EnviFlux Vn1.0: Technical and User Guide

November 14, 2025

Ross Bannister, National Centre for Earth Observation, University of Reading, Reading, UK

r.n.bannister@reading.ac.uk

1 Introduction

EnviFlux (Environmental Flux inversion tool) is a global 4DVar system to infer surface fluxes of trace gases in the Earth's atmosphere from measurements of the gases within a time window. This document describes general, technical, and operational aspects of the system including:

- How the problem is represented. This includes the structure of the state vector, the underlying grid, and the processes represented.
- The formulation of the forward model (a transport/source problem). This includes the way that the tracer transport equations are solved.
- The inverse problem (details of the 4DVar algorithm). This includes the structure of the cost function, the derivation of its gradient with respect to the state vector, and the changes of variables introduced to make this efficient.
- How to run the various master programs. This includes:
  - Generating wind fields at the suitable resolution which will drive the dynamics.
  - Generating suitable initial conditions and surface fluxes for the purposes of testing the forward model and generating data for a nature run.
  - Running the forward model.
  - Running back-trajectories.
  - Generating a-priori state vectors.
  - Generating synthetic observations.
  - Running the full 4DVar system to infer surface fluxes.

The code is designed to be relatively light and efficient, and can be run in a Linux environment (from a laptop to a supercomputer). No parallelisation is exploited.

2 How the forward problem is represented

- There are four kinds of time-step in EnviFlux:
  - Minor time-step: this is δt (or dt in the code), the time-step used by the Runge-Kutta solver in the back-trajectory part of the semi-Lagrangian advection scheme.
  - Wind file time-step (or major time-step): this is Δt (or Dt in the code), the time separation between wind files. It is also the semi-Lagrangian time step (how often the semi-Lagrangian scheme is run). Δt must be a multiple of δt.
  - Diffusion time-step: this is τ_κ (or $kappa\_dt$ in the code), the time-step of the diffusion scheme. Δt must be a multiple of τ_κ.
  - Source time-step: this is τ_ρ. This represents how frequently the source field is updated in the inverse problem.
- Let $\gamma = \Delta t/\delta t$ and is the number of minor steps per major step.
- Let there be $\gamma(T+1)$ minor time levels and $T+1$ major time levels.
- Let the augmented state vector be $\mathbf{x}$, comprising $\boldsymbol{\chi}(0)$ (the initial conditions ($t=0$) for the tracer) and $\boldsymbol{\rho}(t)$ (the surface flux at major time t):

$$\mathbf{x} = \begin{pmatrix} \boldsymbol{\chi}(0) \\ \boldsymbol{\rho} \end{pmatrix}, \qquad \text{where} \qquad \boldsymbol{\rho} = \begin{pmatrix} \boldsymbol{\rho}(0) \\ \boldsymbol{\rho}(\tau_\rho) \\ \vdots \\ \boldsymbol{\rho}(T\tau_\rho) \end{pmatrix}. \tag{1}$$

In EnviFlux, $\boldsymbol{\chi}(0)$ has units of ppb, and $\boldsymbol{\rho}$ has units of $\mu\mathrm{gm}^{-2}\mathrm{s}^{-1}$.

- The model evolution between two major time steps is done in three stages.
  - The first stage is the advection stage (performed by semi-Lagrangian (**SL**) over γ minor time-steps, labelled with k).
$$\boldsymbol{\chi}^{t+1}(k\Delta t) = \mathbf{SL}_{k\to k+1}\boldsymbol{\chi}^{t}(k\Delta t),$$
    where the argument represents the major time-step and the super/subscript represent the minor time-steps.
  - The second stage is the diffusion stage. This solves the following diffusion equation over pseudo time τ:
$$\frac{\partial \boldsymbol{\chi}^{t+1}(\tau)}{\partial \tau} = \kappa_\mathrm{h}\nabla^2_\mathrm{h}\boldsymbol{\chi}^{t+1}(\tau) + \kappa_\mathrm{v}\frac{\partial^2\boldsymbol{\chi}^{t+1}(\tau)}{\partial z^2}, \tag{2}$$
    where κ_h is the horizontal diffusion coefficient and κ_v is the vertical diffusion coefficient.
  - The third stage is the source/sink stage.
$$\boldsymbol{\chi}([k+1]\Delta t) = \mathbf{M}_{k\to k+1}\boldsymbol{\chi}^{k}(k\Delta t) + \mathbf{S}\boldsymbol{\rho}(I(k\Delta t/\tau_\rho))\,\Delta t, \tag{3}$$
    where $I(x)$ is the integer operator, used here to select the relevant flux field, $\mathbf{S}$ is the 'surface operator', which converts the 2D source field to the surface level of a 3D field and divides by $\Delta z_0 d_0$ at the surface layer (layer thickness Δz_0, surface air density d_0), and $\mathbf{M}_{k\to k+1}$ is the combined effect of the sequence of $\mathbf{SL}_{k\to k+1}$ operations and diffusion operations over a complete major time-step. For compactness, the full evolution is written as
$$\boldsymbol{\chi}([k+1]\Delta t) = \mathbf{M}^{\mathbf{x}}_{k\to k+1}\boldsymbol{\chi}(k\Delta t), \tag{4}$$
    where the $\mathbf{x}$ in the superscript indicates that the state vector itself (namely the flux part) contributes to the tracer evolution.

The combined advection/diffusion/flux equation may be written as

$$\frac{\partial \chi(t)}{\partial t} = -\mathbf{u}(t) \cdot \nabla \chi(t) + \kappa_{\rm h} \nabla^2_{\rm h} \chi(t) + k_{\rm v} \frac{\partial^2 \chi(t)}{\partial z^2} + \mathbf{S} \boldsymbol{\rho}^{I(t/\tau_\rho)}, \tag{5}$$

where κ is the diffusion coefficient (actually in the developments below, this has different values for horizontal and vertical diffusions).

3 The inverse problem

3.1 The cost function

EnviFlux works by minimising the following cost function:

$$\begin{aligned} J_{\delta\mathbf{x}}(\delta\mathbf{x}) &= \frac{1}{2}\delta\mathbf{x}^{\mathsf{T}}\mathbf{B}^{-1}\delta\mathbf{x} + \\ & \frac{1}{2}\sum_{k=0}^{N_T}\left(\mathbf{y}_k - \mathbf{H}_k\mathbf{M}_{0\to k}^{\mathbf{x}^{\rm b}+\delta\mathbf{x}}\left[\boldsymbol{\chi}^{\rm b}(0)+\delta\boldsymbol{\chi}\right]\right)^{\mathsf{T}}\mathbf{R}_k^{-1}\left(\mathbf{y}_k - \mathbf{H}_k\mathbf{M}_{0\to k}^{\mathbf{x}^{\rm b}+\delta\mathbf{x}}\left[\boldsymbol{\chi}^{\rm b}(0)+\delta\boldsymbol{\chi}\right]\right), \end{aligned} \tag{6}$$

where $\delta\mathbf{x}$ is the proposed analysis increment with tracer component $\delta\boldsymbol{\chi}$ (the actual analysis increment is the one that minimises $J_{\delta\mathbf{x}}(\delta\mathbf{x})$), $\mathbf{B}$ is the background error covariance matrix, k is the major time-step index, $\mathbf{H}_k$ is the observation operator for time-step k, $\mathbf{M}_{0\to k}^{\mathbf{x}^{\rm b}+\delta\mathbf{x}}$ is the compact form of the evolution operator (see Eq. (4)), and $\mathbf{R}_k$ is the observation error covariance matrix for time-step k. The observation error covariance matrix is set-up for uncorrelated observation errors at present. $\delta\mathbf{x}$ is an increment and is related to the full state by $\mathbf{x} = \mathbf{x}^{\rm b} + \delta\mathbf{x}$, where $\mathbf{x}^{\rm b}$ is the background state. The analysis, $\mathbf{x}^{\rm a}$, corresponds to the special value of $\delta\mathbf{x}$ that minimises $J_{\delta\mathbf{x}}(\delta\mathbf{x})$ (call this special value $\delta\mathbf{x}^{\rm a}$). The analysis is then $\mathbf{x}^{\rm a} = \mathbf{x}^{\rm b} + \delta\mathbf{x}^{\rm a}$. Even though the notation in Eq. (6) implies that observations are at major time-steps only, EnviFlux allows observations to be made at intermediate times, which are treated with linear interpolation (see Sect. 3.2).

