Filtering on Physiological Signals

Nathaniel Carlson, Andrew Williams, Zac Yauney

Abstract

In the process of developing a spectrometry-based non-invasive glucose detector, one of the chief impediments is the noise present in the spectrometry data. This noise comes in two varieties: the small constant noise due to imperfections in the spectrometry process as well as intermittent spikes in the data from discrete events such as bumping the device. In this paper we employ state-spaced filtering methods to solve both of these problems. We solve the constant noise by replacing a signal with the filter's estimation of the true value of the underlying state. We solve the spike problem by cutting out the sections with spikes present and replacing the data with innovation from the filter algorithm. Results are promising, with a reduction in noise and a interpolation that reasonably approximates the true data.

1 Motivation and Overview

Background on problem, and goal Issues - two kinds of noise Rationale for methods

Diabetes is a major issue in the world and its prevalence is only increasing. One of the main components of diabetes treatment is keeping the patient's blood sugar within a healthy range. Doing this requires regular measurement of the patient's blood sugar levels so that the level can be adjusted either upwards or downwards. Currently the predominant means of making this measurement is sampling the patient's blood and taking the measurement directly. Due to the physical and logistical inconvenience, alternative non-invasive methods are highly desirable. While there have been many attempts at solving this problem using a variety of methods, none have been successful thus far. This paper pertains to a project which attempts to use a wrist-mounted spectrometer to obtain spectrographic data of the patient's blood. From this data it should then be possible to tease out the amount of glucose in the blood. The novelty in this approach is that we have access to more data than other teams have had due to the successful miniaturization of the spectrometer which allows it to be worn on the wrist. In addition, a linear variable filter allows the spectrometer to gather detailed data for a fairly wide range of wavelengths.

Unfortunately, glucose gives off a spectrum very similar to water and so the glucose signal is very faint. The faint signal means that noise is a major problem when attempting to measure glucose, as it diminishes an already small signal to noise ratio. Thus it is of great importance to develop a method for dealing with the noise. There are two main types of noise we want to deal with. The first is the consistent noise associated with the light sensors and the spectrometer apparatus as a whole. This noise has a relatively small amplitude but it can muddy the waters and make the signal harder to detect. The second type of

noise is relatively infrequent but has large amplitude. This noise is introduced by shocks in the real world, such as the spectrometer being bumped. Data displaying the spikes characteristic of this noise is useless for glucose detection and needs to be interpolated.

The Kalman Filter and particle filters are methods that allow us to solve both of these problems. By estimating the underlying state responsible for the given observations they can reduce noise, and by running the state equations forward and backward in time they can be used to interpolate the correct values for the data in regions where a spike has been introduced. This paper will describe our use of both methods to solve these noise problems.

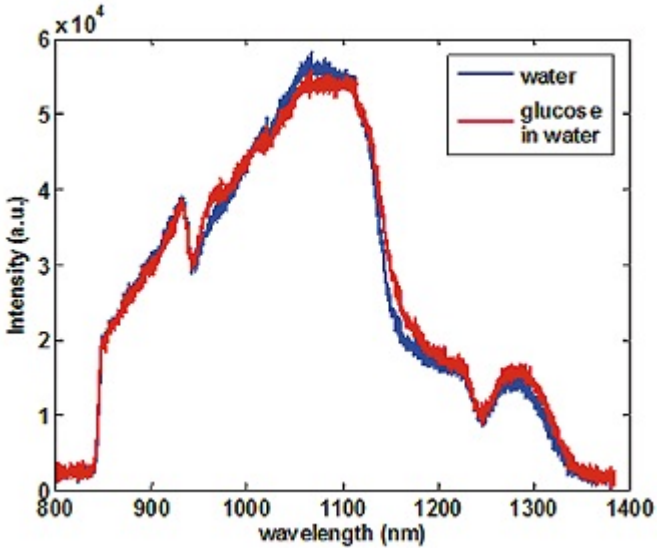

Figure 1: Spectral data for water and glucose. Note how similar they are to each other.

