs[-1].item()
```

[19]:
```
-6.666666666666667
```

Yes, it does. Interestingly, r_{ss} is exactly -6 and 2/3.

1.0.5 Experiment 3: Each agent only has one "source" broadcast (less of a special case than Experiment 2), and vary δ

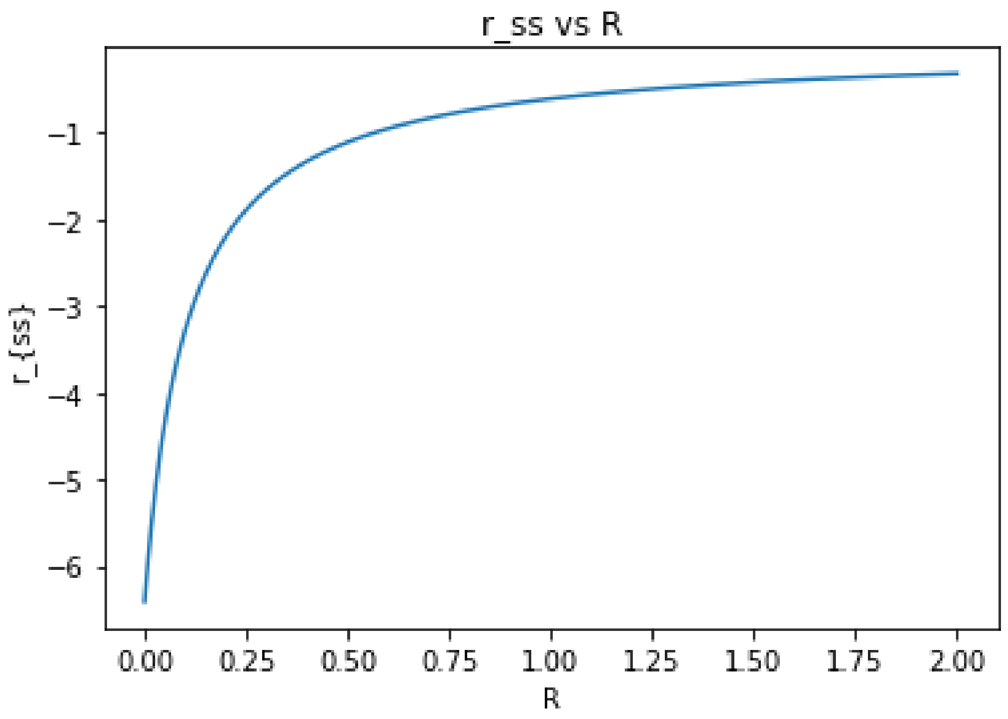
[20]:
```
rsx = []
tsx = []
for delta in np.linspace(0.000001, 0.98, 500):
    A = np.array([
      [0.1, 0.2, 0.2, 0.2, 0.2],
      [0.2, 0.1, 0.2, 0.2, 0.2],
      [0.2, 0.2, 0.1, 0.2, 0.2],
      [0.2, 0.2, 0.2, 0.1, 0.2],
      [0.2, 0.2, 0.2, 0.2, 0.1],
    ], ndmin = 2)
    c = np.array([0.1, 0.1, 0, 0, 0], ndmin = 2).T
    B = np.array([0, 0, 0.1, 0.1, 0.1], ndmin = 2).T
    x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
    rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, x,␣
 ↪10, delta = delta, tol = 10**(-14))
    rsx.append(rs[-1].item())
    tsx.append(ps[-1].item())

plt.scatter(np.linspace(0.000001, 0.98, 500), rsx, s = 2)
plt.title("r_ss vs delta")
plt.xlabel("delta")
plt.ylabel("r_{ss}")
plt.show()
```



This is *almost* linear, but there's a bit close to zero where it is not.

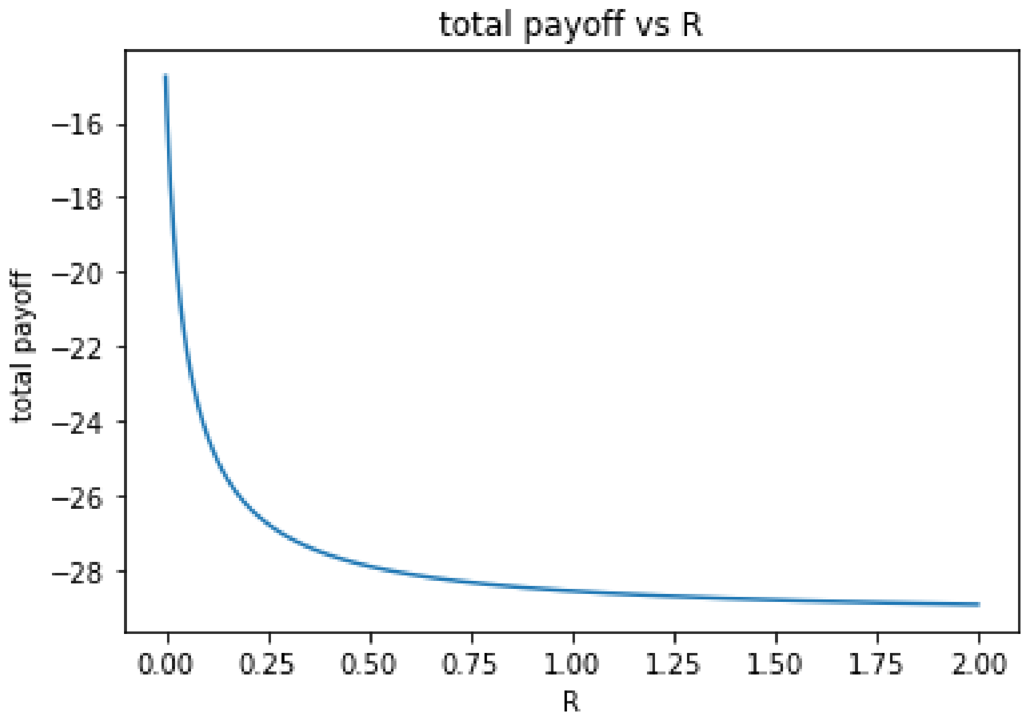
1.0.6 Experiment 4: Take the network in Experiment 3, fix δ at 0.8, and increase R (cost matrix of messages)

[21]:
```
rsx = []
tsx = []
for R in np.linspace(0.000001, 2, 500):
    A = np.array([
      [0.1, 0.2, 0.2, 0.2, 0.2],
      [0.2, 0.1, 0.2, 0.2, 0.2],
      [0.2, 0.2, 0.1, 0.2, 0.2],
      [0.2, 0.2, 0.2, 0.1, 0.2],
      [0.2, 0.2, 0.2, 0.2, 0.1],
    ], ndmin = 2)
    c = np.array([0.1, 0.1, 0, 0, 0], ndmin = 2).T
    B = np.array([0, 0, 0.1, 0.1, 0.1], ndmin = 2).T
    x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
    rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, x,␣
 ↪10, delta = 0.8, tol = 10**(-14), R = R)
    rsx.append(rs[-1].item())
    tsx.append(ps[-1].item())

plt.plot(np.linspace(0.000001, 2, 500), rsx)
plt.title("r_ss vs R")
plt.xlabel("R")
plt.ylabel("r_{ss}")
plt.show()
```



As hypothesized, as R increases, r_{ss} gets closer to zero (this turned out to be a reciprocal function).

