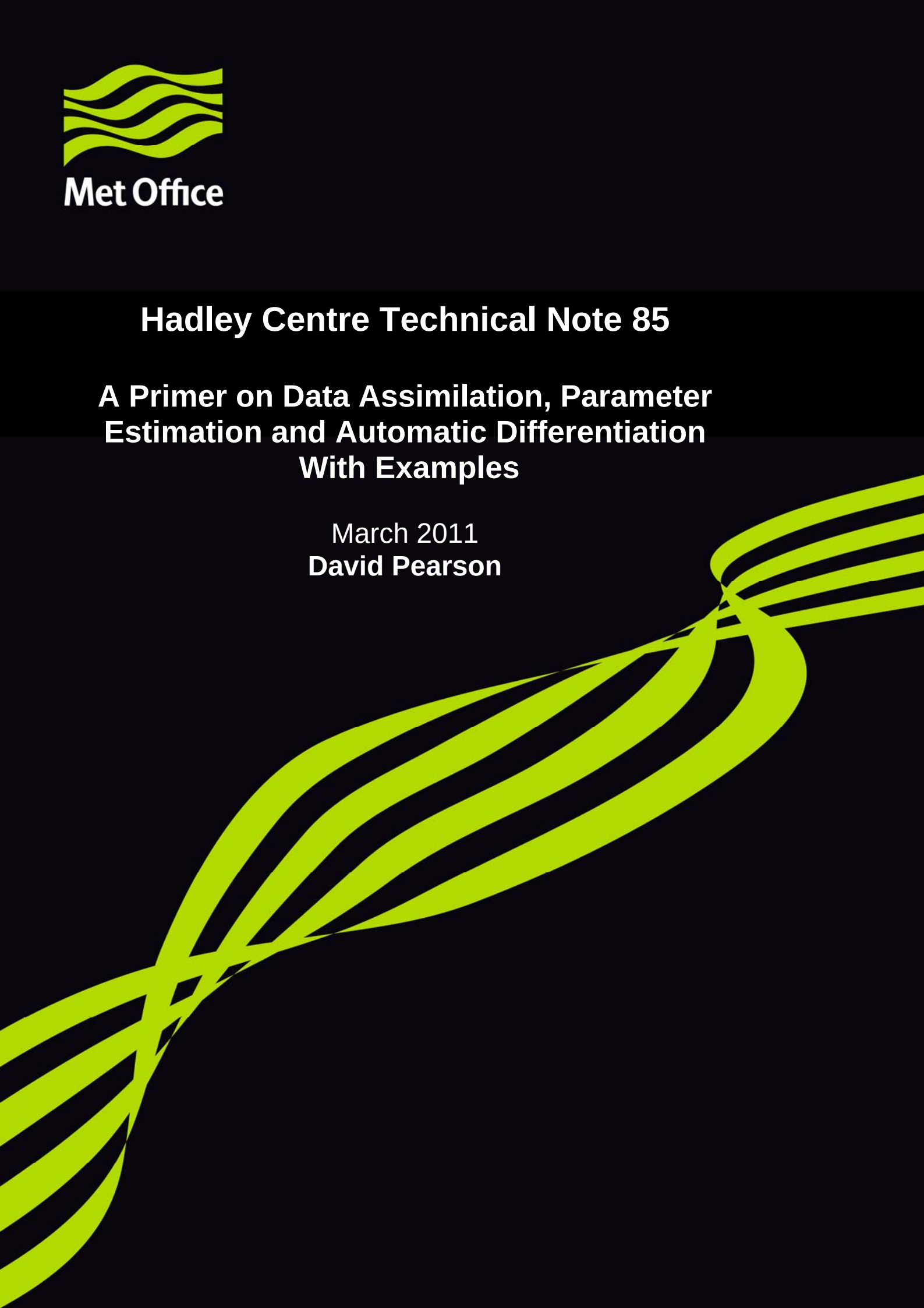
Met Office

Hadley Centre Technical Note 85

A Primer on Data Assimilation, Parameter Estimation and Automatic Differentiation With Examples

March 2011

David Pearson



A Primer on Data Assimilation, Parameter Estimation and Automatic Differentiation — With Examples

David Pearson, Met Office Hadley Centre

Contents

1 Preface

The underlying principles of data assimilation are easy, but the available literature, for the most part, is not. The art of applied data assimilation is also more difficult than the underlying principles might appear to indicate. I have written this note mainly to provide a fairly gentle introduction to these underlying principles, including some examples to make things concrete. Another intention was to help my own poor memory: if I do not write something down, I am likely to forget it. (I find some reassurance in Doob [Sne97].)

Once you have understood the basic principles, it should be easier to get into the research literature and practical applications. As an example of something that I have not written about here, but which is of central importance in practical atmospheric data assimilation [LBB$^+$00], I mention the *control variable transforms* — a topic that may be meaningless until you understand the basics, as presented here and in other tutorials. I hope also that some of the other tutorial material will be easier if you examine it alongside this note, in which I have adopted a different point of view.

Data assimilation is traditionally used to quantify states of models of systems, such as the atmosphere. It can also be used to estimate parameters that control systems, so I have included that application here.

A useful tool in the preparation of computer codes (by which I mean programs, not cryptography) is *automatic differentiation*, also known as *algorithmic differentiation*. In my opinion, this is becoming increasingly important in practical data assimilation, so I have also written a section on that.

Throughout, the presentation is informal and idiosyncratic, based on my own areas of research and understanding. Basically, it fills in what I did not understand when I started learning about data assimilation. In this spirit, I have included some brief digressions into more specialised areas — I hope these are useful. Where I am unsure of something, I say so.

Thanks to Edmund Ryan, who at the time of writing is at the University of Sheffield, for pointing out some errors and inconsistencies in an earlier version. Any remaining errors are, of course, entirely my own fault. I will be grateful for any more feedback from readers.

At this point, you might want to read §5, starting on page 9, before coming back to the next section. Be sure not to skip §5 altogether.

2 Introduction: What is Data Assimilation (DA)?

We have a system, and a numerical model of it. The sort of system we are interested in is something like the atmosphere, or an ocean, or the land surface at a certain geographical location. The model is then something like a spatially and temporally discretised mathematical description of one of these systems. (There are DA methods for continuous models, but I do not understand them, and they are not directly relevant here.)

The state of the model is denoted by its state vector at time t: $\boldsymbol{x}(t)$, or $\boldsymbol{x}_t$. If the time steps are numbered, it becomes $\boldsymbol{x}_i$. This *model state vector* contains all the numbers that describe the state of the model. For a model of the atmosphere, these could be: the velocity components, pressures, potential temperatures and humidities at all grid-points in a model of the atmosphere. For a 1-dimensional model of the land surface, they could be: soil water potential in soil layers, radiation intensity and carbon dioxide concentrations in vegetation layers, screen-height temperature, etc. Generally, they are a collection of numbers that are the values of prognostic variables in our model.

At the same time, the system is described by $\boldsymbol{\xi}(t)$, or $\boldsymbol{\xi}_t$ — this can be a vector if the system is physically discrete, or a set of functions (even though I use a vector notation) if it is continuous, or some hybrid of these. We call it the *system state vector*. Sometimes I get carried away and refer to the system as "the world".

We have at our disposal a set of measurements of the system. These are often temporally discrete. They need not be direct measurements of things that are represented by prognostic or diagnostic variables in the model. But we will see later that we need to be able to predict what these measurements should be, from the state of the model. For example, the system might be a land-surface chunk sitting in a lysimeter, which measures (by weighing) the amount of water in the soil, while our model is of soil water potential in layers in the soil. If we know the soil-water relations for the type of soil, we can predict what the weight should be from the modelled values of the water potential. Or if a satellite-borne instrument measures an intensity of infrared radiation from the atmosphere, this is a spatially integrated quantity (because the radiometer samples along a column in the atmosphere), modified by a complicated instrument function. Given the model's state (quantities at grid-points), and our knowledge of the instrument, we can predict that measurement.

The model is wrong in various systematic and random ways. So are the measurements: the lysimeter is not perfect, and the satellite's radiometer may have offset or gain drifts and pointing error.

Data assimilation is the process of using the measurements and the numbers in the model's state vector, to produce a best estimate of the state of the system. The *estimated state vector* is $\hat{\boldsymbol{x}}(t)$, $\hat{\boldsymbol{x}}_t$ or $\hat{\boldsymbol{x}}_i$. By "best", I mean we hope to get a statistically optimal estimate. For various reasons, this is often not achievable in practice (or even in principle), and we do something statistically sub-optimal instead. Much of the art of DA is concerned with doing something useful when the optimal cannot be attained.

There may be *parameters* in the model, that we are not too sure of. For example, the single-scattering albedo of leaves, or θ_{wilt}. Or, if we expand our definition of system a bit, the offset or gain of our modelled satellite instrument. These affect the data stream emerging from the model, or our predicted measurements, or both, but they are not generally considered to be evolving quantities in the usual sense. However, they can be treated by augmented DA systems

in ways that can be anything from *ad hoc* to mathematically rigorous.

3 Two Methods of Data Assimilation

The two fundamental modes of DA are *sequential* and *variational*. These are illustrated in Fig. (1) on page 81. A picture like this can be found in any introduction to DA.

Both parts of the figure represent the same model, having a 0-dimensional state vector: the scalar x. In both parts of the figure, we see a number of observations, represented by filled circles with error bars. The first of these is special: it is not an observation, but the *prior estimate* of the state at the beginning of the DA process. In fact, it need not come from an observation at all: it can be, and often is, an estimate from climatology, or from a previous DA process.

The top part illustrates sequential DA, which is exemplified by the Kalman filter. We start running the model, initialised by the prior estimate, and stop the model when the first observation of the system becomes available. In this simple case, the observation is a direct measurement of x, not a more complicated function of x. At this time, we have an observation with error bars, and a model state with error bars. The latter are known or assumed parts of our knowledge of the model and system — see §4, below. Given these two data (model and observation) and their error bars, we form the best estimate as a weighted sum of the two data, the weights being inversely proportional to their respective errors. The small vertical arrow represents this adjustment process. We then calculate the error bars on the estimate, from the observation errors and model errors. Then we switch the model on again, and proceed exactly as before. In this second iteration, our new estimate takes the place of the prior estimate. At the end of each iteration, we update the estimate's state and error bars, and this estimate is made available as a prior estimate of the model's state for the next iteration. The following pseudocode sums it up.

---

1. have a prior estimate of the state and its error bars
2. do
3. run the model for a while
4. get the observation and its error bars
5. update the estimate of the state and its error bars
6. end do

In a multi-dimensional system with multiple kinds of observations: the states of the system, model and estimate are represented as evolving state vectors, while the observations at any time are represented as a vector, and the error bars are represented as evolving covariance matrices.

The bottom part of Fig. (1) represents variational data assimilation. Here, we run the model for the whole length of the DA process, from some initial condition that is known as the "first guess" or "background estimate". It is the same thing as the prior estimate in sequential data assimilation. We then iteratively improve this by adjusting the initial condition until we have the best fit to the prior estimate and the observations, that is a continuous trajectory of the model. For example, the dashed line is a poor trajectory, because it goes outside all the error bars. The unbroken line is better, because it is closer to the prior estimate and the observations. Both lines are runs of the model without any adjustment except for the initial condition. That is, the whole trajectory is adjusted to find the best one, hence the name "variational". A disadvantage of variational DA, compared with sequential DA, is that it does not propagate the estimate's errors through time, so we do not have error bars for the estimated state vector. Variational DA ("Var") generalises to multiple dimensions and covariance matrices, much as does sequential DA. Much of practical Var is concerned with the iterative adjustment of the initial conditions. The following pseudocode sums up Var.

1. have a "first guess" estimate of the initial state and its error bars
2. do
3. run the model for the whole time of interest
4. get the observations and their error bars
5. compare the model trajectory with the observations and the initial state
6. if the comparison is not good enough, adjust the first guess estimate ...
7. ... but if it is good enough, exit this loop
8. end do

4 Systems, Models and Errors

4.1 Introduction

For the purposes of this paper, intuitive notions of model error, observation error and estimation error are probably sufficient. (You may want to skip ahead to §5 then §6 or §7 now.) A full understanding, though, depends on a deeper appreciation of what we mean by "error". In general, it depends on the system being analysed, and the assumptions that are made.

The next subsection, §4.2, is based on [Coh97a] and [Coh97b][1]. It covers models of deterministic systems.

4.2 Deterministic Systems

Let $\boldsymbol{\xi}_k$ be the continuous state of the world at time k. Ignoring any stochastic forcing (e.g. variations in Solar radiative input, cosmic ray intensity, and other inputs from the exterior), the world evolves in a deterministic manner:

$$\boldsymbol{\xi}_{k+1} = g(\boldsymbol{\xi}_k), \tag{4.1}$$

for some function g, called the system propagator. We do not know g, but we know it exists. In the computer, we have the model state vector, $\boldsymbol{x}_k$, for this k. This is a function of the system state vector. We write $\boldsymbol{x}_k^t$ to mean the *true* model state vector, which is defined as a correct discretisation of the system state vector:

$$\boldsymbol{x}_k^t = \mathcal{D}(\boldsymbol{\xi}_k), \tag{4.2}$$

where $\mathcal{D}$ is the discretisation operator. It is the superscripted t that is the distinguishing feature of the symbol $\boldsymbol{x}_k^t$.

In the computer, we have some function f, called the model propagator, that we use to model the evolution of the model state vector. It is, in fact, our program that models the weather or climate or whatever. It is, of course, imperfect, and we next find an expression for the model error. By the definition of f,

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k). \tag{4.3}$$

Then we can perform the following sequence of manipulations:

$$\begin{aligned}
\boldsymbol{x}_{k+1}^t &= \mathcal{D}(\boldsymbol{\xi}_{k+1}) && \text{(4.4a)}\\
&= \mathcal{D}(g[\boldsymbol{\xi}_k]) && \text{(4.4b)}\\
&= \mathcal{D}(g[\boldsymbol{\xi}_k]) + f(\boldsymbol{x}_k^t) - f(\boldsymbol{x}_k^t) && \text{(4.4c)}\\
&= f(\boldsymbol{x}_k^t) + \mathcal{D}(g[\boldsymbol{\xi}_k]) - f(\mathcal{D}[\boldsymbol{\xi}_k]) && \text{(4.4d)}\\
&= f(\boldsymbol{x}_k^t) + \boldsymbol{w}_k, && \text{(4.4e)}
\end{aligned}$$

[1] These papers are either very similar or identical.

where

$$\boldsymbol{w}_k = \mathcal{D}(g[\boldsymbol{\xi}_k]) - f(\mathcal{D}[\boldsymbol{\xi}_k]) \tag{4.5}$$

is the model error. It is "discretised propagated system minus propagated discretised system". The general idea is that, even if we discretise the system's state as well as we can (by whatever criteria you like), and use the best model we can, then the propagated model state is unlikely to be identical with the discretised propagated system. We generally assume that $\langle \boldsymbol{w}_k \rangle = 0$; and *a fortiori* that $\boldsymbol{w}_k$ is multivariate normal with zero mean.

Another way to think about it is as the failure of the commutativity diagram, (4.6).

$$\begin{array}{ccc} \boldsymbol{\xi}_k & \xrightarrow{\ g\ } & \boldsymbol{\xi}_{k+1} \\ \text{FAILS!} \quad \Big\downarrow \mathcal{D} & & \Big\downarrow \mathcal{D} \quad \text{FAILS!} \\ \boldsymbol{x}_k & \xrightarrow{\ f\ } & \boldsymbol{x}_{k+1} \end{array} \tag{4.6}$$

At first sight, you might think something funny is going on here. Equation (4.5) shows that $\boldsymbol{w}_k$ is the difference between two perfectly deterministic quantities, so it must be deterministic itself. But we treat it as a *random* vector in data assimilation. How can these facts be reconciled, without concluding that something has gone badly wrong? In fact, although $\boldsymbol{w}_k$ is indeed deterministic, it is unknown and unknowable. Therefore we treat it as a random vector. I think a Bayesian approach would be most consistent from here on, but since I do not know how to do this, I continue to treat $\boldsymbol{w}_k$ as a random vector in what follows.

[Coh97a] analyses observation errors and errors of representation in ways related to the above treatment, but these are not covered here.

The best estimate of the model of the state, also called the *analysis*, is in error as well. The error is called the *analysis error*, and it is described by the analysis error covariance matrix. The analysis is not the same as $\boldsymbol{x}_k^t$ because our data assimilation system does not give the true value, but an estimate of it. The analysis is "contaminated" by the various kinds of error.

4.3 Stochastic Systems

What if the system has a stochastic component? A system that we are interested in could certainly behave in such a way. For example, a small area of the surface of the land could be influenced by external phenomena that are unknown, and therefore might be treated as stochastic.

Now the system has a deterministic part as above, and an additive stochastic component:

$$\boldsymbol{\xi}_{k+1} = g(\boldsymbol{\xi}_k) + \boldsymbol{\gamma}_k, \tag{4.7}$$

where $\boldsymbol{\gamma}_k$ is a random field or set of random fields. We can proceed as above, to get:

$$\begin{aligned} \boldsymbol{x}_{k+1}^t &= f(\boldsymbol{x}_k^t) + \mathcal{D}(g[\boldsymbol{\xi}_k] + \boldsymbol{\gamma}_k) - f(\mathcal{D}[\boldsymbol{\xi}_k]) && \text{(4.8a)} \\ &= f(\boldsymbol{x}_k^t) + \boldsymbol{w}_k, && \text{(4.8b)} \end{aligned}$$

where now

$$\boldsymbol{w}_k = \mathcal{D}(g[\boldsymbol{\xi}_k] + \boldsymbol{\gamma}_k) - f(\mathcal{D}[\boldsymbol{\xi}_k]). \tag{4.9}$$

Can we now assume that $\langle \boldsymbol{w}_k \rangle = 0$? If $\mathcal{D}$ is a linear operator, and $\boldsymbol{\gamma}_k$ has zero mean, then equation (4.9) reduces to (4.5), and we can treat this case just like that of the deterministic system. When things are more complicated, a further analysis should be done.

4.4 Discrete Systems

By "discrete", I mean that the system is fully described by a finite set of numbers, unlike the continuous system of the previous few paragraphs, which had an infinite number of degrees of freedom. For example, the system might be a remote sensing aircraft, described by its position vector (three numbers) and its orientation (another three numbers). And the system in §8, starting on page 18, is described by only two numbers, x and y.

Now, the discretisation operator is just the identity operator:

$$\boldsymbol{x}_k^t = \mathcal{D}(\boldsymbol{\xi}_k) = \mathcal{I}(\boldsymbol{\xi}_k) = \boldsymbol{\xi}_k. \tag{4.10}$$

So equations (4.5) and (4.9) become, respectively:

$$\boldsymbol{w}_k = g(\boldsymbol{\xi}_k) - f(\boldsymbol{\xi}_k), \tag{4.11a}$$

$$\boldsymbol{w}_k = g(\boldsymbol{\xi}_k) + \boldsymbol{\gamma}_k - f(\boldsymbol{\xi}_k), \tag{4.11b}$$

where $\boldsymbol{\gamma}_k$ is now a random vector.

4.5 Broken Models

Finally, we must think about models that are simply incorrect. For example, a discretisation may be inconsistent with the continuous equations; or the equations that are thought to describe the system, do not describe the system very well. If we are very lucky, this might lead to a model error that is stochastic with zero mean, as above. More likely, we will end up with model *bias*, i.e. a deterministic error, or a stochastic error with non-zero mean, and possibly a mean that increases in expectation. Model bias is not discussed further in this paper. See [Dd98] and references to and from it for further information.

Models can also be wrong, not because the equations are wrong, but because the parameters (e.g. coefficients and other fixed numbers) in them are wrong. This problem can often be dealt with by parameter estimation, which is discussed in §14.

5 A Note on Notation

The notation in this paper is not completely unambiguous, but I think everything is correct and makes sense in its context. I like Cantwell's justification on page 15 of [Can02]:

> It will usually be obvious from the context which function name applies to a given configuration of variables. In any case, what is important is the concept, not the symbols used to explain the concept.

We are trying to understand a system, which has a Greek bold symbol for its state: $\boldsymbol{\xi}$. This always means the system state vector.

We have a model of the system, and this model has a state that is denoted by a Roman bold symbol: $\boldsymbol{x}$. However, although $\boldsymbol{\xi}$ always means the state, $\boldsymbol{x}$ usually means the model, but can also mean the state. In the Appendices, anything goes.

Observations are, similarly, one step removed from the system, so they have a Roman bold symbol too: $\boldsymbol{y}$.

Our best knowledge about the system is embodied in the estimate, which has a hat: $\hat{\boldsymbol{x}}$.

In the development of the Kalman filter, the following notation will simplify the page considerably:

$$\boxed{\boldsymbol{S}_k = \boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k, \qquad \forall k.} \tag{5.1}$$

$\boldsymbol{I}$ is, as usual, an identity matrix. But I do not specify its size. Throughout this tutorial, I use the symbol $\boldsymbol{I}$, unadorned, to mean an identity matrix of the size that is necessary for the equation it is in to make sense.

Two standard-ish pieces of notation can be used for the statistical or probabilistic expectation: angle brackets, and the $\hat{\mathcal{E}}$ symbol. Thus, for some random vector $\hat{\boldsymbol{x}}$ (which can, if you like, contain only one entry and thus be a scalar), the expectation of $\hat{\boldsymbol{x}}$ can be written:

$$\hat{\mathcal{E}}[\hat{\boldsymbol{x}}] = \langle \hat{\boldsymbol{x}} \rangle. \tag{5.2}$$

The conditional expectation can be written in these two ways:

$$\hat{\mathcal{E}}[\hat{\boldsymbol{x}}|\boldsymbol{\zeta}] = \langle \hat{\boldsymbol{x}}|\boldsymbol{\zeta} \rangle, \tag{5.3}$$

for some other random vector $\boldsymbol{\zeta}$. I use the angle bracket notation in this paper. Random vectors and conditional expectation are summarised in §A, starting on page 59.

A superscripted T, attached to a vector or matrix, indicates the transpose of that vector or matrix. A superscripted t indicates a true value, e.g. $\boldsymbol{x}^t$ is the true discretisation of a system's state $\boldsymbol{\xi}$.

To avoid confusion about the term "linear", I must say that a linear transformation of a vector is just pre-multiplication by a matrix, and an affine transformation of a vector is a linear transformation followed by addition of another vector. Thus, $\boldsymbol{y} = \boldsymbol{S}\boldsymbol{x}$ is a linear transformation, while $\boldsymbol{y} = \boldsymbol{S}\boldsymbol{x} + \boldsymbol{\xi}$ is affine. I generally say "linear" to mean the deterministic part of a transformation, but often this is followed by the addition of a stochastic vector, which makes the whole transformation, strictly speaking, an affine one. So equation 7.1 is an affine transformation, but the deterministic part (premultiplication by $\boldsymbol{M}_k$) is linear.

6 The Rest of This Paper

Var is easier to understand in an intuitive way than is sequential filtering. But the latter seems to be more fundamental than the former, so I describe and derive it first in what follows. After the mathematical discussion of the Kalman filter, I present a simple example of how it works, then a derivation of Var, and a simple example of that.

The Kalman filter is necessarily all about linear systems and linear observation operators (these being the functions that take model states and return predictions of observations), so the demo is linear. Var easily works with nonlinear systems, but does not consider model errors. So the demo for Var is nonlinear, but does not treat analysis errors. Later parts of the paper relax both these limitations.

Another large chunk of the paper is about practical aspects. This is where adjoints and other linearisations become essential.

The subject is vast and complicated, so many things are left out — I briefly list some of them later in the paper. I hope they will be more accessible after you have read all or some of the stuff here.

7 The Kalman Filter (KF)

7.1 Introduction

Here I derive the Kalman filter. The derivation is close to that of [BH97], both in detail and in its spirit of being fairly simple — but it is not identical. There are many other derivations in books and refereed papers — I like, for example, [Dee91].

A much more complete (and difficult) derivation is that of [Coh97a], which is dwarfed by, and partly covered by, [Tod99]. I only discuss the case of *discrete time*, i.e. where the model has discrete time steps, and observations are available at certain discrete times. The continuous time KF is much harder, and requires understanding of the measure-theoretic probability theory [CK00], or the non-measure-theoretic method of [Jaz70], which is just as difficult.

The Kalman filter and its relatives are all about optimal estimates of models of states of systems. For a given system that evolves in time (e.g. the atmosphere, or the trajectory and orientation of an aircraft), we have a mathematical model and a stream of observations — that is, we have to come up with a data assimilation method. Unlike Var and its relatives, which work with a whole system trajectory at once (hence the name "variational"), the KF and friends are *sequential*, meaning that optimal estimates are produced sequentially, and even in real time for many control engineering situations. Furthermore, the filter is *recursive*, meaning that at any time k, it only uses the state at the previous time $k-1$ and the observations at the present time k. It does not have to remember or look up earlier states or observations, or anticipate or look up future states or observations.

Most of this tutorial is about state estimation, which is the original and common purpose of the KF. I also discuss parameter estimation in §14, starting on page 35.

7.2 Assumptions

The plain KF is all about *linear* systems and *linear* observations. That is: the state vector evolves by matrix multiplication; and an observation vector comes from matrix multiplication of the state vector. General vector random variables are explained in §A, starting on page 59, and so are scalar normal ("Gaussian") random variables and multivariate normal ("MVN") random vectors.

The model propagates the true model state vector like this:

$$\boldsymbol{x}_{k+1}^t = \boldsymbol{M}_k \boldsymbol{x}_k^t + \boldsymbol{w}_k, \qquad \boldsymbol{w}_k \sim N(\boldsymbol{0}, \boldsymbol{Q}_k), \tag{7.1}$$

where $\boldsymbol{x}_k^t \sim (n_x \times 1)$ is the true discretisation of the system's state at a time indicated by its subscript; $\boldsymbol{M}_k \sim (n_x \times n_x)$ is the evolution operator or transition matrix at time k; and $\boldsymbol{w}_k \sim (n_x \times 1)$ is an additive noisy quantity at time k, distributed as a multivariate normal random vector having zero mean and covariance matrix $\boldsymbol{Q}_k$. We want to estimate $\boldsymbol{x}_k^t$ for all k. Our estimate — also called the analysis — is denoted $\hat{\boldsymbol{x}}_k$. The model is assumed to be unbiased, i.e. $\langle \boldsymbol{w}_k \rangle = 0$, so any drift of the model away from the system is a random walk phenomenon. The job of the KF is to use the observations to stop this random walk from forever staggering away in a divergent manner.

Our model propagates model state vectors forward in time:

$$\boldsymbol{x}_{k+1} = \boldsymbol{M}_k \boldsymbol{x}_k. \tag{7.2}$$

Equation (7.2) is a statement that defines what the model does. Equation (7.1) is a statement about the objective truth of the model. Both equations are correct, but they are to be interpreted in different ways.

Observations, or measurements, are also linear in $\boldsymbol{x}$:

$$\boldsymbol{y}_k = \boldsymbol{H}_k \boldsymbol{x}_k^t + \boldsymbol{v}_k, \qquad \boldsymbol{v}_k \sim N(\boldsymbol{0}, \boldsymbol{R}_k), \tag{7.3}$$

where $\boldsymbol{y}_k \sim (n_{y_k} \times 1)$ is a vector of observations at time k; $\boldsymbol{H}_k \sim (n_{y_k} \times n_x)$ is the observation operator, or observation matrix at time k; and $\boldsymbol{v}_k \sim (n_{y_k} \times 1)$ is the zero-mean *observation error* at time k, assumed to be multivariate normal with mean zero and covariance matrix $\boldsymbol{R}_k$. For a spatially continuous system, the latter is not simply instrumental error — see [Coh97a] for a discussion of the *error of representation.* Observe that vectors of observations can be of different lengths at different times (i.e. we do not necessarily have the same set of types of observations at each time step), which is why lengths have double subscripts, as in n_{y_k}. Like the model, the observation vector is assumed to be unbiased, i.e. the expectation of the observation error is zero.

I show below (in the paragraph beginning "The discussion so far" on page 13) that $\boldsymbol{w}_k$ starts at $\boldsymbol{w}_0$, but $\boldsymbol{v}_k$ starts at $\boldsymbol{v}_1$.

Now we look again — just for confirmation and confidence — at the covariance matrices of the model error and observation error:

$$\langle \boldsymbol{w}_k \boldsymbol{w}_k^T \rangle = \boldsymbol{Q}_k, \tag{7.4a}$$

$$\langle \boldsymbol{v}_k \boldsymbol{v}_k^T \rangle = \boldsymbol{R}_k, \tag{7.4b}$$

where $\boldsymbol{Q}_k$ is the model error covariance matrix; $\boldsymbol{R}_k$ is the observation error covariance matrix; and $\langle \cdot \rangle$ is the expectation operator. Recall that the product of a column vector with a row-vector, with the column to the left of the row, is a matrix. The matrices on the RHS of these two equations contain the expectations of their individual entries, e.g. $(\boldsymbol{Q}_k)_{i,j} = \langle (\boldsymbol{w}_k)_i (\boldsymbol{w}_k)_j \rangle$. $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$ are assumed to be known for all k.

Equations (7.4) do not look quite right at first. For some vector random variable $\boldsymbol{x}$, the covariance matrix is $\langle (\boldsymbol{x} - \langle \boldsymbol{x} \rangle)(\boldsymbol{x} - \langle \boldsymbol{x} \rangle)^T \rangle$. Only if $\langle \boldsymbol{x} \rangle = 0$, does this become $\langle \boldsymbol{x}\boldsymbol{x}^T \rangle$, which has the same from as (7.4). However, we have assumed that $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ have zero expectations, so the covariances as written in equations (7.4) are indeed correct.

We assume also that the error vectors are white, and orthogonal to each other, which means these three things:

$$\langle \boldsymbol{w}_k \boldsymbol{w}_m^T \rangle = 0, \qquad \text{for } k \neq m, \tag{7.5a}$$

$$\langle \boldsymbol{v}_k \boldsymbol{v}_m^T \rangle = 0, \qquad \text{for } k \neq m, \tag{7.5b}$$

$$\langle \boldsymbol{w}_k \boldsymbol{v}_m^T \rangle = 0, \qquad \text{for all } k \text{ and all } m. \tag{7.5c}$$

These are essential simplifying assumptions. Without them, I do not know if the KF would be feasible at all. (If any reader knows, please tell me.) It will later turn out that another three conditions are required. These will be introduced in the derivation, §7.4, and are listed together in §7.5 on page 16.

7.3 The Plan

The KF has a repeating predict-correct structure. We use our optimal estimate at time k, passing it through the model equations to predict the state at time $k+1$; and then correct this prediction by using observation data, which gives us our optimal estimate at time $k+1$. Then we do it all again, using our optimal estimate from time $k+1$ to get an optimal estimate at time $k+2$; and so on.

You can find a summary of a KF job in the "Timing Diagram", figure 2 on page 82. I copied this excellent pedagogical and mnemonic device from [Gel74].

Note: a time t_k is the time of validity of the k^{th} observation vector, not necessarily of the k^{th} time step of the model. The model can run through one or more time steps between observations.

7.4 The Details

This derivation of the Kalman filter does not proceed in the order in which the various results are used. If the ordering is confusing, you can look at the "Timing Diagram", figure 2 on page 82; and at the summary of use of the KF in §7.5 on page 16. We are going to work with these three things:

$$\boldsymbol{x}_k^t, \text{ the correct discretisation of the system at time } t_k; \tag{7.6a}$$

$$\hat{\boldsymbol{x}}_k^-, \text{ the modelled prior estimate of the system at time } t_k; \tag{7.6b}$$

$$\hat{\boldsymbol{x}}_k, \text{ the modelled posterior estimate of the system at time } t_k. \tag{7.6c}$$

The hats on $\hat{\boldsymbol{x}}_k^-$ and $\hat{\boldsymbol{x}}_k$ indicate that these are *optimal* estimates, and not just any old estimates. We work with general integer $k > 0$, with a short preamble for $k = 0$, which is

a special case. By "prior", I mean the "predict" part of the predict-correct algorithm that I mentioned in §7.3. $\hat{\boldsymbol{x}}_k^-$ will be a *prior* estimate, which is why it has a superscripted minus sign.

First, we make an estimate of the state of the discretised system at the beginning of the data assimilation job, i.e. at $t = t_0 = 0$; call this estimate $\hat{\boldsymbol{x}}_0$. It has a known or assumed error covariance matrix $\boldsymbol{P}_0 = \langle(\boldsymbol{x}_0 - \langle\boldsymbol{x}_0\rangle)(\boldsymbol{x}_0 - \langle\boldsymbol{x}_0\rangle)^T\rangle$. This is the same as the background error covariance matrix $\boldsymbol{B}$ in Var, 9.2. Where do $\hat{\boldsymbol{x}}_0$ and $\boldsymbol{P}_0$ come from? We can use whatever prior information we have, such as a climatology or a previous data assimilation job.

Next, we cycle the KF as follows, for $k > 0$, including $k = 1$, which is *not* a special case. Remember that a time t_k is the time of validity of the k^{th} observation vector, not necessarily the k^{th} time step of the model (as noted a short distance above).

For general $k > 0$, we first form the optimal prior estimate, thus:

$$\hat{\boldsymbol{x}}_k^- = \boldsymbol{M}_{k-1}\hat{\boldsymbol{x}}_{k-1}, \qquad \text{e.g. } \hat{\boldsymbol{x}}_1^- = \boldsymbol{M}_0\hat{\boldsymbol{x}}_0. \tag{7.7}$$

This seems like a natural way to proceed with the data and operators that are available. However, I am not saying that $\hat{\boldsymbol{x}}_k^-$ is without error; it does have the following *prior error:*

$$\boldsymbol{e}_k^- = \hat{\boldsymbol{x}}_k^- - \hat{\boldsymbol{x}}_k^t, \qquad \text{e.g. } \boldsymbol{e}_1^- = \hat{\boldsymbol{x}}_1^- - \hat{\boldsymbol{x}}_1^t. \tag{7.8}$$

This prior error has a *prior error covariance matrix,* $\boldsymbol{P}_k^- = \langle(\boldsymbol{e}_k^- - \langle\boldsymbol{e}_k^-\rangle)(\boldsymbol{e}_k^- - \langle\boldsymbol{e}_k^-\rangle)^T\rangle$. I will show, later, that under a certain reasonable assumption that I will also make plain later, $\langle\boldsymbol{e}_k^-\rangle = 0$, so

$$\boldsymbol{P}_k^- = \langle\boldsymbol{e}_k^-(\boldsymbol{e}_k^-)^T\rangle. \tag{7.9}$$

At time t_k, we also have the observation vector $\boldsymbol{y}_k$. We form the *prior observation increment* $\boldsymbol{d}_k$ like this:

$$\boldsymbol{d}_k = \boldsymbol{y}_k - \boldsymbol{H}_k\hat{\boldsymbol{x}}_k^-, \qquad \text{e.g. } \boldsymbol{d}_1 = \boldsymbol{y}_1 - \boldsymbol{H}_1\hat{\boldsymbol{x}}_1^-. \tag{7.10}$$

Then, to make the best estimate, which is the posterior estimate $\hat{\boldsymbol{x}}_k$, we use $\boldsymbol{x}_k^-$, $\boldsymbol{d}_k$ and an intermediate result called the *Kalman gain matrix* $\boldsymbol{K}_k$ (which is derived later):

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k\boldsymbol{d}_k, \qquad \text{e.g. } \hat{\boldsymbol{x}}_1 = \hat{\boldsymbol{x}}_1^- + \boldsymbol{K}_1\boldsymbol{d}_1. \tag{7.11}$$

We will later determine $\boldsymbol{K}_k$ in such a way that $\hat{\boldsymbol{x}}_k$ really is optimal, on a measure that is defined below. Why do we choose this way of doing it? Because if $\boldsymbol{y}_k = \boldsymbol{H}_k\hat{\boldsymbol{x}}_k^-$ then our prediction, $\boldsymbol{H}_k\hat{\boldsymbol{x}}_k^-$, is perfect, and we have no additional information to fold into the analysis. But if $\boldsymbol{y}_k \neq \boldsymbol{H}_k\hat{\boldsymbol{x}}_k^-$ then our prediction must be wrong, and we should adjust $\hat{\boldsymbol{x}}_k^-$ accordingly.

The discussion so far, and the Timing Diagram (Fig. 2 on page 82), indicate that $\hat{\boldsymbol{x}}_k$ comes from $\boldsymbol{y}_k$ and $\hat{\boldsymbol{x}}_k^-$. This is true for all k, except for the start of the sequence at $k = 0$. The first time at which we fully use the KF algorithm is $k = 1$. This is also the time of the first available observation vector $\boldsymbol{y}_1$. There is no $\boldsymbol{y}_0$ — if there was, we could just increase the k labels of everything by one, and get our "standard" Timing Diagram back again, as shown here. To start the filtering process, we need some kind of first guess at the state of the system — we call this $\hat{\boldsymbol{x}}_0$, as indicated above. And, as indicated above, this has an error covariance matrix $\boldsymbol{P}_0$. So in equations (7.5), $\boldsymbol{w}_k$ starts at $\boldsymbol{w}_0$, but $\boldsymbol{v}_k$ starts at $\boldsymbol{v}_1$.

Our $\hat{\boldsymbol{x}}_k$, for $k > 0$, has a *posterior error* $\boldsymbol{e}_k$, and this has a *posterior error covariance matrix* $\boldsymbol{P}_k$, as follows.

$$\boldsymbol{e}_k = \hat{\boldsymbol{x}}_k - \boldsymbol{x}_k^t, \qquad \text{e.g. } \boldsymbol{e}_1 = \hat{\boldsymbol{x}}_1 - \boldsymbol{x}_1^t, \tag{7.12}$$

and

$$\boldsymbol{P}_k = \langle(\boldsymbol{e}_k - \langle\boldsymbol{e}_k\rangle)(\boldsymbol{e}_k - \langle\boldsymbol{e}_k\rangle)^T\rangle. \tag{7.13}$$

It is useful to insist that $\langle\boldsymbol{e}_k\rangle = 0$, leading to

$$\boldsymbol{P}_k = \langle\boldsymbol{e}_k\boldsymbol{e}_k^T\rangle, \qquad \text{e.g. } \boldsymbol{P}_1 = \langle\boldsymbol{e}_1\boldsymbol{e}_1^T\rangle. \tag{7.14}$$

The next paragraph shows how this leads to a condition, $\langle \boldsymbol{e}_0 \rangle = 0$.

We start with the definition of $\boldsymbol{e}_k$, and make a series of substitutions:

$$\begin{aligned}
\boldsymbol{e}_k &= \hat{\boldsymbol{x}}_k - \boldsymbol{x}_k^t && (7.15a)\\
&= \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k \boldsymbol{d}_k - \boldsymbol{x}_k^t && (7.15b)\\
&= \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k (\boldsymbol{y}_k - \boldsymbol{H}_k \hat{\boldsymbol{x}}_k^-) - \boldsymbol{x}_k^t && (7.15c)\\
&= \boldsymbol{e}_k^- + \boldsymbol{K}_k (\boldsymbol{H}_k \boldsymbol{x}_k^t + \boldsymbol{v}_k - \boldsymbol{H}_k \hat{\boldsymbol{x}}_k^-) && (7.15d)\\
&= \boldsymbol{e}_k^- + \boldsymbol{K}_k \boldsymbol{H}_k (\boldsymbol{x}_k^t - \hat{\boldsymbol{x}}_k^-) + \boldsymbol{K}_k \boldsymbol{v}_k && (7.15e)\\
&= \boldsymbol{e}_k^- - \boldsymbol{K}_k \boldsymbol{H}_k \boldsymbol{e}_k^- + \boldsymbol{K}_k \boldsymbol{v}_k && (7.15f)\\
&= \boldsymbol{S}_k \boldsymbol{e}_k^- + \boldsymbol{K}_k \boldsymbol{v}_k && (7.15g)
\end{aligned}$$

The first equality is a definition; the second uses the definition of $\hat{\boldsymbol{x}}_k$; the third uses the definition of $\boldsymbol{d}_k$; the fourth uses the definition of $\boldsymbol{e}_k^-$ and equation (7.3); the fifth is a rearrangement; the sixth again uses the definition of $\boldsymbol{e}_k^-$; and the seventh uses the definition of $\boldsymbol{S}_k$. Since $\boldsymbol{K}_k$ is fixed, i.e. it is not random, its expectation is simply itself, and we immediately deduce the following from equation (7.15g):

$$\langle \boldsymbol{e}_k \rangle = \boldsymbol{S}_k \langle \boldsymbol{e}_k^- \rangle. \tag{7.16}$$

But we can transform $\boldsymbol{e}_k^-$ as well, as follows:

$$\begin{aligned}
\boldsymbol{e}_k^- &= \hat{\boldsymbol{x}}_k^- - \boldsymbol{x}_k^t && (7.17a)\\
&= \boldsymbol{M}_{k-1} \hat{\boldsymbol{x}}_{k-1} - (\boldsymbol{M}_{k-1} \boldsymbol{x}_{k-1}^t + \boldsymbol{w}_{k-1}) && (7.17b)\\
&= \boldsymbol{M}_{k-1} \boldsymbol{e}_{k-1} - \boldsymbol{w}_{k-1}, && (7.17c)
\end{aligned}$$

where the manipulations are similar in spirit to those in equations (7.15). Thus

$$\langle \boldsymbol{e}_k^- \rangle = \boldsymbol{M}_{k-1} \langle \boldsymbol{e}_{k-1} \rangle. \tag{7.18}$$

We can put equations (7.16) and (7.18) together in two different ways, to get the following two recurrence relations:

$$\begin{aligned}
\langle \boldsymbol{e}_k \rangle &= \boldsymbol{S}_k \boldsymbol{M}_{k-1} \langle \boldsymbol{e}_{k-1} \rangle, && (7.19a)\\
\langle \boldsymbol{e}_k^- \rangle &= \boldsymbol{M}_{k-1} \boldsymbol{S}_{k-1} \langle \boldsymbol{e}_{k-1}^- \rangle. && (7.19b)
\end{aligned}$$

Thus

$$\begin{aligned}
\langle \boldsymbol{e}_k \rangle &= \boldsymbol{S}_k \boldsymbol{M}_{k-1} \langle \boldsymbol{e}_{k-1} \rangle && (7.20a)\\
&= \boldsymbol{S}_k \boldsymbol{M}_{k-1} \boldsymbol{S}_{k-1} \boldsymbol{M}_{k-2} \langle \boldsymbol{e}_{k-2} \rangle && (7.20b)\\
&= \dots && (7.20c)\\
&= [\text{a product of matrices}] \times \langle \boldsymbol{e}_0 \rangle. && (7.20d)
\end{aligned}$$

So to make everything in the KF work nicely, we *assume* $\langle \boldsymbol{e}_0 \rangle = 0$, i.e. $\langle \hat{\boldsymbol{x}}_0 \rangle = \boldsymbol{x}_0^t$. Then $\langle \boldsymbol{e}_k \rangle = 0$ for all k.

This statement is a little problematic, for there was no indication before now that $\hat{\boldsymbol{x}}_0$ is a random vector. A Bayesian interpretation would be most appropriate here, but for the purposes of this tutorial, we just assume it is a random vector. After all, before the KF job is started, it is indeed unknown. Perhaps of more concern is the very assumption that $\langle \hat{\boldsymbol{x}}_0 \rangle = \boldsymbol{x}_0^t$. What reason do we have for this? Very little — but in order to proceed, we must make this assumption. The quotation from [Dee91] in §7.7 applies here.

Having made this assumption, we can use equation (7.18) and say that $\langle \boldsymbol{e}_k^- \rangle = 0$ for all k.

Now we have $\hat{\boldsymbol{x}}_k^-$ and $\hat{\boldsymbol{x}}_k$ for all k, and leaving aside the question of what is $\boldsymbol{K}_k$ until later, we can calculate $\boldsymbol{P}_k^-$ and $\boldsymbol{P}_k$, as follows.

$$\begin{aligned}
\boldsymbol{P}_k^- &= \langle \boldsymbol{e}_k^- (\boldsymbol{e}_k^-)^T \rangle && \text{(7.21a)}\\
&= \langle (\hat{\boldsymbol{x}}_k^- - \boldsymbol{x}_k^t)(\hat{\boldsymbol{x}}_k^- - \boldsymbol{x}_k^t)^T \rangle && \text{(7.21b)}\\
&= \langle (\boldsymbol{M}_{k-1}\boldsymbol{e}_{k-1} - \boldsymbol{w}_{k-1})(\boldsymbol{M}_{k-1}\boldsymbol{e}_{k-1} - \boldsymbol{w}_{k-1})^T \rangle && \text{(7.21c)}\\
&= \boldsymbol{M}_{k-1}\langle \boldsymbol{e}_{k-1}\boldsymbol{e}_{k-1}^T \rangle \boldsymbol{M}_{k-1}^T + \langle \boldsymbol{w}_{k-1}\boldsymbol{w}_{k-1}^T \rangle - \boldsymbol{M}_{k-1}\langle \boldsymbol{e}_{k-1}\boldsymbol{w}_{k-1}^T \rangle - \langle \boldsymbol{w}_{k-1}\boldsymbol{e}_{k-1}^T \rangle \boldsymbol{M}_{k-1}^T && \text{(7.21d)}\\
&= \boldsymbol{M}_{k-1}\boldsymbol{P}_{k-1}\boldsymbol{M}_{k-1}^T + \boldsymbol{Q}_{k-1}, && \text{(7.21e)}
\end{aligned}$$

where the first equality is a definition; the second uses the definition of $\boldsymbol{e}_k^-$ in equation 7.17a; the third uses equations (7.17c); the fourth expands the multiplication and takes fixed matrices outside the expectation angle-brackets. The fifth uses equations (7.14) and (7.4a); then starts from a reasonable assumption that $\langle \boldsymbol{e}_0 \boldsymbol{w}_{k-1}^T \rangle = 0$, and uses the following recurrence relation to show that it follows that $\langle \boldsymbol{e}_{k-1} \boldsymbol{w}_{k-1}^T \rangle = 0$.

$$\begin{aligned}
\langle \boldsymbol{e}_k \boldsymbol{w}_k^T \rangle &= \langle (\boldsymbol{S}_k \boldsymbol{e}_k^- + \boldsymbol{K}_k \boldsymbol{v}_k) \boldsymbol{w}_k^T \rangle \qquad \text{from equation (7.15)} && \text{(7.22a)}\\
&= \boldsymbol{S}_k \langle \boldsymbol{e}_k^- \boldsymbol{w}_k^T \rangle + \boldsymbol{K}_k \langle \boldsymbol{v}_k \boldsymbol{w}_k^T \rangle && \text{(7.22b)}\\
&= \boldsymbol{S}_k \langle (\boldsymbol{M}_{k-1}\boldsymbol{e}_{k-1} - \boldsymbol{w}_{k-1}) \boldsymbol{w}_k^T \rangle \qquad \text{from equations (7.5c) and (7.17)} && \text{(7.22c)}\\
&= \boldsymbol{S}_k \boldsymbol{M}_{k-1} \langle \boldsymbol{e}_{k-1} \boldsymbol{w}_K^T \rangle - \boldsymbol{S}_k \langle \boldsymbol{w}_{k-1} \boldsymbol{w}_k^T \rangle && \text{(7.22d)}\\
&= \boldsymbol{S}_k \boldsymbol{M}_{k-1} \langle \boldsymbol{e}_{k-1} \boldsymbol{w}_k^T \rangle \qquad \text{from equation (7.5a)} && \text{(7.22e)}\\
&= [\text{a product of matrices}] \times \langle \boldsymbol{e}_0 \boldsymbol{w}_k^T \rangle \qquad \text{by induction.} && \text{(7.22f)}
\end{aligned}$$

Similarly,

$$\begin{aligned}
\boldsymbol{P}_k &= \langle \boldsymbol{e}_k \boldsymbol{e}_k^T \rangle && \text{(7.23a)}\\
&= \langle (\boldsymbol{S}_k \boldsymbol{e}_k^- + \boldsymbol{K}_k \boldsymbol{v}_k)(\boldsymbol{S}_k \boldsymbol{e}_k^- + \boldsymbol{K}_k \boldsymbol{v}_k)^T \rangle && \text{(7.23b)}\\
&= \boldsymbol{S}_k \langle \boldsymbol{e}_k^- (\boldsymbol{e}_k^-)^T \rangle \boldsymbol{S}_k^T + \boldsymbol{K}_k \langle \boldsymbol{v}_k \boldsymbol{v}_k^T \rangle \boldsymbol{K}_k^T + \boldsymbol{S}_k \langle \boldsymbol{e}_k^- \boldsymbol{v}_k^T \rangle \boldsymbol{K}_k^T + \boldsymbol{K}_k \langle \boldsymbol{v}_k (\boldsymbol{e}_k^-)^T \rangle \boldsymbol{S}_k^T && \text{(7.23c)}\\
&= \boldsymbol{S}_k \boldsymbol{P}_k^- \boldsymbol{S}_k^T + \boldsymbol{K}_k \boldsymbol{R}_k \boldsymbol{K}_k^T, && \text{(7.23d)}
\end{aligned}$$

where the first equality is a definition; the second uses equation (7.15g); the third expands the product and takes fixed matrices outside the expectation angle-brackets. The fourth equality starts from a reasonable assumption that $\langle \boldsymbol{e}_0^- \boldsymbol{v}_k^T \rangle = 0$, and uses a recurrence relation for $\langle \boldsymbol{e}_k^- \boldsymbol{v}_k^T \rangle$ that is quite similar to that in equations (7.22).

