COMP6247 - Kalman and Particle Filters

Luke McClure
lam3g17@soton.ac.uk
29573904

1 TIME VARYING AR - SIS/R

By using Kalman Filtering we were able to learn the covariances within a time varying AR in order to forecast the sequence.

We can use sampling techniques to model the likelihood function without explicitly calculating covariances, this is useful in cases where a likelihood function would be impossible to sample from directly and is the basis of SMCM methods Sequential Importance Sampling and Sequential Importance Resampling.

1.1 Importance Sampling

Importance Sampling relies on modelling a likelihood function using a number of weighted samples (particles), these weights dictate the likelihood an input at a timestep came from a specific particle via probability density functions.

This suffers with an issue known as weight degeneracy, where often one or two weights within the set will dominate. This results in a less accurate representation of the probability density function and so less accurate predictions.

Figure 1 shows the normalised magnitude of weights using the alpha value during a run of the SIS algorithm on a time varying AR problem. It is clear to see from how strong the colours present are that typically there is only a few particles being used at a given time step.

Effective Sample Size measures how many of the weights are in use by an importance sampling algorithm, the ESS accompanying this run is shown in Figure 3.

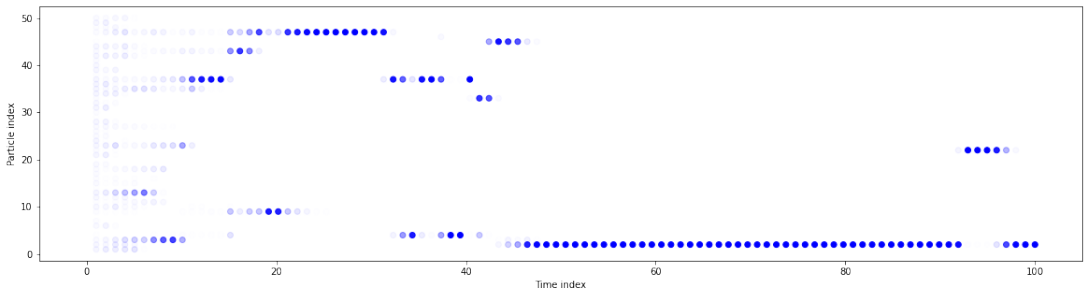

Figure 1: Weight Degredation using SIS

1.2 Introducing Resampling

Resampling allows the algorithm to choose particles that are more relevant to the problem and therefore replenish the samples and weights, this has the effect of focusing the particles sampled on to key areas and spreading what could have been one weight in SIS into a much wider proportion of the sample pool.

This combats weight degredation and improves on the accuracy of predictions the sampling algorithm is able to produce.

The normalised weight plot Figure 2 shows a stark difference when compared to Figure 1. Weights in every timestep are significantly lighter in alpha signifying smaller weight values, spread throughout the entire sample pool so as to gather information from more particles. There are points such as at $t = 58$ where these samples condense into a relatively smaller pool of larger weights before spreading back out again.

This is shown clearer when comparing the ESS of SIS with and without resampling in Figure 3. The plot for SIS collapses quickly into using less than 5 weights while the SIR algorithm with resampling fluctuates but uses about 70% of the available samples. The dips in sample size from Figure 2 are also clear in this plot, with several large dips in ESS.

