

Approximating the inverse Hessian in 4D-Var data assimilation

Alison Ramage

Department of Mathematics and Statistics



With thanks to...

Kirsty Brown (Strathclyde), Igor Gejadze (IRSTEA, France),
Amos Lawless and Nancy Nichols (Reading)

Happy birthday Des!



Approximating the inverse Hessian in 4D-Var data assimilation

Alison Ramage

Department of Mathematics and Statistics



With thanks to...

Kirsty Brown (Strathclyde), Igor Gejadze (IRSTEA, France),
Amos Lawless and Nancy Nichols (Reading)

Data assimilation

- Numerical weather prediction is an IVP: given initial conditions, forecast atmospheric evolution.
- Data assimilation is a technique for combining information such as observational and background data with numerical models to obtain the best estimate of state of a system (initial condition).
- Other application areas include hydrology, oceanography, environmental science, data analytics, sensor networks...
- Variational assimilation is used to find the optimal analysis that minimises a specific cost function.

Motivation



Four-dimensional Variational Assimilation (4D-Var)

4D-Var aims to find the solution of a numerical forecast model that best fits sequences of observations distributed in space over a finite time interval.

Minimise cost function

$$J(\mathbf{v}_0) = (\mathbf{v}_0 - \mathbf{v}_0^B)^T B^{-1} (\mathbf{v}_0 - \mathbf{v}_0^B) + \sum_{i=0}^n (\mathcal{H}(\mathbf{v}_i) - \mathbf{y}_i)^T R^{-1} (\mathcal{H}(\mathbf{v}_i) - \mathbf{y}_i)$$

with constraint $\mathbf{v}_i = \mathcal{M}^{i,0}(\mathbf{v}_0)$.

analysis	\mathbf{v}_0
background (short-term forecast)	\mathbf{v}_0^B
observations	\mathbf{y}
observation operator	\mathcal{H}
model dynamics	$\mathbf{v}_{i+1} = \mathcal{M}(\mathbf{v}_i)$
background error covariance matrix	B
observation error covariance matrix	R

Incremental 4D-Var

- Linearise \mathcal{H} , \mathcal{M} and solve resulting **unconstrained** optimisation problem iteratively:

$$\bar{H}_{k-1}^i \equiv \left. \frac{\partial \mathcal{H}^i}{\partial \mathbf{v}} \right|_{\mathbf{v}=\mathbf{v}_{k-1}}, \quad \bar{M}_{k-1}^{i,0} \equiv \left. \frac{\partial \mathcal{M}^{i,0}}{\partial \mathbf{v}} \right|_{\mathbf{v}=\mathbf{v}_{k-1}}$$

Incremental 4D-Var

- Linearise \mathcal{H} , \mathcal{M} and solve resulting **unconstrained** optimisation problem iteratively:

$$\bar{H}_{k-1}^i \equiv \left. \frac{\partial \mathcal{H}^i}{\partial \mathbf{v}} \right|_{\mathbf{v}=\mathbf{v}_{k-1}}, \quad \bar{M}_{k-1}^{i,0} \equiv \left. \frac{\partial \mathcal{M}^{i,0}}{\partial \mathbf{v}} \right|_{\mathbf{v}=\mathbf{v}_{k-1}}$$

- **Hessian** of the cost function is

$$\mathbb{H} = B^{-1} + \hat{H}^T \hat{R}^{-1} \hat{H}$$

where

$$\begin{aligned}\hat{H} &= [(\bar{H}^0)^T, (\bar{H}^1 \bar{M}^{1,0})^T, \dots, (\bar{H}^N \bar{M}^{N,0})^T]^T \\ \hat{R} &= \text{bldiag}(R_i), \quad i = 1, \dots, N.\end{aligned}$$

Motivation

Why approximate \mathbb{H}^{-1} ?

Why approximate \mathbb{H}^{-1} ?

- \mathbb{H}^{-1} represents an approximation of the Posterior Covariance Matrix (PCM).
- The PCM can be used to find confidence intervals and carry out *a posteriori* error analysis.
- $\mathbb{H}^{-1/2}$ can be used in ensemble forecasting.
- \mathbb{H}^{-1} , $\mathbb{H}^{-1/2}$ can be used for preconditioning in a Gauss-Newton method.

Issues with approximating the inverse Hessian

- State and observation vectors used in realistic applications can be of length $10^9 - 10^{12}$ and $10^6 - 10^9$, respectively.

Issues with approximating the inverse Hessian

- State and observation vectors used in realistic applications can be of length $10^9 - 10^{12}$ and $10^6 - 10^9$, respectively.
- Cannot store \mathbb{H} as a matrix: only action of applying \mathbb{H} to a vector is available.

Issues with approximating the inverse Hessian

- State and observation vectors used in realistic applications can be of length $10^9 - 10^{12}$ and $10^6 - 10^9$, respectively.
- Cannot store \mathbb{H} as a matrix: only action of applying \mathbb{H} to a vector is available.
- Evaluating $\mathbb{H}\mathbf{v}$ is expensive in terms of computing time and memory (involves both forward and backward model solves with a sequence of tangent linear and adjoint problems).

Issues with approximating the inverse Hessian

- State and observation vectors used in realistic applications can be of length $10^9 - 10^{12}$ and $10^6 - 10^9$, respectively.
- Cannot store \mathbb{H} as a matrix: only action of applying \mathbb{H} to a vector is available.
- Evaluating $\mathbb{H}\mathbf{v}$ is expensive in terms of computing time and memory (involves both forward and backward model solves with a sequence of tangent linear and adjoint problems).
- No obvious equivalent option exists for evaluating $\mathbb{H}^{-1}\mathbf{v}$.

