

Assimilation Algorithms

Lecture 2: 3D-Var

Mike Fisher

ECMWF

May 31, 2013

Outline

- 1 From Optimal Interpolation to 3D-Var
- 2 The Maximum Likelihood Approach
- 3 Minimization
- 4 Summary

From Optimal Interpolation to 3D-Var

- In my last lecture, we derived the **linear analysis equation**

$$\mathbf{x}_a = \mathbf{x}_b + \mathbf{K}(\mathbf{y} - \mathcal{H}(\mathbf{x}_b))$$

where

$$\mathbf{K} = \mathbf{P}^b \mathbf{H}^T \left[\mathbf{H} \mathbf{P}^b \mathbf{H}^T + \mathbf{R} \right]^{-1} \equiv \left[(\mathbf{P}^b)^{-1} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H} \right]^{-1} \mathbf{H}^T \mathbf{R}^{-1}$$

- Optimal Interpolation (OI) applies **direct solution methods** to invert the matrix $\left[\mathbf{H} \mathbf{P}^b \mathbf{H}^T + \mathbf{R} \right]$.
- Data selection is applied to reduce the dimension of the matrix.
- Direct methods require access to the matrix elements. In particular, $\mathbf{H} \mathbf{P}^b \mathbf{H}^T$ must be available in matrix form.
- This limits us to very simple observation operators.

From Optimal Interpolation to 3D-Var

- Iterative methods have significant advantages over the direct methods used in OI.
- They can be applied to much larger problems than direct techniques, so we can avoid data selection.
- They do not require access to the matrix elements.
- Typically, to solve $\mathbf{Ax} = \mathbf{b}$, requires only the ability to calculate **matrix-vector products**: \mathbf{Ax} .
- This allows us to use operators defined by pieces of code rather than explicitly as matrices.
- Examples of such operators include radiative-transfer codes, numerical models, Fourier transforms, etc.

Example: Conjugate Gradients

To solve $\mathbf{Ax} = \mathbf{b}$, where \mathbf{A} is real, symmetric and positive-definite:

$$\mathbf{r}_0 := \mathbf{b} - \mathbf{Ax}_0 \quad \mathbf{p}_0 := \mathbf{r}_0 \quad k := 0$$

repeat until \mathbf{r}_{k+1} is sufficiently small

$$\alpha_k := \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}$$

$$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

$$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{p}_k$$

$$\beta_k := \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$$

$$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$$

$$k := k + 1$$

The result is \mathbf{x}_{k+1}

From Optimal Interpolation to 3D-Var

- There are two ways to apply iterative methods to the linear analysis equation, depending which expression we adopt for \mathbf{K} :
- For $\mathbf{K} = \mathbf{P}^b \mathbf{H}^T [\mathbf{H} \mathbf{P}^b \mathbf{H}^T + \mathbf{R}]^{-1}$ we have:

$$\mathbf{x}_a = \mathbf{x}_b + \mathbf{P}^b \mathbf{H}^T \mathbf{z} \quad \text{where} \quad [\mathbf{H} \mathbf{P}^b \mathbf{H}^T + \mathbf{R}] \mathbf{z} = \mathbf{y} - \mathcal{H}(\mathbf{x}_b)$$

- For $\mathbf{K} = [(\mathbf{P}^b)^{-1} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}]^{-1} \mathbf{H}^T \mathbf{R}^{-1}$, we have:

$$\mathbf{x}_a = \mathbf{x}_b + \delta \mathbf{x} \quad \text{where} \quad [(\mathbf{P}^b)^{-1} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}] \delta \mathbf{x} = \mathbf{H}^T \mathbf{R}^{-1} (\mathbf{y} - \mathcal{H}(\mathbf{x}_b))$$

- The first of these alternatives is called **PSAS**
- The second (although it may not look like it yet) is **3D-Var**

3D-Var

- As we have seen, (linear) 3D-Var analysis can be seen as an application of iterative solution methods to the linear analysis equation.
- Historically, 3D-Var was not developed this way.
- We will now consider this alternative derivation.
- We will need to apply **Bayes' theorem**:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$

where $p(A|B)$ is the probability of A **given** B , etc.

