StrategicInfluence3

June 21, 2021

1 Experimentation with Strategic Influence Network Model, Part 3

James Yu

20 June 2021

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

I assume here that A still needs to be diminished in row 2 to account for part of the second agent's opinion being influenced by the bot (to ensure everything sums up to 1).

[2]:
```
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

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)
A_tilde
```

[2]:
```
array([[0.217     , 0.2022    , 0.2358    , 0.1256    , 0.1403    ,
        0.        ],
       [0.22443036, 0.00961716, 0.20977992, 0.11522616, 0.3397464 ,
        0.1012    ],
       [0.1285    , 0.0907    , 0.3185    , 0.2507    , 0.2116    ,
```

```
        0.        ],
       [0.1975    , 0.0629    , 0.2863    , 0.2396    , 0.2137    ,
        0.        ],
       [0.1256    , 0.0711    , 0.0253    , 0.2244    , 0.5536    ,
        0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        1.        ]])
```

[3]:
```
B = np.array([
  0.0791,
  0,
  0,
  0,
  0,
], ndmin = 2).T
B_tilde = np.concatenate((B, np.array([0], ndmin = 2)), axis = 0)
B_tilde
```

[3]:
```
array([[0.0791],
       [0.    ],
       [0.    ],
       [0.    ],
       [0.    ],
       [0.    ]])
```

[4]:
```
x = np.array([
  -0.98,
  -4.62,
  2.74,
  4.67,
  2.15,
], ndmin = 2).T

z = 10

x_tilde = np.concatenate((x, np.array([z], ndmin = 2)), axis = 0)
x_tilde
```

[4]:
```
array([[-0.98],
       [-4.62],
       [ 2.74],
       [ 4.67],
       [ 2.15],
       [10.  ]])
```

[5]:
```
Q = 0.2 * np.identity(5)
Q_tilde = 0.2 * np.identity(6)
Q_tilde[5, :] = 0
Q_tilde
```

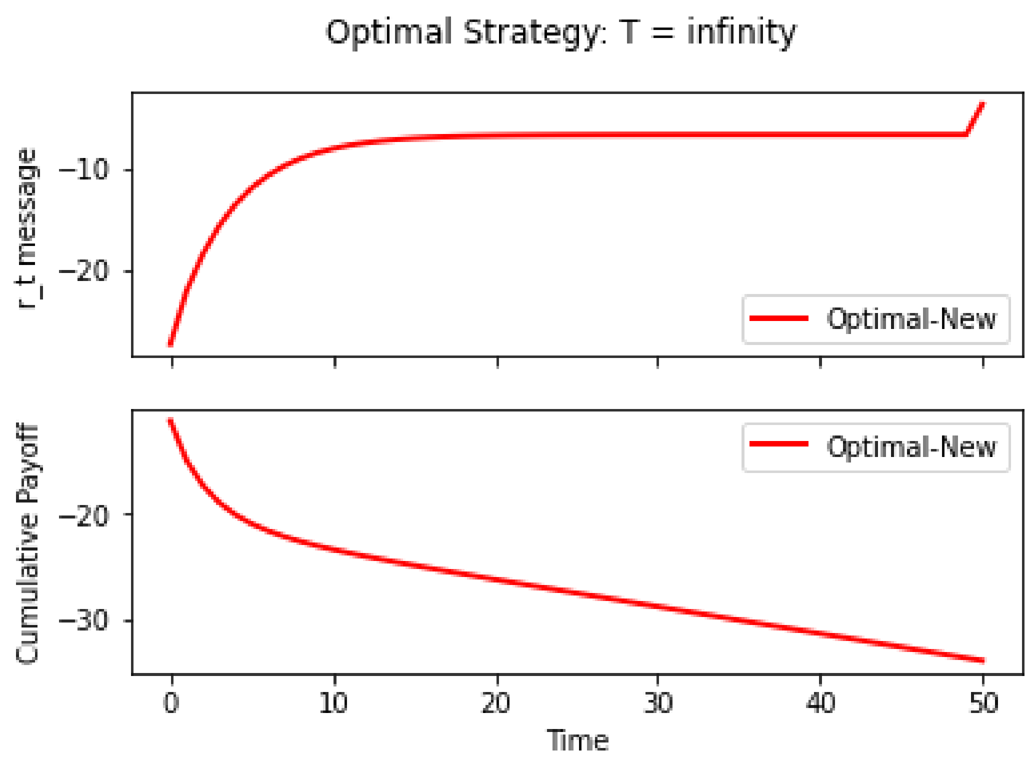
[5]:
```
array([[0.2, 0. , 0. , 0. , 0. , 0. ],
       [0. , 0.2, 0. , 0. , 0. , 0. ],
       [0. , 0. , 0.2, 0. , 0. , 0. ],
       [0. , 0. , 0. , 0.2, 0. , 0. ],
       [0. , 0. , 0. , 0. , 0.2, 0. ],
       [0. , 0. , 0. , 0. , 0. , 0. ]])
```

