Computational Data Assimilation: Project 3

November 27, 2022

1 Resources

The following resources may be useful while working on the project, beyond the class notes:

1. The syllabus contains pointers to three data assimilation textbooks, by Asch et al., Law et al., and Reich et al. The Asch et al. book is written at an introductory level and is particularly well accessible.

2. The book "Data Assimilation: A Mathematical Introduction" by K.J.H. Law et al., first part of which is available on arxiv at `https://arxiv.org/pdf/1506.07825.pdf`. See Chapters 4.2 for EnKF and 4.3 for particle filters, and Chapter 5 for Matlab programs.

3. The OTP Matlab package available from URL `https://github.com/ComputationalScienceLaboratory/ODE-Test-Problems`. It contains implementations of the Lorenz models, among others. Start with the readme file.

4. The DATools Matlab package available from `https://github.com/ComputationalScienceLaboratory/DATools`. It contains implementations of all the data assimilation algorithms we will discuss in class, including various particle filters. Start with the readme file.

5. The MATLODE Matlab package available from `https://github.com/ComputationalScienceLaboratory/MATLODE`. It contains implementations of many time stepping algorithms and direct and adjoint sensitivity analyses.

2 Models

2.1 Lorenz three-variables system

The Lorenz three-variables system Lorenz-63 [**?**] is described by the equations:

$$\begin{cases} x_1' = \sigma\,(x_2 - x_1), \\ x_2' = x_1\,(\rho - x_3) - x_2, \quad t_0 \le t \le t_{\mathrm{F}}, \quad \mathbf{x}(t_0) = \mathbf{x}_0, \\ x_3' = x_1\,x_2 - \beta\,x_3, \end{cases} \tag{1a}$$

with the parameter values

$$\sigma = 10, \quad \rho = 28, \quad \beta = 8/3. \tag{1b}$$

For these values of parameters the Lorenz-63 system has a chaotic dynamics. The chaotic solutions (started from appropriate initial values) fall onto the Lorenz attractor. This physical system is completely deterministic and yet inherently unpredictable.

We consider the evolution of nature (**??**) to be known and governed by the Lorenz equations (1a) with the system parameters (1b). The true state has $n = 3$ components. The initial conditions, however, are uncertain, and therefore the evolution of the entire system is uncertain.

At different times t_i we measure the first two components $x_1(t_i)$ and $x_2(t_i)$. The inverse problem is to estimate the entire state, and therefore to follow closely the model trajectory, from these measurements. The system in general ODE notation reads:

$$\mathbf{x}' = f(\mathbf{x})$$

where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad f(\mathbf{x}) = \begin{bmatrix} \sigma\,(x_2 - x_1) \\ x_1\,(\rho - x_3) - x_2 \\ x_1\,x_2 - \beta\,x_3 \end{bmatrix}. \tag{2}$$

The Jacobian of the ODE function is the matrix of partial derivatives:

$$f_{\mathbf{x}}(\mathbf{x}) = \left(\frac{\partial f_i(\mathbf{x})}{\partial \mathbf{x}_j} \right)_{1 \le i,j \le n}.$$

For our function (2) the Jacobian matrix reads:

$$f_{\mathbf{x}}(\mathbf{x}) = \begin{bmatrix} -\sigma & \sigma & 0 \\ (\rho - x_3) & -1 & -x_1 \\ x_2 & x_1 & -\beta \end{bmatrix}. \tag{3}$$

We also consider the following initial condition and climatological covariance matrix at the initial time:

$$\mathbf{x}_0 = \begin{bmatrix} -10.0375 \\ -4.3845 \\ 34.6514 \end{bmatrix}, \quad \mathbf{B}_0 = \begin{bmatrix} 12.4294 & 12.4323 & -0.2139 \\ 12.4323 & 16.0837 & -0.0499 \\ -0.2139 & -0.0499 & 14.7634 \end{bmatrix}. \tag{4}$$

Observations are taken at equidistant times, with $\Delta t = t_i - t_{i-1} = 0.1$ (units). Over each interval $[t_i, t_{i-1}]$ a numerical method is used to solve the ODE (1) using 50 equidistant steps, with a step size $h = \Delta t/50$.