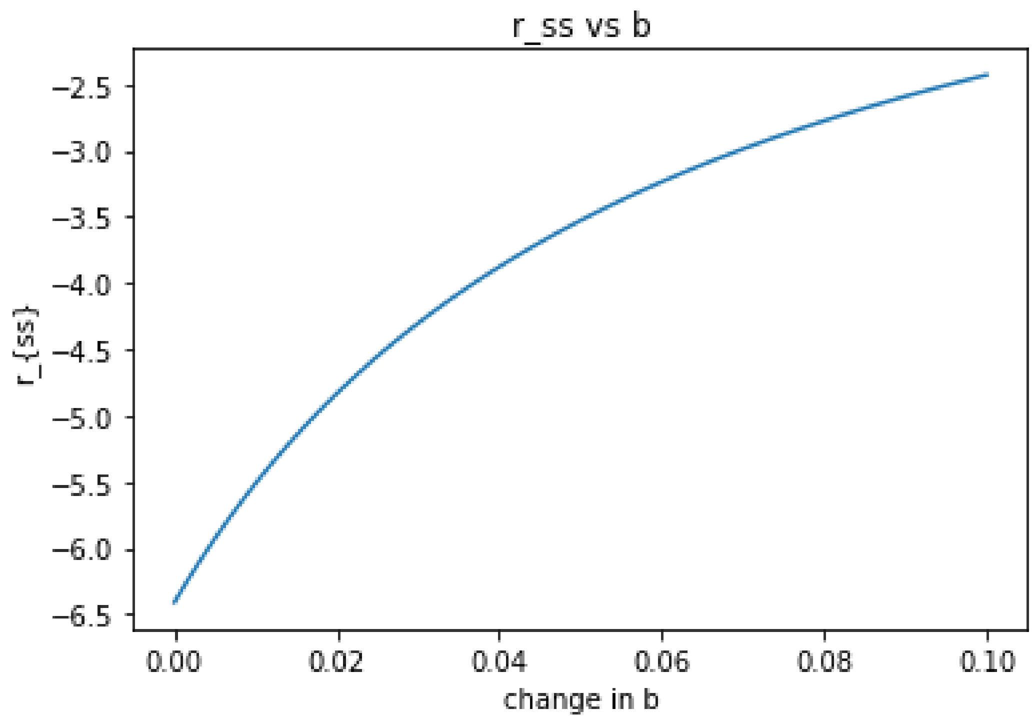
[22]:
```
plt.plot(np.linspace(0.000001, 2, 500), tsx)
plt.title("total payoff vs R")
plt.xlabel("R")
plt.ylabel("total payoff")
plt.show()
```



Payoff is similarly oriented.

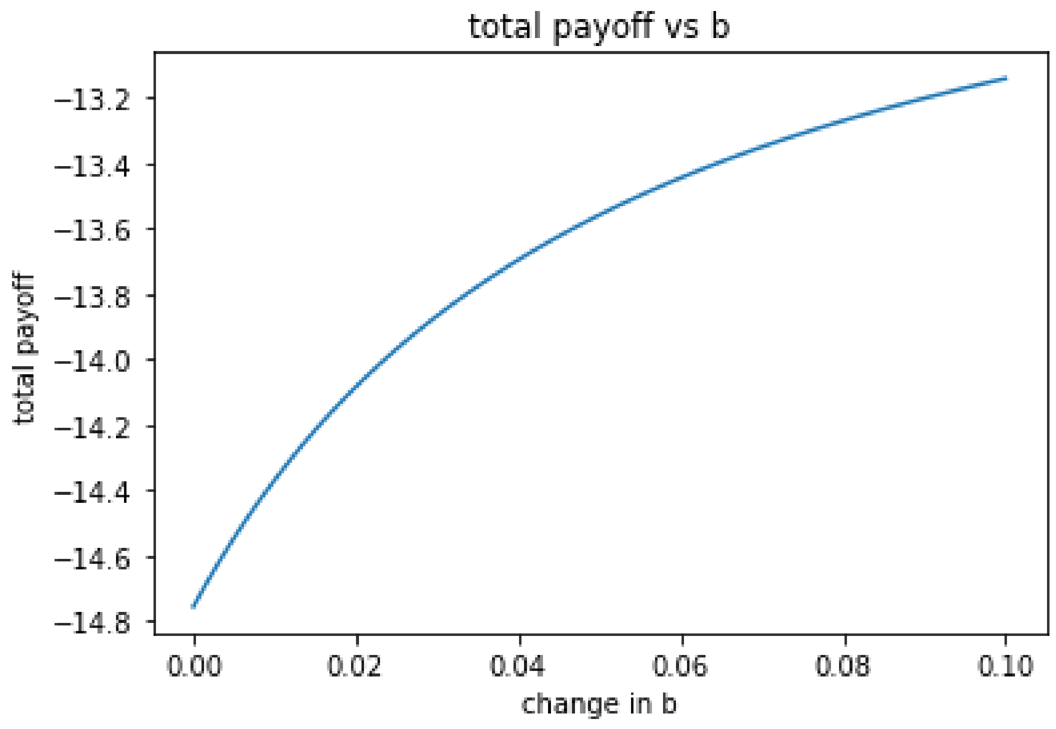
1.0.7 Experiment 5: Raise b uniformly

[23]:
```
rsx = []
tsx = []
for b_delta in np.linspace(0.000001, 0.1, 500):
    A = np.array([
      [0.1 - b_delta/5, 0.2 - b_delta/5, 0.2 - b_delta/5, 0.2 - b_delta/5, 0.2␣
 ↪- b_delta/5],
      [0.2 - b_delta/5, 0.1 - b_delta/5, 0.2 - b_delta/5, 0.2 - b_delta/5, 0.2␣
 ↪- b_delta/5],
      [0.2 - b_delta/5, 0.2 - b_delta/5, 0.1 - b_delta/5, 0.2 - b_delta/5, 0.2␣
 ↪- b_delta/5],
      [0.2 - b_delta/5, 0.2 - b_delta/5, 0.2 - b_delta/5, 0.1 - b_delta/5, 0.2␣
 ↪- b_delta/5],
      [0.2 - b_delta/5, 0.2 - b_delta/5, 0.2 - b_delta/5, 0.2 - b_delta/5, 0.1␣
 ↪- b_delta/5],
    ], ndmin = 2)
    c = np.array([0.1, 0.1, 0, 0, 0], ndmin = 2).T
    B = np.array([0 + b_delta, 0 + b_delta, 0.1 + b_delta, 0.1 + b_delta, 0.1 +␣
 ↪b_delta], ndmin = 2).T
    x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
    rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, x,␣
↪10, delta = 0.8, tol = 10**(-14))
    rsx.append(rs[-1].item())
    tsx.append(ps[-1].item())

plt.plot(np.linspace(0.000001, 0.1, 500), rsx)
plt.title("r_ss vs b")
plt.xlabel("change in b")
plt.ylabel("r_{ss}")
plt.show()
```



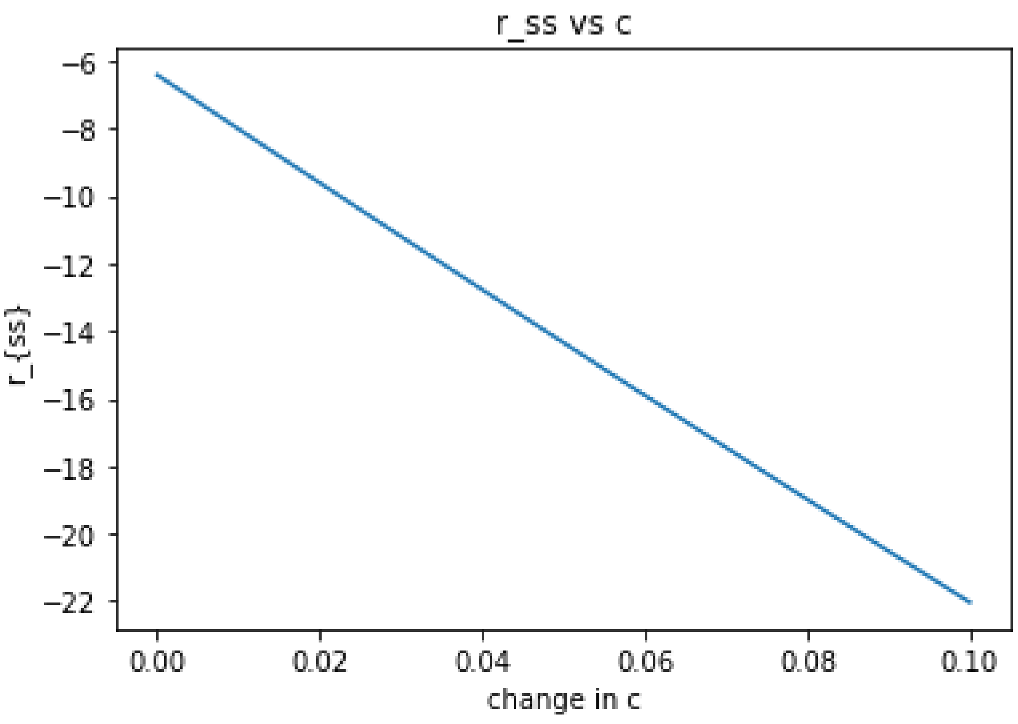
This is a diminishing curve too.

