# Parallel Algorithms for Four-Dimensional Variational Data Assimilation

Mike Fisher

ECMWF

October 24, 2011

# Brief Introduction to 4D-Var

- Four-Dimensional Variational Data Assimilation 4D-Var is the method used by most national and international Numerical Weather Forecasting Centres to provide initial conditions for their forecast models.
- 4D-Var combines observations with a prior estimate of the state, provided by an earlier forecast.
- The method is described as Four-Dimensional because it takes into account observations that are distributed in space and over an interval of time (typically 6 or 12 hours), often called the analysis window.
- It does this by using a complex and computationally expensive numerical model to propagate information in time.
- In many applications of 4D-Var, the model is assumed to be perfect.
- In this talk, I will concentrate on so-called weak-constraint 4D-Var, which takes into account imperfections in the model.

# Brief Introduction to 4D-Var

- Weak-Constraint 4D-Var represents the data-assimilation problem as a very large least-squares problem.

$$
\begin{aligned}
J(x_0, x_1, \ldots, x_N) \quad = \quad & \frac{1}{2}(x_0 - x_b)^{\mathrm{T}} B^{-1} (x_0 - x_b) \\
& + \frac{1}{2} \sum_{k=0}^{N} (y_k - \mathcal{H}_k(x_k))^{\mathrm{T}} R_k^{-1} (y_k - \mathcal{H}_k(x_k)) \\
& + \frac{1}{2} \sum_{k=1}^{N} (q_k - \bar{q})^{\mathrm{T}} Q_k^{-1} (q_k - \bar{q})
\end{aligned}
$$

where $q_k = x_k - \mathcal{M}_k(x_{k-1})$.

- Here, the cost function $J$ is a function of the states $x_0, x_1, \ldots, x_N$ defined at the start of each of a set of sub-windows that span the analysis window.

# Brief Introduction to 4D-Var

$$
\begin{aligned}
J(x_0, x_1, \ldots, x_N) \quad = \quad & \frac{1}{2}(x_0 - x_b)^{\mathrm{T}} B^{-1} (x_0 - x_b) \\
& + \frac{1}{2} \sum_{k=0}^{N} (y_k - \mathcal{H}_k(x_k))^{\mathrm{T}} R_k^{-1} (y_k - \mathcal{H}_k(x_k)) \\
& + \frac{1}{2} \sum_{k=1}^{N} (q_k - \bar{q})^{\mathrm{T}} Q_k^{-1} (q_k - \bar{q})
\end{aligned}
$$

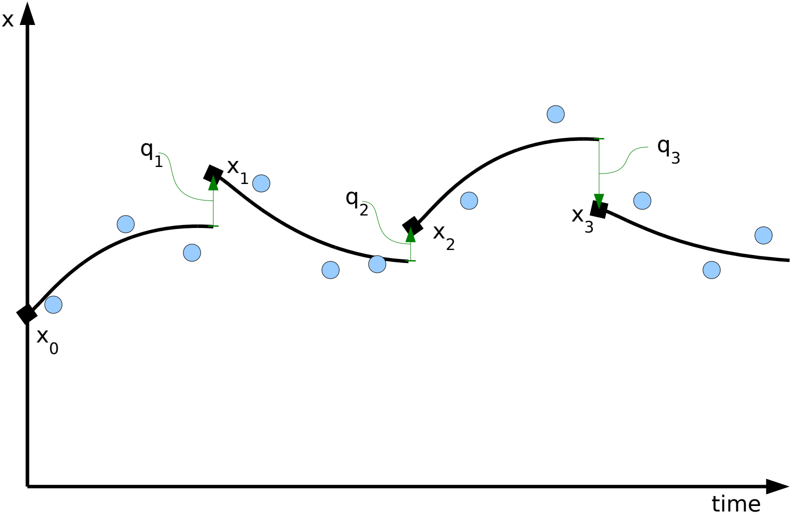
- Each $x_i$ contains $\approx 10^7$ elements.
- Each $y_i$ contains $\approx 10^5$ elements.
- The operators $\mathcal{H}_k$ and $\mathcal{M}_k$, and the matrices $B$, $R_k$ and $Q_k$ are represented by codes that apply them to vectors.
- We do not have access to their elements.

# Brief Introduction to 4D-Var

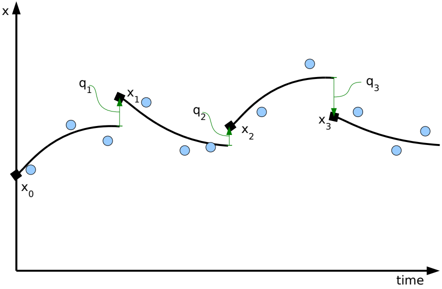
$$
\begin{aligned}
J(x_0, x_1, \ldots, x_N) \quad = \quad & \frac{1}{2}\,(x_0 - x_b)^{\mathrm{T}}\, B^{-1}\,(x_0 - x_b) \\
& + \frac{1}{2} \sum_{k=0}^{N} (y_k - \mathcal{H}_k(x_k))^{\mathrm{T}}\, R_k^{-1}\,(y_k - \mathcal{H}_k(x_k)) \\
& + \frac{1}{2} \sum_{k=1}^{N} (q_k - \bar{q})^{\mathrm{T}}\, Q_k^{-1}\,(q_k - \bar{q})
\end{aligned}
$$

- $\mathcal{H}_k$ and $\mathcal{M}_k$ involve integrations of the numerical model, and are computationally expensive.
- The covariance matrices $B$, $R_k$ and $Q_k$ are less expensive to apply.