Next, we need to derive an expression for $\boldsymbol{K}_k$, since it has not yet been specified. It could be anything (well, any matrix with the right shape), and equation (7.23d) would follow from equations (7.10) and (7.11) and the assumptions. However, we do want to make $\boldsymbol{K}_k$ optimal, and we do this by working on $\boldsymbol{P}_k$, as follows.

If you need to know about traces of matrices, and how to differentiate them, first read §B starting on page 68.

One measure of the badness of the posterior estimate, $\hat{\boldsymbol{x}}_k$, is the sum of the variances of its errors. These variances are the numbers in the leading diagonal of $\boldsymbol{P}_k$ in equation (7.23d). The sum is the trace of $\boldsymbol{P}_k$. To minimise the badness, we need to minimise the trace, and the thing we can adjust to do this minimisation is the Kalman gain matrix, $\boldsymbol{K}_k$. Variances are positive semidefinite, so there is no ambiguity in this. A measure involving the covariances would be problematic, because covariances can be negative.

We can write (7.23d) in a more useful (but not as nice) way like this (remembering that covariance matrices are necessarily symmetric):

$$\boldsymbol{P}_k = \boldsymbol{P}_k^- - \boldsymbol{K}_k \boldsymbol{H}_k \boldsymbol{P}_k^- - (\boldsymbol{K}_k \boldsymbol{H}_k \boldsymbol{P}_k^-)^T + \boldsymbol{K}_k (\boldsymbol{H}_k \boldsymbol{P}_k^- \boldsymbol{H}_k^T + \boldsymbol{R}_k) \boldsymbol{K}_k^T. \qquad \text{(7.24)}$$

The first term does not involve $\boldsymbol{K}_k$. The second and third are linear in $\boldsymbol{K}_k$, and the last is quadratic in $\boldsymbol{K}_k$. So we can apply equations (B.3) to the second and third terms:

$$\frac{d\,\mathrm{tr}(\boldsymbol{K}_k\boldsymbol{H}_k\boldsymbol{P}_k^-)}{d\boldsymbol{K}_k} = (\boldsymbol{H}_k\boldsymbol{P}_k^-)^T. \tag{7.25}$$

And we can apply equations (B.4) to the fourth term:

$$\frac{d\,\mathrm{tr}[\boldsymbol{K}_k(\boldsymbol{H}_k\boldsymbol{P}_k^-\boldsymbol{H}_k^T + \boldsymbol{R}_k)\boldsymbol{K}_k^T]}{d\boldsymbol{K}_k} = 2\boldsymbol{K}_k(\boldsymbol{H}_k\boldsymbol{P}_k^-\boldsymbol{H}_k^T + \boldsymbol{R}_k). \tag{7.26}$$

So, using the fact that the trace of a square matrix is equal to the trace of the transpose of that matrix,

$$\frac{d\,\mathrm{tr}(\boldsymbol{P}_k)}{d\boldsymbol{K}_k} = -2(\boldsymbol{H}_k\boldsymbol{P}_k^-)^T + 2\boldsymbol{K}_k(\boldsymbol{H}_k\boldsymbol{P}_k^-\boldsymbol{H}_k^T + \boldsymbol{R}_k). \tag{7.27}$$

This must be zero at the minimum, so finally we can say:

$$\boldsymbol{K}_k = \boldsymbol{P}_k^-\boldsymbol{H}_k^T(\boldsymbol{H}_k\boldsymbol{P}_k^-\boldsymbol{H}_k^T + \boldsymbol{R}_k)^{-1}, \tag{7.28}$$

which is what we need to use in (7.11) to finally get our optimal posterior estimate of the state.

This completes the derivation of the Kalman filter. The evolving error covariance matrices have to be calculated for each t_k, as an essential part of the filter. However, they provide important information in their own right, on the variances and covariances of the errors in our estimate of the state of the system.

7.5 Summary

The Kalman filter equations are (7.7), (7.21e), (7.28), (7.10), (7.11) and (7.23d). The conditions in equations (7.5) must be met, as must the following three:

$$\langle \boldsymbol{e}_0 \rangle = 0, \tag{7.29a}$$

$$\langle \boldsymbol{e}_0\boldsymbol{w}_k^T \rangle = 0, \qquad \forall k, \tag{7.29b}$$

$$\langle \boldsymbol{e}_0\boldsymbol{v}_k^T \rangle = 0, \qquad \forall k. \tag{7.29c}$$

The loopiness of the KF works like this:

---

1. Start with $\hat{\boldsymbol{x}}_0$ and $\boldsymbol{P}_0$
2. do $k = 1 \to n$
3. calculate $\hat{\boldsymbol{x}}_k^- = \boldsymbol{M}_{k-1}\hat{\boldsymbol{x}}_{k-1}$
4. calculate $\boldsymbol{P}_k^- = \boldsymbol{M}_{k-1}\boldsymbol{P}_{k-1}\boldsymbol{M}_{k-1}^T + \boldsymbol{Q}_{k-1}$
5. calculate $\boldsymbol{K}_k = \boldsymbol{P}_k^-\boldsymbol{H}_k^T(\boldsymbol{H}_k\boldsymbol{P}_k^-\boldsymbol{H}_k^T + \boldsymbol{R}_k)^{-1}$
6. get observations $\boldsymbol{y}_k$
7. calculate $\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k(\boldsymbol{y}_k - \boldsymbol{H}_k\hat{\boldsymbol{x}}_k^-)$
8. calculate $\boldsymbol{P}_k = \boldsymbol{S}_k\boldsymbol{P}_k^-\boldsymbol{S}_k^T + \boldsymbol{K}_k\boldsymbol{R}_k\boldsymbol{K}_k^T$ ... and also see equations (7.34)–(7.36).
9. end do

This shows the recursive nature of the filter, i.e. the fact that at any time, it only uses the estimate at the previous time and the observations at the present time. It does not have to remember or look up earlier states or observations.

Observe, also, that the matrices $\boldsymbol{P}^-$ and $\boldsymbol{P}$ do not depend on the estimates, $\hat{\boldsymbol{x}}^-$ and $\hat{\boldsymbol{x}}$, although the estimates do depend on the matrices. If you like, then, you can calculate the $\boldsymbol{P}^-$ and $\boldsymbol{P}$ matrices offline and store them for later use.

7.6 Another View: the Conditional Mean

The derivation presented above works by minimising the sum of the variances of the analysis errors — i.e. it is a *minimum variance* derivation, and the KF is thus a minimum variance filter. The quantity that is minimised is $\mathrm{tr}(\langle \boldsymbol{e}_k \boldsymbol{e}_k^T \rangle) = \langle \boldsymbol{e}_k^T \boldsymbol{e}_k \rangle$. This can be written with an identity matrix in the middle, like this:

$$\mathrm{tr}(\boldsymbol{P}_k) = \langle \boldsymbol{e}_k^T \boldsymbol{e}_k \rangle = \langle \boldsymbol{e}_k^T \boldsymbol{I} \boldsymbol{e}_k \rangle. \tag{7.30}$$

We could make a different scalar thing that we have to minimise, like this:

$$L_k(\boldsymbol{e}_k) = \langle \boldsymbol{e}_k^T \boldsymbol{J} \boldsymbol{e}_k \rangle, \tag{7.31}$$

where $\boldsymbol{J} \sim (n_x \times n_x)$ is *any* fixed (i.e. deterministic) positive definite matrix, and L_k is the scalar result of this equation. This includes the simple $\boldsymbol{J} = \boldsymbol{I}$. Since $\boldsymbol{e}_k$ is a random vector, then $L_k(\boldsymbol{e}_k)$ is a random variable, and it has an expectation $\langle L_k(\boldsymbol{e}_k) \rangle$.

Now let us consider another way of estimating the state:

$$\hat{\boldsymbol{x}}_k = \langle \boldsymbol{x}_k^t | \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_k \rangle, \tag{7.32}$$

which is called the *conditional mean* estimate. [Coh97a] shows that the value of $\hat{\boldsymbol{x}}_k$ that minimises the expected loss function, $\langle L_k(\boldsymbol{e}_k) \rangle$, is the conditional mean estimate. That is, the minimum variance estimate for any valid $\boldsymbol{J}$ is the conditional mean estimate.

Now we have that result, then by conditional mean reasoning we can easily show that the analysis is unbiased. If we write Y_k for all the observations up to and including time k, i.e. $Y_k = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k\}$, then by the chain rule for conditional expectations(§A.5):

$$\langle \boldsymbol{e}_k \rangle = \langle \langle \boldsymbol{x}_k^t | Y_k \rangle - \boldsymbol{x}_k^t \rangle = \langle \langle \boldsymbol{x}_k^t | Y_k \rangle \rangle - \langle \boldsymbol{x}_k^t \rangle = \langle \boldsymbol{x}_k^t \rangle - \langle \boldsymbol{x}_k^t \rangle = \boldsymbol{0}. \tag{7.33}$$

Here, the first equality expresses the definitions of $\boldsymbol{e}_k$ and the conditional mean estimate; the second uses linearity; and the third uses the chain rule.

7.7 Remarks

1. According to [BH97], there are three other expressions for the analysis error covariance matrix $\boldsymbol{P}_k$. They are algebraically identical with each other and with equation (7.23d), but may have different numerical properties. I have checked that they are correct. Here they are:

$$\boldsymbol{P}_k = \boldsymbol{P}_k^- - \boldsymbol{P}_k^- \boldsymbol{H}_k^T (\boldsymbol{H}_k \boldsymbol{P}_k^- \boldsymbol{H}_k^T + \boldsymbol{R}_k)^{-1} \boldsymbol{H}_k \boldsymbol{P}_k^-, \tag{7.34}$$

$$\boldsymbol{P}_k = \boldsymbol{P}_k^- - \boldsymbol{K}_k (\boldsymbol{H}_k \boldsymbol{P}_k^- \boldsymbol{H}_k^T + \boldsymbol{R}_k) \boldsymbol{K}_k^T, \tag{7.35}$$

and

$$\boldsymbol{P}_k = \boldsymbol{S}_k \boldsymbol{P}_k^-. \tag{7.36}$$

2. Here is another way to think about the update, which is given by equations (7.11):

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k (\boldsymbol{y}_k - \boldsymbol{H}_k \hat{\boldsymbol{x}}_k^-) \tag{7.37a}$$

$$= (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k \boldsymbol{y}_k. \tag{7.37b}$$

This says that the posterior estimate is a weighted sum of the prior estimate and the observations. $\boldsymbol{K}_k \boldsymbol{y}_k$ can be thought of as the observations, transformed into state space and weighted. $\boldsymbol{I} \hat{\boldsymbol{x}}_k^-$ is just the prior state estimate. $\boldsymbol{K}_k \boldsymbol{H}_k \hat{\boldsymbol{x}}_k^-$ is the prior estimate, transformed into observation space, and weighted and transformed back into state space. If $\boldsymbol{K}_k$ is big (in some sense), then the observations win. If $\boldsymbol{K}_k$ is small, the prior estimate wins.

3. There are many assumptions in the derivations of the Kalman filter. To quote [Dee91]:

   > Almost none of the information implicit in the statistical assumptions [... *list of equations* ...] is actually available in realistic situations. [...] There is no compelling theoretical reason to assume, for instance, that model error forcing is additive and that it only occurs at regular time intervals that correspond to the forecast model time-step. Any actual implementation of the KF algorithm therefore contains a multitude of approximations, all of which affect the calculation of the forecast error covariance.

   However, there are *robust filters* that are not too badly perturbed when the assumptions do not stand. For example, the H_∞-filter that is discussed in [Sim06] appears to be an important development.

4. Problems can arise with filter *divergence* and *stability*. These are generally outside the scope of this introduction, but it is important to at least be aware of the *square-root filters*. The covariance matrix $\boldsymbol{P}$ can become non-positive-definite due to rounding errors. This can be avoided by working with a factorisation of the covariance matrix, which is guaranteed to preserve positive-definiteness. See e.g. [BH97] for a simple introduction.

5. The minimum variance derivation of the KF does not require the random variables to be multivariate normal (MVN). However, it is useful if they are, because linear transformations of MVN random vectors produce new MVN random vectors. Since the whole of the KF analysis is linear, and the model and observation operators are assumed linear, then starting with MVN random vectors will ensure that the analyses and their errors are MVN as well.

8 A Demonstration of the Kalman Filter

8.1 The Rotational System

Here, I present a working example of the plain KF for a very simple system that has two state variables. The dynamics and the observation operator are linear, so the plain KF is directly applicable. A filter for a nonlinear system is discussed later, and its validity is demonstrated by showing that it gives the same results as the plain KF when applied to this simple linear system.

The system has a deterministic component and a stochastic component. The deterministic part consists of a particle in uniform circular motion, anticlockwise around a circle that is centred on the origin. Each time step, Gaussian noise is added to the x and y components of the position of the particle — this is the stochastic part. The state vector is:

$$\boldsymbol{x}^t = \boldsymbol{\xi} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{8.1}$$

It evolves like this:

$$\frac{d\boldsymbol{\xi}}{dt} = \boldsymbol{M}\boldsymbol{\xi} + \boldsymbol{w} = \begin{bmatrix} 0 & -\omega \\ \omega & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} w_x \\ w_y \end{bmatrix}, \tag{8.2}$$

where ω is the angular velocity of the particle, and $\boldsymbol{w}$ is a white, multivariate normal (MVN) noise vector with mean zero and covariance matrix $\boldsymbol{Q}$. We could say "bivariate" instead of "multivariate" for this case of two entries in the vector. This situation is described by §4.3 and §4.4.

The observation operator is just the identity matrix, and we add Gaussian zero-mean noise with a covariance matrix $\boldsymbol{R}$, to simulate measurement error. Thus an observation is given by

$$\boldsymbol{y} = \boldsymbol{\xi} + \boldsymbol{v}, \tag{8.3}$$

where $\boldsymbol{v}$ is a white bivariate normal noise vector with mean zero and covariance matrix $\boldsymbol{R}$.

Given an initial value for $\boldsymbol{\xi}$, and an angular velocity ω, we want to model the system to provide a trajectory (i.e. $\boldsymbol{\xi}(t)$) and a set of simulated observations. Given these, we want to use a KF to try to estimate this trajectory. So we need a way of integrating the equations of motion as an initial value problem.

The obvious discretisation is Euler's method. Thus for time step dt:

$$\boldsymbol{\xi}_{n+1} = \boldsymbol{\xi}_n + dt\boldsymbol{M}_n\boldsymbol{\xi}_n + \boldsymbol{w}_n, \tag{8.4}$$

but this is numerically unstable, and the trajectory spirals out exponentially away from the origin, even if the noise is omitted.[2]

However, an implicit scheme works fine. The deterministic part is defined by the following discretisation:

$$\frac{x_{n+1} - x_n}{dt} = -\omega\frac{y_{n+1} + y_n}{2}, \tag{8.5}$$

$$\frac{y_{n+1} - y_n}{dt} = \omega\frac{x_{n+1} + x_n}{2}. \tag{8.6}$$

This leads by simple rearrangement to the following:

$$\begin{bmatrix} 1 & \frac{\omega\,dt}{2} \\ -\frac{\omega\,dt}{2} & 1 \end{bmatrix} \begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & -\frac{\omega\,dt}{2} \\ \frac{\omega\,dt}{2} & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix}, \tag{8.7}$$

so:

$$\boldsymbol{\xi}_{n+1} = \begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \boldsymbol{M}_n\boldsymbol{\xi}_n = \frac{1}{1 + \omega^2 dt^2/4} \begin{bmatrix} 1 - \omega^2 dt^2/4 & -\omega\,dt \\ \omega\,dt & 1 - \omega^2 dt^2/4 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix}. \tag{8.8}$$

After every time step, the bivariate noise is simply added to the state vector. It is easy to verify (by doing the matrix multiplication) that $\boldsymbol{M}^T\boldsymbol{M} = \boldsymbol{I}$, i.e. that $\boldsymbol{M}$ is orthogonal. So it preserves the length of a vector, like this:

$$\boldsymbol{\xi}_{n+1}^T\boldsymbol{\xi}_{n+1} = \boldsymbol{\xi}_n^T\boldsymbol{M}_n^T\boldsymbol{M}_n\boldsymbol{\xi}_n = \boldsymbol{\xi}_n^T\boldsymbol{I}\boldsymbol{\xi}_n = \boldsymbol{\xi}_n^T\boldsymbol{\xi}_n. \tag{8.9}$$

Furthermore, we can see that it is, as required, a rotation matrix, because

$$\frac{(1 - \omega^2 dt^2/4)^2 + \omega^2 dt^2}{(1 + \omega^2 dt^2/4)^2} = 1, \tag{8.10}$$

i.e. $(1 - \omega^2 dt^2/4)/(1 + \omega^2 dt^2/4) = \cos\theta$ and $\omega\,dt/(1 + \omega^2 dt^2/4) = \sin\theta$ for some angle θ. Furthermore, in the limit $dt \to 0$, $\boldsymbol{M}$ becomes the rotation matrix for differentially small angles, as you can verify by use of Taylor's theorem. So our numerical scheme is not only numerically stable, it preserves lengths of vectors exactly, and rotates them as required. The evolution operator is a matrix, i.e. the evolution is linear (as required for the plain KF) and accurate.

Now if we make an initial guess at $\hat{\boldsymbol{x}}_0$ and $\boldsymbol{P}_0$, we have everything we need to cycle the Kalman filter as summarised in §7.5: $\boldsymbol{M}$ is defined in equation (8.8); $\boldsymbol{Q}$ and $\boldsymbol{R}$ are chosen in advance, when running the system forward; and $\boldsymbol{H}$ is the identity matrix, $\boldsymbol{I}$. If you use a high-level language such as Matlab or R, that can do a matrix operation with a single operator, then the whole KF code only takes a few lines. For example, here is a chunk of R code that does most of the KF work, after a forward run has been done to simulate the system. Initial values of matrices and vectors have been assigned before the loop that is shown, and simulated observations have been saved. The hash mark indicates a comment, and `%*%` is the matrix multiplication operator:

[2] N.B. this $\boldsymbol{w}$ is not the same as that in equation (8.2). Here, it is a temporally discrete stochastic process, while in (8.2) it is a temporally continuous stochastic process. I use the same notation, $\boldsymbol{w}$, anyway! Refer to [Jac10] or [Rob09] to learn more about continuous Wiener processes.

```
for (i in 1:n) {
   # Create next prior state:
   state.prior = transition %*% state.post
   # Calculate next prior cov:
   P.minus = transition %*% P.post %*% t(transition) + Q
   # Calculate Kalman gain matrix:
   if (got.obs[i]) { K = P.minus %*% t(H) %*% solve(H %*% P.minus %*% t(H) + R) } else
      { K = matrix( rep(0,4) , 2 ) }
   # Calculate obs. increment:
   obinc = matrix(obs.keep[,i]) - H %*% state.prior
   # Calculate posterior state:
   state.post = state.prior + K %*% obinc
   # Calculate posterior cov:
   S = matrix( diag(c(1,1)) , 2 , 2 ) - K %*% H
   P.post = S %*% P.minus
   # Make sure symmetric
   P.post = 0.5 * (P.post + t(P.post))
}
```

The model is propagated forward in small time steps between the arrival of observations, so $\boldsymbol{K}$ updates are not required for most time steps. Observe that there is a line that ensures that $\boldsymbol{P}$ is symmetric. Covariance matrices are, by definition, symmetric. But numerical errors can cause asymmetry, which can break the algorithm if symmetry is not forced. This breaking of the algorithm is discussed, along with other problems, in §6.6, "Divergence problems", of [BH97]. The square-root filter is also relevant here, as in §7.7 of this tutorial.

8.2 Forward Runs

First, let us look at two results from forward runs of the system. The coupled differential equations are integrated as in equation (8.8). The angular velocity is $\omega = 1$, giving a rotational period of 2π. Total run time is 100, so nearly 16 full rotations are simulated. The starting position is $x = 1$, $y = 0$. The time step is 0.2, and bivariate Gaussian noise is added at the end of the deterministic evolution every time step, with $\boldsymbol{Q} = \boldsymbol{I}$. Observations are simulated every 25 time steps, i.e. every 5 time units, which is a bit more than once per full rotation. The measurement is simply the values of x and y at observation time, with additive Gaussian noise, $\boldsymbol{R} = 10\boldsymbol{I}$. Figure 3 on page 83 shows what happens in this simple case. Observe that there is a tendency for the radius, r, to increase with time, even though the noise that is added to x and y is unbiased. We can understand this by considering what happens from one time step to the next. Given $\boldsymbol{\xi}_k$, we can calculate the squared radius at the next time step:

$$r_{k+1}^2 = \boldsymbol{\xi}_{k+1}^T \boldsymbol{\xi}_{k+1} \tag{8.11a}$$
$$= (\boldsymbol{\xi}_k^T \boldsymbol{M}^T + \boldsymbol{w}_k^T)(\boldsymbol{M}\boldsymbol{\xi}_k + \boldsymbol{w}_k) \tag{8.11b}$$
$$= \boldsymbol{\xi}_k^T \boldsymbol{\xi}_k + \boldsymbol{\xi}_k^T \boldsymbol{M}^T \boldsymbol{w}_k + \boldsymbol{w}_k^T \boldsymbol{M}\boldsymbol{\xi}_k + \boldsymbol{w}_k^T \boldsymbol{w}_k \tag{8.11c}$$
$$= r_k^2 + \boldsymbol{\xi}_k^T \boldsymbol{M}^T \boldsymbol{w}_k + \boldsymbol{w}_k^T \boldsymbol{M}\boldsymbol{\xi}_k + \boldsymbol{w}_k^T \boldsymbol{w}_k. \tag{8.11d}$$

Taking expectations and assuming that $\langle \boldsymbol{w}_k \rangle = 0$,

$$\langle r_{k+1}^2 \rangle - \langle r_k^2 \rangle = \langle \boldsymbol{w}_k^T \boldsymbol{w}_k \rangle. \tag{8.12}$$

Since $\boldsymbol{w}_k^T \boldsymbol{w}_k$ is positive semidefinite, then so is the expected change in squared radius. This is the expectation; the actual change at any time step can be negative, as in equations (8.11). This

system is very strange, and may not correspond to any physically realisable mechanism. Since ω is constant, it represents a particle whizzing around at greater and greater speeds and radii (in expectation), with some wobble about the spiral trajectory. Nevertheless, it is the simplest linear example that I could think of, that is not completely trivial and not univariate.

We can play around with the model and generate more interesting trajectories. Figure 4 on page 84 repeats Figure 3, the only difference being that now, $\boldsymbol{Q} = \begin{bmatrix} 3.01 & -3.0 \\ -3.0 & 3.01 \end{bmatrix}$. So the components of system noise are anticorrelated, leading to the diagonal sense of the trajectory in the figure.

8.3 Filtering Results

Figure 5, starting on page 85, shows the results of a KF job on the system and observations that are summarised in Figure 3. I used correct values of $\boldsymbol{Q}$, $\boldsymbol{R}$ and initial conditions; and a small, diagonal $\boldsymbol{P}_0$. Remember that the KF is trying to reproduce the coloured curve, given: the observations; the model of the system; a first guess at the initial state; and guesses at $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{P}_0$. Observe the occasional jumps at the times of observations. This is because the KF makes its adjustments to the estimated state when observations come in, as in §7.5, and the model evolves according to the noise-free dynamics between observations. Thus the filtered radius is stepwise constant in the third and fourth graphs of figure 5. The fourth plot shows how well (or badly) the KF estimate tracks the forward run's trajectory.

In the second page of Figure 5, the evolution of the error bars is very clear; they increase in time between observations, and abruptly decrease when observations are processed. The cyclical nature of the variances is clear. The covariance is close to zero. This is at it should be, because the model error $\boldsymbol{Q}$, and observation error $\boldsymbol{R}$, both have diagonal covariances, i.e. each has uncorrelated x and y parts. In the top two plots, observe that there is a roughly constant phase error in the estimate. It appears that the KF is trying to "catch up" with the true position — but I do not know why.

Figure 6, starting on page 87, is similar to figure 5. There are only two differences in how they were produced. The first is that the initial state is deliberately made very incorrect. The second is that the initial $\boldsymbol{P}_0$ is made large and not diagonal. The graphs show that the KF quickly corrects both of these "deliberate mistakes", and soon becomes practically indistinguishable from the earlier run that had a good initialisation.

More can be done with this simple system. For example, we could investigate what happens when we use incorrect values of $\boldsymbol{Q}$ and $\boldsymbol{R}$. In a real life analysis, we are likely to have a reasonably good understanding of the observations' error bars (i.e. of $\boldsymbol{R}$). But information on $\boldsymbol{Q}$ is generally scant. Then, $\boldsymbol{Q}$ can be thought of as a *tuning parameter*, which can (indeed, must) be adjusted to get certain output statistics right. This kind of tuning and filter divergence are closely related. Divergence is mentioned above. Tuning is mentioned by [Inv07], which is a technical but readable introduction to the KF.

Figure 7, starting on page 89, repeats figure 6; but now, the Kalman filter has been given $\boldsymbol{Q} = 0.1\boldsymbol{I}$, which is a factor of ten too small. This means that $\boldsymbol{P}$ is too small, and so is $\boldsymbol{K}$. The former means the estimated error bars are too small, as you can see on the graphs. The latter means that corrections at observation times are too small, i.e. too much weight is given to the prior estimate (from the model), and too little is given to the observations.

Figure 8, starting on page 91, makes the opposite deliberate mistake: now, the Kalman filter has been given $\boldsymbol{Q} = 5\boldsymbol{I}$, which is a factor of five too big. Corrections at observation times are now too big, i.e. too little weight is given to the prior estimate, and too much is given to the observations. You can see that the error bars are too big, and the estimated trajectory jumps directly to the observations, largely ignoring what the model says.

9 Variational Data Assimilation

9.1 Introduction

For linear systems (and models of them) and linear observation operators, the Kalman filter cannot be beaten. It is optimal under many assumptions, including those given in §7. It can also be extended to nonlinear systems, models and observation operators, albeit with the loss of optimality, as I discuss later in this paper. Nonlinearity is the rule in numerical weather prediction (NWP), climate prediction, and almost any Earth system. Any kind of Kalman filter is too expensive for operational NWP, or a GCM climate run, or any other large system. Indeed, for an n_x-dimensional state vector, [Dee91] points out that calculation of $\boldsymbol{P}$ is roughly $2n_x$ times more expensive than running the model forward to predict the prior state. So a cheaper method is used: 4-dimensional variational data assimilation. This has the additional advantage that it is directly applicable to nonlinear models of nonlinear systems with nonlinear observation operators. It has the disadvantage that it does not update the estimate's covariance matrix.

On nomenclature, "4-dimensional" refers to the three spatial dimensions of the atmosphere and the one dimension of time. Abbreviations include "4D-Var" and "Var".

If you look at the cost function below (equation (9.1)), you will notice two types of covariance matrix: a single background error covariance matrix $\boldsymbol{B}$, and a set $\{\boldsymbol{R}_k\}$, each member of which includes measurement error and error of representation. It is plain that an optimal analysis can only be produced if these covariance matrices are optimally estimated. Unfortunately, there is no known way to acquire an optimal estimate of $\boldsymbol{B}$; but sub-optimal methods are known, and are found throughout the literature. For now, it is useful to understand why the cost function takes this particular form in the first place.

9.2 The Cost Function

The usual 4D-Var *cost function*, or *penalty function*, is this scalar:

$$J(\boldsymbol{x}_0) = \frac{1}{2}(\boldsymbol{x}_0 - \boldsymbol{x}_b)^T \boldsymbol{B}^{-1}(\boldsymbol{x}_0 - \boldsymbol{x}_b) + \frac{1}{2}\sum_{k=1}^{N}(\boldsymbol{y}_k - \hat{H}_k(\boldsymbol{x}_k))^T \boldsymbol{R}_k^{-1}(\boldsymbol{y}_k - \hat{H}_k(\boldsymbol{x}_k)), \tag{9.1}$$

where $\boldsymbol{x}_k$ is an n_x-vector containing the model's state at time step k; $\boldsymbol{B}$ is the $n_x \times n_x$ error covariance matrix of $\boldsymbol{x}_b$, i.e. of a prior estimate of $\boldsymbol{x}$ at the beginning of the data assimilation time window; k labels the time step; $\boldsymbol{y}_k$ is an n_{y_k}-vector of observations of the system at time t_k; $\hat{H}_k(\cdot)$ is the operator that predicts $\boldsymbol{y}_k$ from $\boldsymbol{x}_k$; and $\boldsymbol{R}$ is the $n_{y_k} \times n_{y_k}$ error covariance matrix of $\boldsymbol{y}_k$. The job of Var is to find an initial state of the system, $\boldsymbol{x}_0$, that minimises J. That done, the model's trajectory through its phase space is optimal for that model, that set of observations, and that set of covariances. In particular, at the end of the run of N time steps, the model is in its optimal state, i.e. it gives the best estimate at the end of the run (and at the start, and through the run, too). Our best estimate of $\boldsymbol{x}_0$ is the analysis, $\hat{\boldsymbol{x}}_0$. (Since the model is deterministic, all the other model state vectors are analyses as well.)

The first term in the cost function penalises differences between $\boldsymbol{x}_0$ and $\boldsymbol{x}_b$. That is, if $\boldsymbol{x}_0$ is changed so that this measure of $\boldsymbol{x}_0 - \boldsymbol{x}_b$ increases, then J gets bigger, while we are trying to find an $\boldsymbol{x}_0$ that makes it as small as possible. The remaining terms, under the summation sign, penalise differences between actual and predicted observations. Since covariance matrices are positive semidefinite, each term in the equation is positive semidefinite. As a compromise between the observations and the model, this process *seems* reasonable, in an intuitive kind of way. Next, I show that this process really is, in a sense to be defined below, optimal.

In his Chapter 5, [Jaz70] derives a conditional mode filter for the case of discrete (in time) measurements of a discrete (in time) dynamical system. This system can be truly discrete, or a discrete representation (because finitely sampled) of a temporally continuous system. [Coh97a] also discusses this, and other, filters. Estimation of the state of the system in the past, i.e.

posterior analysis, is known as smoothing ([Jaz70], [BH97]). In the present case, we want to use the conditional mode filter to get an estimate, $\hat{\boldsymbol{x}}_0$. My §9.4 follows [Jaz70] closely, but omits one or two details that are not required in an introductory paper, as well as changing the notation appropriately.

9.3 Assumptions

The assumptions here are very much like those of §7.2. Terms such as true state and model error are defined in §4.

Let the state of the system at time step k be represented by $\boldsymbol{\xi}_k$. In work on the atmosphere, $\boldsymbol{\xi}_k$ could contain velocity components, humidity, density, temperature, tracer concentration, and so forth, at each point (i.e. at an infinite number of points — $\boldsymbol{\xi}_k$ contains fields, not a finite number of numbers). In the model, we have the finite n_x-vector $\boldsymbol{x}_k$, which is a finite discretisation of $\boldsymbol{\xi}_k$. The model evolves any model state vector like this:

$$\boldsymbol{x}_{k+1} = \hat{M}_k(\boldsymbol{x}_k), \tag{9.2}$$

where $\hat{M}_k(\cdot)$ is the model evolution operator (i.e. a complicated function that will generally be nonlinear). The true state is assumed to evolve in time as a Markov process, like this:

$$\boldsymbol{x}^t_{k+1} = \hat{M}_k(\boldsymbol{x}^t_k) + \boldsymbol{w}_k, \tag{9.3}$$

where $\hat{M}_k$ is *the same* function as in the previous equation; $\boldsymbol{w}_k$ is an n_x-vector white noise, Gaussian sequence $\boldsymbol{w}_k \sim N(\boldsymbol{0}, \boldsymbol{Q}_k)$. A true state comes from the previous true state, propagated by the model, then "contaminated" by model error. $\hat{M}_k$ is the same as f in equations (4.4).

Measurements of the system are also noisy:

$$\boldsymbol{y}_k = \hat{H}_k(\boldsymbol{x}^t_k) + \boldsymbol{v}_k, \tag{9.4}$$

where $\boldsymbol{y}_k$ is the n_{y_k}-vector of measurements at time step k; $\hat{H}_k(\cdot)$ is most generally a nonlinear function; and $\boldsymbol{v}_k$ is an n_{y_k}-vector Gaussian white noise sequence, $\boldsymbol{v}_k \sim N(\boldsymbol{0}, \boldsymbol{R}_k)$. It is assumed that $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ are independent of each other, as well as being white themselves. Observe that n_{y_k} has a k as part of its subscript, indicating that the number of observations may change as time goes on.

9.4 Derivation of the Cost Function

Assume now that we have a set of N observation vectors, $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_N\}$, each of these vectors being acquired at the time step indicated by its subscript. Write this set of vectors as a single symbol, and similarly the $\boldsymbol{x}^t$-vectors, thus:

$$\boldsymbol{X}^t_N = (\boldsymbol{x}^t_0, \boldsymbol{x}^t_1, \ldots, \boldsymbol{x}^t_N), \tag{9.5}$$

$$\boldsymbol{Y}_N = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_N). \tag{9.6}$$

These are ordered sets, but $\boldsymbol{Y}_N$ is not necessarily a matrix, because the $\boldsymbol{y}$-vectors are not necessarily all the same length. Observe that there are N observation vectors, and $(N+1)$ $\boldsymbol{x}^t$-vectors, because the latter set contains $\boldsymbol{x}^t_0$.

The conditional mode filtering problem is, then, to maximise the following conditional density with respect to the set of $\boldsymbol{x}^t$-vectors:

$$p(\boldsymbol{X}^t_N | \boldsymbol{Y}_N). \tag{9.7}$$

Using Bayes' rule, this density is written as

$$p(\boldsymbol{X}^t_N | \boldsymbol{Y}_N) = \frac{p(\boldsymbol{Y}_N | \boldsymbol{X}^t_N) p(\boldsymbol{X}^t_N)}{p(\boldsymbol{Y}_N)} \tag{9.8}$$

The denominator does not depend on $\boldsymbol{X}_N^t$, so we can ignore it in maximising $p(\boldsymbol{X}_N^t|\boldsymbol{Y}_N)$ with respect to $\boldsymbol{X}_N^t$.

Now, because the sequence $\{\boldsymbol{v}_k\}$ is white, the random variables $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\}$ are independent, once $\{\boldsymbol{x}_0^t, \ldots, \boldsymbol{x}_N^t\}$ are given. Thus:

$$p(\boldsymbol{Y}_N|\boldsymbol{X}_N^t) = \prod_{k=1}^{N} p_{v,k}(\boldsymbol{y}_k - \hat{H}_k(\boldsymbol{x}_k^t)), \tag{9.9}$$

where $p_{v,k}(\boldsymbol{y}_k - \hat{H}_k(\boldsymbol{x}_k^t)) = p_{v,k}(\boldsymbol{v}_k) \sim N[\boldsymbol{0}, \boldsymbol{R}_k]$, a Gaussian density.

Because the sequence $\{\boldsymbol{x}_0^t, \ldots, \boldsymbol{x}_N^t\}$ is a Markov sequence,

$$p(\boldsymbol{X}_N^t) = p(\boldsymbol{x}_0^t) \prod_{k=1}^{N} p(\boldsymbol{x}_k^t|\boldsymbol{x}_{k-1}^t) \tag{9.10}$$

$$= p(\boldsymbol{x}_0^t) \prod_{k=1}^{N} p_{w,k}(\boldsymbol{x}_k^t - \hat{M}_{k-1}(\boldsymbol{x}_{k-1}^t)), \tag{9.11}$$

where $p_{w,k}(\boldsymbol{x}_k^t - \hat{M}_{k-1}(\boldsymbol{x}_{k-1}^t)) = p_{w,k}(\boldsymbol{w}_{k-1}) \sim N[\boldsymbol{0}, \boldsymbol{Q}_{k-1}]$.

We assume that a prior estimate of $\boldsymbol{x}_0^t$ is available, known as the background state, and that it is multivariate normal (MVN):

$$\boldsymbol{x}_b^t \sim N(\langle \boldsymbol{x}_b^t \rangle, \boldsymbol{B}), \tag{9.12}$$

where $\boldsymbol{B}$ is the prior state error covariance matrix, also known as the background error covariance matrix. MVN random vectors are defined in equation (A.19) on page 62.

Putting these pieces together, remembering that the product of exponentials is the exponential of a sum, we obtain:

$$\begin{aligned} p(\boldsymbol{x}_0^t, \ldots, \boldsymbol{x}_N^t|\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N) \\ = \frac{c(\boldsymbol{B}, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_N)}{p(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N)} \exp\Bigg\{ -\frac{1}{2}(\boldsymbol{x}_0^t - \boldsymbol{x}_b^t)^T \boldsymbol{B}^{-1} (\boldsymbol{x}_0^t - \boldsymbol{x}_b^t) \\ -\frac{1}{2}\sum_{k=1}^{N} (\boldsymbol{y}_k - \hat{H}_k(\boldsymbol{x}_k^t))^T \boldsymbol{R}_k^{-1} (\boldsymbol{y}_k - \hat{H}_k(\boldsymbol{x}_k^t)) \\ -\frac{1}{2}\sum_{k=1}^{N} (\boldsymbol{x}_k^t - \hat{M}_{k-1}(\boldsymbol{x}_{k-1}^t))^T \boldsymbol{Q}_{k-1}^{-1} (\boldsymbol{x}_k^t - \hat{M}_{k-1}(\boldsymbol{x}_{k-1}^t)) \Bigg\}, \end{aligned} \tag{9.13}$$

where $c(\boldsymbol{B}, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_N)$ is a simplified notation for the parts of the density that are powers of 2π and determinants of covariance matrices, as in the quotient term in the third part of equation (A.19), not depending on any of the vectors $\{\boldsymbol{x}_k^t\}$. N.B. equation (9.13) is a Bayesian statement about the true states.

As stated close to equation (9.7), we need to maximise this conditional probability w.r.t. $\boldsymbol{X}_N^t$. This is equivalent to finding a set of $\boldsymbol{x}^t$-vectors that *minimise* this cost function:

$$\begin{aligned} J(\boldsymbol{x}_0^t) = \frac{1}{2}(\boldsymbol{x}_0^t - \boldsymbol{x}_b^t)^T \boldsymbol{B}^{-1} (\boldsymbol{x}_0^t - \boldsymbol{x}_b^t) \\ + \frac{1}{2}\sum_{k=1}^{N} (\boldsymbol{y}_k - \hat{H}_k(\boldsymbol{x}_k^t))^T \boldsymbol{R}_k^{-1} (\boldsymbol{y}_k - \hat{H}_k(\boldsymbol{x}_k^t)) \\ + \frac{1}{2}\sum_{k=1}^{N} (\boldsymbol{x}_k^t - \hat{M}_{k-1}(\boldsymbol{x}_{k-1}^t))^T \boldsymbol{Q}_{k-1}^{-1} (\boldsymbol{x}_k^t - \hat{M}_{k-1}(\boldsymbol{x}_{k-1}^t)) \end{aligned} \tag{9.14}$$

The three factors of 1/2 could be deleted, with no effect. But it is usual to keep them, so I have done the same. We have a tool for providing sets of $\boldsymbol{x}^t$-vectors: this is the model. However, the model is exactly deterministic[3], and has no $\boldsymbol{w}$ terms in it, so the third summation in equation (9.14) goes to zero. We are left with the best model equivalent to the system's cost function:

$$J(\boldsymbol{x}_0) = \frac{1}{2}(\boldsymbol{x}_0 - \boldsymbol{x}_b^t)^T \boldsymbol{B}^{-1}(\boldsymbol{x}_0 - \boldsymbol{x}_b^t) + \frac{1}{2}\sum_{k=1}^{N}(\boldsymbol{y}_k - \hat{H}_k(\boldsymbol{x}_k))^T \boldsymbol{R}_k^{-1}(\boldsymbol{y}_k - \hat{H}_k(\boldsymbol{x}_k)). \tag{9.15}$$

This process is sometimes known as making *the perfect model assumption*, but that is a misnomer. In fact, we are using the deterministic model because that is what we have to work with. If anything, we are making a *deterministic system assumption*.

The cost function we have ended up with in equation (9.15), is optimal under the assumptions that the $\{\boldsymbol{w}_k\}$ and $\{\boldsymbol{v}_k\}$ are white, Gaussian and uncorrelated in with each other; that the true state is governed by equation (9.3); and that measurements are described by equation (9.4). Our $\boldsymbol{x}_0$ is not likely to be a true value, i.e. not likely to be $\boldsymbol{x}_0^t$, even if we do our best. Then, equation (9.2) still holds, and the superscripted "t" are therefore omitted from the model's states in equation (9.15). Thus we have a further deviation from the "ideal" data assimilation system that is represented by equation (9.14).

The optimising value of $\boldsymbol{x}_0$ in equation (9.15) is an analysis, $\hat{\boldsymbol{x}}_0$, and it can be referred to as $\boldsymbol{x}_a$.

9.5 Remarks

1. On model error. The third summation in equation (9.14) is called the *model error term* — it penalises jumps in the evolution of the true state. That is, if we allow the model to make a jump at the end of every time step instead of proceeding deterministically, then those jumps are "discouraged" by the model error term. The cost function is then a compromise between trying to hit the background estimate, trying to hit the observations, and trying not to jump around too much.

   At least two factors mitigate against including the model error term in Var. First: we do not know what the $\boldsymbol{Q}_k$ are (although one could also point out that we do not know what $\boldsymbol{B}$ is). Second: there is considerable additional cost in calculating the sum of model error norms.

2. On statistics and probability. The derivation is Bayesian, but Bayes' rule applies to random variables as well as to Bayesian measures of uncertainty.

3. On approximations. As in [Dee91]'s critique of the Kalman filter on page 18, few, if any, of the assumptions that go into Var are truly met in reality.

4. It is sometimes said, e.g. by [BC], that the analysis error covariance matrix is equal to double the inverse of the Hessian matrix of the cost function, at the minimum of the cost function. Although this statement is true, it is only strictly true for linear systems.

5. It is not obvious, but it turns out that $\boldsymbol{B}$ controls the whole Var job. Consider the cost function again:

$$J(\boldsymbol{x}_0) = \frac{1}{2}(\boldsymbol{x}_0 - \boldsymbol{x}_b)^T \boldsymbol{B}^{-1}(\boldsymbol{x}_0 - \boldsymbol{x}_b) + \frac{1}{2}\sum_{k=1}^{N}(\text{Observation term})_k \tag{9.16}$$

[3] I omit discussion of stochastic physics models.

Taking the derivative w.r.t. $\boldsymbol{x}_0$, we get:

$$\boldsymbol{B}^{-1}(\boldsymbol{x}_0 - \boldsymbol{x}_b) = \sum_{k=1}^{N} \frac{\partial(\text{Observation term})_k}{\partial \boldsymbol{x}_0}. \tag{9.17}$$

Thus we can get an expression for the *analysis increment*, $(\boldsymbol{x}_0 - \boldsymbol{x}_b)$:

$$\boldsymbol{x}_0 - \boldsymbol{x}_b = \boldsymbol{B} \sum_{k=1}^{N} \frac{\partial(\text{Observation term})_k}{\partial \boldsymbol{x}_0}. \tag{9.18}$$

That is, the analysis increment is "proportional" to $\boldsymbol{B}$. More precisely, it is in the range of $\boldsymbol{B}$, and if $\boldsymbol{B}$ is not of full rank, then a certain subspace is not accessible by the Var job. Since $\boldsymbol{B}$ is unknown, various non-optimal methods are used to estimate it (see below). So, although it is obvious that we should try to get the covariance error matrices right, equation (9.18) shows in a concrete way that our methods for getting a $\boldsymbol{B}$ have a direct effect on the data assimilation, and hence on the forecast product.

6. Related to 4D-Var is 3D-Var, which does not take proper account of variations through time — see [Kal03]. 3D-FGAT ("First Guess at Appropriate Time") has intermediate complexity — see [Law10].

7. In Var, $\boldsymbol{B}$ is unknown and unknowable. But this does not mean we have zero knowledge about it. One way to think about it is to run a set of forecasts that are valid at the same time, but that have slightly different initial conditions — which is the essence of the "NMC Method". This set of forecasts contains differences that can be considered to be proxies for background error. See [Kal03], §5.5.3, "Final comments on the relative advantages of 3D-Var, PSAS and OI".

10 A Demonstration of Var

10.1 The Rotational System

Variational data assimilation is bound to fail with the rotational system that is discussed above. This is because the system itself is noisy in such a way that the particle roughly spirals out away from the origin. Var would choose initial values of x and y, using the deterministic part of the model to predict the true state through time. Since the deterministic part of the model just propagates the particle in a circle of fixed radius, Var would choose some sort of compromise that put the particle in a fixed orbit somewhere between the innermost and outermost segments of the stochastic orbit of the system.

10.2 DALEC

DALEC — "Data Assimilation Linked ECosystem" — is a simple carbon cycle model for forests. I took it, with permission, from the web site at [Fox]. Only the briefest of overviews is given here — look at the web site and at [WSL+05] if you need more details. Figure 9 on page 93 shows and explains the carbon pools and fluxes that are modelled within DALEC. The details are not very important for the present discussion, but if you want to see the simple code, you can find it in §C, starting on page 69. Running of the model is controlled by a number of parameters that are not described here, but are fixed. Also, meteorology data to drive the model must be provided. DALEC is also used in my ensemble Kalman filter demo at §13.2, starting on page 34.

For a demonstration, I did two very simple Var jobs on DALEC. The baseline run, that produced the "target function" of LAI vs. time, is explained later, in §13.2, and plotted in the

first part of Figure 14, on page 102. The cost function involved only the observation components, i.e. there was no background term. I decided to involve only LAI, so the cost function was:

$$J = \sum[(\text{forward-run LAI})_k - (\text{Var simulated LAI})_k]^2/R, \tag{10.1}$$

where the observation error covariance $R = 0.25$. That is, for a single Var job, the initial values of the five carbon pools were systematically adjusted to minimise this cost function. The minimisation was done by the R library function optim(), using the downhill simplex method [PTVF92]. That is, optim() found the five initial carbon pools that, when used to initialise the DALEC model, gave a good fit to the simulated observations. I repeated this job 100 times, each time using a different set of pseudo-random numbers to simulate the observation error, with zero mean and a standard deviation of $\sigma_{\text{LAI}} = 0.25$. Since the observation errors were different for each job, we would expect the optimal initial values to be a bit different each time. I made it easy by giving optim() the correct initial values. The first page of Figure 10, which starts on page 94, shows the results of this experiment. As you can see, there is some scatter in the results, but each time the Var job finds nearly the right result. Also, the mean result is very close to the correct set of numbers.

