HW #5, ECE 6555

Theodore [illegible]
Due: 12/16/22

Goal: Compare EKF w/ Particle Filter.

Scalar, non-linear model:

$$x_k = \frac{1}{2} x_{k-1} + \frac{25 x_{k-1}}{1 + x_{k-1}^2} + 8\cos(1.2(k-1)) + u_k \qquad (1)$$

$$y_k = \frac{1}{20} x_k^2 + v_k \qquad (2)$$

$\{u_k\}, \{v_k\}$ are white, Gaussian noise sequences w/ unit variance.

$Q_k = 1$, $R_k = 1$

initial state: $x_0 = 0.1$, $\hat{x}_0 = 0$, $\hat{\sigma}_0 = 2$

note: both state & measurement processes are non-linear.

$\boxed{\text{Q1) Plot the state process via python (See python code for plot)}}$

Now for the Particle Filter

method:
1) draw n samples from the prior $x_0^{(i)} \sim p(x_0)$ $\quad i = 1 \ldots n$
   and set $w^{(i)} = 1/n$ for $i = 1 \ldots n$
2) For each $k = 1 \ldots T$
   a) draw samples $x_k^{(i)}$ from importance distributions
   $$x_k^{(i)} \sim \pi(x_k \mid x_{0:k-1}^{(i)}\, y_{0:k}) \qquad i = 1 \ldots n$$
   b) compute new weights
   $$w_k^{(i)} \propto w_{k-1}^{(i)} \frac{p(y_k \mid x_k^{(i)})\, p(x_k^{(i)} \mid x_{k-1}^{(i)})}{\pi(x_k^{(i)} \mid x_{0:k-1}^{(i)}\, y_{0:k})}$$ and normalize
3) If the effective # of particle is too low, re-sample and reset to uniform weights, where $n_{eff} \approx \frac{1}{\sum_{j=1}^{n} (w_k^{(j)})^2}$ (i.e. the # of effective particles at a particular time-step)

But, in class we did not specify how to choose the importance distribution and how to perform the sampling.

(basically, try to get as close to real distr. as possible, and be Markovian)

Now, we choose

$$\pi(x_k^{(i)} \mid x_{0:k-1}^{(i)}\, y_{1:k}) \triangleq p(x_k^{(i)} \mid x_{k-1}^{(i)}) \qquad (3)$$

that is, we only push the current particle $x_{k-1}^{(i)}$ through a process update.

Q2] Given a set of particles $\{x_{k-1}^{(i)}\}$, show that sampling from the importance distribution $\pi(x_k \mid x_{0:k-1}^{(i)}\, y_{1:k})$ then reduces to computing:

$$\frac{1}{2} x_{k-1}^{(i)} + \frac{25\, x_{k-1}^{(i)}}{1 + (x_{k-1}^{(i)})^2} + 8\cos(1.2(k-1)) \qquad (4)$$

and adding the realization of the 0-mean white Gaussian noise.

Q2 cont'd

from (3) we have

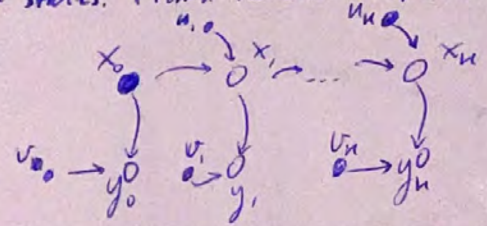
$$\pi(x_k \mid x^{(i)}_{0:k-1}, y_{1:k}) \triangleq p(x_k \mid x^{(i)}_{k-1})$$

from eqtn (1) $\Rightarrow$

$$p(x_k \mid x^{(i)}_{k-1}) = p\left(\tfrac{1}{2}x_{k-1} + \frac{25\,x_{k-1}}{1+x_{k-1}^2} + 8\cos(1.2(k-1)) + u_k \,\Big|\, x^{(i)}_{k-1}\right)$$

$$= \frac{1}{2}x^{(i)}_{k-1} + \frac{25\,x^{(i)}_{k-1}}{1+(x^{(i)}_{k-1})^2} + 8\cos(1.2(k-1)) + p(u_k \mid x^{(i)}_{k-1})$$

Since u_k is Gaussian white noise, it is independent of past and present states. From a Functional Dependence Graph perspective, we see that indeed



u_k is independent of x_{k-1}.

Thus, $p(u_k \mid x^{(i)}_{k-1}) = p(u_k) = 0$

In summary,

$$\pi(x_k \mid x^{(i)}_{0:k-1}, y_{1:k}) \triangleq p(x_k \mid x^{(i)}_{k-1}) = \frac{1}{2}x^{(i)}_{k-1} + \frac{25\,x^{(i)}_{k-1}}{1+(x^{(i)}_{k-1})^2} + 8\cos(1.2(k-1)) + p(u_k)$$

so that when we sample from the importance distribution, given $\{x^{(i)}_{k-1}\}$, then the resultant sample is eqtn (1) plus the realization of $\{u_k\}$.

[Q3] Explicit expression for computation of weights $w_k^{(i)}$ as a function of $w_{k-1}^{(i)}$. Make the subsequent normalization to avoid unnecessary computations.

$\longrightarrow$

Q3 cont'd

$$w_n^{(i)} \propto w_{n-1}^{(i)} \frac{p(y_n \mid x_n^{(i)})\, p(x_n^{(i)} \mid x_{n-1}^{(i)})}{\pi(x_n^{(i)} \mid x_{0:n-1}^{(i)}, y_{1:n})}$$

~~how to deal w/ this? is it the same as $\pi(x_n \mid x_{0:n-1}^{(i)}, y_{1:n})$? I don't think so... there's a comma there... ↳ it is the same - should be a comma always!~~

where $\pi(x_n^{(i)} \mid x_{0:n-1}^{(i)}\ y_{0:n})$

$$= p(x_n^{(i)} \mid x_{n-1}^{(i)})$$

thus,

$$w_{n-1}^{(i)} \frac{p(y_n \mid x_n^{(i)})\, \cancel{p(x_n^{(i)} \mid x_{n-1}^{(i)})}}{\cancel{\pi(x_n^{(i)} \mid x_{0:n-1}^{(i)}\ y_{0:n})} \to \cancel{p(x_n^{(i)} \mid x_{n-1}^{(i)})}} = w_{n-1}^{(i)}\, p(y_n \mid x_n^{(i)})$$

and $\mathbb{E}_p(y_n \mid x_n^{(i)}) = \mathbb{E}_p(\frac{1}{20} x_n^2 + v_n \mid x_n^{(i)}) = \mathbb{E}_p(\frac{1}{20} x_n^2 \mid x_n^{(i)}) + \mathbb{E}_p(v_n \mid x_n^{(i)})$

note: $\mathbb{E}_p(v_n \mid x_n^{(i)}) = \mathbb{E}_p(v_n)$ b/c v_n is independent of $x_n^{(i)}$

and ~~thus~~ $\mathbb{E}_p(v_n) = 0$, thus;

$$\boxed{\mathbb{E}_p(y_n \mid x_n^{(i)}) = \mathbb{E}_p(\tfrac{1}{20} x_n^2 \mid x_n^{(i)}) = \frac{1}{20}(x_n^{(i)})^2} \qquad (5)$$