# Brief Introduction to 4D-Var



# Brief Introduction to 4D-Var




The cost function contains 3 terms:

- $\frac{1}{2}(x_0 - x_b)^{\mathrm{T}} B^{-1} (x_0 - x_b)$ ensures that $x_0$ stays close to the prior estimate.
- $\frac{1}{2}\sum_{k=0}^{N}(y_k - \mathcal{H}_k(x_k))^{\mathrm{T}} R_k^{-1} (y_k - \mathcal{H}_k(x_k))$ keeps the estimate close to the observations (blue circles).
- $\frac{1}{2}\sum_{k=1}^{N}(q_k - \bar{q})^{\mathrm{T}} Q_k^{-1} (q_k - \bar{q})$ keeps the jumps between sub-windows small.

# Gauss-Newton (Incremental) Algorithm

- It is usual to minimize the cost function using a modified Gauss-Newton algorithm.
- Nearly all the computational cost is in the inner loop, which minimizes the quadratic cost function:

$$
\begin{aligned}
\hat{J}(\delta x_0^{(n)}, \ldots, \delta x_N^{(n)}) \quad = \quad & \frac{1}{2}\left(\delta x_0 - b^{(n)}\right)^{\mathrm{T}} B^{-1} \left(\delta x_0 - b^{(n)}\right) \\
& + \frac{1}{2} \sum_{k=0}^{N} \left(H_k^{(n)} \delta x_k - d_k^{(n)}\right)^{\mathrm{T}} R_k^{-1} \left(H_k^{(n)} \delta x_k - d_k^{(n)}\right) \\
& + \frac{1}{2} \sum_{k=1}^{N} \left(\delta q_k - c_k^{(n)}\right)^{\mathrm{T}} Q_k^{-1} \left(\delta q_k - c_k^{(n)}\right)
\end{aligned}
$$

$\delta q_k = \delta x_k - M_k^{(n)} \delta x_{k-1}$,

and where $b^{(n)}$, $c_k^{(n)}$ and $d_k^{(n)}$ come from the outer loop:

$$
b^{(n)} = x_b - x_0^{(n)}, \qquad c_k^{(n)} = \bar{q} - q_k^{(n)}, \qquad d_k^{(n)} = y_k - \mathcal{H}_k(x_k^{(n)})
$$

# Gauss-Newton (Incremental) Algorithm

$$
\begin{aligned}
\hat{J}(\delta x_0^{(n)}, \ldots, \delta x_N^{(n)}) \quad = \quad & \frac{1}{2}\left(\delta x_0 - b^{(n)}\right)^{\mathrm{T}} B^{-1}\left(\delta x_0 - b^{(n)}\right) \\
& + \frac{1}{2}\sum_{k=0}^{N}\left(H_k^{(n)}\delta x_k - d_k^{(n)}\right)^{\mathrm{T}} R_k^{-1}\left(H_k^{(n)}\delta x_k - d_k^{(n)}\right) \\
& + \frac{1}{2}\sum_{k=1}^{N}\left(\delta q_k - c_k^{(n)}\right)^{\mathrm{T}} Q_k^{-1}\left(\delta q_k - c_k^{(n)}\right)
\end{aligned}
$$

- Note that 4D-Var requires tangent linear versions of $\mathcal{M}_k$ and $\mathcal{H}_k$:
  - $M_k^{(n)}$ and $H_k^{(n)}$, respectively
- It also requires the transposes (adjoints) of these operators:
  - $(M_k^{(n)})^{\mathrm{T}}$ and $(H_k^{(n)})^{\mathrm{T}}$, respectively

# Why do we need more parallel algorithms?

- In its usual implementation, 4D-Var is solved by applying a conjugate-gradient solver.
- This is highly sequential:
  - Iterations of CG.
  - Tangent Linear and Adjoint integrations run one after the other.
  - Model timesteps follow each other.
- Computers are becoming ever more parallel, but processors are not getting faster.
- Unless we do something to make 4D-Var more parallel, we will soon find that 4D-Var becomes un-affordable (even with a 12-hour window).
- We cannot make the model more parallel.
  - The inner loops of 4D-Var run with a few 10's of grid columns per processor.
  - This is barely enough to mask inter-processor communication costs.
- We have to use more parallel algorithms.

# Parallelising within an Iteration

- The model is already parallel in both horizontal directions.
- The modellers tell us that it will be hard to parallelise in the vertical (and we already have too little work per processor).
- We are left with parallelising in the time direction.

