Kalman Filter

Progetto di Intelligenza Artificiale e Laboratorio, Parte 3

Emanuele Gentiletti, Alessandro Caputo

Indice

Introduzione

La seconda parte dell'esercitazione relativa al progetto di Uncertainty consiste nell'implementazione di un Kalman Filter, e nell'esecuzione di diverse simulazioni variando i parametri iniziali. In seguito descriviamo l'implementazione da noi scritta, e mostriamo, commentandoli, i dati ottenuti dalle varie prove eseguite.

Implementazione

Kalman Filter

L'implementazione del Kalman Filter è contenuta nel file `kalman_filter/kalman.py`. Il Kalman Filter è implementato per eseguire una previsione sulla posizione e sulla velocità di un oggetto. Il codice è il seguente:

```
def kalman_filter(state, state_cov, mea_cov, control=0,
                  timedelta=1, pnoise_cov=np.diag((0.1, 0.1))):
    measurement = yield
    while True:
        state = calculate_movement(state, timedelta, control)
        state_cov = get_a_priori_error_cov(state_cov, timedelta,
            pnoise_cov)
        gain = get_gain(state_cov, mea_cov)
        state = get_a_posteriori_estimate(state, measurement, gain)
        state_cov = get_a_posteriori_error_cov(state_cov, gain)
        measurement = yield KalmanResult(state, state_cov, gain)
```

Il Kalman Filter è implementato come un generatore. C'è un loop infinito, che si interrompe ogni volta nell'esecuzione dell'espressione `yield`, aspettando la misurazione successiva tramite cui calcolare la previsione.

L'uso del Kalman Filter è il seguente:

```
kf = kalman_filter(...)
next(kf)
result1 = kf.send(measurement1)
result2 = kf.send(measurement2)
```

Tramite l'esecuzione di `next`, il generatore si esegue fino alla prima richiesta di valori, rappresentata da `measurement = yield`. Chiamando poi `kf.send`, si invia al generatore il valore da asse-

gnare a `measurement`, che riprende quindi ad eseguirsi fino a produrre il risultato relativo a quella misurazione.

La logica all'interno di `kalman_filter` funziona in questo modo:

- Calcola una stima a priori dello stato basata sulla legge oraria del moto accelerato, tramite `calculate_movement`
- Calcola una stima a priori della covarianza dello stato, a partire dalla covarianza dello stato precedente (o quella iniziale) e dal rumore del processo.
- Calcola il gain.
- Calcola la stima a posteriori dello stato, basata su quella a priori, sulla misurazione e sul gain.
- Calcola la covarianza a posteriori dello stato, pesandola sul gain.
- Produci come risultato lo stato, la stima della covarianza dello stato e il gain.

Le stime a priori sono fatte tramite le leggi della fisica, in modo da stimare la posizione e la velocità di un oggetto in movimento. Sono eseguite dalle seguenti funzioni:

```
1  # La @ é l'operatore moltiplicazione di matrice.
2
3  def calculate_movement(state: np.ndarray, timedelta: float, control:
       float):
4      A = np.array([[1, timedelta], [0, 1]])
5      B = np.array([[0.5 * timedelta ** 2], [timedelta]])
6      return A @ state + B * control
7
8
9  def get_a_priori_error_cov(state_cov, timedelta, Q):
10     A = np.array([[1, timedelta], [0, 1]])
11     return A @ state_cov @ A.T + Q
```

`calculate_movement` è la legge del moto accelerato sotto forma matriciale, dove il parametro `control` sta a indicare l'accelerazione dell'oggetto. La usiamo sia per fare le predizioni all'interno del Kalman Filter, sia per simulare poi l'oggetto in movimento.

`get_a_priori_error_cov` fa invece la stima a priori della covarianza. Tramite Q, va a tenere in conto dell'incertezza dovuta al rumore del processo.

Le stime a posteriori sono calcolate tramite le seguenti funzioni:

```
def get_gain(state_cov, mea_cov):
    return state_cov @ inv(state_cov + mea_cov)

def get_a_posteriori_estimate(a_priori_estimate, measurement, gain):
    return a_priori_estimate + gain @ (measurement - a_priori_estimate)


def get_a_posteriori_error_cov(previous_cov, gain):
    return (np.identity(2) - gain) @ previous_cov
```

Tramite `get_gain`, si va a calcolare il Kalman Gain, mentre le altre due funzioni vanno a calcolare la stima e la covarianza a posteriori basate sulla misurazione e sul gain.