[24]:
```
plt.plot(np.linspace(0.000001, 0.1, 500), tsx)
plt.title("total payoff vs b")
plt.xlabel("change in b")
plt.ylabel("total payoff")
plt.show()
```



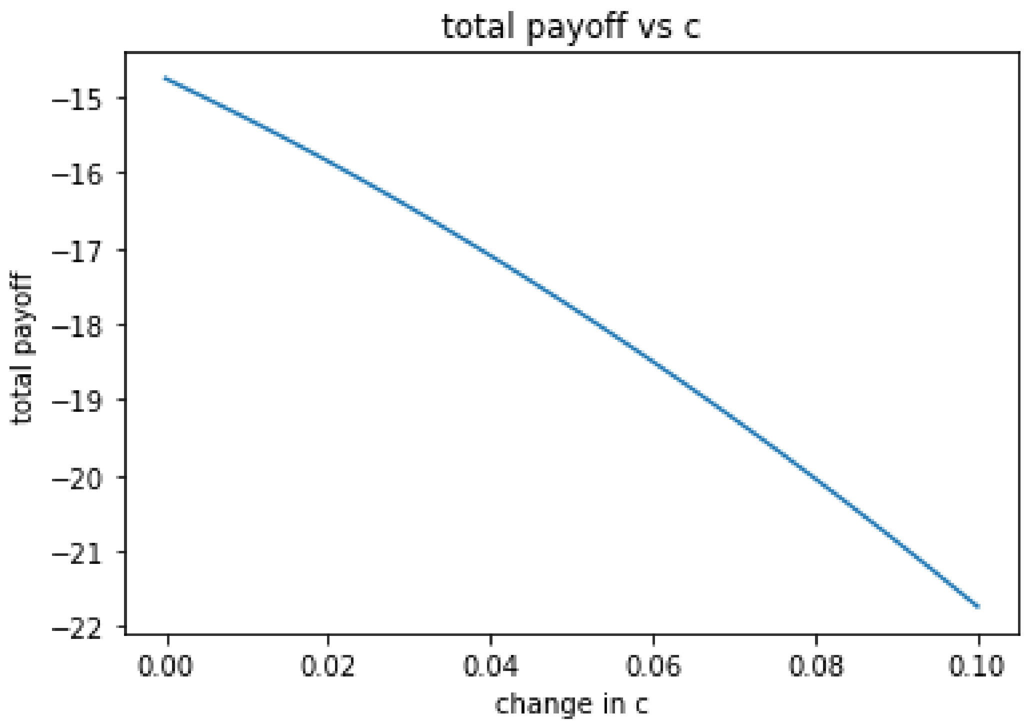
1.0.8 Experiment 6: Do the same to c

[25]:
```
rsx = []
tsx = []
# here I still name it b_delta to avoid having to rewrite the code
for b_delta in np.linspace(0.000001, 0.1, 500):
    A = np.array([
      [0.1 - b_delta/5, 0.2 - b_delta/5, 0.2 - b_delta/5, 0.2 - b_delta/5, 0.2␣
 ↪- b_delta/5],
      [0.2 - b_delta/5, 0.1 - b_delta/5, 0.2 - b_delta/5, 0.2 - b_delta/5, 0.2␣
 ↪- b_delta/5],
      [0.2 - b_delta/5, 0.2 - b_delta/5, 0.1 - b_delta/5, 0.2 - b_delta/5, 0.2␣
 ↪- b_delta/5],
      [0.2 - b_delta/5, 0.2 - b_delta/5, 0.2 - b_delta/5, 0.1 - b_delta/5, 0.2␣
 ↪- b_delta/5],
      [0.2 - b_delta/5, 0.2 - b_delta/5, 0.2 - b_delta/5, 0.2 - b_delta/5, 0.1␣
 ↪- b_delta/5],
    ], ndmin = 2)
    c = np.array([0.1 + b_delta, 0.1 + b_delta, 0 + b_delta, 0 + b_delta, 0 +␣
 ↪b_delta], ndmin = 2).T
    B = np.array([0, 0, 0.1, 0.1, 0.1], ndmin = 2).T
    x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
    rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, x,␣
↪10, delta = 0.8, tol = 10**(-14))
    rsx.append(rs[-1].item())
    tsx.append(ps[-1].item())

plt.plot(np.linspace(0.000001, 0.1, 500), rsx)
plt.title("r_ss vs c")
plt.xlabel("change in c")
plt.ylabel("r_{ss}")
plt.show()
```



This appears linear.

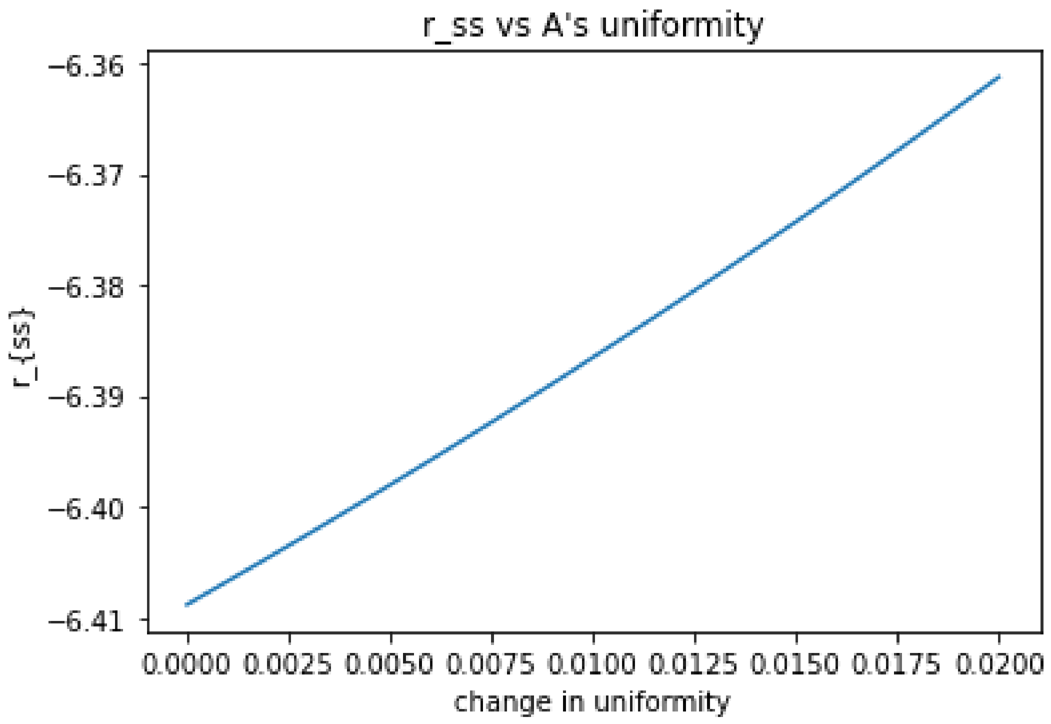
[26]:
```
plt.plot(np.linspace(0.000001, 0.1, 500), tsx)
plt.title("total payoff vs c")
plt.xlabel("change in c")
plt.ylabel("total payoff")
plt.show()
```



This, however, is not linear.