The gradient of the cost function is

$$\begin{aligned} \nabla_{\delta\mathbf{x}}J(\delta\mathbf{x}) &= \mathbf{B}^{-1}\delta\mathbf{x} + \\ & \sum_{k=0}^{N_T}\left(\mathbf{M}_{0\to k}^{\mathbf{x}^{\rm b}+\delta\mathbf{x}}\right)^{\mathsf{T}}\mathbf{R}_k^{-1}\left(\mathbf{y}_k - \mathbf{H}_k\mathbf{M}_{0\to k}^{\mathbf{x}^{\rm b}+\delta\mathbf{x}}\left[\boldsymbol{\chi}^{\rm b}(0)+\delta\boldsymbol{\chi}\right]\right), \end{aligned} \tag{7}$$

which is used by EnviFlux's conjugate gradient-based descent algorithm ([?, ?]) to minimise the cost function. A version of the gradient calculation that is more efficient to compute than (7) is used in the code, which is documented in Sect. 3.2.

3.2 Intermediate observation times and an efficient gradient computation

The cost function, Eq. (6), and the gradient, Eq. (7) are written above in their simplest forms. In the EnviFlux code there are two differences that are not evident in these equations. The first is that observations may be made at arbitrary times (and not just at the major time-steps). The second is that the gradient in Eq. (7) is not efficient as some factorisation can be done so that a separate adjoint of $\mathbf{M}_{0\to k}^{\mathbf{x}^{\rm b}+\delta\mathbf{x}}$ does not need to be done for each k.

3.2.1 Intermediate observation times

To allow intermediate observation times, define a new operator, $\boldsymbol{\mathcal{H}}_k$ that acts on the state between time-steps $k-1$ and k (in order to do an interpolations in time, and to allow for observations of the flux itself).

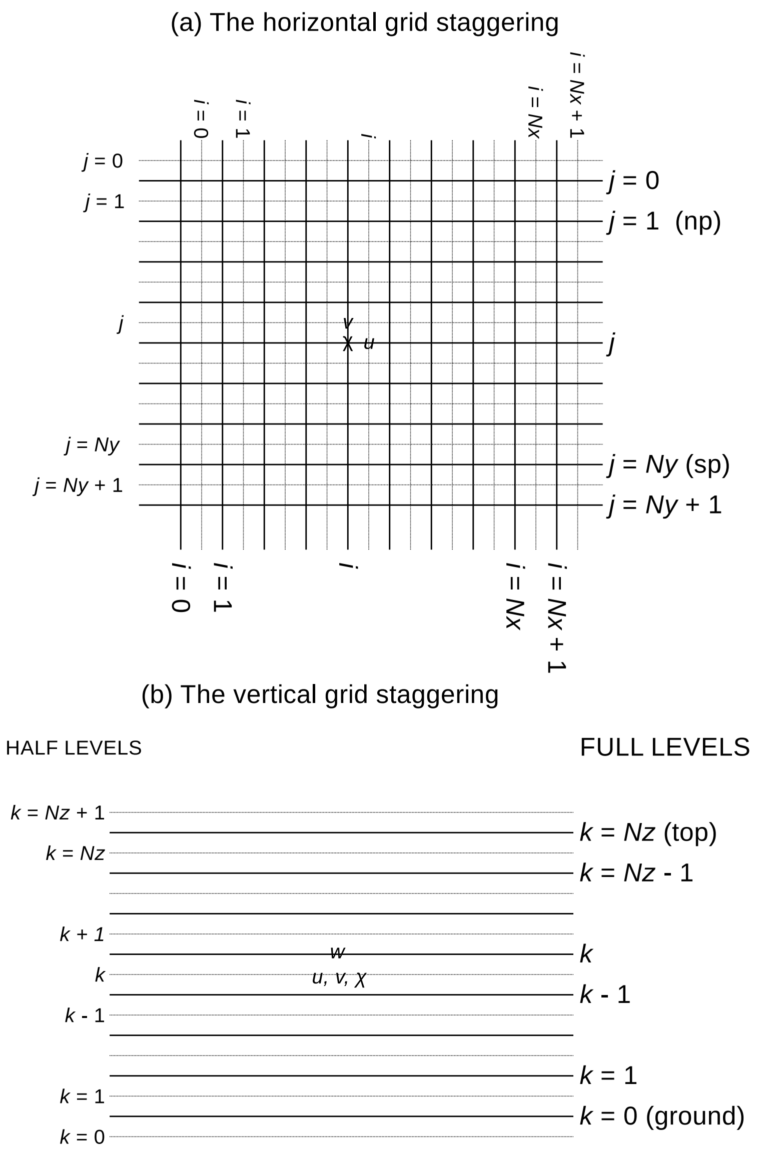

Figure 1: Working grid for the system. Panel (a) is for the horizontal grid. The labels for the u-latitudes are shown on the right of the panel and for the v-latitudes on the left. The labels for the v-longitudes are shown at the bottom and for the u-longitudes at the top. Panel (b) is for the vertical grid. The labels for the u, v levels (half-levels) are shown on the left and for the w-levels (full-levels) on the right. The tracer χ shares its longitude positions on the grid with v, its latitude positions with u and its level positions with u and v. The grid staggering is 'Arakawa C' in the horizontal and 'Charney–Phillips' grid in the vertical.

To do this we introduce the state labeled by a capital $\mathbf{X}(k\Delta t)$, which contains tracer pairs ($\boldsymbol{\chi}([k-1]\Delta t)$ and $\boldsymbol{\chi}(k\Delta t)$), and the fluxes:

$$\mathbf{X}(k\Delta t) = \begin{pmatrix} \boldsymbol{\chi}([k-1]\Delta t) \\ \boldsymbol{\chi}(k\Delta t) \\ \boldsymbol{\rho} \end{pmatrix}, \tag{8}$$

with

$$\mathbf{X}(0) = \begin{pmatrix} \mathbf{0} \\ \boldsymbol{\chi}(0) \\ \boldsymbol{\rho} \end{pmatrix}. \tag{9}$$

This $\mathbf{X}(0)$ is different from $\mathbf{x}$ in (1), but $\mathbf{X}(0)$ can be constructed from $\mathbf{x}$ as follows:

$$\mathbf{X}(0) = \mathbf{C}\mathbf{x} = \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \mathbf{x}. \tag{10}$$

The observation operator acts as follows:

$$\boldsymbol{\mathcal{H}}_k \mathbf{X}_k = \boldsymbol{\mathcal{H}}_k \begin{pmatrix} \boldsymbol{\chi}([k-1]\Delta t) \\ \boldsymbol{\chi}(k\Delta t) \\ \boldsymbol{\rho} \end{pmatrix}. \tag{11}$$