The second page of the Figure shows what happened when I made it harder, by initiating optim() with values that were 50% too high. Now we can see outliers, where optim() has found a local minimum of the cost function. Use of a background term — which was deliberately omitted in these simple examples — might have brought these outliers closer to the centroid. On the other hand, the whole cluster of points, and its centroid, would then have shifted.

11 An Introduction to Nonlinearity

11.1 Introduction

Var deals naturally with nonlinear systems and observation operators, but the plain KF can cope with neither. Therefore these sections, §11 and §12, explain how sequential filters can be adapted to nonlinear problems.

11.2 Nonlinear Observation Operators

So far, the discussion of the Kalman filter has assumed that the observation vector is a linear function of the state vector. That is, that it is just a matrix multiplication of the state vector. The case of a nonlinear observation operator is easy to deal with, albeit with a loss of exact optimality.

Equation (7.3) says:

$$\boldsymbol{y}_k = \boldsymbol{H}_k \boldsymbol{x}_k^t + \boldsymbol{v}_k, \tag{11.1}$$

leading to equations (7.15), which say:

$$\boldsymbol{e}_k = \hat{\boldsymbol{x}}_k - \boldsymbol{x}_k^t = \boldsymbol{S}_k \boldsymbol{e}_k^- + \boldsymbol{K}_k \boldsymbol{v}_k. \tag{11.2}$$

We now replace equation (7.3) with this nonlinear equivalent:

$$\boldsymbol{y}_k = \hat{H}_k(\boldsymbol{x}_k^t) + \boldsymbol{v}_k, \tag{11.3}$$

where $\hat{H}_k(\cdot)$ is a vector-valued function of its vector-valued argument. In the linear case, this is

simply $\hat{H}_k(\boldsymbol{x}_k^t) = \boldsymbol{H}_k \boldsymbol{x}_k^t$. Equations (7.15) and (11.2) then become:

$$\begin{aligned}
\boldsymbol{e}_k = \hat{\boldsymbol{x}}_k - \boldsymbol{x}_k^t &= \{\hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k[\boldsymbol{y}_k - \hat{H}_k(\hat{\boldsymbol{x}}_k^-)]\} - \boldsymbol{x}_k^t && (11.4a)\\
&= (\hat{\boldsymbol{x}}_k^- - \boldsymbol{x}_k^t) + \boldsymbol{K}_k[\hat{H}_k(\boldsymbol{x}_k^t) + \boldsymbol{v}_k - \hat{H}_k(\hat{\boldsymbol{x}}_k^-)] && (11.4b)\\
&= \boldsymbol{e}_k^- + \boldsymbol{K}_k[\boldsymbol{v}_k + \hat{H}_k(\boldsymbol{x}_k^t) - \hat{H}_k(\hat{\boldsymbol{x}}_k^-)] && (11.4c)\\
&= \boldsymbol{e}_k^- + \boldsymbol{K}_k[\boldsymbol{v}_k + \hat{H}_k(\hat{\boldsymbol{x}}_k^- - \boldsymbol{e}_k^-) - \hat{H}_k(\hat{\boldsymbol{x}}_k^-)] && (11.4d)\\
&= \boldsymbol{e}_k^- + \boldsymbol{K}_k\{\boldsymbol{v}_k + \hat{H}_k(\hat{\boldsymbol{x}}_k^-) - \boldsymbol{H}_k\boldsymbol{e}_k^- + O[(\boldsymbol{e}_k^-)^2] - \hat{H}_k(\hat{\boldsymbol{x}}_k^-)\} && (11.4e)\\
&= \boldsymbol{e}_k^- + \boldsymbol{K}_k\{\boldsymbol{v}_k - \boldsymbol{H}_k\boldsymbol{e}_k^- + O[(\boldsymbol{e}_k^-)^2]\} && (11.4f)\\
&\simeq (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\boldsymbol{e}_k^- + \boldsymbol{K}_k\boldsymbol{v}_k && (11.4g)\\
&= \boldsymbol{S}_k\boldsymbol{e}_k^- + \boldsymbol{K}_k\boldsymbol{v}_k. && (11.4h)
\end{aligned}$$

Here, $\boldsymbol{H}_k$ is the Jacobian of $\hat{H}_k$. If you do not understand why it arises here, read §D, starting on page 71. In the linear case, it is trivially equal to the matrix $\boldsymbol{H}_k$ that was used above, so I use the same notation in this nonlinear case. The rest of the derivation of the KF continues as before. Loss of optimality arises from discarding the $O[(\boldsymbol{e}_k^-)^2]$ term.

11.3 Nonlinear Systems and Models

There are many variations of the KF, that are designed to deal with nonlinear systems. Here, I discuss only the ensemble Kalman filter, or EnKF. First, I list some others. The linearised Kalman filter uses a fixed reference trajectory and linearises the KF equations for small perturbations about the reference trajectory [BH97]. The extended Kalman filter, or EKF, is similar, but instead of having a fixed reference trajectory, it updates it at observation times [BH97]. According to [RME02],

> For nonlinear dynamics, the extended Kalman filter (EKF) can be used, although it is notoriously unstable if the nonlinearities are strong ....

The unscented Kalman filter follows a rationally chosen set of model runs [JU97], and so is related to the EnKF. And there are many more.

12 The Ensemble Kalman Filter for Nonlinear Systems

12.1 Introduction

The abbreviation for the Ensemble KF is "EnKF". The EnKF works by running an ensemble of slightly different Kalman filters, replacing equations (7.21) and (7.23) by purely numerical calculations that use members of the ensemble, for example in equation (12.6). This allows nonlinear dynamics and nonlinear observation operators.

The word "ensemble" means more than "family" or "collection". It means a collection of vector or scalar random variables that are independent and identically distributed ("iid").

Geir Evensen invented the EnKF, as presented in [Eve94]. That paper does not make any sense to me (surely my fault, not Evensen's). Evensen's later paper, [Eve03], corrects the earlier formulation, but I cannot understand the theory or how it practically works, from that paper. Indeed, it is hard to find a description of how it actually works, and an understandable account of the theory. [SO08] is the best I have found. This section is based on that account, and adds some probability and statistics theory to back it up.

I use the overbar notation to indicate an ensemble average. For an ensemble of vectors $\boldsymbol{x}_\nu$, $\nu = 1, \ldots, N_e$, the ensemble average is given by:

$$\overline{\boldsymbol{x}} = \frac{1}{N_e}\sum_{j=1}^{N_e} \boldsymbol{x}_j. \qquad (12.1)$$

The ensemble covariance matrix of these vectors is

$$\overline{\boldsymbol{P}} = \frac{1}{N_e - 1} \sum_{j=1}^{N_e} [\boldsymbol{x}_j - \overline{\boldsymbol{x}}][\boldsymbol{x}_j - \overline{\boldsymbol{x}}]^T \tag{12.2}$$

We can also calculate expectations of ensemble averages and ensemble covariances, because these are, respectively, a random vector and a random matrix: $\langle \overline{\boldsymbol{x}} \rangle$ and $\langle \overline{\boldsymbol{P}} \rangle$.

We need to extend the notation, to accommodate time step and ensemble member in the specification of a vector. In the plain KF, a model state vector at time t_k was denoted $\boldsymbol{x}_k$. Now we have an ensemble of state vectors at time k, one of which we denote $\boldsymbol{x}_\nu(t_k)$, meaning ensemble member number ν at time k. Matrices just use a subscripted time step number, as in the plain KF. A high tilde, as in $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{P}}$, denotes the EnKF version of something that occurs in the plain KF.

12.2 The EnKF in the Linear Case

The propagation operator and the observation operator are now matrices, i.e. they represent linear transformations.

12.2.1 How it Works — the Recipe

Before considering nonlinearities, which are the *raison d'être* of the EnKF, I present the EnKF for the fully linear system and observations. I will say "$\forall \nu$" to mean "$1 \leq \nu \leq N_e$", where this is not clear from the context.

We start with our plain KF first guess, $\hat{\boldsymbol{x}}_0$, and construct an ensemble of N_e first guesses around it like this:

$$\tilde{\boldsymbol{x}}_\nu(t_0) = \hat{\boldsymbol{x}}_0 + \boldsymbol{\phi}_\nu, \quad \boldsymbol{\phi}_\nu \sim N(\boldsymbol{0}, \boldsymbol{P}_0), \quad \forall \nu, \tag{12.3}$$

where the $\boldsymbol{\phi}_\nu$ vectors contain Gaussian random numbers. Observe that these random vectors are specified like the background error, i.e. like the assumed error in $\hat{\boldsymbol{x}}_0$ in the plain KF.

We then proceed to advance the ensemble members in time, very much like the plain KF; but in the EnKF we add some extra random vectors that are distributed like the model error:

$$\tilde{\boldsymbol{x}}_\nu^-(t_1) = \boldsymbol{M}_0 \tilde{\boldsymbol{x}}_\nu(t_0) + \boldsymbol{\zeta}_\nu(t_0), \quad \boldsymbol{\zeta}_\nu(t_0) \sim N(\boldsymbol{0}, \boldsymbol{Q}_0), \quad \forall \nu. \tag{12.4}$$

Then we can form the prior estimate covariance matrix like this:

$$\overline{\boldsymbol{x}^-}(t_1) = \frac{1}{N_e} \sum_{j=1}^{N_e} \tilde{\boldsymbol{x}}_j^-(t_1), \tag{12.5}$$

$$\tilde{\boldsymbol{P}}_1^- = \frac{1}{N_e - 1} \sum_{j=1}^{N_e} [\tilde{\boldsymbol{x}}_j^-(t_1) - \overline{\boldsymbol{x}^-}(t_1)][\tilde{\boldsymbol{x}}_j^-(t_1) - \overline{\boldsymbol{x}^-}(t_1)]^T. \tag{12.6}$$

Next, we make a Kalman gain matrix, using $\tilde{\boldsymbol{P}}_1^-$ instead of $\boldsymbol{P}_1^-$:

$$\tilde{\boldsymbol{K}}_1 = \tilde{\boldsymbol{P}}_1^- \boldsymbol{H}_1^T (\boldsymbol{H}_1 \tilde{\boldsymbol{P}}_1^- \boldsymbol{H}_1^T + \boldsymbol{R}_1)^{-1}. \tag{12.7}$$

We calculate the posterior estimate as in the plain KF; but with the addition of extra random vectors that are distributed as the observation errors. If this is not done, the ensemble's evolving variance will be too low [BvE98], and can even collapse. Thus, the augmented observation increment is:

$$\tilde{\boldsymbol{d}}_\nu(t_1) = \boldsymbol{y}(t_1) - \boldsymbol{H}_1 \tilde{\boldsymbol{x}}_\nu^-(t_1) + \boldsymbol{\eta}_\nu(t_1), \quad \boldsymbol{\eta}_\nu(t_1) \sim N(\boldsymbol{0}, \boldsymbol{R}_1), \quad \forall \nu, \tag{12.8}$$

which should be compared with equations (7.10).

Then we get the ensemble of posterior estimates:

$$\tilde{\boldsymbol{x}}_\nu(t_1) = \tilde{\boldsymbol{x}}_\nu^-(t_1) + \tilde{\boldsymbol{K}}_1 \tilde{\boldsymbol{d}}_\nu(t_1), \quad \forall \nu, \tag{12.9}$$

and finally,

$$\tilde{\boldsymbol{P}}_1 = \tilde{\boldsymbol{S}}_1 \tilde{\boldsymbol{P}}_1^- \tilde{\boldsymbol{S}}_1^T + \tilde{\boldsymbol{K}}_1 \boldsymbol{R}_1 \tilde{\boldsymbol{K}}_1^T, \tag{12.10}$$

or the ensemble equivalents of (7.34)–(7.36).

For later time steps, we just repeat this predict-correct cycle, working on each member of the ensemble. At any time k, the best posterior estimate of the system is $\overline{\tilde{\boldsymbol{x}}}(t_k)$, the sample mean of the ensemble of posterior estimates. Its covariance is given by $\tilde{\boldsymbol{P}}_k$.

In summary:

---

1. Start with $\tilde{\boldsymbol{x}}_\nu(t_0) = \hat{\boldsymbol{x}}_0 + \boldsymbol{\phi}_\nu$, where $\boldsymbol{\phi}_\nu \sim N(\boldsymbol{0}, \boldsymbol{P}_0), \quad 1 \le \nu \le N_e$
2. do $k = 1 \to n$
3. $\quad \forall \nu$ calculate $\tilde{\boldsymbol{x}}_\nu^-(t_k) = \boldsymbol{M}_{k-1} \tilde{\boldsymbol{x}}_\nu(t_{k-1}) + \boldsymbol{\zeta}_\nu(t_{k-1})$, where $\boldsymbol{\zeta}_\nu(t_{k-1}) \sim N(\boldsymbol{0}, \boldsymbol{Q}_{k-1})$
4. $\quad$ calculate $\overline{\boldsymbol{x}^-}(t_k) = \frac{1}{N_e} \sum_{j=1}^{N_e} \tilde{\boldsymbol{x}}_j^-(t_k)$
5. $\quad$ calculate $\tilde{\boldsymbol{P}}_k^- = \frac{1}{N_e - 1} \sum_{j=1}^{N_e} [\tilde{\boldsymbol{x}}_j^-(t_1) - \overline{\boldsymbol{x}^-}(t_1)][\tilde{\boldsymbol{x}}_j^-(t_1) - \overline{\boldsymbol{x}^-}(t_1)]^T$
6. $\quad$ calculate $\tilde{\boldsymbol{K}}_k = \tilde{\boldsymbol{P}}_k^- \boldsymbol{H}_k^T (\boldsymbol{H}_k \tilde{\boldsymbol{P}}_k^- \boldsymbol{H}_k^T + \boldsymbol{R}_k)^{-1}$
7. $\quad \forall \nu$ calculate $\tilde{\boldsymbol{d}}_\nu(t_k) = \boldsymbol{y}(t_k) - \boldsymbol{H}_k \tilde{\boldsymbol{x}}_\nu^-(t_k) + \boldsymbol{\eta}_\nu(t_k)$, where $\boldsymbol{\eta}_\nu(t_k) \sim N(\boldsymbol{0}, \boldsymbol{R}_k)$
8. $\quad \forall \nu$ calculate $\tilde{\boldsymbol{x}}_\nu(t_k) = \tilde{\boldsymbol{x}}_\nu^-(t_k) + \tilde{\boldsymbol{K}}_k \tilde{\boldsymbol{d}}_\nu(t_k)$
9. $\quad$ calculate $\tilde{\boldsymbol{P}}_k = \tilde{\boldsymbol{S}}_k \tilde{\boldsymbol{P}}_k^- \tilde{\boldsymbol{S}}_k^T + \tilde{\boldsymbol{K}}_k \boldsymbol{R}_k \tilde{\boldsymbol{K}}_k^T$ ... and also see equations (7.34)–(7.36)
10. $\quad$ calculate $\hat{\boldsymbol{x}}(t_k) = \overline{\tilde{\boldsymbol{x}}}(t_k) = \frac{1}{N_e} \sum_{j=1}^{N_e} \tilde{\boldsymbol{x}}_j(t_k)$
11. end do

N.B. you might not want $\hat{\boldsymbol{x}}(t_k)$ at each time step. In that case, you do not need to calculate it until you want to print it.

12.2.2 Another Recipe

The following changed notation can be easier to think about, and easier to code. It is not obvious that it does exactly the same thing, so I discuss it here. It involves storing all the ensemble members at a given time in a single matrix. Thus, instead of having N_e ensemble members at time k, we have a single $n_x \times N_e$ matrix called $\tilde{\boldsymbol{X}}_k$, which contains the column vectors $\tilde{\boldsymbol{x}}_\nu(t_k)$ side-by-side, like this:

$$\tilde{\boldsymbol{X}}_k = [\tilde{\boldsymbol{x}}_1(t_k) \,|\, \tilde{\boldsymbol{x}}_2(t_k) \,|\, \cdots \,|\, \tilde{\boldsymbol{x}}_{N_e}(t_k)]\,. \tag{12.11}$$

Similarly,

$$\tilde{\boldsymbol{X}}_k^- = \left[\tilde{\boldsymbol{x}}_1^-(t_k) \,|\, \tilde{\boldsymbol{x}}_2^-(t_k) \,|\, \cdots \,|\, \tilde{\boldsymbol{x}}_{N_e}^-(t_k)\right], \tag{12.12}$$

and

$$\overline{\boldsymbol{X}_k^-} = \left[\overline{\boldsymbol{x}^-}(t_k) \,|\, \overline{\boldsymbol{x}^-}(t_k) \,|\, \cdots \,|\, \overline{\boldsymbol{x}^-}(t_k)\right]. \tag{12.13}$$

Note that all the columns of $\tilde{\boldsymbol{X}}_k$ differ from one another, and the same for $\tilde{\boldsymbol{X}}_k^-$; but all the columns of $\overline{\boldsymbol{X}_k^-}$ are the same as each other.

Then it can be shown easily (if you know what to expect) that

$$\tilde{\boldsymbol{X}}_k^- = \boldsymbol{M}_{k-1} \tilde{\boldsymbol{X}}_{k-1} + \tilde{\boldsymbol{Z}}_{k-1}, \tag{12.14}$$

where $\tilde{\boldsymbol{Z}}_{k-1}$ is a matrix populated with N_e random vectors $\boldsymbol{\zeta}_\nu \sim N(\boldsymbol{0}, \boldsymbol{Q}_{k-1}), \forall \nu$. And equation (12.6) can be replaced by this:

$$\tilde{\boldsymbol{P}}_k^- = \frac{1}{N_e - 1}[\boldsymbol{X}_k^- - \overline{\boldsymbol{X}_k^-}][\boldsymbol{X}_k^- - \overline{\boldsymbol{X}_k^-}]^T \tag{12.15}$$

Using matrices, we can replace the N_e equations (12.8) and the N_e equations (12.9) with the following:

$$\tilde{\boldsymbol{Y}}_k = [\boldsymbol{y}(t_k) \,|\, \cdots \,|\, \boldsymbol{y}(t_k)]\,, \tag{12.16a}$$

$$\tilde{\boldsymbol{E}}_k = [\boldsymbol{\eta}_1(t_k) \,|\, \cdots \,|\, \boldsymbol{\eta}_{N_e}(t_k)], \quad \boldsymbol{\eta}_\nu \sim N(\boldsymbol{0}, \boldsymbol{R}_k), \tag{12.16b}$$

$$\tilde{\boldsymbol{D}}_k = \tilde{\boldsymbol{Y}}_k - \boldsymbol{H}_k \tilde{\boldsymbol{X}}_k^- + \tilde{\boldsymbol{E}}_k, \tag{12.16c}$$

$$\tilde{\boldsymbol{X}}_k = \tilde{\boldsymbol{X}}_k^- + \tilde{\boldsymbol{K}}_k \boldsymbol{D}_k. \tag{12.16d}$$

In summary:

---

1. Start with $\tilde{\boldsymbol{x}}_\nu(t_0) = \hat{\boldsymbol{x}}_0 + \boldsymbol{\phi}_\nu$, where $\boldsymbol{\phi}_\nu \sim N(\boldsymbol{0}, \boldsymbol{P}_0)$
2. Arrange the vectors $\tilde{\boldsymbol{x}}_\nu(t_0)$ into a matrix $\tilde{\boldsymbol{X}}_0$
3. do $k = 1 \to n$
4. calculate $\tilde{\boldsymbol{Z}}_{k-1} = [\boldsymbol{\zeta}_1(t_{k-1}) \,|\, \cdots \,|\boldsymbol{\zeta}_{N_e}(t_{k-1})]$, where $\boldsymbol{\zeta}_\nu(t_{k-1}) \sim N(\boldsymbol{0}, \boldsymbol{Q}_{k-1})$
5. calculate $\tilde{\boldsymbol{X}}_k^- = \boldsymbol{M}_{k-1}\tilde{\boldsymbol{X}}_{k-1} + \tilde{\boldsymbol{Z}}_{k-1}$
6. calculate $\overline{\boldsymbol{X}_k^-} = \left[\overline{\boldsymbol{x}^-}(t_k) \,|\, \ldots \,|\, \overline{\boldsymbol{x}^-}(t_k)\right]$
7. calculate $\tilde{\boldsymbol{P}}_k^- = \frac{1}{N_e - 1}\left[\hat{\boldsymbol{X}}_k^- - \overline{\boldsymbol{X}_k^-}\right]\left[\hat{\boldsymbol{X}}_k^- - \overline{\boldsymbol{X}_k^-}\right]^T$
8. calculate $\tilde{\boldsymbol{K}}_k = \tilde{\boldsymbol{P}}_k^- \boldsymbol{H}_k^T (\boldsymbol{H}_k \tilde{\boldsymbol{P}}_k^- \boldsymbol{H}_k^T + \boldsymbol{R}_k)^{-1}$
9. calculate $\tilde{\boldsymbol{E}}_k = [\boldsymbol{\eta}_1(t_k) \,|\, \ldots \,|\, \boldsymbol{\eta}_{N_e}(t_k)]$, where $\boldsymbol{\eta}_\nu(t_k) \sim N(\boldsymbol{0}, \boldsymbol{R}_k)$
10. calculate $\tilde{\boldsymbol{D}}_k = \tilde{\boldsymbol{Y}}_k - \boldsymbol{H}_k \tilde{\boldsymbol{X}}_k^- + \tilde{\boldsymbol{E}}_k$, where $\tilde{\boldsymbol{Y}}_k = [\boldsymbol{y}(t_k) \,|\, \ldots \,|\, \boldsymbol{y}(t_k)]$
11. calculate $\tilde{\boldsymbol{X}}_k = \tilde{\boldsymbol{X}}_k^- + \tilde{\boldsymbol{K}}_k \tilde{\boldsymbol{D}}_k$
12. calculate $\tilde{\boldsymbol{P}}_k = \tilde{\boldsymbol{S}}_k \tilde{\boldsymbol{P}}_k^- \tilde{\boldsymbol{S}}_k^T + \tilde{\boldsymbol{K}}_k \boldsymbol{R}_k \tilde{\boldsymbol{K}}_k^T$ ... and also see equations (7.34)–(7.36)
13. calculate $\hat{\boldsymbol{x}}(t_k) = \overline{\tilde{\boldsymbol{x}}}(t_k) = \frac{1}{N_e}\sum_j \tilde{\boldsymbol{x}}_j(t_k)$, where $\tilde{\boldsymbol{x}}_\nu(t_k)$ are the columns of $\tilde{\boldsymbol{X}}_k$
14. end do

N.B. you might not want $\hat{\boldsymbol{x}}(t_k)$ at each time step. In that case, you do not need to calculate it until you want to print it.

12.2.3 Does it Work?

This all looks quite plausible, but it is important to confirm, if possible, that it is consistent with the plain KF. We can investigate this by examining expectations and covariances of the results. Direct comparison is possible because the above recipe for the EnKF assumes operators are linear.

We first examine $\hat{\boldsymbol{x}}_\nu(t_0)$:

$$\langle \hat{\boldsymbol{x}}_\nu(t_0) \rangle = \langle \hat{\boldsymbol{x}}_0 + \boldsymbol{\phi}_\nu \rangle = \hat{\boldsymbol{x}}_0 + \langle \boldsymbol{\phi}_\nu \rangle = \hat{\boldsymbol{x}}_0, \tag{12.17a}$$

$$\text{so} \quad \langle \hat{\boldsymbol{X}}_0 \rangle = \left[\hat{\boldsymbol{x}}_0 \,|\, \hat{\boldsymbol{x}}_0 \,|\, \ldots \,|\, \hat{\boldsymbol{x}}_0\right], \tag{12.17b}$$

as required. And:

$$\mathrm{cov}(\hat{\boldsymbol{x}}_\nu(t_0)) = \langle [\hat{\boldsymbol{x}}_\nu(t_0) - \langle \hat{\boldsymbol{x}}_\nu(t_0) \rangle][\hat{\boldsymbol{x}}_\nu(t_0) - \langle \hat{\boldsymbol{x}}_\nu(t_0) \rangle]^T \rangle = \langle \boldsymbol{\phi}_\nu(t_0) \boldsymbol{\phi}_\nu(t_0)^T \rangle = \langle \boldsymbol{P}_0 \rangle, \tag{12.18}$$

as required.

The following expectation works out easily:

$$\langle \tilde{\boldsymbol{x}}_\nu^-(t_1)\rangle = \langle \boldsymbol{M}_0\tilde{\boldsymbol{x}}_\nu(t_0) + \boldsymbol{\zeta}_\nu(t_0)\rangle = \boldsymbol{M}_0\langle \tilde{\boldsymbol{x}}_0^t + \boldsymbol{\phi}_\nu(t_0)\rangle = \boldsymbol{M}_0\hat{\boldsymbol{x}}_0^t, \tag{12.19}$$

as required. The covariance of a single ensemble member is a longer manipulation, but not difficult:

$$\begin{aligned}
\mathrm{cov}(\tilde{\boldsymbol{x}}_\nu^-(t_1)) &= \langle [\tilde{\boldsymbol{x}}_\nu^-(t_1) - \langle \tilde{\boldsymbol{x}}_\nu^-(t_1)\rangle][\tilde{\boldsymbol{x}}_\nu^-(t_1) - \langle \tilde{\boldsymbol{x}}_\nu^-(t_1)\rangle]^T\rangle && (12.20a)\\
&= \langle [\tilde{\boldsymbol{x}}_\nu^-(t_1) - \boldsymbol{M}_0\hat{\boldsymbol{x}}_0^t][\tilde{\boldsymbol{x}}_\nu^-(t_1) - \boldsymbol{M}_0\hat{\boldsymbol{x}}_0^t]^T\rangle && (12.20b)\\
&= \langle [\boldsymbol{M}_0(\hat{\boldsymbol{x}}_0 + \boldsymbol{\phi}_\nu(t_0)) + \boldsymbol{\zeta}_\nu(t_0) - \boldsymbol{M}_0\hat{\boldsymbol{x}}_0][\boldsymbol{M}_0(\hat{\boldsymbol{x}}_0 + \boldsymbol{\phi}_\nu(t_0)) + \boldsymbol{\zeta}_\nu(t_0) - \boldsymbol{M}_0\hat{\boldsymbol{x}}_0]^T\rangle && (12.20c)\\
&= \langle [\boldsymbol{M}_0\boldsymbol{\phi}_\nu(t_0) + \boldsymbol{\zeta}_\nu(t_0)][\boldsymbol{M}_0\boldsymbol{\phi}_\nu(t_0) + \boldsymbol{\zeta}_\nu(t_0)]^T\rangle && (12.20d)\\
&= \langle \boldsymbol{M}_0\boldsymbol{\phi}_\nu(t_0)\boldsymbol{\phi}_\nu(t_0)^T\boldsymbol{M}_0^T + \boldsymbol{M}_0\boldsymbol{\phi}_\nu(t_0)\boldsymbol{\zeta}_\nu(t_0)^T + \boldsymbol{\zeta}_\nu(t_0)\boldsymbol{\phi}_\nu(t_0)^T\boldsymbol{M}_0^T + \boldsymbol{\zeta}_\nu(t_0)\boldsymbol{\zeta}_\nu(t_0)^T\rangle && (12.20e)\\
&= \boldsymbol{M}_0\boldsymbol{P}_0\boldsymbol{M}_0^T + \boldsymbol{Q}_0, && (12.20f)
\end{aligned}$$

as required, since this is the same as for the plain KF, as in equations 7.21.

Now for the analysis of $\tilde{\boldsymbol{P}}_1^-$, i.e. the ensemble covariance matrix.

$$\begin{aligned}
\langle \tilde{\boldsymbol{P}}_1^-\rangle &= \frac{1}{N_e-1}\sum_{j=1}^{N_e}\left\langle \left[\hat{\boldsymbol{x}}_j^-(t_1) - \overline{\boldsymbol{x}^-(t_1)}\right]\left[\hat{\boldsymbol{x}}_j^-(t_1) - \overline{\boldsymbol{x}^-(t_1)}\right]^T\right\rangle && (12.21a)\\
&= \frac{1}{N_e-1}\sum_{j=1}^{N_e}\left\langle \left\{\boldsymbol{M}_0(\hat{\boldsymbol{x}}_0 + \boldsymbol{\phi}_j(t_0)) + \boldsymbol{\zeta}_j(t_0) - \frac{1}{N_e}\sum_{k=1}^{N_e}[\boldsymbol{M}_0(\hat{\boldsymbol{x}}_0 + \boldsymbol{\phi}_k(t_0)) + \boldsymbol{\zeta}_k(t_0)]\right\}\times\right.\\
&\qquad \left.\left\{\boldsymbol{M}_0(\hat{\boldsymbol{x}}_0 + \boldsymbol{\phi}_j(t_0)) + \boldsymbol{\zeta}_j(t_0) - \frac{1}{N_e}\sum_{m=1}^{N_e}[\boldsymbol{M}_0(\hat{\boldsymbol{x}}_0 + \boldsymbol{\phi}_m(t_0)) + \boldsymbol{\zeta}_m(t_0)]\right\}^T\right\rangle && (12.21b)\\
&= \frac{1}{N_e-1}\sum_{j=1}^{N_e}\left\langle \boldsymbol{M}_0\boldsymbol{\phi}_j(t_0)\boldsymbol{\phi}_j(t_0)^T\boldsymbol{M}_0^T + \boldsymbol{M}_0\boldsymbol{\phi}_j(t_0)\boldsymbol{\zeta}_j^T(t_0) - \right.\\
&\qquad \frac{1}{N_e}\boldsymbol{M}_0\boldsymbol{\phi}_j(t_0)\sum_{m=1}^{N_e}[\boldsymbol{M}_0\boldsymbol{\phi}_m(t_0) + \boldsymbol{\zeta}_m(t_0)]^T + \boldsymbol{\zeta}_j(t_0)\boldsymbol{\phi}_j(t_0)^T\boldsymbol{M}_0^T +\\
&\qquad \boldsymbol{\zeta}_j(t_0)\boldsymbol{\zeta}_j^T(t_0) - \frac{1}{N_e}\boldsymbol{\zeta}_j(t_0)\sum_{m=1}^{N_e}[\boldsymbol{M}_0\boldsymbol{\phi}_m(t_0) + \boldsymbol{\zeta}_m(t_0)]^T -\\
&\qquad \frac{1}{N_e}\sum_{k=1}^{N_e}[\boldsymbol{M}_0\boldsymbol{\phi}_k(t_0) + \boldsymbol{\zeta}_k(t_0)]\boldsymbol{\phi}_j(t_0)^T\boldsymbol{M}_0^T - \frac{1}{N_e}\sum_{k=1}^{N_e}[\boldsymbol{M}_0\boldsymbol{\phi}_k(t_0) + \boldsymbol{\zeta}_k(t_0)]\boldsymbol{\zeta}_j^T(t_0) +\\
&\qquad \left.\frac{1}{N_e^2}\sum_{k=1}^{N_e}[\boldsymbol{M}_0\boldsymbol{\phi}_k(t_0) + \boldsymbol{\zeta}_k(t_0)]\sum_{m=1}^{N_e}[\boldsymbol{M}_0\boldsymbol{\phi}_m(t_0) + \boldsymbol{\zeta}_m(t_0)]^T\right\rangle && (12.21c)\\
&= \frac{1}{N_e-1}\sum_{j=1}^{N_e}\left[\boldsymbol{M}_0\boldsymbol{P}_0\boldsymbol{M}_0^T + 0 - \frac{1}{N_e}\boldsymbol{M}_0\boldsymbol{P}_0\boldsymbol{M}_0^T - 0 + 0 + \boldsymbol{Q}_0 - 0 - \frac{1}{N_e}\boldsymbol{Q}_0 - \right.\\
&\qquad \left.\frac{1}{N_e}\boldsymbol{M}_0\boldsymbol{P}_0\boldsymbol{M}_0^T - 0 - 0 - \frac{1}{N_e}\boldsymbol{Q}_0 + \frac{1}{N_e}(\boldsymbol{M}_0\boldsymbol{P}_0\boldsymbol{M}_0^T + 0 + 0 + \boldsymbol{Q}_0)\right] && (12.21d)\\
&= \frac{1}{N_e-1}\sum_{j=1}^{N_e}\left(1 - \frac{1}{N_e}\right)(\boldsymbol{M}_0\boldsymbol{P}_0\boldsymbol{M}_0^T + \boldsymbol{Q}_0) && (12.21e)
\end{aligned}$$

$$= \frac{N_e - 1}{N_e(N_e - 1)} \sum_{j=1}^{N_e} (\boldsymbol{M}_0 \boldsymbol{P}_0 \boldsymbol{M}_0^T + \boldsymbol{Q}_0) \tag{12.21f}$$

$$= \boldsymbol{M}_0 \boldsymbol{P}_0 \boldsymbol{M}_0^T + \boldsymbol{Q}_0 = \boldsymbol{P}_1^-, \tag{12.21g}$$

as required. N.B. equations (12.21) are just an extended version of the proof that the expectation of a scalar sample variance is equal to the population variance [Wik07b]. To get from (12.21b) to (12.21c): first, many of the terms were multiplied out; then cancellations done (e.g. the terms in $\hat{\boldsymbol{x}}_0 \hat{\boldsymbol{x}}_0^T$ all drop out); and expectations of zero-mean random variables were removed.

Having proved that $\langle \tilde{\boldsymbol{P}}_1^- \rangle = \boldsymbol{P}_1^-$, the next steps are to show similar correspondences hold for $\tilde{\boldsymbol{K}}_1$, $\tilde{\boldsymbol{x}}_\nu(t_1)$ and $\tilde{\boldsymbol{P}}_1$. That is, that the expectations of these quantities are the same as in the plain Kalman filter, and that the covariances of $\tilde{\boldsymbol{x}}_\nu(t_1)$ are also the same as in the plain KF; and thus that the same is true for later time steps. Unfortunately, I cannot figure out how to continue this process. I would be grateful if any reader could tell me how to do it.

12.3 The EnKF in the Nonlinear Case

The previous section goes some way to showing that the EnKF is equivalent in expectation and covariance to the plain KF in the linear case. Equivalence for the rotational system is demonstrated in §13.1. When propagation and observation operators, or both, are nonlinear, then a direct comparison cannot be made. Nevertheless, a demonstration is appropriate, and is given below in §13.2, starting on page 34.

To make it work in the nonlinear case, the state propagation matrix is replaced by a function in equation (12.4). Similarly, the matrix observation operator in equation (12.8) is replaced with a function. Elsewhere, $\boldsymbol{H}$ is replaced by the Jacobian of $\hat{H}$, as explained in §11.2.

12.4 Remarks

1. An EnKF analysis scheme is summarised in the pseudocodes on pages 30 and 31. Before making this work practically, I had tried a slightly different scheme, in which $\tilde{\boldsymbol{X}}_k^- = \boldsymbol{M}_{k-1} \tilde{\boldsymbol{X}}_{k-1}$ and $\tilde{\boldsymbol{P}}_k^- = \frac{1}{N_e - 1} \left[\hat{\boldsymbol{X}}_k^- - \overline{\boldsymbol{X}_k^-} \right] \left[\hat{\boldsymbol{X}}_k^- - \overline{\boldsymbol{X}_k^-} \right]^T + \boldsymbol{Q}_{k-1}$. So the ensemble members evolve deterministically between Kalman updates (albeit that they are *initialised* stochastically), and $\tilde{\boldsymbol{P}}_k^-$ has a random matrix added to it, that is distributed as the model error. Just like the correct scheme, this one has $\langle \tilde{\boldsymbol{P}}_1 \rangle = \boldsymbol{P}_1$. In a practical implementation, though, this is not a very good filter. It tracks the system less well than the plain KF and the correct EnKF, and the covariances are too small. The filter goes off the rails after the first iteration, but the reason will remain mysterious until I figure out how to calculate all the expectations in the EnKF (and in its incorrect variation).

2. It seems likely that equation (12.10) can be replaced by a sample covariance matrix of the ensemble of posterior estimates. But I have not yet proved or otherwise demonstrated this, even to myself.

3. In the plain KF, $\boldsymbol{P}^-$ and $\boldsymbol{P}$ did not depend on the state vectors, and could be calculated offline. This is not the case in the EnKF. Now, the matrices must be calculated from the state vectors.

13 Two Demonstrations of the EnKF

13.1 The Rotational System

Because the rotational system is linear, the EnKF should exactly reproduce the analysis that the plain KF does, in the limit $N_e \to \infty$. This section, §13.1, examines finite approximations to

this claim for three sizes of ensemble.

First, $N_e = 25$ seemed like a reasonable number. I ran the EnKF exactly as in §12.2, with an ensemble size $N_e = 25$. The parameters and initiation were exactly as in Figure 6, starting on page 87. The results are shown in Figure 11, starting on page 96. You can see that the trajectory is close to that shown in Figure 6, but the error bars are not so good. The error bar envelopes in the top three panels of the second page of the figure are not as smooth as in the plain KF. This reflects the wobbliness of $P(x, x)$ and $P(y, y)$. The covariance, $P(x, y)$ is very different from the plain KF result, which was practically zero after the transient had disappeared.

Repeating the same numerical experiment, but with $N_e = 250$ gave Figure 12, starting on page 98. The trajectory is now indistinguishable from that in Figure 6. The error bars have smooth envelopes, and the variances are starting to look like those of the plan KF. The covariance is generally smaller, roughly by a factor of $\sqrt{10}$ by the look of it, although I have not quantified this.

Is $N_e = 5$ a big enough ensemble? Figure 13 shows the results, starting on page 100. The error bars, variances and covariance are now all over the place. The trajectory starts off quite different to the plain KF's, but later starts to resemble a slightly rougher version of it.

A useful conclusion would be a statement on how big an ensemble is required. In fact, it appears that the required size of the ensemble must depend on the user's requirements. Is a rough, post-transient trajectory a good enough result? Or do you need smooth error bars? One thing we can say is that a big enough EnKF ensemble can reproduce the plain KF in a linear case where a direct comparison can be made.

13.2 DALEC

Next we apply the EnKF to the DALEC model that was introduced in §10.2. DALEC is non-linear, so the plain KF is not applicable. Nevertheless, it is useful to see what the EnKF can do here.

For a certain set of parameters (it does not really matter what they are, for the purposes of this tutorial) and initial values of the carbon pools, a forward run of the DALEC model is shown in page 1 of Figure 14. The carbon pools evolve over three years as in the top five panels. LAI is proportional to C_{f}, and the line in the bottom panel shows this. The coloured circles show simulated observations of LAI, calculated by adding zero-mean Gaussian random noise with $\sigma = 0.25$, to the forward run values. A sparse subset is assumed, corresponding to a notional sampling by a satellite-borne instrument looking through an atmosphere that is sometimes cloudy.

I ran an ensemble KF with 100 members in three different ways. The variables involved were the five carbon pools, and the observations were those simulated in the forward run. That is, $\boldsymbol{x} = (C_{\mathrm{f}}, C_{\mathrm{r}}, C_{\mathrm{w}}, C_{\mathrm{lit}}, C_{\mathrm{som}})$, and $\boldsymbol{y} = \mathrm{LAI}$.

In the first run, shown on page 2 of Figure 14, which starts on page 102, the EnKF was started with the correct initial values of the carbon pools. The model error was set as $\boldsymbol{Q} = \mathrm{diag}(2.75, 1.8, 11, 0.5, 99)^2$. All the carbon pools and the LAI are tracked well. But some problems are immediately apparent. The error bars on C_{w} and C_{som} show no sign of being constrained by the observations. C_{f} and C_{lit}, on the other hand, have pinch points in their error bars when observations are available. C_{r} seems to be in an intermediate state.

In the second run, shown on page 3 of Figure 14, I repeated the first, with the exception that I doubled the initial values of the carbon pools that were supplied to the EnKF. Now we can see clearly that C_{f} and C_{lit} are well controlled by the filter. Previously, we were not sure of the status of C_{r}, but now we can see that it is controlled by the filter, albeit not strongly. Again, C_{w} and C_{som} appear not to be controlled at all.

Finally, the third run, shown on page 4 of Figure 14, repeats the third, but decreases the model error variance of C_{som} to 22^2. The only change is that C_{som} is now even more poorly controlled.

We can conclude that the KF does not work fully for this system with these observations. This is a problem of *observability*. As [BH97] say,

> There is a third kind of divergence problem that may occur when the system is not observable. Physically, this means that there are one or more state variables (or linear combinations thereof) that are hidden from the view of the observer (i.e. the measurements). As a result, if the unobserved processes are unstable, the corresponding estimation errors will be similarly unstable. This problem has nothing to do with roundoff error or inaccurate system modelling. It is just a simple fact of life that sometimes the measurement does not provide enough information to estimate all the state variables of the system.

There are mathematical tests of observability, such as in [Jaz70]. Practically, two ways to fix the problem are: do not try to control an unobservable state variable; get other observations that do allow all relevant state variables to be controlled.

In the case of DALEC, LAI is proportional to C_f, so this is well constrained. Also, C_f directly influences GPP, which in turn directly affects C_r and C_w, so we might expect these to be controlled well. However, this is not the case for C_w, presumably because the coupling of GPP to C_w is weak. C_f also has a direct influence on C_lit, so the latter is well constrained. But C_som is two steps away from C_f, which is why it is not well constrained.

I had hoped to demonstrate successful use of the EnKF. It is known to work in many circumstances, (e.g. [Eve09]) but I assume that the set-up I used here is not amenable to estimation by the EnKF. It is, however, useful to see a partial failure, since this is rarely (perhaps never?) found in textbooks.

14 Parameter Estimation

Another job we might want to do, is to estimate some parameters in a model of a system. For example, in the rotational system of §8.1, the angular velocity ω is a parameter. If we did not know it accurately, we would have to estimate it, otherwise the data assimilation job would go haywire.

If we do not have an independent estimate of ω, we might try to estimate it from the data. This is parameter estimation by data assimilation. I have not produced a demonstration of it for this paper.

Another way to think of this is to note that it is a way of accomplishing nonlinear regression. In the common situation of fitting a straight line to (x, y) data, we have to estimate the intercept and slope of that straight line, by a process called linear regression. Linear regression can be extended to more variables and more complicated functions [PTVF92] — the name remains correct so long as the function to be fit is linear or affine in the coefficients that are to be estimated. More general is nonlinear regression, where the job is to estimate some numbers in a function that is neither linear nor affine in those numbers. In the present case, the function is given by the mathematical model. One way of estimating the numbers (i.e. the parameters) is by data assimilation.

14.1 By Nonlinear Sequential Filtering

I assume, for now, that there is only one parameter to be estimated. Extension to more than one is fairly plain. An intuitively appealing approach is to insert the parameter into the vector of things we have to estimate. Then, for example, our rotational model becomes the following, or some related discretisation:

$$\boldsymbol{x}_{n+1} = \begin{bmatrix} x_{n+1} \\ y_{n+1} \\ \omega_{n+1} \end{bmatrix} = \frac{1}{C_n} \begin{bmatrix} 1-\omega_n^2 dt^2/4 & -\omega_n\, dt & 0 \\ \omega_n\, dt & 1-\omega_n^2 dt^2/4 & 0 \\ 0 & 0 & C_n \end{bmatrix} \begin{bmatrix} x_n \\ y_n \\ \omega_n \end{bmatrix} + \begin{bmatrix} w_{x,n} \\ w_{y,n} \\ w_{\omega,n} \end{bmatrix}, \tag{14.1}$$

where $C_n = 1+\omega_n^2 dt^2/4$. (Observe that w and ω both occur here. They are not the same thing.) Thus, ω is stationary in the deterministic part of the model, but is given some additive noise that allows the Kalman filter to change it. The art of solving this problem is largely in tuning the variance of the random process $w_{\omega,n}$, which is the noise added to ω_n. If the noise is too small, the filter can fail because it is slow to arrive at a reasonable value of ω — meanwhile, it fails to follow the system's state because it has the wrong parameter in it. If the noise is too large, the estimate of the parameter is too weakly constrained, and it can wander around too much. Whatever the form, the estimate will wander about to some extent. I have tried this method of estimating ω by the EnKF, and found it much more difficult than merely estimating the state of the system. However, maybe I gave up too easily, because Mat Williams at the University of Edinburgh has made this work [Personal communication], and [LKSW03] have estimated state and parameters by a sequential filter.

Why does the title of this subsection say *Nonlinear*? Looking at equation (14.1), we see a matrix equation. However, the coefficients of x_n and y_n contain ω_n, which we also trying to estimate. The model equations are now a nonlinear set, and the plain KF no longer applies. This is a general problem with estimation of parameters by sequential filtering.

14.2 With a Filter Bank

Consider a linear state estimation problem, which is amenable to the plain KF. If the model contained one unknown parameter, we could run N KF jobs, each with a different value of that parameter. One of these jobs would be a better filter than the others, and the value of the parameter in that job would be the one to choose. These jobs, which could be run in parallel, are known as a *filter bank* [BH97], and this method has been known since 1965 or earlier. [BH97] discuss it, filling in many of the complexities — for example, in the relevant probability theory — that I have omitted here. [AM05] go into more depth and summarise existence proofs.

While this can be a useful tool in the estimation of a single parameter, it becomes problematic for many parameters. If we have n parameters to estimate, and each has N divisions in its range, then our filter bank will contain N^n KF jobs.

I speculate that we could produce a cost function that measures the performance of a single KF job, as a function of a vector of parameters. Then the best set of parameters would be the set that minimised this cost function. I have not yet brought this idea to fruition; but I would not be surprised if it were already thought through and either published or dismissed already by someone else.

14.3 Variationally

There is a simple, at first sight useless, rigorous extension of Var to estimating parameters, by inserting them into the vector of things to be estimated — a method akin to that in §14.1.

Let $\boldsymbol{p}$ be a set of fixed (i.e. they do not change through time, at least through a single Var analysis window) parameters arranged in an n_p-vector. Our state vector from §9 is now augmented by the vector $\boldsymbol{p}$, like this:

$$\tilde{\boldsymbol{x}}_k = (\boldsymbol{x}_k^T, \boldsymbol{p}^T)^T = \begin{bmatrix} \boldsymbol{x}_k \\ \hdashline \boldsymbol{p}_k \end{bmatrix}, \tag{14.2}$$

i.e. $\tilde{\boldsymbol{x}}$ is a column vector of length $n_x + n_p$. The second equality uses a partitioned matrix notation, applied to the vector.

The true augmented model state vector evolves like this:

$$\tilde{\boldsymbol{x}}_{k+1}^t = \hat{M}_k(\tilde{\boldsymbol{x}}_k^t) + \tilde{\boldsymbol{w}}_k, \tag{14.3}$$

where I have used the $\hat{M}_k$ notation again, because it is the same function as before. Using the partitioned matrix notation, this equation means:

$$\left[\begin{array}{c} \boldsymbol{x}_{k+1}^t \\ \hdashline \boldsymbol{p} \end{array}\right] = \left[\begin{array}{c} \hat{M}_k(\boldsymbol{x}_k^t) \\ \hdashline \boldsymbol{p} \end{array}\right] + \left[\begin{array}{c} \boldsymbol{w}_k \\ \hdashline \boldsymbol{0} \end{array}\right]. \tag{14.4}$$

So the evolution operator is essentially the same as in §9, because the $\boldsymbol{p}$ part of the augmented state vector is constant (but unknown).

Measurements of the system are also noisy, and may depend on the parameters.

$$\boldsymbol{y}_k = \hat{H}_k(\tilde{\boldsymbol{x}}_k^t) + \boldsymbol{v}_k, \tag{14.5}$$

similar to the §9 case. As before, it is assumed that $\tilde{\boldsymbol{w}}_k$ and $\boldsymbol{v}_k$ are independent.