First level preconditioning

$$\mathbb{H} = B^{-1} + \hat{H}^T \hat{R}^{-1} \hat{H}$$

- Precondition \mathbb{H} based on the background covariance matrix

$$H = (B^{1/2})^T \mathbb{H} B^{1/2} = I + (B^{1/2})^T \hat{H}^T \hat{R}^{-1} \hat{H} B^{1/2}$$

First level preconditioning

$$\mathbb{H} = B^{-1} + \hat{H}^T \hat{R}^{-1} \hat{H}$$

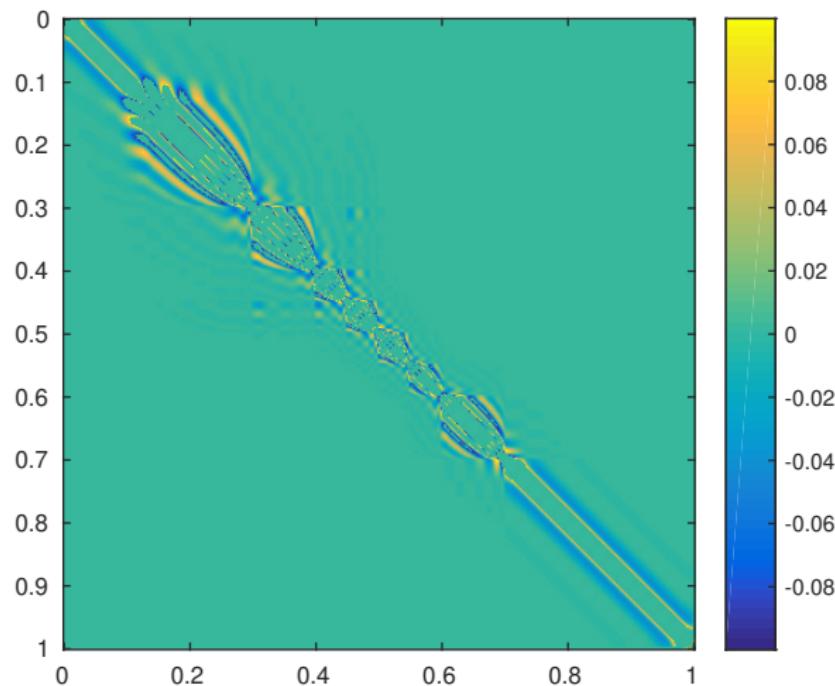
- Precondition \mathbb{H} based on the background covariance matrix

$$H = (B^{1/2})^T \mathbb{H} B^{1/2} = I + (B^{1/2})^T \hat{H}^T \hat{R}^{-1} \hat{H} B^{1/2}$$

- Eigenvalues of H are bounded below by one: more details on the full **eigenspectrum** can be found in HABEN ET AL. (2011), TABEART ET AL. (2018).

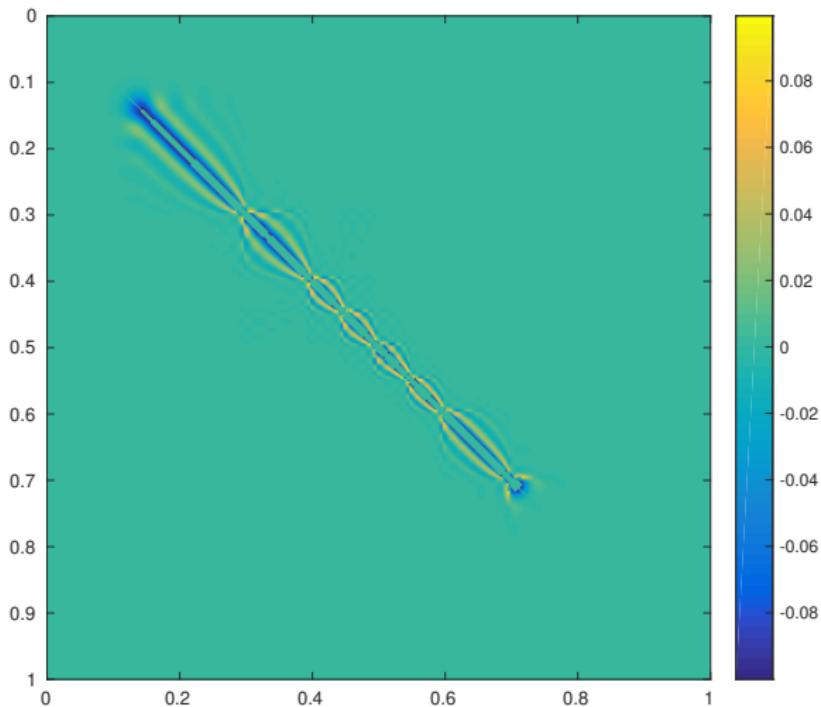
Correlation matrix

- \mathbb{H}^{-1} (scaled to have unit diagonal)



Preconditioned correlation matrix

- H^{-1} (scaled to have unit diagonal)



Approximating the inverse Hessian

- Aim here is to construct an approximation to the action of H^{-1} using **only matrix-vector multiplication with H .**
- H has many eigenvalues close to 1.
- H^{-1} is close to the identity in some sense.
- H and H^{-1} are amenable to **limited-memory approximation**.

Limited-memory approximation

- Find n_e leading eigenvalues and orthonormal eigenvectors using the **Lanczos** method (needs only $H\mathbf{v}$).