The time-stepping of the state vector is defined as

$$\mathbf{X}(k\Delta t) = \boldsymbol{\mathcal{M}}_{0\to k}\mathbf{X}(0) = \boldsymbol{\mathcal{M}}_{k-1\to k}\cdots\boldsymbol{\mathcal{M}}_{1\to 2}\boldsymbol{\mathcal{M}}_{0\to 1}\mathbf{X}(0). \tag{12}$$

This (calligraphic $\boldsymbol{\mathcal{M}}_{k\to k+1}$) propagates the tracer pairs, but keeps the fluxes in the state vector. In terms of the propagation of the tracer $\boldsymbol{\chi}(k\Delta t)$ to $\boldsymbol{\chi}([k+1]\Delta t)$, the time-stepping is (Roman $\mathbf{M}_{k\to k+1}$)

$$\delta\boldsymbol{\chi}([k+1]\Delta t) = \mathbf{M}_{k\to k+1}\delta\boldsymbol{\chi}(k\Delta t) + \mathbf{S}\boldsymbol{\rho}^{I(k\Delta t/\gamma)}\Delta t, \tag{13}$$

where $\mathbf{S}$ is defined after Eq. (3) and the function $I(\bullet)$ integerises its argument, so time $k\Delta t$ is associated with flux time $I(k\Delta t/\gamma)$. Re-indexing (13) gives

$$\delta\boldsymbol{\chi}(k\Delta t) = \mathbf{M}_{k-1\to k}\delta\boldsymbol{\chi}([k-1]\Delta t) + \mathbf{S}\boldsymbol{\rho}^{I([k-1]\Delta t/\gamma)}\Delta t. \tag{14}$$

Putting this together gives the relationship between $\boldsymbol{\mathcal{M}}_{k-1\to k}$ and $\mathbf{M}_{k-1\to k}$:

$$\begin{aligned} \mathbf{X}(k\Delta t) &= \boldsymbol{M}_{k-1\to k}\mathbf{X}([k-1]\Delta t) \\ \begin{pmatrix} \boldsymbol{\chi}([k-1]\Delta t) \\ \boldsymbol{\chi}(k\Delta t) \\ \boldsymbol{\rho} \end{pmatrix} &= \begin{pmatrix} \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_{k-1\to k} & \mathbf{S} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{\chi}([k-2]\Delta t) \\ \boldsymbol{\chi}([k-1]\Delta t) \\ \boldsymbol{\rho} \end{pmatrix}, \end{aligned} \tag{15}$$

where $\mathbf{S}$ now also picks-out the relevant flux time. Alternatively, the above can be written

$$\begin{aligned} \mathbf{X}([k+1]\Delta t) &= \boldsymbol{\mathcal{M}}_{k\to k+1}\mathbf{X}(k\Delta t) \\ \begin{pmatrix} \boldsymbol{\chi}(k\Delta t) \\ \boldsymbol{\chi}([k+1]\Delta t) \\ \boldsymbol{\rho} \end{pmatrix} &= \begin{pmatrix} \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_{k\to k+1} & \mathbf{S} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{\chi}([k-1]\Delta t) \\ \boldsymbol{\chi}(k\Delta t) \\ \boldsymbol{\rho} \end{pmatrix}. \end{aligned} \tag{16}$$

The adjoint of this step is

$$\begin{aligned} \hat{\mathbf{X}}([k-1]\Delta t) &:= \boldsymbol{\mathcal{M}}^{\mathsf{T}}_{k-1\to k}\hat{\mathbf{X}}(k\Delta t) \\ \begin{pmatrix} \hat{\boldsymbol{\chi}}([k-2]\Delta t) \\ \hat{\boldsymbol{\chi}}([k-1]\Delta t) \\ \hat{\boldsymbol{\rho}} \end{pmatrix} &:= \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{I} & \mathbf{M}^{\mathsf{T}}_{k-1\to k} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}^{\mathsf{T}} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \hat{\boldsymbol{\chi}}([k-1]\Delta t) \\ \hat{\boldsymbol{\chi}}(k\Delta t) \\ \hat{\boldsymbol{\rho}} \end{pmatrix} \end{aligned} \tag{17}$$

or

$$
\begin{aligned}
\hat{\mathbf{X}}\left(k\Delta t\right) &:= \boldsymbol{\mathcal{M}}^{\mathsf{T}}_{k\to k+1}\hat{\mathbf{X}}\left([k+1]\Delta t\right) \\
\begin{pmatrix} \hat{\boldsymbol{\chi}}\left([k-1]\Delta t\right) \\ \hat{\boldsymbol{\chi}}\left(k\Delta t\right) \\ \hat{\boldsymbol{\rho}} \end{pmatrix} &:= \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{I} & \mathbf{M}^{\mathsf{T}}_{k\to k+1} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}^{\mathsf{T}} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \hat{\boldsymbol{\chi}}\left(k\Delta t\right) \\ \boldsymbol{\chi}\left([k+1]\Delta t\right) \\ \hat{\boldsymbol{\rho}} \end{pmatrix}.
\end{aligned}
\tag{18}
$$

In terms of $\boldsymbol{\mathcal{M}}$, the cost function (6) is

$$
\begin{aligned}
J_{\delta\mathbf{x}}(\delta\mathbf{x}) &= \underbrace{\frac{1}{2}\delta\mathbf{x}^{\mathsf{T}}\mathbf{B}^{-1}\delta\mathbf{x}}_{J_{\mathrm{b}}} + \\
&\quad \underbrace{\frac{1}{2}\sum_{k=0}^{N_T}\left(\mathbf{y}_k - \boldsymbol{\mathcal{H}}_k\boldsymbol{\mathcal{M}}_{0\to k}\mathbf{C}\left[\mathbf{x}_{\mathrm{b}}+\delta\mathbf{x}\right]\right)^{\mathsf{T}}\mathbf{R}_k^{-1}\left(\mathbf{y}_k - \boldsymbol{\mathcal{H}}_k\boldsymbol{\mathcal{M}}_{0\to k}\mathbf{C}\left[\mathbf{x}_{\mathrm{b}}+\delta\mathbf{x}\right]\right)}_{J_{\mathrm{o}}},
\end{aligned}
\tag{19}
$$

where $\mathbf{y}_t$ and $\mathbf{R}_k$ now relate to observations between time-steps $k-1$ and k (or just at 0 for $k=0$).

3.2.2 Efficient gradient calculation

The gradient of J_{b} is trivial:

$$
\frac{\partial J_{\mathrm{b}}}{\partial\delta\mathbf{x}} = \mathbf{B}^{-1}\delta\mathbf{x}. \tag{20}
$$

The gradient of J_{o} is non-trivial and is found as follows.