2 Data

The spectrographic data is a time series, with readings taken every 10ms. Each reading consists of 128 values, each giving the recorded light intensity at a different wavelength. These wavelengths are equally spaced from 872nm to 1654nm. When working with this data we typically use data from a pixel recording light at approximately 1050nm. This wavelength gives us the clearest signal and shows a visible dicrotic notch, evidence that we are indeed detecting the patient's heartbeat.

Our dataset consists of dozens of hours of data recorded over a period of several months and spanning several test subjects, but because these methods are not deep learning methods we will only use a small subset of that data to develop the filtering algorithms. The data is reliable in that we trust the source of the data, but there are reliability concerns regarding the accuracy of the spectrometer because it is still undergoing active development. The data may or may not be suitable for actually answering questions about blood glucose

content, the point of the research is to determine whether or not it is. The dataset should be perfectly suitable, however, for testing the efficacy of various filtering methods on the spectrometer data.

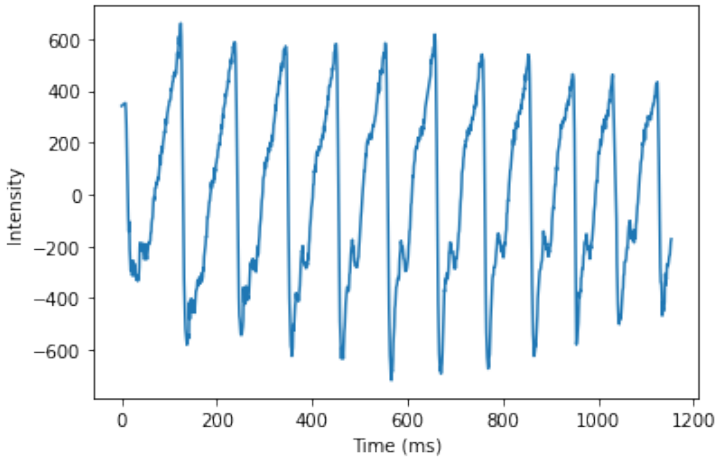

Figure 2: Normal spectrographic data at 1050nm. This image displays data recorded over a period of approximately 10 seconds.

3 Methods

First we discuss the data processing pipeline that was built to convert the raw data into appropriate input for our algorithms. Then we discuss two different type of state-space models that were employed: The Kalman filter and particle filtering.

3.1 Data Engineering

In its raw form the data consists of two sets of measurements interleaved together into one dataframe. The first type of measurement is called "dark current" and it records the amount of light the detector registers when the device's LEDs are turned off. Because this varies over time due to ambient light conditions and other factors, it is periodically re-measured. The other measurement is the actual spectrographic data taken with the LEDs on. We subtract the mean of each dark current reading from the subsequent spectrographic data so that ambient light conditions do not skew the readings.

After subtracting off the dark current we detrend the data using a high-pass Butterworth filter. This removes the influence of factors such as the patient's respiration. It also gives us data with zero mean.

Once these steps have been taken the data is ready for processing with the filtering methods.

3.2 Kalman Filter

The Kalman filter is a standard method for estimating true state based on noisy observation data. The Kalman filter algorithm essentially uses a state function to predict the next state of a system and then compares that state to the observed truth with an observation function. Then the algorithm updates the state based on that observation. One caveat with the Kalman filter is that a well defined state function is needed in order for the algorithm to correctly to interpret the data. In our case, we have no idea what these underlying state functions are, since the underlying dynamics are hopelessly complex.

One solution to this issue is to use an auto-correlation algorithm called SBR (Stochastic Balanced Realization) in order to estimate the underlying state equations. Using this algorithm, the states can be estimated, and we can compare the true (noisy) data with the output generated by the product of the observation matrix and the estimated state.