3 4D-Var data assimilation

Consider the assimilation window $[t_0, t_{\mathrm{N}}]$. At each time t_i, $0 \leq i \leq \mathrm{N}$, observations are taken:

$$\mathsf{y}_i^{\mathrm{obs}} = \mathcal{H}_i\left(\mathbf{x}_i\right) + \varepsilon_i^{\mathrm{obs}}, \quad i = 0, \cdots, \mathrm{N}; \quad \varepsilon_i^{\mathrm{obs}} \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_i). \tag{5}$$

The model that advances the state from time t_{i-1} to tome t_i is perfect, in the sense that the model error is zero:

$$\mathbf{x}_i = \mathcal{M}_{i-1,i}\left(\mathbf{x}_{i-1}\right), \quad i = 1, \cdots, \mathrm{N}. \tag{6}$$

The model (6) uniquely determines the trajectory the system $\mathbf{x}_{0:\mathrm{N}}$ at all times in the assimilation window, given the initial condition $\mathbf{x}_0$.

3.1 The 4D-Var problem

The 4D-Var analysis over window $[t_0, t_{\mathrm{N}}]$ is the MAP estimate of the initial state conditioned by all observations in this window. It is therefore computed as the initial condition that minimizes the negative log posterior distribution subject to the model equation constraints:

$$\begin{aligned} \Psi\left(\mathbf{x}_0\right) &= \frac{1}{2}\left\|\mathbf{x}_0 - \mathbf{x}_0^{\mathrm{b}}\right\|_{\mathbf{B}_0^{-1}}^2 + \frac{1}{2}\sum_{i=0}^{\mathrm{N}}\left\|\mathcal{H}_i\left(\mathbf{x}_i\right) - \mathsf{y}_i^{\mathrm{obs}}\right\|_{\mathbf{R}_i^{-1}}^2, \\ \mathbf{x}_0^{\mathrm{a}} &= \arg\min_{\mathbf{x}_0} \Psi\left(\mathbf{x}_0\right) \\ &\text{subject to:} \quad \mathbf{x}_i = \mathcal{M}_{i-1,i}\left(\mathbf{x}_{i-1}\right), \quad i = 1, \cdots, \mathrm{N}. \end{aligned} \tag{7}$$

To solve (7) one typically uses a gradient based optimization method, which requires the computation of $\nabla_{\mathbf{x}_0}\Psi\left(\mathbf{x}_0\right)$.

The analysis trajectory $\mathbf{x}_{0:\mathrm{N}}^{\mathrm{a}}$ over the entire window is obtained by propagating the analysis initial condition through the model (6):

$$\mathbf{x}_i^{\mathrm{a}} = \mathcal{M}_{i-1,i}\left(\mathbf{x}_{i-1}^{\mathrm{a}}\right), \quad i = 1, \cdots, \mathrm{N}. \tag{8}$$

4D-Var is applied in a cyclic manner. After the analysis (7)–(8) over the current assimilation window $[t_0, t_{\mathrm{N}}]$ is complete, we move our attention to the next assimilation window $[t_{\mathrm{N}+1}, t_{2\mathrm{N}+1}]$. At each time t_i, $\mathrm{N}+1 \leq i \leq 2\mathrm{N}+1$, observations (5) are taken. The analysis state (8) propagated to $t_{\mathrm{N}+1}$ provides the background initial condition for the new window:

$$\mathbf{x}_{\mathrm{N}+1}^{\mathrm{b}} = \mathcal{M}_{\mathrm{N},\mathrm{N}+1}\left(\mathbf{x}_{\mathrm{N}}^{\mathrm{a}}\right).$$

The same initial covariance matrix $\mathbf{B}_{\mathrm{N}+1} = \mathbf{B}_0$ is typically used, and the analysis process (7)–(8) is repeated over $[t_{\mathrm{N}+1}, t_{2\mathrm{N}+1}]$. After that, the next assimilation window is considered, etc.

