Opposite_Bias_FOC

November 22, 2021

1 Opposite Bias: FOC

James Yu, 22 November 2021

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

[2]:
```
def M(K, B, R, L, delta):
    """Computes M_{t-1} given B_l \forall l, K_t^l \forall l,
        R_l \forall l, number of strategic agents L, and delta."""
    # handle the generic structure first, with the correct pairings:
    base = [[(B[l_prime].T @ K[l_prime] @ B[l]).item() for l in range(L)] for␣
↪l_prime in range(L)]
    # then change the diagonals to construct M_{t-1}:
    for l in range(L): base[l][l] = (B[l].T @ K[l] @ B[l] + R[l]/delta).item()
    return np.array(base, ndmin = 2)

def H(B, K, A, L):
    """Computes H_{t-1} given B_l \forall l, K_t^l \forall l,
        A, and number of strategic agents L."""
    return np.concatenate(tuple(B[l].T @ K[l] @ A for l in range(L)), axis = 0)

def C_l(B, K, k, h, L, c, x, n):
    """Computes C_{t-1}^h (displayed as C_{t-1}^l) given B_l \forall l, K_t^l␣
↪\forall l,
        k_t^l \forall l, a specific naive agent h, number of strategic agents␣
↪L,
        c_l \forall l, x_l \forall l, and number of naive agents n"""
    return np.concatenate(tuple(B[l].T @ K[l] @ A @ ((x[h] - x[l]) * np.
↪ones((n, 1)))
                                + B[l].T @ K[l] @ c[l]
                                + 0.5 * B[l].T @ k[l].T for l in range(L)), axis = 0)

def E(M_, H_):
    """Computes the generic E_{t-1} given M_{t-1} and H_{t-1}."""
    return np.linalg.inv(M_) @ H_
```

```
def F(M_, C_l_, l):
    """Computes F_{t-1}^l given M_{t-1}, C_{t-1}^l, and specific naive agent l.
 ↪"""
    return (np.linalg.inv(M_) @ C_l_)[l:l+1, :]

def G(A, B, E_, L):
    """Computes the generic G_{t-1} given A, B_l \forall l,
        E_{t-1}, and number of strategic agents L."""
    return A - sum([B[l] @ E_[l:l+1, :] for l in range(L)])

def g_l(B, E_, h, x, F_, L):
    """Computes g_{t-1}^l given B_l \forall l, E_{t-1}^l,
        a particular naive agent h, x_l \forall l, F_{t-1}^l \forall l,
        number of strategic agents L, number of naive agents n, and c_h."""
    return - sum([B[l] @ (E_[l:l+1, :] @ ((x[h] - x[l]) * np.ones((n, 1))) +␣
 ↪F_[l]) for l in range(L)]) + c[h]
```

[3]:
```
def K_t_minus_1(Q, K, E_, R, G_, L, delta):
    return [Q[l] + E_[l:l+1, :].T @ R[l] @ E_[l:l+1, :]
            + delta * G_.T @ K[l] @ G_ for l in range(L)]

def k_t_minus_1(K, k, G_, g, E_, F_, R, L, delta):
    return [2*delta* g[l].T @ K[l] @ G_ + delta * k[l] @ G_
            + 2 * F_[l].T @ R[l] @ E_[l:l+1, :] for l in range(L)]

def kappa_t_minus_1(K, k, kappa, g_, F_, R, L, delta):
    return [-delta * (g_[l].T @ K[l] @ g_[l] + k[l] @ g_[l] - kappa[l])
            - (F_[l].T @ R[l] @ F_[l]) for l in range(L)]
```