1.0.9 Experiment 7: Drive A to uniformity

[27]:
```
rsx = []
tsx = []

for b_delta in np.linspace(0.000001, 0.02, 500):
    A = np.array([
      [0.1 + b_delta * 4, 0.2 - b_delta, 0.2 - b_delta, 0.2 - b_delta, 0.2 -␣
 ↪b_delta],
      [0.2 - b_delta, 0.1 + b_delta * 4, 0.2 - b_delta, 0.2 - b_delta, 0.2 -␣
 ↪b_delta],
      [0.2 - b_delta, 0.2 - b_delta, 0.1 + b_delta * 4, 0.2 - b_delta, 0.2 -␣
 ↪b_delta],
      [0.2 - b_delta, 0.2 - b_delta, 0.2 - b_delta, 0.1 + b_delta * 4, 0.2 -␣
 ↪b_delta],
      [0.2 - b_delta, 0.2 - b_delta, 0.2 - b_delta, 0.2 - b_delta, 0.1 +␣
 ↪b_delta * 4],
    ], ndmin = 2)
    c = np.array([0.1, 0.1, 0, 0, 0], ndmin = 2).T
    B = np.array([0, 0, 0.1, 0.1, 0.1], ndmin = 2).T
    x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
    rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, x,␣
↪10, delta = 0.8, tol = 10**(-14))
    rsx.append(rs[-1].item())
    tsx.append(ps[-1].item())

plt.plot(np.linspace(0.000001, 0.02, 500), rsx)
plt.title("r_ss vs A's uniformity")
plt.xlabel("change in uniformity")
plt.ylabel("r_{ss}")
plt.show()
```

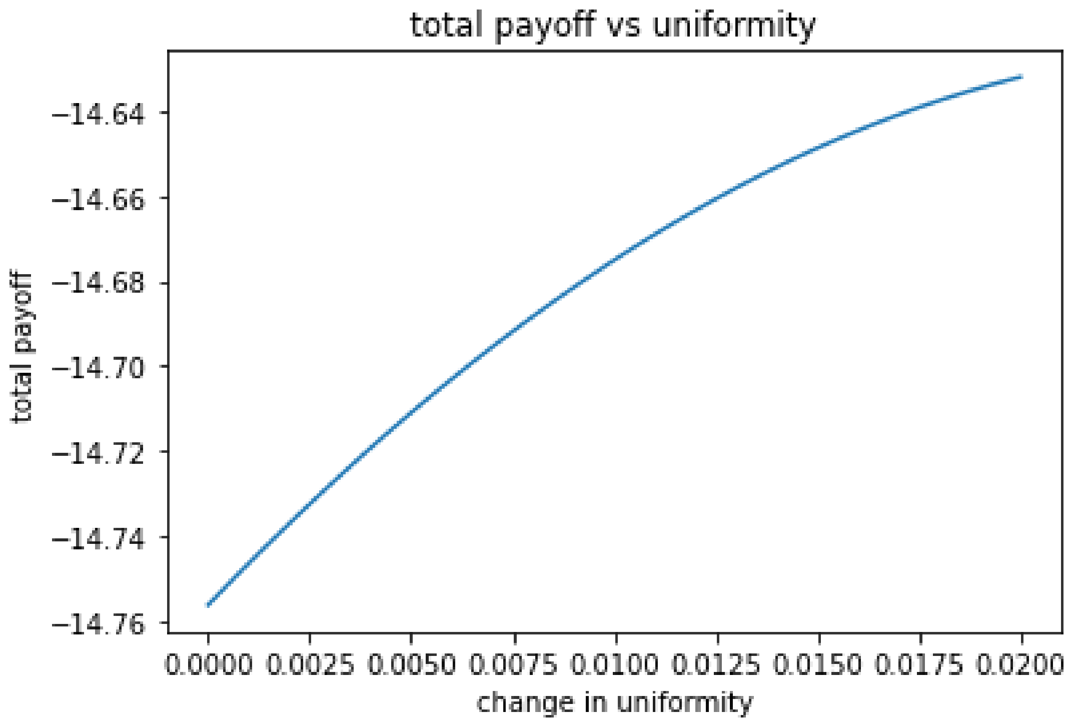


[28]:
```
A
```

[28]:
```
array([[0.18, 0.18, 0.18, 0.18, 0.18],
       [0.18, 0.18, 0.18, 0.18, 0.18],
       [0.18, 0.18, 0.18, 0.18, 0.18],
       [0.18, 0.18, 0.18, 0.18, 0.18],
       [0.18, 0.18, 0.18, 0.18, 0.18]])
```

(target is a perfectly uniform matrix)
As the uniformity increases, there is a concave upward positive increase.

[29]:
```
plt.plot(np.linspace(0.000001, 0.02, 500), tsx)
plt.title("total payoff vs uniformity")
plt.xlabel("change in uniformity")
plt.ylabel("total payoff")
plt.show()
```

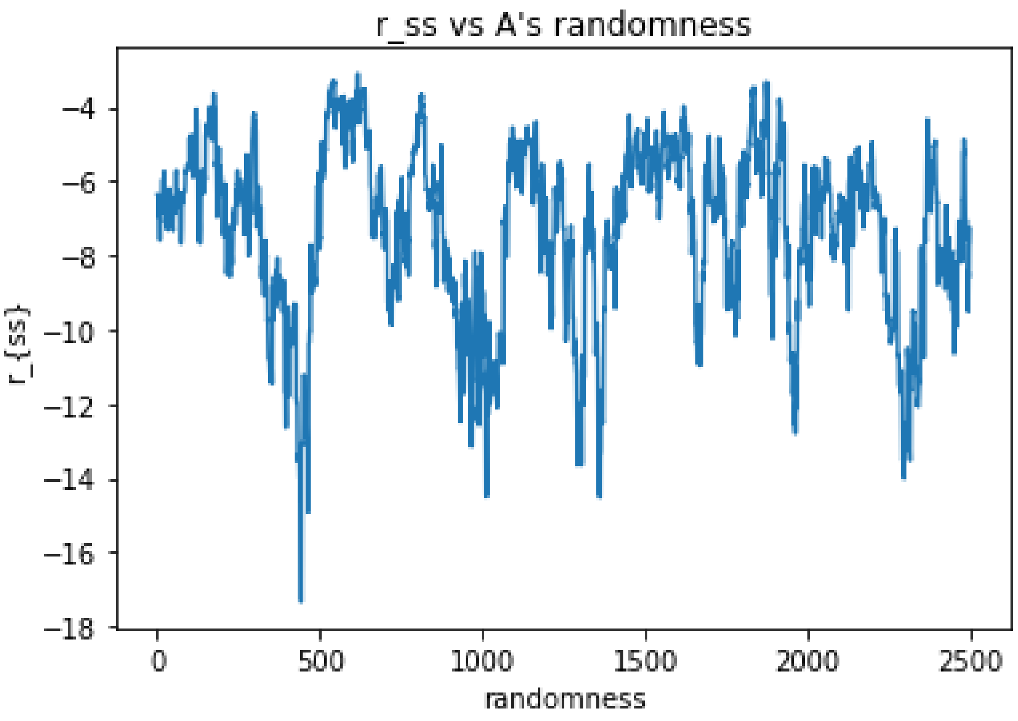


Note this represents shifting from listening to other naive agents to having more weight on one's own opinion and so this is the variable here.