Limited-memory approximation

- Find n_e leading eigenvalues and orthonormal eigenvectors using the **Lanczos** method (needs only $H\mathbf{v}$).
- Construct approximation

$$H \approx I + \sum_{i=1}^{n_e} (\lambda_i - 1) \mathbf{u}_i \mathbf{u}_i^T$$

Limited-memory approximation

- Find n_e leading eigenvalues and orthonormal eigenvectors using the **Lanczos** method (needs only $H\mathbf{v}$).
- Construct approximation

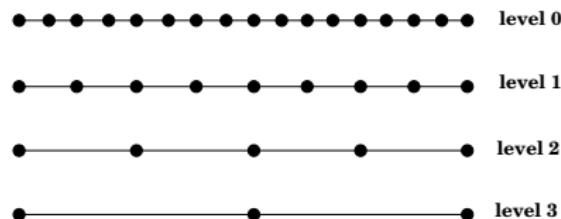
$$H \approx I + \sum_{i=1}^{n_e} (\lambda_i - 1) \mathbf{u}_i \mathbf{u}_i^T$$

- Can also use this to easily approximate matrix powers (including H^{-1} and $H^{-1/2}$):

$$H^p \approx I + \sum_{i=1}^{n_e} (\lambda_i^p - 1) \mathbf{u}_i \mathbf{u}_i^T$$

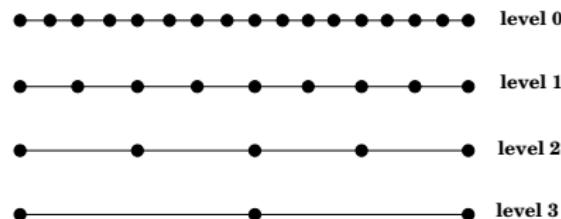
Multilevel limited-memory approximation

- Sequence of grid levels $k = 0, 1, 2, \dots$



Multilevel limited-memory approximation

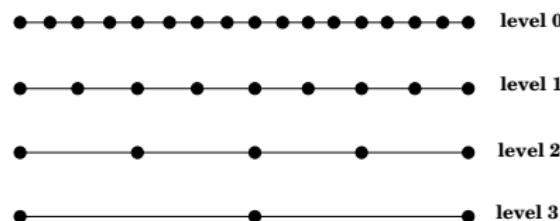
- Sequence of grid levels $k = 0, 1, 2, \dots$



- Matrix H_0 is available on finest grid $k = 0$.

Multilevel limited-memory approximation

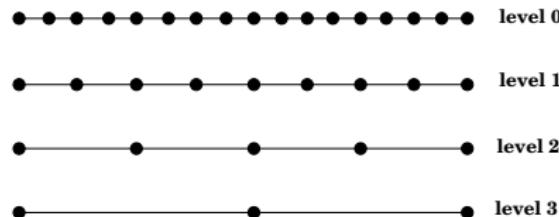
- Sequence of grid levels $k = 0, 1, 2, \dots$



- Matrix H_0 is available on finest grid $k = 0$.
- Construct a **multilevel** approximation to H_0^{-1} based on limited-memory approximations on a sequence of nested grids.

Multilevel limited-memory approximation

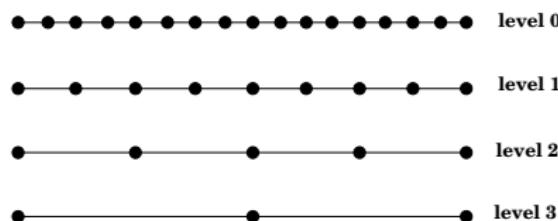
- Sequence of grid levels $k = 0, 1, 2, \dots$



- Matrix H_0 is available on finest grid $k = 0$.
- Construct a **multilevel** approximation to H_0^{-1} based on limited-memory approximations on a sequence of nested grids.
- Need **grid transfer operators** (more shortly).

Multilevel limited-memory approximation

- Sequence of grid levels $k = 0, 1, 2, \dots$



- Matrix H_0 is available on finest grid $k = 0$.
- Construct a **multilevel** approximation to H_0^{-1} based on limited-memory approximations on a sequence of nested grids.
- Need **grid transfer operators** (more shortly).
- Identity matrix I_k on grid level k .
- $[H]_{\rightarrow k}$ means “matrix H transferred to grid level k ”.

Grid transfers for vectors

- Coarse grid level $k = c$; fine grid level $k = f$.

Grid transfers for vectors

- Coarse grid level $k = c$; fine grid level $k = f$.
- Restriction matrix R ; prolongation matrix P : assume “perfect interpolation”, i.e., $RP = I_c$.

Grid transfers for vectors

- Coarse grid level $k = c$; fine grid level $k = f$.
- Restriction matrix R ; prolongation matrix P : assume “perfect interpolation”, i.e., $RP = I_c$.
- Split fine grid vector into two parts:

$$\mathbf{v}_f = \mathbf{v}_f^{(1)} + \mathbf{v}_f^{(2)} = (I_f - PR)\mathbf{v}_f + PR\mathbf{v}_f.$$

Grid transfers for vectors

- Coarse grid level $\mathbf{k} = \mathbf{c}$; fine grid level $\mathbf{k} = \mathbf{f}$.
- Restriction matrix R ; prolongation matrix P : assume “perfect interpolation”, i.e., $RP = I_c$.
- Split fine grid vector into two parts:

$$\mathbf{v}_f = \mathbf{v}_f^{(1)} + \mathbf{v}_f^{(2)} = (I_f - PR)\mathbf{v}_f + PR\mathbf{v}_f.$$

- Restrict \mathbf{v}_f to coarse grid:

$$\mathbf{v}_c^{(1)} = R\mathbf{v}_f^{(1)} = R(I_f - PR)\mathbf{v}_f = (R - (RP)R)\mathbf{v}_f = \mathbf{0}$$

$$\mathbf{v}_c^{(2)} = R\mathbf{v}_f^{(2)} = (RP)R\mathbf{v}_f = R\mathbf{v}_f.$$

- Modes in $\mathbf{v}_f^{(1)}$ are not supported on coarse grid.