[4]:
```
def solve_finite(K_t, k_t, kappa_t, A, B, delta, n, m, L, Q, R, x, c, T):
    historical_K = [K_t]
    historical_k = [k_t]
    historical_kappa = [kappa_t]
    max_distances = defaultdict(list)
    counter = 0
    for i in range(T):
        M_ = M(K_t, B, R, L, delta)
        H_ = H(B, K_t, A, L)
        E_ = E(M_, H_)
        G_ = G(A, B, E_, L)
        K_new = K_t_minus_1(Q, K_t, E_, R, G_, L, delta)
        F_ = [F(M_, C_l(B, K_t, k_t, l, L, c, x, n), l) for l in range(L)]
        g = [g_l(B, E_, h, x, F_, L) for h in range(L)]
        k_new = k_t_minus_1(K_t, k_t, G_, g, E_, F_, R, L, delta)
        kappa_new = kappa_t_minus_1(K_t, k_t, kappa_t, g, F_, R, L, delta)
        cd_K = [np.max(np.abs(K_t[l] - K_new[l])) for l in range(L)]
```

```
        cd_k = [np.max(np.abs(k_t[l] - k_new[l])) for l in range(L)]
        cd_kappa = [np.max(np.abs(kappa_t[l] - kappa_new[l])) for l in range(L)]
        K_t = K_new
        k_t = k_new
        kappa_t = kappa_new
        historical_K.insert(0, K_t)
        historical_k.insert(0, k_t)
        historical_kappa.insert(0, kappa_t)
        for l in range(L):
            max_distances[(l+1, "K")].append(cd_K[l])
            max_distances[(l+1, "k")].append(cd_k[l])
            max_distances[(l+1, "kappa")].append(cd_kappa[l])
        counter += 1

    return max_distances, historical_K, historical_k, historical_kappa
```

[5]:
```
def optimal(X_init, historical_K, historical_k, historical_kappa):
    X_t = [a.copy() for a in X_init]
    xs = defaultdict(list)
    for l in range(L):
        xs[l].append(X_t[l])

    rs = defaultdict(list)
    payoffs = defaultdict(list)
    payoff = defaultdict(lambda: 0)
    i = 0
    while i < len(historical_K):
        K_t = historical_K[i]
        k_t = historical_k[i]
        M_ = M(K_t, B, R, L, delta)
        H_ = H(B, K_t, A, L)
        E_ = E(M_, H_)
        G_ = G(A, B, E_, L)
        F_ = [F(M_, C_l(B, K_t, k_t, l, L, c, x, n), l) for l in range(L)]
        g = [g_l(B, E_, h, x, F_, L) for h in range(L)]
        for l in range(L):
            Y_new = -1 * E_[l:l+1, :] @ X_t[l] - F(M_, C_l(B, K_t, k_t, l, L,␣
↪c, x, n), l)
            rs[l].append(Y_new)
            payoff[l] += (-1 * delta**i * (X_t[l].T @ Q[l] @ X_t[l])).item() +␣
↪(-1 * delta**i * (Y_new.T @ R[l] @ Y_new)).item()
            payoffs[l].append(payoff[l])
            X_new = G_ @ X_t[l] + g[l]
            xs[l].append(X_new)
            X_t[l] = X_new
        i += 1
```

```
    return xs, rs, payoffs
```