3.2 Gradient of the cost function

We denote the Jacobian of the model dynamical operator (6) by:

$$\mathbf{M}_{i-1,i}(\mathbf{x}_{i-1}) := \frac{\partial \mathcal{M}_{i-1,i}(\mathbf{x}_{i-1})}{\partial \mathbf{x}_{i-1}} \in \mathbb{R}^{n\times n}, \tag{9}$$

and the Jacobian of the observation operator (5) by:

$$\mathbf{H}_{i}(\mathbf{x}_{i-1}) := \frac{\partial \mathcal{H}_{i}(\mathbf{x}_{i})}{\partial \mathbf{x}_{i}}. \in \mathbb{R}^{n\times n}, \tag{10}$$

The gradient of the 4D-var cost function (7) is obtained by direct differentiation:

$$\nabla_{\mathbf{x}_0}\Psi(\mathbf{x}_0) = \mathbf{B}_0^{-1}\left(\mathbf{x}_0 - \mathbf{x}_0^{\mathrm{b}}\right) + \sum_{i=0}^{\mathrm{N}} \mathbf{M}_{0,i}^{T}\,\mathbf{H}_i^{T}\,\mathbf{R}_i^{-1}\left(\mathcal{H}_i(\mathbf{x}_i) - \mathsf{y}_i^{\mathrm{obs}}\right). \tag{11}$$

The gradient (11) can be obtained efficiently by running the following adjoint model:

$$\begin{aligned} \boldsymbol{\lambda}_{\mathrm{N}} &= \mathbf{H}_{\mathrm{N}}^{T}\,\mathbf{R}_{\mathrm{N}}^{-1}\left(\mathcal{H}_{\mathrm{N}}(\mathbf{x}_{\mathrm{N}}) - \mathsf{y}_{\mathrm{N}}^{\mathrm{obs}}\right), \\ \boldsymbol{\lambda}_{i-1} &= \mathbf{M}_{i-1,i}^{T}\,\boldsymbol{\lambda}_i + \mathbf{H}_i^{T}\,\mathbf{R}_i^{-1}\left(\mathcal{H}_i(\mathbf{x}_i) - \mathsf{y}_i^{\mathrm{obs}}\right), \quad i = \mathrm{N},\ldots,1, \end{aligned} \tag{12}$$

and, accounting for the background term, the gradient (11) is:

$$\nabla_{\mathbf{x}_0}\Psi = \mathbf{B}_0^{-1}\left(\mathbf{x}_0 - \mathbf{x}_0^{\mathrm{b}}\right) + \boldsymbol{\lambda}_0. \tag{13}$$

The backward-in-time algorithm for adjoint integration is given in Algorithm 3.1.

Algorithm 3.1 Adjoint algorithm for computing gradients in 4D-Var data assimilation

Input: Initial condition $\mathbf{x}_0$; Observations $(t_i, \mathrm{y}_i^{\mathrm{obs}}, \mathbf{R}_i)_{0 \le i \le \mathrm{N}}$
Output: Cost function gradient $\nabla_{\mathbf{x}_0} \Psi$

1: **for** $i = 1$ to N **do**
2: Store forward model state $\mathbf{x}_{i-1}$:
3: Run forward model (6):

$$\mathbf{x}_i = \mathcal{M}_{i-1,i}(\mathbf{x}_{i-1});$$

4: **end for**

5: Initialize adjoint variables:

$$\boldsymbol{\lambda}_{\mathrm{N}}^{-} = \mathbf{H}_{\mathrm{N}}^{T} \mathbf{R}_{\mathrm{N}}^{-1} \left(\mathcal{H}_{\mathrm{N}}(\mathbf{x}_{\mathrm{N}}) - \mathrm{y}_{\mathrm{N}}^{\mathrm{obs}} \right).$$

6: **for** $i = \mathrm{N}$ down to 1 **do** ▷ (on each interval t_i to t_{i-1})
7: Load forward model state $\mathbf{x}_{i-1}$;
8: Construct and run adjoint model:

$$\boldsymbol{\lambda}_{i-1}^{+} = \mathbf{M}_{i-1,i}^{T}(\mathbf{x}_{i-1})\, \boldsymbol{\lambda}_{i}^{-}; \tag{14}$$

9: Add observation forcing at t_{i-1}:

$$\boldsymbol{\lambda}_{i-1}^{-} = \boldsymbol{\lambda}_{i-1}^{+} + \mathbf{H}_{i-1}^{T} \mathbf{R}_{i-1}^{-1} \left(\mathcal{H}_{i-1}(\mathbf{x}_{i-1}) - \mathrm{y}_{i-1}^{\mathrm{obs}} \right)$$