Grid transfers for matrices

- Consider action of coarse grid matrix H_c on a fine grid vector:

$$\begin{aligned}[H_c]_{\rightarrow f} \mathbf{v}_f &= \mathbf{v}_f^{(1)} + PH_c R \mathbf{v}_f^{(2)} \\ &= (I_f - PR) \mathbf{v}_f + PH_c (RP) R \mathbf{v}_f \\ &= (P(H_c - I_c)R + I_f) \mathbf{v}_f\end{aligned}$$

Grid transfers for matrices

- Consider action of coarse grid matrix H_c on a fine grid vector:

$$\begin{aligned}[H_c]_{\rightarrow f} \mathbf{v}_f &= \mathbf{v}_f^{(1)} + PH_c R \mathbf{v}_f^{(2)} \\ &= (I_f - PR) \mathbf{v}_f + PH_c (RP) R \mathbf{v}_f \\ &= (P(H_c - I_c)R + I_f) \mathbf{v}_f\end{aligned}$$

- This motivates matrix transfer operators

- From coarse grid to fine grid

$$[H_c]_{\rightarrow f} = P(H_c - I_c)R + I_f$$

- From fine grid to coarse grid

$$[H_f]_{\rightarrow c} = R(H_f - I_f)P + I_c$$

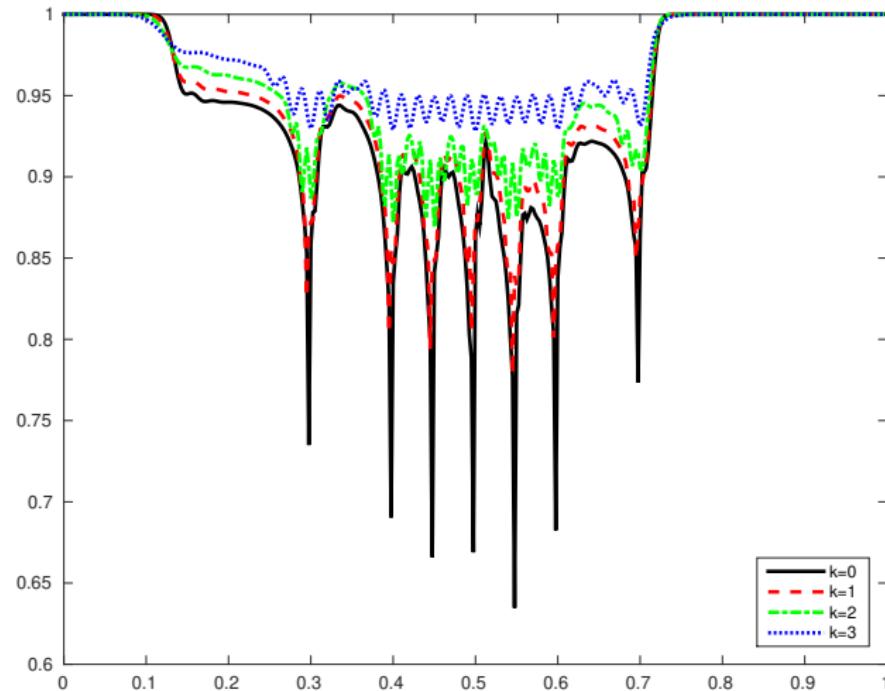
Test problem 1

- Model is 1D Burgers' equation.
- Discretise evolution equation on a grid with $m + 1$ nodes (level 0) to represent full Hessian H_0 .
- Grid level k contains $m_k = m/2^k + 1$ nodes.
- 1D uniform grid with 7 sensors located at 0.3, 0.4, 0.45, 0.5, 0.55, 0.6, and 0.7 in $[0, 1]$.
- Construct a multilevel approximation to H^{-1} with four grid levels:

k	0	1	2	3
grid points	401	201	101	51

Hessian in a multilevel framework

- Diagonal of H^{-1} on various grid levels:



Eigenvalues of Hessian at each level

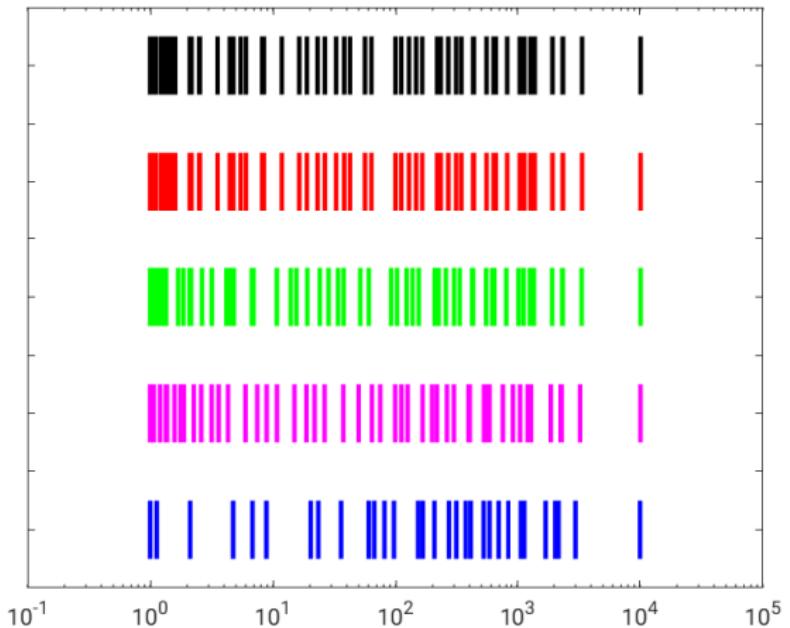
H_0

$H_0 = [H_0]_{\rightarrow 0}$

$H_1 = [H_0]_{\rightarrow 1}$

$H_2 = [H_0]_{\rightarrow 2}$

$H_3 = [H_0]_{\rightarrow 3}$



Motivating idea

- Eigenvalues of $[H_c^{-1/2}]_{\rightarrow f} H_f [H_c^{-1/2}]_{\rightarrow f}$ should be clustered around 1.