1.1 Testing finite horizon:

[6]:
```
K_t = [Q_tilde]
K = Q_tilde
i = 0
delta = 1
T = 50 # 50 periods
for i in range(T):
    K_new = delta * (A_tilde.T @ (K - (K @ B_tilde @ np.linalg.inv(B_tilde.T @␣
 ↪K @ B_tilde) @ B_tilde.T @ K)) @ A_tilde) + Q_tilde
    K_t.insert(0, K_new)
```

[7]:
```
def L(K_entry):
    return -1 * np.linalg.inv(B_tilde.T @ K_entry @ B_tilde) @ B_tilde.T @␣
 ↪K_entry @ A_tilde

x_t = x
x_ts = [x]
r_ts = []
payoffs = []
payoff = 0
for K_ent in K_t:
    expr = A_tilde + B_tilde @ L(K_ent)
    A_tilde_n = expr[:5, :5]
    c_nplus1 = np.array(expr[:5, 5], ndmin = 2).T
    payoff += (-1 * (x_t.T @ Q @ x_t)).item()
    payoffs.append(payoff)
    x_tp1 = A_tilde_n @ x_t + c_nplus1 * z
    x_ts.append(x_tp1)
    r_ts.append(L(K_ent) @ np.concatenate((x_tp1, np.array([z], ndmin = 2)),␣
 ↪axis = 0))
    x_t = x_tp1

fig, sub = plt.subplots(2, sharex=True)
fig.suptitle("Optimal Strategy: T = infinity")
sub[0].plot(range(len(K_t)), [a.item() for a in r_ts], 'r', label =␣
 ↪"Optimal-New", linewidth=2)
sub[0].set(ylabel = "r_t message")
sub[1].plot(range(len(K_t)), payoffs, 'r', label = "Optimal-New", linewidth=2)
sub[1].set(xlabel = "Time", ylabel =  "Cumulative Payoff")
```

```
sub[0].legend()
sub[1].legend()
plt.show()
```



[8]:
```
x_ts[-1]
```

[8]:
```
array([[-3.41473285e-17],
       [ 1.05619214e+00],
       [ 1.74093549e-01],
       [ 1.17325603e-01],
       [ 1.52184249e-01]])
```

Same results as before; the opinions also converge to some values that are not exactly zero, while the agent which listens to the strategic agent has an opinion which does converge to zero.