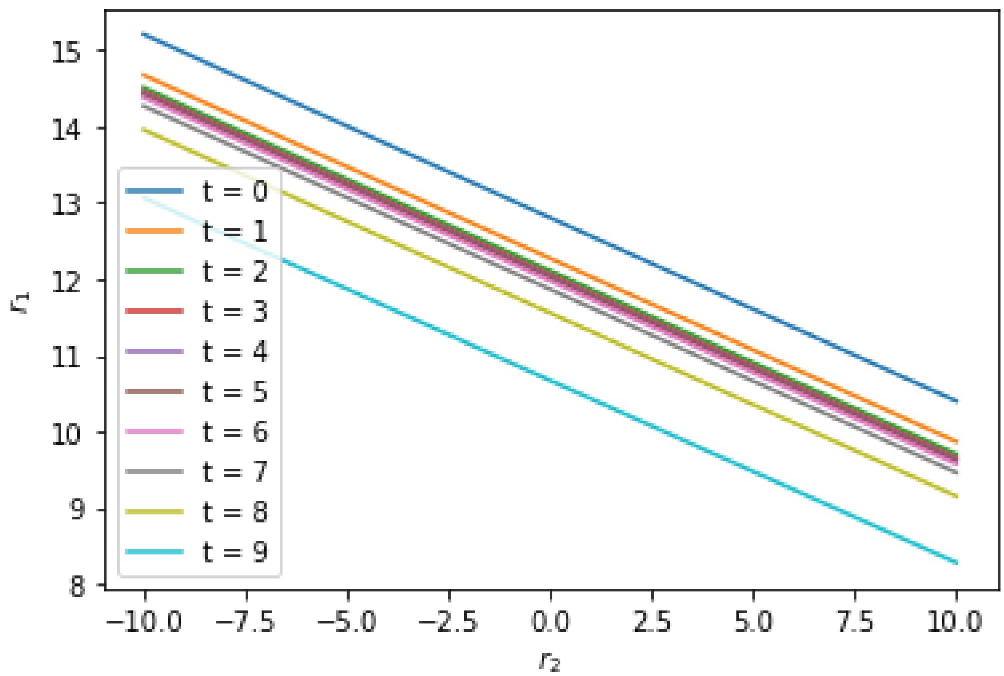
2 1. $r_1 = r_2 = 0$

[11]:
```
A = np.array([
  [0.5],
], ndmin = 2)

B_1 = np.array([
  0.25,
], ndmin = 2).T

B_2 = np.array([
  0.25,
], ndmin = 2).T

B = [B_1, B_2]

x0 = 0

X_0_1 = np.array([ # \chi_0^1
  x0 - 10, # agenda here is 10
], ndmin = 2).T

X_0_2 = np.array([ # \chi_0^2
  x0 + 5, # agenda here is -5
], ndmin = 2).T
X_0 = [X_0_1, X_0_2]

delta = 0.9
n = 1
m = 1
L = 2
Q = [1 * np.identity(n), 1 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [10, -5]
r = [0, 0]
c_base = sum([B[l] @ np.array([[r[l]]], ndmin = 2) for l in range(L)])
c = [c_base + (A - np.identity(n)) @ (x[l] * np.ones((n, 1))) for l in range(L)]
max_distances, historical_K, historical_k, historical_kappa = solve_finite(Q,␣
↪[np.zeros((1, n)), np.zeros((1, n))], [0, 0], A, B, delta, n, m, L, Q, R, x,␣
↪c, 10) # 10 periods
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
```

[13]:
```
historical_K
```

[13]: [[array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.12498245]]), array([[1.12498245]])],
 [array([[1.12498154]]), array([[1.12498154]])],
 [array([[1.12496399]]), array([[1.12496399]])],
 [array([[1.12462445]]), array([[1.12462445]])],
 [array([[1.11808]]), array([[1.11808]])],
 [array([[1.]]), array([[1.]])]]

[20]:
```
def find_r1(r2, t):
    B1 = 0.25
    B2 = 0.25
    A = 0.5
    K = historical_K
    k = historical_k
    R = 0.2
    delta = 0.9
    chi_t = xs[0][t].item() # this variable contains \chi = x - 10 and is␣
 ↪actually \chi_{t-1} because the first entry is \chi_0
    RIGHT_HAND_SIDE = -1 * B1 * K[t][0].item() * A * chi_t - B1 * K[t][0].
 ↪item() * B2 * r2 - B1 * K[t][0].item() * c[0] - 0.5 * B1 * k[t][0].item()
    return (RIGHT_HAND_SIDE / (B1 * K[t][0].item() * B1 + R / delta)).item()

for t in range(10):
    grid = np.linspace(-10, 10, 100)
    plt.plot(grid, [find_r1(a, t) for a in grid], label = f"t = {t}")
plt.xlabel("$r_2$")
plt.ylabel("$r_1$")
plt.legend()
plt.show()
```



At $t = 0$ for example, r_1 as a function of r_2 gives the following coordinate pairs:

[25]:
```
print([(round(a, 4), round(find_r1(a, 0), 4)) for a in np.linspace(-10, 10,␣
↪20)])
```

```
[(-10.0, 15.2091), (-8.9474, 14.9561), (-7.8947, 14.7031), (-6.8421, 14.4501),
(-5.7895, 14.1971), (-4.7368, 13.9441), (-3.6842, 13.6911), (-2.6316, 13.4381),
(-1.5789, 13.1851), (-0.5263, 12.9321), (0.5263, 12.6791), (1.5789, 12.4261),
(2.6316, 12.1731), (3.6842, 11.9201), (4.7368, 11.6671), (5.7895, 11.4141),
(6.8421, 11.1611), (7.8947, 10.9081), (8.9474, 10.6551), (10.0, 10.402)]
```