Motivating idea

- Eigenvalues of $[H_c^{-1/2}]_{\rightarrow f} H_f [H_c^{-1/2}]_{\rightarrow f}$ should be clustered around 1.
- Construct an approximation to $H_c^{-1/2}$:
 - Precondition H_c to obtain $\tilde{H}_c = M^T H_c M$ with eigenvalues closer to 1.
 - Build \hat{H}_c , a **limited memory approximation** for \tilde{H}_c using n_c eigenvalues with the **Lanczos** method.
 - Note that

$$H_c^{-1} = M \tilde{H}_c^{-1} M^T \simeq M \hat{H}_c^{-1} M^T$$

so

$$H_c^{-1/2} = M \tilde{H}_c^{-1/2} \simeq M \hat{H}_c^{-1/2}.$$

Motivating idea

- Eigenvalues of $[H_c^{-1/2}]_{\rightarrow f} H_f [H_c^{-1/2}]_{\rightarrow f}$ should be clustered around 1.
- Construct an approximation to $H_c^{-1/2}$:
 - Precondition H_c to obtain $\tilde{H}_c = M^T H_c M$ with eigenvalues closer to 1.
 - Build \hat{H}_c , a **limited memory approximation** for \tilde{H}_c using n_c eigenvalues with the **Lanczos** method.
 - Note that

$$H_c^{-1} = M \tilde{H}_c^{-1} M^T \simeq M \hat{H}_c^{-1} M^T$$

so

$$H_c^{-1/2} = M \tilde{H}_c^{-1/2} \simeq M \hat{H}_c^{-1/2}.$$

- Use $\hat{M} = [M \hat{H}_c^{-1/2}]_{\rightarrow f}$ as a preconditioner on the level above.

Outline of multilevel concept

Step 1. Start on coarsest grid level.

Step 2. Represent H_0 on grid level k as $H_k = [H_0]_{\rightarrow k}$.

Step 3. Precondition this to obtain $\tilde{H}_k = M_k^T H_k M_k$.

Step 4. Build limited memory approximation $\hat{H}_k^{-1/2}$.

Step 5. Project $\hat{M}_k = M_k \hat{H}_k^{-1/2}$ to the level above to be used as preconditioner at the next coarsest level.

Step 6. Move up one grid level and repeat from step 2.

Preconditioners

- On coarsest grid, level $k + 1$ does not exist so set $M_k = I_k$.

Preconditioners

- On coarsest grid, level $k + 1$ does not exist so set $M_k = I_k$.
- For other levels, M_k is constructed on level $k + 1$ and applied on level k .

Preconditioners

- On coarsest grid, level $k + 1$ does not exist so set $M_k = I_k$.
- For other levels, M_k is constructed on level $k + 1$ and applied on level k .
- Preconditioners are constructed **recursively**:

$$M_k = [\hat{M}_{k+1}]_{\rightarrow k} = \left[M_{k+1} \hat{H}_{k+1}^{-1/2} \right]_{\rightarrow k}.$$

Preconditioners

- On coarsest grid, level $k + 1$ does not exist so set $M_k = I_k$.
- For other levels, M_k is constructed on level $k + 1$ and applied on level k .
- Preconditioners are constructed **recursively**:

$$M_k = [\hat{M}_{k+1}]_{\rightarrow k} = \left[M_{k+1} \hat{H}_{k+1}^{-1/2} \right]_{\rightarrow k}.$$

- At level 0, final inverse Hessian approximation H_{approx}^{-1} will contain eigenvalue information from **all levels**.

Algorithm in practice

- use $N_e = (n_0, n_1, \dots, n_{k_c})$ eigenvalues at each level

```
[Λ, U] = MLalg(H₀, Ne)
for k = kc, kc - 1, ..., 0
    compute by the Lanczos method
        {λki, Uki}, i = 1, ..., nk of H₀→k
    using preconditioner Mk
end
```

Algorithm in practice

- use $N_e = (n_0, n_1, \dots, n_{k_c})$ eigenvalues at each level

```
[Λ, U] = MLalg(H₀, Ne)
for k = kc, kc - 1, ..., 0
    compute by the Lanczos method
    {λki, Uki}, i = 1, ..., nk of H₀→k
    using preconditioner Mk
end
```

- storage:

$$\begin{aligned} \Lambda &= [\lambda_0^1, \dots, \lambda_0^{n_0}, \lambda_1^1, \dots, \lambda_1^{n_1}, \dots, \lambda_{k_c}^1, \dots, \lambda_{k_c}^{n_{k_c}}], \\ U &= [U_0^1, \dots, U_0^{n_0}, U_1^1, \dots, U_1^{n_1}, \dots, U_{k_c}^1, \dots, U_{k_c}^{n_{k_c}}]. \end{aligned}$$

Assessing approximation accuracy

- Riemannian distance:

$$\delta(A, B) = \| \ln(B^{-1}A) \|_F = \left(\sum_{i=1}^n \ln^2 \lambda_i \right)^{1/2}$$

Assessing approximation accuracy

- Riemannian distance:

$$\delta(A, B) = \| \ln(B^{-1}A) \|_F = \left(\sum_{i=1}^n \ln^2 \lambda_i \right)^{1/2}$$

- Compare eigenvalues of H^{-1} and H_{approx}^{-1} on the finest grid level $k = 0$ using distance function

$$D = \frac{\delta(H^{-1}, H_{approx}^{-1})}{\delta(H^{-1}, I)}$$

Assessing approximation accuracy

- Riemannian distance:

$$\delta(A, B) = \| \ln(B^{-1}A) \|_F = \left(\sum_{i=1}^n \ln^2 \lambda_i \right)^{1/2}$$