$$
\begin{aligned}
\frac{\partial J_{\mathrm{o}}}{\partial\delta\mathbf{x}} &= \frac{\partial}{\partial\delta\mathbf{x}}\left[\frac{1}{2}\sum_{k=0}^{N_T}\left(\mathbf{y}_k - \boldsymbol{\mathcal{H}}_k\boldsymbol{\mathcal{M}}_{0\to k}\mathbf{C}\left[\mathbf{x}_{\mathrm{b}}+\delta\mathbf{x}\right]\right)^{\mathsf{T}}\mathbf{R}_k^{-1}\left(\mathbf{y}_k - \boldsymbol{\mathcal{H}}_k\boldsymbol{\mathcal{M}}_{0\to k}\mathbf{C}\left[\mathbf{x}_{\mathrm{b}}+\delta\mathbf{x}\right]\right)\right] \\
&= -\mathbf{C}^{\mathsf{T}}\sum_{k=0}^{N_T}\boldsymbol{\mathcal{M}}^{\mathsf{T}}_{0\to k}\boldsymbol{\mathcal{H}}^{\mathsf{T}}_k\mathbf{R}_k^{-1}\left(\mathbf{y}_k - \boldsymbol{\mathcal{H}}_k\boldsymbol{\mathcal{M}}_{0\to k}\mathbf{C}\left[\mathbf{x}_{\mathrm{b}}+\delta\mathbf{x}\right]\right) \\
&= \mathbf{C}^{\mathsf{T}}\sum_{k=0}^{N_T}\boldsymbol{\mathcal{M}}^{\mathsf{T}}_{0\to k}\boldsymbol{\mathcal{H}}^{\mathsf{T}}_k\frac{\partial J_{\mathrm{o}}}{\partial\mathbf{y}^{\mathrm{m}}_k},
\end{aligned}
\tag{21}
$$

where $\partial J_{\mathrm{o}}/\partial\mathbf{y}^{\mathrm{m}}_k = -\mathbf{R}_k^{-1}\left(\mathbf{y}_k - \mathbf{y}^{\mathrm{m}}_k\right)$ and $\mathbf{y}^{\mathrm{m}}_t = \boldsymbol{\mathcal{H}}_k\boldsymbol{\mathcal{M}}_{0\to k}\mathbf{C}\left[\mathbf{x}_{\mathrm{b}}+\delta\mathbf{x}\right]$.

$$
\tfrac{\partial J_{\mathrm{o}}}{\partial\delta\mathbf{x}} = \mathbf{C}^{\mathsf{T}}
\left[
\begin{array}{llllllll}
\boldsymbol{\mathcal{M}}^{\mathsf{T}}_{0\to 1} & \boldsymbol{\mathcal{M}}^{\mathsf{T}}_{1\to 2} & \cdots & \boldsymbol{\mathcal{M}}^{\mathsf{T}}_{k-1\to k} & \cdots & \boldsymbol{\mathcal{M}}^{\mathsf{T}}_{T-2\to T-1} & \boldsymbol{\mathcal{M}}^{\mathsf{T}}_{T-2\to T-1} & \boldsymbol{\mathcal{M}}^{\mathsf{T}}_{T-1} \\
\boldsymbol{\mathcal{M}}^{\mathsf{T}}_{0\to 1} & \boldsymbol{\mathcal{M}}^{\mathsf{T}}_{1\to 2} & \cdots & \boldsymbol{\mathcal{M}}^{\mathsf{T}}_{k-1\to k} & \cdots & \boldsymbol{\mathcal{M}}^{\mathsf{T}}_{T-2\to T-1} & \boldsymbol{\mathcal{M}}^{\mathsf{T}}_{T-2\to T-1} & \boldsymbol{\mathcal{H}}^{\mathsf{T}}_{N_T-1} \\
\boldsymbol{\mathcal{M}}^{\mathsf{T}}_{0\to 1} & \boldsymbol{\mathcal{M}}^{\mathsf{T}}_{1\to 2} & \cdots & \boldsymbol{\mathcal{M}}^{\mathsf{T}}_{k-1\to k} & & \boldsymbol{\mathcal{M}}^{\mathsf{T}}_{T-2\to T-1} & \boldsymbol{\mathcal{H}}^{\mathsf{T}}_{N_T-2}\frac{\partial J_{\mathrm{o}}}{\partial\mathbf{y}^{\mathrm{m}}_{N_T-2}}+ & \\
\vdots & \vdots & \ddots & \vdots & & & & \\
\boldsymbol{\mathcal{M}}^{\mathsf{T}}_{0\to 1} & \boldsymbol{\mathcal{M}}^{\mathsf{T}}_{1\to 2} & \cdots & \boldsymbol{\mathcal{M}}^{\mathsf{T}}_{k-1\to k} & \boldsymbol{\mathcal{H}}^{\mathsf{T}}_k\frac{\partial J_{\mathrm{o}}}{\partial\mathbf{y}^{\mathrm{m}}_k} & & & \\
\vdots & \vdots & & & & & & \\
\boldsymbol{\mathcal{M}}^{\mathsf{T}}_{0\to 1} & \boldsymbol{\mathcal{M}}^{\mathsf{T}}_{1\to 2} & \boldsymbol{\mathcal{H}}^{\mathsf{T}}_2\frac{\partial J_{\mathrm{o}}}{\partial\mathbf{y}^{\mathrm{m}}_2}+ & & & & & \\
\boldsymbol{\mathcal{M}}^{\mathsf{T}}_{0\to 1} & \boldsymbol{\mathcal{H}}^{\mathsf{T}}_1\frac{\partial J_{\mathrm{o}}}{\partial\mathbf{y}^{\mathrm{m}}_1} & & & & & & \\
\boldsymbol{\mathcal{H}}^{\mathsf{T}}_0\frac{\partial J_{\mathrm{o}}}{\partial\mathbf{y}^{\mathrm{m}}_0} & & & & & & &
\end{array}
\right]
$$

4 Building the master programs

4.1 Software required

This software requires the following free software and libraries to be installed on the host system.

- *C++ compiler* (gnu): this is produces most of the executables as EnviFlux is written in C++.
- *NetCDF library*: this is needed to handle the input and output of fields.
- *FFTW library*: this is needed to do the fast Fourier transforms.
- *SHTools library*: this is needed to produce a data file of Gaussian co-latitudes, Gaussian weights, associated Legendre polynomials (for a spectral transform that is used by EnviFlux), and other data. This library is called only once per grid configuration and produces a file, *cvt_ ass_ legendre_ poly.dat*, used by EnviFlux.
- *Fortran compiler* (gnu): this is needed to produce the CVT files (standing for Control Variable Transform, which is the way that the B-matrix is encoded).
- Python: this is needed for post-processing and to visualise the outputs of EnviFlux.

4.2 Contents of the EnviFlux download package

- **Calibration** This directory contains ForTran code to produce the file that contains all the information to represent the B-matrix of the problem (the CVT file – see Sect. 4.1). The code is legacy ForTran code used to generate covariance files for INVICAT.
- **data** This directory contains data needed to run the flux inversion system. This includes calibration files, wind data files, and the associated Legendre polynomial data file needed to perform spectral transforms.
- **docs** This directory contains documentation, including this document.
- **graphics** This directory contains python scripts to visualise output from EnviFlux.
- **src** This directory contains the C++ source code of EnviFlux. Most of the system is written in C++. The only exception is the calibration code, which is written in Fortran-90 (this is the same code as used for InviCat, but EnviFlux does not use all the options that can be set), and the visualisation code, which is written in python.

4.3 Build commands

To build one of the programs (each program is documented below), use the following command:

```
make Master_program_name
```

This will compile and link the required program. The running instructions are provided in each subsection.

5 Running the master test programs

There is a suite of test programs to test coding of (i) read/write routines, (ii) adjoint routines, (iii) implied covariances, and (iv) the gradient of the cost function. In these programs, the input/output/specifications are hardwired into the code. Here is the list of master test programs that do these tasks.

- **Master_TestReadWrite.cpp** Tests reading and writing of state and wind data.
- **Master_AdjointTests_CVT.cpp** Tests the adjoints of the following routines: halos, spherical, cvt_h, cvt_v, cvt_t, cvt_total.
- **Master_AdjointTestsObObs.cpp** Tests the adjoint of the observation operator.
- **Master_AdjointTests_SemiLagrangian.cpp** Tests the adjoint of the semi-Lagrangian advection scheme.
- **Master_AdjointTests_SL+ObsOp.cpp** Tests the adjoint of the combined semi-Lagrangian advection scheme and the observation operator.
- **Master_ImpliedCov.cpp** Tests the CVT to give some implied covariances.
- **Master_GradTest.cpp** Tests the gradient produced by the pen-and-grad routine in EnviFlux.