1.2 Testing Infinite Horizon

[9]:
```
K = np.zeros((6, 6)) # initial K

K_t = [Q_tilde, K] # saved K
K = Q_tilde
i = 0
delta = 1
```

```
while True:
    K_new = delta * (A_tilde.T @ (K - (K @ B_tilde @ np.linalg.inv(B_tilde.T @␣
 ↪K @ B_tilde) @ B_tilde.T @ K)) @ A_tilde) + Q_tilde
    K_t.insert(0, K_new)
    current_difference = np.max(np.abs(K - K_new))
    i += 1
    if i % 1000 == 0:
        print(i, current_difference)
        print("\n".join([str(list(k)) for k in K_new - K]))
        print()
        print("\n".join([str(list(k)) for k in K_new]))
        K = K_new
        break
    K = K_new
    if abs(current_difference) == 0:
        break
```

```
1000 0.0020672159198698026
[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, 0.0, 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, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0020672159198698026]

[0.24810209926345816, 0.017627731466811385, 0.056913765604162996,
0.06166393518681971, 0.0999712231528843, 0.013304623920956361]
[0.017627731466811385, 0.20820434787975997, 0.022380822442098925,
0.02658461038805592, 0.040438941026328015, 0.00421209102645786]
[0.056913765604162996, 0.022380822442098925, 0.27836151355163985,
0.07590845897143529, 0.10665301863266985, 0.013954229634764396]
[0.0616639351868197, 0.026584610388055915, 0.0759084589714353,
0.288234666260614, 0.1373620826064683, 0.015487697242314427]
[0.0999712231528843, 0.04043894102632801, 0.10665301863266984,
0.1373620826064683, 0.4413769762411032, 0.028915667835081205]
[0.01330462392095636, 0.004212091026457859, 0.013954229634764392,
0.015487697242314425, 0.028915667835081205, 2.067492543259903]
```

Most of the matrix converges except for the entry corresponding to that of the bot.

1.3 Testing Limit Matrix for Delta = 1 Case

[10]:
```
expr = A_tilde + B_tilde @ L(K_t[0])
print(expr)
```

```
[[-0.14512021 -0.06577221 -0.16931995 -0.21566798 -0.34886268 -0.06081589]
 [ 0.22443036  0.00961716  0.20977992  0.11522616  0.3397464   0.1012    ]
```

```
[ 0.1285     0.0907     0.3185     0.2507     0.2116     0.         ]
[ 0.1975     0.0629     0.2863     0.2396     0.2137     0.         ]
[ 0.1256     0.0711     0.0253     0.2244     0.5536     0.         ]
[ 0.         0.         0.         0.         0.         1.         ]]
```

[11]:
```
print(np.linalg.matrix_power(expr, 2))
```

```
[[-1.01870595e-01 -4.48146368e-02 -1.13726313e-01 -1.48688240e-01
  -2.46765572e-01 -5.86464237e-02]
 [ 6.19750349e-02  3.57619428e-02  7.24166878e-02  1.09144812e-01
   1.82068887e-01  8.85243242e-02]
 [ 1.18725346e-01  5.21222878e-02  1.75840565e-01  1.70136387e-01
   2.24097094e-01  1.36399794e-03]
 [ 9.64066973e-02  4.38472284e-02  1.44945107e-01  1.41791150e-01
   1.82557589e-01 -5.64565857e-03]
 [ 1.14832110e-01  4.81932208e-02  7.99586166e-02  1.65441472e-01
   3.40119536e-01 -4.43155970e-04]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  1.00000000e+00]]
```

[12]:
```
print(np.linalg.matrix_power(expr, 1000000000000))
```

```
[[ 0.          0.          0.          0.          0.         -0.05327854]
 [ 0.          0.          0.          0.          0.          0.0855562 ]
 [ 0.          0.          0.          0.          0.         -0.00463246]
 [ 0.          0.          0.          0.          0.         -0.01043657]
 [ 0.          0.          0.          0.          0.         -0.00687255]
 [ 0.          0.          0.          0.          0.          1.        ]]
```

The last column of the limit matrix converges to numbers that are 1/10th of the numbers which appear in the following tests. (This is because $z = 10$.)