# Weak Constraint 4D-Var: Inner Loop

Dropping the outer loop index ($n$), the inner loop of weak-constraints 4D-Var minimises:

$$
\begin{aligned}
\hat{J}(\delta x_0, \ldots, \delta x_N) \quad = \quad & \frac{1}{2}(\delta x_0 - b)^{\mathrm{T}} B^{-1} (\delta x_0 - b) \\
& + \frac{1}{2} \sum_{k=0}^{N} (H_k \delta x_k - d_k)^{\mathrm{T}} R_k^{-1} (H_k \delta x_k - d_k) \\
& + \frac{1}{2} \sum_{k=1}^{N} (\delta q_k - c_k)^{\mathrm{T}} Q_k^{-1} (\delta q_k - c_k)
\end{aligned}
$$

where $\delta q_k = \delta x_k - M_k \delta x_{k-1}$,
and where $b$, $c_k$ and $d_k$ come from the outer loop:

$$
\begin{aligned}
b &= x_b - x_0 \\
c_k &= \bar{q} - q_k \\
d_k &= y_k - \mathcal{H}_k(x_k)
\end{aligned}
$$

ECMWF

# Weak Constraint 4D-Var: Inner Loop

We can simplify this further by defining some 4D vectors and matrices:

$$\delta \mathbf{x} = \begin{pmatrix} \delta x_0 \\ \delta x_1 \\ \vdots \\ \delta x_N \end{pmatrix} \qquad \delta \mathbf{p} = \begin{pmatrix} \delta x_0 \\ \delta q_1 \\ \vdots \\ \delta q_N \end{pmatrix}$$

These vectors are related through $\delta q_k = \delta x_k - M_k \delta x_{k-1}$.
We can write this relationship in matrix form as:

$$\boxed{\delta \mathbf{p} = \mathbf{L} \delta \mathbf{x}}$$

where:

$$\mathbf{L} = \begin{pmatrix} I & & & & \\ -M_1 & I & & & \\ & -M_2 & I & & \\ & & \ddots & \ddots & \\ & & & -M_N & I \end{pmatrix}$$

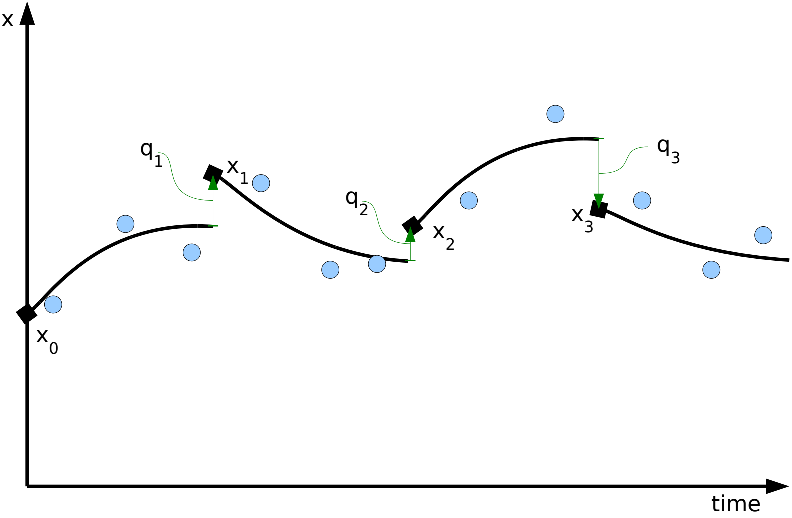
# Weak Constraint 4D-Var: Inner Loop

$$\mathbf{L} = \begin{pmatrix} I & & & & \\ -M_1 & I & & & \\ & -M_2 & I & & \\ & & \ddots & \ddots & \\ & & & -M_N & I \end{pmatrix}$$

$\delta\mathbf{p} = \mathbf{L}\delta\mathbf{x}$ can be done in parallel: $\delta q_k = \delta x_k - M_k \delta x_{k-1}$.
We know all the $\delta x_{k-1}{}'s$. We can apply all the $M_k{}'s$ simultaneously.

$\delta\mathbf{x} = \mathbf{L}^{-1}\delta\mathbf{p}$ is sequential: $\delta x_k = M_k \delta x_{k-1} + \delta q_k$.
We have to generate each $\delta x_{k-1}$ in turn before we can apply the next $M_k$.

