BotInfluence3

August 16, 2021

1 New comparisons in the bot strategic model

James Yu

15 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_infinite(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
    while True:
        x_tp1 = A_tilde_n @ x_t + c_nplus1 * z
        x_ts.append(x_tp1)
        if np.max(np.abs(x_t - x_tp1)) == 0:
            x_t = x_tp1
            break
        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))).item() # account for discounting
        payoffs.append(payoff)
        i += 1

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

[4]:
```
def myopic(A, B, c, x, z, delta):
    # NEW FUNCTION: determines the myopic strategy sequence of r_t given parameters
    A_tilde = np.concatenate((np.concatenate((A, c), axis = 1), # A c
```

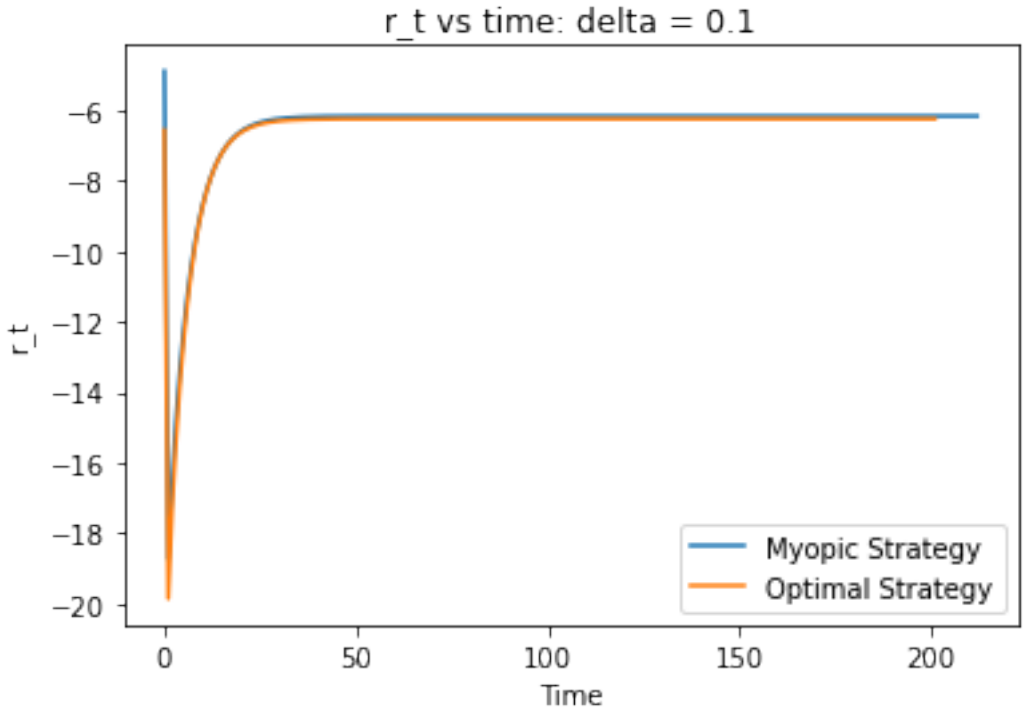
```
    np.concatenate((np.zeros((1, 5)), np.array([1], ndmin = 2)), axis = 1)), #␣
↪0 1
    axis = 0)
    B_tilde = np.concatenate((B, np.array([0], ndmin = 2)), axis = 0)
    w_0 = np.concatenate((x, np.array([10], ndmin = 2)), axis = 0)
    Q = 0.2 * np.identity(5)
    Q_tilde = 0.2 * np.identity(6)
    Q_tilde[5, :] = 0
    L_m = -np.linalg.inv(B_tilde.T @ Q_tilde @ B_tilde) @ B_tilde.T @ Q_tilde @␣
↪A_tilde
    x_t = w_0
    rs = []
    x_ts = [w_0]
    while True:
        rs.append(L_m @ x_t)
        x_tp1 = (A_tilde + B_tilde @ L_m) @ x_t
        x_ts.append(x_tp1)
        if np.max(np.abs(x_t - x_tp1)) == 0:
            x_t = x_tp1
            break
        x_t = x_tp1

    return rs, x_ts
```

1.1 Check 1: Myopic strategy for low levels of δ vs. optimal strategy for low levels of δ

using $\delta = 0.1$ and the original network from the first reports:

[5]:
```
# FIRST, CHECK THE OPTIMAL STRATEGY USING EXISTING CODE
delta = 0.1
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_infinite(A, c, B, x, 10,␣
↪delta = delta)

# NEXT, DO THE MYOPIC STRATEGY UNDER THE SAME PARAMETERS
rs2, xs2 = myopic(A, B, c, x, 10, delta)
```

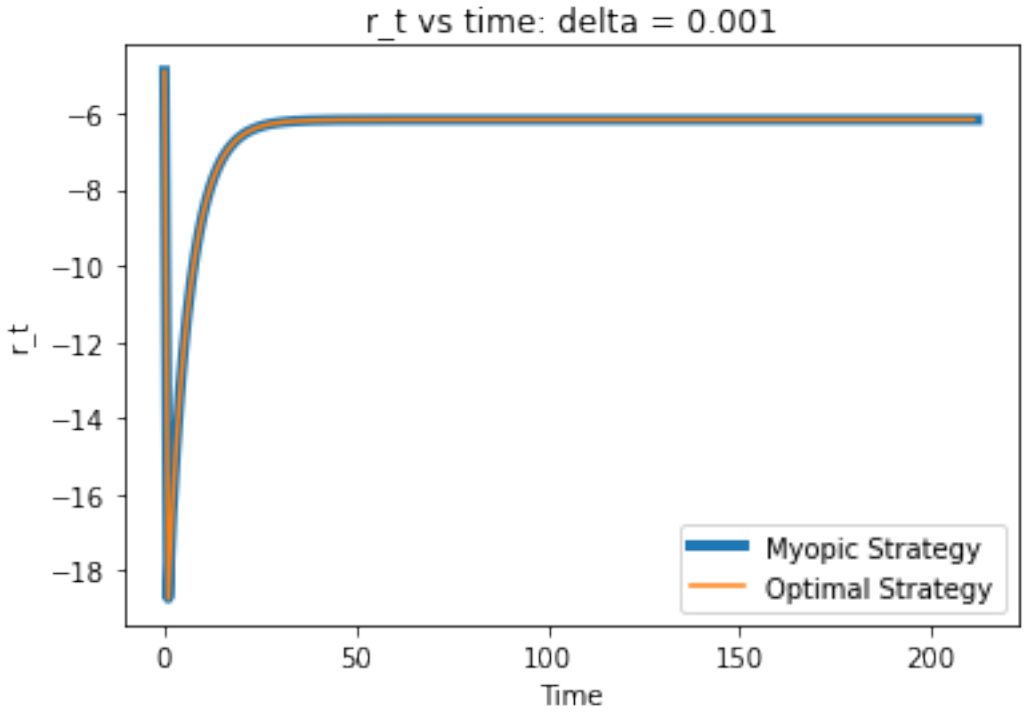
```
plt.plot(range(len(rs2)), [rx.item() for rx in rs2], label = "Myopic Strategy")
plt.plot(range(len(rs)), [rx.item() for rx in rs], label = "Optimal Strategy")
plt.xlabel("Time")
plt.ylabel("r_t")
plt.title(f"r_t vs time: delta = {delta}")
plt.legend()
plt.show()
```