~~such that~~

$$\cancel{w_k^{(i)} \propto w_{k-1}^{(i)}\, p(y_n \mid x_n^{(i)}) = \frac{w_{n-1}^{(i)} (x_{n-1}^{(i)})^2}{20}}$$

$$\Rightarrow \cancel{\boxed{w_n^{(i)} \propto \frac{1}{20} w_{n-1}^{(i)} (x_{n-1}^{(i)})^2}} \qquad \cancel{(6)}$$

→

Q3 cont'd)

each y_n is Gaussian distributed around the mean, due to the Gaussian noise $\{v_n\}$. Thus,

$$y_n \sim \mathcal{N}(\mathbb{E}[y_n], 1) \qquad \text{since } v_n \sim \mathcal{N}(0,1)$$

Note: $K_y = \mathbb{E}[(y_n - \mu_y)(y_n - \mu_y)^T] = \mathbb{E}\left[\left(\cancel{\tfrac{1}{20}x_n^{(i)}} + v_n - \cancel{\tfrac{1}{20}x_n^{(i)}}\right)\left(\cancel{\tfrac{1}{20}x_n^{(i)}} + v_n - \cancel{\tfrac{1}{20}x_n^{(i)}}\right)\right]$

$$= \mathbb{E}(v_n^2) = R_n = 1$$

Now, $p_y(y_n | x_n^{(i)}) = \dfrac{1}{\sigma\sqrt{2\pi}} \exp\left[-\dfrac{1}{2}\left(\dfrac{y_n - \mu_y}{\sigma^2}\right)\right]$

$$\Rightarrow \boxed{p_{y_n}(y_n | x_n^{(i)}) = \frac{1}{\sqrt{2\pi}} \exp\left[-\tfrac{1}{2}\left(y_n - \tfrac{1}{20}x_n^{(i)}\right)\right]} \qquad (7)$$

Such that,

$$w_n^{(i)} \propto w_{n-1}^{(i)} p(y_n | x_n^{(i)})$$

$$\Rightarrow \boxed{w_n^{(i)} \propto w_{n-1}^{(i)}\left(\frac{1}{\sqrt{2\pi}} \exp\left[-\tfrac{1}{2}\left(y_n - \tfrac{1}{20}x_n^{(i)}\right)\right]\right)} \qquad (8)$$

Q4| Implement particle Filter (without) the resampling step. Use 200 part.

4.1) Provide a graph w/ at least mean & var. of filter superposed to data.

4.2) Plot the n_{eff} as a function of time index k.

$\boxed{\hookrightarrow \text{see python code}}$

(recall PF method on page 1, given in problem statement)

1) draw n samples from the prior $x_0^{(i)} \sim p(x_0)$ $i = 1 \ldots n$
and set $w^{(i)} = \frac{1}{n}$ for $i = 1 \ldots n$.

$\hookrightarrow$ note: $x_0 \sim (0, 2)$ from problem statement: [$x_0 = 0.1$, $\hat{x}_0 = 0$, $\sigma_0 = 2$]

2) For each $k = 1 \ldots T$

a) draw samples $x_k^{(i)}$ from importance distributions

$$x_k^{(i)} \sim \pi(x_k^{(i)} | x_{0:k-1}^{(i)} y_{0:k})$$

$$\Rightarrow x_k^{(i)} \sim p(x_k^{(i)} | x_{k-1}^{(i)})$$

where $p(x_k^{(i)} | x_{k-1}^{(i)}) = \frac{1}{2} x_{k-1}^{(i)} + \frac{25 x_{k-1}^{(i)}}{1 + (x_{k-1}^{(i)})^2} + 8 \cos(1.2(k-1)) + p(u_k)$

and $u_k \sim (0, 1)$

b) compute new weights, and normalize:

$$w_k^{(i)} \propto w_{k-1}^{(i)} \frac{p(y_k | x_k^{(i)}) \, p(x_k^{(i)} | x_{k-1}^{(i)})}{\pi(x_k^{(i)} | x_{0:k-1}^{(i)} y_{0:k}^{(i)})} \Rightarrow w_k^{(i)} \propto \cancel{\frac{1}{20} w_{k-1}^{(i)} (x_{k-1}^{(i)})^2}$$

$$w_{k-1}^{(i)} \left(\frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2} \left(y_k - \frac{x_{k-1}^{(i)}}{20} \right) \right] \right)$$

then, $\hat{w}_k^{(i)} = \frac{w_k^{(i)}}{\cancel{\| w_k^{(i)} \|} \sum_{i=1}^{n} w_k^{(i)}}$ ← per lecture, $i = 1 \ldots n$, $\cancel{\| w_k^{(i)} \| = \sqrt{\sum_{i=1}^{n} (w_k^{(i)})^2}}$

want only that $\sum w_k^{(i)} = 1$, to prepare for re-sampling & to standardize weights

3) skip for this part/problem

Q4 cont'd

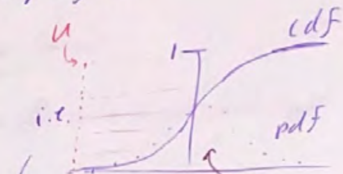
weighted mean: $\mu_w = \dfrac{\sum_{i=1}^n w_i x_i}{\sum_{i=1}^n w_i}$

biased weighted variance: $\hat{\sigma}_w^2 = \dfrac{\sum_{i=1}^n w_i (x_i - \mu_w)^2}{\sum_{i=1}^n w_i}$

Q5 Let X be a random variable width PDF p_x and CDF F. Let U be ~~an~~ a random variable uniformly distributed in $[0,1]$. Show that the variable $F^{-1}(U)$ is distributed according to p_x. s.t. we can sample from