Again, we define

$$\tilde{\boldsymbol{X}}_N^t = (\tilde{\boldsymbol{x}}_0^t, \tilde{\boldsymbol{x}}_1^t, \ldots, \tilde{\boldsymbol{x}}_N^t), \tag{14.6}$$

$$\boldsymbol{Y}_N = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_N). \tag{14.7}$$

There is now a tilde on the $\tilde{\boldsymbol{X}}_N^t$ because we include the system state and the parameters that are to be estimated. But $\boldsymbol{Y}_N$ is the same as before. We want to maximise the following conditional density with respect to the set of $\tilde{\boldsymbol{x}}$-vectors:

$$p(\tilde{\boldsymbol{X}}_N^t | \boldsymbol{Y}_N). \tag{14.8}$$

Using Bayes' rule, this density is written as

$$p(\tilde{\boldsymbol{X}}_N^t | \boldsymbol{Y}_N) = \frac{p(\boldsymbol{Y}_N | \tilde{\boldsymbol{X}}_N^t) p(\tilde{\boldsymbol{X}}_N^t)}{p(\boldsymbol{Y}_N)}. \tag{14.9}$$

The denominator does not depend on $\tilde{\boldsymbol{X}}_N^t$, so we can ignore it in maximising $p(\tilde{\boldsymbol{X}}_N^t | \boldsymbol{Y}_N)$ with respect to $\tilde{\boldsymbol{X}}_N^t$.

The analysis in §9 still works in the following two ways. First,

$$p(\boldsymbol{Y}_N | \tilde{\boldsymbol{X}}_N^t) = \prod_{k=1}^{N} p_{v,k}(\boldsymbol{y}_k - \hat{H}_k(\tilde{\boldsymbol{x}}_k^t)), \tag{14.10}$$

where $p_{v,k}(\boldsymbol{y}_k - \hat{H}_k(\tilde{\boldsymbol{x}}_k^t)) = p_{v,k}(\boldsymbol{v}_k) \sim N[\boldsymbol{0}, \boldsymbol{R}_k]$, a Gaussian density. Second, the sequence $\{\tilde{\boldsymbol{x}}_0^t, \ldots, \tilde{\boldsymbol{x}}_N^t\}$ is a Markov sequence, so

$$p(\tilde{\boldsymbol{X}}_N^t) = p(\tilde{\boldsymbol{x}}_0) \prod_{k=1}^{N} p(\tilde{\boldsymbol{x}}_k^t | \tilde{\boldsymbol{x}}_{k-1}^t) \tag{14.11}$$

$$= p(\tilde{\boldsymbol{x}}_0) \prod_{k=1}^{N} p_{w,k}(\tilde{\boldsymbol{x}}_k^t - \hat{M}_{k-1}(\tilde{\boldsymbol{x}}_{k-1}^t)), \tag{14.12}$$

where $p_{w,k}(\tilde{\boldsymbol{x}}_k - \hat{M}_{k-1}(\tilde{\boldsymbol{x}}_{k-1})) = p_{w,k}(\boldsymbol{w}_{k-1}) \sim N[\boldsymbol{0}, \boldsymbol{Q}_{k-1}]$.

We now have a more complicated background term. Our $\boldsymbol{x}_b$ becomes $\tilde{\boldsymbol{x}}_b$, a vector containing a background estimate not only of the initial value of the state vector, but also our prior estimate of the parameters. The background error covariance matrix must now be augmented as follows:

$$\boldsymbol{B} = \left[\begin{array}{c:c} \boldsymbol{B}_{xx} & \boldsymbol{B}_{xp} \\ \hdashline \boldsymbol{B}_{px} & \boldsymbol{B}_{pp} \end{array}\right], \tag{14.13}$$

where $\boldsymbol{B}_{xx}$ and $\boldsymbol{B}_{pp}$ are the background error covariance matrices for $\boldsymbol{x}^t$ and $\boldsymbol{p}$, respectively; and $\boldsymbol{B}_{xp}$ and $\boldsymbol{B}_{px}$ are the cross-covariance matrices. There is a bit of unpleasantness here: we need $\boldsymbol{B}^{-1}$, but the inverse of a partitioned matrix is not, generally, formed from the inverses of the blocks: see §A.6. However, if the cross-covariance matrices are zero, then the inverse *is* formed from the inverses of the diagonal blocks, and that is what I assume here. That is, the background errors of the system state and of the parameters are assumed to be uncorrelated so we can write

$$\boldsymbol{B}^{-1} = \left[\begin{array}{c:c} \boldsymbol{B}_{xx}^{-1} & \boldsymbol{0} \\ \hdashline \boldsymbol{0} & \boldsymbol{B}_{pp}^{-1} \end{array}\right]. \tag{14.14}$$

If this assumption is not thought to stand, the full formulæ for the inverse can be used, at least in principle, as in §A.6.

Equation (9.14) of §9 now becomes

$$\begin{aligned} J(\boldsymbol{x}_0, \boldsymbol{p}) = \frac{1}{2}(\boldsymbol{x}_0 - \boldsymbol{x}_b)^T \boldsymbol{B}_{xx}^{-1}(\boldsymbol{x}_0 - \boldsymbol{x}_b) + \frac{1}{2}(\boldsymbol{p} - \boldsymbol{p}_b)^T \boldsymbol{B}_{pp}^{-1}(\boldsymbol{p} - \boldsymbol{p}_b) \\ + \frac{1}{2}\sum_{k=1}^{N}(\boldsymbol{y}_k - \hat{H}_k(\tilde{\boldsymbol{x}}_k))^T \boldsymbol{R}_k^{-1}(\boldsymbol{y}_k - \hat{H}_k(\tilde{\boldsymbol{x}}_k)) \\ + \frac{1}{2}\sum_{k=1}^{N}(\boldsymbol{x}_k - \hat{M}_{k-1}(\boldsymbol{x}_{k-1}))^T (\boldsymbol{Q}_{k-1})^{-1}(\boldsymbol{x}_k - \hat{M}_{k-1}(\boldsymbol{x}_{k-1})). \end{aligned} \tag{14.15}$$

In the final term, we have $\boldsymbol{x}$ instead of $\tilde{\boldsymbol{x}}$, because it is only the system that evolves, not the parameters. (This is by assumption, and if this assumption is incorrect than we have a theory for an assimilation system that can be used, but which is not optimal.) The only explicit term involving the parameters is that containing $\boldsymbol{B}_{pp}^{-1}$, and this represents a statistical statement that penalises departures of the parameters from the prior (background) estimate. If you have no prior estimate, this term can be omitted — but that is probably not a very good idea.

Again, the model is our tool for minimising the cost function, and we have:

$$\begin{aligned} J(\boldsymbol{x}_0, \boldsymbol{p}) = \frac{1}{2}(\boldsymbol{x}_0 - \boldsymbol{x}_b)^T \boldsymbol{B}_{xx}^{-1}(\boldsymbol{x}_0 - \boldsymbol{x}_b) + \frac{1}{2}(\boldsymbol{p} - \boldsymbol{p}_b)^T \boldsymbol{B}_{pp}^{-1}(\boldsymbol{p} - \boldsymbol{p}_b) \\ + \frac{1}{2}\sum_{k=1}^{N}(\boldsymbol{y}_k - \hat{H}_k(\tilde{\boldsymbol{x}}_k))^T \boldsymbol{R}_k^{-1}(\boldsymbol{y}_k - \hat{H}_k(\tilde{\boldsymbol{x}}_k)). \end{aligned} \tag{14.16}$$

I said, a couple of pages back, that this method is "at first sight useless." This is because a parameter error is likely to lead at least to a bias, and probably a drift, of the model's trajectory w.r.t. that of the discretised system. However, the derivation of Var assumes that the model is unbiased, as in §9.3. But this is explicitly contradicted by the expected bias or drift that we expect from a model with incorrect parameters. That is, incorrect parameters cause an effect that contradicts the assumptions that start the derivation (unless we are very lucky — and even then, we probably would not know!).

However, variational estimation of parameters is likely to work anyway. Imagine that the model had correct initial values and correct parameters. Then the cost function would indeed be smaller than if incorrect initial values, or parameters, or both, were used. Thus, variational estimation of parameters is a rational method, albeit not necessarily an optimal one. Plainly, this variational method is akin to the filter bank in sequential estimation, and I suspect that it is, therefore, a good method, despite my lack of a proof. Perhaps a proof exists somewhere in the literature, and I have just not found it.

15 Introduction to Finding the Minimum in Var

The job in Var is to find a vector $\boldsymbol{x}_0$ that minimises $J(\boldsymbol{x}_0)$. This special value of $\boldsymbol{x}_0$ is the analysis, $\hat{\boldsymbol{x}}_0$ or $\boldsymbol{x}_a$. The literature on minimising functions, in textbooks and research, is vast. The process of minimisation is often called *optimisation*. There are robust methods that use only values of the function at different values of $\boldsymbol{x}_0$, such as the downhill simplex method [PTVF92]. These are generally slow, except under special circumstances. Faster methods, e.g. the BFGS method (e.g. [PTVF92] and [Fle87]), use values of $J(\boldsymbol{x}_0)$ and its gradient $\partial J(\boldsymbol{x}_0)/\partial \boldsymbol{x}_0$ at different values of $\boldsymbol{x}_0$.

So we need to find cheap ways of evaluating $J(\boldsymbol{x}_0)$ and $\partial J(\boldsymbol{x}_0)/\partial \boldsymbol{x}_0$. The former is done by a form of linearisation called *incremental Var*, in which most of the function evaluations are done with a linearised version of the model. It is linearised about a run of the full, nonlinear model, this run being known as a *linearisation state*. There are two basic modes of linearising the model. One is to start with the equations that are coded into the model. The equations are linearised, then the linearised equations are coded, to make what is known (in the Met Office) as a *perturbation forecast model*, or PF model. I do not discuss the PF model further in this tutorial. The second is to start with the nonlinear model code, and linearise this to form what is known as the *tangent linear* code. This is discussed below, in §20, starting on page 45.

The gradient can be found by a linearised model known as the *adjoint*, as explained below, also in §18.

16 Overview of Incremental Var.

§18 shows how to find the gradient of the cost function w.r.t. the initial state vector. This enables us to use gradient methods to minimise the cost function. However, it is still necessary to run the model forward many times, harvesting the various vectors and matrices each time. *Incremental Var* lets us speed this up by linearising the Var process about a single full model run called the linearisation state. Iterations of the minimiser then use the adjoints, etc., of the full model run (i.e. a linearised model), so this should be faster. These iterations are called *inner loops*, with the implication that there are *outer loops* somewhere. An outer loop is when you do a second (or third, ...) full model run, which is a new linearisation state around which to do another set of linear inner loops. N.B. there is no proof, that I know of, that this process will converge quickly, or even that it will converge at all! But practically, it works.

Incremental Var can also involve running models at lower spatial resolution than is normally attainable, to speed up the iterative procedure further. I do not discuss that here.

17 Details of Incremental 4D-Var

Here is one way of doing incremental Var. In it, I explicitly label various terms in accordance with outer loop iteration and inner loop iteration, which (I hope) will make the process easier to understand.

Some symbols that are usually subscripts are, in this section, superscripts. This is to prevent the clutter that would arise, because more subscripts are required here then elsewhere in the paper. For example, the background estimate is usually denoted by $\boldsymbol{x}_b$, but in this section by $\boldsymbol{x}^b$. And similarly with others. Since the argument of $J(\cdot)$, which is usually denoted by $\boldsymbol{x}_0$, is a dummy argument, I replace it in this section by $\boldsymbol{\xi}$, which makes the presentation a bit tidier.

Superscripts and subscripts are as follows.

- Time increments are denoted by τ, with $0 \le \tau \le T$;
- Outer loop increments are denoted by k, with $1 \le k \le K$;

- Inner loop increments are denoted by i, with $1 \leq i \leq I_k$, I_k depending on the outer loop that it is within, as the notation suggests.

The basic picture is that an inner loop does iterations about a fixed linearisation state (i.e. k is fixed, but i changes), while an outer loop changes the linearisation state (i.e. k changes).

Using a norm notation that I copied from Gordon Inverarity at the Met Office, the 4D-Var cost function is

$$J(\boldsymbol{\xi}) = \|\boldsymbol{\xi} - \boldsymbol{x}^b\|^2_{\boldsymbol{B}} + \sum_{\tau=1}^{T} \|\boldsymbol{y}^\tau - \hat{H}^\tau[\boldsymbol{\xi}^\tau]\|^2_{\boldsymbol{R}^\tau}, \tag{17.1}$$

where

$$\|\boldsymbol{\zeta}\|^2_{\mathbf{S}} = \boldsymbol{\zeta}^T \mathbf{S}^{-1} \boldsymbol{\zeta} \tag{17.2}$$

for any conformable vector $\boldsymbol{\zeta}$ and matrix $\mathbf{S}$.

Now consider the cost function at outer loop k and inner loop i, with initial state $\boldsymbol{\xi}^0_{k,i}$. That is, the initial state varies between inner loops *and* outer loops.

$$J(\boldsymbol{\xi}^0_{k,i}) = \|\boldsymbol{\xi}^0_{k,i} - \boldsymbol{x}^b\|^2_{\boldsymbol{B}} + \sum_{\tau=1}^{T} \|\boldsymbol{y}^\tau - \hat{H}^\tau[\boldsymbol{\xi}^\tau_{k,i}]\|^2_{\boldsymbol{R}^\tau}. \tag{17.3}$$

Let the initial state be split between a part that depends only on the outer loop, and the remainder:

$$\boldsymbol{\xi}^0_{k,i} = \boldsymbol{\xi}^{g,0}_k + \delta\boldsymbol{\xi}^0_{k,i}. \tag{17.4}$$

Also, let the model evolution operator be $\hat{M}^{0\to\tau}$, taking any initial state to the state at discrete time τ, by running the full nonlinear model:

$$\begin{aligned}
\boldsymbol{\xi}^\tau_{k,i} &= \hat{M}^{0\to\tau}[\boldsymbol{\xi}^0_{k,i}] && (17.5a)\\
&= \hat{M}^{0\to\tau}[\boldsymbol{\xi}^{g,0}_k + \delta\boldsymbol{\xi}^0_{k,i}] && (17.5b)\\
&\simeq \hat{M}^{0\to\tau}[\boldsymbol{\xi}^{g,0}_k] + \boldsymbol{M}^{0\to\tau}_k \delta\boldsymbol{\xi}^0_{k,i} && (17.5c)\\
&= \boldsymbol{\xi}^{g,\tau}_k + \boldsymbol{M}^{0\to\tau}_k \delta\boldsymbol{\xi}^0_{k,i}, && (17.5d)
\end{aligned}$$

where $\boldsymbol{M}^{0\to\tau}_k$ is the Jacobian and $\boldsymbol{\xi}^{g,\tau}_k$ is, by definition, $\hat{M}^{0\to\tau}[\boldsymbol{\xi}^{g,0}_k]$. A short way below, the operator $\hat{M}^\tau$ (observe that the superscripted "$0\to$" is absent) means the evolution from one time-step to the next.

Now we "incrementalise" equation (17.3), looking at the two parts of the cost function in turn.

$$\begin{aligned}
J_{\boldsymbol{B}} &= \|\boldsymbol{\xi}^0_{k,i} - \boldsymbol{x}^b\|^2_{\boldsymbol{B}} && (17.6a)\\
&= \|\boldsymbol{\xi}^{g,0}_k + \delta\boldsymbol{\xi}^0_{k,i} - \boldsymbol{x}^b\|^2_{\boldsymbol{B}} && (17.6b)\\
&= \|\delta\boldsymbol{\xi}^0_{k,i} - (\boldsymbol{x}^b - \boldsymbol{\xi}^{g,0}_k)\|^2_{\boldsymbol{B}} && (17.6c)\\
&= \|\delta\boldsymbol{\xi}^0_{k,i} - \delta\boldsymbol{\xi}^{g,0}_k\|^2_{\boldsymbol{B}}, && (17.6d)
\end{aligned}$$

where, by definition, $\delta\boldsymbol{\xi}^{g,0}_k = \boldsymbol{x}^b - \boldsymbol{\xi}^{g,0}_k$. Now

$$\begin{aligned}
J_{\boldsymbol{R}^\tau} &= \|\boldsymbol{y}^\tau - \hat{H}^\tau[\boldsymbol{\xi}^\tau_{k,i}]\|^2_{\boldsymbol{R}^\tau} && (17.7a)\\
&\simeq \|\boldsymbol{y}^\tau - \hat{H}^\tau[\boldsymbol{\xi}^{g,\tau}_k + \boldsymbol{M}^{0\to\tau}_k \delta\boldsymbol{\xi}^0_{k,i}]\|^2_{\boldsymbol{R}^\tau} && (17.7b)\\
&\simeq \|\boldsymbol{y}^\tau - \hat{H}^\tau[\boldsymbol{\xi}^{g,\tau}_k] - \boldsymbol{H}^\tau_k \boldsymbol{M}^{0\to\tau}_k \delta\boldsymbol{\xi}^0_{k,i}\|^2_{\boldsymbol{R}^\tau} && (17.7c)\\
&= \|\boldsymbol{d}^\tau_k - \boldsymbol{H}^\tau_k \boldsymbol{M}^{0\to\tau}_k \delta\boldsymbol{\xi}^0_{k,i}\|^2_{\boldsymbol{R}^\tau}, && (17.7d)
\end{aligned}$$

where, by definition, $\boldsymbol{H}_k^\tau$ is the Jacobian and $\boldsymbol{d}_k^\tau = \boldsymbol{y}^\tau - \hat{H}^\tau[\boldsymbol{\xi}_k^{g,\tau}]$.

So equation (17.3) becomes

$$J(\boldsymbol{\xi}_{k,i}^0) = \|\delta\boldsymbol{\xi}_{k,i}^0 - \delta\boldsymbol{\xi}_k^{g,0}\|_{\boldsymbol{B}}^2 + \sum_{\tau=1}^{T} \|\boldsymbol{d}_k^\tau - \boldsymbol{H}_k^\tau \boldsymbol{M}_k^{0\to\tau}\delta\boldsymbol{\xi}_{k,i}^0\|_{\boldsymbol{R}^\tau}^2. \tag{17.8}$$

It is now easy to see which objects must be calculated when. Those with subscripts k, i must be calculated every inner loop:

$$\delta\boldsymbol{\xi}_{k,i}^0 \qquad \boldsymbol{H}_k^\tau \boldsymbol{M}_k^\tau \delta\boldsymbol{\xi}_{k,i}^0, \tag{17.9}$$

and those with subscript k alone need only be calculated every outer loop:

$$\delta\boldsymbol{\xi}_k^{g,0} \qquad \boldsymbol{d}_k^\tau \qquad \boldsymbol{H}_k^\tau \qquad \boldsymbol{M}_k^\tau. \tag{17.10}$$

The $\boldsymbol{M}_k^\tau$ are linearisations of the full, nonlinear model. They are realised as either tangent linear models or PF models, and assembled from single-time-step linearisations, like this:

$$\boldsymbol{\xi}_{k,i}^1 = \hat{M}_{k,i}^0(\boldsymbol{\xi}_{k,i}^0) = \hat{M}_{k,i}^0(\boldsymbol{\xi}_k^{g,0} + \delta\boldsymbol{\xi}_{k,i}^0) \simeq \hat{M}_{k,i}^0(\boldsymbol{\xi}_k^{g,0}) + \boldsymbol{M}_k^0\delta\boldsymbol{\xi}_{k,i}^0 = \boldsymbol{\xi}_k^{g,1} + \boldsymbol{M}_k^0\delta\boldsymbol{\xi}_{k,i}^0. \tag{17.11}$$

Then,

$$\boldsymbol{\xi}_{k,i}^2 = \hat{M}_{k,i}^1(\boldsymbol{\xi}_{k,i}^1) \simeq \hat{M}_{k,i}^1[\boldsymbol{\xi}_k^{g,1} + \boldsymbol{M}_k^0\delta\boldsymbol{\xi}_{k,i}^0)] \simeq \hat{M}_{k,i}^1(\boldsymbol{\xi}_k^{g,1}) + \boldsymbol{M}_k^1\boldsymbol{M}_k^0\delta\boldsymbol{\xi}_{k,i}^0 = \boldsymbol{\xi}_k^{g,2} + \boldsymbol{M}_k^1\boldsymbol{M}_k^0\delta\boldsymbol{\xi}_{k,i}^0. \tag{17.12}$$

And,

$$\boldsymbol{\xi}_{k,i}^3 = \hat{M}_{k,i}^2(\boldsymbol{\xi}_{k,i}^3) \simeq \hat{M}_{k,i}^2[\boldsymbol{\xi}_k^{g,2} + \boldsymbol{M}_k^1\boldsymbol{M}_k^0\delta\boldsymbol{\xi}_{k,i}^0)] \simeq \hat{M}_{k,i}^2(\boldsymbol{\xi}_k^{g,2}) + \boldsymbol{M}_k^2\boldsymbol{M}_k^1\boldsymbol{M}_k^0\delta\boldsymbol{\xi}_{k,i}^0 = \boldsymbol{\xi}_k^{g,3} + \boldsymbol{M}_k^2\boldsymbol{M}_k^1\boldsymbol{M}_k^0\delta\boldsymbol{\xi}_{k,i}^0. \tag{17.13}$$

So, by induction:

$$\boldsymbol{\xi}_{k,i}^n \simeq \hat{M}_{k,i}^{n-1}(\boldsymbol{\xi}_k^{g,n-1}) + \boldsymbol{M}_k^{n-1}\boldsymbol{M}_k^{n-2}\cdots\boldsymbol{M}_k^0\delta\boldsymbol{\xi}_{k,i}^0 = \boldsymbol{\xi}_k^{g,n} + \boldsymbol{M}_k^{n-1}\boldsymbol{M}_k^{n-2}\cdots\boldsymbol{M}_k^0\delta\boldsymbol{\xi}_{k,i}^0. \tag{17.14}$$

The $\hat{M}_k^i$ are just steps of the full nonlinear model. The $\boldsymbol{M}_k^i$ are the steps of the linearised model.

The whole process starts with $k = 1$, i.e. the first outer loop. Then the first guess is $\boldsymbol{\xi}_{1,1}^0 = \boldsymbol{\xi}_1^g + \boldsymbol{0}$, i.e. $\delta\boldsymbol{\xi}_{1,1}^0 = \boldsymbol{0}$. The full model is run to produce the first linearisation state, and a Var job is done using the linearised model, until convergence has been achieved or for a pre-defined number of iterations (i.e. of inner loops). Then a second outer loop is initiated by forming $\boldsymbol{\xi}_2^{g,0} = \boldsymbol{\xi}_1^{g,0} + \delta\boldsymbol{\xi}_{1,M}^0$, where $\boldsymbol{\xi}_{1,M}^0$ is the optimal value of $\boldsymbol{\xi}_{1,i}^0$. Then another linearised optimisation is done, with inner loops. In this way, the outer loops can be iterated either to convergence or for a pre-determined number of iterations.

18 Gradient Calculation for State Estimation

The analysis is a little intricate, so before presenting the useful derivation for a general vector $\boldsymbol{x}_0$, I give a derivation for the simplified case where the model, the system and observation vectors are scalars.

18.1 Gradient of J for Scalar Systems.

The Var equation is now

$$J(x_0) = \frac{(x_0 - x_b)^2}{2B} + \sum_{i=1}^{N} \frac{(y_i - \hat{H}(x_i))^2}{2R_i}. \tag{18.1}$$

Minimisation of the background term J_b, does not involve linearisation.

Let us now consider the usual Var situation, in which we are only trying to find a best starting value of x_0, with the parameters being fixed. We want to find $\partial J_o/\partial x_0$, (where J_o is the observations part of the cost function, i.e. the subscript is "oh", not "zero"), so we can do an efficient search using derivatives. First, observe that $J_o = \sum_1^N J_{o,i}$, to use an obvious notation. Then

$$\frac{\partial J_{o,i}}{\partial x_0} = \frac{\partial}{\partial x_0}\frac{[y_i - \hat{H}(x_i)]^2}{2R_i} \tag{18.2a}$$

$$= \frac{[\hat{H}(x_i) - y_i]}{R_i}\frac{\partial}{\partial x_0}\hat{H}(x_i) \tag{18.2b}$$

$$= \frac{d_i H_i}{R_i}\frac{\partial x_i}{\partial x_0} \tag{18.2c}$$

$$= \frac{d_i H_i}{R_i}\frac{\partial x_i}{\partial x_{i-1}}\frac{\partial x_{i-1}}{\partial x_{i-2}}\cdots\frac{\partial x_2}{\partial x_1}\frac{\partial x_1}{\partial x_0}, \tag{18.2d}$$

where $d_i = \hat{H}(x_i) - y_i$ and $H_i = \partial\hat{H}(x_i)/\partial x_i$. Now we can write

$$\frac{\partial x_i}{\partial x_{i-1}} = \frac{\partial\hat{M}_{i-1}(x_{i-1})}{\partial x_{i-1}} = M_{i-1}, \tag{18.3}$$

which defines the scalar M_{i-1}. So

$$\frac{\partial J_{o,i}}{\partial x_0} = \frac{d_i H_i}{R_i}M_{i-1}M_{i-2}\ldots M_0. \tag{18.4}$$

So, as well as the derivatives, we have to do $i+1$ multiplications and 1 division in calculating $\partial J_{o,i}/\partial x_0$. So in calculating $\partial J_o/\partial x_0$, it looks like we need $O(N^2)$ multiplications, N divisions and $N-1$ additions.

A clever factorisation (Horner's Rule, [Weia]) reduces this from $O(N^2)$ to $O(N)$. I illustrate this for $N = 5$, using for brevity, $\zeta_i = d_i H_i/R_i$:

$$\begin{aligned}\zeta_1 M_0 + \zeta_2 M_1 M_0 + \zeta_3 M_2 M_1 M_0 + \zeta_4 M_3 M_2 M_1 M_0 + \zeta_5 M_4 M_3 M_2 M_1 M_0 \\ = M_0(\zeta_1 + M_1(\zeta_2 + M_2(\zeta_3 + M_3(\zeta_4 + M_4\zeta_5))))),\end{aligned} \tag{18.5}$$

which has a complexity of $O(N)$ multiplications, N divisions and $N-1$ additions.

18.2 Gradient of J for Vector Systems

The scalar analysis is a step-up to understanding the vector case. I next show how it works for a vector state function, $\boldsymbol{x}(t)$, and a vector of observations, $\boldsymbol{y}(t)$. Again, I will show how to get a useful expression for the gradient of the i^{th} term of J_o.

Now, the Var equation is more familiar:

$$J(\boldsymbol{x}_0) = \frac{1}{2}(\boldsymbol{x}_0 - \boldsymbol{x}_b)^T\boldsymbol{B}^{-1}(\boldsymbol{x}_0 - \boldsymbol{x}_b) + \frac{1}{2}\sum_{k=1}^{N}(\boldsymbol{y}_k - \hat{H}_k(\boldsymbol{x}_k))^T\boldsymbol{R}_k^{-1}(\boldsymbol{y}_k - \hat{H}_k(\boldsymbol{x}_k)), \tag{18.6}$$

A problem arises in the usual vector-matrix notation, which I now explain for a simpler example. Let $\boldsymbol{f}(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ be two vector-valued functions of some other vector $\boldsymbol{x}$. If they have the same lengths, we can define this scalar:

$$Z(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x})^T\boldsymbol{g}(\boldsymbol{x}). \tag{18.7}$$

Then $\nabla Z = \partial Z/\partial \boldsymbol{x}$ must be a column vector (well, it could be a row vector, but the gradient of a scalar is usually thought of as a column; and even if we think of it as a row, the same essential problem will arise). Now, using the product rule of differential calculus,

$$\frac{\partial Z(\boldsymbol{x})}{\partial \boldsymbol{x}} = \frac{\partial}{\partial \boldsymbol{x}}[\boldsymbol{f}(\boldsymbol{x})^T \boldsymbol{g}(\boldsymbol{x})] \tag{18.8a}$$

$$= \frac{\partial \boldsymbol{f}(\boldsymbol{x})^T}{\partial \boldsymbol{x}} \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{f}(\boldsymbol{x})^T \frac{\partial \boldsymbol{g}(\boldsymbol{x})}{\partial \boldsymbol{x}}. \tag{18.8b}$$

But the first term in equation (18.8b) is a column vector, while the second term is a row vector. We can avoid this apparent nonsense (and it really is only apparent — caused by pushing the notation further than it can cope) by using index notation and the summation convention [Wik07a] that repeated indices imply summation. To illustrate, Z and its gradient become:

$$Z(\boldsymbol{x}) = f_\alpha(\boldsymbol{x}) g_\alpha(\boldsymbol{x}), \tag{18.9a}$$

$$\frac{\partial Z}{\partial \boldsymbol{x}} = \frac{\partial Z}{\partial x_\beta} \tag{18.9b}$$

$$= \frac{\partial f_\alpha(\boldsymbol{x}) g_\alpha(\boldsymbol{x})}{\partial x_\beta} \tag{18.9c}$$

$$= \frac{\partial f_\alpha(\boldsymbol{x})}{\partial x_\beta} g_\alpha(\boldsymbol{x}) + \frac{\partial g_\alpha(\boldsymbol{x})}{\partial x_\beta} f_\alpha(\boldsymbol{x}). \tag{18.9d}$$

Extension to the slightly more complicated case of the observation part of the cost function is fairly easy if you are confident with this notation.

We want to find $\partial J_{o,i}/\partial \boldsymbol{x}_0$, as follows. Assume that we want to compute the subscript-γ component of the gradient:

$$\frac{\partial J_{o,i}}{\partial \boldsymbol{x}_0} = \frac{\partial J_{o,i}}{\partial [\boldsymbol{x}_0]_\gamma} \tag{18.10a}$$

$$= \frac{1}{2} \frac{\partial}{\partial [\boldsymbol{x}_0]_\gamma} \left\{ [\boldsymbol{y}_i - \hat{H}_i(\boldsymbol{x}_i)]_\alpha [\boldsymbol{R}_i^{-1}]_{\alpha\beta} [\boldsymbol{y}_i - \hat{H}_i(\boldsymbol{x}_i)]_\beta \right\} \tag{18.10b}$$

$$= \frac{1}{2} \frac{\partial [\boldsymbol{y}_i - \hat{H}_i(\boldsymbol{x}_i)]_\alpha}{\partial [\boldsymbol{x}_0]_\gamma} [\boldsymbol{R}_i^{-1}]_{\alpha\beta} [\boldsymbol{y}_i - \hat{H}_i(\boldsymbol{x}_i)]_\beta + \frac{1}{2} \frac{\partial [\boldsymbol{y}_i - \hat{H}_i(\boldsymbol{x}_i)]_\beta}{\partial [\boldsymbol{x}_0]_\gamma} [\boldsymbol{R}_i^{-1}]_{\alpha\beta} [\boldsymbol{y}_i - \hat{H}_i(\boldsymbol{x}_i)]_\alpha \tag{18.10c}$$

$$= \frac{\partial [\boldsymbol{y}_i - \hat{H}_i(\boldsymbol{x}_i)]_\alpha}{\partial [\boldsymbol{x}_0]_\gamma} [\boldsymbol{R}_i^{-1}]_{\alpha\beta} [\boldsymbol{y}_i - \hat{H}_i(\boldsymbol{x}_i)]_\beta \tag{18.10d}$$

$$= -\frac{\partial [\hat{H}_i(\boldsymbol{x}_i)]_\alpha}{\partial [\boldsymbol{x}_0]_\gamma} [\boldsymbol{R}_i^{-1}]_{\alpha\beta} [\boldsymbol{y}_i - \hat{H}_i(\boldsymbol{x}_i)]_\beta \tag{18.10e}$$

$$= -\frac{\partial [\hat{H}_i(\boldsymbol{x}_i)]_\alpha}{\partial [\boldsymbol{x}_i]_\delta} \frac{\partial [\boldsymbol{x}_i]_\delta}{\partial [\boldsymbol{x}_0]_\gamma} [\boldsymbol{R}_i^{-1}]_{\alpha\beta} [\boldsymbol{y}_i - \hat{H}_i(\boldsymbol{x}_i)]_\beta \tag{18.10f}$$

$$= -\left[\frac{\partial [\boldsymbol{x}_i]_\delta}{\partial [\boldsymbol{x}_0]_\gamma}\right]_{\delta\gamma} \left[\frac{\partial [\hat{H}_i(\boldsymbol{x}_i)]_\alpha}{\partial [\boldsymbol{x}_i]_\delta}\right]_{\alpha\delta} [\boldsymbol{R}_i^{-1}]_{\alpha\beta} [\boldsymbol{y}_i - \hat{H}_i(\boldsymbol{x}_i)]_\beta \tag{18.10g}$$

$$= -\left[\frac{\partial [\boldsymbol{x}_i]_\delta}{\partial [\boldsymbol{x}_0]_\gamma}\right]^T_{\gamma\delta} \left[\frac{\partial [\hat{H}_i(\boldsymbol{x}_i)]_\alpha}{\partial [\boldsymbol{x}_i]_\delta}\right]^T_{\delta\alpha} [\boldsymbol{R}_i^{-1}]_{\alpha\beta} [\boldsymbol{y}_i - \hat{H}_i(\boldsymbol{x}_i)]_\beta. \tag{18.10h}$$

The first equality re-writes the gradient using the index notation. The second expands $J_{o,i}$. The third applies the product rule, remembering that $\boldsymbol{R}^{-1}$ is constant. The fourth is a rearrangement, remembering that indices that are summed over are dummy indices, and can be given any names you please. The fifth arises because $\boldsymbol{y}_i$ is constant. The sixth expands $\partial \hat{H}(\boldsymbol{x}_i)/\partial \boldsymbol{x}_0$ using the chain rule of differential calculus. The seventh is a first step from index/summation notation

back to matrix/vector notation — big square brackets are placed around objects that can be thought of as matrices. The eighth transposes the matrix-like objects, so all the objects in the equation are in the correct order to give a subscripted γ at the left — we want this because the gradient has a subscripted γ at the first equality.

We need to convert all this subscripty stuff into matrix and vector notation, remembering that the gradient of $J_{o,i}$ is a column-vector. Now $\left[\frac{\partial[\boldsymbol{x}_i]_\delta}{\partial[\boldsymbol{x}_0]_\gamma}\right]^T_{\gamma\delta}$ is the transpose, or *adjoint,* of the Jacobian of $\boldsymbol{x}_i$ w.r.t. $\boldsymbol{x}_0$. (That Jacobian is also known as the tangent linear model.) A similar statement can be made for $\left[\frac{\partial[\hat{H}(\boldsymbol{x}_i)]_\alpha}{\partial[\boldsymbol{x}_i]_\delta}\right]^T_{\delta\alpha}$. It is the transpose of the Jacobian of $\hat{H}(\boldsymbol{x}_i)$ w.r.t. $\boldsymbol{x}_i$. Putting all this back together, we get:

$$\frac{\partial J_{o,i}}{\partial \boldsymbol{x}_0} = -\left[\frac{\partial \boldsymbol{x}_i}{\partial \boldsymbol{x}_0}\right]^T \left[\frac{\partial \hat{H}_i(\boldsymbol{x}_i)}{\partial \boldsymbol{x}_i}\right]^T \boldsymbol{R}_i^{-1}[\boldsymbol{y}_i - \hat{H}(\boldsymbol{x}_i)] \tag{18.11a}$$

$$= -\left[\frac{\partial \boldsymbol{x}_i}{\partial \boldsymbol{x}_{i-1}}\frac{\partial \boldsymbol{x}_{i-1}}{\partial \boldsymbol{x}_{i-2}}\cdots\frac{\partial \boldsymbol{x}_2}{\partial \boldsymbol{x}_1}\frac{\partial \boldsymbol{x}_1}{\partial \boldsymbol{x}_0}\right]^T \left[\frac{\partial \hat{H}_i(\boldsymbol{x}_i)}{\partial \boldsymbol{x}_i}\right]^T \boldsymbol{R}_i^{-1}[\boldsymbol{y}_i - \hat{H}(\boldsymbol{x}_i)] \tag{18.11b}$$

$$= -[\boldsymbol{M}_{i-1}\boldsymbol{M}_{i-2}\ldots\boldsymbol{M}_1\boldsymbol{M}_0]^T \boldsymbol{H}_i^T \boldsymbol{R}_i^{-1}[\boldsymbol{y}_i - \hat{H}(\boldsymbol{x}_i)] \tag{18.11c}$$

$$= -\boldsymbol{M}_0^T\boldsymbol{M}_1^T\ldots\boldsymbol{M}_{i-2}^T\boldsymbol{M}_{i-1}^T\boldsymbol{H}_i^T\boldsymbol{R}_i^{-1}\boldsymbol{d}_i, \tag{18.11d}$$

where $\boldsymbol{M}_i$ is the Jacobian or tangent linear of $\hat{M}_i$, so $\boldsymbol{M}_i^T$ is the adjoint; $\boldsymbol{H}_i$ is the Jacobian of $\hat{H}_i$; and $\boldsymbol{d}_i = \boldsymbol{y}_i - \hat{H}(\boldsymbol{x}_i)$.

Then $\partial J_o/\partial \boldsymbol{x}_0$ is the sum of all the $\partial J_{o,i}/\partial \boldsymbol{x}_0$ terms, like it was in the 1-D case above. Again, we can use Horner's rule to factorise the sum of products[4]. This is much more important now, because we are dealing with matrix products and sums instead of scalars. For $N = 5$:

$$\frac{\partial J_o}{\partial \boldsymbol{x}_0} = \boldsymbol{M}_0^T(\boldsymbol{H}_1^T\boldsymbol{R}_1^{-1}\boldsymbol{d}_1+\boldsymbol{M}_1^T(\boldsymbol{H}_2^T\boldsymbol{R}_2^{-1}\boldsymbol{d}_2+\boldsymbol{M}_2^T(\boldsymbol{H}_3^T\boldsymbol{R}_3^{-1}\boldsymbol{d}_3+\boldsymbol{M}_3^T(\boldsymbol{H}_4^T\boldsymbol{R}_4^{-1}\boldsymbol{d}_4+\boldsymbol{M}_4^T\boldsymbol{H}_5^T\boldsymbol{R}_5^{-1}\boldsymbol{d}_5)))). \tag{18.12}$$

To calculate this monster, you need to run the model forward, calculating and storing (or just storing) $\boldsymbol{d}_i$, $\boldsymbol{M}_i^T$, $\boldsymbol{R}_i^{-1}$ and $\boldsymbol{H}_i^T$ at each time-step. Then do all these matrix multiplications and additions, starting with the last (number 4/5 in this example) and proceeding to the first. This is the origin of the infamous phrase "integrating backwards through time".

Instead of working as stated here, we could have taken the transpose of everything to end up with row-vectors and non-transposed matrices. The result would be the same, considered as a set of numbers.

The tangent linears and adjoints are multidimensional generalisations of the scalar M things we saw in analysis of the 1-D problem. As the latter are scalars, transposition is irrelevant, and the "adjoint" jargon only appears in the multidimensional case.

19 Gradient Calculation for Parameter Estimation

The specification of tangent linear and adjoint is a bit more complicated when we need the gradient w.r.t. a parameter, or parameters. However, it is easy to follow it through, once you have understood the previous section, so I omit it, except for a couple of notes of guidance.

For scalars, we want to calculate $\partial J_{o,i}/\partial p$, for some parameter p.

$$\frac{\partial J_{o,i}}{\partial p} = \frac{\hat{H}(x_i;p) - y_i}{R_i}\frac{\partial \hat{H}(x_i;p)}{\partial p}. \tag{19.1}$$

[4] [GV96] shows that Hornerisation of a matrix polynomial (which this is) is not optimal. However, the optimal algorithm is complex, and probably not worth the additional effort.

Now we encounter a common notational problem with partial derivatives. $\hat{H}(x_i; p)$ can change by direct changes in p, and also by changes in x_i that are themselves caused by changes in p. A good way to deal with it is to adopt a convention that is used in many thermodynamics textbooks, like this:

$$\frac{\partial \hat{H}(x_i; p)}{\partial p} = \left[\frac{\partial \hat{H}(x_i; p)}{\partial p}\right]_{x_i} + \left[\frac{\partial \hat{H}(x_i; p)}{\partial x_i}\right]_p \frac{\partial x_i}{\partial p}, \tag{19.2}$$

where the large square brackets have subscripts that indicate what is being kept constant while something else is varying.

For vector systems with vectors of parameters, the derivation is similar, but with a more complicated vector analysis.

20 Automatic Differentiation (AD) of Code

20.1 Introduction

In the earlier parts of this tutorial, I showed the need for tangent linear and adjoint codes. There are two fundamental modes of automatic production of linearised code from nonlinear code: *forward mode* and *backward mode*. For these two modes, there are three methods that I am aware of: *source code transformation*, *operator overloading*, and *the complex step method*. Each of these methods involves changes to the code, which may be done, to some extent, automatically. In practice, it is usually found that considerable human intervention is required, so "AD" is often used to mean "Algorithmic Differentiation". Automatic Differentiation is synonymous with Algorithmic Differentiation.

The following subsections discuss:

- the source code transformation method for forward and backward modes;
- the operator overloading method for forward mode;
- the complex step method for forward mode.

Results of a forward mode analysis are often referred to as tangent linears, while results of a backward mode analysis are often referred to as adjoints. (Sometimes people will say "adjoint" to mean either or both!) I find this nomenclature unhelpful, since both modes provide derivatives of inputs w.r.t. outputs. The meaning of "adjoint" earlier in this tutorial is "transpose". When discussing AD, I suggest that the term *linearisation* be used, alongside forward or backward mode.

It is most important to understand that these modes and methods provide algorithms for generating real numbers; they are not symbolic manipulators, such as Maple or Mathematica. When you write one as computer code, it outputs real numbers.

To learn more, Griewank's book [Gri00] is a comprehensive, but difficult (for me, at least) reference on this subject.

20.2 Source Code Transformation

20.2.1 Rosenbrock's Banana

The banana function, due to Rosenbrock, is useful as a tester and demonstrator of optimisation algorithms. Here it is:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2. \tag{20.1}$$

If you plot it as a surface, it looks nothing like a banana. But it does contain a curved valley in which a mathematical banana could stably rest. Here it is just a simple, but not trivial, function

that I will show how to differentiate algorithmically. I do not know the reference to the first publication of this function, but [Weib] seems to think it is [Ros60].

The following pseudocode shows how we could construct the function, given x and y as inputs. I have spun it out a bit to show how the method works.

1. Input x and y
2. $w_1 = (1 - x)^2$
3. $w_2 = y - x^2$
4. $w_3 = 100w_2^2$
5. $f = w_1 + w_3$
6. Output f

Remember that x and y are real numbers, therefore all the intermediate w variables and the output f are also real numbers. This is about working with numbers, not algebraic symbols.

20.2.2 Differentiating the Banana: Forward Mode

Now we can differentiate it line-by-line, using the chain rule to get the differential of the whole function. I say $d.X$ to mean the differential of X, for any symbol X.

1. Input x, $d.x$, y and $d.y$
2. $w_1 = (1 - x)^2$
3. $\quad d.w_1 = 2(x - 1)\, d.x$
4. $w_2 = y - x^2$
5. $\quad d.w_2 = d.y - 2x\, d.x$
6. $w_3 = 100w_2^2$
7. $\quad d.w_3 = 200w_2\, d.w_2$
8. $f = w_1 + w_3$
9. $\quad d.f = d.w_1 + d.w_3$
10. Output f and $d.f$

Now x, $d.x$, y and $d.y$ are all real numbers, so the intermediate w variables are real numbers, so the outputs f and $d.f$ are real numbers. This is not about the symbolic calculus that Maple and Mathematica can do — it is about getting numbers out of an algorithm, and $d.f$ is a number that is the value of a derivative of f at the input values of x and y.

If we substitute all the partial differentials in order until we only get functions of $d.x$ and $d.y$ on the right-hand side, we find:

$$\begin{aligned} d.f &= d.w_1 + d.w_3 && \text{(20.2a)}\\ &= 2(x - 1)\, d.x + 200w_2\, d.w_2 && \text{(20.2b)}\\ &= 2(x - 1)\, d.x + 200(y - x^2)(d.y - 2x\, d.x) && \text{(20.2c)}\\ &= [2(x - 1) - 400x(y - x^2)]\, d.x + 200(y - x^2)\, d.y && \text{(20.2d)}\\ &= \frac{\partial f}{\partial x} dx + \frac{\partial f}{\partial y} dy. && \text{(20.2e)} \end{aligned}$$

We can get $\partial f/\partial x$ in our pseudocode by setting $d.y \leftarrow 0$ and $d.x \leftarrow 1$; and we can obtain $\partial f/\partial y$ by setting $d.y \leftarrow 1$ and $d.x \leftarrow 0$. So we do not need two algorithmic differential codes for the

two partial derivatives. *We just use one code,* calling it twice with different input values of $d.x$ and $d.y$.

In summary, the forward mode calculates the differentials in the same order that they are calculated. To get the full gradient of the scalar output, i.e. $\partial f/\partial x$ and $\partial f/\partial x$, we have to call the code twice. We could also calculate a directional derivative by calling it once, for example with $d.x \leftarrow 0.5$ and $d.y \leftarrow 0.866$.

The full gradient ∇f can be calculated in one call of a slightly adapted forward mode — see §20.2.9.

I think an extension to vector-valued functions can be done the same way, but I need to understand that better before writing it up.

There is *much* more to AD than what is presented here, but this is a useful start. In case forward mode is still a bit mysterious, I next apply it to a more complicated scalar function of three input variables.

20.2.3 Pearson's Squiggle

Here it is:

$$f(x, y, z) = \left[xy + \sin\left(\frac{x}{yz}\right) + (xyz)^2\right]^3. \tag{20.3}$$

A merit of the Banana is that it is useful in other contexts. The Squiggle is only useful for the purposes of this note.

Next I present some pseudocode to calculate the Squiggle. Observe that the inputs, x, y and z are directly represented as code variables, w, unlike the Banana's pseudocode above. Although this is entirely a matter of taste, [Gri00] does it, so I am doing it here.

1. Input x, y and z.
2. $w_1 = x$
3. $w_2 = y$
4. $w_3 = z$
5. $w_4 = w_1 w_2$
6. $w_5 = w_3 w_4$
7. $w_6 = w_5^2$
8. $w_7 = \frac{w_1}{w_2 w_3}$
9. $w_8 = \sin w_7$
10. $w_9 = w_4 + w_6 + w_8$
11. $w_{10} = w_9^3$
12. $f = w_{10}$
13. Output f

Thus, for example, $w_5 = xyz$. As with the Banana, all the symbols represent real numbers.

20.2.4 Differentiating the Squiggle: Forward Mode

To find the derivatives of f with respect to x, y and z, i.e. $\partial f/\partial x$, $\partial f/\partial y$ and $\partial f/\partial z$, we differentiate this algorithm line-by-line. Each line's right-hand side contains only variables that have been calculated already, i.e. in lines above it[5]. By "differentiate", I mean we form the differential, not a single partial derivative. Thus:

[5]What would happen in a recursive statement, such as are allowed in modern Fortrans and some other languages? If you know, please tell me.

1. Input x, $d.x$, y, $d.y$, z and $d.z$.
2. $w_1 = x$
3. $\quad d.w_1 = d.x$
4. $w_2 = y$
5. $\quad d.w_2 = d.y$
6. $w_3 = z$
7. $\quad d.w_3 = d.z$
8. $w_4 = w_1 w_2$
9. $\quad d.w_4 = d.w_1\, w_2 + w_1\, d.w_2$
10. $w_5 = w_3 w_4$
11. $\quad d.w_5 = d.w_3\, w_4 + w_3\, d.w_4$
12. $w_6 = w_5^2$
13. $\quad d.w_6 = 2w_5\, d.w_5$
14. $w_7 = \frac{w_1}{w_2 w_3}$
15. $\quad d.w_7 = \frac{d.w_1}{w_2 w_3} - \frac{w_1\, d.w_2}{w_2^2 w_3} - \frac{w_1\, d.w_3}{w_2 w_3^2}$
16. $w_8 = \sin w_7$
17. $\quad d.w_8 = \cos w_7\, d.w_7$
18. $w_9 = w_4 + w_6 + w_8$
19. $\quad d.w_9 = d.w_4 + d.w_6 + d.w_8$
20. $w_{10} = w_9^3$
21. $\quad d.w_{10} = 3w_9^2\, d.w_9$
22. $f = w_{10}$
23. $\quad d.f = d.w_{10}$
24. Output f and $d.f$

As usual, all the symbols represent real numbers. This works just like differentiating the Banana in forward mode, but it is a bit more complicated. Calculating the full gradient ∇f at some point (x, y, z) requires three calls of the routine, with, respectively, $(d.x, d.y, d.z) = (1, 0, 0)$, $(d.x, d.y, d.z) = (0, 1, 0)$ and $(d.x, d.y, d.z) = (0, 0, 1)$. Calculation of a single directional derivative requires only one call.