# Brief Introduction to 4D-Var



# Weak Constraint 4D-Var: Inner Loop

We will also define:

$$\mathbf{R} = \begin{pmatrix} R_0 & & & \\ & R_1 & & \\ & & \ddots & \\ & & & R_N \end{pmatrix}, \qquad \mathbf{D} = \begin{pmatrix} B & & & \\ & Q_1 & & \\ & & \ddots & \\ & & & Q_N \end{pmatrix},$$

$$\mathbf{H} = \begin{pmatrix} H_0 & & & \\ & H_1 & & \\ & & \ddots & \\ & & & H_N \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} b \\ c_1 \\ \vdots \\ c_N \end{pmatrix} \qquad \mathbf{d} = \begin{pmatrix} d_0 \\ d_1 \\ \vdots \\ d_N \end{pmatrix}.$$

ECMWF

# Weak Constraint 4D-Var: Inner Loop

With these definitions, we can write the inner-loop cost function either as a function of $\delta\mathbf{x}$:

$$\boxed{J(\delta\mathbf{x}) = (\mathbf{L}\delta\mathbf{x} - \mathbf{b})^{\mathrm{T}}\mathbf{D}^{-1}(\mathbf{L}\delta\mathbf{x} - \mathbf{b}) + (\mathbf{H}\delta\mathbf{x} - \mathbf{d})^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{H}\delta\mathbf{x} - \mathbf{d})}$$

Or as a function of $\delta\mathbf{p}$:

$$\boxed{J(\delta\mathbf{p}) = (\delta\mathbf{p} - \mathbf{b})^{\mathrm{T}}\mathbf{D}^{-1}(\delta\mathbf{p} - \mathbf{b}) + (\mathbf{H}\mathbf{L}^{-1}\delta\mathbf{p} - \mathbf{d})^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{H}\mathbf{L}^{-1}\delta\mathbf{p} - \mathbf{d})}$$

# Forcing Formulation

$$\boxed{J(\delta\mathbf{p}) = (\delta\mathbf{p} - \mathbf{b})^{\mathrm{T}}\mathbf{D}^{-1}(\delta\mathbf{p} - \mathbf{b}) + (\mathbf{HL}^{-1}\delta\mathbf{p} - \mathbf{d})^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{HL}^{-1}\delta\mathbf{p} - \mathbf{d})}$$

- This version of the cost function is sequential.
  - It contains $\mathbf{L}^{-1}$.
- It closely resembles strong-constraint 4D-Var.
- We can precondition it using $\mathbf{D}^{1/2}$:

$$J(\chi) = \chi^{\mathrm{T}}\chi + (\mathbf{HL}^{-1}\delta\mathbf{p} - \mathbf{d})^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{HL}^{-1}\delta\mathbf{p} - \mathbf{d})$$

  where $\delta\mathbf{p} = \mathbf{D}^{1/2}\chi + \mathbf{b}$.
- This guarantees that the eigenvalues of $J''$ are bounded away from zero.
- We understand how to minimise this.

# 4D State Formulation

$$\boxed{J(\delta \mathbf{x}) = (\mathbf{L}\delta \mathbf{x} - \mathbf{b})^{\mathrm{T}} \mathbf{D}^{-1} (\mathbf{L}\delta \mathbf{x} - \mathbf{b}) + (\mathbf{H}\delta \mathbf{x} - \mathbf{d})^{\mathrm{T}} \mathbf{R}^{-1} (\mathbf{H}\delta \mathbf{x} - \mathbf{d})}$$

- This version of the cost function is parallel.
  - It does not contain $\mathbf{L}^{-1}$.
- We could precondition it using $\delta \mathbf{x} = \mathbf{L}^{-1}(\mathbf{D}^{1/2}\chi + \mathbf{b})$.
- This would give exactly the same $J(\chi)$ as before.
- But, we have introduced a sequential model integration (i.e. $\mathbf{L}^{-1}$) into the preconditioner.

# Plan A: State Formulation, Approximate Preconditioner

- In the forcing ($\delta\mathbf{p}$) formulation, and in its Lagrangian dual formulation (4D-PSAS) $\mathbf{L}^{-1}$ appears in the cost function.
  - These formulations are inherently sequential.
  - We cannot modify the cost function without changing the problem.
- In the 4D-state ($\delta\mathbf{x}$) formulation, $\mathbf{L}^{-1}$ appears in the preconditioner.
  - We are free to modify the preconditioner as we wish.
- This suggests we replace $\mathbf{L}^{-1}$ by a cheap approximation:

$$\delta\mathbf{x} = \tilde{\mathbf{L}}^{-1}(\mathbf{D}^{1/2}\chi + \mathbf{b})$$

- If we do this, we can no longer write the first term as: $\chi^{\mathrm{T}}\chi$.
- We have to calculate $\delta\mathbf{x}$, and explicity evaluate it as

$$(\mathbf{L}\delta\mathbf{x} - \mathbf{b})^{\mathrm{T}}\mathbf{D}^{-1}(\mathbf{L}\delta\mathbf{x} - \mathbf{b})$$

- This is where we run into problems: $\mathbf{D}$ is very ill-conditioned.