[13]:
```
evals, evecs = np.linalg.eig((A_tilde + B_tilde @ L(K_t[0]))) # transpose for␣
↪left eigenvectors
np.array(evals, ndmin = 2).T
```

[13]:
```
array([[ 6.91882113e-01],
       [-7.01142129e-02],
       [-3.64430099e-16],
       [ 2.68553466e-02],
       [ 3.27573700e-01],
       [ 1.00000000e+00]])
```

[14]:
```
print(evals)
```

```
[ 6.91882113e-01 -7.01142129e-02 -3.64430099e-16  2.68553466e-02
  3.27573700e-01  1.00000000e+00]
```

[15]:
```
print(evecs)
```

```
[[ 0.44953821  0.0130008  -0.27322607 -0.00262916  0.04625751 -0.05300531]
 [-0.33352732  0.9651972  -0.90218936  0.7778138   0.03071561  0.08511744]
 [-0.49704666 -0.20935885  0.08623623  0.16561802 -0.62521245 -0.00460871]
 [-0.41067296  0.0812542   0.31935518 -0.59010781 -0.53486765 -0.01038304]
 [-0.52054516 -0.13338655  0.04446853  0.13907634  0.56563485 -0.0068373 ]
 [ 0.          0.          0.          0.          0.          0.99487168]]
```

The COLUMNS are the eigenvectors corresponding to the i-th entry in `evals`. Commentary on the eigenvector of the 1 eigenvalue comes later.

1.4 Test of Steady-State x_{ss} Formula

[16]:
```
def L(K_entry):
    return -1 * np.linalg.inv(B_tilde.T @ K_entry @ B_tilde) @ B_tilde.T @␣
 ↪K_entry @ A_tilde

expr = A_tilde + B_tilde @ L(K_t[0])
print(expr)
A_tilde_n = expr[:5, :5]
c_nplus1 = np.array(expr[:5, 5], ndmin = 2).T
x_ss = np.linalg.inv(np.identity(5) - A_tilde_n) @ (c_nplus1 * z)
x_ss
```

```
[[-0.14512021 -0.06577221 -0.16931995 -0.21566798 -0.34886268 -0.06081589]
 [ 0.22443036  0.00961716  0.20977992  0.11522616  0.3397464   0.1012    ]
 [ 0.1285      0.0907      0.3185      0.2507      0.2116      0.        ]
 [ 0.1975      0.0629      0.2863      0.2396      0.2137      0.        ]
 [ 0.1256      0.0711      0.0253      0.2244      0.5536      0.        ]
 [ 0.          0.          0.          0.          0.          1.        ]]
```

[16]:
```
array([[-0.53278541],
       [ 0.85556199],
       [-0.04632463],
       [-0.10436566],
       [-0.06872549]])
```

1.5 Test of x_{t+1} formula with L_{ss}

[17]:
```
x_t = x
x_ts = [x]
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
```