20.2.5 The Reverse Mode of AD: the Recipe

The reverse mode is a more efficient way of calculating the gradient of a scalar function of more than one input. I have nowhere seen an adequate explanation of reverse mode, which is why I am writing it up here in detail, after eventually figuring it out. It works easily (once you know how and why) with the Banana.

In reverse mode, we first calculate f by "running the code forward", i.e. by running it in the usual way from top to bottom. Then we calculate derivatives by stepping backwards through the code — something that probably means nothing until you see an example. We are calculating *adjoint variables* that are marked by an overbar. For example, we can have $\overline{w}_4$, which we define to mean $\partial f/\partial w_4$, and generally $\overline{w}_i \equiv \partial f/\partial w_i$. The **main rule** is that, every time we see a statement like this:

$$w_q = g(w_m, w_n, w_o, w_p, \dots), \tag{20.4}$$

we define these adjoint variables:

$$\overline{w}_m = \overline{w}_m + \frac{\partial w_q}{\partial w_m}\overline{w}_q, \tag{20.5a}$$

$$\overline{w}_n = \overline{w}_n + \frac{\partial w_q}{\partial w_n}\overline{w}_q, \tag{20.5b}$$

$$\overline{w}_o = \overline{w}_o + \frac{\partial w_q}{\partial w_o}\overline{w}_q, \tag{20.5c}$$

$$\overline{w}_p = \overline{w}_p + \frac{\partial w_q}{\partial w_p}\overline{w}_q, \tag{20.5d}$$

$$\vdots \tag{20.5e}$$

Note carefully how the overbars and subscripts are arranged in equations (20.4) and (20.5).

This works because in every statement like (20.4), the right-hand side variables will have been defined already; and in the adjoint statements like (20.5), $\overline{w}_q$ will have been defined already because we are stepping through the code backwards.

20.2.6 Reverse Mode: How it Works (Rosenbrock's Banana Revisited)

The pseudocode begins with the usual definition of the Banana, then goes back through the code, defining the adjoint variables as in the rules above. The ordinary code ends with line 8 and the AD code begins with line 9. I have replaced the partial derivatives that arise in the main rule, with their symbolic values. For example, from line 6 we have $\partial w_5/\partial w_4 = 200w_4$, which comes up in line 15. Lines 9–11 initialise the adjoint variables in a way that is explained below the code.

1. Input x and y
2. $w_1 = x$
3. $w_2 = y$
4. $w_3 = (1 - w_1)^2$
5. $w_4 = w_2 - w_1^2$
6. $w_5 = 100w_4^2$
7. $w_6 = w_3 + w_5$
8. $f = w_6$
9. $\overline{f} = 1$
10. $\overline{x} = \overline{y} = 0$
11. $\overline{w}_\alpha = 0, \quad \forall\alpha$
12. $\overline{w}_6 = \overline{w}_6 + (\partial f/\partial w_6)\overline{f} = \overline{f}$ (from line 8)
13. $\overline{w}_3 = \overline{w}_3 + (\partial w_6/\partial w_3)\overline{w}_6 = \overline{w}_6$ (from line 7)
14. $\overline{w}_5 = \overline{w}_5 + (\partial w_6/\partial w_5)\overline{w}_6 = \overline{w}_6$ (from line 7)
15. $\overline{w}_4 = \overline{w}_4 + (\partial w_5/\partial w_4)\overline{w}_5 = 200w_4\overline{w}_5$ (from line 6)
16. $\overline{w}_2 = \overline{w}_2 + (\partial w_4/\partial w_2)\overline{w}_4 = \overline{w}_4$ (from line 5)
17. $\overline{w}_1 = \overline{w}_1 + (\partial w_4/\partial w_1)\overline{w}_4 = -2w_1\overline{w}_4$ (from line 5)
18. $\overline{w}_1 = \overline{w}_1 + (\partial w_3/\partial w_1)\overline{w}_3 = -2w_1\overline{w}_4 - 2(1 - w_1)\overline{w}_3$ (from line 4)
19. $\overline{y} = \overline{y} + (\partial w_2/\partial y)\overline{w}_2 = \overline{w}_2$ (from line 3)

20. $\overline{x} = \overline{x} + (\partial w_1/\partial x)\overline{w}_1 = \overline{w}_1$ (from line 2)
21. Output f, $\overline{x}$ and $\overline{y}$
22. Note: $\nabla f = (\overline{x}, \overline{y})$

Remember that all these symbols and operations on them represent operations on real numbers, because they represent variables' names and their interaction in a language such as Fortran.

I now assert that $\overline{x} = \partial f/\partial x$ and $\overline{y} = \partial f/\partial y$, and explain why in §20.2.7.

Let us check the validity of this statement for the Banana example, before trying to prove it. By pencil-and-paper calculus, we can show that:

$$\frac{\partial f(x,y)}{\partial x} = 2(x-1) - 400x(y-x^2), \tag{20.6a}$$
$$\frac{\partial f(x,y)}{\partial y} = 200(y-x^2). \tag{20.6b}$$

By making all the substitutions in the pseudocode *except for the initialisation of* $\overline{f}$, we find that:

$$\overline{x} = \left[2(x-1) - 400x(y-x^2)\right]\overline{f}, \tag{20.7a}$$
$$\overline{y} = \left[200(y-x^2)\right]\overline{f}. \tag{20.7b}$$

If we had initialised $\overline{f}$ to one, the equations (20.6) and (20.7) would agree, i.e. the method works. If we had not initialised $\overline{f}$ to one, these equations show that we would have had to set $\overline{f} = 1$ to get the right answers.

20.2.7 Reverse Mode: Why it Works (Rosenbrock's Banana Revisited)

Above, I have shown *how* it works. Next, I show *why*. First, observe that in line 9, $\overline{f}$ is initialised to one. This is because the recipe in §20.2.5 says that $\overline{f} = \partial\overline{f}/\partial\overline{f}$, which obviously is equal to one. I gave another reason just after equations (20.7). Also, in lines 10 and 11, the adjointed input variables and all the adjointed intermediate variables are initialised to zero. I will explain this below.

Consider $\overline{x}$ only for now. With the correct initialisation, $\overline{f} = 1$, we want to get from $\overline{w}_6 = \partial f/\partial w_6$ to $\overline{x} = \partial f/\partial x$. First, at line 12, we have by the chain rule:

$$\frac{\partial f}{\partial w_6} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial w_6}, \tag{20.8}$$

i.e. $\overline{w}_6 = \overline{f}\overline{w}_6 = \overline{f}$. Since $\overline{w}_6$ was initialised to zero, it does no harm to add it to the right-hand side, and this will be a useful tactic in more complicated statements below. So we get an expression for $\overline{w}_6$ that follows the main rule, equation (20.4) and equations (20.5).

At line 7 we see that w_6 depends on w_3 and w_6. So we can get expressions for $\partial f/\partial w_3$ and $\partial f/\partial w_5$ like this:

$$\frac{\partial f}{\partial w_3} = \frac{\partial f}{\partial w_6}\frac{\partial w_6}{\partial w_3} = \overline{w}_6\frac{\partial w_6}{\partial w_3} = \overline{w}_3, \tag{20.9}$$
$$\frac{\partial f}{\partial w_5} = \frac{\partial f}{\partial w_6}\frac{\partial w_6}{\partial w_5} = \overline{w}_6\frac{\partial w_6}{\partial w_5} = \overline{w}_5. \tag{20.10}$$

Again, since $\overline{w}_3$ and $\overline{w}_5$ were initialised to zero, we can add them to the right-hand sides and not change the results. We have now made $\overline{w}_3$ and $\overline{w}_5$ follow equation (20.4) and equations (20.5), and got a bit closer to $\overline{x}$ and $\overline{y}$.

At line 6, we can see how, using the same method, we get $\overline{w}_4$ from w_5 and $\overline{w}_5$.

Lines 5 and 4 introduce a complication. In line 5, the dependence of w_4 on w_2 is straightforward, since this is the first (going backwards!) occurrence of w_2, so calculation of $\overline{w}_2$ is not problematic. But w_1 occurs on lines 5 and 4. Immediately after processing line 6, $\overline{w}_1$ still has its initial value of zero, so the following holds as usual at line 5:

$$\overline{w}_1 = \overline{w}_1 + \frac{\partial w_4}{\partial w_1}\overline{w}_4 \tag{20.11a}$$

$$= \frac{\partial w_4}{\partial w_1}\overline{w}_4. \tag{20.11b}$$

But then, at line 4,

$$\overline{w}_1 = \overline{w}_1 + \frac{\partial w_3}{\partial w_1}\overline{w}_3 \tag{20.12a}$$

$$= \frac{\partial w_4}{\partial w_1}\overline{w}_4 + \frac{\partial w_3}{\partial w_1}\overline{w}_3. \tag{20.12b}$$

That is,

$$\overline{w}_1 = \frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial w_4}\frac{\partial w_4}{\partial w_1} + \frac{\partial f}{\partial w_3}\frac{\partial w_3}{\partial w_1}. \tag{20.13}$$

This is why, in calculation of e.g. $\overline{w}_n$ in equations (20.5), the existing value is added to the partial derivative term. It allows f to depend on w_n via multiple routes through the dependency tree of the code. It also explains why $\overline{w}_n$ must be initialised to zero, for if it were not, then the first time it was encountered on a right-hand side (going backwards through the code), then it would not be calculated correctly.

We can then continue the above method until we get to $\overline{x}$, which, by reasoning identical to that just presented, is ultimately equal to $\partial f/\partial x$. Extension to $\overline{y} = \partial f/\partial y$ follows the same rules for the same reason.

20.2.8 Reverse Mode: Pearson's Squiggle Revisited

Here, I present pseudocode to calculate the gradient of the Squiggle in reverse mode, without explanation. It follows from the same rules as above, so it should be reasonably straightforward to understand. If not, then this paper is deficient, so please tell me how I have not made it understandable.

1. Input x, y and z.
2. $w_1 = x$
3. $w_2 = y$
4. $w_3 = z$
5. $w_4 = w_1 w_2$
6. $w_5 = w_3 w_4$
7. $w_6 = w_5^2$
8. $w_7 = \frac{w_1}{w_2 w_3}$
9. $w_8 = \sin w_7$
10. $w_9 = w_4 + w_6 + w_8 = \overline{w}_9$
11. $w_{10} = w_9^3$
12. $f = w_{10}$
13. $\overline{f} = 1$

14. $\overline{x} = \overline{y} = \overline{z} = 0$
15. $\overline{w}_\alpha = 0, \quad \forall \alpha$
16. $\overline{w}_1 0 = \overline{f}$
17. $\overline{w}_9 = 3w_9^2 \overline{w}_1 0$
18. $\overline{w}_4 = \overline{w}_6 = \overline{w}_8$
19. $\overline{w}_7 = \cos w_7 \, \overline{w} 8$
20. $\overline{w}_1 = \frac{\overline{w}_7}{w_2 w_3}$
21. $\overline{w}_2 = \frac{-\overline{w}_7 w_1}{w_2^2 w_3}$
22. $\overline{w}_3 = \frac{-\overline{w}_7 w_1}{w_2 w_3^2}$
23. $\overline{w}_5 = 2w_5 \overline{w}_6$
24. $\overline{w}_3 = \overline{w}_3 + w_4 \overline{w}_5$
25. $\overline{w}_4 = \overline{w}_4 + w_3 \overline{w}_5$
26. $\overline{w}_1 = \overline{w}_1 + w_2 \overline{w}_4$
27. $\overline{w}_2 = \overline{w}_2 + w_1 \overline{w}_4$
28. $\overline{z} = \overline{w}_3$
29. $\overline{y} = \overline{w}_2$
30. $\overline{x} = \overline{w}_1$
31. Output f, $\overline{x}$, $\overline{y}$ and $\overline{z}$
32. Note: $\nabla f = (\overline{x}, \overline{y}, \overline{z})$

20.2.9 Nabla Forward Mode

Here is another way of doing AD, using the chain rule on the gradient operator, $\nabla = (\partial/\partial x, \partial/\partial y, \partial/\partial z)$. I demonstrate only with the Banana, in which case we can leave out $\partial/\partial z$.

1. Input x and y
2. $\quad \nabla x = (1, 0)$
3. $\quad \nabla y = (0, 1)$
4. $w_1 = x$
5. $\quad \nabla w_1 = (\partial w_1/\partial x) \nabla x$
6. $w_2 = y$
7. $\quad \nabla w_2 = (\partial w_2/\partial y) \nabla y$
8. $w_3 = (1 - w_1)^2$
9. $\quad \nabla w_3 = (\partial w_3/\partial w_1) \nabla w_1$
10. $w_4 = w_2 - w_1^2$
11. $\quad \nabla w_4 = (\partial w_4/\partial w_1) \nabla w_1 + (\partial w_4/\partial w_2) \nabla w_2$
12. $w_5 = 100 w_4^2$
13. $\quad \nabla w_5 = (\partial w_5/\partial w_4) \nabla w_4$
14. $w_6 = w_3 + w_5$

15. $\nabla w_6 = (\partial w_6/\partial w_3)\nabla w_3 + (\partial w_6/\partial w_5)\nabla w_5$
16. $f = w_6$
17. $\nabla f = \nabla w_6$
18. Output f, ∇f

Observe that, at every derivative step, everything on the right-hand side is already available, because it has already been calculated — just as in the simpler forward mode of §20.2.2. At the end, we get the gradient of the Banana with respect to the two input variables.

In fact, this method is just the same as running through the forward mode code twice, once for $\partial/\partial x$ and again for $\partial/\partial y$. It is not significantly more efficient than that, because it contains about the same number of floating point operations.

Directional derivatives can be calculated outside the routine.

20.2.10 Remarks

The simple forward mode to calculate directional derivatives is easy to understand, by virtue of its working step-by-step with the original code, using the chain rule as taught to students. It is best when you have a lot of output variables but few input variables. By "best", I mean computationally cheapest.

If you want the full gradient of a scalar, you can use the nabla forward mode. Although I figured it out independently, it is not new. Something that I think is the same thing is covered in [Gri00].

The other way to get the full gradient is to use reverse mode. This is a bit hard to get, because it does not immediately flow from one's intuition, and I have, despite looking at many explanations, not found one that makes any sense to me. [Gri00] says, "To some casual observers, the reverse mode always remains a little mysterious". So I figured it out and presented it in — I hope — enough detail here. Reverse mode has the merit of working with scalars only, even though it gives the multidimensional gradient at the end. It does this by calculating the dimensions separately, e.g. $\overline{x}$ and $\overline{y}$ are calculated separately. It is best when you have few output variables (e.g. one cost function) and many input variables (e.g. a bunch of parameters and/or initial values).

According to [Gri00], forward mode and reverse mode codes each cost only a small scalar times their undifferentiated codes' costs, even if — for example — a reverse mode AD calculates a gradient of a function of millions of input variables. I have seen this stated elsewhere, too, but I do not know of a proof that I can both understand and cite.

20.3 Operator Overloading and AD in Fortran

20.3.1 Introduction

In the source transformation method, we take an algorithm and insert statements that explicitly calculate differential quantities. For example, in forward mode AD, we might have the following piece of Fortran90 code to calculate the Rosenbrock banana:

```
! banana = (1-x)**2 + 100(y - x**2)**2
temp1 = 1 - x
temp2 = temp1 ** 2
temp3 = y - x**2
temp4 = 100 * temp3**2
banana = temp2 + temp4
```

To calculate the differential of the banana, we transform the code, either automatically or by hand (or both), like this:

```
! banana = (1-x)**2 + 100(y - x**2)**2
temp1 = 1 - x
d.temp1 = -d.x                    ! Differential of temp1
temp2 = temp1 ** 2
d.temp2 = 2 * temp1 * d.temp1     ! Differential of temp2
temp3 = y - x**2
d.temp3 = d.y - 2 * x * d.x       ! Differential of temp3
temp4 = 100 * temp3**2
d.temp4 = 200 * temp3 * d.temp3   ! Differential of temp4
banana = temp2 + temp4
d.banana = d.temp2 + d.temp4      ! Differential of banana
```

We can calculate ∂banana$/\partial x$ by setting d.x $\leftarrow 1$ and d.y $\leftarrow 0$. We can calculate ∂banana$/\partial y$ by setting d.x $\leftarrow 0$ and d.y $\leftarrow 1$. We can calculate a directional derivative of the banana by setting d.x $\leftarrow \xi$ and d.y $\leftarrow \zeta$, with $\xi^2 + \zeta^2 = 1$.

Another method involves hiding the differential quantity in an augmented variable, and redefining Fortran's mathematical operators so they can handle — and differentiate properly — these augmented variables. Fortran90 and its descendants can do this fairly easily, but Fortran77 and its ancestors probably could not.

It is a bit fiddly, and involves some of the weirdness of modern Fortran, but once you have figured it out then it is not too bad. The basic idea is put all the definitions in a module. A working example is in §E. A working program that demonstrates differentiation of the banana is in §F, and the output from running it is in §G. I will go through these bit-by-bit as well.

20.3.2 Details

The whole thing revolves around the new kind of variable — known as a "derived type". The following part of the module defines a derived type called `diffy`:

```
type diffy
   real     :: value , differential
end type diffy
```

A `diffy` variable is a container, holding two variables. If `x` is a `diffy` variable, then it contains two `real` variables called `x%value` and `x%differential`. As the names suggest, the former is the value of the variable, and the latter is its differential.

Fortran does not have an intrinsic routine for adding `diffy` variables. We could write one, called, for example, `add_two_diffy`, but then we would have to replace every addition in the code with a call to `add_two_diffy`. And so on with every other mathematical operation. We would have a huge code-transformation job to do, and we would have to re-do it every time the code changed.

A better way is to redefine the `+` operator so it knows what to do when given a pair of `diffy` variables to add. The following fragments of the module do this:

```
interface operator (+)
   module procedure diffy_plus_diffy
end interface
```

```
function diffy_plus_diffy(x1 , x2) result(sum)
   type(diffy) , intent(in)      :: x1 , x2
   type(diffy)                   :: sum
   sum%value = x1%value + x2%value
   sum%differential = x1%differential + x2%differential
end function diffy_plus_diffy
```

The first part informs the compiler that the `+` operator can be augmented — "overloaded" — with the function called `diffy_plus_diffy`. The second part defines that function. As you can see, it takes in two `diffy` variables and returns their sum. The values add, and so do the differentials, because $d(x + y) = dx + dy$. The important thing to understand here is that **the compiler knows that if the `+` is between two `diffy` variables, instead of two "ordinary" Fortran variables, it will send them to the `diffy_plus_diffy` function and add them as `diffy` variables.** What would happen if we tried to add an ordinary variable to a `diffy` variable? This situation is not defined, and a compile error would happen. If we want to do this kind of addition, we need another function called something like `diffy_plus_real`, and another called `real_plus_diffy` ... and so on.

The program `main.f90` in §F demonstrates addition of `diffy` variables:

```
x = diffy(10.0 , 0.1)
y = diffy(3.141 , 0.7654321)
z = x + y
print *, "diffy plus diffy:"
print *, "(+) Result should be " ,                          &
      (10.0 + 3.141) ,                                      &
      " , " ,                                               &
      0.1 + 0.7654321
print *, "(+) Result is        " , z
```

The output file shows that this works as it should:

```
 diffy plus diffy:
 (+) Result should be    13.14100      ,   0.8654321
 (+) Result is           13.14100      0.8654321
```

We need another overload for the `-` operator:

```
   interface operator (-)
      module procedure real_minus_diffy , diffy_minus_diffy , int_minus_diffy
   end interface
```

This says that the `-` operator is to be overloaded with three new functions, as listed. These are defined in the module, thus:

```
   function diffy_minus_diffy(x1 , x2) result(sub)
      type(diffy) , intent(in)      :: x1 , x2
      type(diffy)                   :: sub
      sub%value = x1%value - x2%value
      sub%differential = x1%differential - x2%differential
   end function diffy_minus_diffy
!-------------------------------------------------|
   function real_minus_diffy(r1 , x2) result(res)
      real , intent(in)             :: r1
      type(diffy) , intent(in)      :: x2
      type(diffy)                   :: res
      res%value = r1 - x2%value
      res%differential = - x2%differential
   end function real_minus_diffy
!-------------------------------------------------|
   function int_minus_diffy(i1 , x2) result(res)
      integer , intent(in)          :: i1
      type(diffy) , intent(in)      :: x2
      type(diffy)                   :: res
```

```
      res%value = i1 - x2%value
      res%differential = - x2%differential
   end function int_minus_diffy
```

Observe that the non-`diffy` variables involved in these functions do not contribute any differentials to the results.

There is not much more to say on operator overloading in the present context, except perhaps in the case of nonlinear combinations of variables, such as the `*` operator:

```
   interface operator (*)
      module procedure diffy_times_diffy , real_times_diffy , diffy_times_real
   end interface
```

```
   function diffy_times_diffy(x1 , x2) result(prod)
      type(diffy) , intent(in)      :: x1 , x2
      type(diffy)                   :: prod
      prod%value = x1%value * x2%value
      prod%differential = x1%value * x2%differential + x2%value * x1%differential
   end function diffy_times_diffy
```

This is so because $d(xy) = xdy + ydx$.

A slight complexity arises in overloading an external function, such as our banana function `rose()`. We have:

```
   interface rose
      module procedure r_rose , d_rose
   end interface
```

This says that when `rose()` is encountered, then either `r_rose()` or `d_rose()` is called, depending on the kind of arguments that are supplied ("ordinary" or `diffy`). These two functions are:

```
   function r_rose(x , y) result(rosy)
      real , intent(in)             :: x , y
      real                          :: rosy
      rosy = (1 - x)**2 + 100.0 * ((y-x**2)**2)
   end function r_rose
!-------------------------------------------------------|
   function d_rose(x , y) result(rosy)
      type(diffy) , intent(in)      :: x , y
      type(diffy)                   :: rosy
      rosy = (1 - x)**2 + 100.0 * ((y-x**2)**2)
   end function d_rose
```

At first, it looks a bit strange that the functions are the same. But observe that the inputs and result in `d_rose()` are `diffy` variables. All the operators in the function $(1-x)^2 + 100(y-x^2)^2$ have already been overloaded in the module, so the compiler knows what to do. The main program tests this by comparison with a divided-differences approximation, and the output file shows that this works well.

Instead of this overloading of `rose`, we could have said:

```
   interface rose
      module procedure rose , d_rose
   end interface
```

This says that when `rose()` is encountered, then either `rose()` or `d_rose()` is called, depending on the kind of arguments that are supplied ("ordinary" or `diffy`). These two functions are:

```
   function rose(x , y) result(rosy)
      real , intent(in)             :: x , y
      real                          :: rosy
      rosy = (1 - x)**2 + 100.0 * ((y-x**2)**2)
   end function rose
!-------------------------------------------------------|
   function d_rose(x , y) result(rosy)
      type(diffy) , intent(in)      :: x , y
      type(diffy)                   :: rosy
      rosy = (1 - x)**2 + 100.0 * ((y-x**2)**2)
   end function d_rose
```

That is, `rose(x_real , y_real)` would call the function `rose()`, which, now, is just the same as the old `r_rose()`. That is, we do not have to give the function a new name.

20.3.3 Practical Use

At first sight, it might appear that we need to make very few changes to the code, in order to make this work. These changes could be:

- Make a module for overloading of *all* math operators that are used in our program;
- Put a `USE` in each subprogram, pointing to our module;
- Put `IMPLICIT NONE` everywhere;
- Change all relevant `real` (or double precision, or whatever) variables to `diffy`;
- Write a routine that goes between inputting all the initial values and parameters and running the rest of the code, to convert the `real` input numbers to `diffy`.

Having done this, future code changes would not require major changes to the differentiation part of the code, because it is dealt with automatically by the overloaded operators. This contrasts with source transformation, which needs new differentiation code to be explicitly written when the value code is changed. I suspect, however, that operator overloading is not that easy in practice, for a large code.

20.4 The Complex Step Method

20.4.1 How and Why

This method is easy to understand and relatively easy to implement by hand. Modern references are [ST98] and [BCFH03].

As [ST98] points out, divided difference methods are far from ideal in numerically calculating derivatives. That paper discusses accuracy problems in the central difference formula. Another difficulty in the present context of data assimilation, is that divided difference methods need a lot of evaluations of the function that is to be differentiated.

As we have seen, algorithmic differentiation, a.k.a. automatic differentiation, is a way around these problems. Another way is the *complex step method*. It works like this. We want to differentiate a real function $f(\cdot)$ w.r.t. its single real scalar argument. Here, f can be represented by an arbitrarily large and loopy code. But let us instead give it a complex argument, $x + i\delta$, and make a Taylor series around x:

$$f(x + i\delta) = f(x) + i\delta f'(x) - \frac{\delta^2}{2!} f''(x) - \frac{\delta^3}{3!} i f^{(3)}(x) + O(\delta^4). \tag{20.14}$$

There are probably important considerations of analyticity and convergence here, but [ST98] is reassuring on these. Anyway, we can immediately say

$$\frac{\Im[f(x+i\delta)]}{\delta} = f'(x) + O(\delta^2). \tag{20.15}$$

The squiggly $\Im$ is LaTeX's version of the Im function. Observe two things: (i) this is second-order accurate in δ; and (ii) we do not have the trade-off between truncation error and cancellation error that damages the divided differences methods.

However, this does rely on the complex functions' behaving themselves in whatever programming language we are using. It should be OK in Fortran90, and it seems to work in R, as I have shown but not included here.

For a function of more than one variable, we can add a tiny imaginary number to the arguments one-at-a-time, each time keeping the others constant and real, to give a set of numerical partial derivatives. This is akin to working with a function of more than one argument in the forward source transformation method.

For functions that are not one-liners, but are made up of many steps (including subroutines, etc.), it all still works. We do not have to worry about chain rules for derivatives (which, in essence, is what AD is all about), because in the end we just have a complex number that comes from $x+i\delta$ through a complicated series of steps. Applying the $\Im/\delta$ operator to our final complex number gives the derivative of the function.

For functions that evolve in time, like $C_l(t)$, we can get the derivative at each time-step, because at each time-step we have a complex scalar that we can apply the $\Im/\delta$ operator to. Each time-step follows from the previous one, so it all works out nicely.

20.4.2 Remark(s)

I had not tried this before, because I was worried about the integrity of the complex library functions in Fortran90 and R. But when I tried it, certainly *did* work. Whether it *will* work in complicated codes (e.g. JULES) is a matter for testing. As AD needs testing anyway, in any practical use, this is not in itself a strong reason not to use the complex step method.

I dimly foresee problems that may arise when complex step is applied to iterative problems, which loop a number of times that is not known in advance. Real problems usually contain iterative sub-problems, e.g. to solve implicit equations or to solve matrix equations. But a similar caveat applies to the application of the other AD methods to iterative problems, and I suspect that wider knowledge of iterative AD will inform iterative complex-step.

21 Topics Not Covered

21.1 Introduction

As the title suggests, this paper is an introduction to data assimilation. There are many others. This one goes into an unusual amount of detail, which is something that I would have found useful a few years ago when learning about all this stuff. Especially confusing was the question of what, exactly, was in the model state vector. Following [Coh97a], I have therefore defined $\boldsymbol{x}^t$ and model error carefully. It also presents some simple, but instructive, examples of the plain KF and EnKF operating on the linear rotational system. It brings to the fore the statistical relationship between the EnKF and the fundamental plain KF. This paper also uses a carbon cycle model, courtesy of Matthew Williams at the University of Edinburgh, to demonstrate Var and the EnKF, and how things can go wrong. This is one reason I think it is especially useful for beginners. All this detail notwithstanding, the field of data assimilation is a huge one, and I have omitted most of it. But maybe the literature will be more accessible if you have this paper alongside. If you want another overview, I recommend Kalnay's book for depth and breadth [Kal03].

21.2 Some Subjects I Have Omitted

1. Large problems, such as operational NWP, need even more help than linearisation and incrementalisation. The problems must also be *preconditioned*. This is a large sub-topic in Var, as well as being an important part of numerical analysis generally. There are many methods of preconditioning. The method in use at the Met Office for NWP is known as transforming the control variables. This consists of finding a linear transformation of the physical variables of interest (velocity components, etc.) that generates new variables that are approximately statistically orthogonal to each other. Then further transformations break up the new variables geometrically. I suggest [Ban07] and references to and from it, to understand this topic.

2. A fundamentally different way of looking at Var arises from consideration of constrained variational (in the classical mathematical physics sense) problems. For example, you might want to find a minimum of some function of the observations, subject to the condition that the system obeys a certain differential equation (the model). A basis for estimating parameters can be established this way. See [WLNR07] and [EDS98] for more about this.

3. I have not said much about model bias. [Dd98] is good on this.

4. I have not said much about filter divergence and filter tuning. [BH97] and [AM05] cover these topics in readable ways.

5. Generally, I highly recommend [Kal03] if you want a broader knowledge of research into, and the practice of, data assimilation for continuous systems. That book covers all or most of the contents of this paper, albeit with less attention to detail; and the things I have listed in this list of omissions; and very much more, such as singular values, 3D-Var, optimal interpolation, questions of predictability, the method of representers, *et cetera*.

A Scalar and Vector Random Variables

A.1 Introduction

This section provides an overview of some manipulations on families of random variables. (I say "family" instead of "set" or "group", because this has nothing to do with set theory or group theory. I could have said "collection" instead of "family".) I do not think it is necessary to define a random variable in detail here. Suffice it to say that a scalar random variable X has a realisation x, i.e we denote random variables with capital letters and their realisations with lower case letters. X has probability density function $p_X(\cdot)$. It usually makes sense to say $p_X(x)$, but we could equally write $p_X(\xi)$ or, for that matter, $p_X(\clubsuit)$ — the important thing is the p_X, which shows that we are thinking about the probability density of the RV[6] X, for $X = x$ or $X = \xi$ or $X = \clubsuit$, etc. This lets us use dummy arguments in integrals of densities.

A.2 Scalar Random Variables

A quick reminder of results on pairs of random variables. We will call these X and Y. Their joint density is:

$$p_{XY}(x, y) = \text{probability}(X = x \text{ and } Y = y). \tag{A.1}$$

[6] I use "RV" to mean "random variable", but it may also mean "random vector", according to context.

Their marginal densities are:

$$p_X(x) = \int_{-\infty}^{\infty} p_{XY}(x,y)\,dy, \tag{A.2a}$$

$$p_Y(y) = \int_{-\infty}^{\infty} p_{XY}(x,y)\,dx. \tag{A.2b}$$

The means or expectations of X and Y are:

$$\langle X \rangle = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x\, p_{XY}(x,y)\,dx\,dy, \tag{A.3a}$$

$$\langle Y \rangle = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} y\, p_{XY}(x,y)\,dx\,dy. \tag{A.3b}$$

The expectation of any function $f(X,Y)$ is:

$$\langle f(X,Y) \rangle = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\, p_{XY}(x,y)\,dx\,dy. \tag{A.4}$$

The conditional density of Y given X and of X given Y:

$$p_{Y|X}(y|x) = \frac{p_{XY}(x,y)}{p_X(x)}, \tag{A.5a}$$

$$p_{X|Y}(x|y) = \frac{p_{XY}(x,y)}{p_Y(y)}, \tag{A.5b}$$

whence Bayes' Rule:

$$p_{Y|X}(y|x) = \frac{p_{X|Y}(x|y)p_Y(y)}{p_X(x)}. \tag{A.6}$$

The RVs are said to be *uncorrelated* if and only if:

$$\langle XY \rangle = \langle X \rangle \langle Y \rangle. \tag{A.7}$$

The conditional expectation of $f(X,Y)$ is defined like this:

$$\langle f(X,Y)|Y \rangle = \int_{-\infty}^{\infty} f(x,y) p_{X|Y}(x|y)\,dx. \tag{A.8}$$

Multivariate normal (MVN) random variables, also known as multivariate Gaussian random variables, play a large part in the theory. The univariate normal, or Gaussian, RV is defined by the following density:

$$N(\mu,\sigma^2) \sim p_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{A.9}$$

where μ is the mean and σ is the standard deviation. The MVN density is then given by equation (A.19), below.

A.3 Vector Random Variables

Now we extend our family of random variables to a larger number, $\nu = n + m$. We call the RVs X_i, with $1 \le i \le \nu$. We can then write the joint density $p_{X_1 X_2 \ldots X_\nu}(x_1, x_2, \ldots, x_\nu)$. But this is cumbersome, and even more so when we try to write equations for marginal densities and all the rest. So instead we assemble families of RVs into vectors, which enables a concise and fairly transparent notation.

Now let $\boldsymbol{X}$ be an $(n \times 1)$ vector of random variables. This means merely that we have n RVs that are jointly distributed as $p_{X_1 X_2 \ldots X_n}$. No relation between any sub-family of them is implied. Any or all of them could be dependent or independent — putting them into a single vector implies nothing other than that we are thinking about them together. Now instead of writing $p_{X_1 X_2 \ldots X_n}$, we write $p_{\boldsymbol{X}}(\boldsymbol{x})$.

The vector notation can be used to concisely represent $\langle \boldsymbol{X} \rangle$ as follows.

$$\langle \boldsymbol{X} \rangle = \begin{bmatrix} \langle X_1 \rangle \\ \langle X_2 \rangle \\ \vdots \\ \langle X_n \rangle \end{bmatrix} = \begin{bmatrix} \int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} x_1 p_{\boldsymbol{X}}(\boldsymbol{x})\, dx_1 \ \ldots \ dx_n \\ \int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} x_2 p_{\boldsymbol{X}}(\boldsymbol{x})\, dx_1 \ \ldots \ dx_n \\ \vdots \\ \int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} x_n p_{\boldsymbol{X}}(\boldsymbol{x})\, dx_1 \ \ldots \ dx_n \end{bmatrix} \tag{A.10a}$$

$$= \int \boldsymbol{x}\, p_{\boldsymbol{X}}(\boldsymbol{x})\, d\boldsymbol{x}. \tag{A.10b}$$

The first equality just defines $\langle \boldsymbol{X} \rangle$ as a vector containing the expectations of the individual RVs that populate $\boldsymbol{X}$. The second equality just fills in the definitions of these expectations, in terms of the family of RVs. The third equality defines the concise notation $\int \boldsymbol{x} p_{\boldsymbol{X}}(\boldsymbol{x})\, d\boldsymbol{x}$.

Now for the vector extensions of the rest of equations (A.1)–(A.8), we need another random vector $\boldsymbol{Y} \sim (m \times 1)$. We are now thinking about $\nu = n + m$ RVs again, but in two separate families arranged into vectors. We can, at the same time, think about all ν RVs as a single vector $\boldsymbol{Z} = (\boldsymbol{X}^T, \boldsymbol{Y}^T)^T$, i.e. the vector formed by joining $\boldsymbol{Y}$ to the bottom of $\boldsymbol{X}$.

Then the joint density is:

$$p_{\boldsymbol{Z}}(\boldsymbol{z}) = p_{\boldsymbol{XY}}(\boldsymbol{x}, \boldsymbol{y}). \tag{A.11}$$

The marginal densities are:

$$p_{\boldsymbol{X}}(\boldsymbol{x}) = \int p_{\boldsymbol{Z}}(\boldsymbol{z})\, d\boldsymbol{y} = \int p_{\boldsymbol{XY}}(\boldsymbol{x}, \boldsymbol{y})\, d\boldsymbol{y}, \tag{A.12a}$$

$$p_{\boldsymbol{Y}}(\boldsymbol{y}) = \int p_{\boldsymbol{Z}}(\boldsymbol{z})\, d\boldsymbol{x} = \int p_{\boldsymbol{XY}}(\boldsymbol{x}, \boldsymbol{y})\, d\boldsymbol{x}. \tag{A.12b}$$

The means or expectations of X and Y are:

$$\langle \boldsymbol{X} \rangle = \int \boldsymbol{x}\, p_{\boldsymbol{Z}}(\boldsymbol{z})\, d\boldsymbol{z} = \int \boldsymbol{x}\, p_{\boldsymbol{XY}}(\boldsymbol{x}, \boldsymbol{y})\, d\boldsymbol{x}\, d\boldsymbol{y}, \tag{A.13a}$$

$$\langle \boldsymbol{Y} \rangle = \int \boldsymbol{y}\, p_{\boldsymbol{Z}}(\boldsymbol{z})\, d\boldsymbol{z} = \int \boldsymbol{y}\, p_{\boldsymbol{XY}}(\boldsymbol{x}, \boldsymbol{y})\, d\boldsymbol{x}\, d\boldsymbol{y}. \tag{A.13b}$$

The expectation of any function $f(\boldsymbol{Z}) = f(\boldsymbol{X}, \boldsymbol{Y})$ is:

$$\langle f(\boldsymbol{X}, \boldsymbol{Y}) \rangle = \int f(\boldsymbol{z})\, p_{\boldsymbol{Z}}(\boldsymbol{z})\, d\boldsymbol{z} = \int f(\boldsymbol{x}, \boldsymbol{y})\, p_{\boldsymbol{XY}}(\boldsymbol{x}, \boldsymbol{y})\, d\boldsymbol{x}\, d\boldsymbol{y}. \tag{A.14}$$

The RVs are said to be *uncorrelated* if and only if:

$$\langle \boldsymbol{X}\boldsymbol{Y}^T \rangle = \langle \boldsymbol{X} \rangle \langle \boldsymbol{Y}^T \rangle. \tag{A.15}$$

The conditional density of $\boldsymbol{Y}$ given $\boldsymbol{X}$ and of $\boldsymbol{X}$ given $\boldsymbol{Y}$:

$$p_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{y}|\boldsymbol{x}) = \frac{p_{\boldsymbol{Z}}(\boldsymbol{z})}{p_{\boldsymbol{X}}(\boldsymbol{x})} = \frac{p_{\boldsymbol{XY}}(\boldsymbol{x}, \boldsymbol{y})}{p_{\boldsymbol{X}}(\boldsymbol{x})}, \tag{A.16a}$$

$$p_{\boldsymbol{X}|\boldsymbol{Y}}(\boldsymbol{x}|\boldsymbol{y}) = \frac{p_{\boldsymbol{Z}}(\boldsymbol{z})}{p_{\boldsymbol{Y}}(\boldsymbol{y})} = \frac{p_{\boldsymbol{XY}}(\boldsymbol{x}, \boldsymbol{y})}{p_{\boldsymbol{Y}}(\boldsymbol{y})}, \tag{A.16b}$$

whence Bayes' Rule:

$$p_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{y}|\boldsymbol{x}) = \frac{p_{\boldsymbol{X}|\boldsymbol{Y}}(\boldsymbol{x}|\boldsymbol{y})p_{\boldsymbol{Y}}(\boldsymbol{y})}{p_{\boldsymbol{X}}(\boldsymbol{x})}. \tag{A.17}$$

The conditional expectation of $f(\boldsymbol{X}, \boldsymbol{Y})$ becomes:

$$\langle f(\boldsymbol{Z})|\boldsymbol{Y}\rangle = \langle f(\boldsymbol{X}, \boldsymbol{Y})|\boldsymbol{Y}\rangle = \int f(\boldsymbol{x}, \boldsymbol{y})\, p_{\boldsymbol{X}|\boldsymbol{Y}}(\boldsymbol{x}|\boldsymbol{y})\, d\boldsymbol{x}. \tag{A.18}$$

The MVN density is:

$$N(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \sim p_{\boldsymbol{X}}(\boldsymbol{x}) = \frac{1}{\sqrt{|\boldsymbol{\Sigma}|}(2\pi)^{N/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right), \tag{A.19}$$

where $\boldsymbol{x}$ is a random vector, $\boldsymbol{\mu}$ is its mean, and $\boldsymbol{\Sigma}$ is its covariance matrix.

A.4 Covariances

Another useful bit of notation is that for the covariance of a random vector:

$$\mathrm{cov}(\boldsymbol{X}) = \langle \boldsymbol{X} - \langle \boldsymbol{X}\rangle\rangle\langle \boldsymbol{X} - \langle \boldsymbol{X}\rangle\rangle^T = \begin{bmatrix} \mathrm{var}(X_1) & \mathrm{cov}(X_1, X_2) & \dots & \mathrm{cov}(X_1, X_n) \\ \mathrm{cov}(X_2, X_1) & \mathrm{var}(X_2) & \dots & \mathrm{cov}(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{cov}(X_n, X_1) & \mathrm{cov}(X_n, X_2) & \dots & \mathrm{var}(X_n) \end{bmatrix}. \tag{A.20}$$

Then the covariance of two random vectors is:

$$\mathrm{cov}(\boldsymbol{X}, \boldsymbol{Y}) = \langle \boldsymbol{X} - \langle \boldsymbol{X}\rangle\rangle\langle \boldsymbol{Y} - \langle \boldsymbol{Y}\rangle\rangle^T = \begin{bmatrix} \mathrm{cov}(X_1, Y_1) & \mathrm{cov}(X_1, Y_2) & \dots & \mathrm{cov}(X_1, Y_m) \\ \mathrm{cov}(X_2, Y_1) & \mathrm{cov}(X_2, Y_2) & \dots & \mathrm{cov}(X_2, Y_m) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{cov}(X_n, Y_1) & \mathrm{cov}(X_n, Y_2) & \dots & \mathrm{cov}(X_n, Y_m) \end{bmatrix}. \tag{A.21}$$

A.5 The Chain Rule

The chain rule for conditional expectations of scalars is this:

$$\langle\langle f(X, Y)|Y\rangle\rangle = \langle f(X, Y)\rangle. \tag{A.22}$$

But what does this mean? In equation (A.22), $\langle f(X, Y)|Y\rangle$ is a function of Y only, because the dependence on X is integrated away in (A.8). Let us call this $g(Y)$. So we can then calculate the unconditional expectation of this new function $g(Y)$, as in (A.4) (with $g(Y)$ in place of $f(X, Y)$), to give a single number. Equation (A.22) asserts that this single number is equal to the unconditional expectation of $f(X, Y)$, as calculated by (A.4).

Here is proof:

$$g(Y) = \langle f(X, Y)|Y\rangle = \int_{-\infty}^{\infty} f(x, y)\, p_{X|Y}(x|y)\, dx \tag{A.23a}$$

$$= \int_{-\infty}^{\infty} f(x, y)\frac{p_{XY}(x, y)}{p_Y(y)}\, dx. \tag{A.23b}$$

Then

$$\langle g(Y)\rangle = \int_{-\infty}^{\infty} g(y)\, p_Y(y)\, dy \tag{A.24a}$$

$$= \int_{-\infty}^{\infty} \left(\int_{-\infty}^{\infty} f(x,y) \frac{p_{XY}(x,y)}{p_Y(y)}\, dx \right) p_Y(y)\, dy \tag{A.24b}$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\, p_{XY}(x,y)\, dx\, dy \tag{A.24c}$$

$$= \langle f(X,Y)\rangle. \tag{A.24d}$$

For random vectors, the proof is similar. We get:

$$\langle\langle f(\boldsymbol{X},\boldsymbol{Y})|\boldsymbol{Y}\rangle\rangle = \langle f(\boldsymbol{X},\boldsymbol{Y})\rangle. \tag{A.25}$$

A.6 Useful Results on MVN and Block Matrices

Matrices are upper case bold, like $\boldsymbol{A}$. Vectors are lower-case bold, like $\boldsymbol{y}$. A partitioned matrix (a.k.a. block matrix) is like this:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{1,1} & \boldsymbol{A}_{1,2} \\ \boldsymbol{A}_{2,1} & \boldsymbol{A}_{2,2} \end{bmatrix}, \tag{A.26}$$

where $\boldsymbol{A}_{i,j}$ is a matrix, which may be square or not, and may contain only one element or more than one.

A.6.1 The Inverse of a Block Matrix

You can find two versions of this result in [PTVF92], but not a derivation.

The job is to find the inverse of a block matrix. Let us call the inverse $\boldsymbol{B}$. Then by definition,

$$\boldsymbol{B} = \boldsymbol{A}^{-1} = \begin{bmatrix} \boldsymbol{A}_{1,1} & \boldsymbol{A}_{1,2} \\ \boldsymbol{A}_{2,1} & \boldsymbol{A}_{2,2} \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{B}_{1,1} & \boldsymbol{B}_{1,2} \\ \boldsymbol{B}_{2,1} & \boldsymbol{B}_{2,2} \end{bmatrix}. \tag{A.27}$$

It is assumed that $\boldsymbol{A}_{1,1}$ and $\boldsymbol{A}_{2,2}$ are square, and similarly for $\boldsymbol{B}$. Using an obvious notation for the partitioned identity matrix,

$$\begin{bmatrix} \boldsymbol{A}_{1,1} & \boldsymbol{A}_{1,2} \\ \boldsymbol{A}_{2,1} & \boldsymbol{A}_{2,2} \end{bmatrix} \begin{bmatrix} \boldsymbol{B}_{1,1} & \boldsymbol{B}_{1,2} \\ \boldsymbol{B}_{2,1} & \boldsymbol{B}_{2,2} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_{1,1} & \boldsymbol{I}_{1,2} \\ \boldsymbol{I}_{2,1} & \boldsymbol{I}_{2,2} \end{bmatrix}. \tag{A.28}$$

Writing this out in full gives:

$$\boldsymbol{A}_{1,1}\boldsymbol{B}_{1,1} + \boldsymbol{A}_{1,2}\boldsymbol{B}_{2,1} = \boldsymbol{I} \qquad \boldsymbol{A}_{1,1}\boldsymbol{B}_{1,2} + \boldsymbol{A}_{1,2}\boldsymbol{B}_{2,2} = \boldsymbol{0} \tag{A.29}$$

$$\boldsymbol{A}_{2,1}\boldsymbol{B}_{1,1} + \boldsymbol{A}_{2,2}\boldsymbol{B}_{2,1} = \boldsymbol{0} \qquad \boldsymbol{A}_{2,1}\boldsymbol{B}_{1,2} + \boldsymbol{A}_{2,2}\boldsymbol{B}_{2,2} = \boldsymbol{I} \tag{A.30}$$

First we eliminate $\boldsymbol{B}_{1,2}$ between equations (A.29-RHS) and (A.30-RHS). We easily find that:

$$\boldsymbol{B}_{1,2} = -\boldsymbol{A}_{1,1}^{-1}\boldsymbol{A}_{1,2}\boldsymbol{B}_{2,2}, \tag{A.31}$$

so:

$$\boldsymbol{I} = -\boldsymbol{A}_{2,1}\boldsymbol{A}_{1,1}^{-1}\boldsymbol{A}_{1,2}\boldsymbol{B}_{2,2} + \boldsymbol{A}_{2,2}\boldsymbol{B}_{2,2} \tag{A.32}$$

$$= (\boldsymbol{A}_{2,2} - \boldsymbol{A}_{2,1}\boldsymbol{A}_{1,1}^{-1}\boldsymbol{A}_{1,2})\boldsymbol{B}_{2,2}, \tag{A.33}$$

and we thus get:

$$\boldsymbol{B}_{2,2} = \left(\boldsymbol{A}^{-1}\right)_{2,2} = (\boldsymbol{A}_{2,2} - \boldsymbol{A}_{2,1}\boldsymbol{A}_{1,1}^{-1}\boldsymbol{A}_{1,2})^{-1}. \tag{A.34}$$

Then equations (A.31) and (A.34) immediately give us $\boldsymbol{B}_{1,2}$:

$$\boldsymbol{B}_{1,2} = -\boldsymbol{A}_{1,1}^{-1}\boldsymbol{A}_{1,2}(\boldsymbol{A}_{2,2} - \boldsymbol{A}_{2,1}\boldsymbol{A}_{1,1}^{-1}\boldsymbol{A}_{1,2})^{-1}. \tag{A.35}$$

We could have found a different but equivalent result by first swapping equation (A.30-RHS) around to get an expression for $\boldsymbol{B}_{1,2}$ then plugging this into equation (A.29-RHS). And two other forms could have been found by eliminating $\boldsymbol{B}_{2,2}$ from one or the other of the two equations also. These are not shown here — you can do it yourself now you know how.