Luke McClure
lam3g17@soton.ac.uk
29573904

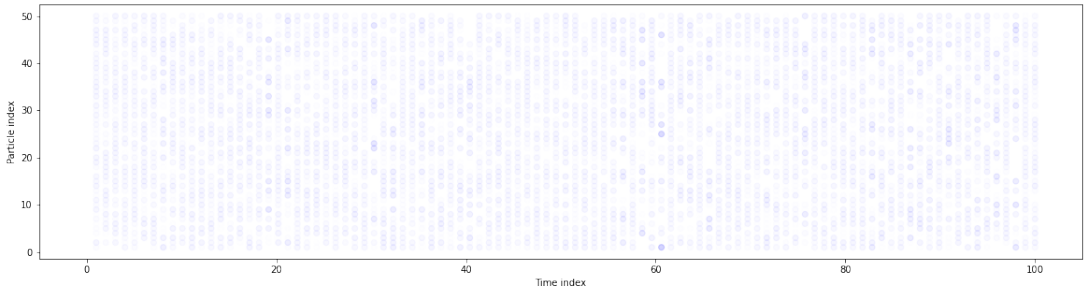

Figure 2: Weight Degredation using SIR

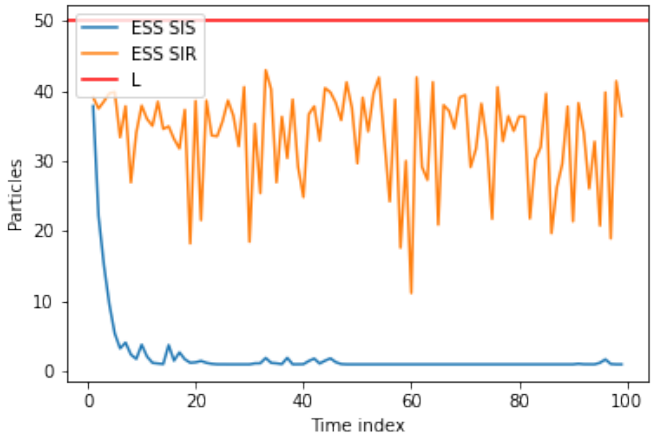

Figure 3: ESS Comparison between SIS and SIR

1.3 Comparing to Kalman Filter

To compare these three methods, I have run each on the same Time Varying AR problem and have compared the predictive outputs in Figure 4. It is clear that Sequential importance Sampling performs the worst, with wild swings above and below the real value. It appears to be far more influenced by the momentum of the input, with these swings typically diverging at the peaks where it wildly overshoots instead of following the real inputs. This makes sense when thinking back to the weight degeneracy problem, if a trend occurs long enough to completely dominate the particle weights then it takes a while for the algorithm to recorrect, resulting in this overshooting behaviour. Sequential Importance Resampling is the best performer of the three, tightly following the input. By resampling with every input allows the algorithm to keep an accurate track of the important particles and focus on these samples. Karman filter was close to the performance of SIR, but appeared to lag behind the real inputs a little. This may be due to the time needed to update the covariance matrix estimation in a magnitude able to effect the predictions tangibly.

Method	MSE
Sequential Importance Sampling	3.74
Sequential Importance Resampling	0.203
Kalman Filter	0.954

There is a clear difference in performance between SIS and SIR, the resampling step is vital to keep the particles sampled relevant to the state of the input as well as the underlying process being modelled.

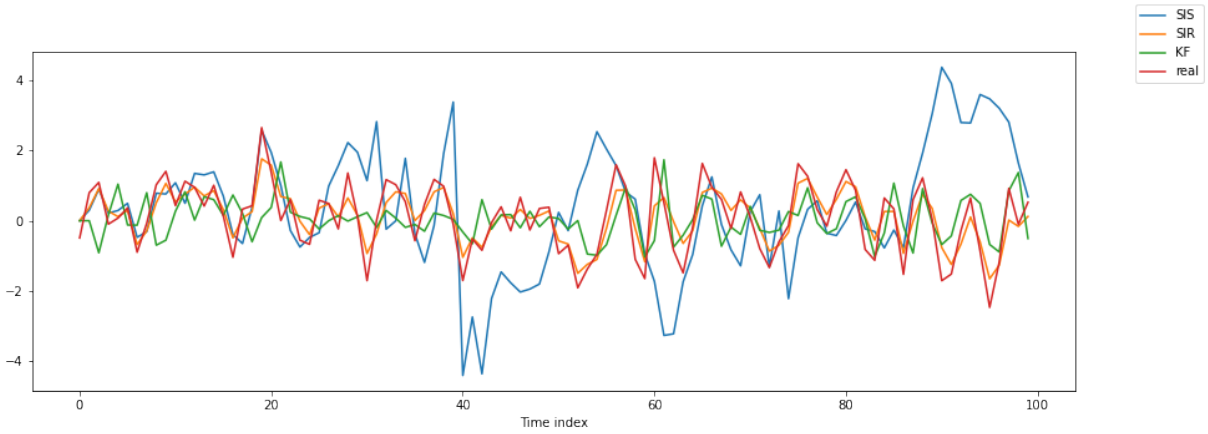

Figure 4: Prediction output of several methods

2 SEQUENTIAL LOGISTIC REGRESSION - EXTENDED KALMAN FILTER

2.1 Algorithm

In order to use Extended Kalman Filter to solve the sequential logistic regression problem, the portion of the kalman filter algorithm that preditcts θ needs to be changed such that it predicts the parameters of

$$f(\theta, \mathbf{x}) = \frac{\mathbf{1}}{\mathbf{1} + \exp\left(-\theta^{\mathbf{T}}\mathbf{x}\right)}$$

Achieving this so that EKF iteratively settles on the solutions for θ given the known classes requires the derivative of f wrt θ.

$$f'(\theta, \mathbf{x}) = \mathbf{f}(\theta, \mathbf{x})(\mathbf{1} - \mathbf{f}(\theta, \mathbf{x}))\mathbf{x}$$

As there are several dimensions to θ, these derivatives form a 2x1 Jacobian matrix H.