1.0.10 Experiment 8: Random noise - going from uniform to not-uniform A

[30]:
```
import random
rsx = []
tsx = []
A = np.array([
      [0.18, 0.18, 0.18, 0.18, 0.18],
      [0.18, 0.18, 0.18, 0.18, 0.18],
      [0.18, 0.18, 0.18, 0.18, 0.18],
      [0.18, 0.18, 0.18, 0.18, 0.18],
      [0.18, 0.18, 0.18, 0.18, 0.18]
    ], ndmin = 2)
for b_delta in range(500):
    for i in range(5):
        c = np.array([0.1, 0.1, 0, 0, 0], ndmin = 2).T
        B = np.array([0, 0, 0.1, 0.1, 0.1], ndmin = 2).T
        x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
        rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B,␣
 ↪x, 10, delta = 0.8, tol = 10**(-14))
```

```
        rsx.append(rs[-1].item())
        tsx.append(ps[-1].item())
        pick_first = random.randrange(5)
        pick_second = random.randrange(5)
        dimx = A[i][pick_first]/2
        A[i][pick_first] -= dimx
        A[i][pick_second] += dimx

plt.plot(range(500*5), rsx)
plt.title("r_ss vs A's randomness")
plt.xlabel("randomness")
plt.ylabel("r_{ss}")
plt.show()
```

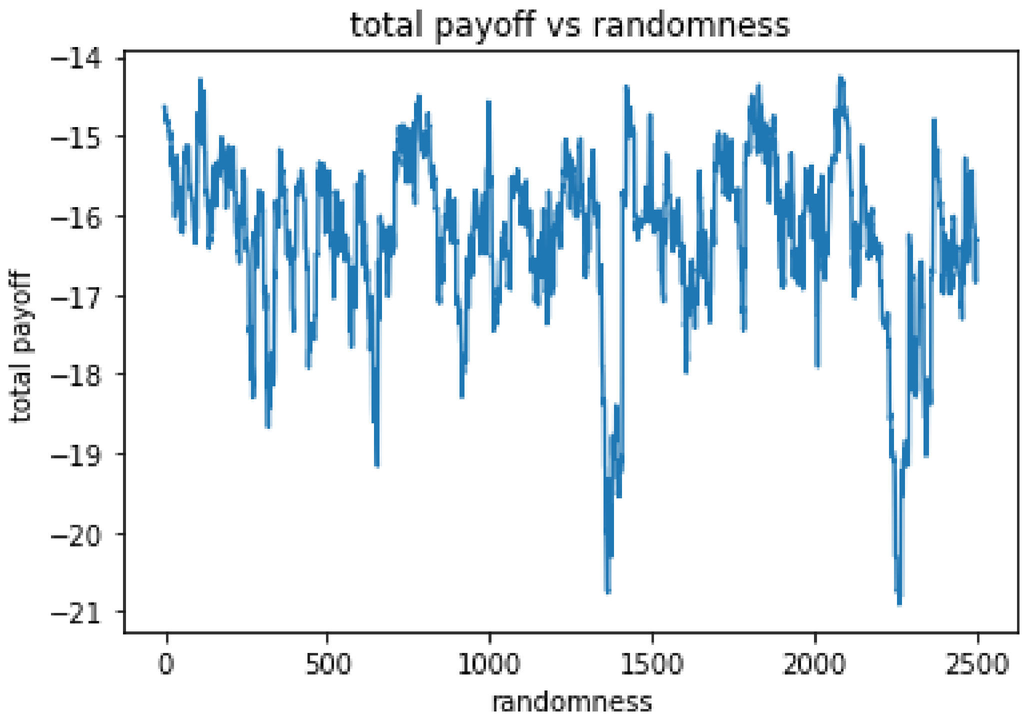


[31]:
```
A
```

[31]:
```
array([[0.22081683, 0.12103677, 0.04328012, 0.4935203 , 0.02134598],
       [0.11665131, 0.3784138 , 0.01577222, 0.32141562, 0.06774705],
       [0.32592029, 0.02738935, 0.02030688, 0.47049001, 0.05589347],
       [0.06752191, 0.4263279 , 0.19628763, 0.07860117, 0.13126139],
       [0.00240409, 0.14820022, 0.13433293, 0.36608478, 0.24897798]])
```

There's nothing tractable with this result, but it does indicate a sensitivity to network structure.

[32]:
```
plt.plot(range(500*5), tsx)
plt.title("total payoff vs randomness")
plt.xlabel("randomness")
plt.ylabel("total payoff")
plt.show()
```



Again, nothing tractable, but one takeaway here is that sensitivity is reduced in the payoff function compared to r_{ss}.