Next we swap equation (A.29-LHS) around to get:

$$\boldsymbol{B}_{1,1} = \boldsymbol{A}_{1,1}^{-1}(\boldsymbol{I} - \boldsymbol{A}_{1,2}\boldsymbol{B}_{2,1}). \tag{A.36}$$

This plugs into equation (A.30-LHS) to give:

$$\begin{aligned}
\boldsymbol{0} &= \boldsymbol{A}_{2,1}\boldsymbol{A}_{1,1}^{-1} - \boldsymbol{A}_{2,1}\boldsymbol{A}_{1,1}^{-1}\boldsymbol{A}_{1,2}\boldsymbol{B}_{2,1} + \boldsymbol{A}_{2,2}\boldsymbol{B}_{2,1} && \text{(A.37a)}\\
&= \boldsymbol{A}_{2,1}\boldsymbol{A}_{1,1}^{-1} + (\boldsymbol{A}_{2,2} - \boldsymbol{A}_{2,1}\boldsymbol{A}_{1,1}^{-1}\boldsymbol{A}_{1,2})\boldsymbol{B}_{2,1} && \text{(A.37b)}\\
&= \boldsymbol{A}_{2,1}\boldsymbol{A}_{1,1}^{-1} + \boldsymbol{B}_{2,2}^{-1}\boldsymbol{B}_{2,1}, && \text{(A.37c)}
\end{aligned}$$

so:

$$\boldsymbol{B}_{2,1} = -\boldsymbol{B}_{2,2}\boldsymbol{A}_{2,1}\boldsymbol{A}_{1,1}^{-1}. \tag{A.38}$$

which also gives us $\boldsymbol{B}_{1,1}$ through equation (A.36). Again, we could have done this in three other ways. We could also have worked one row at a time instead of one column, but I do not know if this would give the same expressions or yet more equivalent forms.

Another form of $\boldsymbol{B}_{2,1}$ and $\boldsymbol{B}_{1,1}$ will be useful in the discussion of MVN below. Plug $\boldsymbol{B}_{2,1}$ back into equation (A.29-LHS):

$$\boldsymbol{A}_{1,1}\boldsymbol{B}_{1,1} - \boldsymbol{A}_{1,2}\boldsymbol{B}_{2,2}\boldsymbol{A}_{2,1}\boldsymbol{A}_{1,1}^{-1} = \boldsymbol{I}, \tag{A.39}$$

so

$$\boldsymbol{B}_{1,1} = \boldsymbol{A}_{1,1}^{-1} + \boldsymbol{A}_{1,1}^{-1}\boldsymbol{A}_{1,2}\boldsymbol{B}_{2,2}\boldsymbol{A}_{2,1}\boldsymbol{A}_{1,1}^{-1}. \tag{A.40}$$

After fiddling around with these equations for a while, I found that the shortest forms of $\boldsymbol{B}_{i,j}$ are found by eliminating the other $\boldsymbol{B}_{i,j}$ in the homogeneous equations in equations (A.29) and (A.30). Thus, to get $\boldsymbol{B}_{1,1}$, rearrange equation (A.30-LHS) to get an expression for $\boldsymbol{B}_{2,1}$ and plug this into equation (A.29-LHS). These simplest forms are:

$$\boxed{\begin{aligned}
\boldsymbol{B}_{1,1} &= \left(\boldsymbol{A}_{1,1} - \boldsymbol{A}_{1,2}\boldsymbol{A}_{2,2}^{-1}\boldsymbol{A}_{2,1}\right)^{-1}\\
\boldsymbol{B}_{1,2} &= \left(\boldsymbol{A}_{2,1} - \boldsymbol{A}_{2,2}\boldsymbol{A}_{1,2}^{-1}\boldsymbol{A}_{1,1}\right)^{-1}\\
\boldsymbol{B}_{2,1} &= \left(\boldsymbol{A}_{1,2} - \boldsymbol{A}_{1,1}\boldsymbol{A}_{2,1}^{-1}\boldsymbol{A}_{2,2}\right)^{-1}\\
\boldsymbol{B}_{2,2} &= \left(\boldsymbol{A}_{2,2} - \boldsymbol{A}_{2,1}\boldsymbol{A}_{1,1}^{-1}\boldsymbol{A}_{1,2}\right)^{-1}
\end{aligned}} \tag{A.41}$$

Well, maybe not the simplest, but there is a nice symmetry in this pile of equations.

Here is a summary of the other forms obtained.

$$\boxed{\begin{aligned}
\boldsymbol{B}_{1,1} &= \boldsymbol{A}_{1,1}^{-1} + \boldsymbol{A}_{1,1}^{-1}\boldsymbol{A}_{1,2}\boldsymbol{B}_{2,2}\boldsymbol{A}_{2,1}\boldsymbol{A}_{1,1}^{-1},\\
\boldsymbol{B}_{1,2} &= -\boldsymbol{A}_{1,1}^{-1}\boldsymbol{A}_{1,2}\boldsymbol{B}_{2,2},\\
\boldsymbol{B}_{2,1} &= -\boldsymbol{B}_{2,2}\boldsymbol{A}_{2,1}\boldsymbol{A}_{1,1}^{-1},\\
\boldsymbol{B}_{2,2} &= (\boldsymbol{A}_{2,2} - \boldsymbol{A}_{2,1}\boldsymbol{A}_{1,1}^{-1}\boldsymbol{A}_{1,2})^{-1}.
\end{aligned}} \tag{A.42}$$

A.6.2 Conditional MVN

There is some excellent tutorial material in chapters I and IV of Giri's book, [Gir77]. But I could not figure out Giri's derivation of the fact that a conditional MVN is also MVN, so I give my own derivation here.

We start with a random vector $\boldsymbol{x}$ with expectation $\boldsymbol{\mu}$. This means just a collection of random variables $x_1 \dots x_N$. By assumption, the vector has a MVN probability density, i.e. the collection of random variables has a MVN probability density. We write the covariance matrix as $\boldsymbol{\Sigma}$ and its inverse as $\boldsymbol{\Xi}$ (Greek upper-case *xi*). Then by the definition of MVN, the density of $\boldsymbol{x}$ is given by:

$$f_{\boldsymbol{x}}(\boldsymbol{x}) = \frac{1}{(2\pi)^{N/2}\sqrt{|\boldsymbol{\Sigma}|}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Xi}(\boldsymbol{x}-\boldsymbol{\mu})\right]. \tag{A.43}$$

We are now going to partition the random vector into components of length p and $N-p$, and similarly with $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ and $\boldsymbol{\Xi}$. The two blocks of $\boldsymbol{x}$ will be denoted $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, and similarly for other relevant vectors and matrices (the latter having four blocks, of course). Thus:

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{1,1} & \boldsymbol{\Sigma}_{1,2} \\ \boldsymbol{\Sigma}_{2,1} & \boldsymbol{\Sigma}_{2,2} \end{bmatrix}, \qquad \boldsymbol{\Xi} = \begin{bmatrix} \boldsymbol{\Xi}_{1,1} & \boldsymbol{\Xi}_{1,2} \\ \boldsymbol{\Xi}_{2,1} & \boldsymbol{\Xi}_{2,2} \end{bmatrix}, \tag{A.44}$$

where $\boldsymbol{\Sigma}_{1,1}$ and $\boldsymbol{\Sigma}_{2,2}$ are square (but not necessarily the same size, so the other two blocks can be non-square), and similarly for $\boldsymbol{\Xi}$.

Equations (A.42) translate into "$\boldsymbol{\Sigma} - \boldsymbol{\Xi}$" notation as follows.

$$\boldsymbol{\Xi}_{1,1} = \boldsymbol{\Sigma}_{1,1}^{-1} + \boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\Sigma}_{1,2}\boldsymbol{\Xi}_{2,2}\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}, \tag{A.45}$$

$$\boldsymbol{\Xi}_{1,2} = -\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\Sigma}_{1,2}\boldsymbol{\Xi}_{2,2}, \tag{A.46}$$

$$\boldsymbol{\Xi}_{2,1} = -\boldsymbol{\Xi}_{2,2}\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}, \tag{A.47}$$

$$\boldsymbol{\Xi}_{2,2} = (\boldsymbol{\Sigma}_{2,2} - \boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\Sigma}_{1,2})^{-1}. \tag{A.48}$$

Note that $\boldsymbol{\Xi}_{1,1} = (\boldsymbol{\Sigma}^{-1})_{1,1} \neq \boldsymbol{\Sigma}_{1,1}^{-1}$.

Observe the following important fact:

$$\begin{aligned} \boldsymbol{\Xi}_{1,2}^T &= -(\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\Sigma}_{1,2}\boldsymbol{\Xi}_{2,2})^T && \text{(A.49)} \\ &= -\boldsymbol{\Xi}_{2,2}^T\boldsymbol{\Sigma}_{1,2}^T(\boldsymbol{\Sigma}_{1,1}^{-1})^T && \text{(A.50)} \\ &= -\boldsymbol{\Xi}_{2,2}\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1} && \text{(A.51)} \\ &= \boldsymbol{\Xi}_{2,1}. && \text{(A.52)} \end{aligned}$$

The first equality is the result derived earlier; the second just uses the rule for transposition of matrix multiplication (i.e. "transpose all the matrices and reverse the order of multiplication"); the third exploits the symmetry of the covariance matrix; the last is again an earlier result.

Before going further, we need to know something about the marginal densities.

A.6.3 Marginals of an MVN

[Gir77] shows fairly straightforwardly:

Statement 1. *If the partitioning is like this:*

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{1,1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_{2,2} \end{bmatrix}, \tag{A.53}$$

then $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are independently MVN with means $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$, respectively, and covariance matrices $\boldsymbol{\Sigma}_{1,1}$ and $\boldsymbol{\Sigma}_{2,2}$ respectively.

Proof. Although it is not necessarily true that $\boldsymbol{\Sigma}_{i,j}^{-1} = \boldsymbol{\Xi}_{i,j}$, in this case it is easily seen by matrix multiplication that

$$\begin{bmatrix} \boldsymbol{\Sigma}_{1,1}^{-1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_{2,2}^{-1} \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{\Sigma}_{1,1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_{2,2} \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{\Xi}_{1,1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Xi}_{2,2} \end{bmatrix}. \tag{A.54}$$

Furthermore, $|\boldsymbol{\Sigma}| = |\boldsymbol{\Sigma}_{1,1}|\,|\boldsymbol{\Sigma}_{2,2}|$. So

$$(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Xi}(\boldsymbol{x}-\boldsymbol{\mu}) = (\boldsymbol{x}_1-\boldsymbol{\mu}_1)^T\boldsymbol{\Xi}_{1,1}(\boldsymbol{x}_1-\boldsymbol{\mu}_1) + (\boldsymbol{x}_2-\boldsymbol{\mu}_2)^T\boldsymbol{\Xi}_{2,2}(\boldsymbol{x}_2-\boldsymbol{\mu}_2), \tag{A.55}$$

and

$$f_{\boldsymbol{x}}(\boldsymbol{x}) = \frac{1}{(2\pi)^{p/2}\sqrt{|\boldsymbol{\Sigma}_{1,1}|}} \exp\left[-\frac{1}{2}(\boldsymbol{x}_1-\boldsymbol{\mu}_1)^T\boldsymbol{\Xi}_{1,1}(\boldsymbol{x}_1-\boldsymbol{\mu}_1)\right] \times \frac{1}{(2\pi)^{(N-p)/2}\sqrt{|\boldsymbol{\Sigma}_{2,2}|}} \exp\left[-\frac{1}{2}(\boldsymbol{x}_2-\boldsymbol{\mu}_2)^T\boldsymbol{\Xi}_{2,2}(\boldsymbol{x}_2-\boldsymbol{\mu}_2)\right] \tag{A.56}$$

□

Now we follow Giri again and show that:

Statement 2. *$\boldsymbol{x}_1$ and $\boldsymbol{x}_2 - \boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{x}_1$ are independently MVN with respective means and covariances $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2 - \boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\mu}_1$, $\boldsymbol{\Sigma}_{1,1}$, $\boldsymbol{\Xi}_{2,2}^{-1}$.*

Proof. Let

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{I}_{1,1} & \mathbf{0} \\ -\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1} & \boldsymbol{I}_{2,2} \end{bmatrix}, \tag{A.57}$$

which is nonsingular (i.e. has an inverse) because the determinant of a triangular matrix is just the product of the terms in the leading diagonal. In fact, $|\boldsymbol{C}| = 1$. Transformation of an MVN by a nonsingular matrix gives another MVN (not proved in this note, but fairly easy given the rule for transformation of a multivariate density). Thus

$$\boldsymbol{C}\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 - \boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{x}_1 \end{bmatrix} \tag{A.58}$$

is a N-variate MVN with mean:

$$\boldsymbol{C}\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 - \boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\mu}_1 \end{bmatrix} \tag{A.59}$$

and covariance matrix:

$$\boldsymbol{C}\boldsymbol{\Sigma}\boldsymbol{C}^T = \begin{bmatrix} \boldsymbol{\Sigma}_{1,1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Xi}_{2,2}^{-1} \end{bmatrix}. \tag{A.60}$$

This is the kind of matrix we saw in Statement 1, hence the result. □

Giri makes the following statement in his Corollary 4.1.1(b), but does not explain why it works.

Statement 3. *The marginal distribution of $\boldsymbol{x}_1$ is p-variate MVN with mean $\boldsymbol{\mu}_1$ and covariance matrix $\boldsymbol{\Sigma}_1$.*

Proof. I do not follow Giri's line of reasoning here. However, I do prove it the hard way, below. □

A.6.4 Conditional MVN

Now we are ready to show that:

Statement 4. *The conditional distribution of* $\boldsymbol{x}_2$ *given* $\boldsymbol{x}_1$ *is MVN with mean* $\boldsymbol{\mu}_2+\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}(\boldsymbol{x}_1-\boldsymbol{\mu}_1)$ *and covariance matrix* $\boldsymbol{\Xi}_{2,2}^{-1}$.

Proof. Writing out the exponent in equation (A.43) in partitioned form:

$$(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Xi}(\boldsymbol{x}-\boldsymbol{\mu}) = (\boldsymbol{x}_1-\boldsymbol{\mu}_1)^T\boldsymbol{\Xi}_{1,1}(\boldsymbol{x}_1-\boldsymbol{\mu}_1) + (\boldsymbol{x}_1-\boldsymbol{\mu}_1)^T\boldsymbol{\Xi}_{1,2}(\boldsymbol{x}_2-\boldsymbol{\mu}_2)+ (\boldsymbol{x}_2-\boldsymbol{\mu}_2)^T\boldsymbol{\Xi}_{2,1}(\boldsymbol{x}_1-\boldsymbol{\mu}_1) + (\boldsymbol{x}_2-\boldsymbol{\mu}_2)^T\boldsymbol{\Xi}_{2,2}(\boldsymbol{x}_2-\boldsymbol{\mu}_2) \quad \text{(A.61)}$$

This is going to get messy, so let us define $\boldsymbol{\delta}_i = \boldsymbol{x}_i - \boldsymbol{\mu}_i$. Then:

$$\boldsymbol{\delta}^T\boldsymbol{\Xi}\boldsymbol{\delta} = \boldsymbol{\delta}_1^T\boldsymbol{\Xi}_{1,1}\boldsymbol{\delta}_1 + \boldsymbol{\delta}_1^T\boldsymbol{\Xi}_{1,2}\boldsymbol{\delta}_2 + \boldsymbol{\delta}_2^T\boldsymbol{\Xi}_{2,1}\boldsymbol{\delta}_1 + \boldsymbol{\delta}_2^T\boldsymbol{\Xi}_{2,2}\boldsymbol{\delta}_2. \quad \text{(A.62)}$$

We now plug in the results for $\boldsymbol{\Xi}_{i,j}$:

$$\boldsymbol{\delta}_1^T\boldsymbol{\Xi}_{1,1}\boldsymbol{\delta}_1 = \boldsymbol{\delta}_1^T\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\delta}_1 + \boldsymbol{\delta}_1^T\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\Sigma}_{1,2}\boldsymbol{\Xi}_{2,2}\boldsymbol{\Sigma}_{1,2}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\delta}_1 \quad \text{(A.63)}$$

$$\boldsymbol{\delta}_1^T\boldsymbol{\Xi}_{1,2}\boldsymbol{\delta}_2 = -\boldsymbol{\delta}_1^T\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\Sigma}_{1,2}\boldsymbol{\Xi}_{2,2}\boldsymbol{\delta}_2 \quad \text{(A.64)}$$

$$\boldsymbol{\delta}_2^T\boldsymbol{\Xi}_{2,1}\boldsymbol{\delta}_1 = -\boldsymbol{\delta}_2^T\boldsymbol{\Xi}_{2,2}\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\delta}_1 \quad \text{(A.65)}$$

Giri's equation (4.4) has this in the exponent:

$$\begin{aligned}(\boldsymbol{\delta}_2-\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\delta}_1)^T\boldsymbol{\Xi}_{2,2}(\boldsymbol{\delta}_2-\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\delta}_1) = \\ \boldsymbol{\delta}_2^T\boldsymbol{\Xi}_{2,2}\boldsymbol{\delta}_2^T - (\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\delta}_1)^T\boldsymbol{\Xi}_{2,2}\boldsymbol{\delta}_2 - \boldsymbol{\delta}_2^T\boldsymbol{\Xi}_{2,2}\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\delta}_1 \\ + (\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\delta}_1)^T\boldsymbol{\Xi}_{2,2}\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\delta}_1 \\ = \boldsymbol{\delta}_2^T\boldsymbol{\Xi}_{2,2}\boldsymbol{\delta}_2^T - \boldsymbol{\delta}_1^T\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\Sigma}_{1,2}\boldsymbol{\Xi}_{2,2}\boldsymbol{\delta}_2 - \boldsymbol{\delta}_2^T\boldsymbol{\Xi}_{2,2}\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\delta}_1 \\ + \boldsymbol{\delta}_1^T\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\Sigma}_{1,2}\boldsymbol{\Xi}_{2,2}\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\delta}_1. \quad \text{(A.66)}\end{aligned}$$

Thus we see that

$$\boldsymbol{\delta}^T\boldsymbol{\Xi}\boldsymbol{\delta} = \boldsymbol{\delta}_1^T\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\delta}_1 + (\boldsymbol{\delta}_2-\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\delta}_1)^T\boldsymbol{\Xi}_{2,2}(\boldsymbol{\delta}_2-\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\delta}_1). \quad \text{(A.67)}$$

Now the following is true:

$$|\boldsymbol{\Sigma}| = |\boldsymbol{\Sigma}_{1,1}||\boldsymbol{\Sigma}_{2,2}-\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\Sigma}_{1,2}|. \quad \text{(A.68)}$$

To see this, first believe that the determinant of a block-triangular matrix is the product of the determinants of the blocks on the leading diagonal. Thus:

$$\begin{vmatrix}\boldsymbol{I} & -\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\Sigma}_{1,2}\\ \boldsymbol{0} & \boldsymbol{I}\end{vmatrix} = 1, \quad \text{(A.69)}$$

and so:

$$|\boldsymbol{\Sigma}| = \begin{vmatrix}\boldsymbol{\Sigma}_{1,1} & \boldsymbol{\Sigma}_{1,2}\\ \boldsymbol{\Sigma}_{2,1} & \boldsymbol{\Sigma}_{2,2}\end{vmatrix}\begin{vmatrix}\boldsymbol{I} & -\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\Sigma}_{1,2}\\ \boldsymbol{0} & \boldsymbol{I}\end{vmatrix} = \begin{vmatrix}\boldsymbol{\Sigma}_{1,1} & \boldsymbol{0}\\ \boldsymbol{\Sigma}_{2,1} & \boldsymbol{\Sigma}_{2,2}-\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\Sigma}_{1,2}\end{vmatrix} \quad \text{(A.70)}$$

$$= |\boldsymbol{\Sigma}_{1,1}||\boldsymbol{\Sigma}_{2,2}-\boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\Sigma}_{1,2}| \quad \text{(A.71)}$$

That is,

$$|\boldsymbol{\Sigma}| = |\boldsymbol{\Sigma}_{1,1}|\,|\boldsymbol{\Xi}_{2,2}^{-1}|. \quad \text{(A.72)}$$

Furthermore,

$$(2\pi)^{N/2} = (2\pi)^{p/2}(2\pi)^{(N-p)/2}. \quad \text{(A.73)}$$

Finally, pulling all these strands together:

$$f_{\boldsymbol{x}}(\boldsymbol{x}) = \frac{1}{(2\pi)^{p/2}\sqrt{|\boldsymbol{\Sigma}_{1,1}|}} \exp\left[-\frac{1}{2}\boldsymbol{\delta}_1^T \boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\delta}_1\right] \times \frac{1}{(2\pi)^{(N+p)/2}\sqrt{|\boldsymbol{\Xi}_{2,2}^{-1}|}} \exp\left[-\frac{1}{2}(\boldsymbol{\delta}_2 - \boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\delta}_1)^T \boldsymbol{\Xi}_{2,2}(\boldsymbol{\delta}_2 - \boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\delta}_1)\right]. \quad \text{(A.74)}$$

This is the product of two MVN densities, the first being $MVN(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{1,1})$, the second being $MVN(\boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}(\boldsymbol{x}_1 - \boldsymbol{\mu}_1), \boldsymbol{\Xi}_{2,2}^{-1})$. But the second one is just another way of looking at the MVN vector that was discussed in Statement 2, so we know that it is independent of the first. Therefore we see that the first is the density of an MVN vector in its own right, and indeed that it is the marginal density of $\boldsymbol{x}_1$. Thus Statement 3 is proved, as promised.

Now $f_{\boldsymbol{x}}(\boldsymbol{x}) = f_{\boldsymbol{x}_1 \cap \boldsymbol{x}_2}(\boldsymbol{x}_1 \cap \boldsymbol{x}_2)$. This is because it is the density of getting $x_1, x_2, \ldots, x_N$ at the same time, which is the same as the probability of getting $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ at the same time. But by the definition of a conditional density,

$$f_{\boldsymbol{x}_1 \cap \boldsymbol{x}_2}(\boldsymbol{x}_1 \cap \boldsymbol{x}_2) = f_{\boldsymbol{x}_2|\boldsymbol{x}_1}(\boldsymbol{x}_2|\boldsymbol{x}_1) f_{\boldsymbol{x}_1}(\boldsymbol{x}_1). \quad \text{(A.75)}$$

Therefore the second density on the RHS of equation (A.74) is equal to $f_{\boldsymbol{x}_2|\boldsymbol{x}_1}(\boldsymbol{x}_2|\boldsymbol{x}_1)$. □

B Traces and Derivatives of Them

Given a square matrix $\boldsymbol{A}$, its trace is the sum of the elements in its leading diagonal, and this is written $\mathrm{tr}(\boldsymbol{A})$. Being the sum of some scalars (the numbers along the diagonal of $\boldsymbol{A}$), it is a scalar. Thus

$$\mathrm{tr}(\boldsymbol{A}) = \sum_i A_{ii} = A_{ii}, \quad \text{(B.1)}$$

where the first equality just says what $\mathrm{tr}(\boldsymbol{A})$ is, and the second says the same thing using the summation convention [Wik07a].

Now we need to define the derivative of a scalar with respect to a matrix. Call the scalar ξ, and define:

$$\frac{d\xi}{d\boldsymbol{A}} = \begin{bmatrix} \frac{\partial \xi}{\partial A_{11}} & \frac{\partial \xi}{\partial A_{12}} & \cdots \\ \frac{\partial \xi}{\partial A_{21}} & \frac{\partial \xi}{\partial A_{22}} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \quad \text{(B.2)}$$

We can use these two results to evaluate $d\,\mathrm{tr}(\boldsymbol{AB})/d\boldsymbol{A}$, for two conformable matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. It only makes sense if $\boldsymbol{AB}$ is square — which it is in this paper. Using the summation convention, $(\boldsymbol{AB})_{mn} = A_{mo}B_{on}$, which I think of as "the Moon rule". Then $\mathrm{tr}(\boldsymbol{AB}) = A_{mo}B_{om}$. In the following equations, there will be a lot of contraction over indices. It can be hard on the eyes and brain to follow what is happening. To help the reader (and the writer), I have used boxes to show which objects combine on a LHS to form an object on a RHS. To illustrate: $\boxed{\delta_{op}} A_{mo} \boxed{B_{pn}} = A_{mo} \boxed{B_{on}}$. So:

$$\frac{d\,\mathrm{tr}(\boldsymbol{AB})}{d\boldsymbol{A}} = \frac{d(A_{ij}B_{ji})}{dA_{\alpha\beta}} = \boxed{\delta_{i\alpha}}\delta_{j\beta}\boxed{B_{ji}} = \delta_{j\beta}\boxed{B_{j\alpha}} \quad \text{(B.3a)}$$

$$= B_{\beta\alpha} = (\boldsymbol{B})^T_{\alpha\beta} \quad \text{(B.3b)}$$

$$= \boldsymbol{B}^T. \quad \text{(B.3c)}$$

We will also need $d\,\mathrm{tr}(\boldsymbol{ACA}^T)/d\boldsymbol{A}$.

$$\frac{d\,\mathrm{tr}(\boldsymbol{ACA}^T)}{d\boldsymbol{A}} = \frac{d(A_{ij}C_{jk}A_{ik})}{dA_{\alpha\beta}} = \delta_{i\alpha}\boxed{\delta_{j\beta}}\boxed{C_{jk}}A_{ik} + A_{ij}\boxed{C_{jk}}\delta_{i\alpha}\boxed{\delta_{k\beta}} \tag{B.4a}$$

$$= \boxed{\delta_{i\alpha}}C_{\beta k}\boxed{A_{ik}} + \boxed{A_{ij}}C_{j\beta}\boxed{\delta_{i\alpha}} = C_{\beta k}\boxed{A_{\alpha k}} + \boxed{A_{\alpha j}}C_{j\beta} \tag{B.4b}$$

$$= (\boldsymbol{AC}^T)_{\alpha\beta} + (\boldsymbol{AC})_{\alpha\beta} \tag{B.4c}$$

$$= \boldsymbol{AC}^T + \boldsymbol{AC} \tag{B.4d}$$

$$= 2\boldsymbol{AC} \qquad \text{if } \boldsymbol{C} \text{ is symmetric.} \tag{B.4e}$$

C R Code for the DALEC Model

Fortran code for DALEC is available at [Fox]. I have translated it into R, and that code is shown here, without much comment. But if you look at Figure 9 at the same time as reading this, it might make sense. The main things to note are:

1. it is an Euler method encoding of a finite difference algorithm;
2. various coefficients and other constants are used, but are not presented here;
3. the latter can be found in, or inferred from, [Fox] and [WSL+05].

The first chunk of code calculates the changes in the carbon pools in one day:

```
# Function to update carbon pools for one day.
carbon.one.day <- function(x , drive , p , a , others , stats=NULL) {
# carbon.one.day <- function(x , drive , p , a , others) {
   # x            vector: is the control variable: c( c.f , c.r , c.w , c.lit , c.som )
   # drive        vector: contains the drive data: c(yearday , mint , maxt , c.a , rad )
   # p            vector: contains the 11 p-parameters
   # a            vector: contains the 10 a-parameters
   # others       vector: contains other stuff: c(psid , rtot , nit)
   c.f = x[1] ; c.r = x[2] ; c.w = x[3] ; c.lit = x[4] ; c.som = x[5]
   yearday = drive[1] ; mint = drive[2] ; maxt = drive[3] ;
   c.a = drive[4] ; rad = drive[5]
   psid = others[1] ; rtot = others[2] ; nit = others[3]
   #
   lai = max(0.1 , c.f/lma)
   gpp.arg = list(); gpp.arg$p = p ; gpp.arg$a = a ; gpp.arg$lai = lai
   gpp.arg$maxt = maxt ; gpp.arg$mint = mint ; gpp.arg$psid = psid
   gpp.arg$rtot = rtot ; gpp.arg$nit = nit ; gpp.arg$c.a = c.a
   gpp.arg$yearday = yearday ; gpp.arg$rad = rad
   # Carbon fluxes
   g = gpp(gpp.arg)
   t.rate = 0.5 * exp(p[10] * 0.5 * (maxt + mint))
   r.a = p[2] * g
   a.f = (g - r.a) * p[3]
   a.r = (g - r.a - a.f) * p[4]
   a.w = g - r.a - a.f - a.r
   l.f = p[5] * c.f
   l.w = p[6] * c.w
   l.r = p[7] * c.r
```

```
   rh.1 = p[8] * c.lit * t.rate
   rh.2 = p[9] * c.som * t.rate
   d = p[1] * c.lit * t.rate
   # Pools:
   c.f = c.f + a.f - l.f
   c.w = c.w + a.w - l.w
   c.r = c.r + a.r - l.r
   c.lit = c.lit + l.f + l.r - rh.1 - d
   c.som = c.som + d - rh.2 + l.w
   new.pools = c( c.f , c.r , c.w , c.lit , c.som )
   lai = H %*% new.pools
   obs = lai
   return( c( new.pools , g , r.a , a.f , a.w , a.r , l.f , l.w , l.r , rh.1 ,
                  rh.2 , d , (r.a+rh.1+rh.2-g) , lai , obs ) )
   }
#
```

The next chunk of code calculates GPP in one day:

```
# Function to calculate GPP for one day:
gpp = function(gpp.arg) {
        # gpp.arg is the stuff it needs.
        # Returns gpp for one day.
        #
        p = gpp.arg$p
        lai = gpp.arg$lai
        maxt = gpp.arg$maxt
        mint = gpp.arg$mint
        psid = gpp.arg$psid
        rtot = gpp.arg$rtot
        nit = gpp.arg$nit
        c.a = gpp.arg$c.a
        yearday = gpp.arg$yearday
        rad = gpp.arg$rad
        # Calculate
        t.range = 0.5 * (maxt - mint)
        g.s = abs(psid)**a[10] / ((a[6] * rtot + t.range))
        pp = lai * nit / g.s * a[1] * exp(a[8] * maxt)
        qq = a[3] - a[4]
        c.i = 0.5 * (c.a+qq-pp + ((c.a+qq-pp)**2 - 4 * (c.a*qq-pp*a[3]))**0.5)
        e.0 = a[7] * lai**2 / (lai**2 + a[9])
        dec = -23.4 * cos((360.0 * (yearday+10.0) / 365.0) * pi/180.0) * pi/180.0
        mult = tan(lat) * tan(dec)
        if (mult >= 1.0 ) {
           dayl = 24.0 } else
        if (mult <= -1.0 ) {
           dayl = 0.0 } else
           { dayl = 24.0 * acos(-mult) / pi}
        cps = e.0 * rad * g.s * (c.a-c.i) / (e.0 * rad + g.s * (c.a-c.i))
        model = cps * (a[2] * dayl + a[5])
        gpp = model     # Return this
```

```
        }
#
```

D Multivariate Taylor Series

We have some vector-valued function of a vector-valued argument, $\boldsymbol{f}(\boldsymbol{x})$. Lengths of these vectors are $\boldsymbol{f} \sim n$ and $\boldsymbol{x} \sim m$. We want to find a first-order Taylor series, i.e. a linearisation, of $\boldsymbol{f}$ around some point $\boldsymbol{x}$. But first, we will linearise a scalar function of a a vector argument, $f(\boldsymbol{x})$:

$$f(\boldsymbol{x}+\boldsymbol{\xi}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \xi_i \frac{\partial f(\boldsymbol{x})}{\partial x_i} + O(|\boldsymbol{\xi}|^2). \tag{D.1}$$

So:

$$\boldsymbol{f}(\boldsymbol{x}+\boldsymbol{\xi}) = \begin{bmatrix} f_1(\boldsymbol{x}+\boldsymbol{\xi}) \\ f_2(\boldsymbol{x}+\boldsymbol{\xi}) \\ \vdots \\ f_n(\boldsymbol{x}+\boldsymbol{\xi}) \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) + \sum_{i=1}^{m} \xi_i \frac{\partial f_1(\boldsymbol{x})}{\partial x_i} + O(|\boldsymbol{\xi}|^2) \\ f_2(\boldsymbol{x}) + \sum_{i=1}^{m} \xi_i \frac{\partial f_2(\boldsymbol{x})}{\partial x_i} + O(|\boldsymbol{\xi}|^2) \\ \vdots \\ f_n(\boldsymbol{x}) + \sum_{i=1}^{m} \xi_i \frac{\partial f_n(\boldsymbol{x})}{\partial x_i} + O(|\boldsymbol{\xi}|^2) \end{bmatrix} \tag{D.2a}$$

$$\simeq \boldsymbol{f}(\boldsymbol{x}) + \begin{bmatrix} \frac{\partial f_1(\boldsymbol{x})}{\partial x_1} & \frac{\partial f_1(\boldsymbol{x})}{\partial x_2} & \cdots & \frac{\partial f_1(\boldsymbol{x})}{\partial x_m} \\ \frac{\partial f_2(\boldsymbol{x})}{\partial x_1} & \frac{\partial f_2(\boldsymbol{x})}{\partial x_2} & \cdots & \frac{\partial f_2(\boldsymbol{x})}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n(\boldsymbol{x})}{\partial x_1} & \frac{\partial f_n(\boldsymbol{x})}{\partial x_2} & \cdots & \frac{\partial f_n(\boldsymbol{x})}{\partial x_m} \end{bmatrix} \cdot \boldsymbol{\xi} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{F}\boldsymbol{\xi}, \tag{D.2b}$$

which defines the Jacobian $\boldsymbol{F}$.

This result is very well known, but the ordering of the $\boldsymbol{F}$ and the $\boldsymbol{\xi}$ always eludes me unless I derive it from scratch. Hence this section.

E The Module

```
! David Pearson
! Module to define overloading of (real , differential) pair operators
! Version 071126a
!-----------------------------------------------------|
module diffy_maths
!-----------------------------------------------------|
   implicit none
!-----------------------------------------------------|
   type diffy
      real     :: value , differential
   end type diffy
!-----------------------------------------------------|
   interface operator (+)
      module procedure diffy_plus_diffy
   end interface
!-----------------------------------------------------|
   interface operator (-)
      module procedure real_minus_diffy , diffy_minus_diffy , int_minus_diffy
   end interface
```

```
!-----------------------------------------------------|
   interface operator (*)
      module procedure diffy_times_diffy , real_times_diffy , diffy_times_real
   end interface
!-----------------------------------------------------|
   interface operator (**)
      module procedure diffy_sup_int
   end interface
!-----------------------------------------------------|
   interface rose
      module procedure r_rose , d_rose
   end interface
!-----------------------------------------------------|
contains
!-----------------------------------------------------|
   function diffy_plus_diffy(x1 , x2) result(sum)
      type(diffy) , intent(in)      :: x1 , x2
      type(diffy)                   :: sum
      sum%value = x1%value + x2%value
      sum%differential = x1%differential + x2%differential
   end function diffy_plus_diffy
!-----------------------------------------------------|
   function diffy_minus_diffy(x1 , x2) result(sub)
      type(diffy) , intent(in)      :: x1 , x2
      type(diffy)                   :: sub
      sub%value = x1%value - x2%value
      sub%differential = x1%differential - x2%differential
   end function diffy_minus_diffy
!-----------------------------------------------------|
   function real_minus_diffy(r1 , x2) result(res)
      real , intent(in)             :: r1
      type(diffy) , intent(in)      :: x2
      type(diffy)                   :: res
      res%value = r1 - x2%value
      res%differential = - x2%differential
   end function real_minus_diffy
!-----------------------------------------------------|
   function int_minus_diffy(i1 , x2) result(res)
      integer , intent(in)          :: i1
      type(diffy) , intent(in)      :: x2
      type(diffy)                   :: res
      res%value = i1 - x2%value
      res%differential = - x2%differential
   end function int_minus_diffy
!-----------------------------------------------------|
   function diffy_times_diffy(x1 , x2) result(prod)
      type(diffy) , intent(in)      :: x1 , x2
      type(diffy)                   :: prod
      prod%value = x1%value * x2%value
      prod%differential = x1%value * x2%differential + x2%value * x1%differential
   end function diffy_times_diffy
!-----------------------------------------------------|
   function real_times_diffy(r1 , x2) result(prod)
      real , intent(in)             :: r1
      type(diffy) , intent(in)      :: x2
      type(diffy)                   :: prod
      prod%value = r1 * x2%value
      prod%differential = r1 * x2%differential
```

```
   end function real_times_diffy
!-------------------------------------------------------|
   function diffy_times_real(x1 , r2) result(prod)
      type(diffy) , intent(in)      :: x1
      real , intent(in)             :: r2
      type(diffy)                   :: prod
      prod%value = r2 * x1%value
      prod%differential = r2 * x1%differential
   end function diffy_times_real
!-------------------------------------------------------|
   function diffy_sup_int(x1 , n2) result(expy)
      type(diffy) , intent(in)      :: x1
      integer , intent(in)          :: n2
      type(diffy)                   :: expy
      expy%value = x1%value ** n2
      expy%differential = n2 * x1%value ** (n2 - 1) * x1%differential
   end function diffy_sup_int
!-------------------------------------------------------|
   function r_rose(x , y) result(rosy)
      real , intent(in)             :: x , y
      real                          :: rosy
      rosy = (1 - x)**2 + 100.0 * ((y-x**2)**2)
   end function r_rose
!-------------------------------------------------------|
   function d_rose(x , y) result(rosy)
      type(diffy) , intent(in)      :: x , y
      type(diffy)                   :: rosy
      rosy = (1 - x)**2 + 100.0 * ((y-x**2)**2)
   end function d_rose
!-------------------------------------------------------|
end module diffy_maths
!-------------------------------------------------------|
! FIN
!-------------------------------------------------------|
```

F The Main Program

```
! David Pearson
! Main prog to banana-test diffy_maths
! Version 071126a
!-------------------------------------------------------------------------------
program main
!-------------------------------------------------------------------------------
use diffy_maths
!
implicit none
type(diffy)       :: x , y , z
real              :: r , s , t , t1 , t2 , delta , deriv
integer           :: nexpp
!-------------------------------------------------------------------------------
!
! First try some elementary mathematical operations:
x = diffy(10.0 , 0.1)
y = diffy(3.141 , 0.7654321)
z = x + y
print *, "diffy plus diffy:"
print *, "(+) Result should be " ,                          &
```

```
      (10.0 + 3.141) ,                                          &
      " , " ,                                                   &
      0.1 + 0.7654321
print *, "(+) Result is        " , z
!
print *, "diffy minus diffy:"
z = x - y
print *, "(-) Result should be " ,                              &
      (10.0 - 3.141) ,                                          &
      " , " ,                                                   &
      0.1 - 0.7654321
print *, "(-) Result is        " , z
!
z = 100.0 - y
print *, "real minus diffy:"
print *, "(-) Result should be " ,                              &
      (100.0 - 3.141) ,                                          &
      " , " ,                                                   &
      - 0.7654321
print *, "(-) Result is        " , z
!
print *, "int minus diffy:"
z = 100 - y
print *, "(-) Result should be " ,                              &
      (100 - 3.141) ,                                          &
      " , " ,                                                   &
      - 0.7654321
print *, "(-) Result is        " , z
!
z = x * y
print *, "diffy times diffy:"
print *, "(*) Result should be " ,                              &
      (10.0*3.141) ,                                            &
      " , " ,                                                   &
      (10.0*0.7654321 + 3.141*0.1)
print *, "(*) Result is        " , z
!
z = x * (x + y)
print *, "diffy combination diffy:"
print *, "(*,+) Result should be " ,                            &
      (10.0 * (10.0+3.141)) ,                                    &
      " , " ,                                                   &
      ( (10.0+3.141)*0.1 + 10.0*(0.1+0.7654321) )
print *, "(*,+) Result is        " , z
!
nexpp = 2
print *, "diffy sup int:"
z = (x + y) ** nexpp
print *, "(+,**) Result should be " ,                           &
      ((10.0+3.141)**2) ,                                        &
      " , " ,                                                   &
      ( 2*(10.0+3.141)*(0.1+0.7654321) )
print *, "(+,**) Result is        " , z
!
! Rosenbrock next:
r = 1
s = 1
t = r_rose(r , s)
```

```
print *, "Real banana:"
print * , "rosenbrock(" , r , "," , s , ") = " , t
!
! Rosenbrock next:
!
! REAL
r = 1.1
s = 2.4
x = diffy(r , 0.0)         ! For ...
y = diffy(s , 0.0)         ! ... later
t1 = rose(r , s)
print *, "Real banana:"
print * , "rosenbrock(" , r , "," , s , ") = " , t1
!
! Another real for the derivative:
delta = 0.0001 * r
r = r * 1.0001
t2 = rose(r , s)
print *, "Real banana:"
print * , "rosenbrock(" , r , "," , s , ") = " , t2
!
! And the derivative:
deriv = (t2 - t1) / delta
print *, "Real banana diff:"
print * , "d(rosenbrock(" , r , "," , s , ")) / dr = " , deriv
!
! DIFFY
x%differential = 1.0
z = rose(x , y)
print *, "Diffy banana:"
print * , "d(rosenbrock(" , x , "," , y , ")) / dr = " , z
!-----------------------------------------------------------------------
!
! REAL
r = 1.1
s = 2.4
x = diffy(r , 0.0)         ! For ...
y = diffy(s , 0.0)         ! ... later
t1 = rose(r , s)
print *, "Real banana:"
print * , "rosenbrock(" , r , "," , s , ") = " , t1
!
! Another real for the derivative:
delta = 0.0001 * s
s = s * 1.0001
t2 = rose(r , s)
print *, "Real banana:"
print * , "rosenbrock(" , r , "," , s , ") = " , t2
!
! And the derivative:
deriv = (t2 - t1) / delta
print *, "Real banana diff:"
print * , "d(rosenbrock(" , r , "," , s , ")) / ds = " , deriv
!
! DIFFY
y%differential = 1.0
z = rose(x , y)
print *, "Diffy banana:"
```

```
print * , "d(rosenbrock(" , x , "," , y , ")) / ds = " , z
!-------------------------------------------------------------------------------
stop
end
!-------------------------------------------------------------------------------
! FIN
!-------------------------------------------------------------------------------
```

G The Output

```
 diffy plus diffy:
 (+) Result should be    13.14100      ,   0.8654321
 (+) Result is           13.14100      0.8654321
 diffy minus diffy:
 (-) Result should be    6.859000      ,  -0.6654321
 (-) Result is           6.859000     -0.6654321
 real minus diffy:
 (-) Result should be    96.85900      ,  -0.7654321
 (-) Result is           96.85900     -0.7654321
 int minus diffy:
 (-) Result should be    96.85900      ,  -0.7654321
 (-) Result is           96.85900     -0.7654321
 diffy times diffy:
 (*) Result should be    31.41000      ,    7.968421
 (*) Result is           31.41000       7.968421
 diffy combination diffy:
 (*,+) Result should be    131.4100      ,    9.968422
 (*,+) Result is           131.4100       9.968421
 diffy sup int:
 (+,**) Result should be    172.6859      ,    22.74529
 (+,**) Result is           172.6859       22.74529
 Real banana:
 rosenbrock(   1.000000     ,   1.000000     ) =   0.0000000E+00
 Real banana:
 rosenbrock(   1.100000     ,   2.400000     ) =    141.6200
 Real banana:
 rosenbrock(   1.100110     ,   2.400000     ) =    141.5624
 Real banana diff:
 d(rosenbrock(   1.100110     ,   2.400000     )) / dr =   -523.5152
 Diffy banana:
 d(rosenbrock(   1.100000       1.000000     ,   2.400000      0.0000000E+00
 )) / dr =    141.6200      -523.4000
 Real banana:
 rosenbrock(   1.100000     ,   2.400000     ) =    141.6200
 Real banana:
 rosenbrock(   1.100000     ,   2.400240     ) =    141.6772
 Real banana diff:
 d(rosenbrock(   1.100000     ,   2.400240     )) / ds =    238.1007
 Diffy banana:
 d(rosenbrock(   1.100000      0.0000000E+00 ,   2.400000       1.000000
 )) / ds =    141.6200       238.0000
```

References

[AM05] Brian D. O. Anderson and John B. Moore. *Optimal Filtering.* Dover, 2005. 36, 59

[Ban07] R. N. Bannister. Can wavelets improve the representation of forecast error covariances in variational data assimilation? *Monthly Weather Review*, 135:387–408, February 2007. 59

[BC] F. Bouttier and P. Courtier. Data assimilation concepts and methods. `http://www.ecmwf.int/newsevents/training/rcourse_notes/DATA_ASSIMILATION/ASSIM_CONCEPTS/Assim_concepts.html`. 25

[BCFH03] R. Boudjemaa, M.G. Cox, A.B. Forbes, and P.M. Harris. Automatic differentiation techniques and their application in metrology. Technical Report CMSC 26/03, National Physical Laboratory, June 2003. 57

[BH97] Brown, Robert Grover and Hwang, Patrick Y. C. *Introduction to Random Signals and Applied Kalman Filtering.* John Wiley and Sons, third edition, 1997. 10, 17, 18, 20, 23, 28, 35, 36, 59

[BvE98] G. Burgers, P. J. van Leeuwen, and G. Evensen. Analysis scheme in the ensemble kalman filter. *Monthly Weather Review*, 126:1719–1724, June 1998. 29

[Can02] Brian J. Cantwell. *Introduction to Symmetry Analysis.* Cambridge University Press, First edition, 2002. 9

[CK00] M. Capiński and E. Kopp. *Measure, Integral and Probability.* Springer, First edition, 2000. 10

[Coh97a] Stephen E. Cohn. An introduction to estimation theory. *Journal of the Meteorological Society of Japan*, 75(1B):257–288, 1997. 7, 8, 10, 11, 17, 22, 58

[Coh97b] Stephen E. Cohn. An introduction to estimation theory. `http://gmao.gsfc.nasa.gov/pubs/docs/Cohn192.pdf`, May 1997. 7

[Dd98] D. P. Dee and A. M. da Silva. Data assimilation in the presence of forecast bias. *Quarterly Journal of the Royal Meteorological Society*, 124:269–295, January 1998. 9, 59

[Dee91] D. P. Dee. Simplification of the Kalman Filter for meteorological data assimilation. *Q. J. R. Meteorol. Soc.*, 117:365–384, 1991. 10, 14, 18, 22, 25

[EDS98] G. Evensen, D. Dee, and J. Schröter. Parameter estimation in dynamical models. In E. P. Chassignet and J. Verron, editors, *Ocean Modeling and Parameterization*, pages 373–398. Springer, 1998. 59

[Eve94] G. Evensen. Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics. *Journal of Geophysical Research*, 99:10143–10162, May 1994. 28

[Eve03] G. Evensen. The ensemble Kalman filter: theoretical formulation and practical implementation. *Ocean Dynamics*, 53:343–367, 2003. 28

[Eve09] Geir Evensen. The ensemble Kalman filter for combined state and parameter estimation. *IEEE Control Systems Magazine*, pages 83–104, May 2009. 35

[Fle87] R. Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, second edition, 1987. 39

[Fox] A. M. Fox. REFLEX: REgional FLux Estimation eXperiment. `http://www.geos.ed.ac.uk/carbonfusion/Reflex.html`. 26, 69, 93

[Gel74] Arthur Gelb, editor. *Applied Optimal Estimation.* The M.I.T. Press, first edition, 1974. 12

[Gir77] Narayan C. Giri. *Multivariate Statistical Inference.* Academic Press, 1977. 65

[Gri00] Andreas Griewank. *Evaluating Derivatives. Principles and Techniques of Algorithmic Differentiation.* Society for Industrial and Applied Mathematics, first edition, 2000. 45, 47, 53

[GV96] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, third edition, 1996. 44

[Inv07] G.W Inverarity. Flight Research Laboratory Kalman Filter / Rauch-Tung-Striebel smoother for inertial navigation correction and wind renavigation. `http://www-nwp/~obr/OBR_Notes/OBR_technical_notes/OBR_Tech_note_67_Inverarity.pdf`, July 2007. 21

[Jac10] Kurt Jacobs. *Stochastic Processes for Physicists.* Cambridge University Press, first edition, 2010. 19

[Jaz70] Andrew H. Jazwinski. *Stochastic Processes and Filtering Theory.* Academic Press, 1970. Also available as a Dover paperback. 10, 22, 23, 35