Simulazione

La simulazione è implementata dal seguente codice, nel file `kalman_filter/kalman.py`.

```
def simulate_moving_object(
    real_state, real_process_cov, real_sensor_cov, acceleration,
    kalman_state=None, kalman_state_cov=None, kalman_process_cov=None,
    kalman_sensor_cov=None, timedelta=1):
    ...
    kf = kalman_filter(...)
    next(kf)
    while True:
        real_state = calculate_movement(real_state, timedelta,
           acceleration)
        real_state += noise(real_process_cov)
        measurement = real_state + noise(real_sensor_cov)
        kalman_prediction = kf.send(measurement)
        yield SimResult(real_state, measurement, kalman_prediction)
```

Lo stato iniziale viene passato tramite il parametro `real_state`. A ogni iterazione, lo stato viene ricalcolato tramite `calculate_movement`. Allo stato viene quindi aggiunto un rumore casuale campionato da una distribuzione gaussiana multivariata, secondo la covarianza `real_process_cov`, passata come argomento alla funzione. Viene quindi simulata una misurazione sullo stato, prendendo lo stato reale e aggiungendogli un rumore casuale, la cui covarianza è configurabile tramite il parametro `real_sensor_cov`.

La misurazione viene poi mandata al Kalman Filter, che provvede a produrre una predizione. La funzione quindi dà in output lo stato reale, la misurazione effettuata, e la corrispondente predizione fatta dal Kalman Filter, per poi procedere all'iterazione successiva in cui ripete il procedimento.

Prove

Le prove vengono eseguite tramite la funzione `kalman_simulation`, che provvede a eseguire la simulazione secondo diversi parametri:

- Parametri della simulazione:
  - Numero di iterazioni
  - Stato iniziale
  - Covarianza del processo
  - Covarianza dei sensori
- Parametri del Kalman Filter:
  - Stato iniziale stimato
  - Covarianza stimata per lo stato
  - Rumore del processo stimato
  - Covarianza del rumore dei sensori stimata
- Parametri in comune:
  - Accelerazione dell'oggetto
  - Tempo tra gli spostamenti simulati

La funzione esegue la simulazione e restituisce una lista di tuple, una per ogni istante della simulazione. Ogni tupla contiene:

- Stato reale dell'oggetto
- Misurazione fatta sull'oggetto
- Stato, covarianza stato e Kalman Gain del Kalman Filter.

Inoltre, la funzione stampa un grafico che rappresenta gli stati reali, le misurazioni, e gli stati predetti dal filtro, che useremo per fare delle valutazioni sui risultati ottenuti.

Stime e covarianze corrette

```
np.random.seed(20190719)
results = kalman_simulation(
    iterations=30,
    real_state=np.array([0, 2]).reshape(2, 1),
    real_process_cov=np.diag([1000, 2]),
    real_sensor_cov=np.diag([1000, 4]),
    acceleration=1,
)
```