The remaining suite of programs concern the running of the system.

- **Master_MakeWinds.cpp** Makes suitable wind files from ERA-5 winds to the format and resolution of EnviFlux. These drive the transport of the chosen tracer.
- **Master_MakeFields.cpp** Generates initial conditions and surface flux fields for a run of the forward model.
- **Master_Invicat2EnviFlux.cpp** Converts data from an INVICAT file to an EnviFlux file.
- **Master_ReplaceFirstField.cpp** makes a copy of an EnviFlux file, but replacing the initial conditions from another file.
- **Master_ForwardTraj.cpp** computes the trajectory of a particle advected by the prescribed winds.
- **Master_GenerateObs.cpp** outpus model observations from a given EnviFlux file.
- **Master_MakeBG.cpp** makes a background state by perturbing a specified truth with a random perturbation.
- **Master_Assimilate.cpp** runs 4DVar using a covariance file, a bakckground file, and some observations to produce an analysis.
- **Master_CalcError.cpp** calculates the differences between two states.

6 Running the master core routines

6.1 Master_MakeWinds: Generating wind fields at the suitable resolution to drive the dynamics

This program takes ECMWF ERA-5 wind fields and generates files of lower resolution winds. To build the code issue the command:

```
make Master_MakeWinds.out
```

in the directory containing the source code and makefile. This will create the executable file *Master_MakeWinds.out*.

To run this code, the user first needs to prepare a number of things. The raw u, v, w winds need to be downloaded from the ECWMF by the user and placed in a specific directory, a wind file list needs to be created, an output directory needs to be made, and a CVT file needs to be available (see below). The run command is

```
./Master_MakeWinds.out \
    <CVT file name (for meta data)> \
    <filename_containing_full-resolution_ECMWF_winds> \
    <output_directory_for_reduced_resolution_winds>
```

- The CVT file (or control variable transform file) is a netCDF file containing information that describes the background error covariance matrix used in the inverse problem. This file is specified here, even though no inverse problem is being solved by this module, as it contains the specification of the longitude/latitude/height grid that the ECMWF winds will be interpolated to (we call this *meta data*). The grid staggering is 'Arakawa C' in the horizontal and 'Charney–Phillips' grid in the vertical. Fields u and v are both placed on so-called 'half-levels', but at staggered locations in the horizontal, and w is placed on 'full-levels' at yet another horizontally staggered position. See Fig. 1.
- The filename containing the specifications of the full-resolution ECMWF wind netcdf files (the wind file list) is a text file of the following strict format.

  Wind file list

  ```
  Comment line 1
  Comment line 2
  Comment line 3
  Comment line 4
  Comment line 5
  Comment line 6
  <Directory containing the ECMWF wind files>
  <Filename of 1st set of winds> <No. of time slices in this file>
  <Filename of 2nd set of winds> <No. of time slices in this file>
  <Filename of 3rd set of winds> <No. of time slices in this file>
  ...
  ```

- The directory name to output the reduced resolution winds.

The full-resolution input ECMWF files are assumed to have a structure such that each file contains a different number of time slices. For example, if the ECMWF wind files have a time slice every 12 hours, and each input file contains a month of winds then the first file (January) is expected to contain 62 wind slices, the second file (February) is expected to contain 56 wind slices, etc. The output (reduced resolution) winds, however, are output as one time slice per file. These will have filename *Windsxxxx.nc*, so *Winds0000.nc* to *Winds0061.nc* will be the wind files extracted for January, *Winds0062.nc* to *Winds0117.nc* will be the wind files for February, etc. The user will need to extract as many wind files as required for the length of the model run or assimilation window.

6.2 Master_MakeFields: Generate initial conditions and surface flux fields for a run of the forward model

This program prepares a field of initial conditions (3D) and surface flux (2D+time) – together called a *state*. The output state can be directly fed into (e.g.) the program that runs the forward model. To build the code, issue the command:

```
make Master_MakeFields.out
```

in the directory containing the source code and makefile. This will create the executable file *Master_MakeFields.out*.

To run this code, the user first needs to prepare a CVT file (see below). The run command is

```
./Master_MakeFields.out \
   <CVT file name (for meta data)> \
   <Previous state file to increment using the procedure below
                    (initial field and flux) (or enter "nil")> \
   --- If not "nil" ---------------------------
     <State file type (i)nvicat or (e)nvi-flux>
     <Multiplication factor for the input file initial field>
     <Multiplication factor for the input file flux>
   <Number of blobs to add on to initial condition (0 for no blobs)> \
   For each initial condition blob:
     <long (deg)> <lat (deg)> <lev (m)> <amplitude> <horiz size (deg)>
                                        <vert size (m)> \
   <Number of surface source fields (one per sDt time step)> \
     <source time-step (source update, specify in days, converted to s)>
     For each sDt time step:
       <Number of blobs (0 for no blobs)> \
       For each surface source blob:
         <long (deg)> <lat (deg)> <size (tracer units per s), +/->
                                        <horiz size (deg)> \
   <Min flux>
   Output file name (will contain initial conditions and surface flux
                                        fields)
```

- The CVT file (or control variable transform file) is a netCDF file containing information that describes the background error covariance matrix used in the inverse problem. This file is specified here, even though no inverse problem is being solved by this module, as it contains the specification of the longitude/latitude/height grid that the ECMWF winds will be interpolated to (we call this *meta data*). The grid staggering is 'Arakawa C' in the horizontal and 'Charney–Phillips' grid in the vertical. Fields u and v are both placed on so-called 'half levels', but at staggered locations in the horizontal, and w is placed on 'full levels' at yet another horizontally staggered position. See Fig. 1.
- A previous state field will be read. The concentration contained in this file is the baseline state, which will be modified by 'blobs' at specified locations. If no previous state file is mentioned ("nil" is specified) then the baseline state is zero.
- The number of blobs to add on to the initial condition specifies how many regions (or blobs) of tracer should be added onto the specified initial conditions. For each one, specify:
  - The longitude of the blob's centre (in degrees).
  - The latitude of the blob's centre (in degrees).
  - The vertical level of the blob's centre (in metres).
  - The amplitude of the blob (in tracer units).

    - The horizontal size of the blob (in degrees).
    - The vertical size of the blob (in metres).

- The number of surface source/sink fields (in time) that will be included in the file.
- The time-step between the fields (specify this in days – the program will convert this to seconds). For each source/sink field:
    - The number of blobs specifies how many regions (or blobs) of flux should be added together. For each one specify:
        * The longitude of the flux blob's centre (in degrees).
        * The latitude of the flux blob's centre (in degrees).
        * The amplitude of the flux blob (in tracer units per second, can be positive or negative).
        * The horizontal size of the blob (in degrees).
- The minimum flux: sets fluxes to this value when fluxes are smaller in mag than this (preserves sign).
- The output filename. This file will contain the initial conditions and the set of flux fields.

6.3 Master_Invicat2EnviFlux: Interpolate INVICAT file, and scale, to EnviFlux

This program reads in prior fluxes and initial tracer fields from an INVICAT file and outputs them interpolated to the EnviFlux grid. It also scales the fluxes and tracer to be consistent with EnviFlux. To build the code, issue the command:

```
make Master_Invicat2EnviFlux.out
```

in the directory containing the source code and makefile. This will create the executable file *Master_MakeFields.out*.