Maximum Likelihood

- We developed the linear analysis equation by searching for a linear combination of observation and background that minimized the variance of the error.
- An alternative approach is to look for the most probable solution, given the background and observations:

$$\mathbf{x}_a = \arg \max_{\mathbf{x}} (p(\mathbf{x}|\mathbf{y} \text{ and } \mathbf{x}_b))$$

- It will be convenient to define a **cost function**

$$J = -\log (p(\mathbf{x}|\mathbf{y} \text{ and } \mathbf{x}_b)) + \text{const.}$$

- Then, since log is a monotonic function:

$$\mathbf{x}_a = \arg \min_{\mathbf{x}} (J(\mathbf{x}))$$

Maximum Likelihood

- Applying Bayes' theorem gives:

$$p(\mathbf{x}|\mathbf{y} \text{ and } \mathbf{x}_b) = \frac{p(\mathbf{y} \text{ and } \mathbf{x}_b|\mathbf{x})p(\mathbf{x})}{p(\mathbf{y} \text{ and } \mathbf{x}_b)}$$

- Now, $p(\mathbf{y} \text{ and } \mathbf{x}_b)$ is independent of \mathbf{x} .
- *A Priori* we know nothing about \mathbf{x} – all values of \mathbf{x} are equally likely.
- Hence, we can regard $p(\mathbf{x})/p(\mathbf{y} \text{ and } \mathbf{x}_b)$ as independent of \mathbf{x} , and write:

$$p(\mathbf{x}|\mathbf{y} \text{ and } \mathbf{x}_b) \propto p(\mathbf{y} \text{ and } \mathbf{x}_b|\mathbf{x})$$

- Furthermore, if observation errors and background errors are uncorrelated, then

$$p(\mathbf{y} \text{ and } \mathbf{x}_b|\mathbf{x}) = p(\mathbf{y}|\mathbf{x})p(\mathbf{x}_b|\mathbf{x})$$

$$\Rightarrow J(\mathbf{x}) = -\log(p(\mathbf{y}|\mathbf{x})) - \log(p(\mathbf{x}_b|\mathbf{x})) + \text{const.}$$

Maximum Likelihood

- The maximum likelihood approach is applicable to any probability density functions $p(\mathbf{y}|\mathbf{x})$ and $p(\mathbf{x}_b|\mathbf{x})$.
- However, let us consider the special case of Gaussian p.d.f.'s:

$$p(\mathbf{x}_b|\mathbf{x}) = \frac{1}{(2\pi)^{N/2} |\mathbf{P}_b|^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{x}_b - \mathbf{x})^T (\mathbf{P}_b)^{-1} (\mathbf{x}_b - \mathbf{x}) \right]$$
$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{(2\pi)^{M/2} |\mathbf{R}|^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{y} - \mathcal{H}(\mathbf{x}))^T \mathbf{R}^{-1} (\mathbf{y} - \mathcal{H}(\mathbf{x})) \right]$$

- Now, $J(\mathbf{x}) = -\log(p(\mathbf{y}|\mathbf{x})) - \log(p(\mathbf{x}_b|\mathbf{x})) + \text{const.}$
- Hence, with an appropriate choice of the constant *const.*:

$$J(\mathbf{x}) = \frac{1}{2} (\mathbf{x}_b - \mathbf{x})^T (\mathbf{P}_b)^{-1} (\mathbf{x}_b - \mathbf{x}) + \frac{1}{2} (\mathbf{y} - \mathcal{H}(\mathbf{x}))^T \mathbf{R}^{-1} (\mathbf{y} - \mathcal{H}(\mathbf{x}))$$

- This is the **3D-Var cost function**

Maximum Likelihood

$$J(\mathbf{x}) = \frac{1}{2} (\mathbf{x}_b - \mathbf{x})^T (\mathbf{P}_b)^{-1} (\mathbf{x}_b - \mathbf{x}) + \frac{1}{2} (\mathbf{y} - \mathcal{H}(\mathbf{x}))^T \mathbf{R}^{-1} (\mathbf{y} - \mathcal{H}(\mathbf{x}))$$