[JU97] S. J. Julier and J. K. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *The Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defense Sensing, Simulation and Controls, Multi Sensor Fusion, Tracking and Resource Management II.* SPIE, 1997. 28

[Kal03] E. Kalnay. *Atmospheric Modeling, Data Assimilation and Predictability.* Cambridge University Press, first edition, 2003. 26, 58, 59

[Law10] A.S. Lawless. A note on the analysis error associated with 3D-FGAT. *Quarterly Journal of the Royal Meteorological Society*, 136:1094–1098, April 2010. 26

[LBB+00] A. C. Lorenc, S. P. Ballard, R. S. Bell, N. B. Ingleby, P. L. F. Andrews, D. M. Barker, J. R. Bray, A. M. Clayton, T. Dalby, D. Li, T. J. Payne, and F. W. Saunders. The Met. Office global three-dimensional variational data assimilation scheme. *Quarterly Journal of the Royal Meteorological Society*, 126:2991–3012, October 2000. 4

[LKSW03] S. N. Losa, G. A. Kivman, J. Schröter, and M. Wenzel. Sequential weak constraint parameter estimation in an ecosystem model. *Journal of Marine Systems*, 43:31–49, September 2003. 36

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in Fortran.* Cambridge University Press, second edition, 1992. 27, 35, 39, 63

[RME02] R. H. Reichle, D. B. McLaughlin, and D Entekhabi. Hydrologic data assimilation with the ensemble Kalman filter. *Monthly Weather Review*, 130:103–114, January 2002. 28

[Rob09] A.J. Roberts. *Elementary Calculus of Financial Mathematics.* SIAM, first edition, 2009. 19

[Ros60] H.H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *Computer Journal*, 3:175–184, 1960. 46

[Sim06] Dan Simon. *Optimal State Estimation: Kalman, H_∞, and Nonlinear Approaches.* Wiley-Interscience, first edition, 2006. 18

[Sne97] J. Laurie Snell. A conversation with Joe Doob. *Statistical Science*, 12(4):301–311, 1997. 4

[SO08] P. Sakov and P. R. Oke. A deterministic formulation of the ensemble Kalman filter: an alternative to ensemble square root filters. *Tellus A*, 60A:361–371, 2008. 28

[ST98] W. Squire and G. Trapp. Using complex variables to estimate derivatives of real functions. *SIAM Review*, 40:110–112, March 1998. 57, 58

[Tod99] Ricardo Todling. Estimation theory and foundations of atmospheric data assimilation. `http://gmao.gsfc.nasa.gov/pubs/docs/Todling180.pdf`, June 1999. 10

[Weia] Eric W. Weisstein. Horner's rule. `http://mathworld.wolfram.com/HornersRule.html`. From MathWorld–A Wolfram Web Resource. 42

[Weib] Eric W. Weisstein. Rosenbrock function. `http://mathworld.wolfram.com/RosenbrockFunction.html`. From MathWorld–A Wolfram Web Resource. 46

[Wik07a] Wikipedia. Einstein notation — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Einstein_notation&oldid=142201577`, 2007. [Online; accessed 19-July-2007]. 43, 68

[Wik07b] Wikipedia. Variance — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Variance&oldid=148134920`, 2007. [Online; accessed 03-August-2007]. 33

[WLNR07] L. R. Watkinson, A. S. Lawless, N. K. Nichols, and I. Roulstone. Weak constraints in four-dimensional variational data assimilation. *Meteorologische Zeitschrift*, 16:767–776, December 2007. 59

[WSL+05] M. Williams, P. A. Schwarz, B. E. Law, J. Irvine, and M. R. Kurpius. An improved analysis of forest carbon dynamics using data assimilation. *Global Change Biology*, 11:89–105, 2005. 26, 69

List of Figures

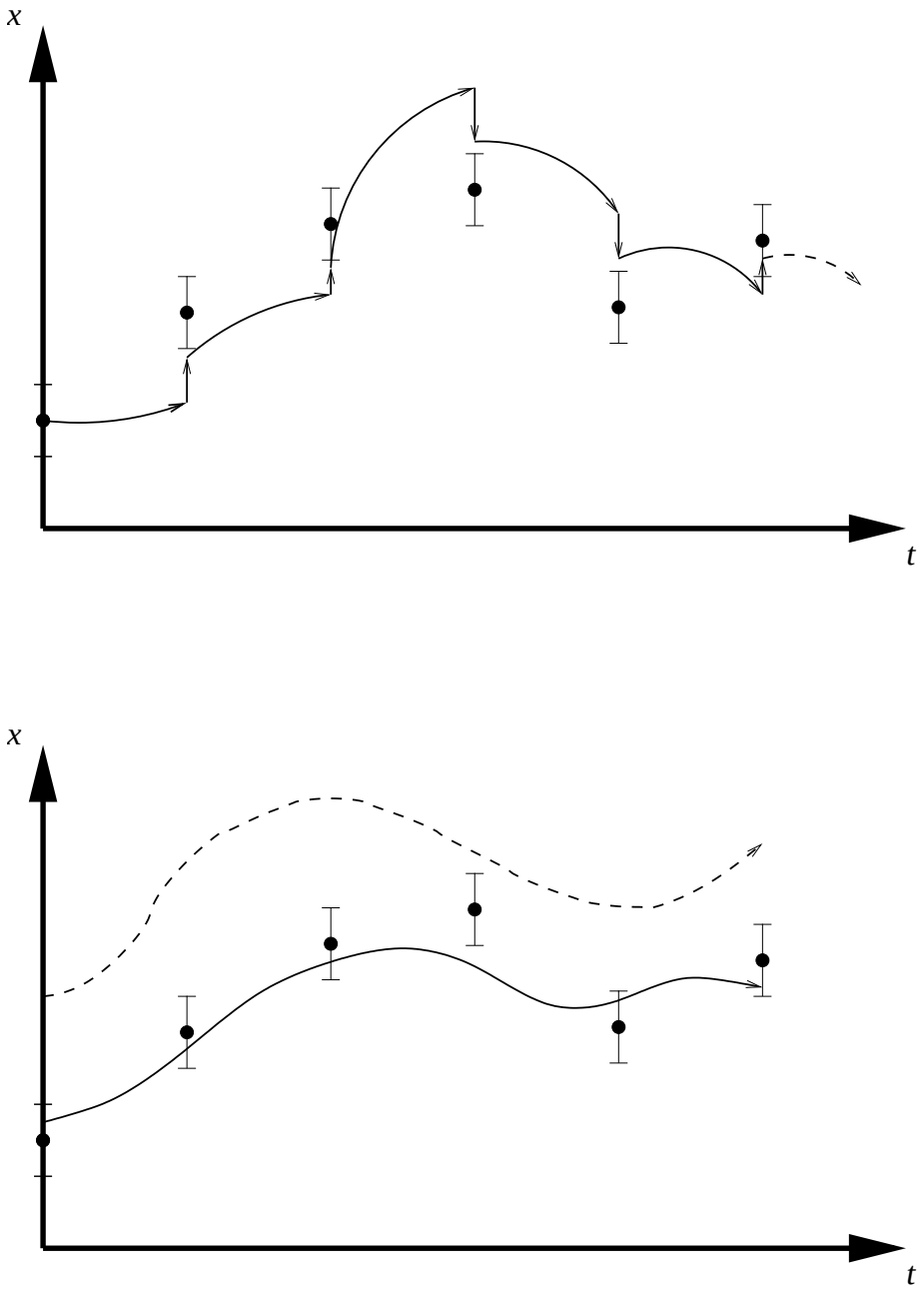


Figure 1: The two fundamental modes of DA. Top: sequential. Bottom: variational.

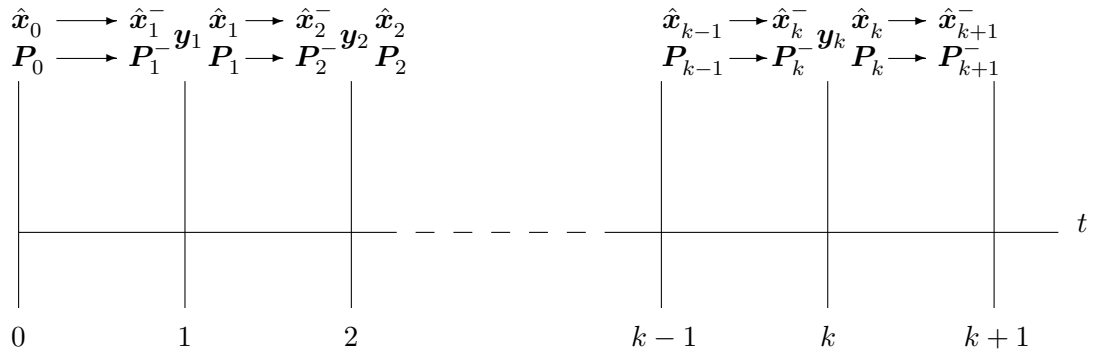


Figure 2: Timing Diagram for the KF. For time k, $\hat{\boldsymbol{x}}_k^-$ and $\hat{\boldsymbol{x}}_k$ are both optimal estimates of $\boldsymbol{x}_k$. The first is an estimate given all the observations before time k, i.e. $\boldsymbol{y}_0, \ldots \boldsymbol{y}_{k-1}$; and the second additionally uses the observations $\boldsymbol{y}_k$.

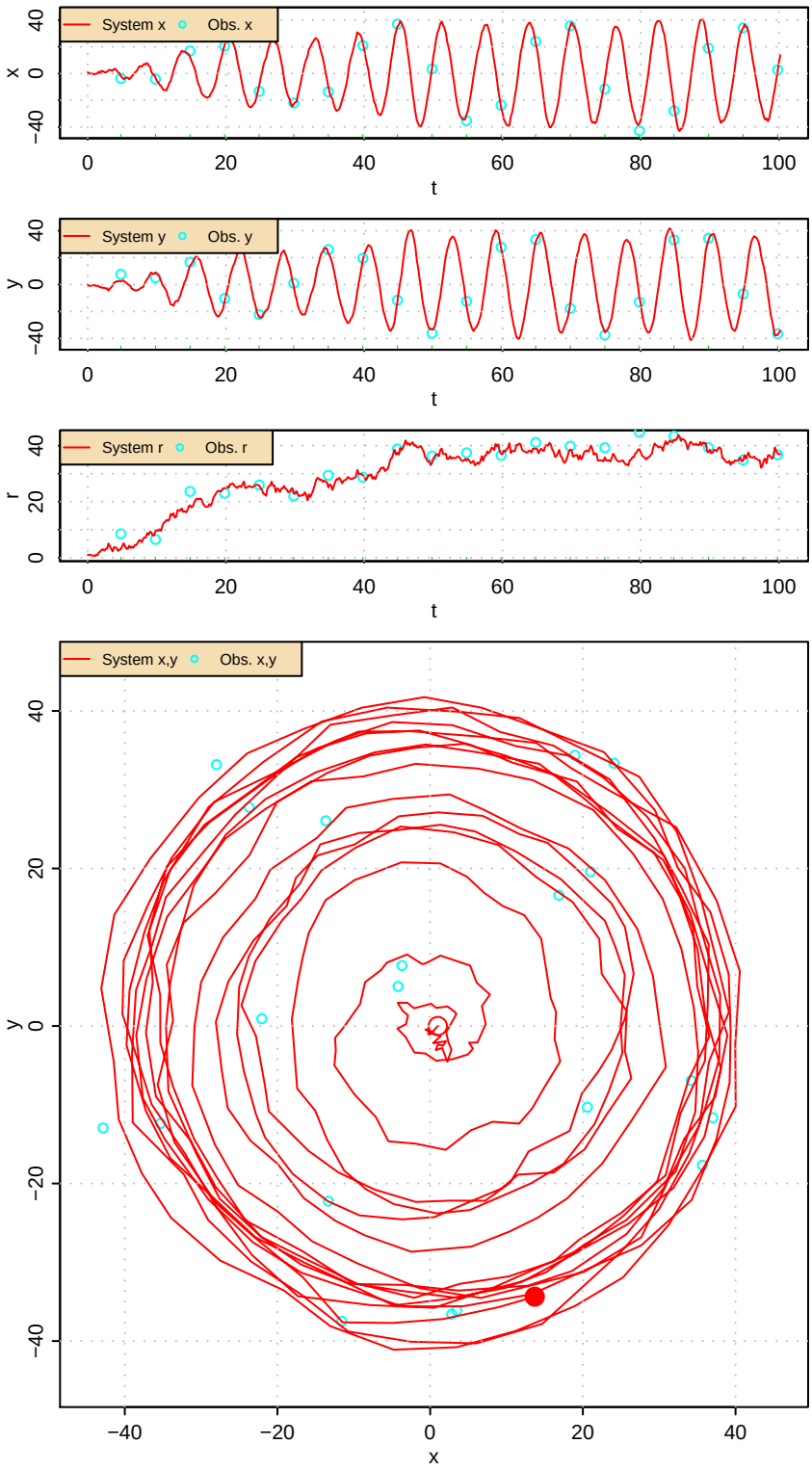


Figure 3: Forward run of the circular motion system. The starting position is $x = 1$, $y = 0$. The angular velocity is $\omega = 1$, giving a rotational period of 2π. Total run time is 100 with a time step of 0.2. The time step is 0.2. After every time step, Gaussian noise with $\boldsymbol{Q} = \boldsymbol{I}$ is added to the co-ordinates. Observations are simulated every 5 time units. The measurement is $\boldsymbol{y} = (x, y) + N(0, \boldsymbol{R})$, with $\boldsymbol{R} = 10\boldsymbol{I}$. The top plot is x vs. time as a wriggly line, with observations as circles, and observation times shown by tick-marks on the interior of the time axis. The next is the same, but for y vs. t; and the third shows radius r vs. t. The big plot at the bottom shows the trajectory, starting at the red open circle close to the origin, ending at the filled red circle.

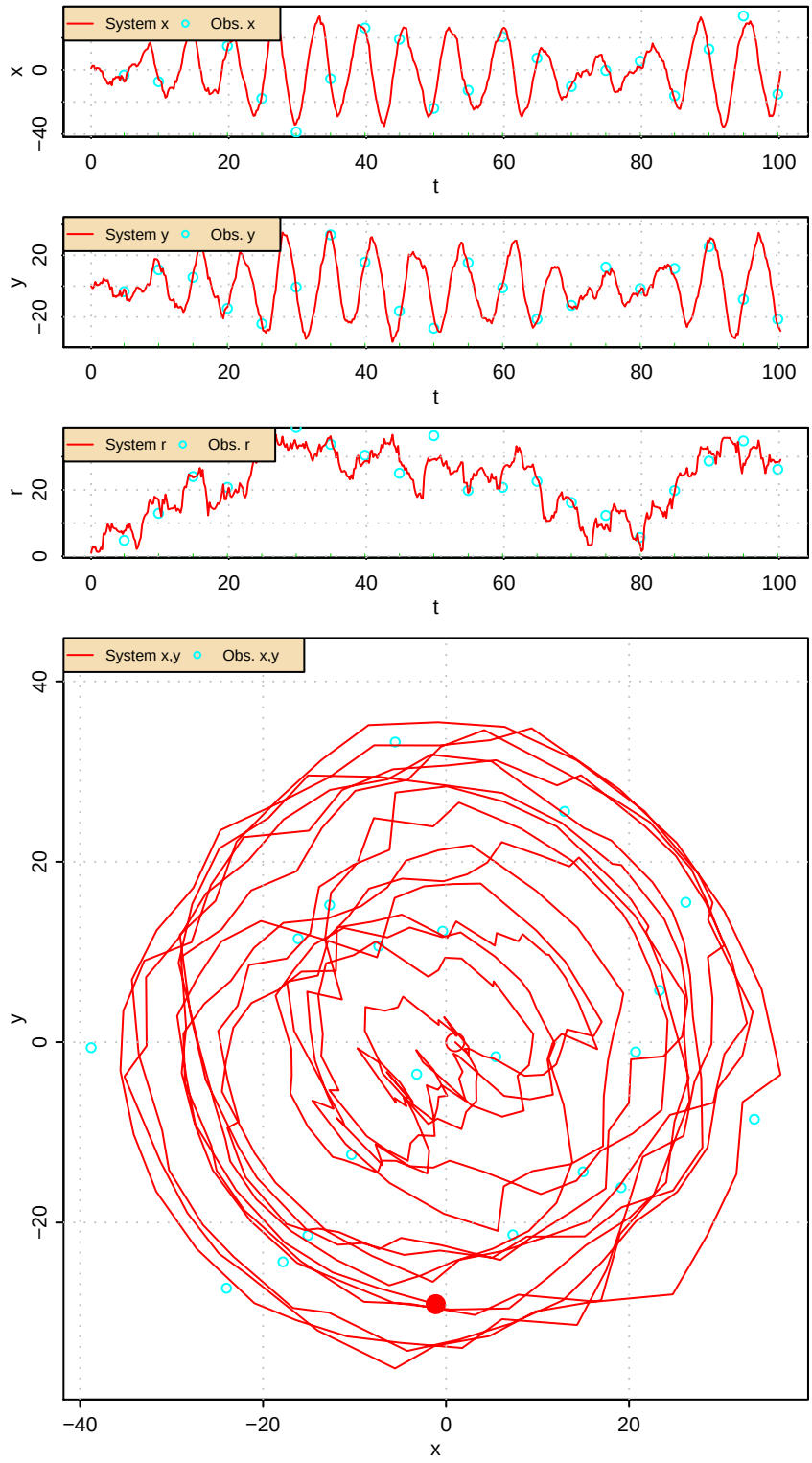


Figure 4: Forward run of the circular motion system. Same as Figure 3, but now with anticorrelated system noise.

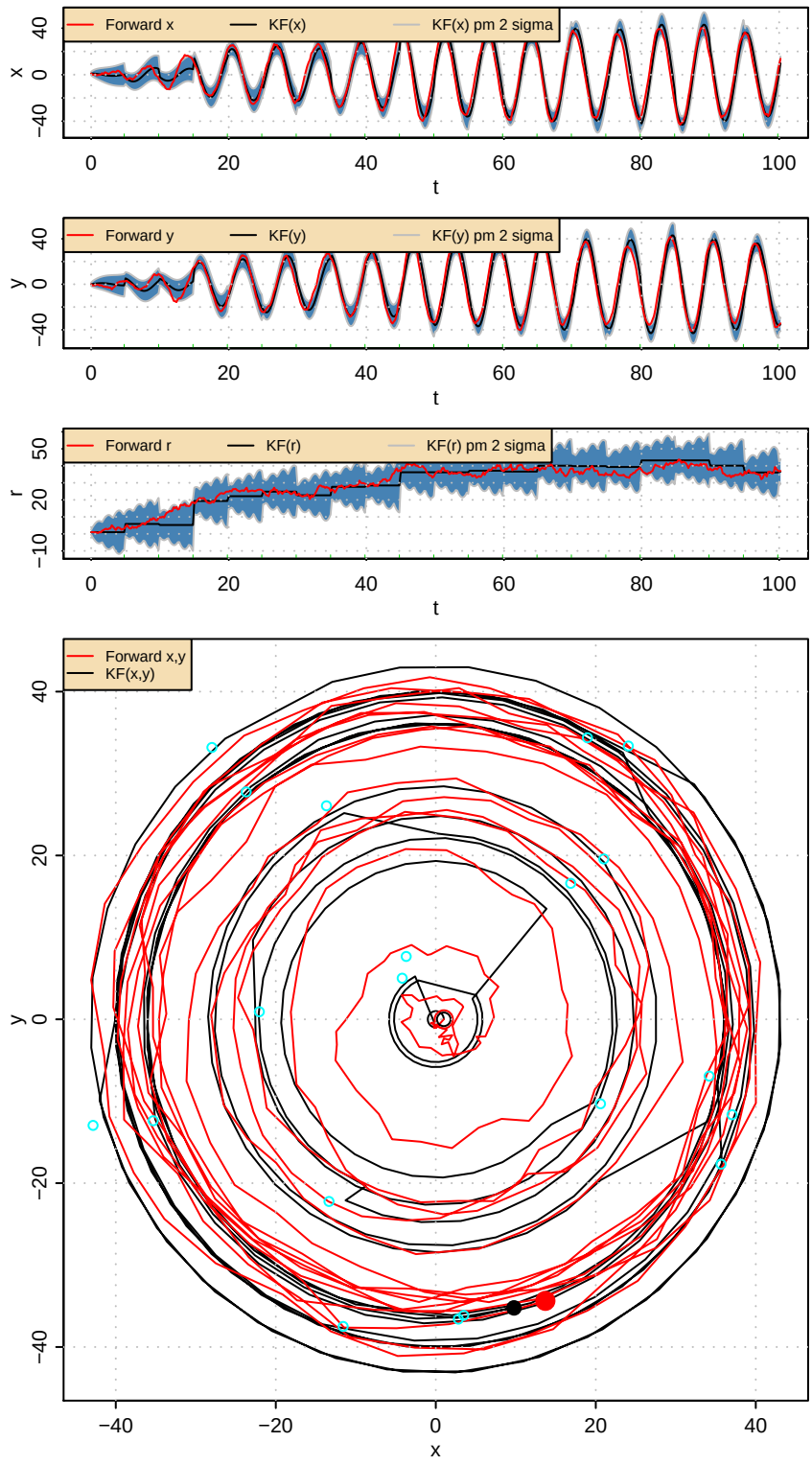


Figure 5: *Part 1 of 2.* KF analysis of the circular motion system that is summarised in Figure 3. The top three plots show estimated x, y and r vs. time. Coloured curves show the forward run; black curves show the KF estimate; and the shaded region represents $\pm 2\sigma$ error bars, as calculated from $\boldsymbol{P}(t)$. The bottom plot shows the forward run and the filtered trajectory.

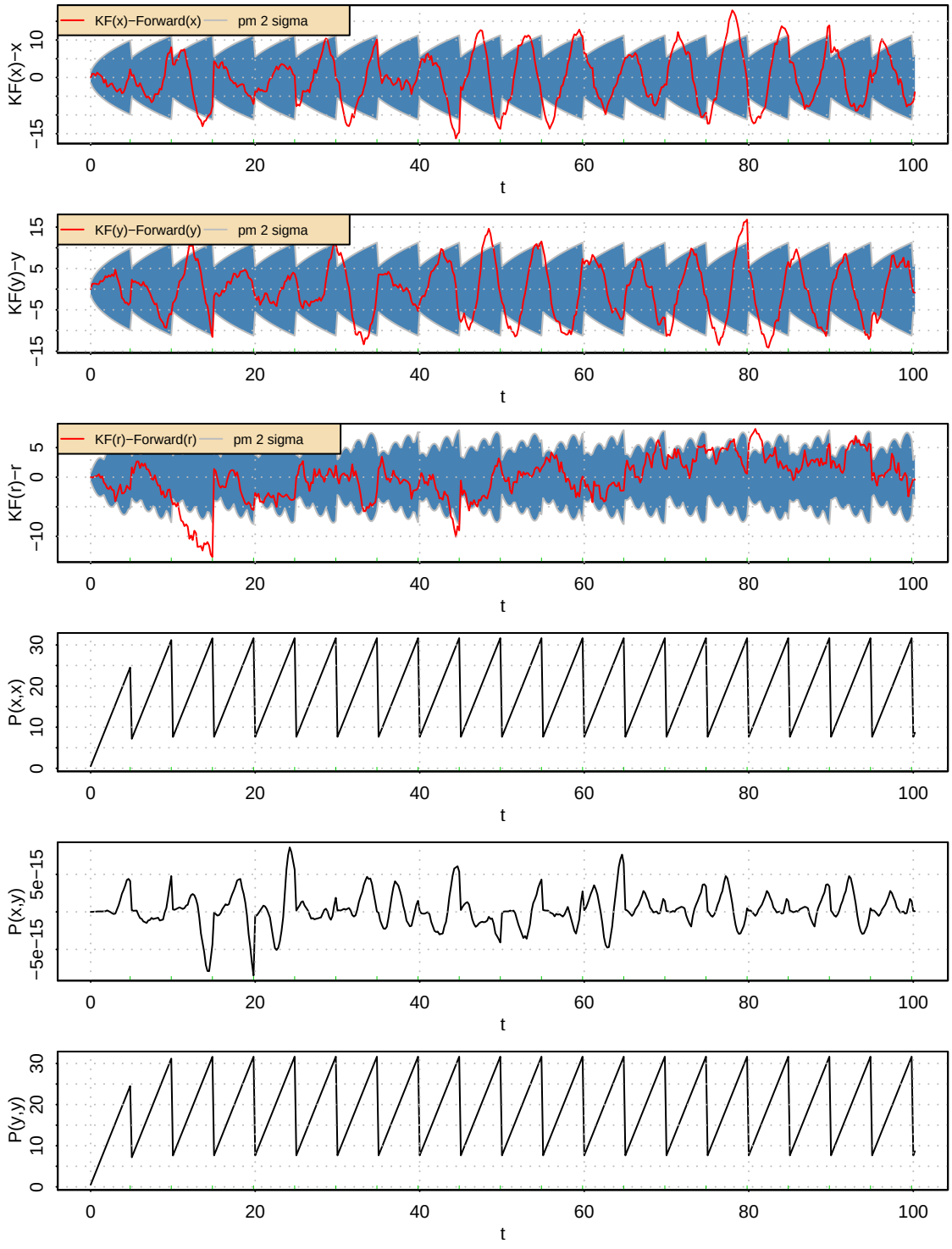


Figure 5: *Part 2 of 2.* More output from the KF job. The top three plots show estimated x, y and r vs. time, with the forward run values subtracted. The bottom three plots show entries in the evolving $\boldsymbol{P}$ matrix.

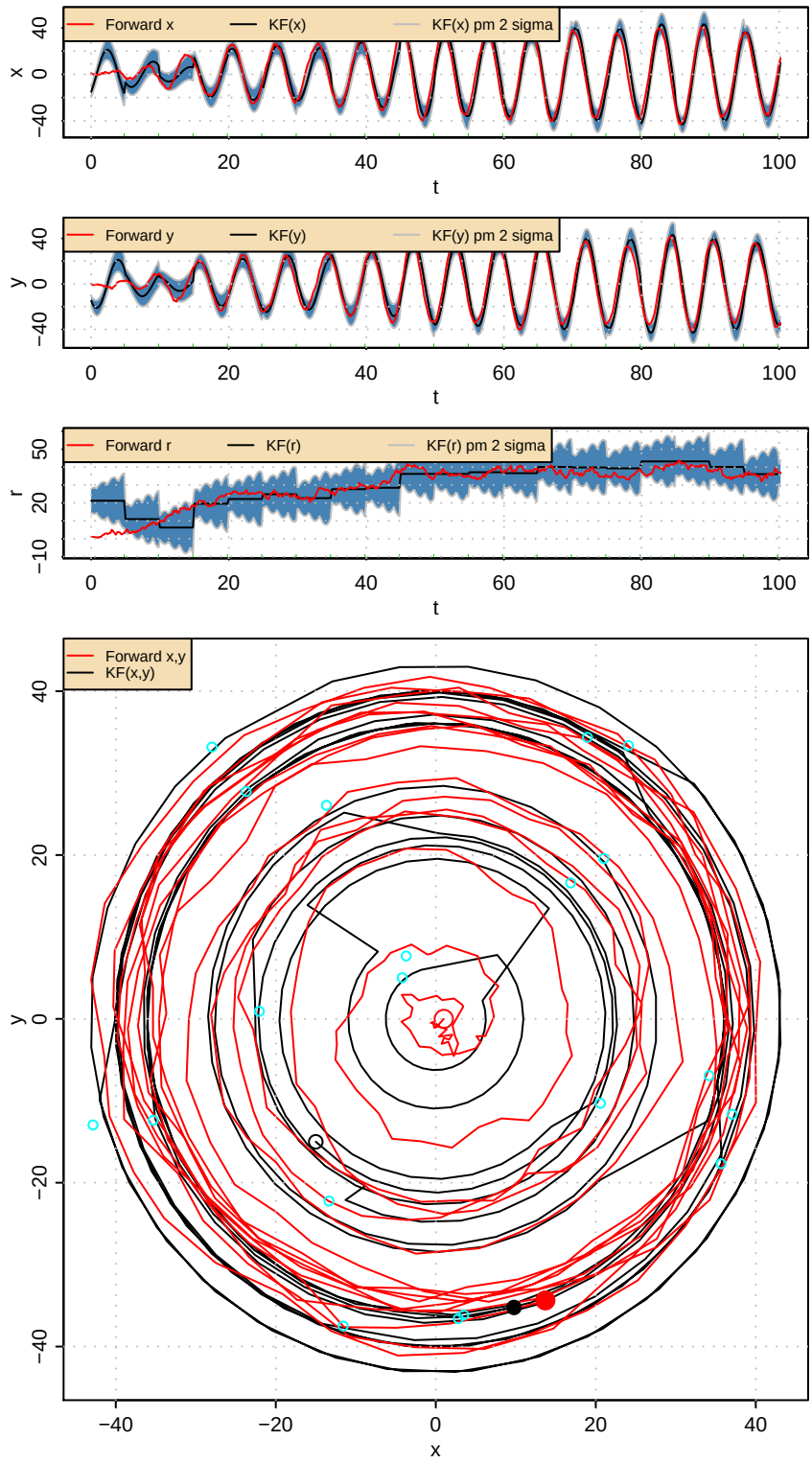


Figure 6: *Part 1 of 2.* As Figure 5 but with incorrect initialisation of the starting position and of $\boldsymbol{P}_0$.

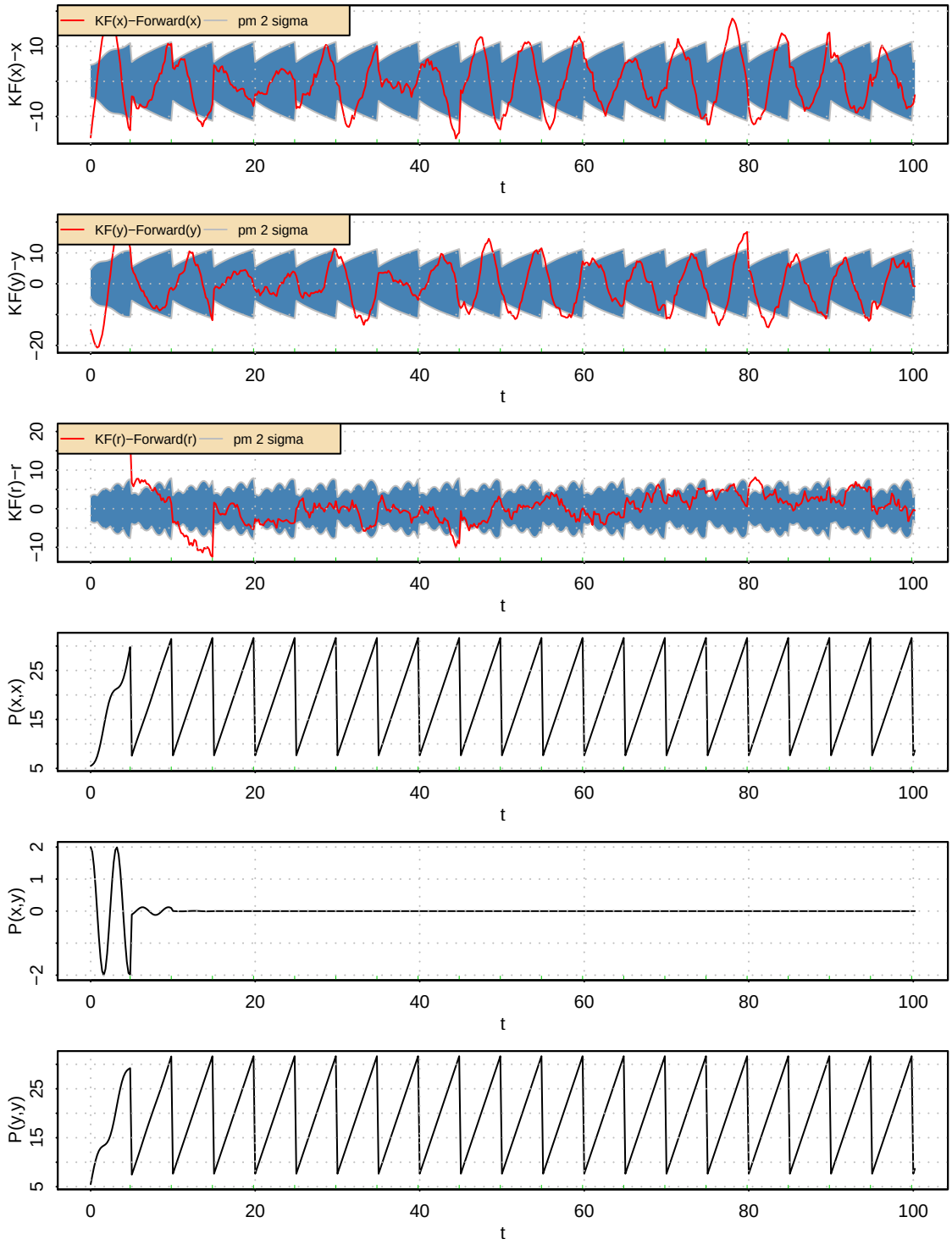


Figure 6: *Part 2 of 2.*

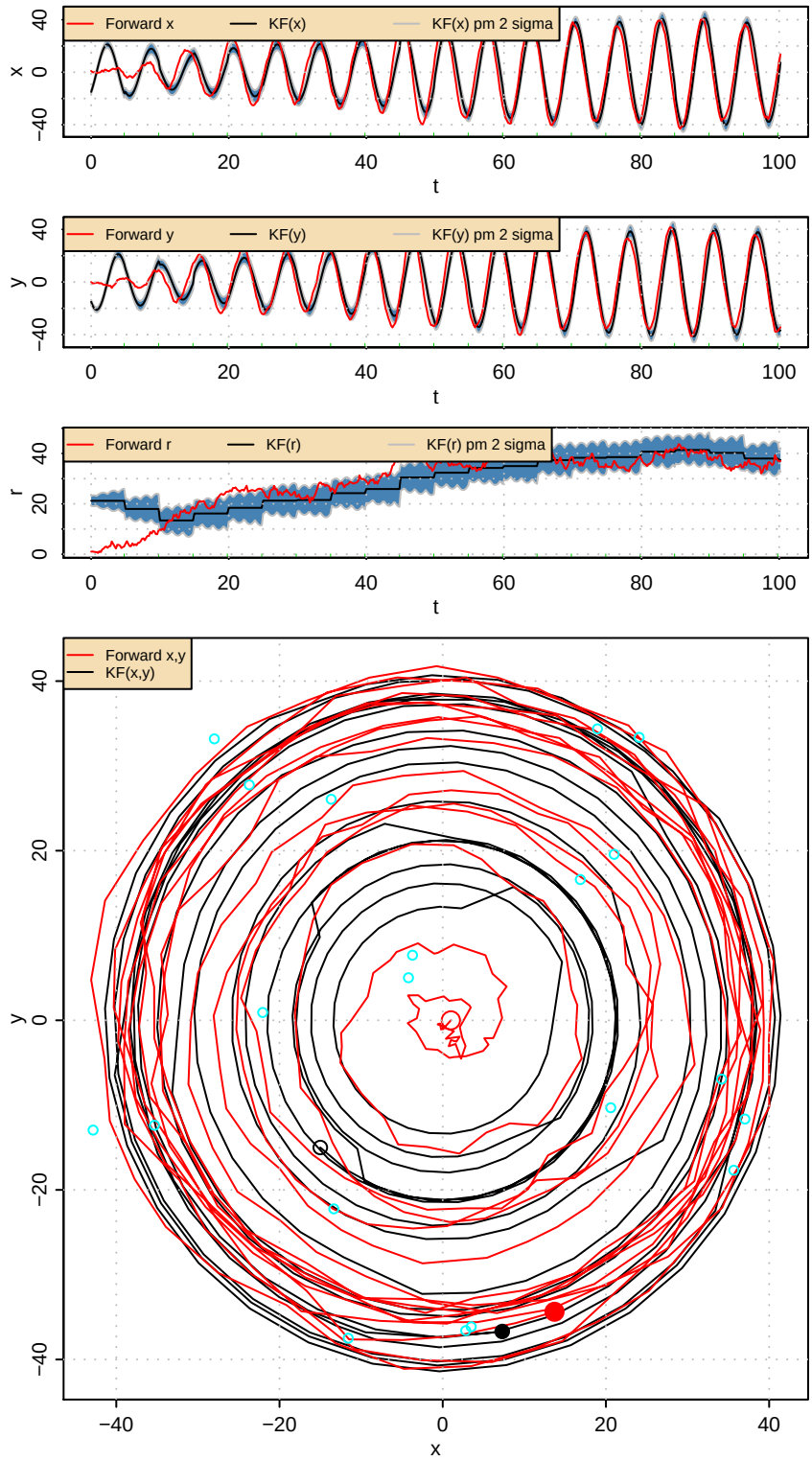


Figure 7: *Part 1 of 2.* As Figure 6. but with $\boldsymbol{Q}$ too small.

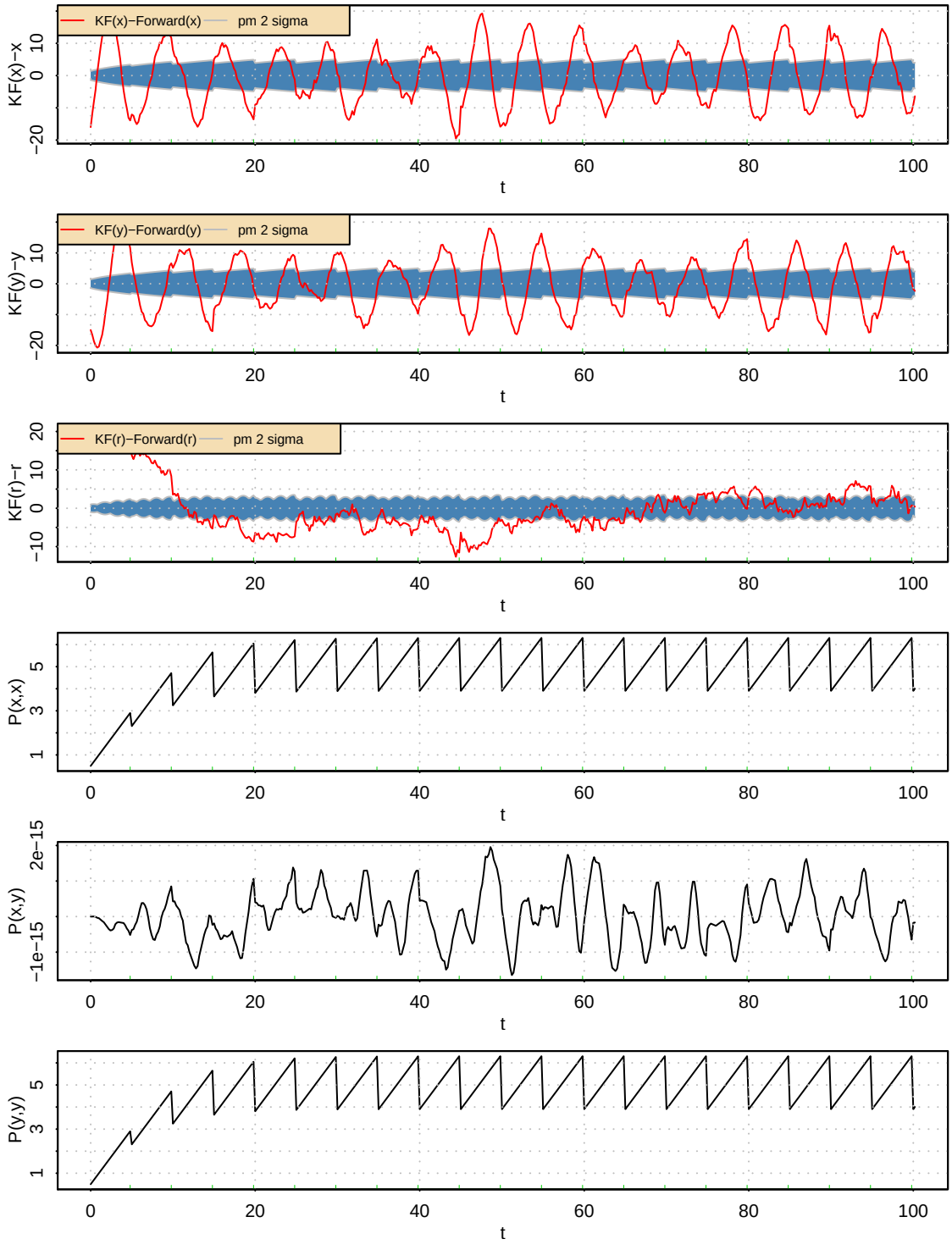


Figure 7: *Part 2 of 2.*

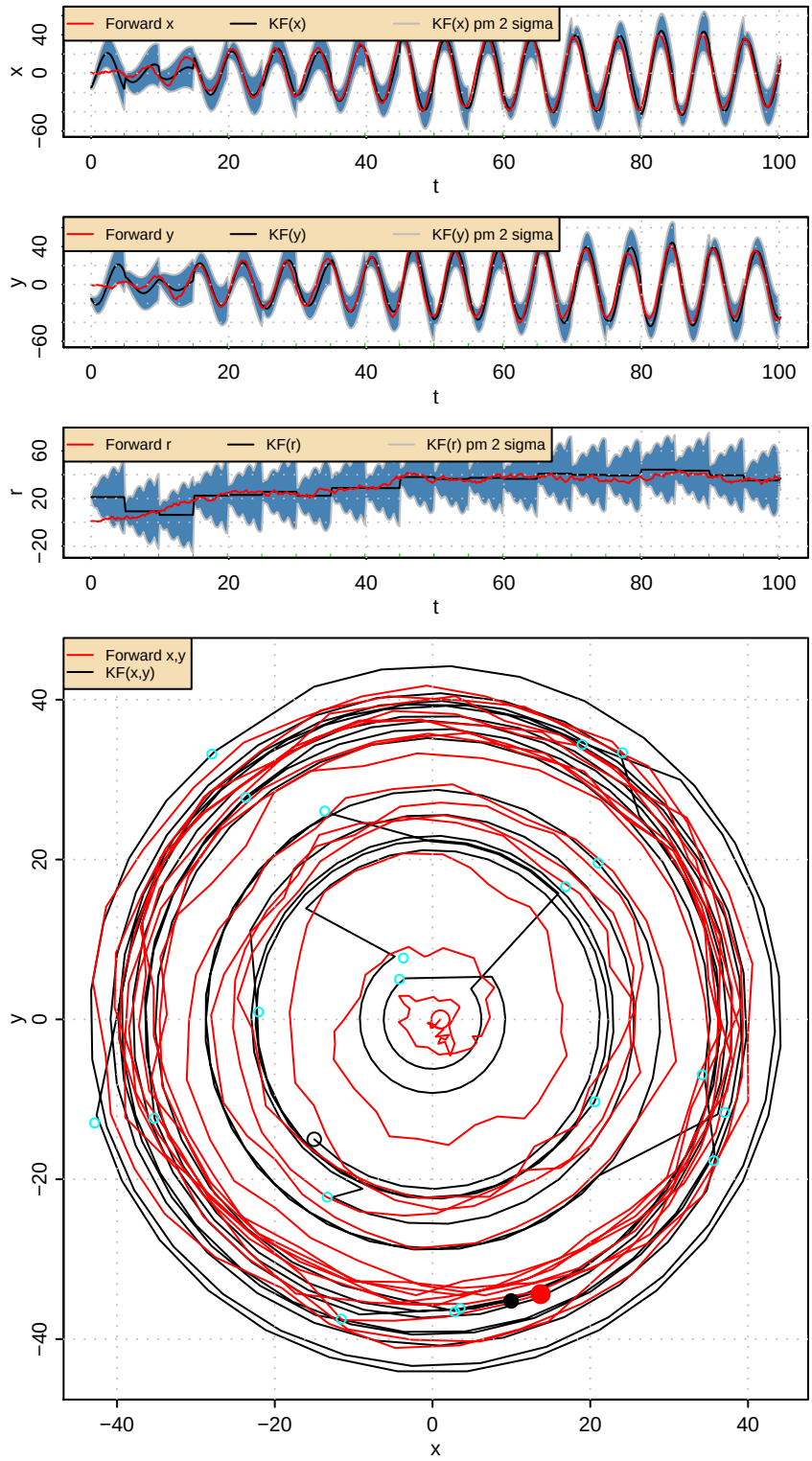


Figure 8: *Part 1 of 2.* As Figure 6. but with $\boldsymbol{Q}$ too big.

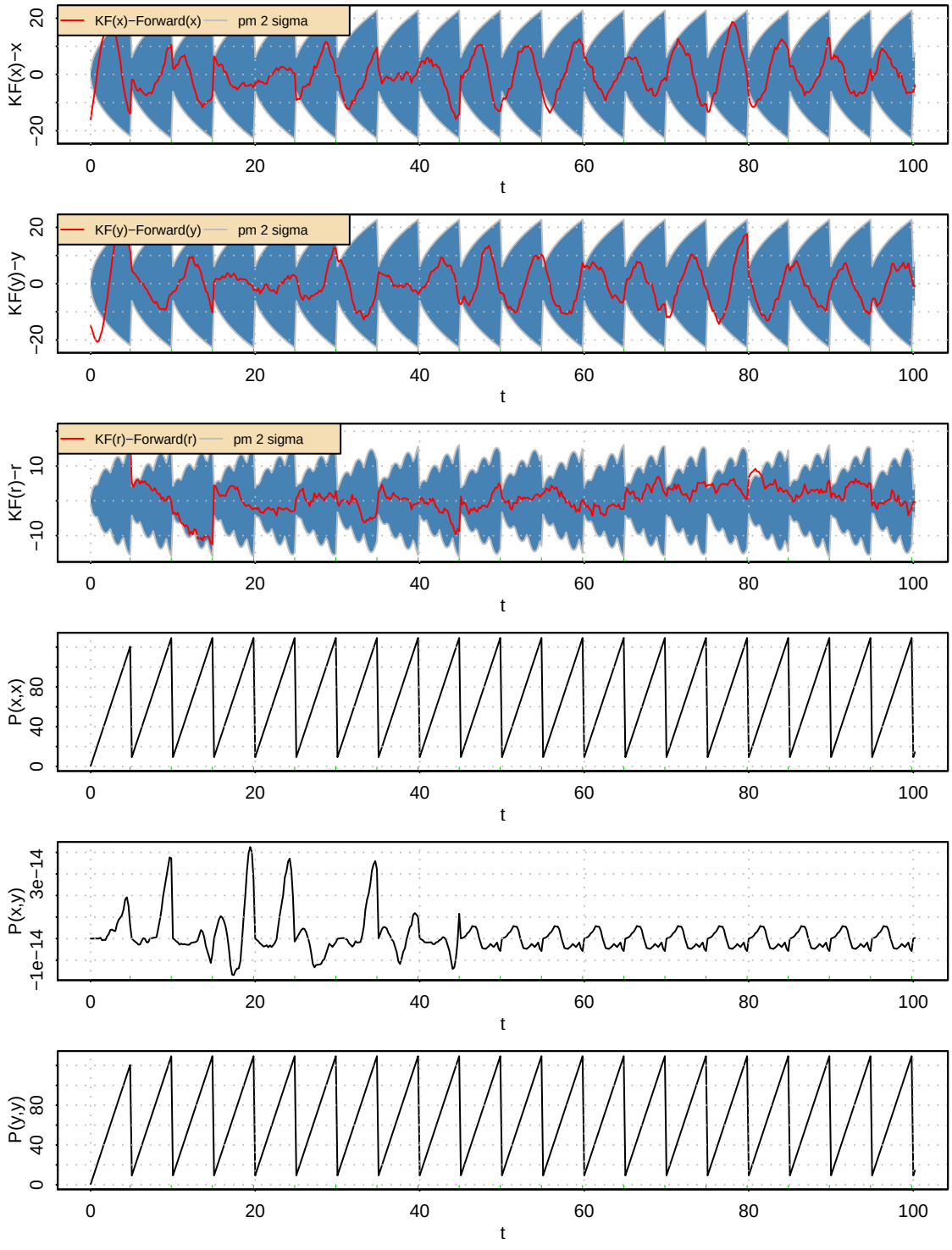


Figure 8: *Part 2 of 2.*

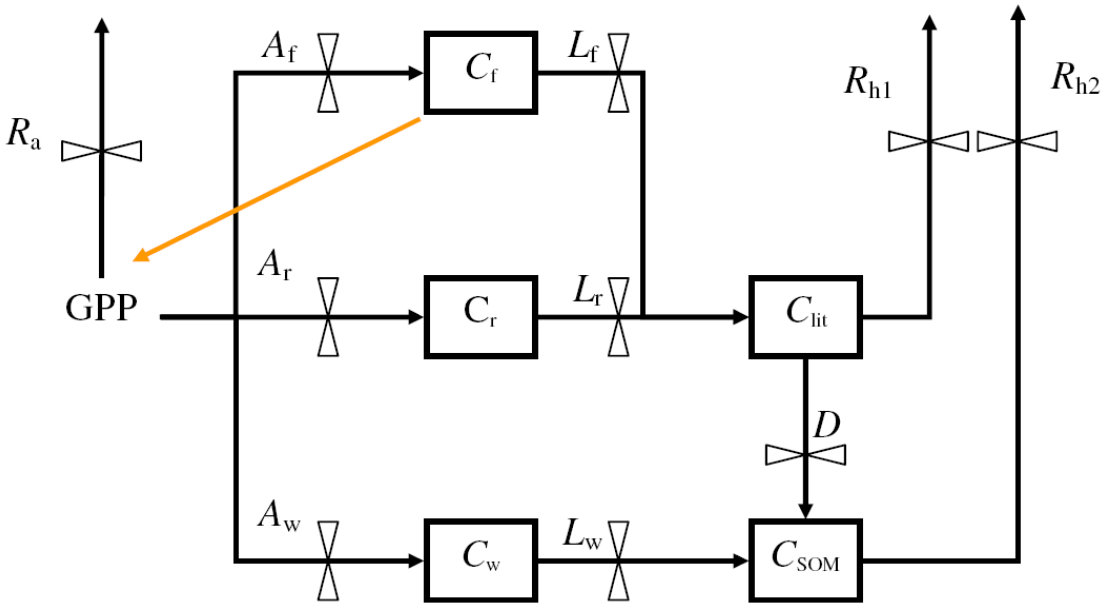


Figure 9: DALEC for evergreen forests. I scanned this diagram from the files available at [Fox]. Carbon pools are denoted by symbols in boxes: $C_{\rm f}$ is foliage; $C_{\rm r}$ is fine roots; $C_{\rm w}$ is woody stems and course roots; $C_{\rm lit}$ is litter; and $C_{\rm som}$ is soil organic matter and coarse woody debris. Carbon fluxes are indicated by symbols associated with the "bow tie" markers: R_a, R_{h1} and R_{h2} are respiration fluxes; and D is decomposition. Decomposition and respiration vary exponentially with temperature T. Gross Primary Productivity (GPP) depends on C_f, T, radiation, length of day, and C_a. It is allocated in fixed proportions to the different carbon pools. Various parameters are fixed by calibration against models or data. Units for pools are $(\mathrm{gC})\mathrm{m}^{-2}$.