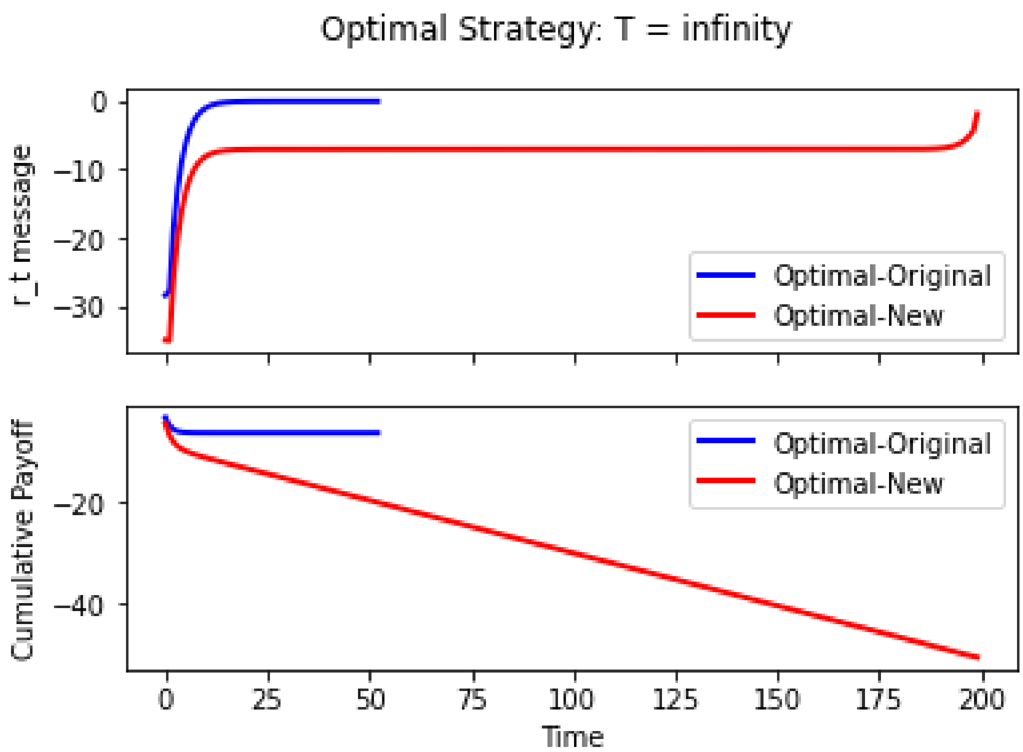
```
x_ts[-1]
```

[17]:
```
array([[-0.53278541],
       [ 0.85556199],
       [-0.04632463],
       [-0.10436566],
       [-0.06872549]])
```

Exactly the same result. We see the opinions converging to nonzero values.

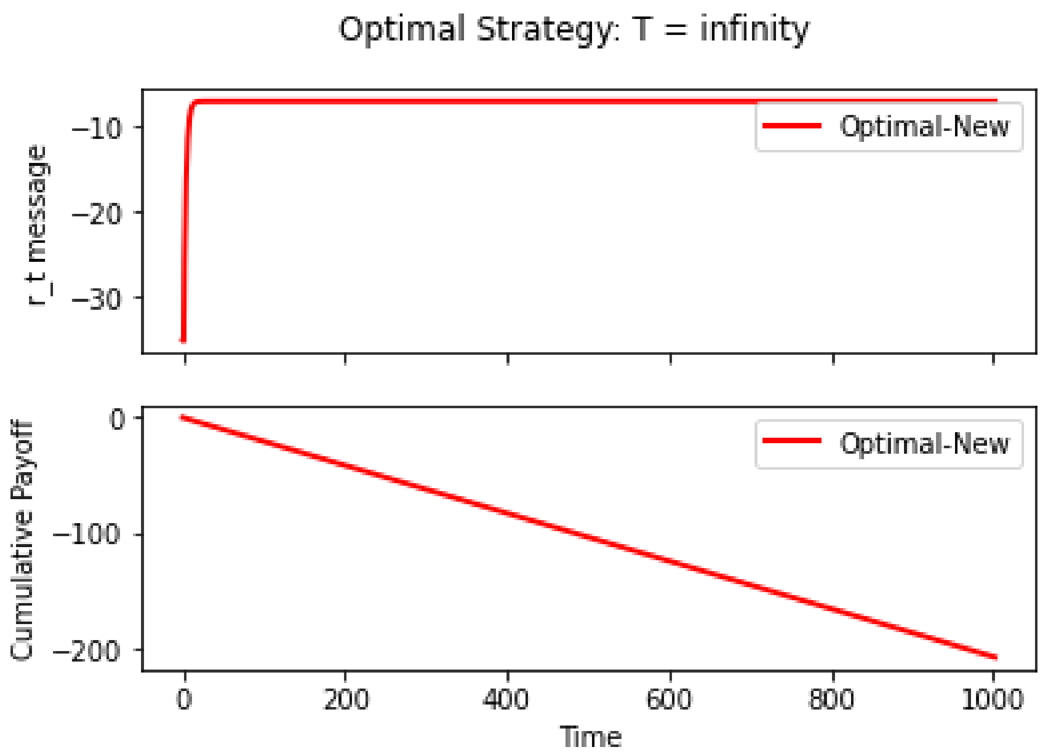
1.6 GRAPHS FROM PREVIOUS NOTEBOOK (saved from prior notebook, not run using current vars):