I can repeat this plot for a few values of δ:

[6]:
```
delta = 0.001
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_infinite(A, c, B, x, 10,␣
 ↪delta = delta)
```

```
rs2, xs2 = myopic(A, B, c, x, 10, delta)
plt.plot(range(len(rs2)), [rx.item() for rx in rs2], label = "Myopic Strategy",␣
 ↪linewidth = 4)
plt.plot(range(len(rs)), [rx.item() for rx in rs], label = "Optimal Strategy")
plt.xlabel("Time")
plt.ylabel("r_t")
plt.title(f"r_t vs time: delta = {delta}")
plt.legend()
plt.show()
```



The strategies are almost identical. We can do a quick check to see if this is true:

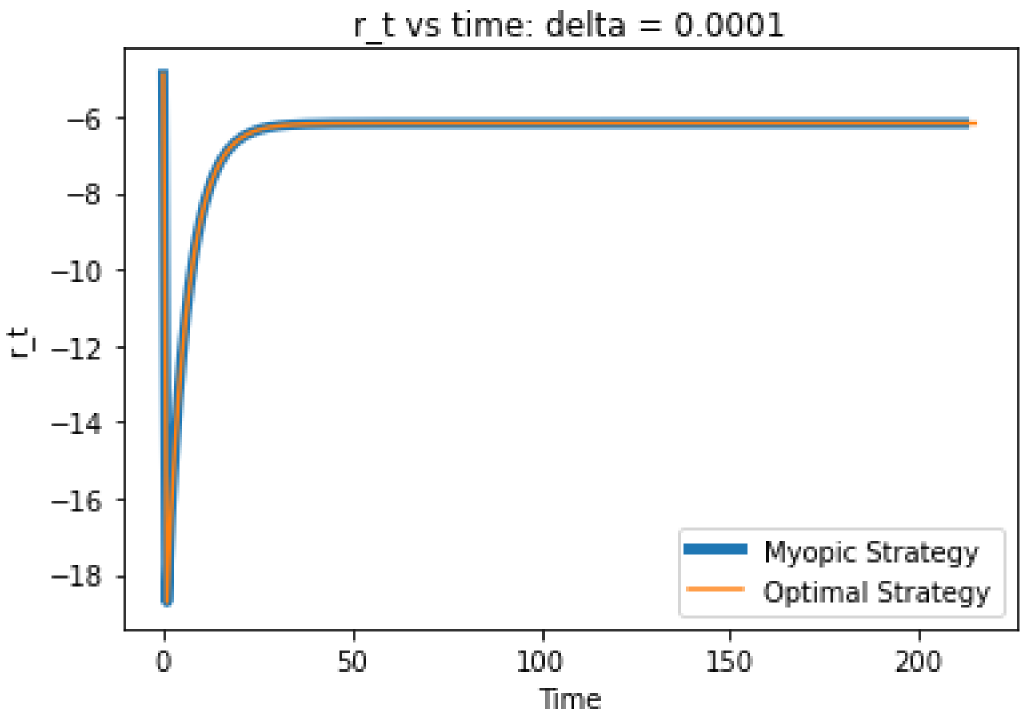
[7]:
```
", ".join([str(round(rx.item() - rx2.item(), 4)) for rx, rx2 in zip(rs, rs2)])
```

[7]:
```
'-0.0159, -0.0109, -0.008, -0.006, -0.0045, -0.0033, -0.0024, -0.0017, -0.0012,
-0.0009, -0.0006, -0.0004, -0.0003, -0.0002, -0.0002, -0.0002, -0.0002, -0.0002,
-0.0003, -0.0003, -0.0003, -0.0004, -0.0004, -0.0005, -0.0005, -0.0005, -0.0005,
-0.0006, -0.0006, -0.0006, -0.0006, -0.0007, -0.0007, -0.0007, -0.0007, -0.0007,
-0.0007, -0.0007, -0.0007, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
```

```
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008, -0.0008,
-0.0008, -0.0008, -0.0008, -0.0008, -0.0008'
```

These numbers are the optimal sequence of r_t minus the myopic sequence of r_t. For example, r_0 optimal minus r_0 myopic is -0.0159. This indicates that, to start with, the myopic strategy r_t is greater than the optimal strategy (in other words, closer to zero since it is negative). For a few periods in the middle, the strategies become most similar to each other, and then they diverge to a fixed difference of -0.0008. These are quite small, indicating that perhaps closer to the limit, they all approach zero.

[8]:
```
delta = 0.0001
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_infinite(A, c, B, x, 10,␣
↪delta = delta)
rs2, xs2 = myopic(A, B, c, x, 10, delta)
plt.plot(range(len(rs2)), [rx.item() for rx in rs2], label = "Myopic Strategy",␣
↪linewidth = 4)
plt.plot(range(len(rs)), [rx.item() for rx in rs], label = "Optimal Strategy")
plt.xlabel("Time")
plt.ylabel("r_t")
plt.title(f"r_t vs time: delta = {delta}")
plt.legend()
plt.show()
```



[9]:
```
", ".join([str(round(rx.item() - rx2.item(), 4)) for rx, rx2 in zip(rs, rs2)])
```