Algorithm 1: EKF algorithm update iteration solving Sequential Logistic Regression

Input: x_t,z_t,$\theta_{t-1|t-1}$,$P_{t-1|t-1}$
Output: $\theta_{t|t}$,$P_{t|t}$
$P_{t|t-1} \leftarrow P_{t-1|t-1} + Q$
$y_t \leftarrow f(\theta_{t-1|t-1}, x_t)$
$e_t \leftarrow z_t - y_t$
$H_t \leftarrow f'(\theta_{t-1|t-1}, x_t)$
$k_t \leftarrow \frac{P_{t|t-1}H_t}{H_t^T P_{t|t-1}H_t + R}$
$\theta_{t|t} \leftarrow \theta_{t|t-1} + k_t * e_t$
$P_{t|t} \leftarrow (I - k_t H^T)P_{t|t-1}$

2.2 Solution Found

algorithm 1 was able to converge on the solutions displayed in Figure 5, how the algorithm settled on the decision boundary shown has been animated in Figure 6.

2.3 Convergence

The optimal solution for this problem is given by:

$$w = \Sigma^{-1}(\mu_1 - \mu_2)$$

The EKF algorithm was rather slow in converging on this and didn't manage to within the 500 iterations given in Figure 7.

On further experimentation and running the algorithm with 5000 samples as shown in Figure 8, the algorithm was much closer to the optimal convergence. The decision boundary found between the two runs is identical in gradient, with the only difference being the decision boundary is much narrower and well defined. This reflects a greater confidence the algorithm has in the solution that has been found. Figure 6 reveals how the EKF algorithm is quick to settle near the optimal gradient of the boundary and simply gain confidence over time.

Luke McClure
lam3g17@soton.ac.uk
29573904

Figure 5: Decision boundary found by EKF (L - 500 sample, R - 5000 samples)

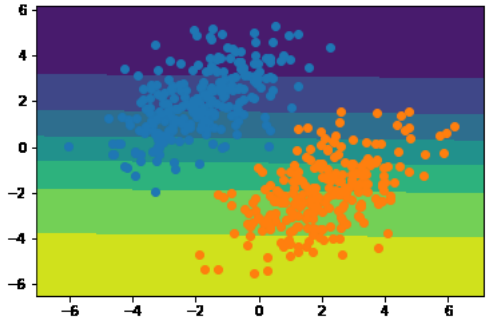

Figure 6: EKF solving logistic regression (animation)

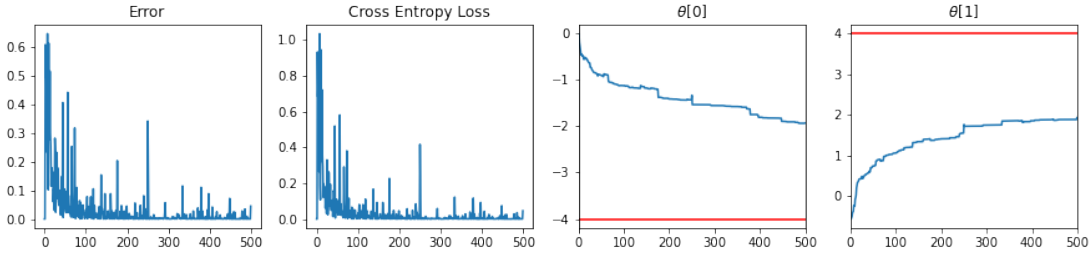

Figure 7: EKF to solve logistic regression

3 SEQUENTIAL LOGISTIC REGRESSION - SIR

3.1 Algorithm

Modifying the SIR algorithm from section 1 follows much the same construct as modifying the KF algorithm into the EKF algorithm used earlier. The same derivative of $f'(\theta, \mathbf{x})$ will be used but instead in a diffent context in how it will be used as we cannot directly alter the covariances with an iterative error term.

There will be three key differences between standard particle filter used and the filter used for sequential logistic regression.

(1) We are going to use two separate particle filters to find the two dimensions of θ.
(2) The probability density function for each dimension will use the respective dimension of $f'(\theta, \mathbf{x})$.
(3) The input of the pdf will be the target of the logistic function at that input x.