# Plan A: State Formulation, Approximate Preconditioner

- When we approximate $\mathbf{L}^{-1}$ in the preconditioner, the Hessian of the first term of the cost function (with respect to $\chi$) is no longer the identity matrix, but:

$$\mathbf{D}^{\mathrm{T}/2}\tilde{\mathbf{L}}^{-\mathrm{T}}\mathbf{L}^{\mathrm{T}}\mathbf{D}^{-1}\mathbf{L}\tilde{\mathbf{L}}^{-1}\mathbf{D}^{1/2}$$

- Because $\mathbf{D}$ is ill-conditioned, This is likely to have some very small (and some very large) eigenvalues, unless $\tilde{\mathbf{L}}$ is a very good approximation for $\mathbf{L}$.
- So far, I have not found a preconditioner that gives condition numbers for the minimisation less than $\mathrm{O}(10^9)$.
- We need a Plan B!

# Plan B: Saddle Point Formulation

$$\boxed{J(\delta\mathbf{x}) = (\mathbf{L}\delta\mathbf{x} - \mathbf{b})^{\mathrm{T}}\mathbf{D}^{-1}(\mathbf{L}\delta\mathbf{x} - \mathbf{b}) + (\mathbf{H}\delta\mathbf{x} - \mathbf{d})^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{H}\delta\mathbf{x} - \mathbf{d})}$$

At the minimum:

$$\nabla J = \mathbf{L}^{\mathrm{T}}\mathbf{D}^{-1}(\mathbf{L}\delta\mathbf{x} - \mathbf{b}) + \mathbf{H}^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{H}\delta\mathbf{x} - \mathbf{d}) = \mathbf{0}$$

Define:

$$\lambda = \mathbf{D}^{-1}(\mathbf{b} - \mathbf{L}\delta\mathbf{x}), \qquad \mu = \mathbf{R}^{-1}(\mathbf{d} - \mathbf{H}\delta\mathbf{x})$$

Then:

$$\left.\begin{array}{rcl} \mathbf{D}\lambda + \mathbf{L}\delta\mathbf{x} & = & \mathbf{b} \\ \mathbf{R}\mu + \mathbf{H}\delta\mathbf{x} & = & \mathbf{d} \\ \mathbf{L}^{\mathrm{T}}\lambda + \mathbf{H}^{\mathrm{T}}\mu & = & \mathbf{0} \end{array}\right\} \Longrightarrow \begin{pmatrix} \mathbf{D} & \mathbf{0} & \mathbf{L} \\ \mathbf{0} & \mathbf{R} & \mathbf{H} \\ \mathbf{L}^{\mathrm{T}} & \mathbf{H}^{\mathrm{T}} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \\ \delta\mathbf{x} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ \mathbf{d} \\ \mathbf{0} \end{pmatrix}$$

# Saddle Point Formulation

$$\boxed{\begin{pmatrix} \mathbf{D} & \mathbf{0} & \mathbf{L} \\ \mathbf{0} & \mathbf{R} & \mathbf{H} \\ \mathbf{L}^{\mathrm{T}} & \mathbf{H}^{\mathrm{T}} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \\ \delta\mathbf{x} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ \mathbf{d} \\ \mathbf{0} \end{pmatrix}}$$

- This is called the saddle point formulation of 4D-Var.
- The matrix is a saddle point matrix.
- The matrix is real, symmetric, indefinite.
- Note that the matrix contains no inverse matrices.
- We can apply the matrix without requiring a sequential model integration (i.e. we can parallelise over sub-windows).
- We can hope that the problem is well conditioned (since we don't multiply by $\mathbf{D}^{-1}$).

# Saddle Point Formulation

Alternative derivation:

$$\min_{\delta\mathbf{p},\delta\mathbf{w}} J(\delta\mathbf{p},\delta\mathbf{w}) = (\delta\mathbf{p}-\mathbf{b})^{\mathrm{T}}\mathbf{D}^{-1}(\delta\mathbf{p}-\mathbf{b}) + (\delta\mathbf{w}-\mathbf{d})^{\mathrm{T}}\mathbf{R}^{-1}(\delta\mathbf{w}-\mathbf{d})$$

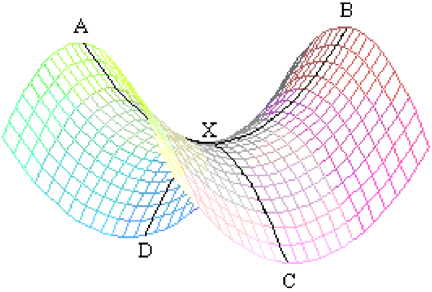
subject to $\delta\mathbf{p} = \mathbf{L}\delta\mathbf{x}$ and $\delta\mathbf{w} = \mathbf{H}\delta\mathbf{x}$.

$$\begin{aligned}\mathcal{L}(\delta\mathbf{x},\delta\mathbf{p},\delta\mathbf{w},\lambda,\mu) = {} & (\delta\mathbf{p}-\mathbf{b})^{\mathrm{T}}\mathbf{D}^{-1}(\delta\mathbf{p}-\mathbf{b}) + (\delta\mathbf{w}-\mathbf{d})^{\mathrm{T}}\mathbf{R}^{-1}(\delta\mathbf{w}-\mathbf{d}) \\ & +\lambda^{\mathrm{T}}(\delta\mathbf{p}-\mathbf{L}\delta\mathbf{x}) + \mu^{\mathrm{T}}(\delta\mathbf{w}-\mathbf{H}\delta\mathbf{x})\end{aligned}$$