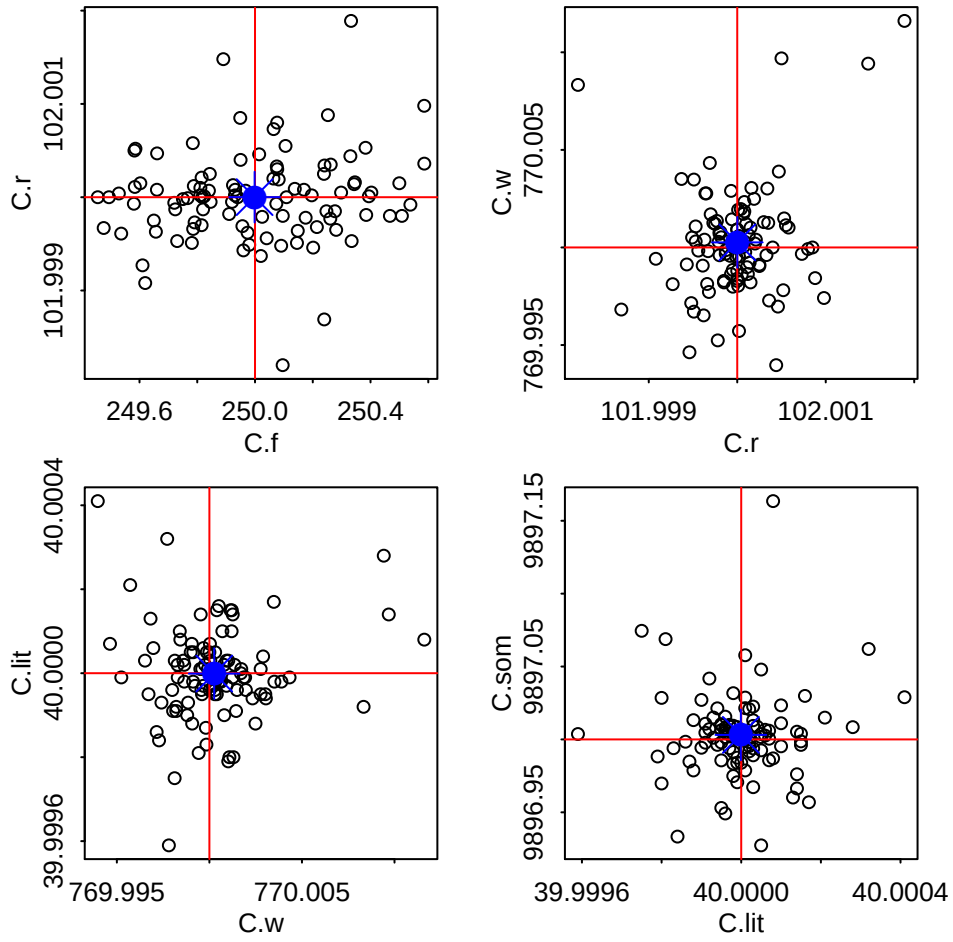


Figure 10: *Part 1 of 2.* Simple Var job to estimate initial carbon pools. The cost function had only observation terms, i.e. no background constraint. The target function was LAI(t). Each circle represents one Var job, each having a different simulated observation error. The coloured fiducial lines cross at the correct values. The coloured spider is at the centroid of the distribution of circles.

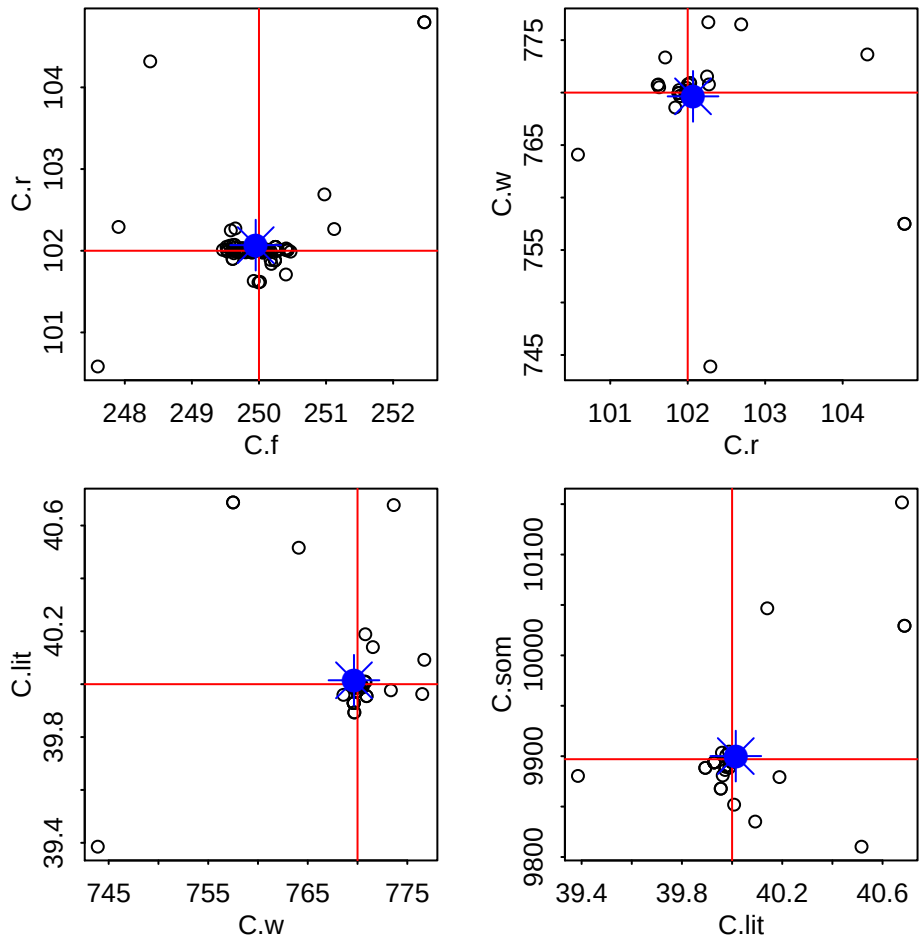


Figure 10: *Part 2 of 2.* The same, but with incorrect initialisation of the Var jobs' initial values. The scales on the axes are not the same as on the preceding page.

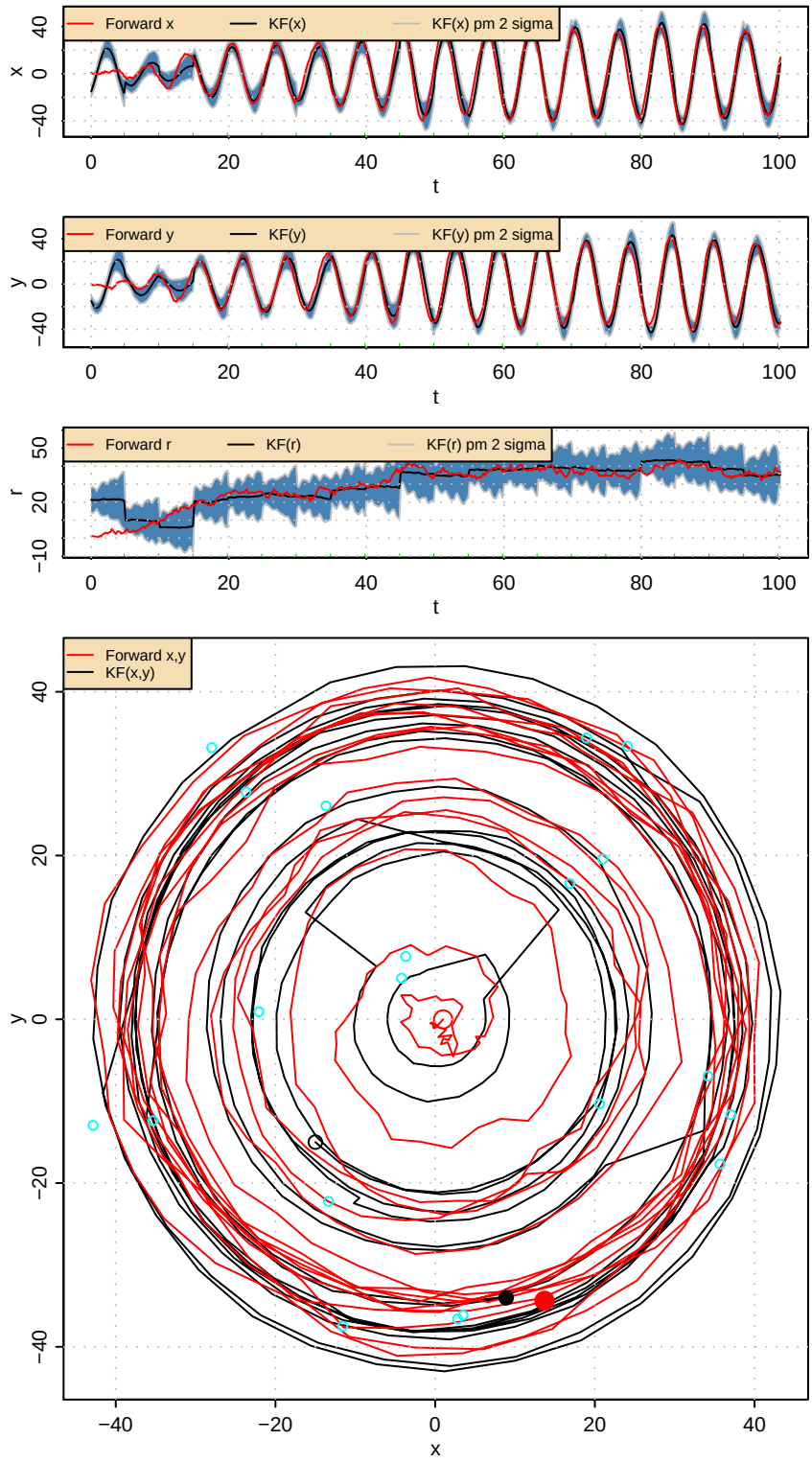


Figure 11: *Part 1 of 2.* Ensemble Kalman filter with $N_e = 25$. This repeats the KF job that led to Figure 6, but uses the EnKF instead of the plain KF.

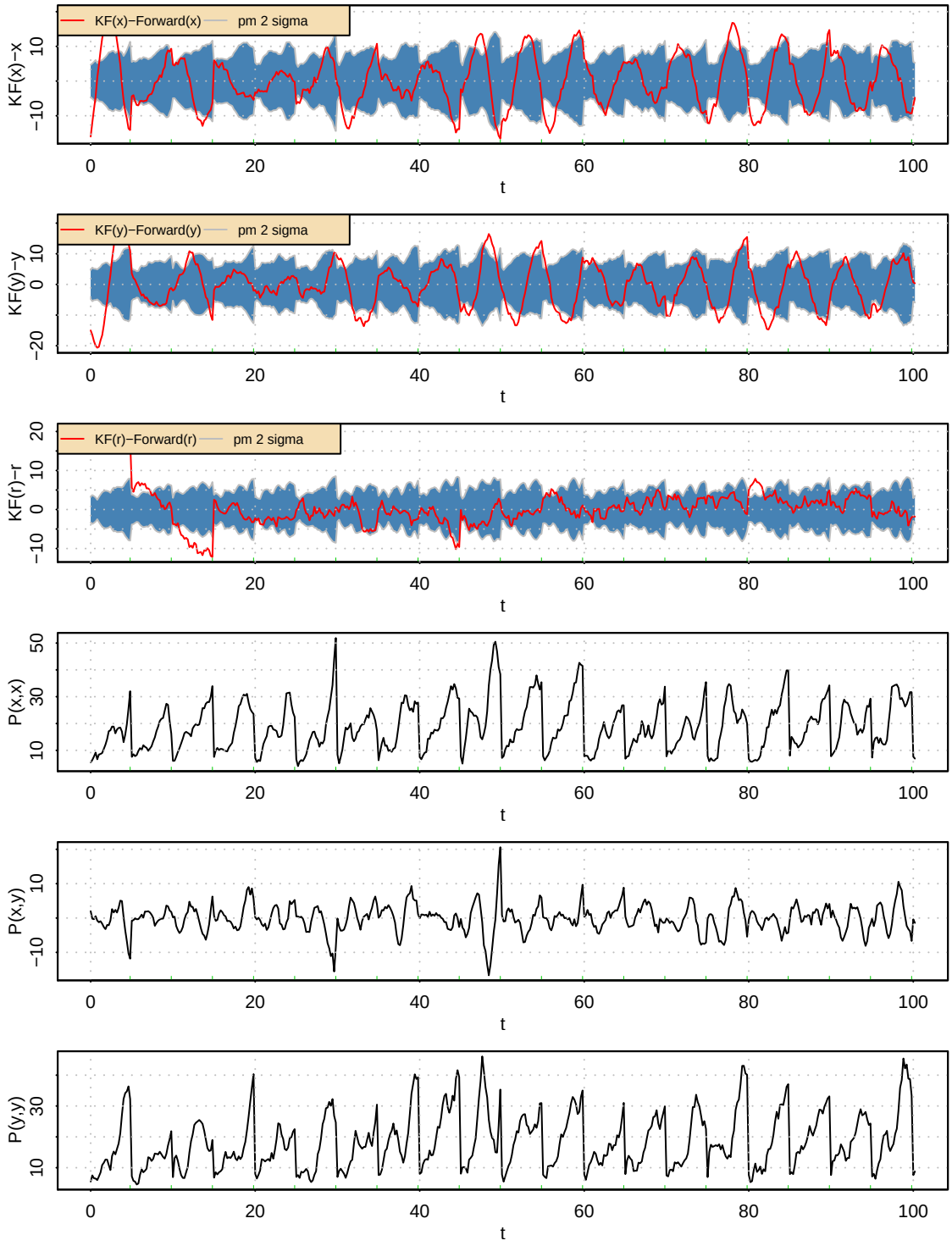


Figure 11: *Part 2 of 2.*

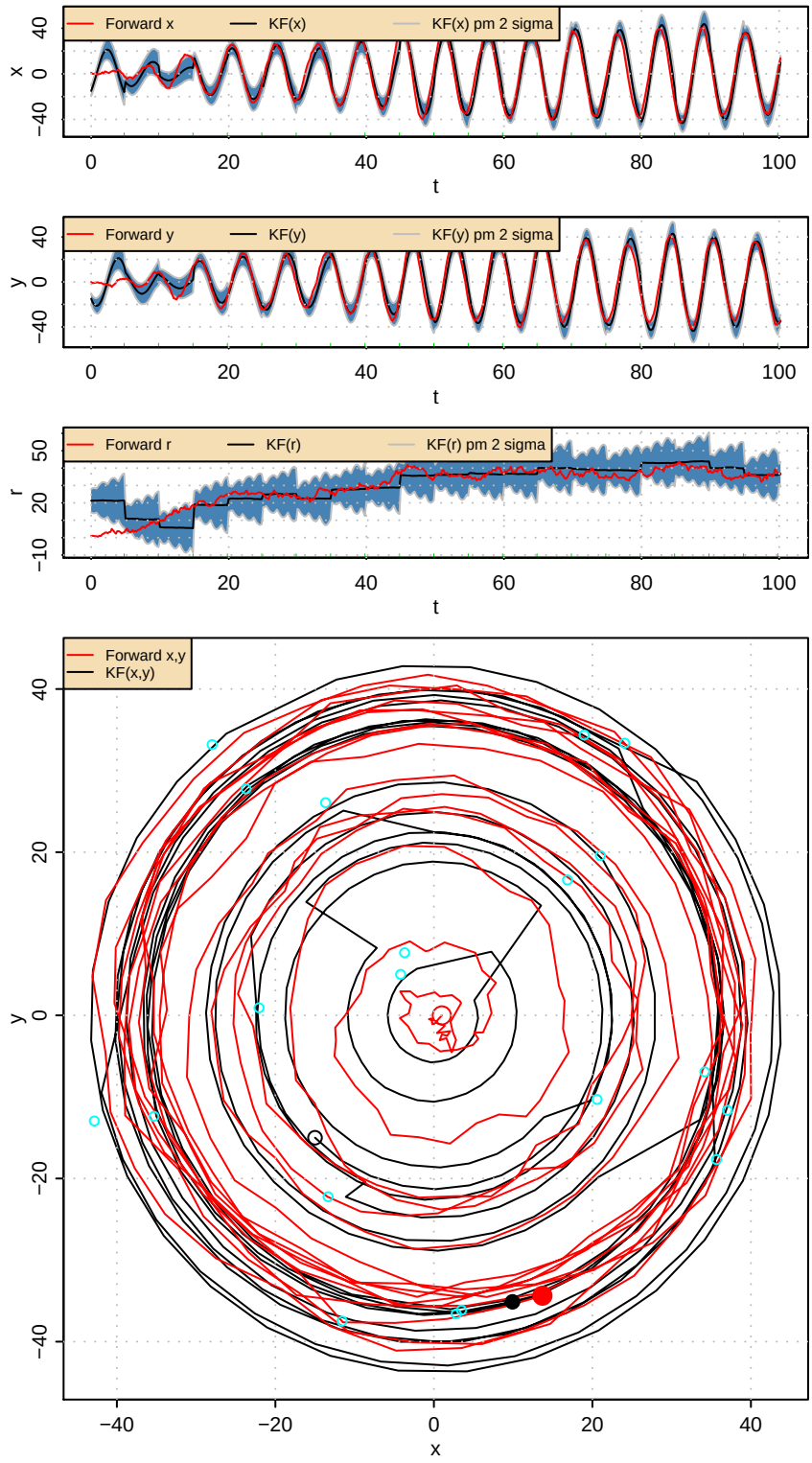


Figure 12: *Part 1 of 2.* Ensemble Kalman filter with $N_e = 250$. This repeats Figure 11, but with a larger ensemble.

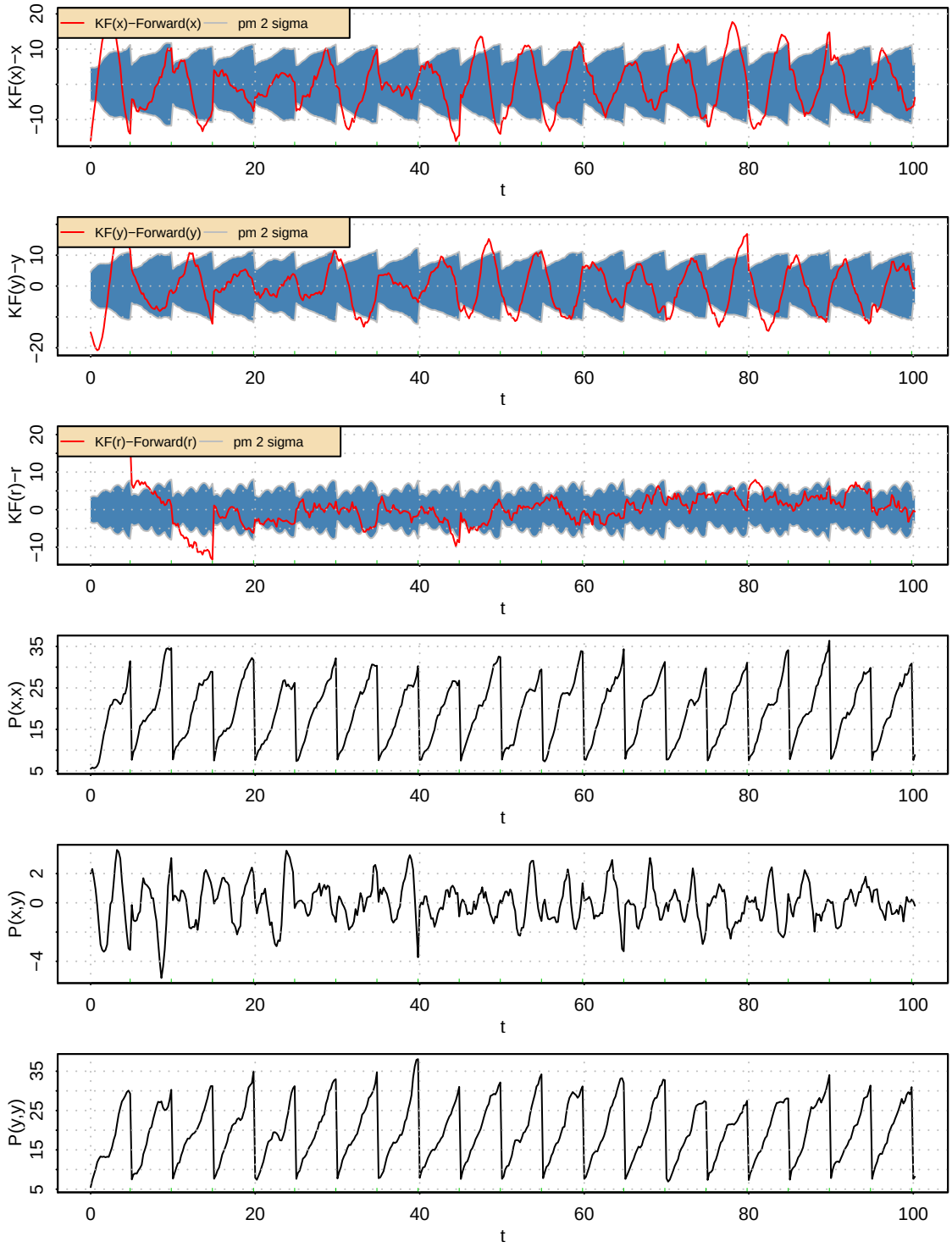


Figure 12: *Part 2 of 2.*

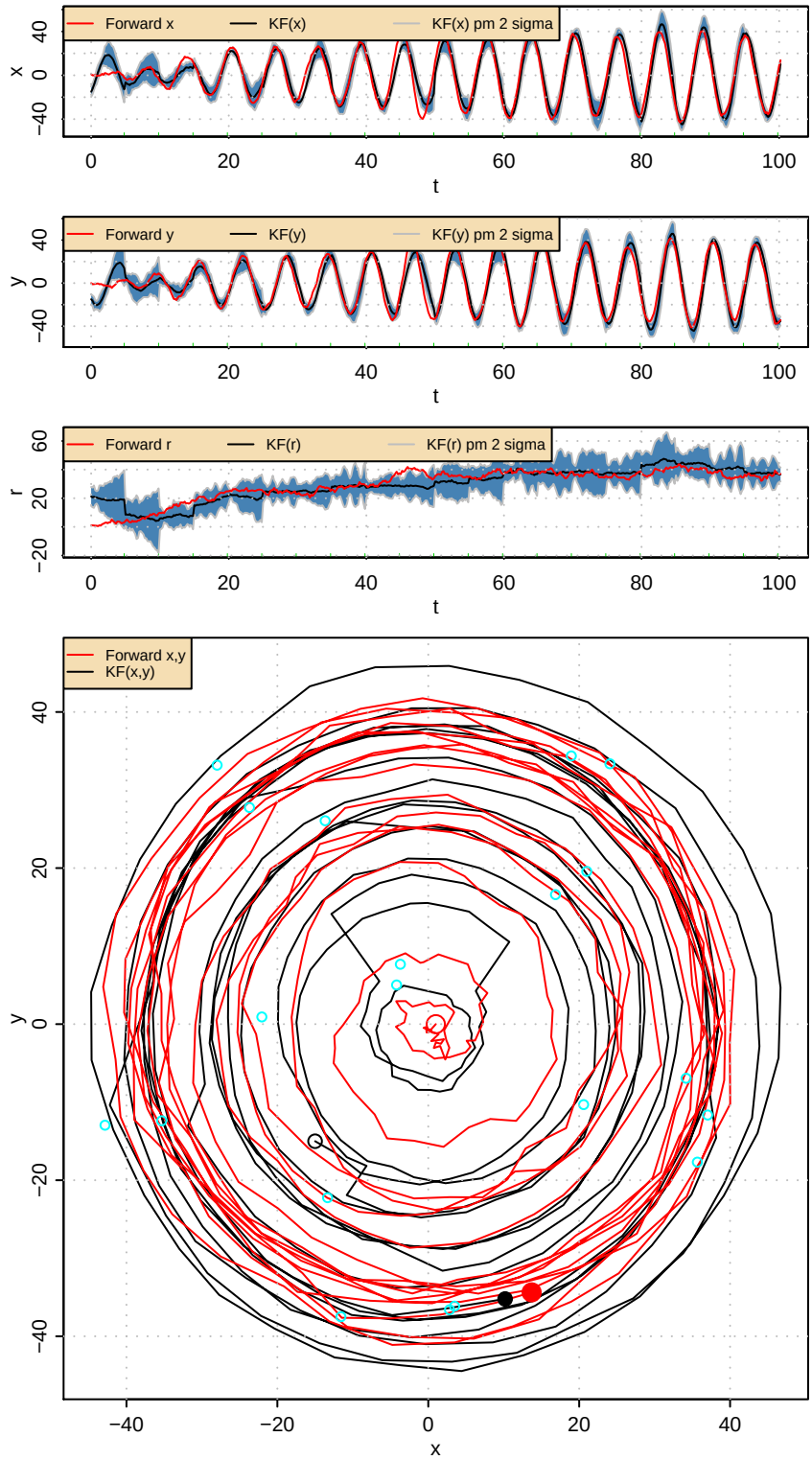


Figure 13: *Part 1 of 2.* Ensemble Kalman filter with $N_e = 5$. This repeats Figure 11, but with a smaller ensemble.

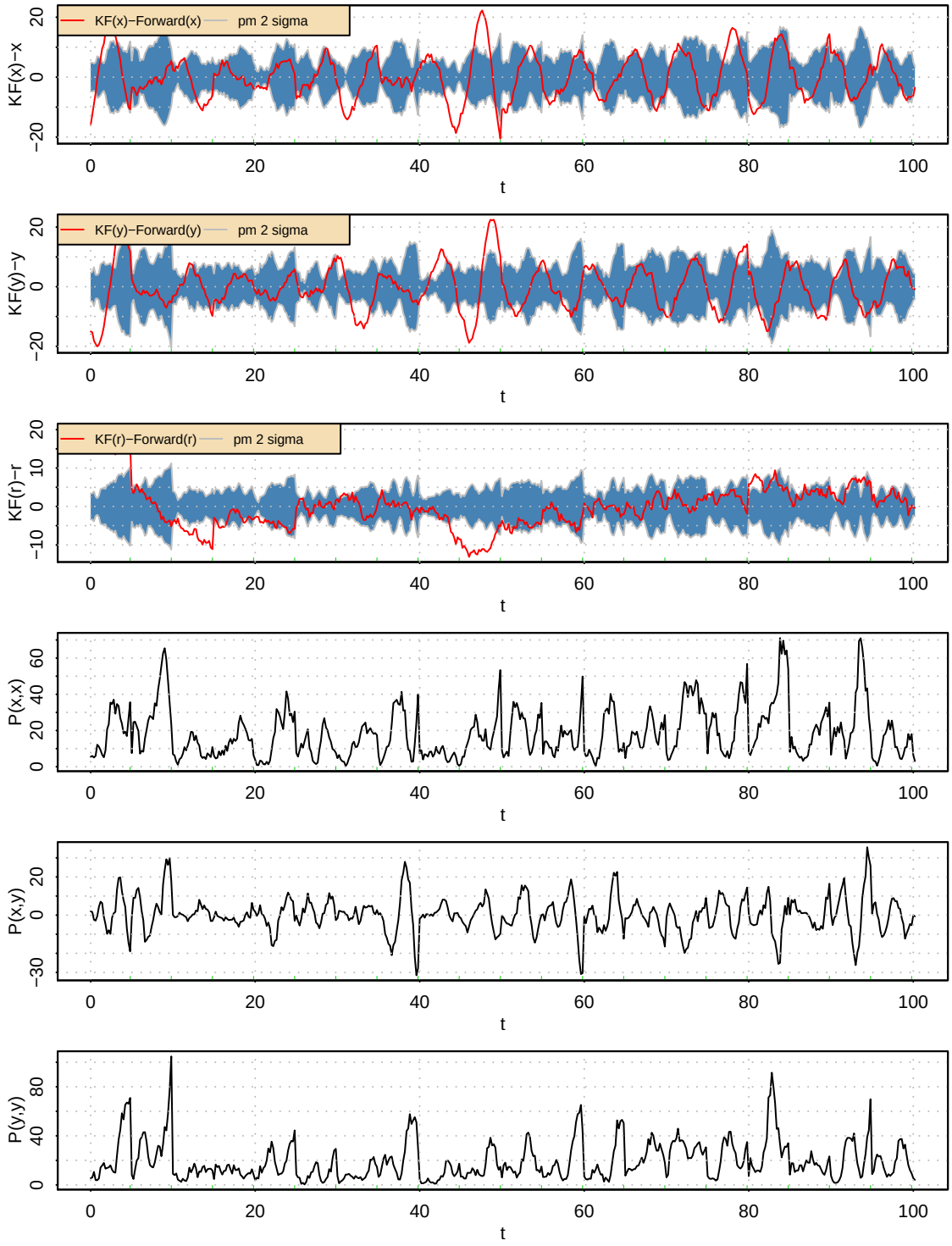


Figure 13: *Part 2 of 2.*

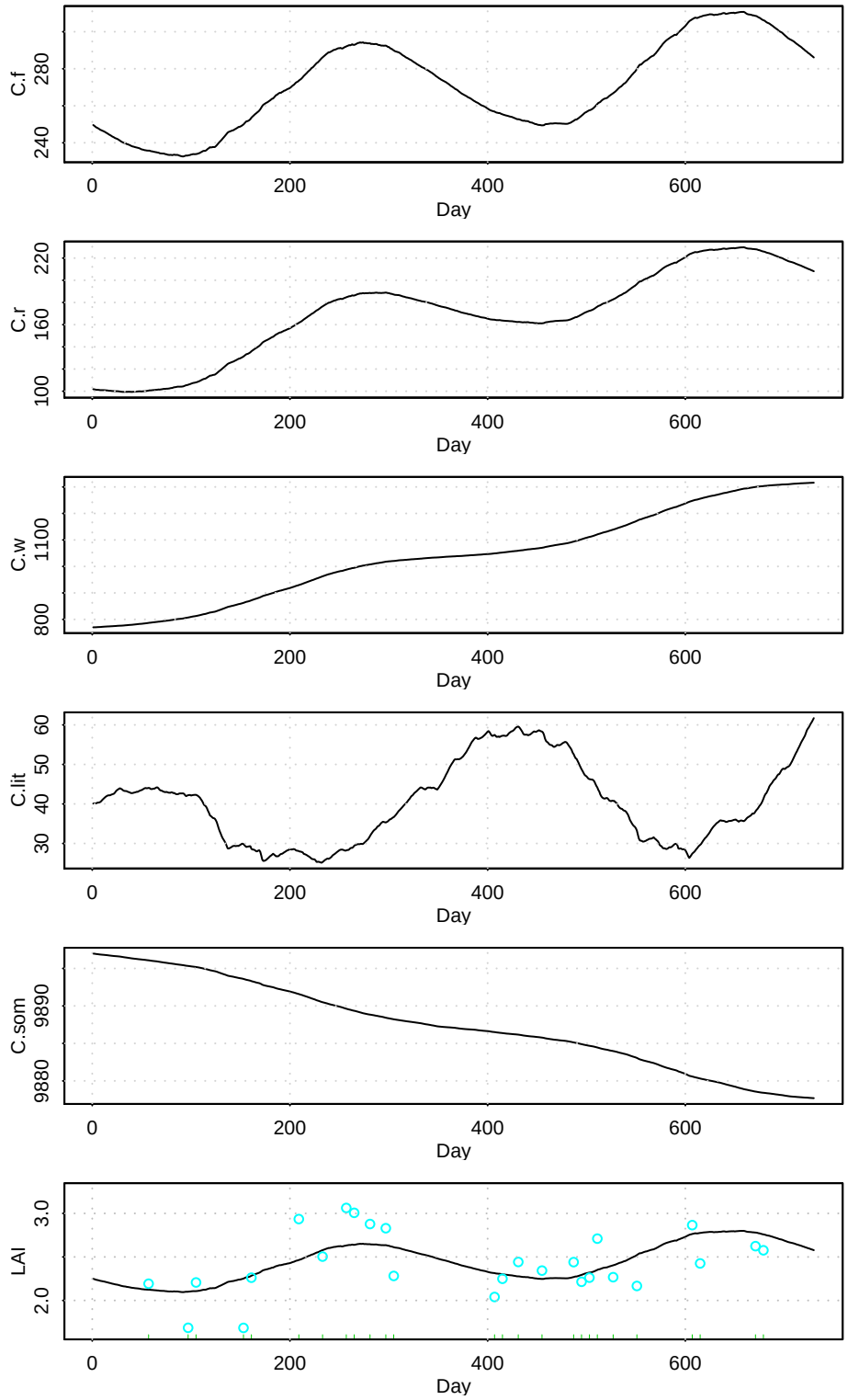


Figure 14: *Part 1 of 4.* Forward run of DALEC. The five carbon pools are shown, along with the modelled (line) and "observed" (circles) LAI. Units are (gC)m^{-2}

.

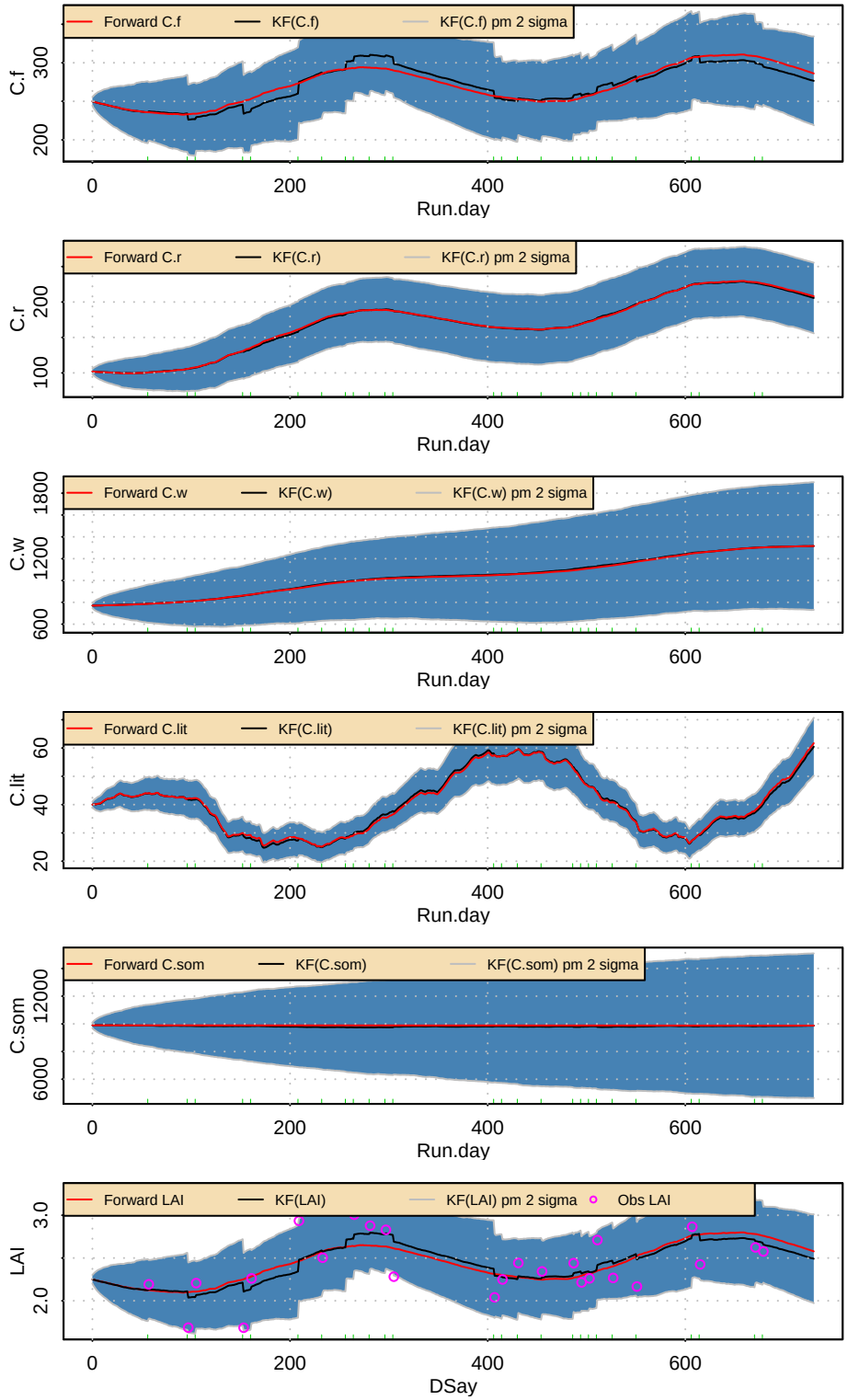


Figure 14: *Part 2 of 4.* For $N_e = 100$, with sparse LAI observations, and correct initial values of the carbon pools, we get these results. Observe that C_{som} and C_{w} are not well constrained — their error bars continually increase.

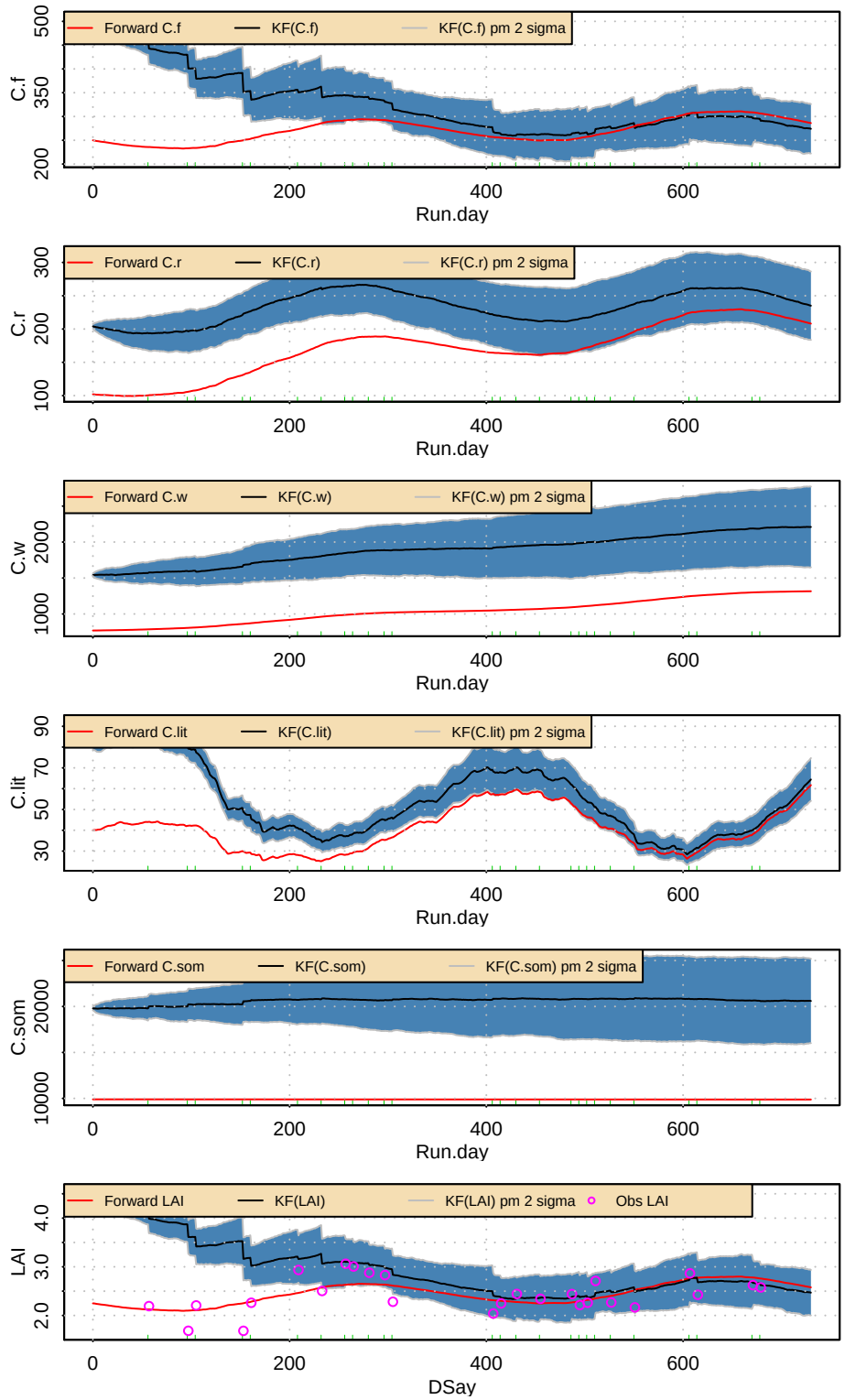


Figure 14: *Part 3 of 4.* As part 2, but here the filter is started with doubled initial conditions. The failure to correct C_{som} is now even more apparent.

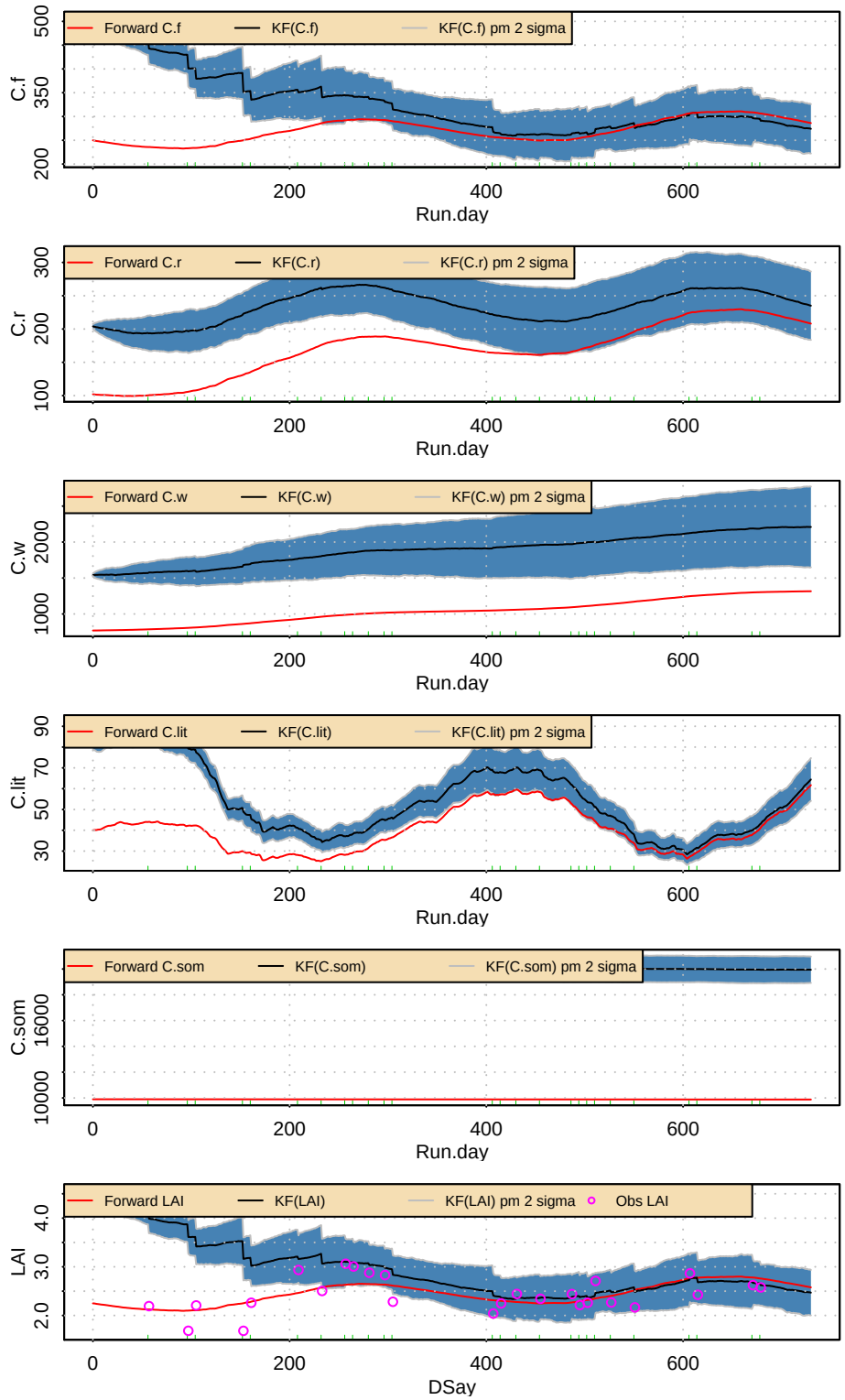


Figure 14: *Part 4 of 4.* As part 4, but now C_{som} is given a smaller variance in $\boldsymbol{Q}$. The failure to correct C_{som} is now convincing.

Met Office
FitzRoy Road, Exeter
Devon, EX1 3PB
UK

Tel: 0870 900 0100
Fax: 0870 900 5050
enquiries@metoffice.gov.uk
www.metoffice.gov.uk