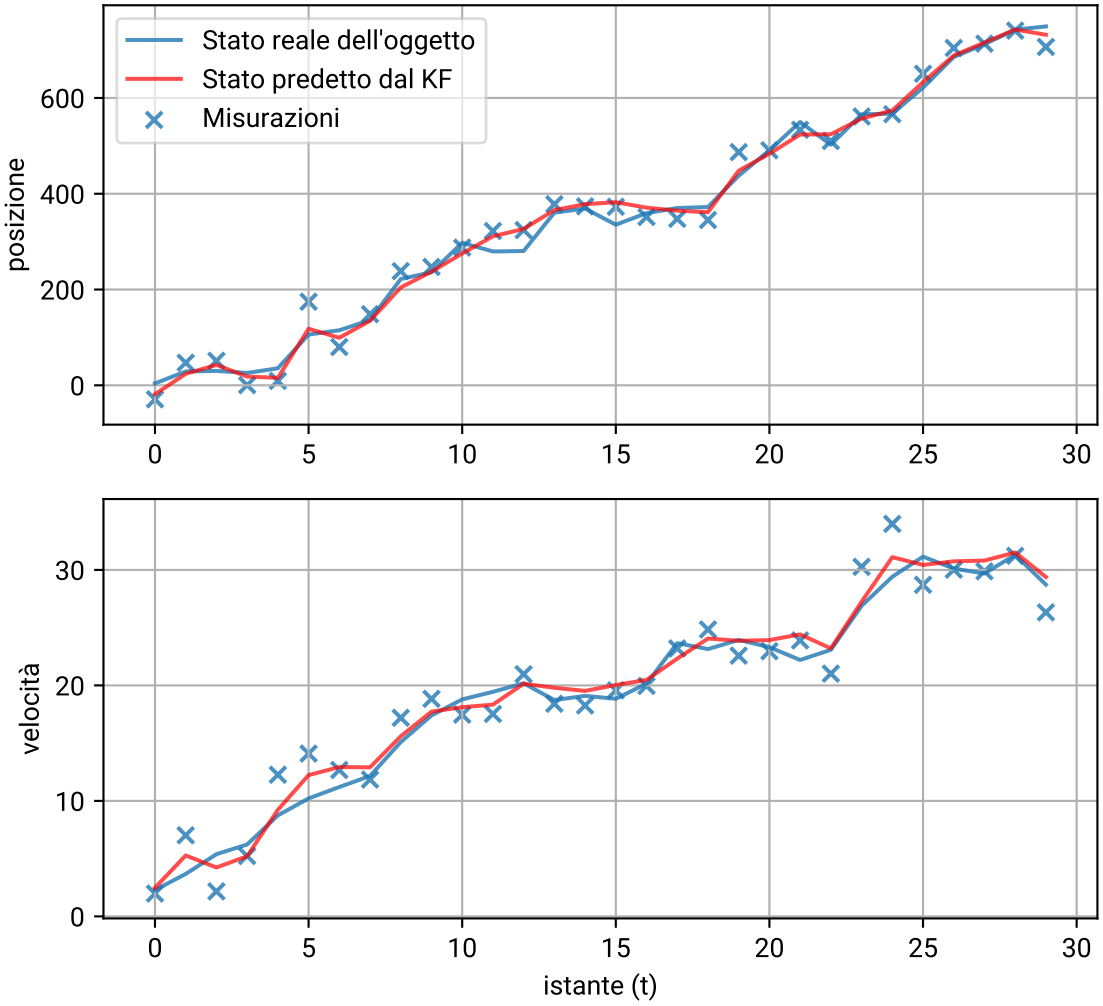

Figura 1: Risultati per il primo esperimento.

Nella prima simulazione, il Kalman Filter riceve i valori giusti riguardanti lo stato e le covarianze dello stato, del processo e dei sensori. Quando la funzione non riceve parametri riguardanti il Kalman Filter, li imposta implicitamente a quelli reali, impostando come covarianza iniziale dello stato la covarianza reale del processo.

Dal grafico possiamo vedere come le misurazioni effettuate nelle fasi iniziali del processo, influiscano in maniera più importante sull'andamento della stima. Nelle ultime fasi invece vediamo che misurazioni molto sbilanciate fanno cambiare di poco la stima, come nel caso della velocità verso l'istante 25.

In seguito mostriamo covarianza dello stato e Kalman Gain nell'ultimo istante di simulazione.

```
results[-1].kalman.state_cov
```

```
## array([[6.184e+02, 4.714e-01],
##        [4.714e-01, 1.999e+00]])
```

```
results[-1].kalman.gain
```

```
## array([[6.184e-01, 1.179e-01],
##        [4.714e-04, 4.998e-01]])
```

Stime scorrette, covarianze corrette

Nel secondo esperimento, proviamo a dare al Kalman Filter stime iniziali molto lontane rispetto alla posizione reale. Al fine di generare gli stessi stati dell'esperimento precedente, assegniamo sempre lo stesso seed al generatore di numeri casuali.

```
np.random.seed(20190719)
results = kalman_simulation(
    iterations=30,
    real_state=np.array([0, 2]).reshape(2, 1),
    real_process_cov=np.diag([1000, 2]),
    real_sensor_cov=np.diag([1000, 4]),
    kalman_state=np.array([2000,100]).reshape(2,1),
    acceleration=1,
)
```