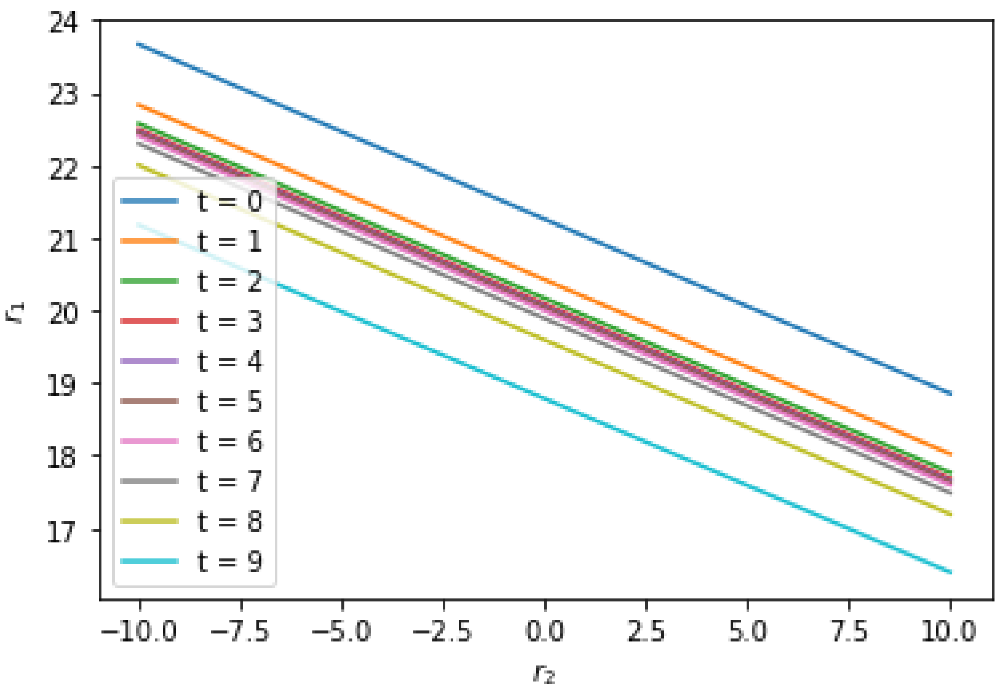
3 2. cost $= r_1, r_2$

[26]:
```
A = np.array([
  [0.5],
], ndmin = 2)

B_1 = np.array([
  0.25,
], ndmin = 2).T

B_2 = np.array([
  0.25,
], ndmin = 2).T
```

```
B = [B_1, B_2]

x0 = 0

X_0_1 = np.array([ # \chi_0^1
  x0 - 10, # agenda here is 10
], ndmin = 2).T

X_0_2 = np.array([ # \chi_0^2
  x0 + 5, # agenda here is -5
], ndmin = 2).T
X_0 = [X_0_1, X_0_2]

delta = 0.9
n = 1
m = 1
L = 2
Q = [1 * np.identity(n), 1 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [10, -5]
r = [10, -5]
c_base = sum([B[l] @ np.array([[r[l]]], ndmin = 2) for l in range(L)])
c = [c_base + (A - np.identity(n)) @ (x[l] * np.ones((n, 1))) for l in range(L)]
max_distances, historical_K, historical_k, historical_kappa = solve_finite(Q,␣
↪[np.zeros((1, n)), np.zeros((1, n))], [0, 0], A, B, delta, n, m, L, Q, R, x,␣
↪c, 10) # 10 periods
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
```

[30]:
```
def find_r1(r2, t):
    gamma_2 = r2 - (-5) # convert from r2 to gamma
    B1 = 0.25
    B2 = 0.25
    A = 0.5
    K = historical_K
    k = historical_k
    R = 0.2
    delta = 0.9
    chi_t = xs[0][t].item() # this variable contains \chi = x - 10 and is␣
↪actually \chi_{t-1} because the first entry is \chi_0
    RIGHT_HAND_SIDE = -1 * B1 * K[t][0].item() * A * chi_t - B1 * K[t][0].
↪item() * B2 * r2 - B1 * K[t][0].item() * c[0] - 0.5 * B1 * k[t][0].item()
    return (RIGHT_HAND_SIDE / (B1 * K[t][0].item() * B1 + R / delta)).item() +␣
↪10 # convert from gamma to r1

for t in range(10):
```

```
    grid = np.linspace(-10, 10, 100)
    plt.plot(grid, [find_r1(a, t) for a in grid], label = f"t = {t}")
plt.xlabel("$r_2$")
plt.ylabel("$r_1$")
plt.legend()
plt.show()
```



It is still downward sloping.