- The maximum likelihood analysis corresponds to the global minimum of the cost function
- At the minimum, the **gradient** of the cost function ($\nabla J(\mathbf{x})$ or $\partial J/\partial \mathbf{x}$) is zero:

$$\nabla J(\mathbf{x}) = (\mathbf{P}_b)^{-1} (\mathbf{x} - \mathbf{x}_b) + \mathbf{H}^T \mathbf{R}^{-1} (\mathcal{H}(\mathbf{x}) - \mathbf{y}) = \mathbf{0}$$

Maximum Likelihood

$$\nabla J(\mathbf{x}) = (\mathbf{P}_b)^{-1} (\mathbf{x} - \mathbf{x}_b) + \mathbf{H}^T \mathbf{R}^{-1} (\mathcal{H}(\mathbf{x}) - \mathbf{y}) = \mathbf{0}$$

- Now, if \mathcal{H} is linear (or if we neglect second-order terms) then

$$\mathcal{H}(\mathbf{x}) = \mathcal{H}(\mathbf{x}_b) + \mathbf{H}(\mathbf{x} - \mathbf{x}_b)$$

- Hence: $(\mathbf{P}_b)^{-1} (\mathbf{x} - \mathbf{x}_b) + \mathbf{H}^T \mathbf{R}^{-1} (\mathcal{H}(\mathbf{x}_b) + \mathbf{H}(\mathbf{x} - \mathbf{x}_b)) - \mathbf{y} = \mathbf{0}$
- Rearranging a little gives:

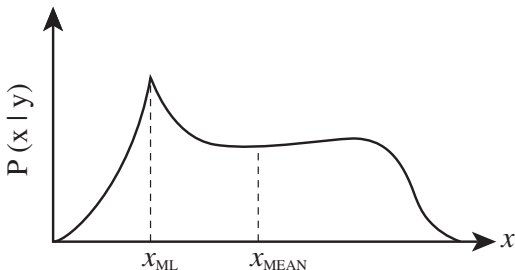
$$\left[(\mathbf{P}_b)^{-1} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H} \right] \delta \mathbf{x} = \mathbf{H}^T \mathbf{R}^{-1} (\mathbf{y} - \mathcal{H}(\mathbf{x}_b))$$

where $\delta \mathbf{x} = \mathbf{x} - \mathbf{x}_b$

- This is exactly the equation for the minimum-variance analysis we derived at the start of the lecture!

Maximum Likelihood

- We have shown that the **maximum likelihood** approach is naturally expressed in terms of a **cost function** representing minus the log of the probability of the analysis state.
- The minimum of the cost function corresponds to the maximum likelihood (probability) solution.
- For Gaussian errors and linear observation operators, the maximum likelihood analysis coincides with the minimum variance solution.
- This is not the case in general:



Maximum Likelihood

- In the nonlinear case, the minimum variance approach is difficult to apply.
- The maximum-likelihood approach is much more generally applicable
- As long as we know the p.d.f's, we can define the cost function
 - ▶ However, finding the global minimum may not be easy for highly non-Gaussian p.d.f's.
- In practice, background errors are usually assumed to be Gaussian (or a nonlinear transformation is applied to *make* them Gaussian).
 - ▶ This makes the background-error term of the cost function quadratic.
- However, non-Gaussian observation errors are taken into account. For example:
 - ▶ Directionally-ambiguous wind observations from scatterometers
 - ▶ Observations contaminated by occasional gross errors, which make outliers much more likely than implied by a Gaussian model.

Minimization

- In 3D-Var, the analysis is found by minimizing the cost function:

$$J(\mathbf{x}) = \frac{1}{2} (\mathbf{x}_b - \mathbf{x})^T (\mathbf{P}_b)^{-1} (\mathbf{x}_b - \mathbf{x}) + \frac{1}{2} (\mathbf{y} - \mathcal{H}(\mathbf{x}))^T \mathbf{R}^{-1} (\mathbf{y} - \mathcal{H}(\mathbf{x}))$$

- This is a very large-scale ($\dim(\mathbf{x}) \approx 10^8$) minimization problem.
- The size of the problem restricts on the algorithms we can use.
- Derivative-free algorithms (which require only the ability to calculate $J(\mathbf{x})$ for arbitrary \mathbf{x}) are too slow.
- This is because each function evaluation gives very limited information about the shape of the cost function.
 - ▶ E.g. a finite difference, $J(\mathbf{x} + \delta\mathbf{v}) - J(\mathbf{x}) \approx \delta\mathbf{v}^T \nabla J(\mathbf{x})$, gives a single component of the gradient.
 - ▶ We need $O(10^8)$ components to work out which direction is “downhill”.