- $\frac{\partial\mathcal{L}}{\partial\lambda} = \mathbf{0} \Rightarrow \quad \delta\mathbf{p} = \mathbf{L}\delta\mathbf{x}$
- $\frac{\partial\mathcal{L}}{\partial\mu} = \mathbf{0} \Rightarrow \quad \delta\mathbf{w} = \mathbf{H}\delta\mathbf{x}$
- $\frac{\partial\mathcal{L}}{\partial\delta\mathbf{p}} = \mathbf{0} \Rightarrow \quad \mathbf{D}^{-1}(\delta\mathbf{p}-\mathbf{b}) + \lambda = \mathbf{0}$
- $\frac{\partial\mathcal{L}}{\partial\delta\mathbf{w}} = \mathbf{0} \Rightarrow \quad \mathbf{R}^{-1}(\delta\mathbf{w}-\mathbf{d}) + \mu = \mathbf{0}$
- $\frac{\partial\mathcal{L}}{\partial\delta\mathbf{x}} = \mathbf{0} \Rightarrow \quad \mathbf{L}^{\mathrm{T}}\lambda + \mathbf{H}^{\mathrm{T}}\mu = \mathbf{0}$

ECMWF

# Saddle Point Formulation




Lagrangian: $\mathcal{L}(\delta\mathbf{x}, \delta\mathbf{p}, \delta\mathbf{w}, \lambda, \mu)$

- 4D-Var solves the primal problem: minimise along AXB.
- 4D-PSAS solves the Lagrangian dual problem: maximise along CXD.
- The saddle point formulation finds the saddle point of $\mathcal{L}$.
- The saddle point formulation is neither 4D-Var nor 4D-PSAS.

# Saddle Point Formulation

- To solve the saddle point system, we have to precondition it.
- Preconditioning saddle point systems is the subject of much current research. It is something of a dark art!
  - See e.g. Benzi and Wathen (2008), Benzi, Golub and Liesen (2005).
- Most preconditioners in the literature assume that $\mathbf{D}$ and $\mathbf{R}$ are expensive, and $\mathbf{L}$ and $\mathbf{H}$ are cheap.
- The opposite is true in 4D-Var!

Example: Diagonal Preconditioner:

$$\mathcal{P}_D = \begin{pmatrix} \hat{\mathbf{D}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{R}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & -\mathbf{S} \end{pmatrix}$$

where $\mathbf{S} \approx -\mathbf{L}^{\mathrm{T}}\mathbf{D}^{-1}\mathbf{L} - \mathbf{H}^{\mathrm{T}}\mathbf{R}^{-1}\mathbf{H}$

# Saddle Point Formulation

- One possibility is an approximate constraint preconditioner (Bergamaschi, *et al.*, 2007 & 2011):

$$\boxed{\tilde{\mathcal{P}} = \begin{pmatrix} \mathbf{D} & \mathbf{0} & \tilde{\mathbf{L}} \\ \mathbf{0} & \mathbf{R} & \mathbf{0} \\ \tilde{\mathbf{L}}^{\mathrm{T}} & \mathbf{0} & \mathbf{0} \end{pmatrix}}$$

$$\Rightarrow \tilde{\mathcal{P}}^{-1} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \tilde{\mathbf{L}}^{-\mathrm{T}} \\ \mathbf{0} & \mathbf{R}^{-1} & \mathbf{0} \\ \tilde{\mathbf{L}}^{-1} & \mathbf{0} & -\tilde{\mathbf{L}}^{-1}\mathbf{D}\tilde{\mathbf{L}}^{-\mathrm{T}} \end{pmatrix}$$

- Note that $\tilde{\mathcal{P}}^{-1}$ does not contain $\mathbf{D}^{-1}$.

# Saddle Point Formulation

- With this preconditioner, we can prove some nice results for the case $\tilde{\mathbf{L}} = \mathbf{L}$:

$$\begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{L}^{-\mathrm{T}} \\ \mathbf{0} & \mathbf{R}^{-1} & \mathbf{0} \\ \mathbf{L}^{-1} & \mathbf{0} & -\mathbf{L}^{-1}\mathbf{D}\mathbf{L}^{-\mathrm{T}} \end{pmatrix} \begin{pmatrix} \mathbf{D} & \mathbf{0} & \mathbf{L} \\ \mathbf{0} & \mathbf{R} & \mathbf{H} \\ \mathbf{L}^{\mathrm{T}} & \mathbf{H}^{\mathrm{T}} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \\ \delta\mathbf{x} \end{pmatrix} = \tau \begin{pmatrix} \lambda \\ \mu \\ \delta\mathbf{x} \end{pmatrix}$$

$$\Rightarrow \mathbf{I} + \begin{pmatrix} \mathbf{0} & \mathbf{L}^{-\mathrm{T}}\mathbf{H}^{\mathrm{T}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{R}^{-1}\mathbf{H} \\ \mathbf{0} & -\mathbf{L}^{-1}\mathbf{D}\mathbf{L}^{-\mathrm{T}}\mathbf{H}^{\mathrm{T}} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \\ \delta\mathbf{x} \end{pmatrix} = \tau \begin{pmatrix} \lambda \\ \mu \\ \delta\mathbf{x} \end{pmatrix}$$

$$\Rightarrow \mathbf{H}\mathbf{L}^{-1}\mathbf{D}\mathbf{L}^{-\mathrm{T}}\mathbf{H}^{\mathrm{T}}\mu + (\tau - 1)^2\mathbf{R}\mu = \mathbf{0}$$