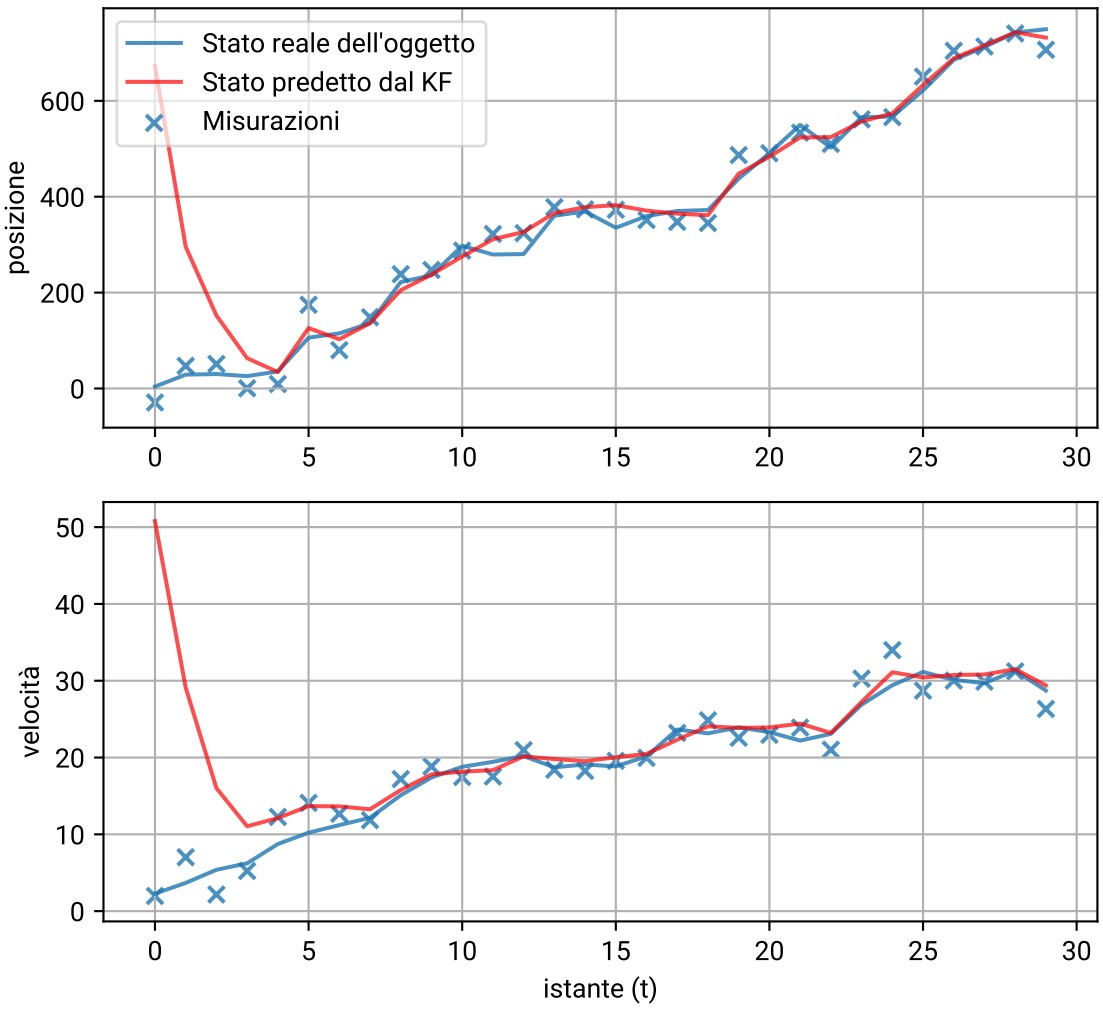

Figura 2: Risultati per il secondo esperimento.

Possiamo osservare come, nonostante ci sia una differenza enorme tra lo stato iniziale del processo e lo stato iniziale del Kalman Filter, questo converga molto presto verso lo stato reale (verso l'istante 8, la stima è già molto vicina). Possiamo verificare come le covarianze e i Kalman Gain cambino dai primi istanti di simulazione rispetto agli ultimi:

```
results[0].kalman.gain
```

```
## array([[6.668e-01, 8.329e-02],
##        [3.332e-04, 4.999e-01]])
```

```
results[-1].kalman.gain
```

```
## array([[6.184e-01, 1.179e-01],
##        [4.714e-04, 4.998e-01]])
```

```
results[0].kalman.state_cov
```

```
## array([[6.668e+02, 3.332e-01],
##        [3.332e-01, 2.000e+00]])
```

```
results[-1].kalman.state_cov
```

```
## array([[6.184e+02, 4.714e-01],
##        [4.714e-01, 1.999e+00]])
```

Stime scorrette, covarianza dello stato iniziale a 0

Nel prossimo esperimento, diamo di nuovo in input al filtro una stima errata dello stato iniziale, ma impostiamo a zero la covarianza dello stato iniziale.

```
np.random.seed(20190719)
results = kalman_simulation(
    iterations=30,
    real_state=np.array([0, 2]).reshape(2, 1),
    real_process_cov=np.diag([1000, 2]),
    real_sensor_cov=np.diag([1000, 4]),
    kalman_state=np.array([2000,100]).reshape(2,1),
    kalman_state_cov=np.diag([0,0]),
    acceleration=1,
)
```