$\sum_{i=1}^n w_u^{(i)} \delta[x - x_u^{(i)}]$ (CDF)

Let $F = \sum_{i=1}^n w_u^{(i)} \delta(x - x_u^{(i)})$ i.e.



something like above...

higher concentration of points around mean for $F^{-1}(U)$

$[P(F^{-1}(U) \le x) = F(x)$

$F^{-1}(\alpha)$ ~~is the~~ will give the value x s.t. $F(x) = \alpha$ $(= P(X \le x))$

so, the probability that $F^{-1}(U)$ ~~is~~ $\le x$ is $P(F^{-1}(U) \le x) =$

$F^{-1}(U) \le x \Rightarrow U \le F(x) \Rightarrow P(F^{-1}(U) \le x) = P(U \le F(x)) = F(x)$

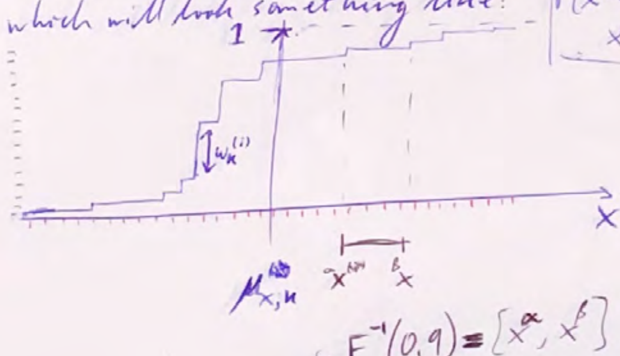
$\Rightarrow \boxed{F^{-1}(U) \propto p_x}$ (distributed according to p_x)

since U is a uniform distribution, any value in $[0,1]$ is equally possible, s.t. the probability that $U < F(x)$ is simply the value of $F(x)$.

So, $F(x) = \sum_{i=1}^n w_u^{(i)} \delta(x - x_u^{(i)})$

$P(x^* \le x) = F(x)$
$x^* \sim p_x$

which will look something like:



so, $F^{-1}(0.9) = [x^\alpha, x^\beta]$

so have to figure out how to split and sample. if $F^{-1}(U_i)$ is between two values, choose the lower one. Then within the range of values that it is $[x^\alpha, x^\beta]$, ~~random~~ randomly select a position in between. (w/ uniform distr.)

α

6

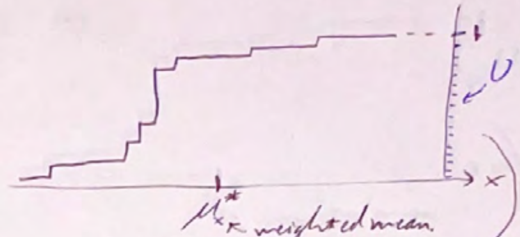
Q6] Plot the PF with re-sampling if $n_{eff} < 20$ (10% of samples / particles)

method for resampling:

at step k, if $n_{eff}[k-1] < 20$:

draw new samples according to:

$F(x) = \sum_{i=1}^{n} w_k^{(i)} \delta(x - x_k^{(i)}) \rightarrow$

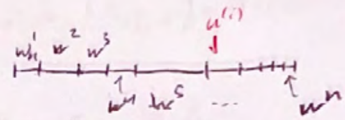


bins = iterative sum of $w^{(i)}$

U = random (uniform, 200)

foreach $U^{(i)}$:

foreach bin: (in reverse)



if $U^{(i)} \geq$ bin:

$x_{new}^{(i)} = x_k^{(i)} + \underbrace{(x_k^{(i)} + x_k^{(i+1)}) * \text{random(uniform)}}$

↳ do this so not the same point each time

break/continue

And continue algorithm...

See python code for implemented PF and algorithm

(Q7) →

Q7 Derive a linearized version of the non-linear system:

System:

$$y_k = \frac{1}{20} x_k^2 + v_k$$

$$x_k = \frac{1}{2} x_{k-1} + \frac{25 x_{k-1}}{1 + x_{k-1}^2} + 8\cos(1.2(k-1)) + u_k$$

---

per Lec 17,

$$x_i = f_{i-1}(x_{i-1}) + u_{i-1} \quad , \quad y_i = h_i(x_i) + v_i$$

$$f_i(x_i) \approx f_i(\hat{x}_{i|i}) + F_i(x_i - \hat{x}_{i|i}) \quad , \quad h_i(x_i) \approx h_i(\hat{x}_{i|i-1}) + H_i(x_i - \hat{x}_{i|i-1})$$

where $F_i = \begin{bmatrix} \frac{\partial f_{i,1}}{\partial x_{i,1}} & \cdots & \frac{\partial f_{i,1}}{\partial x_{i,n}} \\ \vdots & & \\ \frac{\partial f_{i,n}}{\partial x_{i,1}} & \cdots & \frac{\partial f_{i,n}}{\partial x_{i,n}} \end{bmatrix}$ and similar for H_i

let $z_k = x_k - 8\cos(1.2(k-1))$

$$\Rightarrow z_k = \frac{1}{2} x_{k-1} + \frac{25 x_{k-1}}{1 + x_{k-1}^2} + u_k \Rightarrow \text{let } \boxed{f_{k-1}(x_{k-1}) = \frac{1}{2} x_{k-1} + \frac{25 x_{k-1}}{1 + x_{k-1}^2}} \quad (7.1)$$

$$\Rightarrow \boxed{F_k = \frac{\partial f_k}{\partial x_k} = \frac{1}{2} + 25(1 + x_k^2)^{-1} + 25 x_k (-1)(2x_k)^{-2}} \quad (7.2)$$

s.t. $z_{k+1} \approx f_k(\hat{x}_{k|k}) + F_k(\hat{x}_{k|k})(x_k - \hat{x}_{k|k}) + u_k$

$$\Rightarrow \boxed{x_{k+1} \approx f_k(\hat{x}_{k|k}) + F_k(\hat{x}_{k|k})(x_k - \hat{x}_{k|k}) + 8\cos(1.2(k-1)) + u_k} \quad (7.3)$$

---

let $\boxed{h_k(x_k) = \frac{1}{20} x_k^2 \Rightarrow h_k(x_k) \approx h_k(\hat{x}_{k|k-1}) + H_k(\hat{x}_{k|k-1})(x_k - \hat{x}_{k|k-1})}$ (7.4)

s.t. $\boxed{y_k = h_k(\hat{x}_{k|k-1}) + H_k(\hat{x}_{k|k-1})(x_k - \hat{x}_{k|k-1}) + v_k}$ (7.5)