To run this code, the user first needs to prepare a CVT file (see below). The run command is

```
./Master_Invicat2EnviFlux.out \
    <CVT file name (for meta data)> \
    <INVICAT state file> \
    <Output state file> \
    <Maximum flux value> \
    <Maximum tracer value>
```

- The CVT file is for meta data.
- How many months are used for the output file? Whichever is smaller between the number of months in the INVICAT file and in the CVT file.
- The number of levels in the INVICAT file must be $\geq$ the number of levels specified in the CVT file.
- The output state file contains the interpolated and scaled fluxes and initial tracer.
- The interpolated INVICAT fluxes are scaled such that the modulus of the maximum flux is that specified in the run command (if the value in the argument is set to 999.0 then no scaling is done).

- The interpolated INVICAT intitial tracer are scaled such that the modulus of the maximum value is that specified in the run command (if the value in the argument is set to 999.0 then no scaling is done).

6.4 Master_ReplaceFirstField: Make a new state file by replacing the initial tracer with another field

This program prepares a new state file by replacing the field of initial conditions (3D) of an existing state file. The output state can be directly fed into (e.g.) the program that runs the forward model. To build the code, issue the command:

```
make Master_ReplaceFirstField.out
```

in the directory containing the source code and makefile. This will create the executable file *Master_MakeFields.out*.

The run command is

```
./Master_ReplaceFirstField.out \
    <Input state filename> \
    <Input forecast filename> \
    <Forecast time-step number (1 is first)> \
    <Output state filename>
```

- The input state is an existing state file, where the initial concentration is required to be replaced with an alternative 3D field.
- The input forecast is an existing forecast file containing a sequence of tracer fields (one of which is required to replace the initial concentration mentioned above).
- The forecast time-step number is an integer (starting at 1), which specifies which time slice will be extracted from the input forecast file.
- The output state filename is a new file, which will contain the initial tracer extracted from the forecast file, and the flux fields from the input state.

6.5 Master_SemiLagrangian: Run the forward model to produce a forecast

This program runs a forecast using the semi-Lagrangian method. A state vector (intial conditions and surface flux boundary conditions) needs to be specified together with a sequence of wind fields to drive the dynamics. To build the code, issue the command:

```
make Master_SemiLagrangian.out
```

in the directory containing the source code and makefile. This will create the executable file *Master_SemiLagrangian.out*.

The run command is

```
./Master_SemiLagrangian.out \
    <state filename (initial and boundary conds)>\
    <wind directory> \
    <major time-step (between winds, s)> \
    <minor time-step (integration, s)> \
    <diffusion time-step (s) \
    <horiz diffusion coefficient> \
    <vert diffusion coefficient> \
    <include vertical transport?>*** \ 0 or 1
    <w factor>*** \
    <run length (days)> \ Multiple of major time-step
    <output frequency (seconds)> \
    <interpolation type (l or c)> \ linear or cubic
    <animation output filename> \ nil to not output
    <diagnostics output filename> \ nil to not output
    <departure points filename>*** \ Needed in prep. for adjoint run
                                            nil to not output
    <(r)ead or (w)rite departure points file>***

*** See special note below
```

- The state file contains the initial and boundary (surface flux) conditions.
- The wind directory contains the sequence of driving winds.
- The major time-step is the time-step between wind files (seconds).
- The minor time-step is the time-step of the backward trajectories in the semi-Lagrangian scheme (seconds).
- The diffusion time-step is the time-step of the diffusion process (seconds, applied after the semi-Lagrangian.
- The horizontal diffusion coefficient is for the horizontal diffusion scheme (0.0 for no horizontal diffusion).
- The vertical diffusion coefficient is for the vertical diffusion scheme (0.0 for no vertical diffusion).
- Include the vertical transport (0 or 1)?
- The w factor multiplies the vertical wind component by this number.
- The run length indicates the length of the integration (days, must be a multiple of the major time-step).
- The output frequency is the temporal spacing between tracer outputs (seconds).
- The interpolation type is either “l” for linear interpolation or “c” for cubic interpolation.
- The animation output filename specifies the forecast output file (“nil” to not output animation file).
- The diagnostics output filename specifies diagnostics as a function of time (“nil” to not output diagnostics file).

- The departure points filename is a file of semi-Lagrangian departure points output for every major time-step. This is needed in preparation for the adjoint run when data assimilation is run (see Sect. 6.9, “nil” to not output departure points filename).
- *** Warning: there may be an inconsistency between the entries marked above. If a departure points file is specified (“departure points filename”) to be read (“(r)ead or (w)rite departure points file”), the dynamical settings “include vertical transport?” and “w factor” will be ignored. In this case the dynamical settings that were used in the run that produced the departure points filename will be used.

6.6 Master_ForwardTraj: Run the forward trajectory code

This program runs forward trajectory code, computing the position of a particle at future times, driven by prescribed wind fields. To build the code, issue the command:

```
make Master_ForwardTraj.out
```

in the directory containing the source code and makefile. This will create the executable file *Master_ForwardTraj.out*.

The run command is

```
./Master_ForwardTraj.out \
    <wind directory> \
    <major time-step (between winds, s)> \
    <minor time-step (integration, s)> \
    <w factor> \
    <run length (days)> \ Must be multiple of major time-step
    <output filename>
```

- The wind directory contains the sequence of driving winds.
- The major time-step is the time-step between wind files (seconds).
- The minor time-step is the time-step of the backward trajectories in the semi-Lagrangian scheme (seconds).
- The w factor multiplies the vertical wind component by this number.
- The run length indicates the length of the integration (days, must be a multiple of the major time-step).
- The output filename will contain the time-step-by-time-step trajectory of the particle: time, longitude (deg), latitude (deg), level (m).
- The initial particle position is specified inside the code (particle_lon, particle_lat, particle_lev).

6.7 Master_GenerateObs: Generate synthetic observations from specification of an observation network and a truth run

This program generates a set of synthetic observations. An observation network is specified, plus a state vector (intial conditions and surface flux boundary conditions), representing the truth. The observation network is specified with the positions of the observations and their respective instrument error characteristics. A sequence of wind fields is needed to drive the dynamics, and it is possible to add some stochastic errors to this field. The observation values (with metadata) will be output in a file. To build the code, issue the command:

```
make Master_GenerateObs.out
```

in the directory containing the source code and makefile. This will create the executable file *Master_GenerateObs.out*.

The run command is

```
./Master_GenerateObs.out \
    <state filename (initial and boundary conds)>\
    <wind directory> \
    <obs output filename> \
    <departure points filename>*** \ Needed in prep. for adjoint run
                                     nil to not output
    <major time-step (between winds, s)> \
    <minor time-step (integration, s)> \
    <diffusion time-step (s) \
    <horiz diffusion coefficient> \
    <vert diffusion coefficient> \
    <include vertical transport?>*** \ 0 or 1
    <w factor>*** \
    <output frequency, seconds> \
    <interpolation type (l or c)> \ linear or cubic
    <read or write departure points file>*** \ r or w
   --- Observation network specifications ---
      <What is observed> \ t=tracer, f=flux,
                           x=total column tracer
                           e=end of spec
      <Network type> \ i=individual ob, g=grid of obs, s=satellite
      --- If "i" --------------------------
        <lon (deg)> <lat (deg)> <ht (m) (tracer only)> \
        <day> <hour> <min> \
        <error standard deviation> \
      --- If "g" --------------------------
        <start lon (deg)> \ west-most, deg
        <separation in lon (deg)> \
        <number in lon direction> \
        <start lat (deg)> \ south-most, deg
        <separation in lat (deg)> \
        <number in lat direction> \
        <start ht (m) (tracer only)> \
        <separation in ht (m) (tracer only)> \
        <number in ht direction (tracer only)> \
        <start time (day)> <hour> <min> \
        <separation in time (min)> \
        <number in time> \
        <error standard deviation> \
    ——— If "s" ————————————————————————————————
      <orbit inclination (deg)> \
      <orbit period (min)> \
      <t=0 long (deg)> \
      <height (m) (tracer only)> \
      <separation in time (min)> \
      <number in time> \
      <error standard deviation>

*** See special note below
```

- Many of the program's arguments are the same as for Master_SemiLagrangian, see Sect. 6.5.
- *** Warning: there may be an inconsistency between the entries marked above. If a departure points file is specified ("departure points filename") to be read ("(r)ead or (w)rite departure points file"), the dynamical settings "include vertical transport?" and "w factor" will be ignored. In this case the dynamical settings that were used in the run that produced the departure points filename will be used.