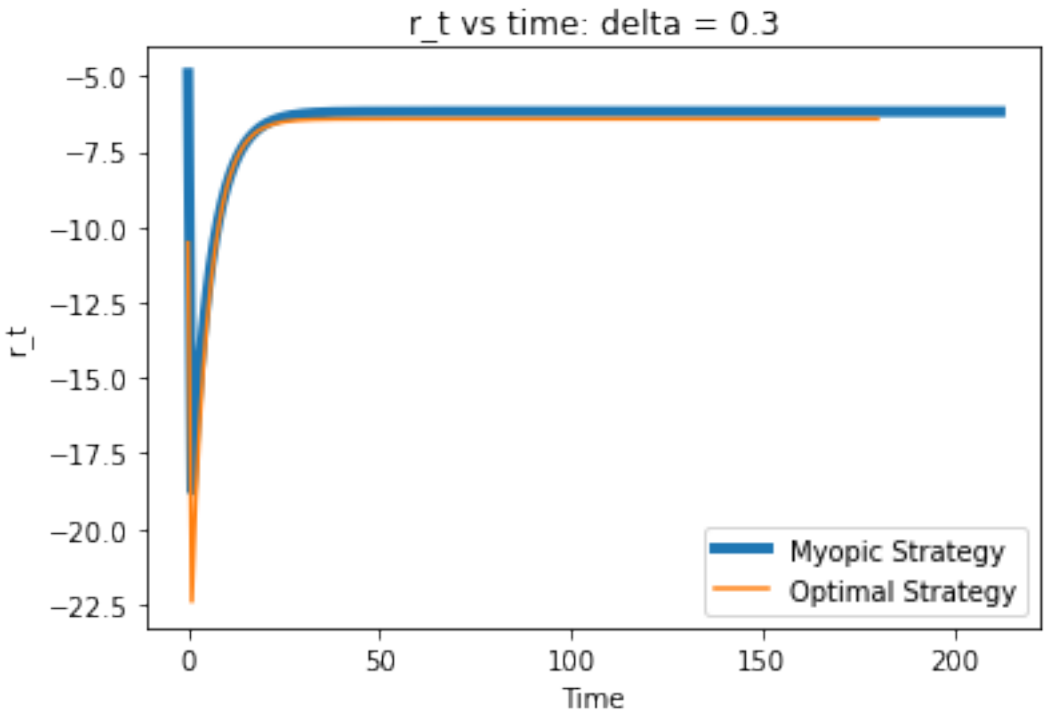
[9]:
```
'-0.0016, -0.0011, -0.0008, -0.0006, -0.0004, -0.0003, -0.0002, -0.0002,
-0.0001, -0.0001, -0.0001, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0,
-0.0, -0.0, -0.0, -0.0, -0.0, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
```

```
-0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001, -0.0001,
-0.0001, -0.0001, -0.0001'
```

These are the same numbers but for $\delta = 0.0001$. Clearly they are smaller (some display as zero exactly, but this is because I round to four decimal places, so they are still a little ways off from zero).

Now, I can try a greater value of δ, perhaps $\delta = 0.3$:

[10]:
```
delta = 0.3
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_infinite(A, c, B, x, 10,␣
 ↪delta = delta)
rs2, xs2 = myopic(A, B, c, x, 10, delta)
plt.plot(range(len(rs2)), [rx.item() for rx in rs2], label = "Myopic Strategy",␣
 ↪linewidth = 4)
plt.plot(range(len(rs)), [rx.item() for rx in rs], label = "Optimal Strategy")
plt.xlabel("Time")
plt.ylabel("r_t")
plt.title(f"r_t vs time: delta = {delta}")
plt.legend()
plt.show()
```



This time we don't need the differences to see that the myopic strategy is closer to zero overall compared to the optimal strategy.