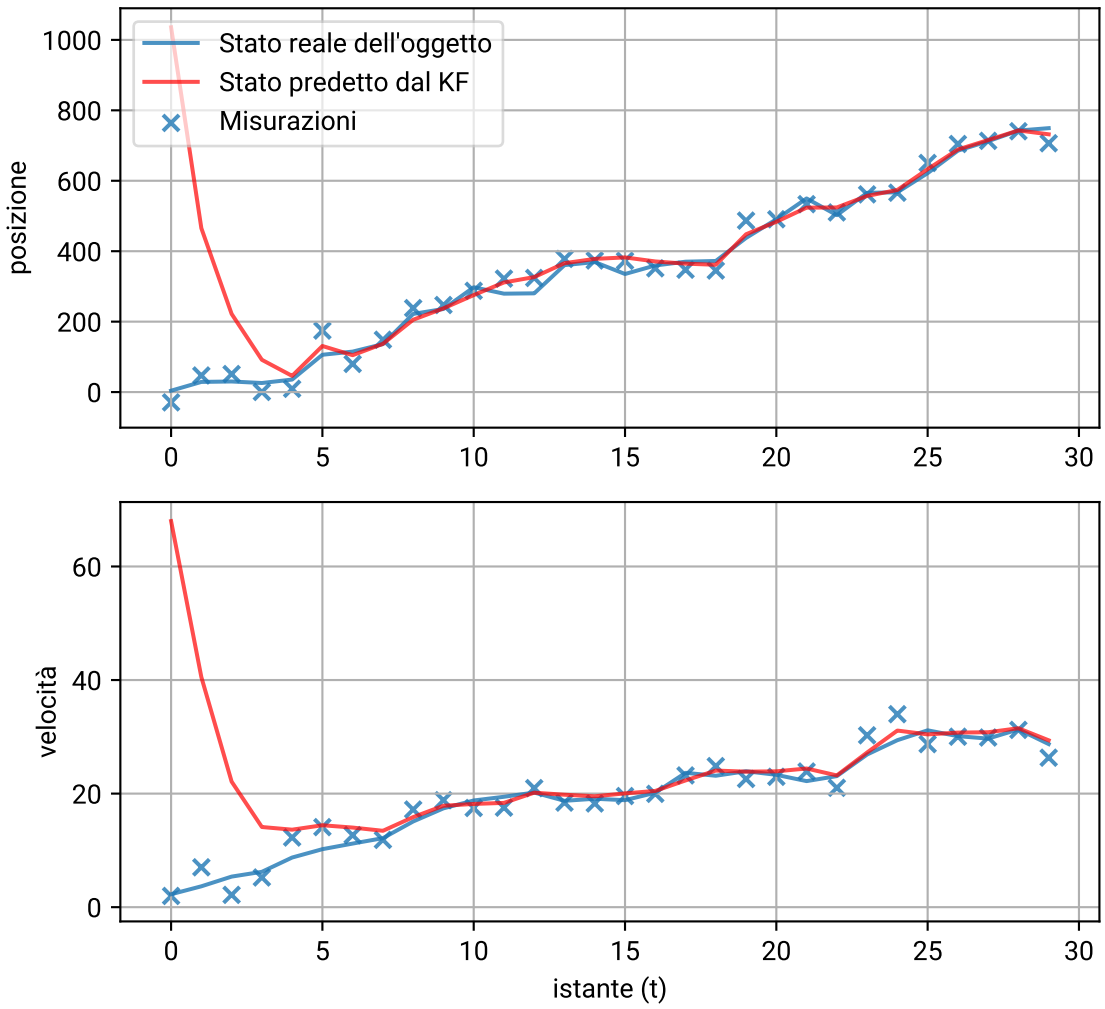

Figura 3: Risultati per il terzo esperimento.

Nonostante la stima iniziale dell'errore sia a zero, il Kalman Filter riesce comunque a convergere verso una stima accettabile più o meno nello stesso tempo. Il comportamento nel secondo e nel terzo esperimento è molto simile, e supponiamo che ciò sia dovuto al fatto che la covarianza del processo esatta data in input al Kalman Filter gli permetta comunque di dare dei pesi adeguati alle misurazioni.

Stime scorrette, covarianza dello stato iniziale a 0 e covarianza processo tendente a 0

In questo esperimento, vogliamo verificare l'ipotesi fatta prima, ovvero che una covarianza di processo ben calibrata permetta al Kalman Filter di dare delle stime corrette anche con stime di errore dello stato

iniziale molto sottostimate. Per fare questo, proviamo a sottostimare anche la covarianza di processo nel Kalman Filter, aspettandoci che le predizioni in questo caso siano completamente errate.

```
np.random.seed(20190719)
results = kalman_simulation(
    iterations=30,
    real_state=np.array([0, 2]).reshape(2, 1),
    real_process_cov=np.diag([1000, 2]),
    real_sensor_cov=np.diag([1000, 4]),
    kalman_state=np.array([2000,100]).reshape(2,1),
    kalman_state_cov=np.diag([0,0]),
    kalman_process_cov=np.diag([0.001,0.001]),
    acceleration=1,
)
```