The output file of the observations has the following overall structure.

General structure of observations file (example with 56 vertical levels)

```
===== Observation file =====
nlevs: 56
===== Mass profile ==========
001 29.470579
002 36.358128
...
056 11.223949
===== Observation number 1 =====
OBSERVATION ENTRY
===== Observation number 2 =====
OBSERVATION ENTRY
...
```

The mass profile is related to the density of each level (needed for total columns observations), and is set by the code. No special flag is needed to the end list of observations (when the observations are read-in by other routines, the end of file is used to terminate the list). Each "OBSERVATION ENTRY" comprises a number of lines with a structure that indicates the obersation type. There are three observation types (*tracer*, *total column tracer*, and *flux*), which have structures as follows.

Observation entry for a tracer observation

```
ob_of: t
ob_type: i=individually specified obs, g=part of a grid of obs,
         s=part of a set of satellite obs (for info only)
time: time_index day hour min secs t_alpha t_beta
lon: lon_index longitude lon_alpha lon_beta
lat: lat_index latitude lat_alpha lat_beta
lev: lev_index level lev_alpha lev_beta
ob: ob_value ob_err_stddev
model_ob: model_ob_value
innov: ob_value − model_ob_value
grad: gradient
```

Notes:

- The time_index is not used for tracer observations, so is set to zero.
- The t_alpha and t_beta are temporal interpolation coefficients (these are set by the code).
- The lon_index is the longitude index associated with this observation.
- The lon_alpha, lon_beta are longitudinal interpolation coefficients (these are set by the code).
- The lat_index is the latitude index associated with this observation.
- The lat_alpha, lat_beta are latitudinal interpolation coefficients (these are set by the code).
- The lev_index is the level index associated with this observation.
- The lev_alpha, lev_beta are level interpolation coefficients (these are set by the code).
- The model_ob_value is the model's version of the observation. For the Master_GenerateObs routine, this contains the model observation corresponding to the true state (this is set by the code, as is the innovation).
- The innov is the observation – model observation(-9999.0 means that it has not been set).
- The gradient is the value of $\partial J_\mathrm{o}/\partial$model_ob (this is set by the code, -9999.0 means that it has not been set).

Observation entry for a total column tracer observation

```
ob_of: t
ob_type: i=individually specified obs, g=part of a grid of obs,
         s=part of a set of satellite obs (for info only)
time: time_index day hour min secs t_alpha t_beta
lon: lon_index longitude lon_alpha lon_beta
lat: lat_index latitude lat_alpha lat_beta
ob: ob_value ob_err_stddev
model_ob: model_ob_value
innov: ob_value − model_ob_value
grad: gradient
```

The above notes apply (there is no level information needed for a total column observation).

Observation entry for a flux observation

```
ob_of: t
ob_type: i=individually specified obs, g=part of a grid of obs,
         s=part of a set of satellite obs (for info only)
time: time_index day hour min secs t_alpha t_beta
lon: lon_index longitude lon_alpha lon_beta
lat: lat_index latitude lat_alpha lat_beta
lev: lev_index level lev_alpha lev_beta
ob: ob_value ob_err_stddev
model_ob: model_ob_value
innov: ob_value - model_ob_value
grad: gradient
```

The above notes apply (there is no level information needed for a flux observation). The exception is the following:

- The time_index is the time index associated with this observation (which indicates which flux field is associated with the observation).

6.8 Master_MakeBG: Make a background state by incrementing a truth with a random perturbation

This program generates a synthetic background state, $\mathbf{x}^{\mathrm{b}}$, by adding a random perturbation, $\delta\mathbf{x}^{\mathrm{b}}$, (with statistics consistent with background errors) on to a truth state, $\mathbf{x}^{\mathrm{t}}$:

$$\mathbf{x}^{\mathrm{b}} = \mathbf{x}^{\mathrm{t}} + \mathbf{U}\boldsymbol{\chi},$$

where $\delta\mathbf{x}^{\mathrm{b}} = \mathbf{U}\boldsymbol{\chi}$, $\mathbf{U}$ being the control variable transform related to the background error covariance matrix $\mathbf{B}$ via, $\mathbf{B} = \mathbf{U}\mathbf{U}^{\mathsf{T}}$, and $\boldsymbol{\chi}$ is a unit Gaussian random vector, $\boldsymbol{\chi} \sim N(0, \mathbf{I})$.

To build the code, issue the command:

```
make Master_MakeBG.out
```

in the directory containing the source code and makefile. This will create the executable file *Master_MakeBG.out*.

The run command is

```
./Master_MakeBG.out \
    <truth filename (initial and boundary conds)> \
    <output filename of background state> \
    <output filename of background perturbation> \
    <CVT file> \
    <multiplication factor for tracer err std> \
    <multiplication factor for flux err std> \
    <multiplication factor for tracer pert> \
    <multiplication factor for flux pert>
```

- The CVT file is produced by the Calibration.f90 program. It contains the details that describe the background error covariance matrix.
- The error standard deviations of the initial tracer and flux from the CVT file are multiplied by the respective std factors.
- The generated background error perturbations of the initial tracer and flux are multiplied by the respective perturbation factors.

6.9 Master_Assimilate: Run a 4D-Var assimilation

This program combines observations of the tracer (and/or flux if required) with a background state to produce a 4D-variational analysis.

To build the code, issue the command:

```
make Master_Assimilate.out
```

in the directory containing the source code and makefile. This will create the executable file *Master_Assimilate.out*.

The run command is

```
./Master_Assimilate.out \
    <filename of background state> \
    <wind directory> \
    <obs filename> \
    <departure points filename>*** \
    <CVT file> \
    <multipication factor for tracer err std> \
    <multipication factor for flux err std> \
    <obs output filename, inc model obs at bg> \
    <output filename of analysis state> \
    <output filename of analysis increment> \
    <output filename of analysis diagnostics> \
    <obs output filename, inc model obs at anal>\
    <major time-step (between winds, s)> \
    <minor time-step (integration, s)> \
    <diffusion time-step (s) \
    <horiz diffusion coefficient> \
    <vert diffusion coefficient> \
    <include vertical transport?>*** \ 0 or 1
    <w factor>*** \
    <interpolation type (l or c)> \ linear or cubic
    <descent algorithm type (c or q)> \ c=conjugate grad
    <convergence criterion> \
    <max number of var iterations>

*** See special note below
```

- The background file contains the background initial and boundary (surface flux) conditions.
- The wind directory contains the sequence of driving winds.