- Compare eigenvalues of H^{-1} and H_{approx}^{-1} on the finest grid level $k = 0$ using distance function

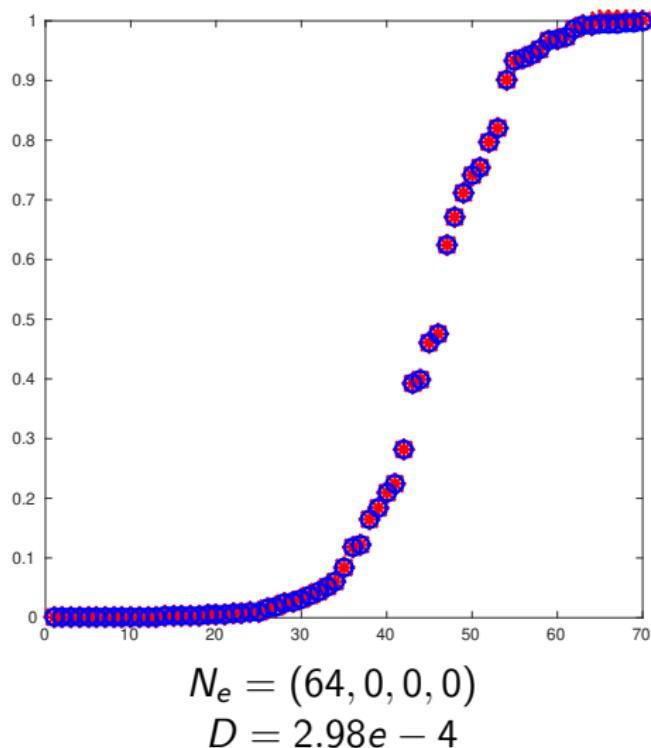
$$D = \frac{\delta(H^{-1}, H_{approx}^{-1})}{\delta(H^{-1}, I)}$$

- Vary number of eigenvalues chosen on each grid level

$$N_e = (n_0, n_1, \dots, n_{k_c})$$

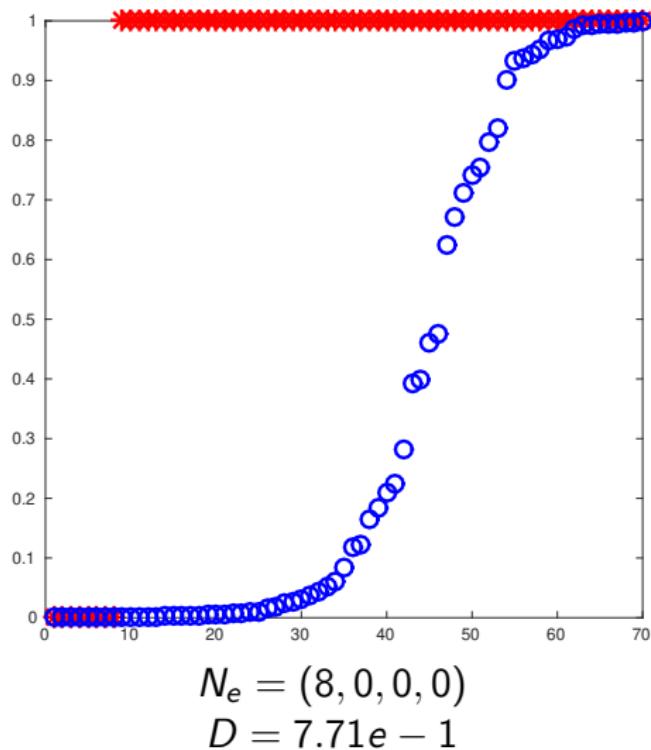
Eigenvalues of the inverse Hessian

- Exact (blue circles), approximated (red stars)



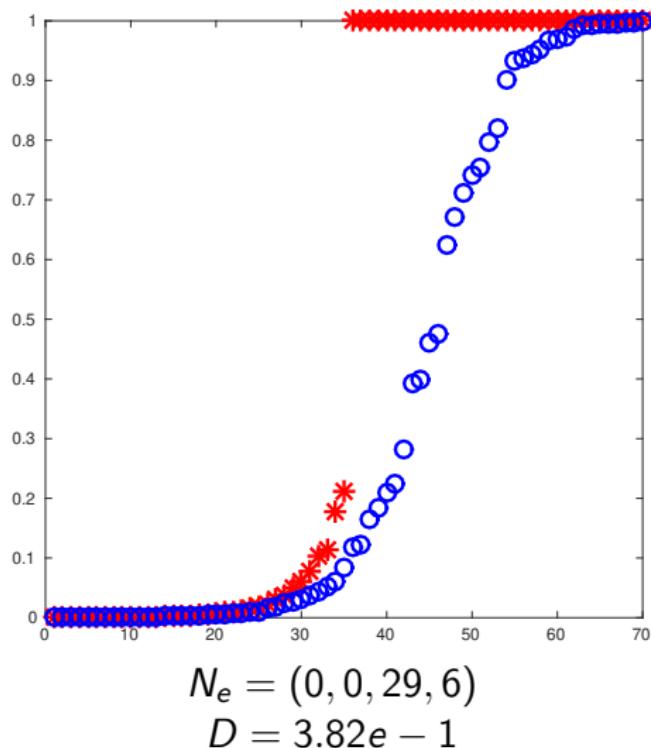
Eigenvalues of the inverse Hessian

- Exact (blue circles), approximated (red stars)