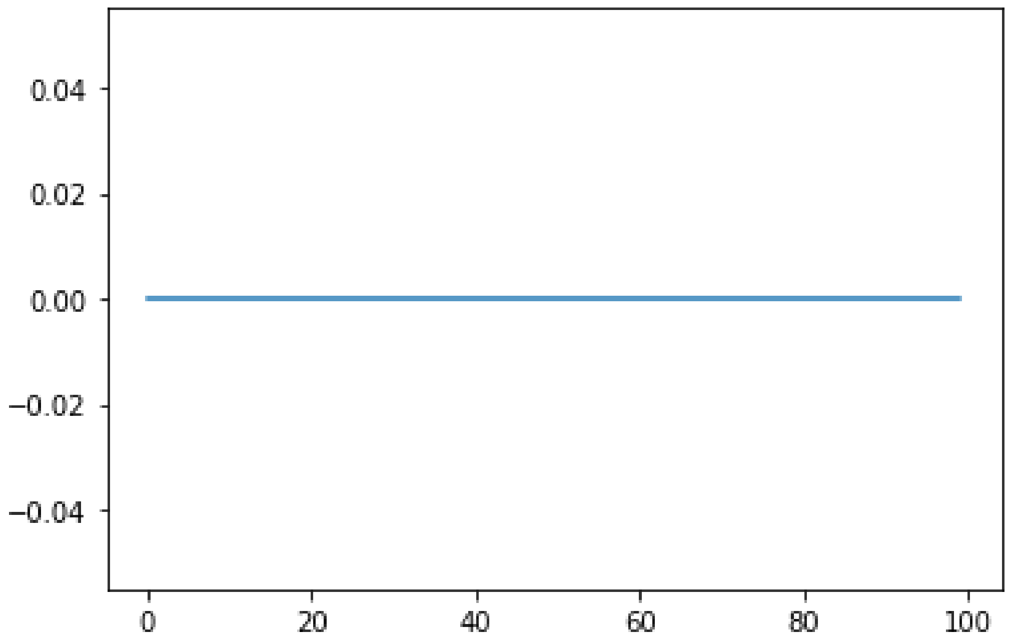
1.1 Conclusions so far

- δ is a dependency of r_{ss}, but the particular dependency is highly network-dependent; more analysis is necessary

The following conclusions use a symmetric matrix with on-diagonal entries being the same and off-diagonal entries being the same: - as R cost increases, r_{ss} moves closer to zero concave down with concave up decreasing payoff - as b uniformly increases, r_{ss} moves to zero concave down with concave down increasing payoff - as c uniformly increases, r_{ss} moves away from zero concave down with concave down decreasing payoff - as A moves closer to a uniform system (that is, greater diagonal), r_{ss} moves to zero concave up with concave down decreasing payoff - as A becomes more random, there is no tractable result, but r_{ss} is sensitive to small changes in A whereas payoff is less sensitive

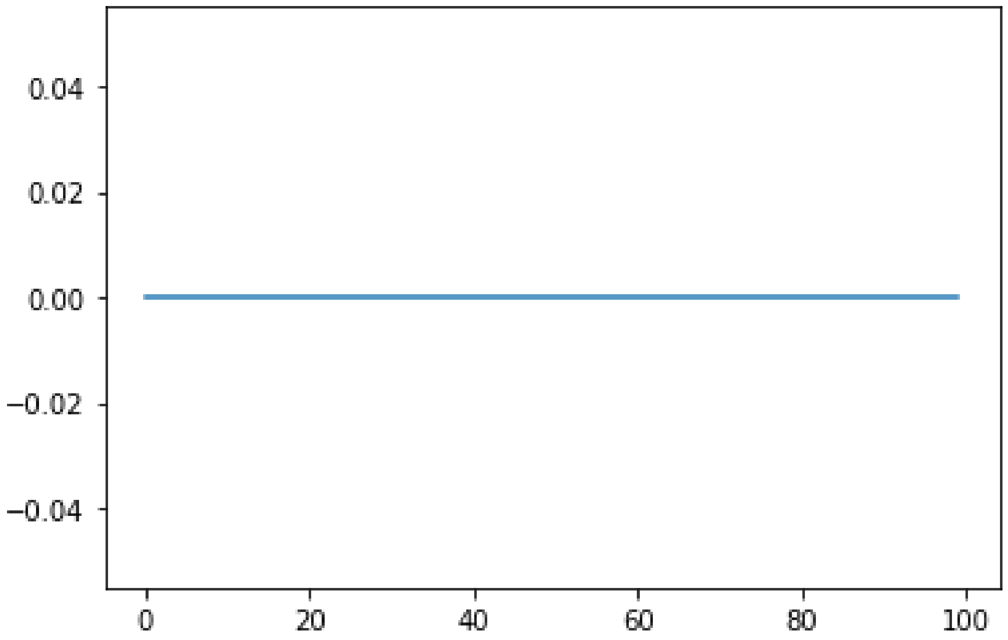
[33]:
```
def optimal_K_enhanced(A, c, B, x, z, delta = 0.8, tol = 10**(-18), R = 0,␣
 ↪num_agents = 2):
    A_tilde = np.concatenate((np.concatenate((A, c), axis = 1), # A c
        np.concatenate((np.zeros((1, num_agents)), np.array([1], ndmin = 2)),␣
↪axis = 1)), # 0 1
        axis = 0)
    B_tilde = np.concatenate((B, np.array([0], ndmin = 2)), axis = 0)
    w_0 = np.concatenate((x, np.array([z], ndmin = 2)), axis = 0)
    Q = 1 * np.identity(num_agents)
    Q_tilde = 1 * np.identity(num_agents+1)
    Q_tilde[num_agents, :] = 0

    def L(K_entry):
        return -1 * np.linalg.inv(B_tilde.T @ K_entry @ B_tilde + np.array(R,␣
↪ndmin = 2)/delta) @ B_tilde.T @ K_entry @ A_tilde

    # first compute the sequence of optimal K_t matrices
    K = np.zeros((num_agents+1, num_agents+1))
    K_t = [Q_tilde, K]
    K = Q_tilde
    current_difference = np.inf
    while abs(current_difference) > tol: # to avoid floating point error, don't␣
↪converge all the way to zero (this is standard)
        K_new = delta * (A_tilde.T @ (K
                - (K @ B_tilde @ np.linalg.inv(B_tilde.T @ K @ B_tilde + np.
↪array(R, ndmin = 2)/delta) @ B_tilde.T @ K))
                @ A_tilde) + Q_tilde
        K_t.insert(0, K_new)
        current_difference = np.max(np.abs(K - K_new))
        K = K_new

    # compute the Gamma matrix to use for later computations
    expr = A_tilde + B_tilde @ L(K_t[0])
    A_tilde_n = expr[:num_agents, :num_agents]
    c_nplus1 = np.array(expr[:num_agents, num_agents], ndmin = 2).T
    x_t = x
    x_ts = [x]

    # compute the resulting sequence of x_t opinion vectors
    for K_ent in K_t:
        x_tp1 = A_tilde_n @ x_t + c_nplus1 * z
        x_ts.append(x_tp1)
        x_t = x_tp1

    # compute the sequence of r_t and cumulative costs
    payoff = 0
    payoffs = []
    r_ts = []
    i = 0
    for x_ent in x_ts:
        r = L(K_t[0]) @ np.concatenate((x_ent, np.array([z], ndmin = 2)), axis␣
 ↪= 0)
        r_ts.append(r)
        payoff += (-1 * delta**i * ((x_ent.T @ Q @ x_ent).item() + (r * R *␣
 ↪r))) # account for discounting
        payoffs.append(payoff)
        i += 1

    return r_ts, A_tilde, B_tilde, w_0, K_t, x_ts, payoffs
```

[83]:
```
rss = []
for i in range(100):
    cutoff_1 = np.random.rand()
    cutoff_2 = np.random.rand()
    A = np.array([
      [cutoff_1, 0.5-cutoff_1],
      [cutoff_2, 0.5-cutoff_2],
    ], ndmin = 2)
    c = np.array([0, 0], ndmin = 2).T
    B = np.array([0.5, 0.5], ndmin = 2).T
    x = np.array([0, 0], ndmin = 2).T
    rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_enhanced(A, c, B, x,␣
 ↪1, tol = 10**(-1))
    rss.append(rs[-1].item())
plt.plot(range(100), rss)
plt.show()
```



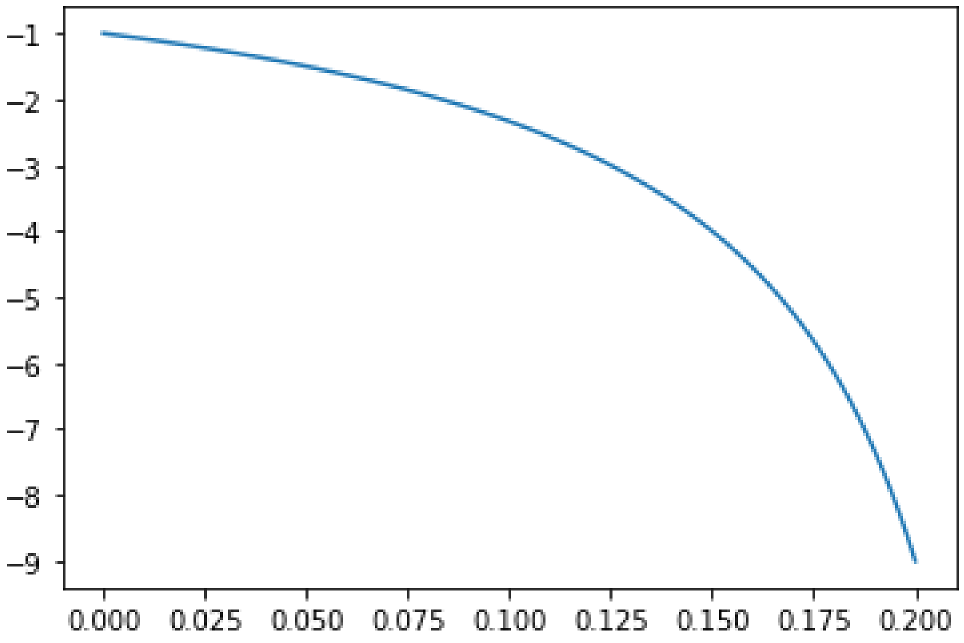
With a fixed B, zero c and a random A, there is no change.