10: **end for**

11: Compute gradient (13):

$$\nabla_{\mathbf{x}_0} \Psi = \mathbf{B}_0^{-1} \left(\mathbf{x}_0 - \mathbf{x}_0^{\mathrm{b}} \right) + \boldsymbol{\lambda}_0^{-}.$$

3.3 The dynamical model

We next consider the case where the physical system dynamics is given by the forward ODE:

$$\mathbf{x}' = \mathbf{f}(t, \mathbf{x}, \boldsymbol{\mu}), \quad t_{i-1} \le t \le t_i, \quad \mathbf{x}(t_{i-1}) = \mathbf{x}_{i-1}, \tag{15}$$

and discuss the implementation of the adjoint evolution equation (14):

$$\boldsymbol{\lambda}_{i-1}^{+} = \mathbf{M}_{i-1,i}^{T}(\mathbf{x}_{i-1})\, \boldsymbol{\lambda}_{i}^{-}, \quad \mathbf{M}_{i-1,i}(\mathbf{x}_{i-1}) = \frac{d\mathcal{M}_{i-1,i}(\mathbf{x}_{i-1})}{d\mathbf{x}_{i-1}} \equiv \frac{d\mathbf{x}_i}{d\mathbf{x}_{i-1}}. \tag{16}$$

The dynamical model operator $\mathcal{M}_{i-1,i}(\mathbf{x}_{i-1})$ (16) advances the model solution from time t_{i-1}, when observations $\mathsf{y}_{i-1}^{\text{obs}}$ are available, to time t_i, when the next set of observations $\mathsf{y}_i^{\text{obs}}$ becomes available.

To advance the solution from time t_{i-1} to time t_i, the dynamical model operator $\mathcal{M}_{i-1,i}(\mathbf{x}_{i-1})$ (16) uses a numerical method to solve the forward ODE model (15). Specifically, the time interval $[t_{i-1}, t_i]$ is divided into K subintervals

$$t_{i-1} \equiv t_0 < \cdots < t_{k-1} < t_k < \cdots < t_K \equiv t_i, \quad h_k := t_k - t_{k-1}, \tag{17}$$

where $h_k = t_k - t_{k-1}$ is the k-th *time step*. One takes K steps of a numerical method $\mathcal{N}$ to discretize (15), i.e., to compute numerical approximations of the model state at the intermediate time points, $\mathsf{x}_k \approx \mathbf{x}(t_k)$. The numerical discretization reads:

$$\begin{aligned} &\mathsf{x}_0 = \mathbf{x}_{i-1}; \\ &\mathsf{x}_k = \mathcal{N}(\mathsf{x}_{k-1}, t_{k-1}, h_k, \mathbf{f}), \quad k = 1, \ldots, K; \\ &\mathbf{x}_i = \mathsf{x}_K. \end{aligned} \tag{18}$$

Examples of numerical schemes include forward Euler method, where $\mathcal{N}(\mathsf{x}_{k-1}, t_{k-1}, h_k, \mathbf{f})$ is defined explicitly:

$$\mathsf{x}_k = \underbrace{\mathsf{x}_{k-1} + h_k \mathbf{f}\left(t_{k-1}, \mathsf{x}_{k-1}\right)}_{\mathcal{N}(\mathsf{x}_{k-1}, t_{k-1}, h_k, \mathbf{f})}, \quad k = 1, \cdots, K, \tag{19}$$

and backward Euler method, where $\mathcal{N}(\mathsf{x}_{k-1}, t_{k-1}, h_k, \mathbf{f})$ is defined implicitly:

$$\mathsf{x}_k = \mathsf{x}_{k-1} + h_k \mathbf{f}\left(t_k, \mathsf{x}_k\right), \quad k = 1, \cdots, K. \tag{20}$$