$\boxed{H_k(x_k) = \frac{\partial h_k}{\partial x_k} = \frac{1}{10} x_k}$ (7.6)

EKF equations:

$$\begin{cases} \hat{x}_{i+1|i} = f_i(\hat{x}_{i|i}) \\ \hat{x}_{i|i} = \hat{x}_{i|i-1} + K_{f,i}\left(y_i - h_i(\hat{x}_{i|i-1})\right) \\ K_{f,i} = P_{i|i-1} H_{i|i-1}^T \left(H_{i|i-1} P_{i|i-1} H_{i|i-1}^T + R_i\right)^{-1} \\ P_{i|i} = (I - K_{f,i} H_{i|i-1}) P_{i|i-1} \\ P_{i+1|i} = F_{i|i} P_{i|i} F_{i|i}^T + G_i Q_i G_i^T \qquad , \quad G_i = \text{Identity} \end{cases}$$

Table of Contents

1 [Q1] Make a pot of the trajectory. This will serve as a reference throughout the problem
2 [Q4] Implement the Particle Filter without the resampling step
  2.1 [Q4.1] Provide a graph with at least the mean and variance of the filter superposed to the data.
  2.2 [Q4.2] Plot the n_{eff} as a function of time index k.
3 [Q6] Implement the Particle Filter **with** the resampling step
  3.1 Illustration of sorted particles and weights at a single step
  3.2 [Q6.1] The Particle Filter algorithm (with resampling step)
  3.3 [Q6.2] Plot mean and variance superposed to trajectory
  3.4 [Q6.3] Plot the n_{eff} as a function of time step k
  3.5 [Q6.4] Discussion of Results

ECE6555 HW5

Author: Teo Wilkening Due Date: 2022-12-16

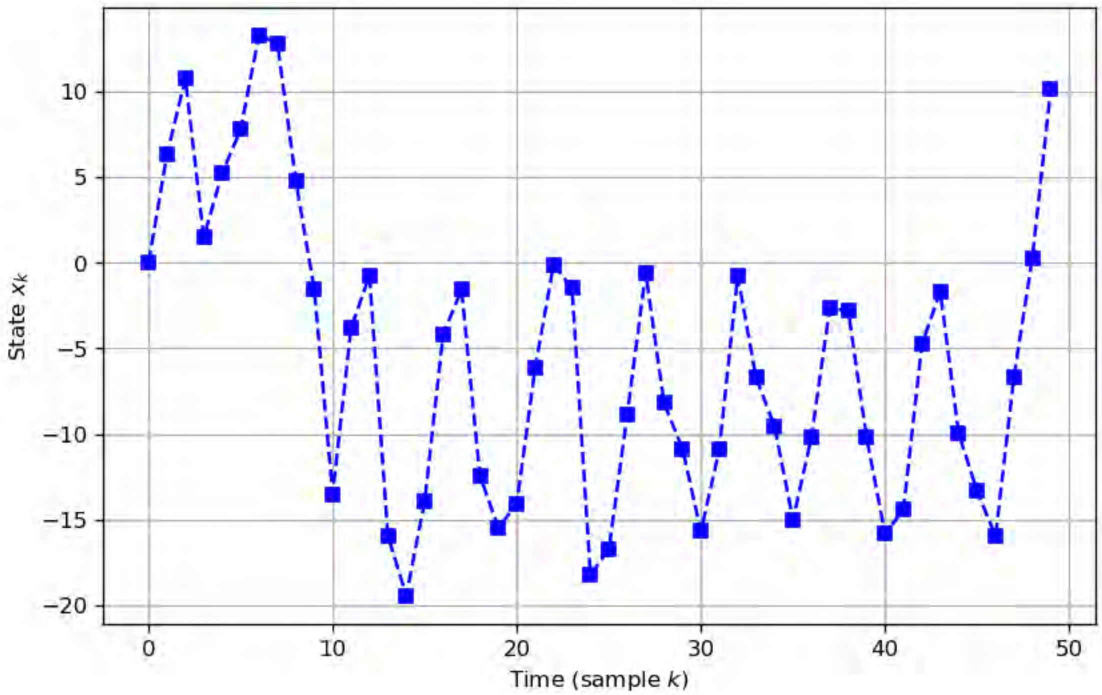
1 [Q1] Make a pot of the trajectory. This will serve as a reference throughout the problem

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt

# Create the trajectory
np.random.seed(202212)
NumSteps = 50
TimeScale = np.arange(1,NumSteps,1)
x0=0
sigma=1

x = [x0]
y = [0]
for k in TimeScale:
    xk = 0.5*x[-1]+25*x[-1]/(1+x[-1]**2)+8*np.cos(1.2*(k-1))+np.random.randn()
    yk = 1/20*xk**2+np.random.randn()
    x.append(xk)
    y.append(yk)

```

In [2]:

```python
# Plot the trajectory
fig, ax = plt.subplots(figsize=(8,5), dpi=120)

ax.plot(np.insert(TimeScale,0,0),x,'b--')
ax.grid(True)
ax.plot(np.insert(TimeScale,0,0),x,'bs',markersize=6)
#plt.legend(['line','markers'])
ax.set_ylabel(r'State $x_k$')
ax.set_xlabel(r'Time (sample $k$)')