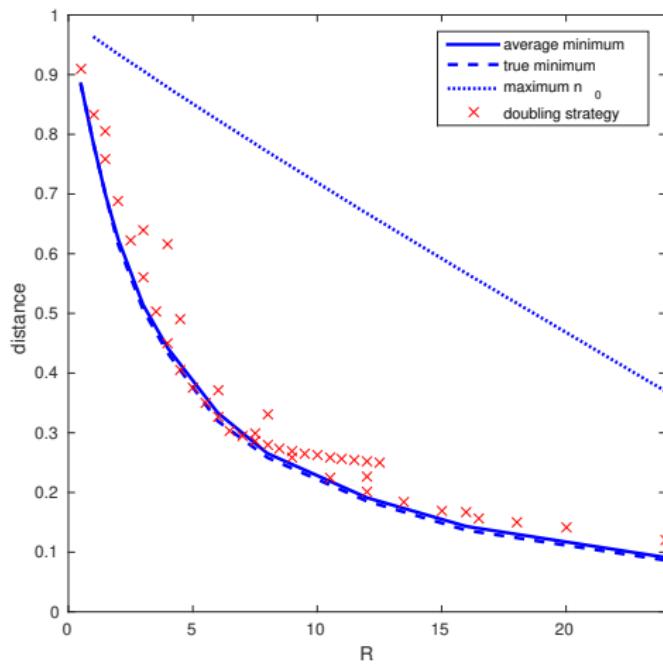
Eigenvalues of the inverse Hessian

- Exact (blue circles), approximated (red stars)



Fixed memory ratio

- Fixed memory ratio $R = \sum_{k=0}^{k_c} \frac{n_k}{2^k}$



Example: using H_{approx}^{-1} as a preconditioner

- Hessian linear system (within a Gauss-Newton method):

$$H(\mathbf{u}_k)\delta\mathbf{u}_k = \mathbf{G}(\mathbf{u}_k).$$

- Solve using Preconditioned Conjugate Gradient iteration (needs only $H\mathbf{v}$).

Example: using H_{approx}^{-1} as a preconditioner

- Hessian linear system (within a Gauss-Newton method):

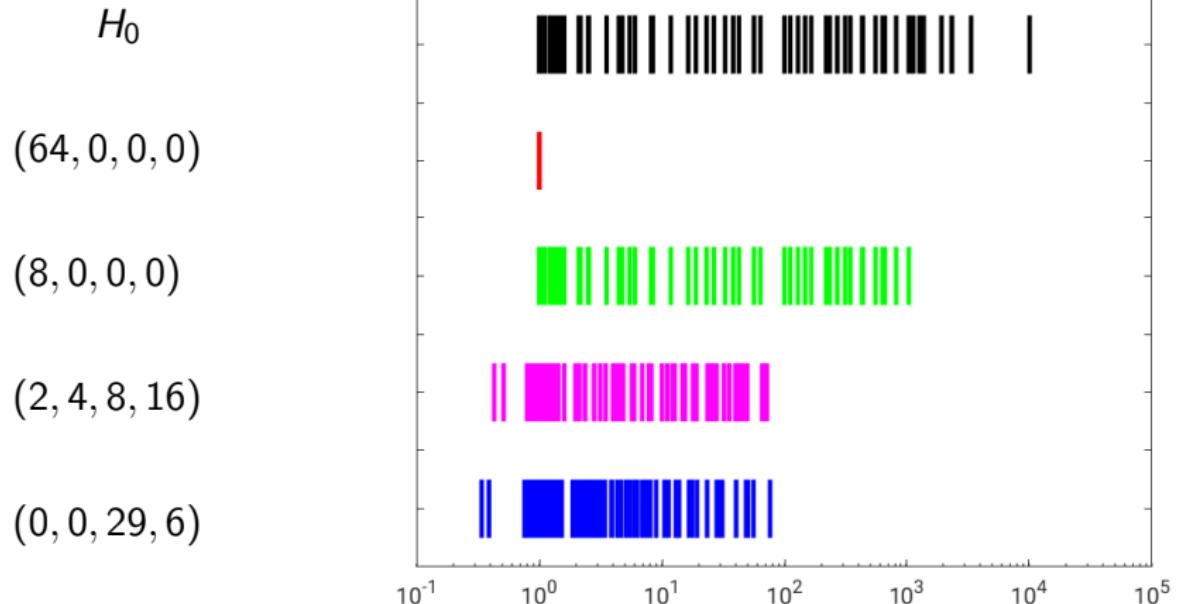
$$H(\mathbf{u}_k)\delta\mathbf{u}_k = G(\mathbf{u}_k).$$

- Solve using Preconditioned Conjugate Gradient iteration (needs only $H\mathbf{v}$).
- Preconditioning with an (invertible) preconditioner M^{-1} : solve

$$(M^{-1/2}HM^{-1/2})M^{1/2}\delta\mathbf{u}_k = M^{-1/2}G.$$

- PCG can converge very quickly if $M^{-1/2}HM^{-1/2}$ has well-clustered eigenvalues.
- Choose M^{-1} so that
 - (i) the eigenvalues of $M^{-1/2}HM^{-1/2}$ are well clustered;
 - (ii) a system of the form $M\mathbf{z} = \mathbf{r}$ is cheap to solve.