Using this approach we can see that reasonable approximations to the underlying state equations have been found by the SBR algorithm, since the data are so well represented by the estimated states.

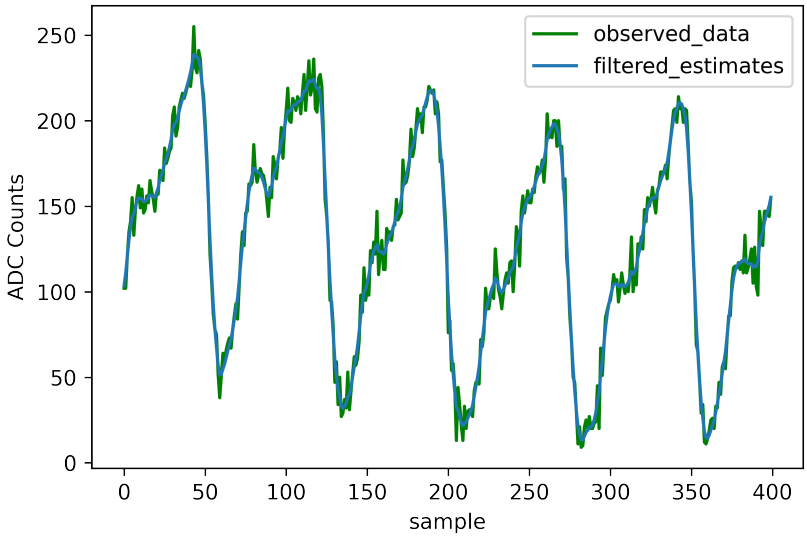

Figure 3: Using our underlying states, we can compare the output of the product of the observation matrix with the state and the true data.

3.3 Particle Filter

Particle filters are a class of Bayesian filtering algorithms that can be used to estimate the true state of a system based on noisy observations. The general idea behind particle filter methods is to sample a set number of K particles $p_1, \ldots, p_K$ from an initial distribution $X \sim \mu(x_0)$. Each of these particles represents a possible belief of the true state of the system. Each particle is assigned a weight

$w_1, \ldots, w_K$, e.i. the probability that a particular particle represents the true system state. At the beginning we have no reason to believe that any one particle is a particularly good estimate so we initialize the weights uniformly to $1 \,/\, K$.

We look to the Particle filter algorithm as an improvement over the Kalman filter to account for nonlinear state functions. The Kalman filter requires that the state function be linear, but we have no guarantee that this is actually the case. Thus we use a probabilistic approach to sample the state space and then use the particle algorithm to account for the error introduced by our assumption that the state functions are linear.

Another benefit of this approach is that a distribution of possible states at each time step is generated, which allows us to quantify our uncertainty at each step. With this approach we may be able to weight the certainty at each step, and allow lower uncertainty states to have more influence in future feature extraction algorithms.

3.3.1 Predict

The first step of the algorithm is the predict step. Here we use our model to transition each of the particles into a new state. The particle filter algorithm requires some prior knowledge of how a system transitions from both state to state and state to observation. We use the linear Gaussian model parameters estimated from the SBR algorithm for the predict and observe steps in our particle filter algorithm. Thus the particles are transitioned from time t to $t+1$ by

$$p_{t+1,j} = Fp_{t,j} + \mathcal{N}(0, Q), \quad j = 1, \ldots, K$$

For our initial distribution $\mu(x_0)$ we sample from a multivariate normal distribution $\mathcal{N}(x_0, 10^{-6} \times I)$ where x_0 is the first state estimate obtained from the Kalman Filter applied to the same data. It should be noted that particle filters do not require that the system in question be either linear or Gaussian. However, we felt that while non-linear parameter estimation is possibly an interesting area of future work, it was outside of the scope of this project.