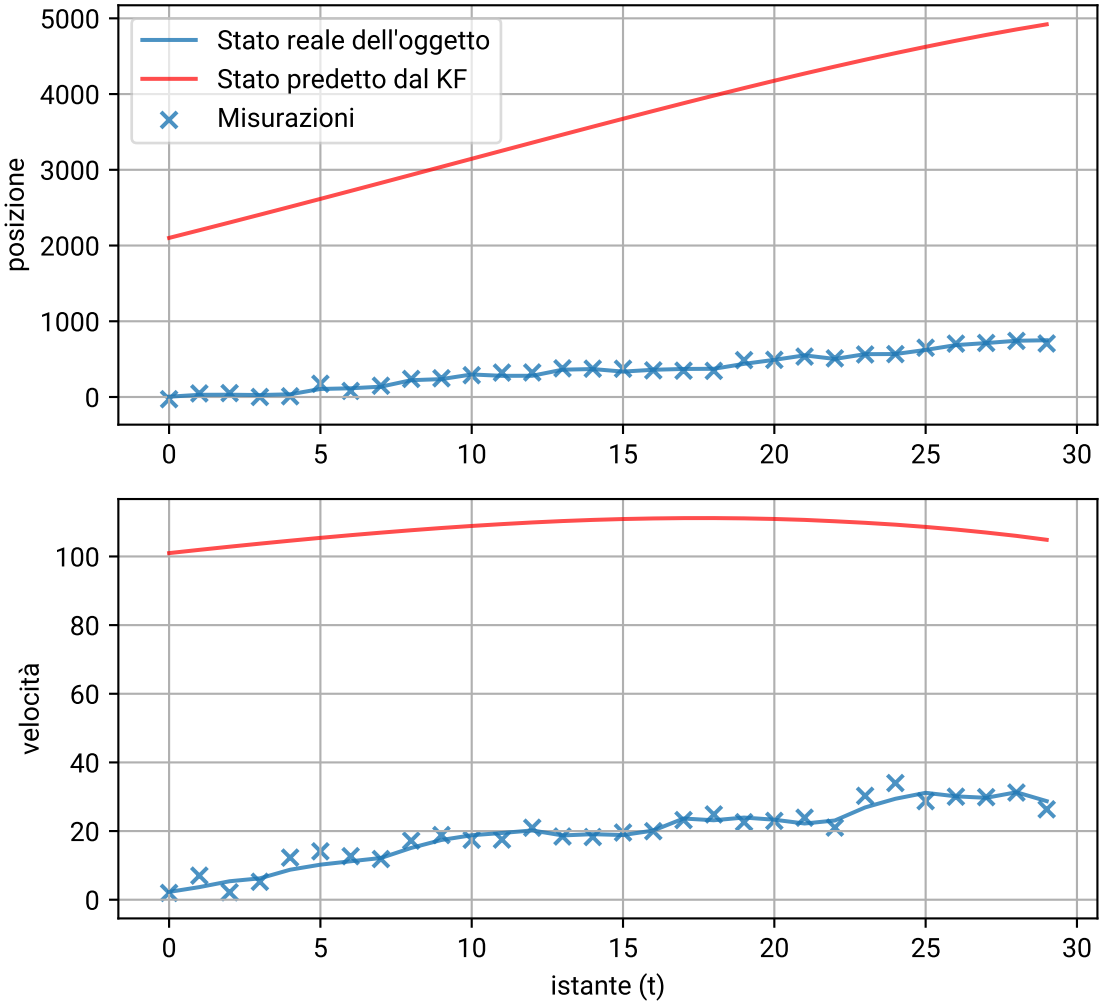

Figura 4: Risultati per il quarto esperimento.

Come ci si può aspettare, impostando la matrice di covarianza del processo verso lo 0, il Kalman Filter non è in grado di stimare correttamente quello che sta accadendo realmente. Possiamo osservare anche come le covarianze e i Kalman Gain siano cambiate nel corso dell'esecuzione rispetto agli esperimenti precedenti.

```
results[15].kalman.gain
```

```
## array([[0.001, 0.029],
##        [0.   , 0.004]])
```

```
results[15].kalman.state_cov
```

```
## array([[1.217, 0.116],
##        [0.116, 0.016]])
```

Essendo la covarianza di processo molto bassa, il Kalman Filter pone molto peso sulla stima del processo, e tiene quindi in conto molto poco delle misurazioni fatte. La stima, nel corso del tempo, resta vicina a quella fatta inizialmente, e cambia di troppo poco per avvicinarsi a quella reale.

Stime corrette, cov. sensori reale alta, cov. sensori stimata vicina a 0

In questo esperimento, vogliamo simulare un processo molto deterministico, dove la stima del processo senza il contributo delle osservazioni darebbe dei buoni risultati di per sé. Tuttavia, vogliamo che il Kalman Filter dia più peso alle osservazioni, che in questo caso faremo essere molto imprecise. Per fare ciò, diamo una covarianza al rumore dei sensori nel processo reale, e una bassa a quella stimata del Kalman Filter.

```
np.random.seed(19072019)
results = kalman_simulation(
    iterations=30,
    real_state=np.array([0, 2]).reshape(2, 1),
    real_process_cov=np.diag([1, 1]),
    real_sensor_cov=np.diag([10000, 4000]),
    kalman_sensor_cov=np.diag([0.001,0.001]),
    acceleration=1,
)
```