- The observation file contains the observations to be assimilated.
- The departure points filename is a file of semi-Lagrangian departure points output for every major time-step. If this file does not exist at the start of the run it is created from the first Var iteration.
- The CVT file is produced by the Calibration.f90 program. It contains the details that describe the background error covariance matrix.
- The multiplication factors respectively multiply the initial tracer and flux standard deviations read in from the CVT file.
- The next observation file is for output; it contains the same information as the input file, but additionally with the model observations for the background, with innovation and gradient information. Specily "nil" to not output.
- The analysis state file contains the 4D-Var analysis.
- The analysis increment file contains the 4D-Var analysis increment.
- The analysis diagnostics file contains the following information: iteration, J^{b}, J^{o}, J, $\|\nabla_{\chi}J\|^2$, $\|\nabla_{\chi}J\|$.
- The last observation file is for output; it contains the same information as the input file, but additionally with the model observations for the analysis, with residual and gradient information (the residual is placed in the field labelled "innov"). Specily "nil" to not output.
- The major time-step is the time-step between wind files (seconds).
- The minor time-step is the time-step of the backward trajectories in the semi-Lagrangian scheme (seconds).
- The diffusion time-step is the time-step of the diffusion process (seconds, applied after the semi-Lagrangian.
- The horizontal diffusion coefficient is for the horizontal diffusion scheme (0.0 for no horizontal diffusion).
- The vertical diffusion coefficient is for the vertical diffusion scheme (0.0 for no vertical diffusion).
- Include the vertical dimension? (0 means use one level, 1 means use all vertical levels).
- The w factor multiplies the vertical wind component by this number.
- The interpolation type is either "l" for linear interpolation or "c" for cubic interpolation.
- The interpolation type can be selected as either "c" for conjugate gradient or "q" for quasi-Newton (latter not yet implemented).
- The convergence criterion sets the critical value of $r = \|\nabla_{\chi}J\|^2$ (iteration n)/ $\|\nabla_{\chi}J\|^2$ (iteration 0) for termination of the iterations. For example if the criterion is set to 0.01 then the iterations will terminate when $r < 0.01$.
- The maximum number of iterations performs as specified.
- *** Warning: there may be an inconsistency between the entries marked above. If a departure points file is specified ("departure points filename"), the dynamical settings "include vertical transport?" and "w factor" will be ignored. In this case the dynamical settings that were used in the run that produced the departure points filename will be used.

6.10 Master_CalcError: Calculate norms of the differences between two states

This program reads in two states and computes some norms of the difference between them.
To build the code, issue the command:

```
make Master_CalcError.out
```

in the directory containing the source code and makefile. This will create the executable file *Master_CalcError.out*.
The run command is

```
./Master_CalcError.out \
    <filename of state 1> \
    <filename of state 2>
    <output filename>
```

The following difference norms are computed for the two states, $\mathbf{x}_1$ and $\mathbf{x}_2$:

$$\begin{aligned}\text{norm1} &= \sum_p (\mathbf{x}_1(p) - \mathbf{x}_2(p)) \\ \text{norm2} &= \sqrt{\sum_p (\mathbf{x}_1(p) - \mathbf{x}_2(p))^2},\end{aligned}$$

where p represents the 'location' in the field. These norms are computed separately for the initial tracer and for the fluxes. For the initial tracer, p represents longitude, latitude, and height; for the fluxes, p represents longitude, latitude, and time. When state 2 is the 'truth', the norms represent the error in state 1.

7 Running procedure

Running and testing a system like EnviFlux is complex. In order to do an assimilation run, many files need to be produced first.

1. **Generate some wind files**. How to do this is detailed in Sect. 6.1. You need to generate wind fields that cover the period of interest.

2. **Generate a covariance file for the system**. EnviFlux needs a CVT file, which is generated with the same FORTRAN program used for the INVICAT system. Some special options are needed for use with the toy system.

   a) The calibration program is at *Calibration/Calibration.f90*. Here are the settings that need to be made to this file.
   If temporal correlations (for the source/sink field) are required, set

   ```
   CVT_TemporalCors % temporal_covs(field) = 1
   ```

   otherwise

   ```
   CVT_TemporalCors % temporal_covs(field) = 0
   ```

If temporal correlations are switched on (as above), set the shape of the temporal correlations (1=Lorentzian, 2=Gaussian, 3=SOAR, 4=exponential), e.g.

```
CVT_TemporalCors % temporal_cor_tpe(field) = 3
```

If temporal correlations are switched on (as above), set the timescale of the correlation (months), e.g.

```
CVT_TemporalCors % timescale_flux(field)    = 1.0
```

Set the standard deviation of the initial field, e.g.

```
CVT_std % std_tracer(lat,lon,lev) = 15.0
```

Set the standard deviation of the source/sink field, e.g.

```
CVT_std % std_flux(1:ylat, 1:ylon, 1:nmonth, field) = 0.00001
```

Set the lengthscale of the initial field (m), e.g.

```
CVT_HorizCors % lengthscale_tracer(1:ylev) = 600000.0
```

Set the shape of the horizontal correlations for the source/sink field (1=Lorentzian, 2=Gaussian, 3=SOAR, 4=exponential), e.g.

```
CVT_HorizCors % horizflux_cor_tpe(field) = 3
```

Set the lengthscale of the source/sink field (m), e.g.

```
CVT_HorizCors % lengthscale_flux(time1,field) = 400000.0
```

b) The next file to edit is *Calibration/main_data.f90*. Here are the settings that need to be made to this file.
Make sure that the following are set (e.g. for $L = 32$)

```
ylon=64, ylat=32, ylev=56
nmonth=37
```

c) The next file to edit is *Calibration/onedvar_data.f90*. Here are the settings that need to be made to this file.
Set the shape of the horizontal correlations for the initial field (1=Lorentzian, 2=Gaussian, 3=SOAR, 4=exponential), e.g.

```
horiztracer_cor_tpe = 3
```

Make sure that the following is set (to use global eigenvalues in the vertical covariances)

```
vert_covs = 3
```

d) Go back to the directory containing the Calibration.90 file and run

```
make Calibration.out
./Calibration.out
```

The file *CVT_calib.nc* is output, which can be moved elsewhere.

3. **Generate a truth state**. A prescription for doing this from scratch is detailed here.
   a) Optional: User the *Master_WorldFlux* utility to convert a given Invicat flux field to the grid and units used by EnviFlux. This creates a state file containing the Invicat fluxes, and with zero for the initial field.
   b) Use the *Master_MakeFields* utility to generate a state field with a constant initial field (achieved by specifiying a single blob with extremely large horizontal and vertical lengthscales), and localised source blobs. See Sect. 6.2.
   c) Run this field forward in time using the *Master_SemiLagrangian* utility. See Sect. 6.5.
   d) Use the *Master_ReplaceFirstField* utility to take the file produced in step 3b above and replace the initial field with the output from step 3c. See Sect. 6.4.
   e) Alternatively, use the Master_WorldFlux utility to generate fields of flux from an existing Invicat file. See Sect. 6.3.

4. **Generate a background state**. A prescription for producing a background state as a perturbed version of the initial field of the true state, and zero background for the flux, use the following prescription.
   a) Use the *Master_MakeFields* utility to generate a state field with zero values for the initial field and flux. See Sect. 6.2.
   b) Use the *Master_ReplaceFirstField* utility to take the file produced in step 4a above and replace the initial field with the output from step 3c when generating the truth. See Sect. 6.4.
   c) Use the *Master_MakeBG* utility to perturb the output of step 4b with random background perturbations specified in the covariance file. The standard deviations of the initial and flux fields can be scaled (separately). In order to have zero perturbations made to the flux field, the scale for the flux can be set to zero. See Sect. 6.8.

5. Use the utility to *Master_GenerateObs* utility to generate some synthetic observations. Observations are generated by integrating the truth forward in time, computing sets of synthetic observations, and adding noise to produce observations that can later be assimilated. Observations can

be individual point measurements, regular grids of measurements, or patterns of observations that represent the nadir of a satellite orbit. Observations can be direct measurement of the tracer or the flux, or total column measurements of the tracer. See Sect. 6.7.

Index