3.3.2 Update

After we compute the new states in the predict step we must update the particle weights according to how well their corresponding predictions match the noisy observations at a given time. The predicted observations for time i are obtained via

$$o_{t,j} = Hp_{t,j} + \mathcal{N}(0, R), \quad j = 1, \ldots, K \tag{1}$$

We measure how well each particle corresponds to our observations using the p.d.f. of the Cauchy distribution, that is at time t for each particle we compute

$$dist(j) = \frac{1}{\pi(1 + (z_t - o_{t.j})^2)}, \quad j = 1, \ldots, K \tag{2}$$

We multiply each of the weights by the result of the *dist* function between their corresponding estimates and the current observation. In a Bayesian framework the *dist* function is our likelihood giving us the probability of observing z_i

given a particular particle. Multiplying this by our prior distribution of weights gives us an updated posterior distribution for how well each particle represents the true state of the system.

We experimented with using the p.d.f of a normal distribution centered at z_i to measure goodness of fit but it gave very poor results as most of the weights would quickly go to zero. The Cauchy distribution p.d.f. has much more mass concentrated near its tails than the normal distribution. We hypothesize that the fatter tails in the Cauchy distribution compared to the normal distribution allowed the weight updates to be more forgiving not assigning a weight near 0 for particles that were slightly further away from a given observation.

3.3.3 Resampling

The naive particle filtering algorithm suffers from a degeneracy problem. If we start with particles drawn from $\mu(x_0)$ with uniformly distributed weights it is possible that only a small number of them will accurately reflect the true state of the system. This results in the majority of the particles being assigned an extremely low weight. If the algorithm is allowed to run like this for several iterations it will eventually collapse leaving us with no particles with meaningful predictive power. Thus, it is necessary to resample particles that have low weights and replace them with more probable ones. A good rule of thumb to determine when it is time to resample is the *effective* N defined by

$$N_{eff} = \frac{1}{\sum_{i=1}^{K} w_i^2} \tag{3}$$

When this value falls below a certain threshold we automatically resample our particles. We found that setting this threshold equal to the total number of particles K yielded good results. We use the systematic resampling algorithm implemented in the FilterPy package to determine which particles to resample. Systematic resampling divides the weights uniformly into N groups and uses a single random offest to determine the index to resample from in each group. After resampling we reset the particle weights to the uniform distribution as per standard practice

3.3.4 Estimate

Once we update the weights according to the new observation, we can make predictions about the current state of the system at time t by computing the mean and covariance of the particles and the observations they generated. .

$$\mu_{state_t} = \sum_{j=1}^{K} w_{tj} p_{tj} \ , \quad \Sigma_{state_t} = \sum_{j=1}^{K} w_{tj} (p_{tj} - \mu_{state_t})(p_{tj} - \mu_{state_t})^T \tag{4}$$

$$\mu_{obs_t} = \sum_{j=1}^{K} w_{tj} o_{tj} \ , \quad \sigma^2_{obs_t} = \sum_{j=1}^{K} w_{tj} (o_{tj} - \mu_{obs_t})^2 \tag{5}$$

Full pseudo code for our particle filter implementation is provide in Algorithm 1 located in the appendix.

4 Results and Analysis

We analyze the results of our state space models on two important tasks. The first is noise reduction, taking a noisy signal and reconstructing a smoother signal that is a better approximation of the "true value" of the underlying phenomenon. The second task is using the filter's innovation capacity to impute values for sections of the data where there was a corrupting influence, i.e. the subject bumped their arm against something at that point.