Minimization

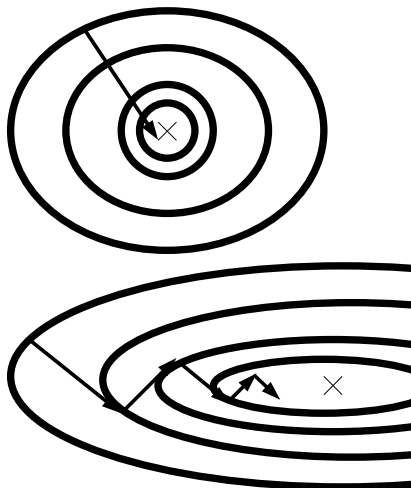
- Practical algorithms for minimizing the 3D-Var cost function require us to calculate its gradient:

$$\nabla J(\mathbf{x}) = (\mathbf{P}_b)^{-1} (\mathbf{x} - \mathbf{x}_b) + \mathbf{H}^T \mathbf{R}^{-1} (\mathcal{H}(\mathbf{x}) - \mathbf{y})$$

- The simplest gradient-based minimization algorithm is called **steepest descent**:
 - ▶ Let \mathbf{x}_0 be an initial guess of the analysis. Repeat the following steps for $k = 0, 1, 2$, etc. until the gradient is sufficiently small:
 - ▶ Define a **descent direction**: $\mathbf{d}_k = -\nabla J(\mathbf{x}_k)$.
 - ▶ Find a **step** α_k , e.g. by **line minimization** of the function $J(\mathbf{x}_k + \alpha \mathbf{d}_k)$, for which $J(\mathbf{x}_k + \alpha \mathbf{d}_k) < J(\mathbf{x}_k)$.
 - ▶ Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha \mathbf{d}_k$.

Minimization

- Steepest descent can work well on very well conditioned problems in which the iso-surfaces of the cost function are nearly spherical.
- In this case, the steepest descent direction points towards the minimum.
- For poorly conditioned problems, with ellipsoidal iso-surfaces, steepest descent is not efficient:



Minimization

- Steepest Descent is inefficient because it does not use information about the **curvature** (i.e. the second derivatives) of the cost function.
- The simplest algorithm that uses curvature is **Newtons method**.
- Newton's method uses a local quadratic approximation:

$$J(\mathbf{x} + \delta\mathbf{x}) \approx J(\mathbf{x}) + \delta\mathbf{x}^T \nabla J(\mathbf{x}) + \frac{1}{2} \delta\mathbf{x}^T J'' \delta\mathbf{x}$$

- Taking the gradient gives:

$$\nabla J(\mathbf{x} + \delta\mathbf{x}) \approx \nabla J(\mathbf{x}) + J'' \delta\mathbf{x}$$

- Since the gradient is zero at the minimum, Newton's method chooses the step at each iteration by solving:

$$J'' \delta\mathbf{x} = -\nabla J(\mathbf{x})$$

Minimization

- Newton's method:
 - ▶ Start with an initial guess, \mathbf{x}_0 .
 - ▶ Repeat the following steps for $k = 0, 1, 2$, etc. until the gradient is sufficiently small:
 - ▶ Solve $J''\delta\mathbf{x}_k = -\nabla J(\mathbf{x}_k)$.
 - ▶ Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \delta\mathbf{x}_k$.
- Newton's method works well for cost functions that are well approximated by a quadratic — i.e. for quasi-linear observation operators.
- However, it suffers from several problems. . .
- There is no control on the step length $\|\delta\mathbf{x}\|$. The method can make huge jumps into regions where the local quadratic approximation is poor.
 - ▶ This can be controlled using line searches, or by **trust region** methods that limit the step size to a region where the approximation is valid.