plt.show()
```



2 [Q4] Implement the Particle Filter without the resampling step

Use 200 particles.

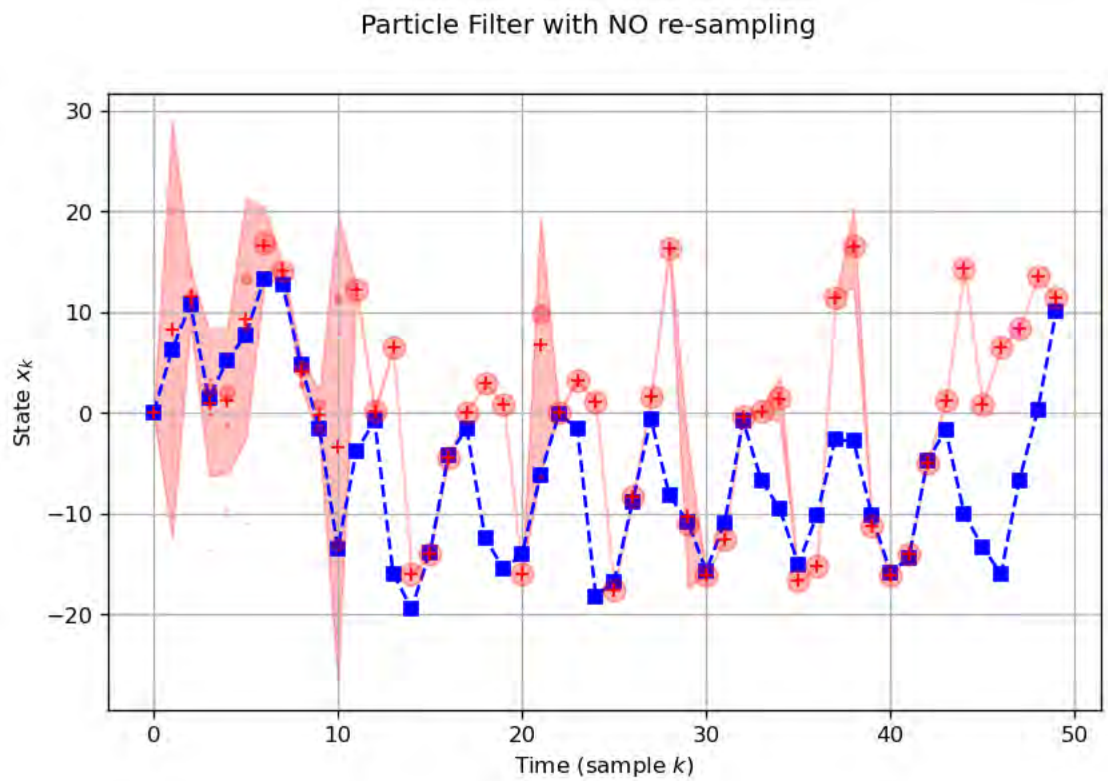
In [12]:

```
# 1) draw n samples from the prior
# 2) for each k = 1...T
#     a) draw samples x_k(i) from the importance distribution
#     b) compute the new weights
#     c) normalize the new weights

# initialize x^i_k and w^i_k matrices to keep track of state estimation distributions and weights
n = 200 # number of particles
xki = np.zeros((NumSteps,n),dtype=float)
wki = np.zeros((NumSteps,n),dtype=float)

# 1) draw n samples from the prior
x0_mu, x0_sigma = 0, np.sqrt(2)
x0 = np.random.normal(x0_mu, x0_sigma, n)
w0 = 1/n*np.ones(n)

# insert the samples from the prior into our matrices for keeping track of things
xki[0,:] = x0
wki[0,:] = w0

# initialize noise Gaussian parameters
u_mu, u_sigma = 0, 1
v_mu, v_sigma = 0, 1

# 2) for each k = 1...T
mean = np.zeros(NumSteps) # keep track of the mean of the particles
var = np.zeros(NumSteps) # keep track of the variance of the particles at each step
neff = np.zeros(NumSteps)

for k in np.arange(1,NumSteps,1):
    # a) draw samples x_k(i) from the importance distribution
    xki[k,:] = 1/2*xki[k-1,:] + 25*xki[k-1,:]/(1 + xki[k-1,:]**2) + 8*np.cos(1.2*(k-1)) + \
                np.random.normal(u_mu, u_sigma,n)
    # print(sum(xki[k,:]))
    # b) compute the new weights
    wki[k,:] = wki[k-1,:]*1/np.sqrt(2*np.pi)*np.exp(-0.5*(y[k] - 1/20*(xki[k-1,:]**2))**2)
    # c) normalize the new weights
    wki[k,:] = wki[k,:]/sum(wki[k,:])
    mean[k] = np.average(xki[k,:],weights=wki[k,:])
    var[k] = np.average((xki[k,:] - mean[k])**2,weights=wki[k,:])
    #var[k] = np.average((xki[k,:])**2,weights=wki[k,:]) - (mean[k])**2
    neff[k] = 1/sum(wki[k,:]**2)

# track mean for later analysis
mean_pf = mean
```

2.1 [Q4.1] Provide a graph with at least the mean and variance of the filter superposed to the data.

In [4]:

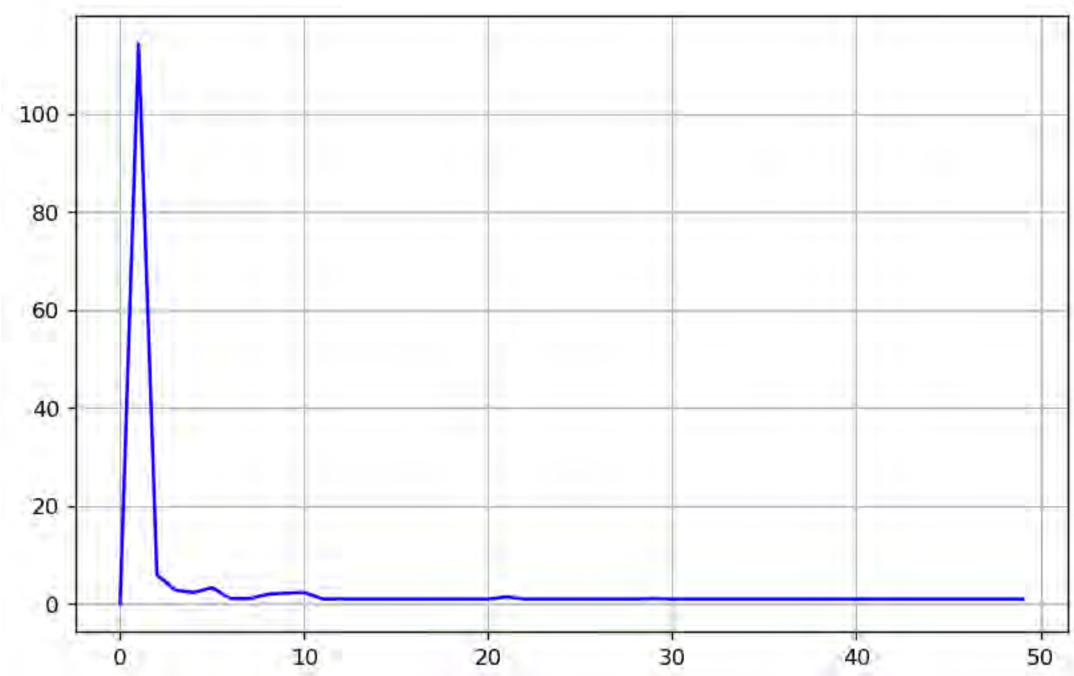
```
# Plot the trajectory
fig, ax = plt.subplots(figsize=(8,5), dpi=120)

ax.plot(np.insert(TimeScale,0,0),x,'b--')
ax.grid(True)
ax.plot(np.insert(TimeScale,0,0),x,'bs',markersize=6)
#plt.legend(['line','markers'])
ax.set_ylabel(r'State $x_k$')
ax.set_xlabel(r'Time (sample $k$)')
for k in np.arange(1,NumSteps,1):
    for i in np.arange(n):
        if wki[k,i] > 1e-3:
            ax.plot(k,xki[k,i],'ro',markersize=10*wki[k,i],alpha=0.3)
ax.plot(mean,'r+')
ax.fill_between(np.arange(NumSteps), mean-2*np.sqrt(var), mean+2*np.sqrt(var), alpha=0.25, color='r')
fig.suptitle('Particle Filter with NO re-sampling')

plt.show()
```



2.2 [Q4.2] Plot the n_{eff} as a function of time index k.

In [5]:

```
# Plot the neff
fig, ax = plt.subplots(figsize=(8,5), dpi=120)

ax.plot(np.insert(TimeScale,0,0),neff,'b')
ax.grid(True)
plt.show()
```



In [ ]:

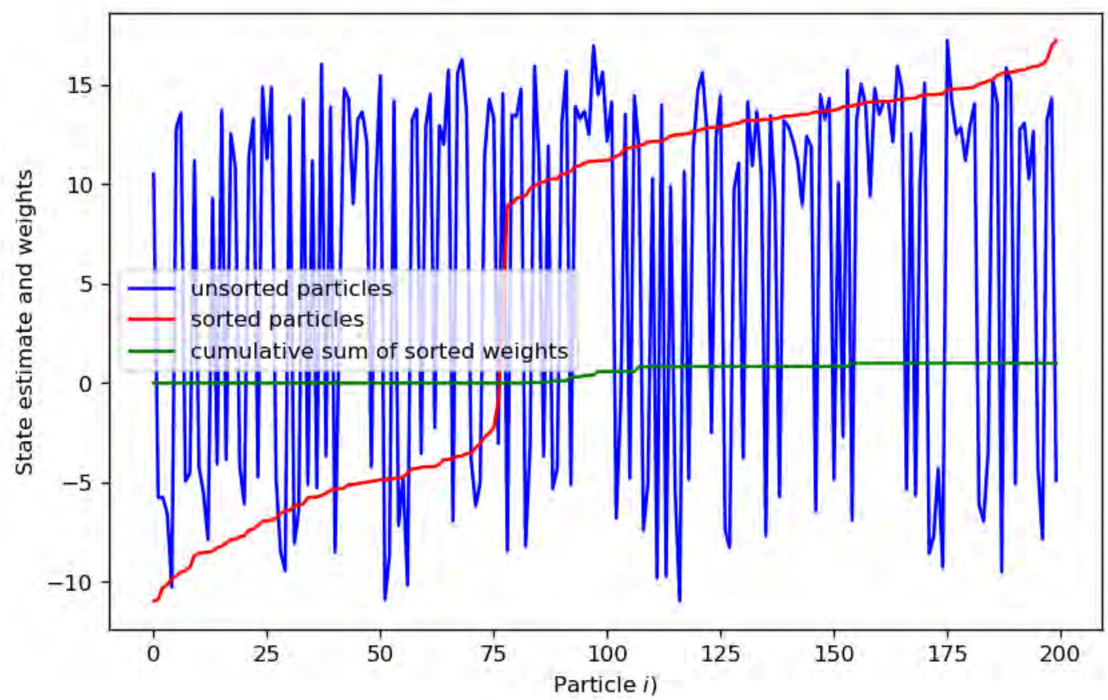
```

```

3 [Q6] Implement the Particle Filter with the resampling step

3.1 Illustration of sorted particles and weights at a single step

In [6]:

```
fig, ax = plt.subplots(figsize=(8,5), dpi=120)


ax.plot(xki[2,:],'b')
ax.plot(np.sort(xki[2,:]),'r')
ax.plot(np.cumsum(np.take_along_axis(wki[2,:],np.argsort(xki[2,:]),axis=0)),'g')
#plt.legend(['line','markers'])
ax.set_ylabel(r'State estimate and weights')
ax.set_xlabel(r'Particle $i$)')
ax.legend(['unsorted particles','sorted particles','cumulative sum of sorted weights'])

plt.show()
```



3.2 [Q6.1] The Particle Filter algorithm (with resampling step)

In [13]:

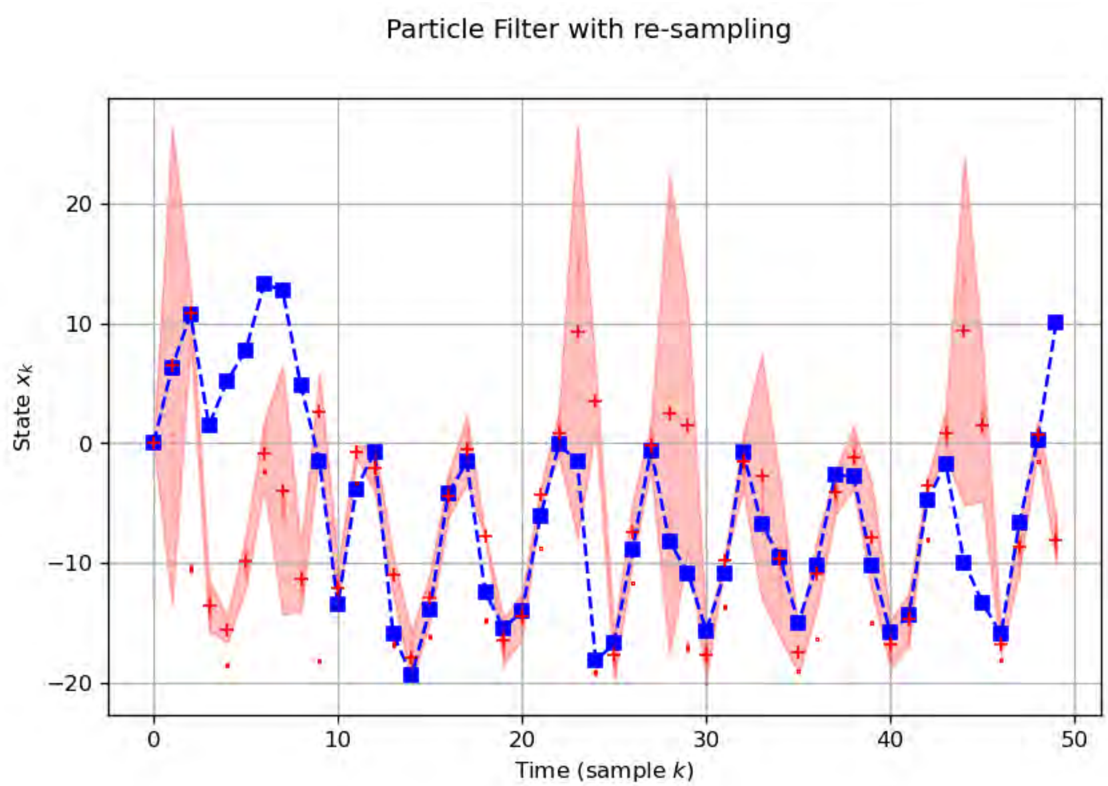
```
# 1) draw n samples from the prior
# 2) for each k = 1...T
#     a) draw samples x_k(i) from the importance distribution
#     b) compute the new weights
#     c) normalize the new weights