2 Check 2: Compute the finite L_t, graph r_t and vary the graph with δ

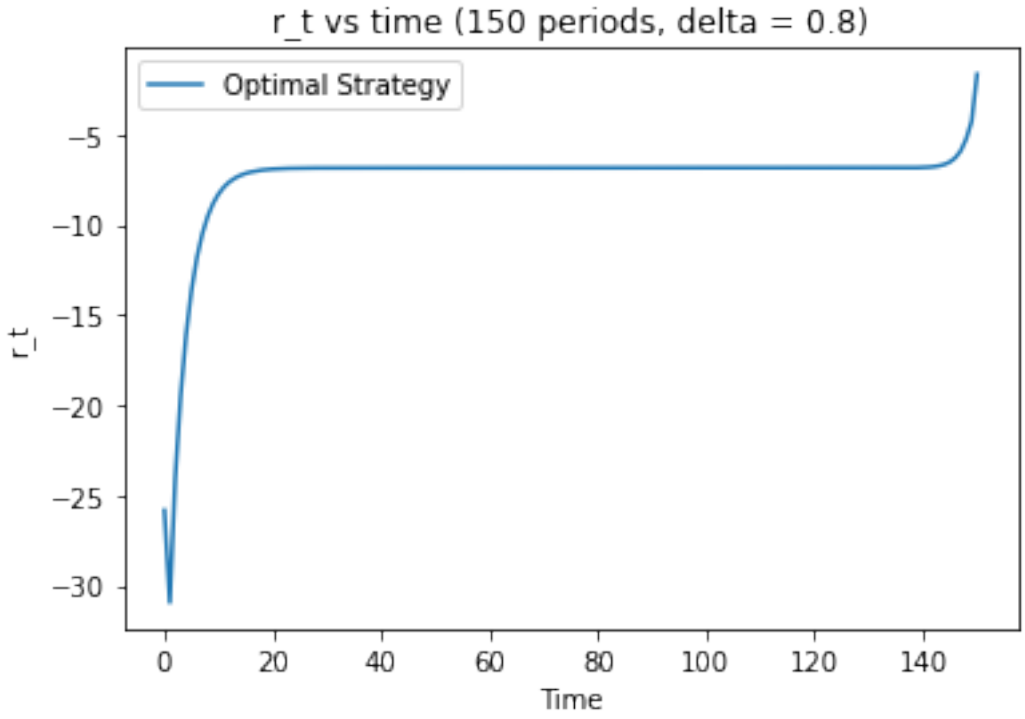
[11]:
```
def optimal_K_finite(A, c, B, x, z, delta = 0.8, tol = 10**(-18), R = 0,␣
↪periods = 1000):
    A_tilde = np.concatenate((np.concatenate((A, c), axis = 1), # A c
        np.concatenate((np.zeros((1, 5)), np.array([1], ndmin = 2)), axis =␣
↪1)), # 0 1
        axis = 0)
    B_tilde = np.concatenate((B, 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
    for i in range(periods):
        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 i in range(periods):
        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[i]) @ 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))).item() # account for discounting
        payoffs.append(payoff)
        i += 1

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

First, a baseline $\delta = 0.8$ finite model with 150 periods:

[12]:
```
delta = 0.8
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],
```

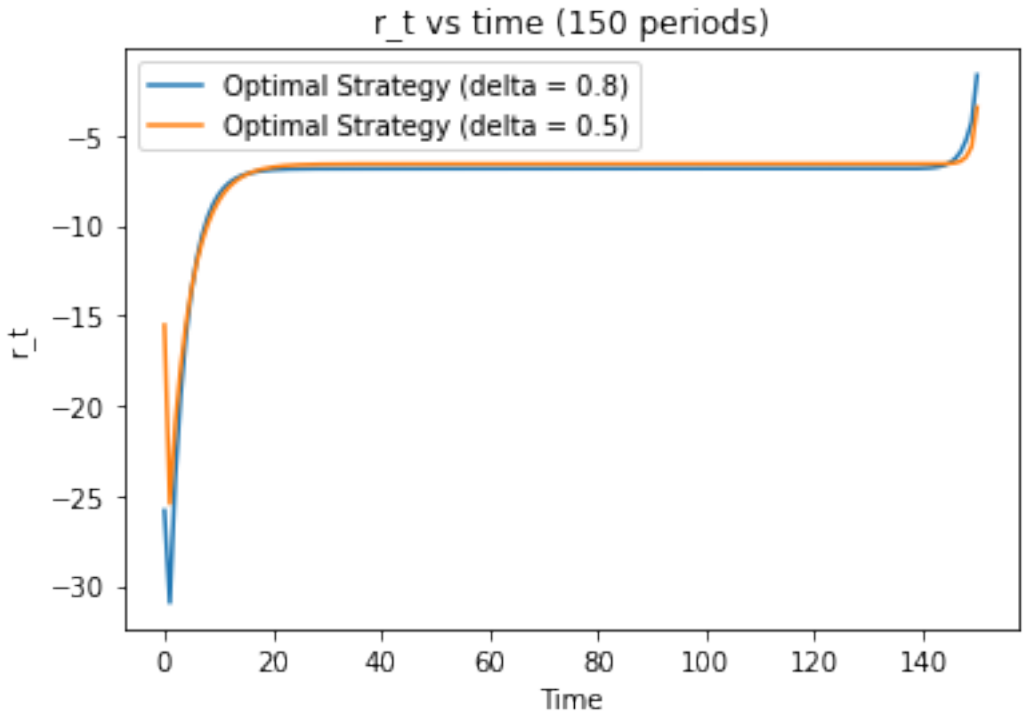
```
  [0.1285,   0.0907,   0.3185,   0.2507,   0.2116],
  [0.1975,   0.0629,   0.2863,   0.2396,   0.2137],
  [0.1256,   0.0711,   0.0253,   0.2244,   0.5536],
], ndmin = 2)
c = np.array([0, 0.1012, 0, 0, 0,], ndmin = 2).T
B = np.array([0.0791, 0, 0, 0, 0,], ndmin = 2).T
x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_finite(A, c, B, x, 10,␣
 ↪delta = delta, periods = 150)
plt.plot(range(len(rs)), [rx.item() for rx in rs], label = "Optimal Strategy")
plt.xlabel("Time")
plt.ylabel("r_t")
plt.title(f"r_t vs time (150 periods, delta = 0.8)")
plt.legend()
plt.show()
```



Now, I overlay $\delta = 0.5$ on the same graph:

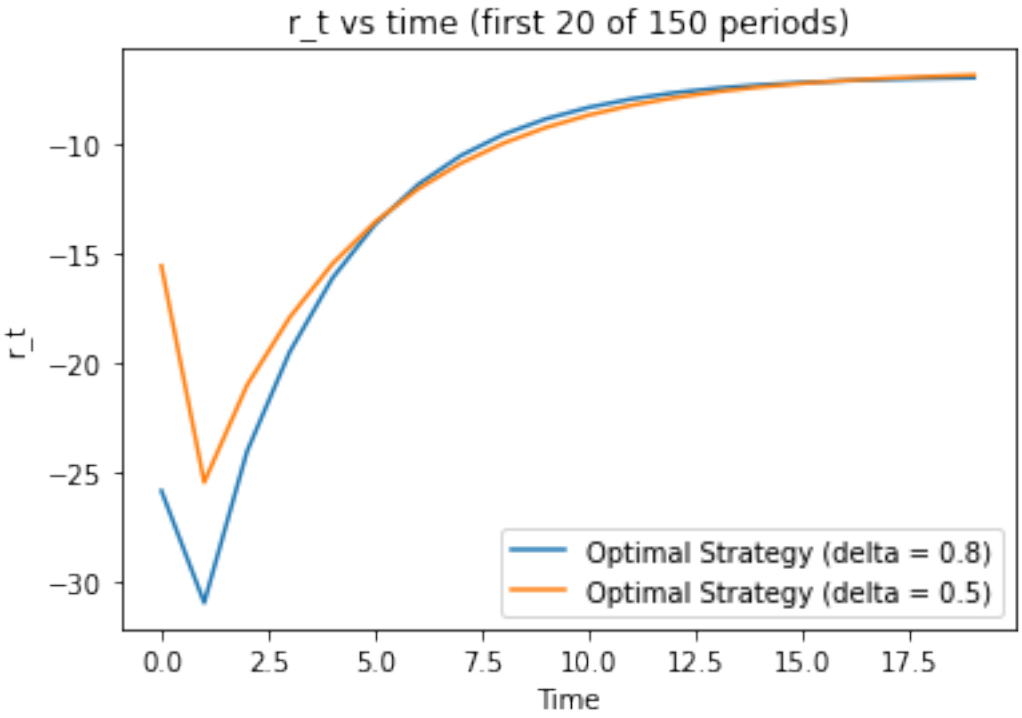
[13]:
```
A = np.array([
  [0.217,    0.2022,   0.2358,   0.1256,   0.1403],
  [0.8988*0.2497,   0.8988*0.0107,   0.8988*0.2334,   0.8988*0.1282,   0.8988*0.
 ↪378],
  [0.1285,   0.0907,   0.3185,   0.2507,   0.2116],
  [0.1975,   0.0629,   0.2863,   0.2396,   0.2137],
  [0.1256,   0.0711,   0.0253,   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_finite(A, c, B, x, 10,␣
 ↪delta = 0.8, periods = 150)
plt.plot(range(len(rs)), [rx.item() for rx in rs], label = "Optimal Strategy␣
 ↪(delta = 0.8)")
rs2, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_finite(A, c, B, x, 10,␣
 ↪delta = 0.5, periods = 150)
plt.plot(range(len(rs2)), [rx.item() for rx in rs2], label = "Optimal Strategy␣
 ↪(delta = 0.5)")
plt.xlabel("Time")
plt.ylabel("r_t")
plt.title(f"r_t vs time (150 periods)")
plt.legend()
plt.show()
```

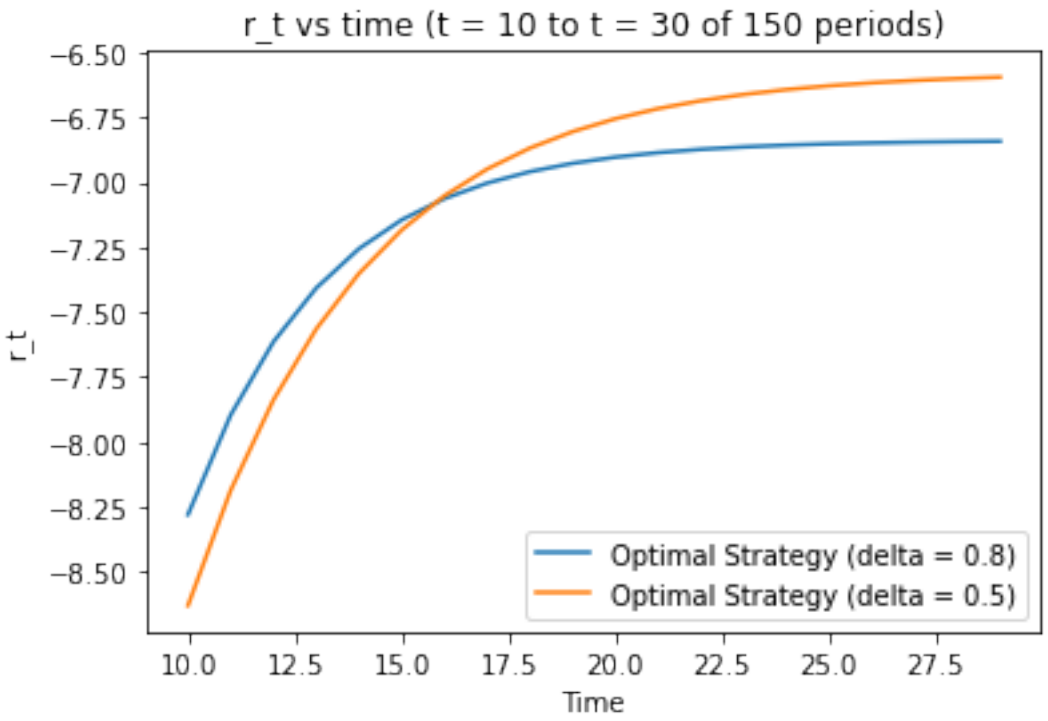


[14]:
```
plt.plot(range(20), [rx.item() for rx in rs][:20], label = "Optimal Strategy␣
 ↪(delta = 0.8)")
rs2, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_finite(A, c, B, x, 10,␣
 ↪delta = 0.5, periods = 150)
```