$\Rightarrow (\tau - 1)$ is imaginary (or zero) since $\mathbf{H}\mathbf{L}^{-1}\mathbf{D}\mathbf{L}^{-\mathrm{T}}\mathbf{H}^{\mathrm{T}}$ is positive semi-definite and $\mathbf{R}$ is positive definite.

# Saddle Point Formulation

- The eigenvalues $\tau$ of $\tilde{\mathcal{P}}^{-1}\mathcal{A}$ lie on the line $\Re(\tau) = 1$ in the complex plane.
- Their distance above/below the real axis is:

$$\pm\sqrt{\frac{\mu_i^{\mathrm{T}}\mathbf{H}\mathbf{L}^{-1}\mathbf{D}\mathbf{L}^{-\mathrm{T}}\mathbf{H}^{\mathrm{T}}\mu_i}{\mu_i^{\mathrm{T}}\mathbf{R}\mu_i}}$$

  where $\mu_i$ is the $\mu$ component of the $i$th eigenvector.
- The fraction under the square root is, roughly speaking, the ratio of background+model error variance to observation error variance associated with the pattern $\mu_i$.
- In the usual implementation of 4D-Var, the condition number is given by the ratio of these variances, not the square-root.

# Saddle Point Formulation

- For the preconditioner, we need an approximate inverse of $\mathbf{L}$.
- One approach is to use the following identity (exercise for the reader!):

$$\boxed{\mathbf{L}^{-1} = \mathbf{I} + (\mathbf{I} - \mathbf{L}) + (\mathbf{I} - \mathbf{L})^2 + \ldots + (\mathbf{I} - \mathbf{L})^{N-1}}$$

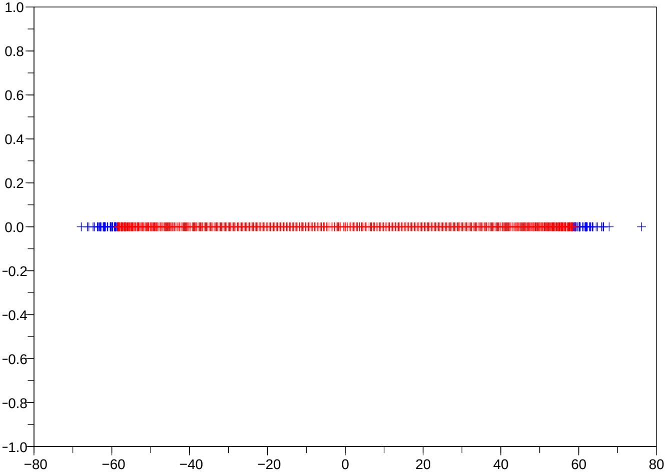
- Since this is a power series expansion, it suggests truncating the series at some order $< N - 1$.
- (A very few iterations of a Krylov solver may be a better idea. I've not tried this yet.)

# Results

- The practical reaults shown in the next few slides are for a simplified (toy) analogue of a real system.
- The model is a two-level quasi-geostrophic channel model with 1600 gridpoints.
- The model has realistic error-growth and time-to-nonlinearity
- There are 100 observations of streamfunction every 3 hours, and 100 wind observations every 6 hours.
- The error covariances are assumed to be horizontally isotropic and homogeneous, with a Gaussian spatial structure.

# Saddle Point Formulation

OOPS QG model. 24-hour window with 8 sub-windows.

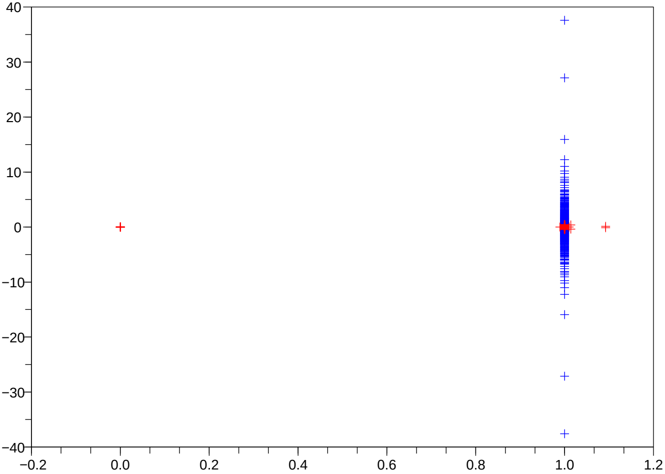



Ritz Values of $\mathcal{A}$.