# initialize x^i_k and w^i_k matrices to keep track of state estimation distributions and weights
n = 200 # number of particles
xki = np.zeros((NumSteps,n),dtype=float)
wki = np.zeros((NumSteps,n),dtype=float)

# 1) draw n samples from the prior
x0_mu, x0_sigma = 0, np.sqrt(2)
x0 = np.random.normal(x0_mu, x0_sigma, n)
w0 = 1/n*np.ones(n)

# insert the samples from the prior into our matrices for keeping track of things
xki[0,:] = x0
wki[0,:] = w0

# initialize noise Gaussian parameters
u_mu, u_sigma = 0, 1
v_mu, v_sigma = 0, 1

# 2) for each k = 1...T
mean = np.zeros(NumSteps) # keep track of the mean of the particles
var = np.zeros(NumSteps) # keep track of the variance of the particles at each step
neff = np.zeros(NumSteps)

for k in np.arange(1,NumSteps,1):
    # a) draw samples x_k(i) from the importance distribution
    xki[k,:] = 1/2*xki[k-1,:] + 25*xki[k-1,:]/(1 + xki[k-1,:]**2) + 8*np.cos(1.2*(k-1)) + \
                np.random.normal(u_mu, u_sigma,n)
    # print(sum(xki[k,:]))
    # b) compute the new weights
    wki[k,:] = wki[k-1,:]*1/np.sqrt(2*np.pi)*np.exp(-0.5*(y[k] - 1/20*(xki[k-1,:]**2))**2)
    # c) normalize the new weights
    wki[k,:] = wki[k,:]/sum(wki[k,:])
    mean[k] = np.average(xki[k,:],weights=wki[k,:])
    var[k] = np.average((xki[k,:] - mean[k])**2,weights=wki[k,:])
    neff[k] = 1/sum(wki[k,:]**2)
    # draw new samples if the number of effective weights is < 20
    if neff[k] < 20:
        print(f"""Effective particles < 20 for step {k}""")
        ind = np.argsort(xki[k,:]) # index sort of the particles
        xki[k,:] = np.take_along_axis(xki[k,:],ind,axis=0)
        wki[k,:] = np.take_along_axis(wki[k,:],ind,axis=0) # sort the weights according to the particles
        bins = np.cumsum(wki[k,:]) # bins from which we are going to sample; cumulative sum of the weights
        uni = np.random.uniform(0,1,n) # uniform distribution used for re-sampling
        uni2 = np.random.uniform(0,1,n) # secondary random sampling for within bins
        for i in np.arange(0,n):
            for j in np.arange(n-1,-1,-1):
                if uni[i] >= bins[j]:
                    xki[k,i] = xki[k,j] + (xki[k,j+1] - xki[k,j])*uni2[i]
        # and reset the weights:
        wki[k,:] = w0

# track mean for later analysis
mean_pf_resamp = mean
```

```
Effective particles < 20 for step 2
Effective particles < 20 for step 4
Effective particles < 20 for step 8
Effective particles < 20 for step 9
Effective particles < 20 for step 11
Effective particles < 20 for step 13
Effective particles < 20 for step 16
Effective particles < 20 for step 19
Effective particles < 20 for step 21
Effective particles < 20 for step 24
Effective particles < 20 for step 26
Effective particles < 20 for step 29
Effective particles < 20 for step 31
Effective particles < 20 for step 35
Effective particles < 20 for step 36
Effective particles < 20 for step 39
Effective particles < 20 for step 42
Effective particles < 20 for step 46
Effective particles < 20 for step 48
```

3.3 [Q6.2] Plot mean and variance superposed to trajectory

In [8]:

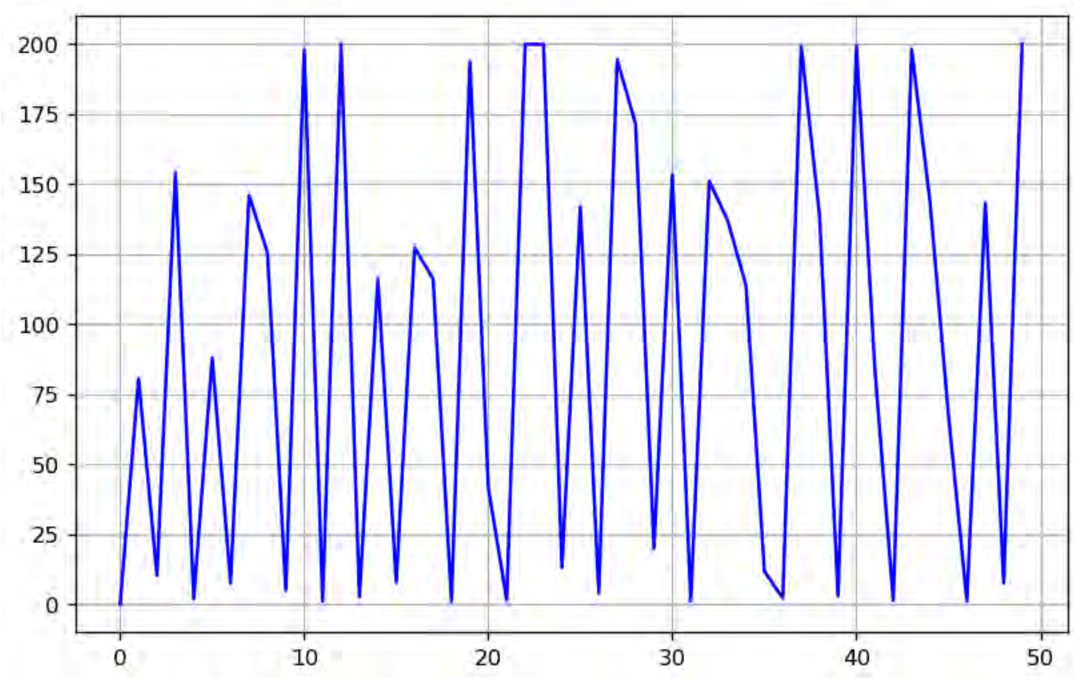
```
# Plot the trajectory
fig, ax = plt.subplots(figsize=(8,5), dpi=120)