```
plt.plot(range(20), [rx.item() for rx in rs2][:20], label = "Optimal Strategy␣
 ↪(delta = 0.5)")
plt.xlabel("Time")
plt.ylabel("r_t")
plt.title(f"r_t vs time (first 20 of 150 periods)")
plt.legend()
plt.show()
```



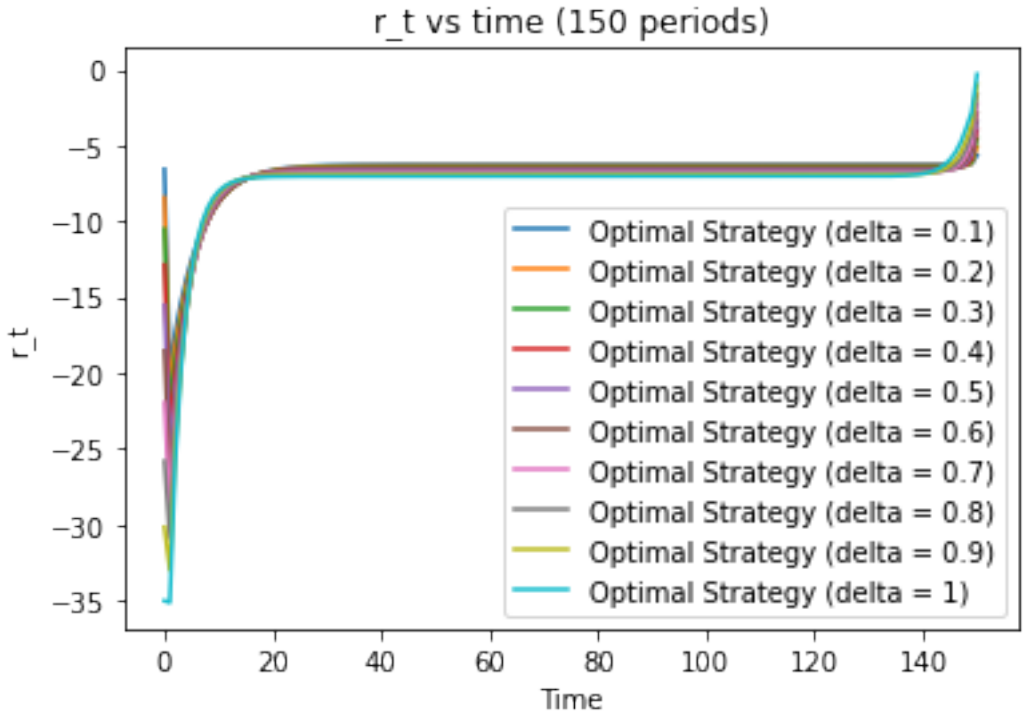
[15]:
```
plt.plot(range(10, 30), [rx.item() for rx in rs][10:30], label = "Optimal␣
 ↪Strategy (delta = 0.8)")
rs2, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_finite(A, c, B, x, 10,␣
 ↪delta = 0.5, periods = 150)
plt.plot(range(10, 30), [rx.item() for rx in rs2][10:30], label = "Optimal␣
 ↪Strategy (delta = 0.5)")
plt.xlabel("Time")
plt.ylabel("r_t")
plt.title(f"r_t vs time (t = 10 to t = 30 of 150 periods)")
plt.legend()
plt.show()
```



In fact the graphs cross multiple times. First of all, we see that as δ decreases, the messages are closer to zero in the short run, further from zero in the medium run, closer to zero in the long run and then further from zero at the end. Second, clearly the graphs are not monotonic due to the initial decrease. The decrease is less sharp as delta increases which indicates a correlation with myopic-like behavior. A more comprehensive graph:

[16]:
```
for delta in [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]:
    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_finite(A, c, B, x, 10,␣
 ↪delta = delta, periods = 150)
    plt.plot(range(len(rs)), [rx.item() for rx in rs], label = f"Optimal␣
 ↪Strategy (delta = {delta})")
plt.xlabel("Time")
plt.ylabel("r_t")
```

