GEOG5790M: Ensemble Kalman Filter

`https://github.com/ksuchak1990/geog5790m`

Keiran Suchak

05/05/19

Contents

1 Aim

This document has been produced as part of the second assessment for the GEOG5790M module. The aim of this module is to further develop Python programming skills. With this aim in mind, the module focuses on the following processes:

- Data processing,
- Analysis and visualisation, and
- Modelling.

As with the second assessment for the preceding introductory module, the scope for this assessment is left relatively open-ended with the hope that something of use can be produced. The code that has been produced therefore aims to implement a data assimilation technique known as the Ensemble Kalman Filter (EnKF) which can aid the simulation process when new data is provided regarding the system that is being modelled.

1.1 Problem Specification

Computational investigations often focus on modelling phenomena that we observe in the real-world. Such models represent our understanding of the systems that we study, and consequently are seldom perfect. As a consequence, simulation runs often diverge from what we observe in the real-world system. Two of the most common methods by which modellers attempt to combat such divergence are:

- State calibration: Estimating the initial state of the model.
- Parameter estimation: Attempting to calculate the best values for the model parameters.

These methods, however, are static — they are typically performed once before the model is set running. As we move forward into the age of Big Data, greater volumes of data are becoming available at a higher velocity. Consequently, we are able to obtain observations of the systems that we are modelling as the model runs.

Although these observations may provide us with an up-to-date idea of the state of the system, they are not without their own flaws. Observational data are typically sparse in time or space (or both). Furthermore, they have observation uncertainty associated with them. As a result, we need to

take care in combining the information contained within our model and the information gained from the observations.

One of the methods for combining the two pieces of information is the process of data assimilation. Data assimilation is typically used by meteorologists in the field of numerical weather prediction (Kalnay, 2003). There exist a wide range of data assimilation schemes — this investigation aims to implement the Ensemble Kalman Filter method, testing it with a very basic kinematic model.

2 Implementation

2.1 Model

The model developed for this assessment is a trivial one, named `Car_Model`. This model aims to simulate a car travelling at a constant speed in the $x - y$ plane. Given its speed in the x-direction and y-direction (u and v respectively), its position is updated each time-step by the rule:

$$x_{t+1} = x_t + u \times \Delta t \tag{1}$$

$$y_{t+1} = y_t + v \times \Delta t \tag{2}$$

where Δt is the time-step.

The model includes the facility to introduce normally distributed random noise to the update at each time-step, aiming to emulate an 'incorrect' model.

2.2 Ensemble Kalman Filter

The Ensemble Kalman Filter is a data assimilation method. It is based on the original Kalman Filter (Kalman, 1960). One of the issues faced by the original Kalman Filter, however, is that as the systems on which we focus become larger and more complex, it becomes much more computationally expensive to maintain the state covariance matrix; the Ensemble Kalman Filter was therefore developed, in which we represent the system state using a random sample and the covariance matrix using a sample covariance from the ensemble (Evensen, 2003).

We represent the system state, $\mathbf{X}$, as an ensemble of realisations:

$$\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N) \tag{3}$$

We represent the data, $\mathbf{D}$, as an ensemble:

$$\mathbf{D} = (\mathbf{d}_1, \ldots, \mathbf{d}_N) \tag{4}$$

Each column, $\mathbf{d}_i$ ($\forall i \in (1, N)$), is constructed from the original input data, $\mathbf{d}$, by adding a vector of normally distributed random noise:

$$\mathbf{d}_i = \mathbf{d} + \varepsilon_i \tag{5}$$

$$\varepsilon_i = \mathcal{N}(0, R) \tag{6}$$

where R is the data covariance.

The state is then updated as follows:

$$\hat{X} = X + K\left(D - HX\right) \tag{7}$$

where H is the observation operator, and K is the Kalman gain matrix given by (Mandel, 2009):

$$K = QH^T\left(HQH^T + R\right)^{-1} \tag{8}$$

where Q is the sample covariance (calculated from the ensemble) and H^T is the transpose of the observation operator.

This has been encoded as a Python class that wraps around a model, stepping it forward in time and updating its state with observations as a when such observations are available. The constructor accepts three arguemnts:

1. `model`: a model to be run.
2. `filter_params`: parameters for the filter, including the number of members of the ensemble and the frequency with which to assimilate observations.
3. `model_params`: parameters to be passed to the model when it is first constructed.

Furthermore, the constructor undertakes a number of checks to ensure that the model has the correct methods and attributes with which to interface with the filter, and that the correct parameters have been been provided.

After having constructed an instance of the filter, we can step the model forward in time using the `step()` method. This method wraps around the processes of prediction and assimilation; the model predicts forward each step and assimilates an observation when it is provided.

3 Usage

The above model and data assimilation scheme have been implemented in separate Python scripts. The implementations are then tested in the `main.py` script. This script aims to carry out the following tasks:

1. Use the model to create some synthetic truth data.
2. Use the model to create some synthetic observation data by adding normally distributed noise.
3. Create an instance of the EnKF based on the model, running it forward with observations being periodically assimilated.
4. Plot the result, comparing the EnKF model with the truth data and the observations.

The script then goes on to apply the above process with a variety of ensemble sizes and assimilation periods.

4 Future Improvements

The current implementation could be further improved as follows:

- Multi-threading: With the filter maintaining an ensemble of model states, we could implement multi-threading in order to step the ensemble members forward in parallel between assimilation updates.
- More complicated models: The model to which the filter has been applied here is very simple; the filter could be further tested using more complicated models to test its efficacy.
- Profiling: In maintaining an ensemble of models, the filter will ultimately require more memory; future investigations could seek to understand the space and time complexity scaling of the filter.
- Abstract model class: In order to ensure that all models are compliant with the form expected by the filter, a simple abstract model class could be developed from which all subsequent models would inherit.
- Access control: The only methods that a user will require access to will be the filter constructor and `step()`; consequently, we may wish to make other methods private such that the user does not mistakenly use them.

References

Evensen, G. (2003). The ensemble kalman filter: Theoretical formulation and practical implementation. *Ocean dynamics*, 53(4):343–367.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45.

Kalnay, E. (2003). *Atmospheric modeling, data assimilation and predictability*. Cambridge university press.

Mandel, J. (2009). A brief tutorial on the ensemble kalman filter. *arXiv preprint arXiv:0901.3725*.

A Required Packages

In order to run the code to which this document is attached, the following Python packages are required:

- `matplotlib`
- `numpy`
- `warnings`