ax.plot(np.insert(TimeScale,0,0),x,'b--')
ax.grid(True)
ax.plot(np.insert(TimeScale,0,0),x,'bs',markersize=6)
#plt.legend(['line','markers'])
ax.set_ylabel(r'State $x_k$')
ax.set_xlabel(r'Time (sample $k$)')
for k in np.arange(1,NumSteps,1):
    for i in np.arange(n):
        if wki[k,i] > 1e-3:
            ax.plot(k,xki[k,i],'ro',markersize=10*wki[k,i],alpha=0.3)
ax.plot(mean,'r+')
ax.fill_between(np.arange(NumSteps), mean-2*np.sqrt(var), mean+2*np.sqrt(var), alpha=0.25, color='r')
fig.suptitle('Particle Filter with re-sampling')

plt.show()
```



3.4 [Q6.3] Plot the n_{eff} as a function of time step k

In [9]:

```
# Plot the neff
fig, ax = plt.subplots(figsize=(8,5), dpi=120)

ax.plot(np.insert(TimeScale,0,0),neff,'b')
ax.grid(True)
plt.show()
```



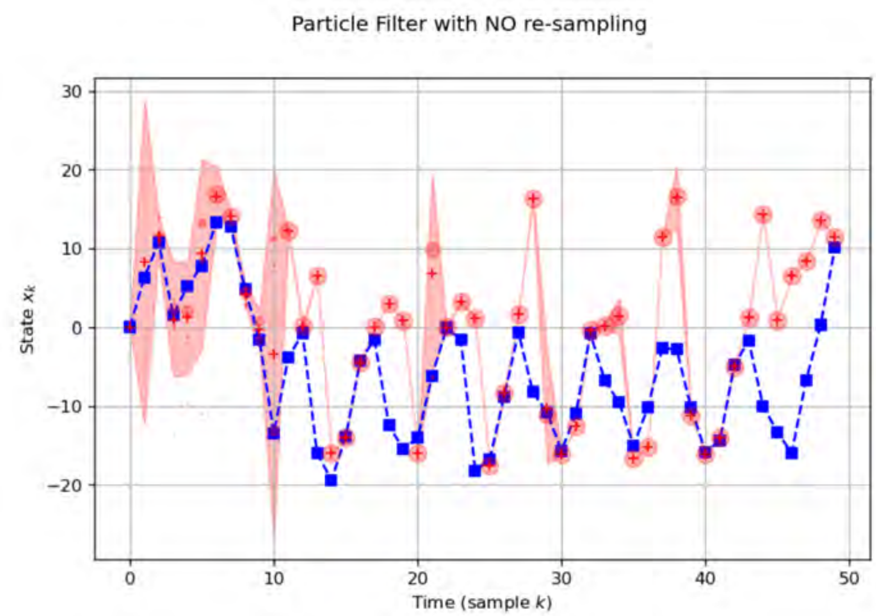
3.5 [Q6.4] Discussion of Results

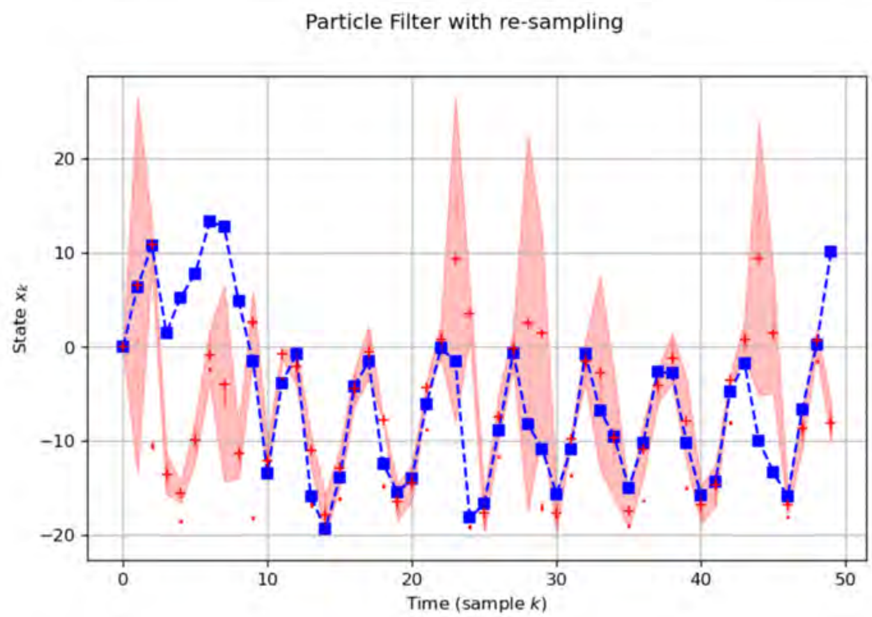
As we can see from the below plots (re-captured from the above code), the particle filter with re-sampling maintains a better tracking on the variance of the estimation, and a slightly better tracking per the MSE calculation below.

In [18]:

```
mse_pf = np.sum((mean_pf - x)**2)/NumSteps
mse_pf_resamp = np.sum((mean_pf_resamp - x)**2)/NumSteps
print(f"""MSE of the PF without re-sampling: {mse_pf}""")
print(f"""MSE of the PF with re-sampling: {mse_pf_resamp}""")
```

```
MSE of the PF without re-sampling: 107.36649647973412
MSE of the PF with re-sampling: 76.75540576179377
```





NOTES:

In [10]:

```
a = np.array([1, 2, 3, 4])
b = np.ones(4) + 1
a - b
a * b
j = np.arange(5)
2**(j + 1) - j
```

Out[10]:

```
array([ 2,  3,  6, 13, 28])
```

In [11]:

```
list = [0, 1, 2, 3, 4]
display([0, list])
len(x)
```

```
[0, [0, 1, 2, 3, 4]]
```

Out[11]:

```
50
```