```
plt.title(f"r_t vs time (150 periods)")
plt.legend()
plt.show()
```

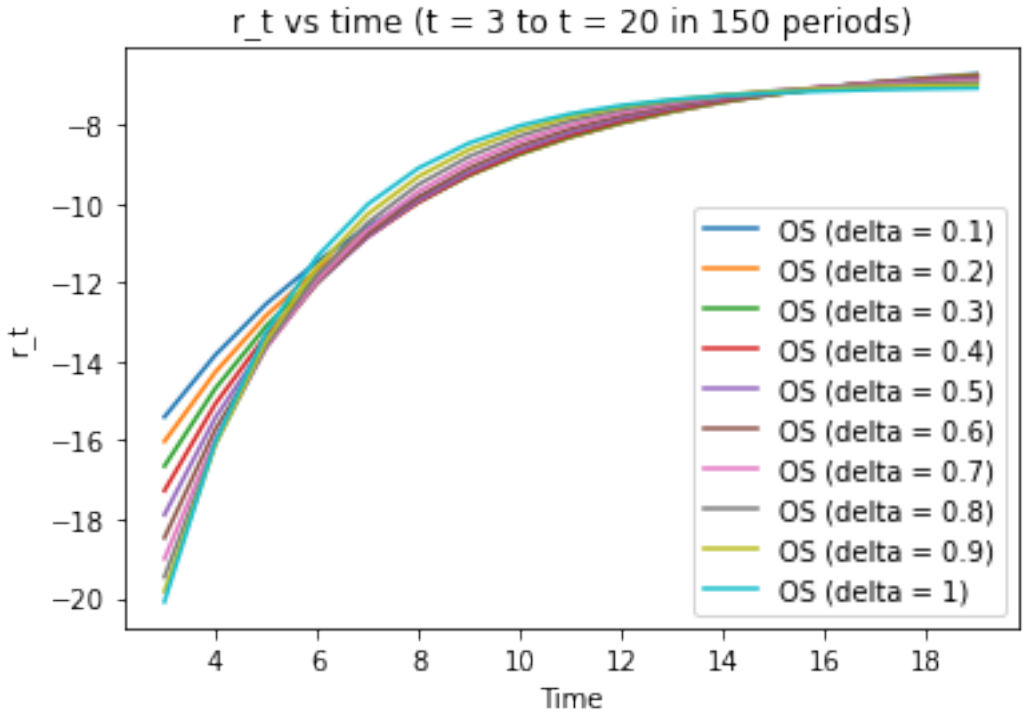


This graph demonstrates all the previously-observed properties. As δ increases, the initial r_0 becomes further away from zero due to less future-discounting, which gradually eliminates the observed "dip" behavior. between $t = 0$ and $t = 20$ there is a pair of intercepts between which, as δ increases, the messages are closer to zero. More specifically it appears that the point of "convergence" i.e. the zone where there is no change in r_t is faster/shorter/closer to $t = 0$ as δ increases, likely due to the larger initial messages used when not discounting the future (since with no discount, there's no difference between the future and now, so it makes sense to use a larger message sooner).

In the long-run, with higher δ, r_t is further away from zero, and then at the very end, again there is a faster turn toward $r_t = 0$ with higher δ. This leads to the conclusion that higher δ means faster convergence, among other subtleties which are corollaries of this result. Between $t = 3$ and $t = 20$ we see the following:

[17]:
```
for delta in [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]:
    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],
```

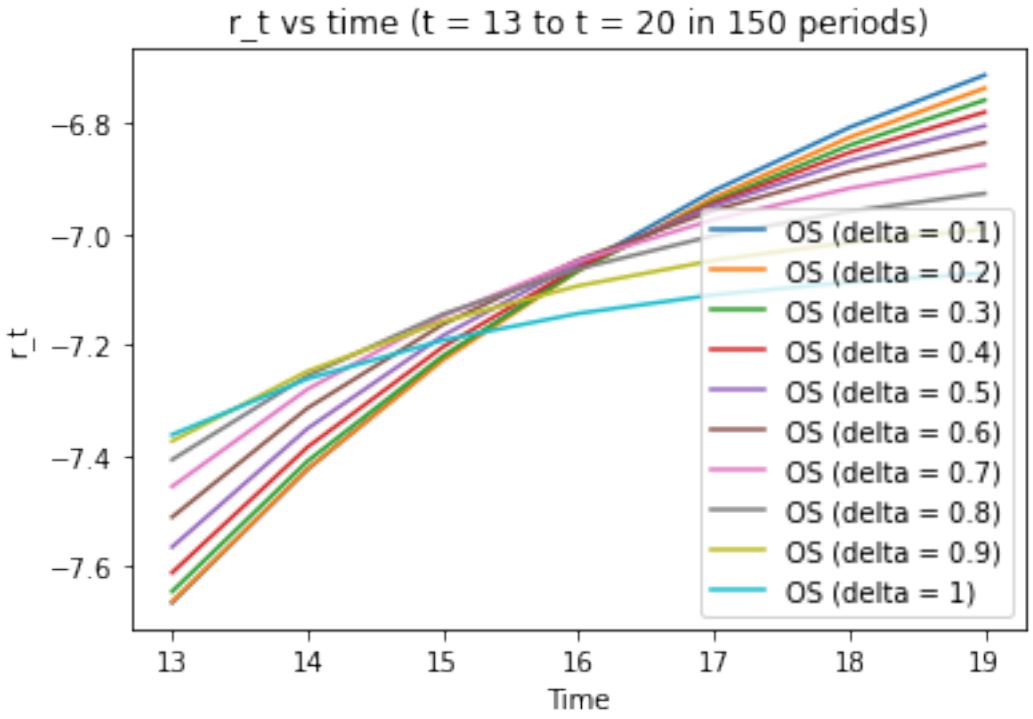
```
    ], ndmin = 2)
    c = np.array([0, 0.1012, 0, 0, 0,], ndmin = 2).T
    B = np.array([0.0791, 0, 0, 0, 0,], ndmin = 2).T
    x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
    rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_finite(A, c, B, x, 10,␣
 ↪delta = delta, periods = 150)
    plt.plot(range(3, 20), [rx.item() for rx in rs][3:20], label = f"OS (delta␣
 ↪= {delta})")
plt.xlabel("Time")
plt.ylabel("r_t")
plt.title(f"r_t vs time (t = 3 to t = 20 in 150 periods)")
plt.legend()
plt.show()
```



Clearly $\delta = 1$ starts furthest from zero, increases sharpest, then stops short compared to the others.

[18]:
```
for delta in [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]:
    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],
```

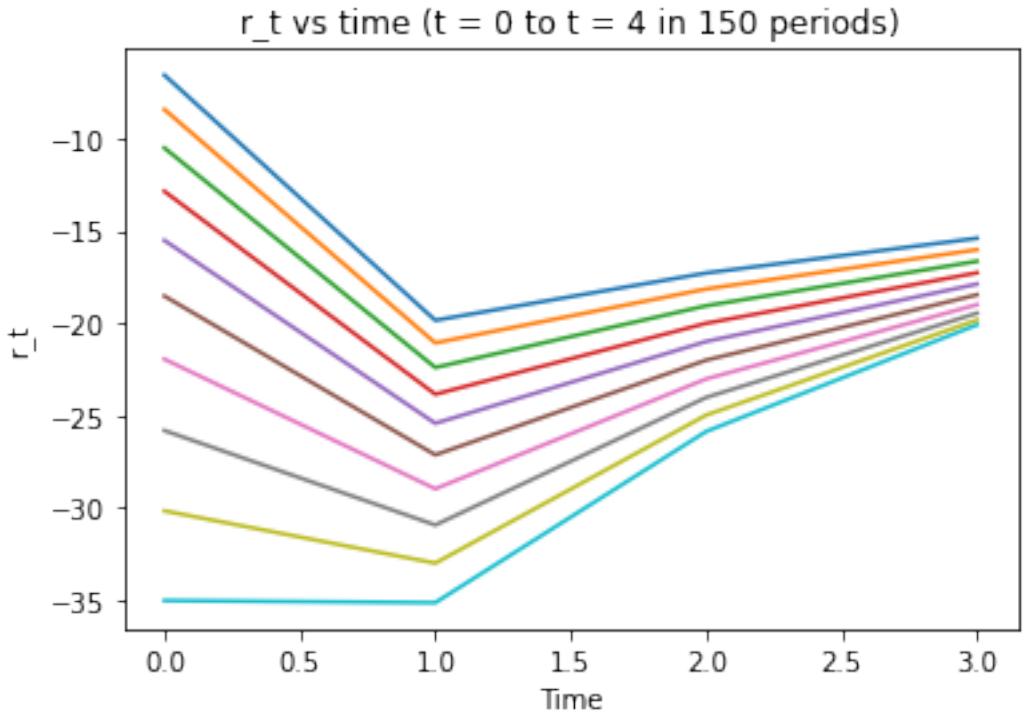
```
    ], ndmin = 2)
    c = np.array([0, 0.1012, 0, 0, 0,], ndmin = 2).T
    B = np.array([0.0791, 0, 0, 0, 0,], ndmin = 2).T
    x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
    rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_finite(A, c, B, x, 10,␣
 ↪delta = delta, periods = 150)
    plt.plot(range(13, 20), [rx.item() for rx in rs][13:20], label = f"OS␣
 ↪(delta = {delta})")
plt.xlabel("Time")
plt.ylabel("r_t")
plt.title(f"r_t vs time (t = 13 to t = 20 in 150 periods)")
plt.legend()
plt.show()
```



It is not a "singularity" in terms of when the trajectories intersect each other, but rather a series of points.

[19]:
```
for delta in [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]:
    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],
```

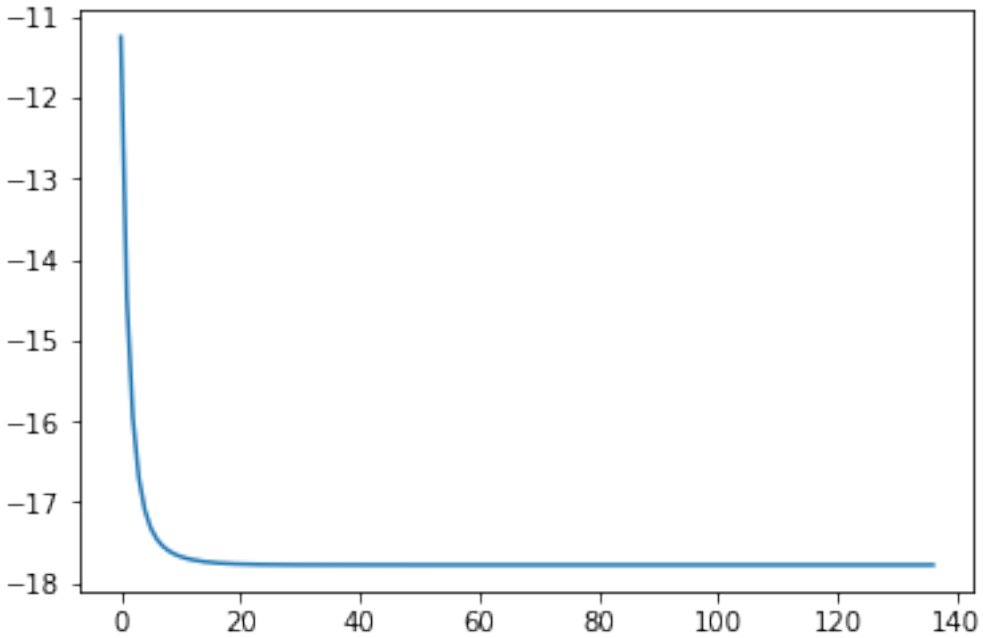
```
      [0.1256,   0.0711,   0.0253,   0.2244,   0.5536],
    ], ndmin = 2)
    c = np.array([0, 0.1012, 0, 0, 0,], ndmin = 2).T
    B = np.array([0.0791, 0, 0, 0, 0,], ndmin = 2).T
    x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
    rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_finite(A, c, B, x, 10,␣
 ↪delta = delta, periods = 150)
    plt.plot(range(4), [rx.item() for rx in rs][:4], label = f"OS (delta =␣
 ↪{delta})")
plt.xlabel("Time")
plt.ylabel("r_t")
plt.title(f"r_t vs time (t = 0 to t = 4 in 150 periods)")
plt.show()
```



Legend omitted for visibility, but the top line is $\delta = 0.1$ and the bottom is $\delta = 1$. Clearly $\delta = 1$ has no decrease at all i.e. is monotonoic.

3 Check 3: V_{ss} in the original example with $\delta = 0.8$ vs the optimal payoff

[20]:
```
delta = 0.8
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_infinite(A, c, B, x, 10,␣
 ↪delta = delta)
plt.plot(range(len(ps)), ps)
plt.show()
```



[21]:
```
ps[-1]
```

[21]:
```
-17.78040998236752
```

This is the payoff in the actual optimal strategy. Using the steady-state strategy, we compute:

[22]:
```
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, w_0_prime