Converged Ritz values after 500 Arnoldi iterations are shown in blue. Unconverged values in red.

# Saddle Point Formulation

OOPS QG model. 24-hour window with 8 sub-windows.

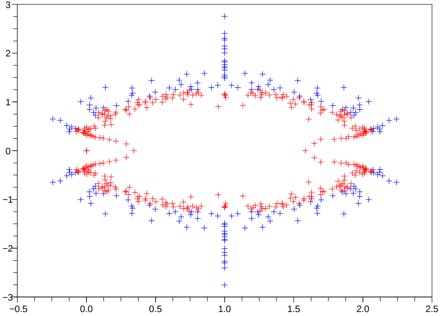



Ritz Values of $\tilde{\mathcal{P}}^{-1}\mathcal{A}$ for $\tilde{\mathbf{L}} = \mathbf{L}$.

Converged Ritz values after 500 Arnoldi iterations are shown in blue. Unconverged values in red.

# Saddle Point Formulation

- It is much harder to prove results for the case $\tilde{\mathbf{L}} \neq \mathbf{L}$.
- Experimentally, it seems that many eigenvalues continue to lie on $\Re(\tau) = 1$, with the remainder forming a cloud around $\tau = 1$.

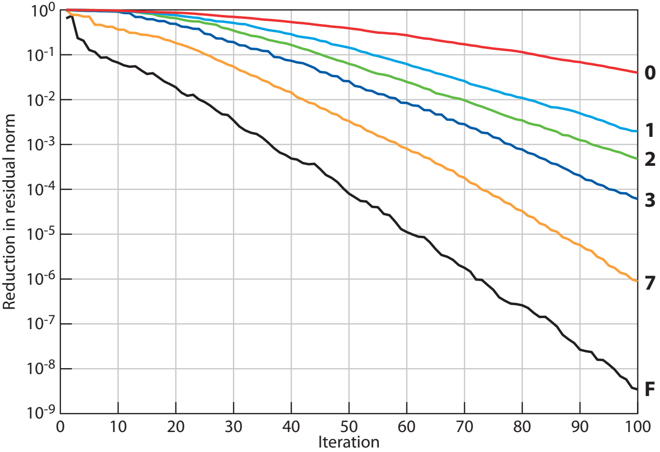



Ritz Values of $\tilde{\mathcal{P}}^{-1}\mathcal{A}$ for $\tilde{\mathbf{L}} = \mathbf{I}$.

Converged Ritz values after 500 Arnoldi iterations are shown in blue. Unconverged values in red.

# Saddle Point Formulation

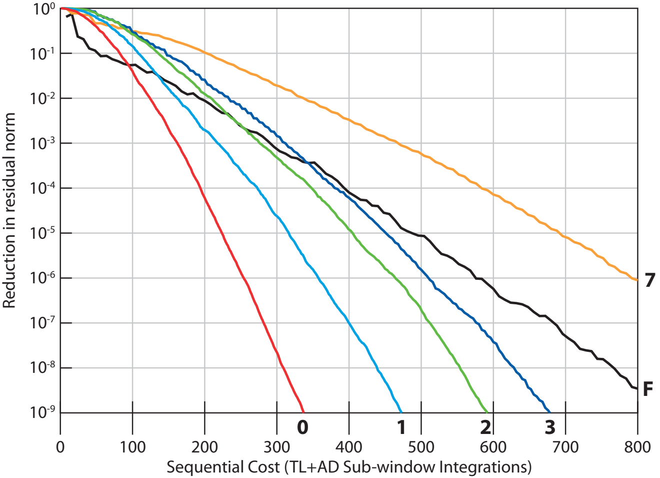
OOPS, QG model, 24-hour window with 8 sub-windows.




Convergence as a function of iteration for different truncations of the series expansion for **L**. ("F" = Forcing formulation.)

# Saddle Point Formulation

OOPS, QG model, 24-hour window with 8 sub-windows.




Convergence as a function of sequential sub-window integrations for different truncations of the series expansion for **L**.

# Conclusions

- 4D-Var was analysed from the point of view of parallelization.
- 4D-PSAS and the forcing formulation are inherently sequential.
- The 4D-state problem is parallel, but ill-conditioning of **D** makes it difficult to precondition.
- The saddle point formulation is parallel, and seems easier to precondition.
  - A saddle point method for strong-constraint 4D-Var was proposed by Thierry Lagarde in his PhD thesis (2000: Univ Paul Sabatier, Toulouse). It didn't catch on:
  - Parallelization was not so important 10 year ago.
  - In strong constraint 4D-Var, we only get a factor-of-two speed-up. This is not enough to overcome the slower convergence due to the fact that the system is indefinite.
- The ability to also parallelize over sub-windows allows a much bigger speed-up in weak-constraint 4D-Var.
- The saddle point formulation is already fast enough to be useful.
- Better solvers and preconditioners can only make faster.