[86]:
```
rss = []
for i in range(100):
    cutoff_1 = np.random.uniform(low = 0, high = 0.5, size = (2,))
    cutoff_2 = np.random.uniform(low = 0, high = 0.5, size = (2,))
    A = np.array([
      [cutoff_1[0], cutoff_1[1]],
      [cutoff_2[0], cutoff_2[1]],
    ], ndmin = 2)
    c = np.array([0, 0], ndmin = 2).T
    B = np.array([1 - cutoff_1[0] - cutoff_1[1], 1 - cutoff_2[0] -␣
 ↪cutoff_2[1]], ndmin = 2).T
    x = np.array([0, 0], ndmin = 2).T
    rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_enhanced(A, c, B, x,␣
 ↪1, tol = 10**(-1))
    rss.append(rs[-1].item())
plt.plot(range(100), rss)
plt.show()
```



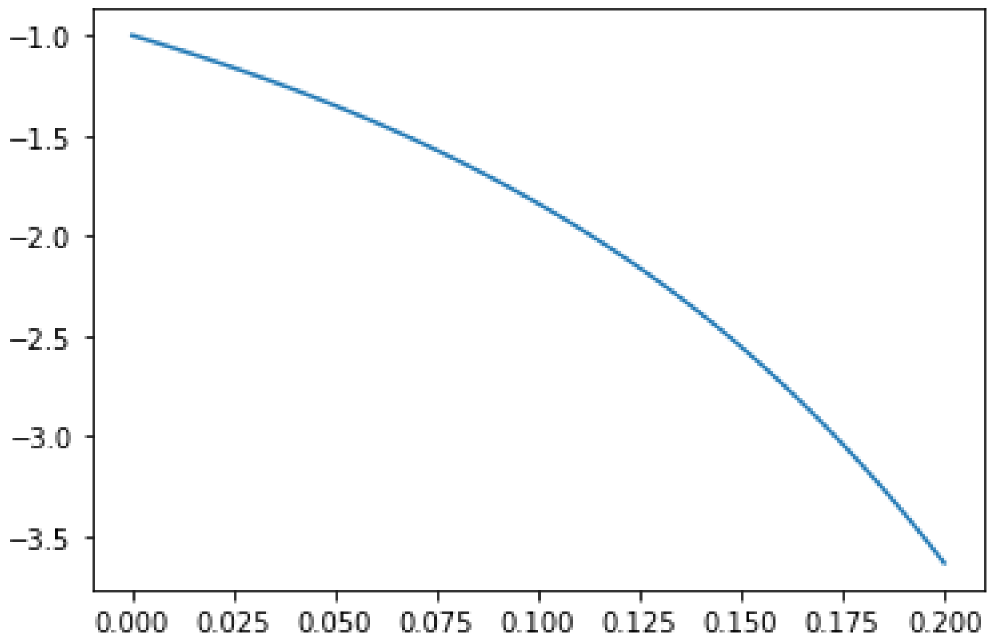
Nor does modifying B do anything. This means the intercept with respect to c is zero i.e. if c is zero, nothing will happen. I can now try modifying c to see what happens.

[93]:
```
rss = []
for i in np.linspace(0, 0.20, 100):
    cutoff_1 = np.random.uniform(low = 0, high = 0.25, size = (2,))
    cutoff_2 = np.random.uniform(low = 0, high = 0.25, size = (2,))
    A = np.array([
      [0.25, 0.25],
      [0.25, 0.25],
    ], ndmin = 2)
    c = np.array([0.25 + i, 0.25 + i], ndmin = 2).T
    B = np.array([0.25 - i, 0.25 - i], ndmin = 2).T
    x = np.array([0, 0], ndmin = 2).T
    rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_enhanced(A, c, B, x,␣
 ↪1, tol = 10**(-1))
    rss.append(rs[-1].item())
plt.plot(np.linspace(0, 0.20, 100), [a for a in rss])
plt.show()
```



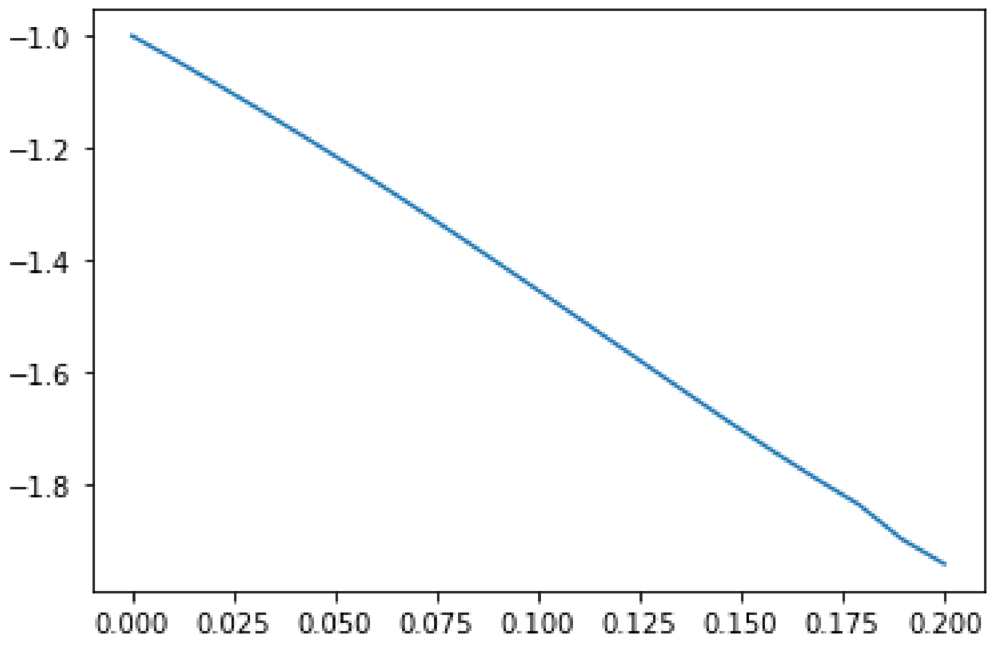
r_{ss} decreasing in increasing c, decreasing B.