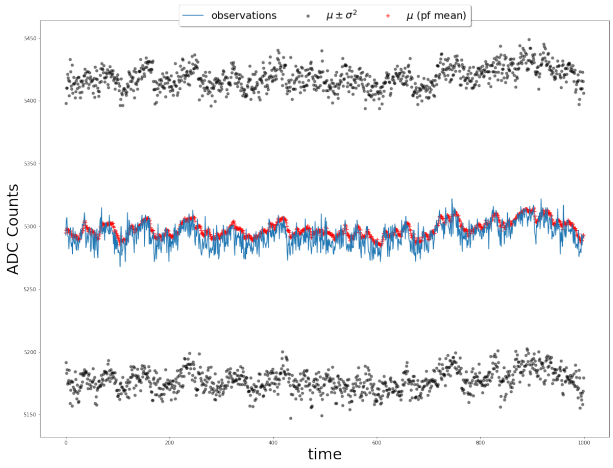

Figure 4: Results of the particle filter on a sample of 1000 readings of spectrometer data. The noisy observations are plotted in blue, while the particle filter mean and variance estimates for the observations are in red and grey respectively.

4.1 Noise Reduction

One method to improve our estimates of the data is to apply a Kalman smoothing algorithm. The Kalman smoother algorithm essentially computes a backward pass of the state equations, taking into account future and past data in order to give a better estimate of the states. This smoothing effect is seen as a reduction of noise in the states, which also translates to a reduction in noise of the predictions. Although this algorithm doesn't dramatically change the state estimates, the smoothed states are certainly less driven by the noise process.

The particle filter is advantageous since instead of giving us a single point estimate for the a state it provides a distribution over the space of all possible states. We found this to be useful when attempting to reduce noise in our data since it allows us to quantify the uncertainty of our predictions. The particle filter yielded good results on the data we applied it to. Figures 4 show its ability to model the underlying signal in the data amidst noisy observations. Figure 5 shows a more focused sample of observations and particle filter estimates. It is clear from this plot that the particle filter is robust to outliers and noise points

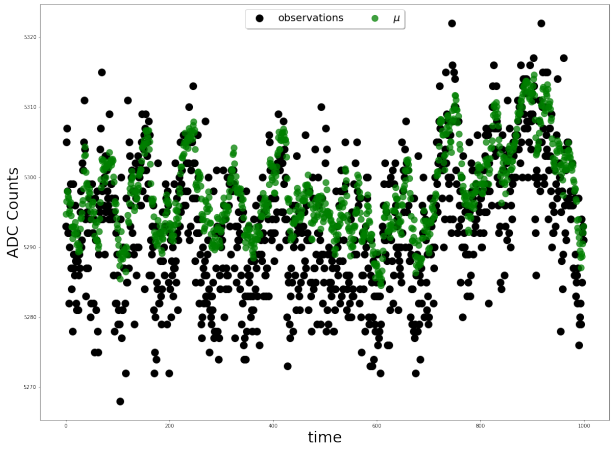

Figure 5: The results of the particle filter noise reduction

that are present in the data, overall it provides a much cleaner signal than the raw data.

4.2 Interpolating Bad Data

The standard Kalman Filter is a powerful way to estimate the true state of a noise corrupted process. One issue with the dataset given is that in order to successfully extract the most important features, long stretches of uninterrupted data are needed, and the data contain many noise artifacts that would interfere with extraction algorithms. One method to ameliorate this situation is to use the Kalman state equations to generate data to fill in the corrupted data.

Using the state equations developed in section 3.2, the interpolation first identifies the state of the data immediately prior to the corrupted data, then the state equations are allowed to run without an update from observations. This generates data which runs according to the same state equations used in the smoother. Although the data generated data does not exactly match the true data, the generated data is significantly less noisy than the corrupted data, making it much more suitable for our feature extraction algorithms.

One issue with this method is that we have no guarantee that the predicted data will line up with the actual data once the interpolation period ends. We might hope that the Kaman smoother would fix this, but we do not see such success in our experiments.

We see that the smoothed estimates are less noisy than the observed data, but still do not link up correctly with the observed data we see after the interpolation period. (The smoothing algorithm is designed to do just that: smooth. If we ask it to drastically change the course of the estimates we are not using the smoother correctly.)