[16]:
```
fig, sub = plt.subplots(2, sharex=True)
fig.suptitle("Optimal Strategy: T = infinity")

sub[0].plot(range(old_length - 2), [a.item() for a in r_ts], 'b', label =␣
 ↪"Optimal-Original", linewidth=2)
sub[0].plot(range(len(K_t) - 2), [a.item() for a in r_ts2], 'r', label =␣
 ↪"Optimal-New", linewidth=2)
sub[0].set(ylabel = "r_t message")

sub[1].plot(range(old_length - 2), payoffs, 'b', label = "Optimal-Original",␣
 ↪linewidth=2)
sub[1].plot(range(len(K_t) - 2), payoffs2, 'r', label = "Optimal-New",␣
 ↪linewidth=2)
sub[1].set(xlabel = "Time", ylabel =  "Cumulative Payoff")

sub[0].legend()
sub[1].legend()
plt.show()
```



1.7 NEW GRAPHS

[18]:
```
payoff = 0
payoffs = []
r_ts = []
for x_ent in x_ts:
    r_ts.append(L(K_t[0]) @ np.concatenate((x_ent, np.array([z], ndmin = 2)),␣
 ↪axis = 0))
    payoff += (-1 * (x_t.T @ Q @ x_t)).item()
    payoffs.append(payoff)

fig, sub = plt.subplots(2, sharex=True)
fig.suptitle("Optimal Strategy: T = infinity")
sub[0].plot(range(len(K_t)+1), [a.item() for a in r_ts], 'r', label =␣
 ↪"Optimal-New", linewidth=2)
sub[0].set(ylabel = "r_t message")
sub[1].plot(range(len(K_t)+1), payoffs, 'r', label = "Optimal-New", linewidth=2)
sub[1].set(xlabel = "Time", ylabel =  "Cumulative Payoff")

sub[0].legend()
sub[1].legend()
plt.show()
```



(exactly the same, and the tail has been removed due to the use of the steady-state values)