[95]:
```
rss = []
for i in np.linspace(0, 0.20, 100):
    cutoff_1 = np.random.uniform(low = 0, high = 0.25, size = (2,))
    cutoff_2 = np.random.uniform(low = 0, high = 0.25, size = (2,))
    A = np.array([
      [0.25, 0.25],
      [0.25, 0.25],
    ], ndmin = 2)
    c = np.array([0.25 + i/2, 0.25 + i], ndmin = 2).T
    B = np.array([0.25 - i/2, 0.25 - i], ndmin = 2).T
    x = np.array([0, 0], ndmin = 2).T
    rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_enhanced(A, c, B, x,␣
↪1, tol = 10**(-1))
    rss.append(rs[-1].item())
plt.plot(np.linspace(0, 0.20, 100), [a for a in rss])
plt.show()
```



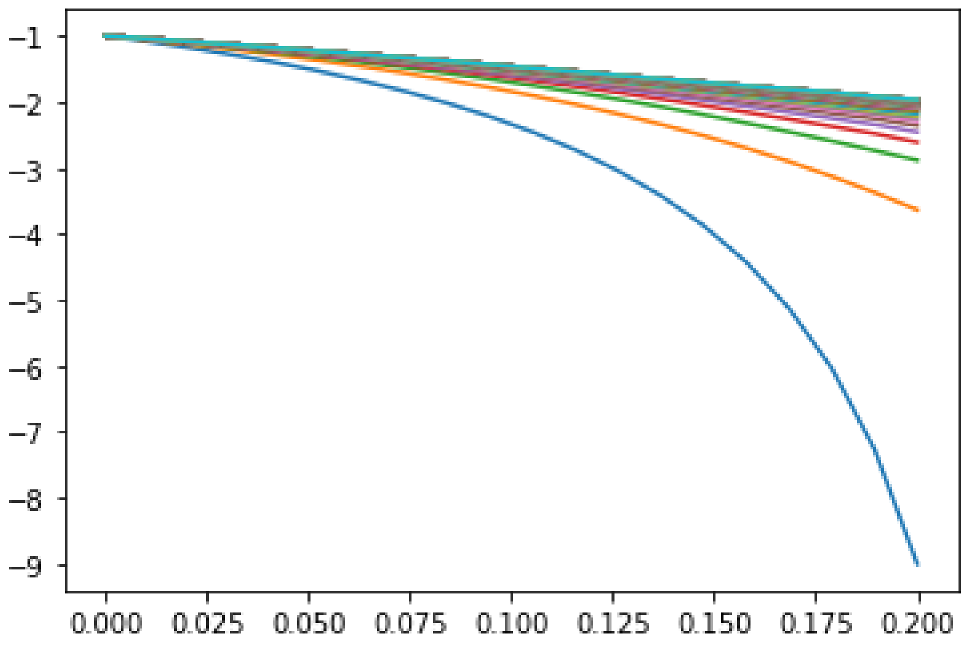
There is some cross-relation to the difference of differences when changing the two elements of B at different rates.

[100]:
```
rss = []
for i in np.linspace(0, 0.20, 20):
    cutoff_1 = np.random.uniform(low = 0, high = 0.25, size = (2,))
    cutoff_2 = np.random.uniform(low = 0, high = 0.25, size = (2,))
    A = np.array([
      [0.25, 0.25],
      [0.25, 0.25],
    ], ndmin = 2)
    c = np.array([0.25, 0.25 + i], ndmin = 2).T
    B = np.array([0.25, 0.25 - i], ndmin = 2).T
    x = np.array([0, 0], ndmin = 2).T
    rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_enhanced(A, c, B, x,␣
↪1, tol = 10**(-1))
    rss.append(rs[-1].item())
plt.plot(np.linspace(0, 0.20, 20), [a for a in rss])
plt.show()
```



The limit is almost a straight line, although there's a bump close to the 0.175 marker.

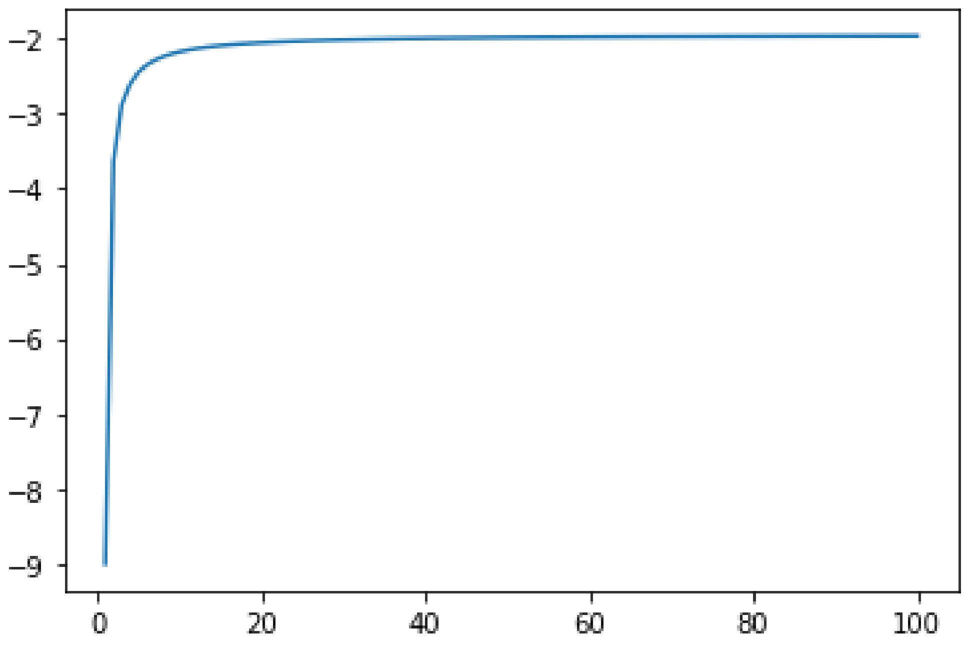
[102]:
```
for divisor in np.linspace(1, 100, 100):
    rss = []
    for i in np.linspace(0, 0.20, 20):
        cutoff_1 = np.random.uniform(low = 0, high = 0.25, size = (2,))
        cutoff_2 = np.random.uniform(low = 0, high = 0.25, size = (2,))
        A = np.array([
          [0.25, 0.25],
          [0.25, 0.25],
        ], ndmin = 2)
        c = np.array([0.25 + i/divisor, 0.25 + i], ndmin = 2).T
        B = np.array([0.25 - i/divisor, 0.25 - i], ndmin = 2).T
        x = np.array([0, 0], ndmin = 2).T
        rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_enhanced(A, c, B,␣
 ↪x, 1, tol = 10**(-1))
        rss.append(rs[-1].item())
    plt.plot(np.linspace(0, 0.20, 20), [a for a in rss])
plt.show()
```



There is an upwards convergence. What this plot means is that, as one of B's entries shifts over to c at a smaller and smaller rate than the other, r_{ss} becomes overall smaller in magnitude (more positive). In other words, because less influence is being lost, the messages don't have to be as high in magnitude.

Conversely, when more influence is shifted to c, the messages overall have to be greater in magnitude (more negative).

[153]:
```
rss = []
for divisor in np.linspace(1, 100, 100):
    cutoff_1 = np.random.uniform(low = 0, high = 0.25, size = (2,))
    cutoff_2 = np.random.uniform(low = 0, high = 0.25, size = (2,))
    A = np.array([
      [0.25, 0.25],
      [0.25, 0.25],
    ], ndmin = 2)
    c = np.array([0.25 + 0.2/divisor, 0.25 + 0.2], ndmin = 2).T
    B = np.array([0.25 - 0.2/divisor, 0.25 - 0.2], ndmin = 2).T
    x = np.array([0, 0], ndmin = 2).T
    rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_enhanced(A, c, B, x,␣
 ↪1, tol = 10**(-1))
    rss.append(rs[-1].item())
plt.plot(np.linspace(1, 100, 100), rss)
#plt.plot(np.linspace(1, 100, 100), [-1/a - 2 for a in np.linspace(1, 100,␣
 ↪100)])
plt.show()
```



This is the trajectory of the convergence of the endpoints of every point on the rainbow plot beforehand. In other words, as the fraction of influence lost to c decreases, r_{ss} becomes smaller in magnitude (more positive).