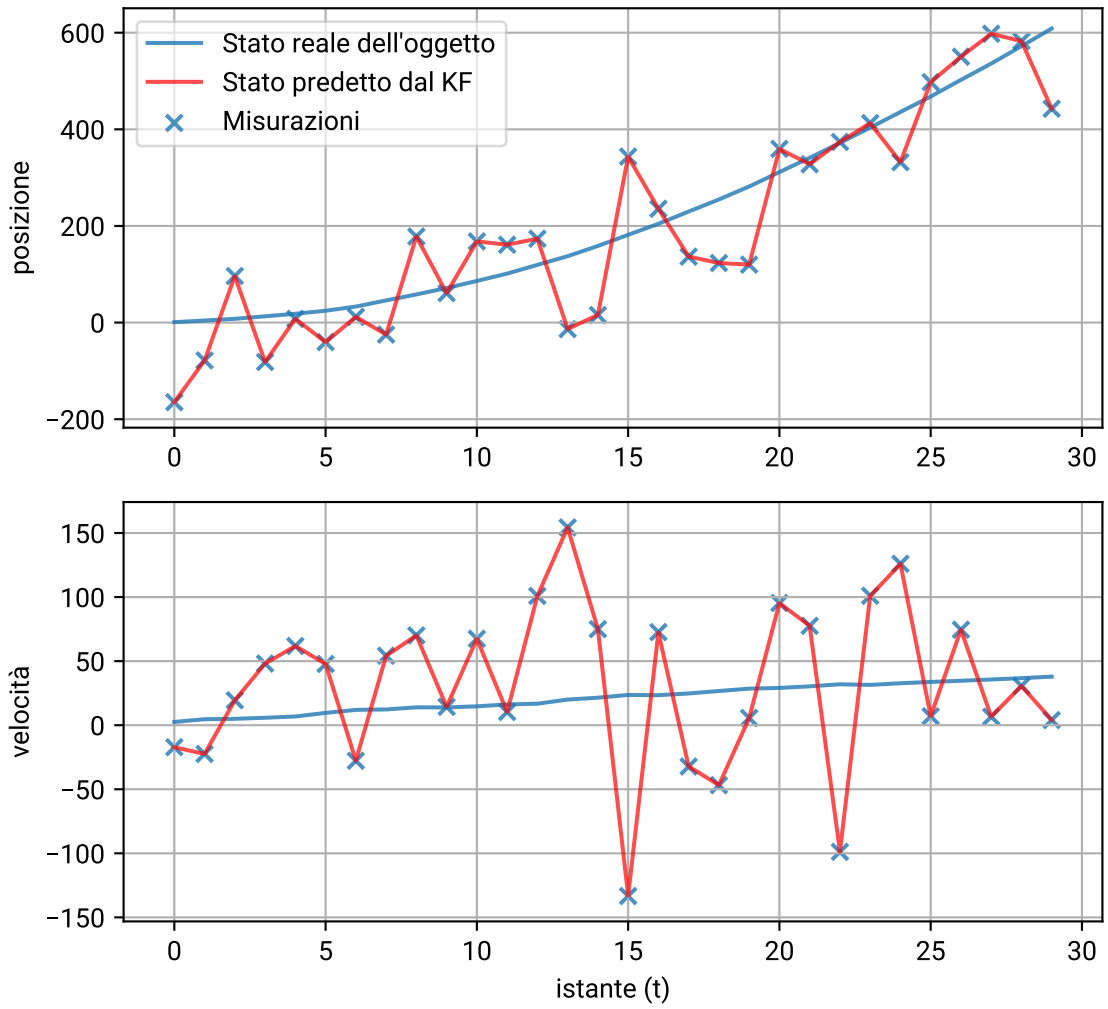

Figura 5: Risultati per il quinto esperimento.

In questo caso, la stima fatta dal Kalman Filter corrisponde quasi esattamente cone le misurazioni dei sensori, come ci aspettavamo.

Stime corrette, cov. sensori reale alta, cov. sensori e processo stimate vicina a 0

In questa prova, replichiamo l'esperimento precedente, dando in input al Kalman Filter covarianze sia dei sensori che del processo vicine allo 0.

```
np.random.seed(19072019)
results = kalman_simulation(
    iterations=30,
    real_state=np.array([0, 2]).reshape(2, 1),
    real_process_cov=np.diag([1, 1]),
    real_sensor_cov=np.diag([10000, 4000]),
    kalman_process_cov=np.diag([0.001,0.001]),
    kalman_sensor_cov=np.diag([0.001,0.001]),
    acceleration=1,
)
```