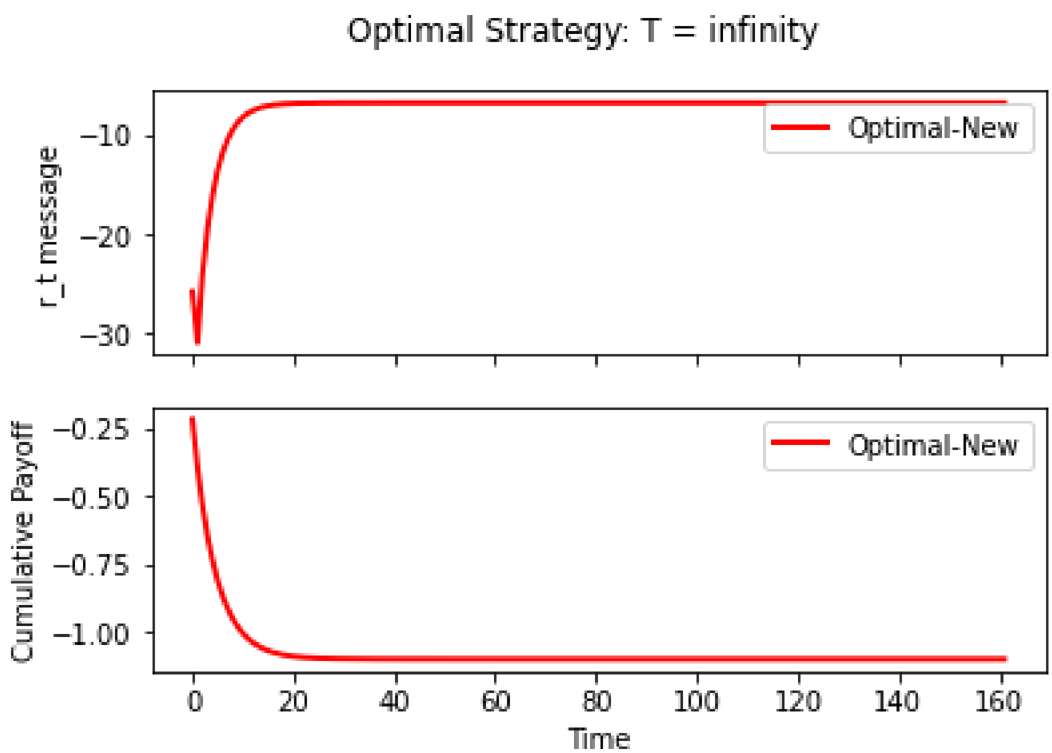
1.8 Testing delta = 0.8

[19]:
```
K = np.zeros((6, 6)) # initial K

K_t = [Q_tilde, K] # saved K
K = Q_tilde
i = 0
delta = 0.8
while True:
    K_new = delta * (A_tilde.T @ (K - (K @ B_tilde @ np.linalg.inv(B_tilde.T @
 ↪K @ B_tilde) @ B_tilde.T @ K)) @ A_tilde) + Q_tilde
    K_t.insert(0, K_new)
    current_difference = np.max(np.abs(K - K_new))
    i += 1
    if abs(current_difference) == 0:
        print(i, current_difference)
        print("\n".join([str(list(k)) for k in K_new - K]))
        print()
        print("\n".join([str(list(k)) for k in K_new]))
        K = K_new
        break
```

```
    K = K_new

expr = A_tilde + B_tilde @ L(K_t[0])
print(expr)
A_tilde_n = expr[:5, :5]
c_nplus1 = np.array(expr[:5, 5], ndmin = 2).T
x_t = x
x_ts = [x]
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

payoff = 0
payoffs = []
r_ts = []
i = 0
for x_ent in x_ts:
    r_ts.append(L(K_t[0]) @ np.concatenate((x_ent, np.array([z], ndmin = 2)),
 axis = 0))
    payoff += (-1 * delta**i * (x_t.T @ Q @ x_t)).item() # account for
 discounting
    payoffs.append(payoff)
    i += 1

fig, sub = plt.subplots(2, sharex=True)
fig.suptitle("Optimal Strategy: T = infinity")
sub[0].plot(range(len(K_t)+1), [a.item() for a in r_ts], 'r', label =
 "Optimal-New", linewidth=2)
sub[0].set(ylabel = "r_t message")
sub[1].plot(range(len(K_t)+1), payoffs, 'r', label = "Optimal-New", linewidth=2)
sub[1].set(xlabel = "Time", ylabel =  "Cumulative Payoff")

sub[0].legend()
sub[1].legend()
plt.show()
```

```
159 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, 0.0, 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, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

[0.23288121495542177, 0.01162188727872566, 0.03920054359933627,
0.04112485133115336, 0.06645613438193713, 0.007815234804637894]
```

```
[0.01162188727872566, 0.2054624371843706, 0.015093894226105013,
0.017628032270041786, 0.026362717006696942, 0.002102901128977268]
[0.039200543599336266, 0.01509389422610501, 0.25531354382135224,
0.05144869564270399, 0.07033562978118012, 0.008081989867727929]
[0.04112485133115336, 0.017628032270041782, 0.05144869564270399,
0.2585514504692229, 0.09005815719588628, 0.008206971139772609]
[0.06645613438193713, 0.026362717006696942, 0.0703356297811801,
0.09005815719588628, 0.3600163594823032, 0.016070731124581312]
[0.007815234804637895, 0.0021029011289772684, 0.00808198986772793,
0.00820697113977261, 0.01607073112458132, 0.008730833351721671]
[[-0.10354903 -0.04714439 -0.12185954 -0.15429751 -0.24828892 -0.03860925]
 [ 0.22443036  0.00961716  0.20977992  0.11522616  0.3397464   0.1012    ]
 [ 0.1285      0.0907      0.3185      0.2507      0.2116      0.        ]
 [ 0.1975      0.0629      0.2863      0.2396      0.2137      0.        ]
 [ 0.1256      0.0711      0.0253      0.2244      0.5536      0.        ]
 [ 0.          0.          0.          0.          0.          1.        ]]
```