3.4 Discrete adjoint approach

The adjoint variable evolution (16)

$$\boldsymbol{\lambda}_{i-1}^{+} = \mathbf{M}_{i-1,i}^{T}(\mathbf{x}_{i-1})\, \boldsymbol{\lambda}_i^{-} = \prod_{k=K}^{1} \left(\frac{d\mathcal{N}(\mathsf{x}_{k-1}, t_{k-1}, h_k, \mathbf{f})}{d\mathsf{x}_{k-1}} \right)^T \boldsymbol{\lambda}_i^{-} \tag{21}$$

can be implemented by the adjoint of the numerical discretization (18)

$$\begin{aligned} &\lambda_K = \boldsymbol{\lambda}_i^{-}; \\ &\lambda_{k-1} = \left(\frac{d\mathsf{x}_k}{d\mathsf{x}_{k-1}} \right)^T \lambda_k = \left(\frac{d\mathcal{N}(\mathsf{x}_{k-1}, t_{k-1}, h_k, \mathbf{f})}{d\mathsf{x}_{k-1}} \right)^T \lambda_k, \quad k = K, \ldots, 1; \\ &\boldsymbol{\lambda}_{i-1}^{+} = \lambda_0. \end{aligned} \tag{22}$$

The process (22) is called the discrete adjoint method.

The forward Euler method (19) has the following discrete adjoint (22):

$$\lambda_{k-1} = \lambda_k + h_k \, \mathbf{f}_{\mathbf{x}}^T \, (t_{k-1}, \mathsf{x}_{k-1}) \; \lambda_k, \tag{23}$$

which is an explicit equation for λ_{k-1}. The backward Euler method (20) has the following discrete adjoint (22):

$$\lambda_{k-1} = \lambda_k + h_k \, \mathbf{f}_{\mathbf{x}}^T \, (t_k, \mathsf{x}_k) \; \lambda_{k-1}, \tag{24}$$

which is a linearly implicit equation in λ_{k-1}.

3.5 Runge-Kutta numerical methods

One step of a general s-stage Runge-Kutta method advances the numerical solution (18) from t_{k-1} to t_k as follows:

$$\begin{aligned} T_{k;i} &= t_{k-1} + c_i \, h_k \\ \mathbf{Y}_{k;i} &= \mathsf{x}_{k-1} + h_k \sum_{j=1}^{s} a_{i,j} \, \mathbf{f} \, (\, T_{k;j}, \, \mathbf{Y}_{k;j} \,) \, , \quad i = 1, \ldots, s; \\ \mathsf{x}_k &= \mathsf{x}_{k-1} + h \sum_{i=1}^{s} b_i \, \mathbf{f} \, (\, T_{k;i}, \, \mathbf{Y}_{k;i} \,) \, , \end{aligned} \tag{25}$$

where the coefficients $a_{i,j}$, b_i and c_i are prescribed for the desired accuracy and stability properties. Equations (25) define the numerical method operator $\mathcal{N}(\mathsf{x}_{k-1}, t_{k-1}, h_k, \mathbf{f})$ in (18).

Any particular method is defined by its coefficients, represented compactly as three arrays of coefficients arranged into a Butcher tableau:

$$\mathbf{A} = (a_{i,j})_{1 \le i,j \le s}, \quad \mathbf{b} = (b_i)_{1 \le i \le s}, \quad \mathbf{c} = (c_i)_{1 \le i \le s}, \quad \text{Method} = \begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array}.$$

For example, for $s = 1$ we have:

$$\text{Forward Euler} = \begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}, \quad \text{Backward Euler} = \begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}.$$

For $s = 2$:

$$\text{Ralston} = \begin{array}{c|cc} 0 & 0 & 0 \\ 2/3 & 2/3 & 0 \\ \hline & 1/4 & 3/4 \end{array}.$$

For $s = 4$:

$$\text{Traditional RK4} = \begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}.$$

A direct calculation shows that the discrete adjoint (22) of the Runge-Kutta method advances the adjoint solution from t_k back to t_{k-1} as follows:

$$\mathbf{u}_i \;=\; h_k\, \mathbf{f}_{\mathbf{x}}^T\,(T_{k;i},\, \mathbf{Y}_{k;i}) \cdot \left(b_i\, \lambda_k + \sum_{j=1}^{s} a_{j,i}\, \mathbf{u}_j \right) , \quad i = s, \ldots, 1\,, \tag{26}$$

$$\lambda_{k-1} \;=\; \lambda_k + \sum_{j=1}^{s} \mathbf{u}_j \;.$$

Equations (26) define the numerical method operator $d\mathcal{N}(\mathsf{x}_{k-1})/d\mathsf{x}_{k-1}$ in (22). Note the following:

- The stages $\mathbf{u}_i$ in (26) are computed in reverse order;
- At each stage i the Jacobian transposed $\mathbf{f}_{\mathbf{x}}^T$ is evaluated at the stage values $T_{k;i}, \mathbf{Y}_{k;i}$ computed during the forward calculation (25).