mat, _, w_0 = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
w_0
```

[22]:
```
array([[-0.98     ],
       [-4.62     ],
       [ 2.74     ],
       [ 4.67     ],
       [ 2.15     ],
       [10.       ],
       [-6.8384501]])
```

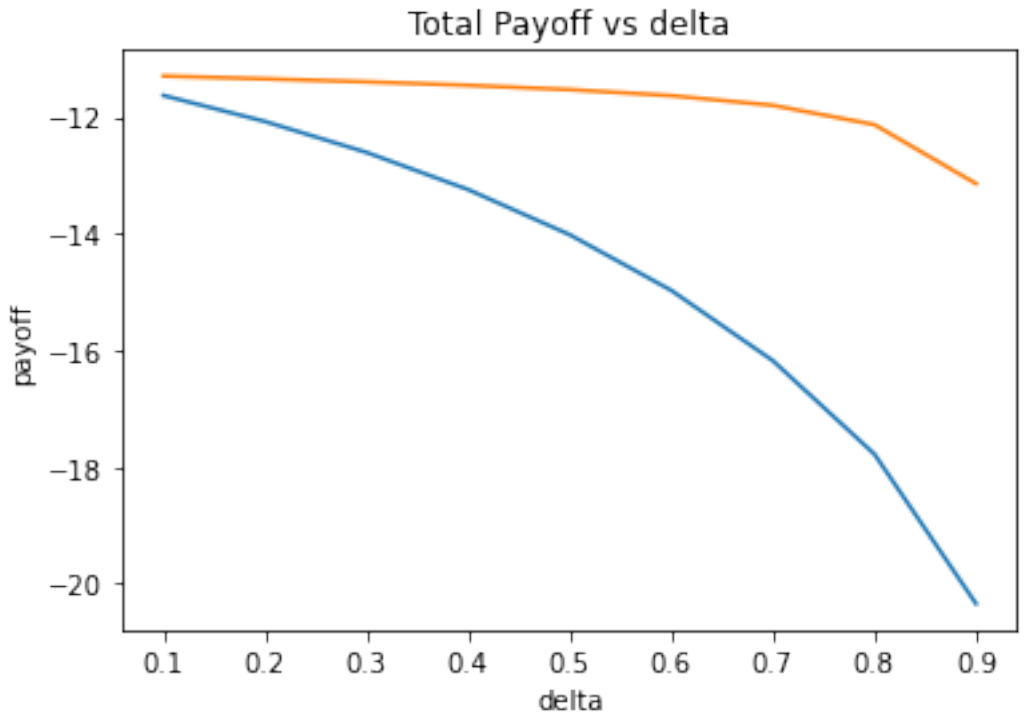
[23]:
```
Q_tilde = 0.2 * np.identity(7)
Q_tilde[5, :] = 0
Q_tilde[6, :] = 0
payoff = sum(-delta**i * (np.linalg.matrix_power(mat, i) @ w_0).T @ Q_tilde @␣
 ↪(np.linalg.matrix_power(mat, i) @ w_0) for i in range(100))
payoff_better_maybe = sum(-delta**i * (np.linalg.matrix_power(mat, i) @ w_0).T␣
 ↪@ Q_tilde @ (np.linalg.matrix_power(mat, i) @ w_0) for i in range(10000))
print(payoff)
print(payoff_better_maybe)
```