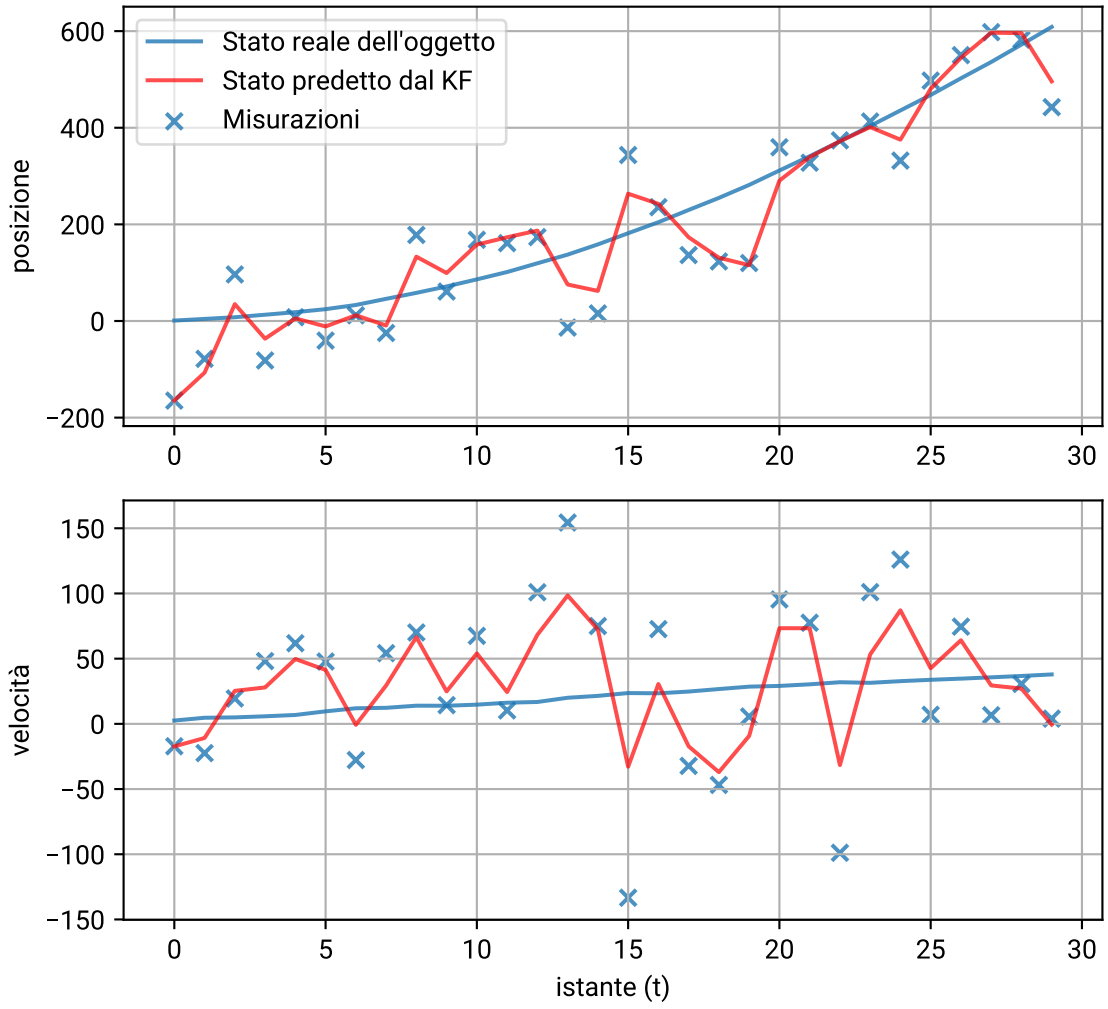

Figura 6: Risultati per il sesto esperimento.

Anche in questo caso, le stime sono molto altalenanti. Quello che si può notare rispetto all'esperimento precedente è che le misurazioni più estreme «trascinano» meno la stima (questo si può osservare in particolare verso l'istante 15 nei due grafici, dove le misurazioni sono lontane dalla media).

```
results[15].kalman.gain
```

```
## array([[0.694, 0.079],
##        [0.079, 0.594]])
```

```
results[15].kalman.state_cov
```

```
## array([[6.944e-04, 7.932e-05],
##        [7.932e-05, 5.939e-04]])
```

Conclusioni

Il Kalman Filter è un potente strumento con cui fare inferenza, e riesce a dare buoni risultati in situazioni di forte incertezza. I parametri più cruciali per la buona configurazione di un Kalman Filter sono le covarianze: con buone stime di covarianza del processo e del rumore dei sensori, le approssimazioni date si avvicinano molto allo stato reale. Sono meno importanti le stime iniziali dello stato e della covarianza dello stato: anche con stime iniziali molto errate rispetto allo stato reale, il Kalman Filter riesce a fare buone approssimazioni in poche iterazioni.