1.9 Testing limit matrix for Delta = 0.8 case

[20]:
```
expr = A_tilde + B_tilde @ L(K_t[0])
print(expr)
print()
```

```
print(np.linalg.matrix_power(expr, 1000000000000))
```

```
[[-0.10354903 -0.04714439 -0.12185954 -0.15429751 -0.24828892 -0.03860925]
 [ 0.22443036  0.00961716  0.20977992  0.11522616  0.3397464   0.1012    ]
 [ 0.1285      0.0907      0.3185      0.2507      0.2116      0.        ]
 [ 0.1975      0.0629      0.2863      0.2396      0.2137      0.        ]
 [ 0.1256      0.0711      0.0253      0.2244      0.5536      0.        ]
 [ 0.          0.          0.          0.          0.          1.        ]]

[[ 0.          0.          0.          0.          0.         -0.04110785]
 [ 0.          0.          0.          0.          0.          0.09612265]
 [ 0.          0.          0.          0.          0.          0.00697612]
 [ 0.          0.          0.          0.          0.          0.00123909]
 [ 0.          0.          0.          0.          0.          0.00476192]
 [ 0.          0.          0.          0.          0.          1.        ]]
```

[21]:
```
x_ts[-1]
```

[21]:
```
array([[-0.41107847],
       [ 0.96122649],
       [ 0.06976123],
       [ 0.01239086],
       [ 0.04761921]])
```

Once again, this is the top right of the limit matrix.

[22]:
```
evals, evecs = np.linalg.eig(expr)
np.array(evals, ndmin = 2).T
```

[22]:
```
array([[ 7.33661311e-01],
       [-7.04763999e-02],
       [-1.22864623e-16],
       [ 2.68277770e-02],
       [ 3.27755446e-01],
       [ 1.00000000e+00]])
```

[23]:
```
print(evecs)
```

```
[[ 0.33668307  0.01050697 -0.27322607 -0.00209693  0.03431352 -0.04088356]
 [-0.36756243  0.96515722 -0.90218936  0.77811541  0.03117031  0.09559819]
 [-0.51717178 -0.20923606  0.08623623  0.16525945 -0.62143601  0.00693806]
 [-0.44370195  0.08256941  0.31935518 -0.58987342 -0.53405618  0.00123233]
 [-0.53591445 -0.13328057  0.04446853  0.13881882  0.57136003  0.00473594]
 [ 0.          0.          0.          0.          0.          0.99454383]]
```

The COLUMNS are the eigenvectors corresponding to the i-th eigenvalue in line 36.

Note the last column being approximately 1/10th the limit steady state opinions, with slight errors.

[24]:
```
expr @ evecs[:, 5]
```

[24]:
```
array([-0.04088356,  0.09559819,  0.00693806,  0.00123233,  0.00473594,
        0.99454383])
```

The same result.

1.9.1 Verifying limit behaviour:

[25]:
```
r = np.linalg.matrix_power(expr, 1000000000000) @ np.concatenate((x, np.
 ↪array([10], ndmin = 2)), axis = 0)
r
```

[25]:
```
array([[-0.41107847],
       [ 0.96122649],
       [ 0.06976123],
       [ 0.01239086],
       [ 0.04761921],
       [10.        ]])
```

These are the same steady-state opinions.