[28]:
```
print([(round(a, 4), round(find_r1(a, 0), 4)) for a in np.linspace(-10, 10,␣
 ↪20)])
```

```
[(-10.0, 23.664), (-8.9474, 23.411), (-7.8947, 23.158), (-6.8421, 22.905),
(-5.7895, 22.652), (-4.7368, 22.399), (-3.6842, 22.146), (-2.6316, 21.893),
(-1.5789, 21.64), (-0.5263, 21.387), (0.5263, 21.134), (1.5789, 20.881),
(2.6316, 20.628), (3.6842, 20.375), (4.7368, 20.122), (5.7895, 19.869), (6.8421,
19.616), (7.8947, 19.363), (8.9474, 19.11), (10.0, 18.857)]
```

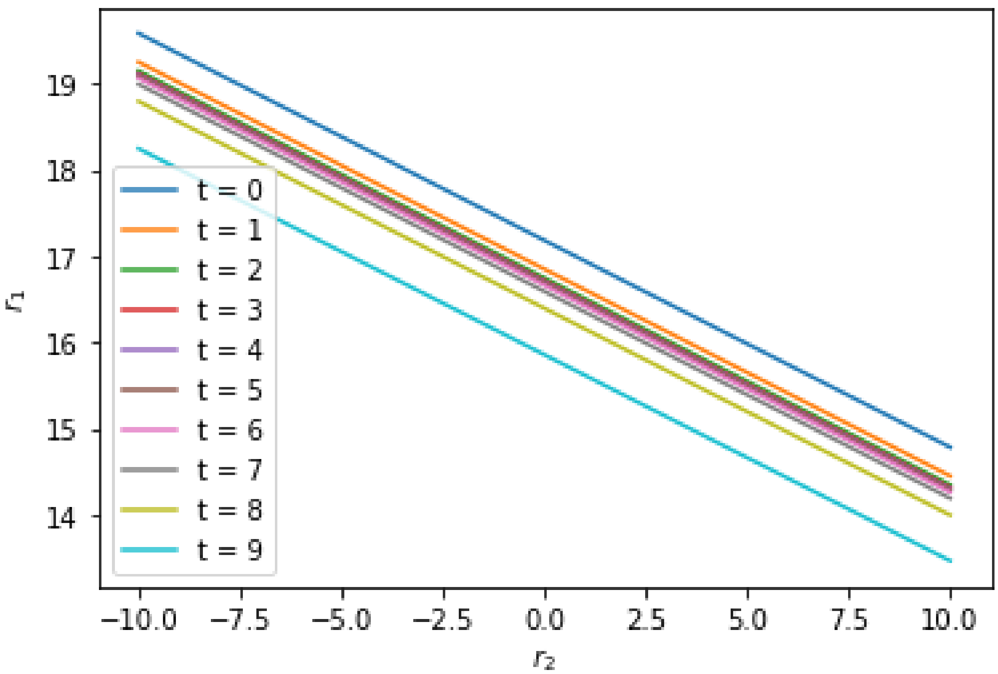
4 3. set $x_0 = 4$ and $x_2 = 0$

[32]:
```
A = np.array([
   [0.5],
], ndmin = 2)
```

```
B_1 = np.array([
  0.25,
], ndmin = 2).T

B_2 = np.array([
  0.25,
], ndmin = 2).T

B = [B_1, B_2]

x0 = 4

X_0_1 = np.array([ # \chi_0^1
  x0 - 10, # agenda here is 10
], ndmin = 2).T

X_0_2 = np.array([ # \chi_0^2
  x0 + 0, # agenda here is 0
], ndmin = 2).T
X_0 = [X_0_1, X_0_2]

delta = 0.9
n = 1
m = 1
L = 2
Q = [1 * np.identity(n), 1 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [10, 0]
r = [10, 0] # now add cost dependent on the agenda
c_base = sum([B[l] @ np.array([[r[l]]], ndmin = 2) for l in range(L)])
c = [c_base + (A - np.identity(n)) @ (x[l] * np.ones((n, 1))) for l in range(L)]
max_distances, historical_K, historical_k, historical_kappa = solve_finite(Q,
 [np.zeros((1, n)), np.zeros((1, n))], [0, 0], A, B, delta, n, m, L, Q, R, x,
 c, 10)
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
```