One way to solve this issue is to implement a pseudo-particle filter strategy: namely by computing an interpolation step that overlaps the actual data by 50

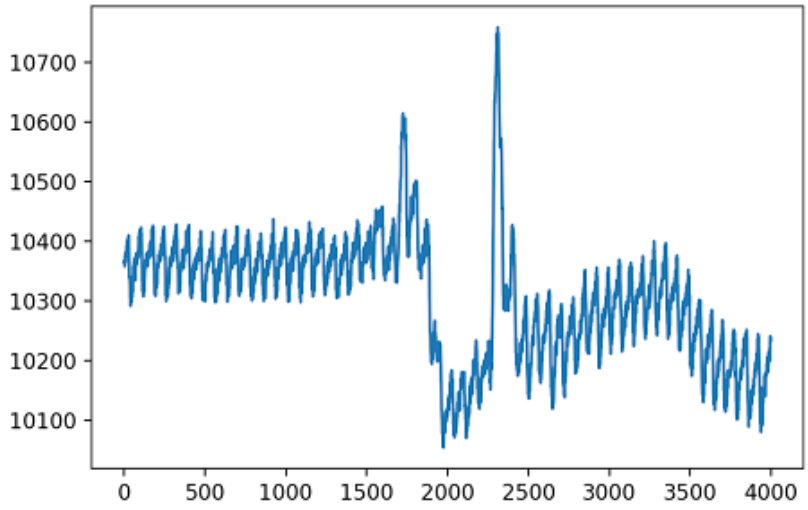

Figure 6: Example of a noise artifact that would interfere with feature extraction algorithms.

data points, and then compute the mean-squared error between the interpolation and the true data in the overlap section. Because the interpolation step is stochastic (since we introduce noise generated by the noise process at each transition step) we essentially compute a multi-step "particle" (really a sample path of a martingale defined by our noise process) and we compute the non-normalized likelihood of the last 50 data points given our observed data. We then only accept the interpolation if the MSE of the overlap is lower than a predetermined threshold.

Here we see that our efforts have resulted in a quite effective interpolation algorithm. More study is needed to perfect this process, but we are impressed with the success of this method.

Reviewing our results, we see our team was able to quite successfully estimate underlying state information and use it to overwrite sections of data corrupted by noise. Using this along with the Kalman smoother in order to extract noise reduced states, we have had great success in preparing a data set that can be easily fed into feature extraction algorithms and models which produce glucose estimates. Although the success of our efforts here is not entirely quantifiable until those feature extraction algorithms are more mature, we do expect to see a great improvement over those algorithms on data that has not been filtered and processed as we have done in this paper.

4.3 Ethics

Ethically, this project is quite strong. If successful, it stands to alleviate human suffering and improve general quality of life for millions of individuals, both those with and without diabetes. In theory it is possible that continuous monitoring

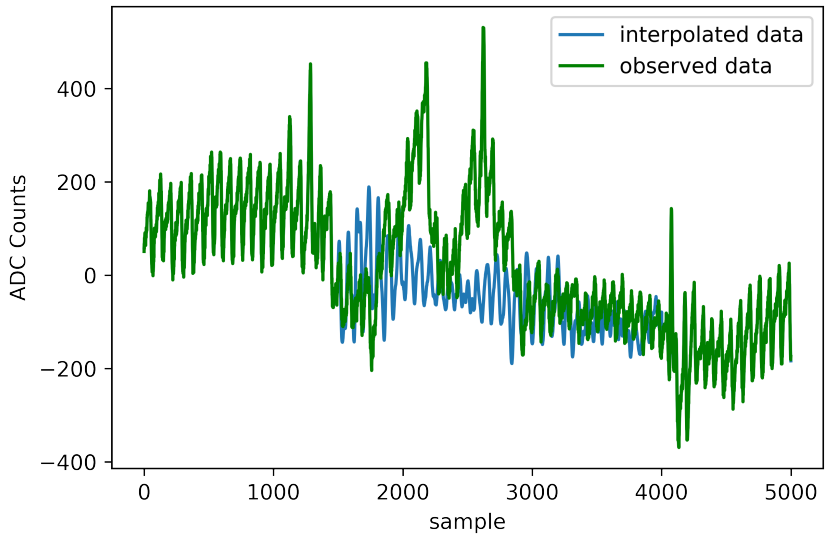

Figure 7: Here we see an example of generated data vs the true data, demonstrating the effectiveness of this technique.