```
[[-12.1289437]]
[[-12.1289437]]
```

So the difference is about 5.66. If we vary delta, we get the following:

[24]:
```
px = []
px2 = []
for delta in [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]:
    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_infinite(A, c, B, x,␣
 ↪10, delta = delta, tol = 10**(-6))
    px.append(ps[-1])
    Q_tilde = 0.2 * np.identity(7)
    Q_tilde[5, :] = 0
    Q_tilde[6, :] = 0
    mat, _, w_0 = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
    payoff = 0
    for i in range(1000):
        payoff += -delta**i * (np.linalg.matrix_power(mat, i) @ w_0).T @␣
 ↪Q_tilde @ (np.linalg.matrix_power(mat, i) @ w_0)
    px2.append(payoff.item())

plt.plot([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9], px, label = "Optimal␣
 ↪Strategy")
plt.plot([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9], px2, label =␣
 ↪"Steady-State Strategy")
plt.xlabel("delta")
plt.ylabel("payoff")
plt.title("Total Payoff vs delta")
plt.show()
```



[25]:
```
print(px)
print(px2)
```

```
[-11.627862378468588, -12.074231032507072, -12.6047993262472,
-13.242021919439505, -14.016428242603599, -14.971668547836572,
-16.17766233871445, -17.780409982367495, -20.346814488879552]
[-11.295119260320305, -11.342932047410693, -11.39458132901215,
-11.45397955867353, -11.528181629551202, -11.63146636280623, -11.79758975542495,
-12.128943705069199, -13.141000858038417]
```

This is a one-to-one comparison of the optimal strategy payoff (top row of numbers) with the steady-state strategy payoff (bottom row of numbers).