[33]:
```
def find_r1(r2, t):
    gamma_2 = r2 - (0) # convert from r2 to gamma
    B1 = 0.25
    B2 = 0.25
    A = 0.5
    K = historical_K
    k = historical_k
    R = 0.2
    delta = 0.9
```

```
    chi_t = xs[0][t].item() # this variable contains \chi = x - 10 and is␣
↪actually \chi_{t-1} because the first entry is \chi_0
    RIGHT_HAND_SIDE = -1 * B1 * K[t][0].item() * A * chi_t - B1 * K[t][0].
↪item() * B2 * r2 - B1 * K[t][0].item() * c[0] - 0.5 * B1 * k[t][0].item()
    return (RIGHT_HAND_SIDE / (B1 * K[t][0].item() * B1 + R / delta)).item() +␣
↪10 # convert from gamma to r1

for t in range(10):
    grid = np.linspace(-10, 10, 100)
    plt.plot(grid, [find_r1(a, t) for a in grid], label = f"t = {t}")
plt.xlabel("$r_2$")
plt.ylabel("$r_1$")
plt.legend()
plt.show()
```



[34]:
```
print([(round(a, 4), round(find_r1(a, 0), 4)) for a in np.linspace(-10, 10,␣
↪20)])
```

```
[(-10.0, 19.5901), (-8.9474, 19.3371), (-7.8947, 19.0841), (-6.8421, 18.8311),
(-5.7895, 18.5781), (-4.7368, 18.325), (-3.6842, 18.072), (-2.6316, 17.819),
(-1.5789, 17.566), (-0.5263, 17.313), (0.5263, 17.06), (1.5789, 16.807),
(2.6316, 16.554), (3.6842, 16.301), (4.7368, 16.048), (5.7895, 15.795), (6.8421,
15.542), (7.8947, 15.289), (8.9474, 15.036), (10.0, 14.783)]
```

5 4. Why is K mostly constant even with $T = 10$ periods?

Take the model from part 3:

[35]:
```
historical_K
```

[35]:
```
[[array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.12498245]]), array([[1.12498245]])],
 [array([[1.12498154]]), array([[1.12498154]])],
 [array([[1.12496399]]), array([[1.12496399]])],
 [array([[1.12462445]]), array([[1.12462445]])],
 [array([[1.11808]]), array([[1.11808]])],
 [array([[1.]]), array([[1.]])]]
```

[36]:
```
historical_k
```

[36]:
```
[[array([[-4.44361509]]), array([[4.44361509]])],
 [array([[-4.44323468]]), array([[4.44323468]])],
 [array([[-4.44218616]]), array([[4.44218616]])],
 [array([[-4.43929606]]), array([[4.43929606]])],
 [array([[-4.43132998]]), array([[4.43132998]])],
 [array([[-4.40937305]]), array([[4.40937305]])],
 [array([[-4.34885822]]), array([[4.34885822]])],
 [array([[-4.18217383]]), array([[4.18217383]])],
 [array([[-3.72498233]]), array([[3.72498233]])],
 [array([[-2.5092]]), array([[2.5092]])],
 [array([[0.]]), array([[0.]])]]
```

[37]:
```
historical_kappa
```

[37]:
```
[[array([[-216.3966915]]), array([[-216.3966915]])],
 [array([[-200.94251047]]), array([[-200.94251047]])],
 [array([[-183.77664135]]), array([[-183.77664135]])],
 [array([[-164.71845573]]), array([[-164.71845573]])],
 [array([[-143.58403932]]), array([[-143.58403932]])],
 [array([[-120.21526985]]), array([[-120.21526985]])],
 [array([[-94.56360367]]), array([[-94.56360367]])],
 [array([[-66.92345457]]), array([[-66.92345457]])],
 [array([[-38.56471465]]), array([[-38.56471465]])],
 [array([[-13.330125]]), array([[-13.330125]])],
 [0, 0]]
```