The calculation of the adjoint operator action (14) using a Runge-Kutta method is summarized in Algorithm 3.2.

Algorithm 3.2 Application of adjoint operator $\mathbf{M}_{i-1,i}^T$ in (14)

Input: Forward model state $\mathbf{x}_{i-1}$ at t_{i-1};
Input: Adjoint model state $\boldsymbol{\lambda}_i^-$ at t_i
Output: Result of applying adjoint operator (14): $\boldsymbol{\lambda}_{i-1}^+ = \mathbf{M}_{i-1,i}^T \boldsymbol{\lambda}_i^-$

Initialize forward model:

$$\mathsf{x}_0 = \mathbf{x}_{i-1}; \tag{27}$$

for $k = 1$ to K **do** ▷ (run forward Runge-Kutta solver)
    Run one step of the Runge-Kutta scheme (25) to advance solution:

$$\mathsf{x}_{k-1} \mapsto \mathsf{x}_k.$$

    Save solution x_{k-1} and all stage variables $\left\{T_{k;i}, \mathbf{Y}_{k;i}\right\}_{i=1,\ldots,s}$.
end for

Initialize adjoint model:

$$\lambda_K = \boldsymbol{\lambda}_i^-; \tag{28}$$

for $k = K$ down to 1 **do** ▷ (run discrete adjoint Runge-Kutta solver)
    Load solution x_{k-1} and all stage variables $\left\{T_{k;i}, \mathbf{Y}_{k;i}\right\}_{i=1,\ldots,s}$.
    Run one step of the discrete adjoint Runge-Kutta scheme (26) to advance solution:

$$\lambda_k \mapsto \lambda_{k-1}.$$

end for
Obtain the adjoint operator solution:

$$\boldsymbol{\lambda}_{i-1}^+ = \mathbf{M}_{i-1,i}^T \boldsymbol{\lambda}_i^- = \lambda_0.$$

Finally, the 4D-Var optimization uses an iterative optimization method that computes iteratively approximations of the analysis state $\mathbf{x}_0^{\{k\}} \to \mathbf{x}_0^{\mathrm{a}}$. This is summarized in Algorithm 3.3.

Algorithm 3.3 4D-Var optimization

Input: Background $\mathbf{x}_0^{\mathrm{b}}, \mathbf{B}_0$;
Input: Observations $(t_i, \mathrm{y}_i^{\mathrm{obs}}, \mathbf{R}_i)_{0 \le i \le \mathrm{N}}$
Output: Analysis trajectory $\mathbf{x}_{0:\mathrm{N}}^{\mathrm{a}}$

1: Initialize the optimization:

$$\mathbf{x}_0^{\{0\}} = \mathbf{x}_0^{\mathrm{b}}; \tag{29}$$

2: **for** $k = 1, 2, \cdots$ **do** ▷ (run optimizer iterations)
3: Run Algorithm (3.1) starting with the initial condition $\mathbf{x}_0^{\{k\}}$, to compute the cost function value $\Psi\big(\mathbf{x}_0^{\{k\}}\big)$ and its gradient $\nabla_{\mathbf{x}_0}\Psi\big(\mathbf{x}_0^{\{k\}}\big)$
4: Run the optimizer to obtain the next iteration:

$$\mathbf{x}_0^{\{k+1\}} = OPT\big(\mathbf{x}_0^{\{k\}}, \Psi\big(\mathbf{x}_0^{\{k\}}\big), \nabla_{\mathbf{x}_0}\Psi\big(\mathbf{x}_0^{\{k\}}\big)\big).$$

5: Stop optimization iterations when solution is acceptable.
6: **end for**
7: Set the analysis initial condition $\mathbf{x}_0^{\mathrm{a}} = \mathbf{x}_0^{\{k\}}$ (solution at last iteration)
8: Run the model (6) starting from $\mathbf{x}_0^{\mathrm{a}}$ to obtain the analysis trajectory $\mathbf{x}_{0:\mathrm{N}}^{\mathrm{a}}$.

basic GD

4 Question 1: prepare the infrastructure

1. Implement the Lorenz'63 ODE function and Jacobian (1). Solve the Lorenz'63 system using Matlab's `ode45` function starting from (4).