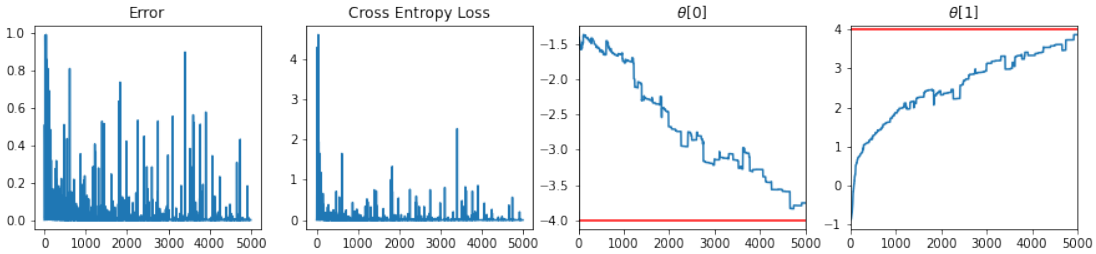

Figure 8: EKF to solve logistic regression (Extended)

To solve this problem, the probability density function's mean is going to be centred on H, in doing so we will be placing the highest weights closer to H. By setting the mean to the derivative that is dependant on the input x_t and θ, and by setting the target of the probability density function to the correct output of the logistic function y_t, we are effectively setting up a likeness to the iterative error method that was seen in EKF. This has the effect of steering the highest weighted particles towards the optimum of the derivative which happens to be θ, which therefore will solve the logistic regression.

In implementation this may be split into two identical algorithms to be run concurrently for each timestep to deal with each dimension of θ_t and the sampling of w_t more efficiently.

Algorithm 2: SIR algorithm update iteration solving Sequential Logistic Regression

Input: x_t,y_t,w_{t-1},z_{t-1}
Output: θ_t,w_t,,z_t
Data: $w_t : array, z_t : array$
$z_t \leftarrow \mathcal{N}(z_t|\phi z_{t-1}, \sigma_z^2)$
$\theta_t \leftarrow w_{t-1} \cdot z_{t-1}$
$H_t \leftarrow f'(\theta_t, x_t)$
$w_t \leftarrow \mathcal{N}(y_t|\beta H_t, \sigma_x^2)$
$w_t \leftarrow normalised(w_t)$
$\theta_t \leftarrow w_t \cdot z_t$
$w_t, z_t \leftarrow resample(w_t, z_t)$

The algorithm relies on two 2xL sized arrays to hold the particles and weights to handle both particle filters being run concurrently, with L being the number of particles the algorithm will track. Python's ability to process entire arrays inplace allows for elegant processing in this manner during implementation rather than accessing every l of L in both arrays sequentially.

3.2 Solution Found

algorithm 2 was able to converge on the solution displayed in Figure 9, how the algorithm settled on the decision boundary shown has been animated on the right.

3.3 Convergence

It is clear to see from the animation in Figure 9 that algorithm 2 is not as stable as algorithm 1. The animation shows the slope of the decision boundary wavering a lot compared to the animation of EKF, for a lot of the animation the decision boundary appears to stay close to parallel with the y axis, only starting to align with the proper decision boundary fairly late.

The plots in Figure 10 confirm this, showing that while the two parameters of θ do trend towards the optimal solution, there are plenty of periods where it is consistently heading away and the cross entropy loss is increasing. In comparison, the results from Figure 7 and Figure 8 show that while there may be small periods of increasing cross entropy loss, EKF is a lot more reliable in converging on a solution and staying on it.

This is most likely due to the direct error comparison and derivative update being employed in EKF, rather than trying to pseudo recreate it with the mean of the probability density function. By directly altering the covariances rather than trying to target the weights close to the derivative creates a much more accurate and consistent update schedule for EKF compared to SIR.

Figure 9: SIR logistic regression (R - animation)

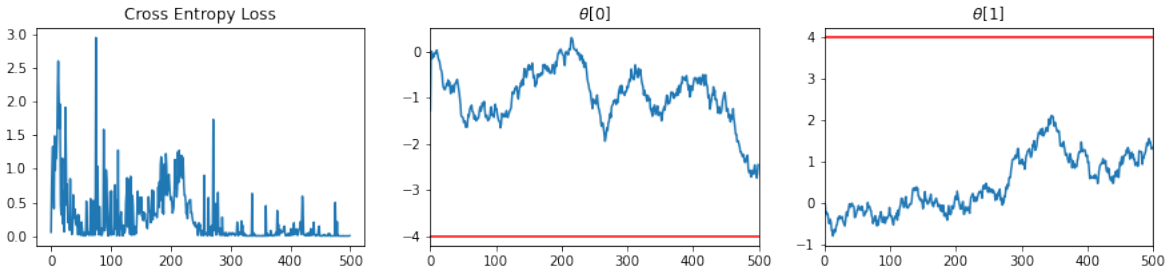

Figure 10: SIR logistic regression