Minimization

- Newton's method requires us to solve $J''\delta\mathbf{x}_k = -\nabla J(\mathbf{x}_k)$ at every iteration.
- Now, J'' is a $\sim 10^8 \times 10^8$ matrix! Clearly, we cannot explicitly construct the matrix, or use direct methods to invert it.
- However, if we have a code that calculates Hessian-vector products, then we can use an iterative method (e.g. conjugate gradients) to solve for $\delta\mathbf{x}_k$.
- Such a code is called a **second order adjoint**. See Wang, Navon, LeDimet, Zou, 1992 Meteor. and Atmos. Phys. 50, pp3-20 for details.
- Alternatively, we can use a method that constructs an approximation to $(J'')^{-1}$.
- Methods based on approximations of J'' or $(J'')^{-1}$ are called **quasi-Newton** methods.

Minimization

- By far the most popular quasi-Newton method is the BFGS algorithm, named after its creators Broyden, Fletcher, Goldfarb and Shanno.
- The BFGS method builds up an approximation to the Hessian:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k \mathbf{s}_k^T} - \frac{\mathbf{B}_k \mathbf{s}_k (\mathbf{B}_k \mathbf{s}_k)^T}{\mathbf{s}_k \mathbf{B}_k \mathbf{s}_k^T}$$


where $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{y}_k = \nabla J(\mathbf{x}_{k+1}) - \nabla J(\mathbf{x}_k)$.

- The approximation is symmetric and positive definite, and satisfies

$$\nabla J(\mathbf{x}_{j+1}) - \nabla J(\mathbf{x}_j) = J''(\mathbf{x}_{j+1} - \mathbf{x}_j) \quad \text{for } j = 0, 1, \dots, k$$

- There is an explicit expression for the inverse of \mathbf{B}_k , which allows Newton's equation to be solved at the cost of $O(Nk)$ operations.

Minimization

- The BFGS quasi-Newton method:
 - ▶ Start with an initial guess at the solution, \mathbf{x}_0 , and an initial approximation of the Hessian (typically, $\mathbf{B}_0 = \mathbf{I}$).
 - ▶ Repeat the following steps for $k = 0, 1, 2$, etc. until the gradient is sufficiently small:
 - ▶ Solve the approximate Newton's equation, $\mathbf{B}_k \delta \mathbf{x}_k = -\nabla J(\mathbf{x}_k)$, to determine the search direction.
 - ▶ Perform a line search to find a step α_k for which for which $J(\mathbf{x}_k + \alpha_k \delta \mathbf{x}_k) < J(\mathbf{x}_k)$.
 - ▶ Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \delta \mathbf{x}_k$.
 - ▶ Generate an updated approximation to the Hessian: \mathbf{B}_{k+1} .
- As k increases, the cost of storing and applying the approximate Hessian increases linearly.
- Moreover, the vectors \mathbf{s}_k and \mathbf{y}_k generated many iterations ago no longer provide accurate information about the Hessian.
- It is usual to construct \mathbf{B}_k from only the $O(10)$ most recent iterations. The algorithm is then called the **limited memory BFGS method**. 

Minimization

- The methods presented so far apply to general nonlinear functions.
- An important special case occurs if the observation operator \mathcal{H} is linear. In this case, the cost function is strictly quadratic, and the gradient is linear:

$$\begin{aligned}\nabla J(\mathbf{x}) &= (\mathbf{P}_b)^{-1} \delta \mathbf{x} + \mathbf{H}^T \mathbf{R}^{-1} (\mathcal{H}(\mathbf{x}_b) + \mathbf{H} \delta \mathbf{x} - \mathbf{y}) \\ &= \left[(\mathbf{P}_b)^{-1} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H} \right] \delta \mathbf{x} + \mathbf{H}^T \mathbf{R}^{-1} (\mathcal{H}(\mathbf{x}_b) - \mathbf{y})\end{aligned}$$

- In this case, it makes sense to determine the analysis by solving the **linear** equation $\nabla J(\mathbf{x}) = \mathbf{0}$.
- Since the matrix $\left[(\mathbf{P}_b)^{-1} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H} \right]$ is symmetric and positive definite, the best algorithm to use is **conjugate gradients**. (The algorithm was presented earlier in this lecture.)
- A good introduction to the method can be found online: Shewchuk (1994) “An Introduction to the Conjugate Gradient Method Without the Agonizing pain”.