of blood contents could be used nefariously if the patient's privacy was breached, but that is true of a plethora of technologies that are widely considered to be good. As long as the data remains in the right hands, it should inform the patient and their doctor in ways that will help them make the best decisions. There just needs to be strong safeguards along HIPAA lines protecting the device's data from those who might misuse it. Along these lines, the device and its software need to be fortified against intrusion in the form of cyberattacks.

If the device were to be sold without the algorithm being effective, then there could be serious medical consequences for the end users because they would take the wrong actions to control their blood glucose.

4.4 Future Research

In the future we will look in to using deep learning to estimate hidden states and interpolate the data. The weakness of our current method is that it requires some a priori knowledge about the distribution of the hidden states, and our intuition may not be correct in this case. A deep learning method might be able to approximate the state equations more effectively by analyzing a large volume of spectrographic data.

References

[1] https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python

[2] https://www.stats.ox.ac.uk/ doucet/doucet$_j ohansen_t utorialPF2011.pdf$

A Pseudo Code

Algorithm 1: Particle Filter

```
function Predict(F, Q, p_1, ... p_K)
    // Predict next particle states
    for i ∈ {1, ..., K} do
        p_i ←F p_i + N(0, Q)
    return p_1, ..., p_K
function Observe(H, R, p_1, ..., p_K)
    // Initialize observations
    o_1, ..., o_k ←0, ...,0
    for i ∈ {1, ..., K} do
        o_i ←Hp_i + N(0, R)
    return o_1, ..., o_K
function Update(H, R, z, p_1, ..., p_K, w_1, ..., w_K)
    o_1, ..., o_K ← Observe(H, R, p_1, ..., p_K)
    // Update weights based on current observation z
    for i ∈ {1, ..., K} do
        w_i ←w_i × (1 / (ε + (z − o_i)^2))
    return w_1 / Σ_{i=1}^K w_i, ..., w_K / Σ_{i=1}^k w_i
function Estimate(w_1, ..., w_K, o_1, ..., o_K)
    // Estimate μ and σ^2
    μ ←Σ_{i=1}^K w_i o_i
    σ^2 ←Σ_{i=1}^K w_i (o_i − μ)^2
    return μ, σ^2
// The main function
function ParticleFilter((z_i)_{i=1}^N, θ = (F, Q, H, R), K, N, μ(x0))
    // Initialize weights and particles
    w_1, ..., w_K ←1 /K, ..., 1 / K
    p_i ∼ μ(x_0) for i ∈ {1, ..., K}
    // Main algorithm
    for i ∈ {1, ..., N} do
        p_1, ..., p_K ← Predict(F, Q, (p_j)_{j=1}^K)
        w_1, ..., w_K ← Update(H, R, z_i, (p_j)_{j=1}^K, (w_j)_{j=1}^K)
        // Resample if effective N falls below threshold
        if 1 / Σ_{j=1}^k w_j^2 < K then
            p_1, ..., p_k ← SystematicResample((p_j)_{j=1}^K, (w_j)_{j=1}^K)
            w_1, ..., w_K ←1 /K, ..., 1 / K
        o_1, ..., o_K ← Observe(H, R, (p_j)_{j=1}^K)
        μ_i, σ_i^2 ← Estimate((w_j)_{j=1}^K, (o_j)_{j=1}^K)
    return (μ_i)_{i=1}^n, (σ_i^2)_{i=1}^n
```