2. Implement a Runge-Kutta method (either Ralston or RK4). Solve the Lorenz'63 system (1) using your Runge Kutta scheme with a fixed step sizes $h = 0.1/50$ (18) starting from (4). Plot your solution against Matlab's, and assess accuracy. If you need a more accurate solution decrease your step size to $h = 0.1/100$, $h = 0.1/200$, etc., until the result is acceptable.

3. Implement your own discrete adjoint Runge-Kutta method (26). Implement the action of the adjoint Lorenz'63 operator as described in ALgorithm 3.2, using 50 Runge-Kutta steps on each data assimilation interval $[t_{i-1}, t_i]$.

4. Implement Algorithm 3.1 for computing the 4D-Var adjoint gradient.

5. Validate your adjoint gradient as follows.

   - Run your Algorithm 3.1 for a single interval $[t_0, t_1]$, and compute $\nabla_{\mathbf{x}_0}\Psi(\mathbf{x}_0)$.
   - Run the Runge-Kutta model (18) starting from $\mathbf{x}_0$ (4) and compute $\Psi(\mathbf{x}_0)$.
   - Run the Runge-Kutta model (18) starting from $\mathbf{x}_0 + \varepsilon\mathbf{e}_k$ (4) and compute $\Psi(\mathbf{x}_0 + \varepsilon\mathbf{e}_k)$. Here $\mathbf{e}_k$ is the k-th column of the identity matrix, and we only perturb the k-th entry of the initial condition $\mathbf{x}_{0;k}$.
   - Check the k-th component of the adjoint gradient against the finite difference approximation:

$$\frac{d\Psi}{d\mathbf{x}_{o;k}} = \left(\nabla_{\mathbf{x}_0}\Psi(\mathbf{x}_0)\right)_k \approx \varepsilon^{-1}\left(\Psi(\mathbf{x}_0 + \varepsilon\mathbf{e}_k) - \Psi(\mathbf{x}_0)\right).$$

   - Repeat for all initial entries $k = 1, 2, 3$. If the accuracy of the gradient is unacceptable, re-check your adjoint computation. Getting a sufficiently accurate adjoint gradient (say, within 1% relative error in the finite difference test) is the most difficult step of the 4D-Var.

5 Question 2: perform 4D-Var data assimilation

1. Choose an assimilation window $[t_0, t_N]$ with $N \geq 2$.

2. Generate reference solution $\mathbf{x}^{\text{ref}}_{0:N}$ by solving the Lorenz'63 system using Matlab's `ode45` function starting from (4).

3. Generate synthetic observations from the reference solution by adding realizations of the observation error, $\mathbf{y}_i = \mathbf{x}^{\text{ref}}_i + \varepsilon^{\text{obs}}_i$. Here the observation operator is the identity operator, and observation errors have a Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{R})$. Choose your $\mathbf{R}$.

4. Choose an initial background state by a random draw from the normal distribution (4):

$$\mathbf{x}^{\text{b}}_0 \sim \mathcal{N}(\mathbf{x}_0, \mathbf{B}_0).$$

   Run the model (6) starting from $\mathbf{x}^{\text{b}}_0$ to obtain the background trajectory $\mathbf{x}^{\text{b}}_{0:N}$.

5. Select your numerical unconstrained optimization solver. Possible choices are:

   - The Poblano toolbox `https://github.com/sandialabs/poblano_toolbox`. Choose a quasi-Newton or a nonlinear conjugate gradients algorithm.
   - Use Matlab's built in optimization functions such as `fminunc`. Use `'SpecifyObjectiveGradient',true` option, and provide both the 4D-Var cost function and its gradient.

6. Run the 4D-Var algorithm 3.3. Compare your background trajectory $\mathbf{x}^{\text{b}}_{0:N}$, analysis trajectory $\mathbf{x}^{\text{a}}_{0:N}$, and reference trajectory $\mathbf{x}^{\text{ref}}_{0:N}$ by plotting them on the same graph. We want to see that the analysis is a better match to the reference than the background.

6 What to submit

1. A well articulated report detailing your work and the results you obtained, and

2. The well-organized, structured, and commented code you developed for each of the steps.