Preconditioning

- We noted that the steepest descent method works best if the iso-surfaces of the cost function are approximately spherical.
- This is generally true of all minimization algorithms.
- In general, expressing the cost function directly in terms of \mathbf{x} will not lead to spherical iso-surfaces.
- The degree of sphericity of the cost function can be measured by the eigenvalues of the Hessian. (Each eigenvalue corresponds to the curvature in the direction of the corresponding eigenvector.)
- In particular, the convergence rate will depend on the **condition number**:

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$$

Preconditioning

- We can speed up the convergence of the minimization by a **change of variables** $\chi = \mathbf{L}^{-1}(\mathbf{x} - \mathbf{x}_b)$, where \mathbf{L} is chosen to make the cost function more spherical.
- A common choice is $\mathbf{L} = (\mathbf{P}_b)^{1/2}$. The cost function becomes:

$$J(\chi) = \frac{1}{2}\chi^T\chi + \frac{1}{2}(\mathbf{y} - \mathcal{H}(\mathbf{x}_b + \mathbf{L}\chi))^T \mathbf{R}^{-1}(\mathbf{y} - \mathcal{H}(\mathbf{x}_b + \mathbf{L}\chi))$$

- With this change of variables, the Hessian becomes:

$$J''_{\chi} = \mathbf{I} + \mathbf{L}^T \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H} \mathbf{L} \text{ (plus higher order terms)}$$

- The presence of the identity matrix in this expression guarantees that the minimum eigenvalue is ≥ 1 .
- There are no small eigenvalues to destroy the conditioning of the problem.

Calculating the Gradient

- To minimize the cost function, we must be able to calculate gradients.
- If we precondition using \mathbf{L} , the gradient (with respect to χ) is:

$$\nabla_{\chi} J(\chi) = \chi + \mathbf{L}^T \mathbf{H}^T \mathbf{R}^{-1} (\mathbf{y} - \mathcal{H}(\mathbf{x}_b + \mathbf{L}\chi))$$

- Typically, \mathbf{R} is diagonal — observation errors are treated as being mutually uncorrelated.
- However, the matrices \mathbf{H}^T , \mathbf{L}^T and \mathbf{L} are not diagonal, and are much too large to be represented explicitly.
- We must represent these as operators (subroutines) that calculate matrix-vector products.

Calculating the Gradient

- Take \mathcal{H} as an example. Each line of the subroutine that applies \mathcal{H} can be considered as a function h_k , so that

$$\mathcal{H}(\mathbf{x}) \equiv h_K (h_{K-1} (\cdots (h_1(\mathbf{x}))))$$

- Each of the functions h_k can be linearized, to give the corresponding linear function \mathbf{h}_k . Each of these is extremely simple, and can be represented by a one or two lines of code.
- The resulting code is called the **tangent linear** of \mathcal{H} .

$$\mathbf{H}(\mathbf{x}) \equiv \mathbf{h}_K \mathbf{h}_{K-1} \cdots \mathbf{h}_1 \mathbf{x}$$

- The transpose, $\mathbf{H}^T(\mathbf{x}) \equiv \mathbf{h}_1^T \mathbf{h}_2^T \cdots \mathbf{h}_K^T \mathbf{x}$, is called the **adjoint** of \mathcal{H} .
- Again, each \mathbf{h}_k^T is extremely simple — just a few lines of code.
- NB: there is a whole 1-hour lecture on tangent linear and adjoint operators later in the course.

Summary

- We showed that 3D-Var can be considered as an iterative procedure for solving the linear (minimum variance) analysis equation.
- We also derived 3D-Var from the **maximum likelihood principle**.
- The Maximum Likelihood approach can be applied to non-Gaussian, nonlinear analysis.
- We introduced the 3D-Var **cost function**.
- We considered how to minimize the cost function using algorithms based on knowledge of its gradient.
- We looked at a simple preconditioning.
- Finally, we saw how it is possible to write code that computes the gradient.