Eigenvalues of preconditioned Hessian



Problem 1: PCG iteration for one Newton step

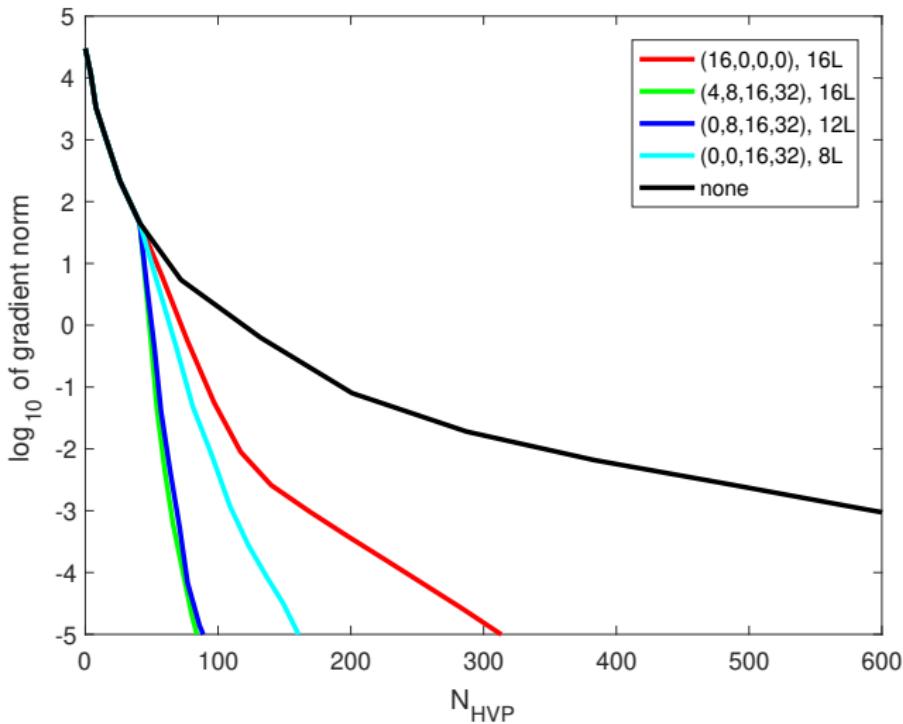
- Test problem 1 (Burgers' equation) with 4 grid levels.
- measurement units
 - storage: length of vector on finest grid L
 - solve cost: cost of HVP on finest grid HVP

Problem 1: PCG iteration for one Newton step

- Test problem 1 (Burgers' equation) with 4 grid levels.
- measurement units
 - storage: length of vector on finest grid L
 - solve cost: cost of HVP on finest grid HVP

Preconditioner	# CG iterations	storage	solve cost
none	57	0 L	57 HVP
ML(400,0,0,0)	1	400 L	402 HVP
ML(4,8,16,32)	4	16 L	34 HVP
ML(0,8,16,32)	5	12 L	14 HVP
ML(0,0,16,32)	8	8 L	10 HVP

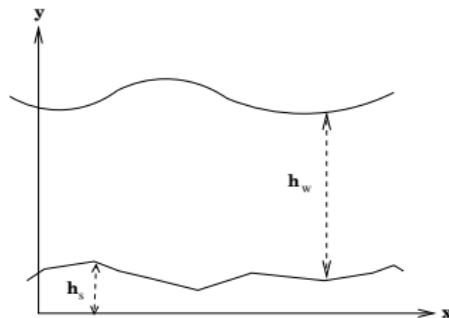
Problem 1: Solve cost measured in number of HVPs



Test problem 2

- Model is 1D shallow water equations for velocity u and geopotential $\phi = gh$.

$$\frac{Du}{Dt} + \frac{\partial \phi}{\partial x} = -g \frac{\partial H}{\partial x}$$
$$\frac{D(\ln \phi)}{Dt} + \frac{\partial u}{\partial x} = 0$$



- Uniformly spaced sensors.
- Four grid multilevel structure as before.
- Vary m_0 (number of nodes on top level grid).

PCG iteration for one Newton step

- Background covariance matrix B constructed using a Laplacian correlation function.

Preconditioner	# PCG iterations			
	$m_0 = 400$	$m_0 = 800$	$m_0 = 1600$	$m_0 = 3200$
none	308	1302	5,879	25,085
ML(4,0,0,0)	38	34	34	47
ML(1,2,4,8)	31	29	28	37
ML(0,2,4,16)	27	26	24	32
ML(0,0,8,16)	26	25	24	30
ML(0,0,0,32)	23	19	19	24

PCG iteration for one Newton step

- Background covariance matrix B constructed using a **Second-Order Auto-Regressive (SOAR)** correlation function.

Preconditioner	# PCG iterations			
	$m_0 = 400$	$m_0 = 800$	$m_0 = 1600$	$m_0 = 3200$
none	509	2,277	10,453	43,915
ML(4,0,0,0)	39	35	35	44
ML(1,2,4,8)	28	26	26	34
ML(0,2,4,16)	23	22	21	27
ML(0,0,8,16)	22	21	20	26
ML(0,0,0,32)	19	16	15	20

Practical implementation: Hessian decomposition

- Partition domain into S subregions and compute local Hessians H^s such that

$$H(\mathbf{v}) = I + \sum_{s=1}^S (H^s(\mathbf{v}) - I).$$

Practical implementation: Hessian decomposition

- Partition domain into S subregions and compute local Hessians H^s such that

$$H(\mathbf{v}) = I + \sum_{s=1}^S (H^s(\mathbf{v}) - I).$$

- Computational advantages of local Hessians:
 - fewer eigenvalues required for limited-memory approximation;
 - can be calculated at a coarser grid level;
 - can use local rather than global models;
 - can be computed in parallel.

Practical approach: Version 1

- ① Compute limited-memory approximations to local sensor-based Hessians on level k using n_k eigenpairs:

$$H_k^s \approx I + \sum_{i=1}^{n_k} (\lambda_i - 1) \mathbf{u}_i \mathbf{u}_i^T$$

- ② Assemble these to form H_a .
- ③ Apply MLalg to H_a based on a fixed N_e .

Practical approach: Version 1

- ① Compute limited-memory approximations to local sensor-based Hessians on level k using n_k eigenpairs:

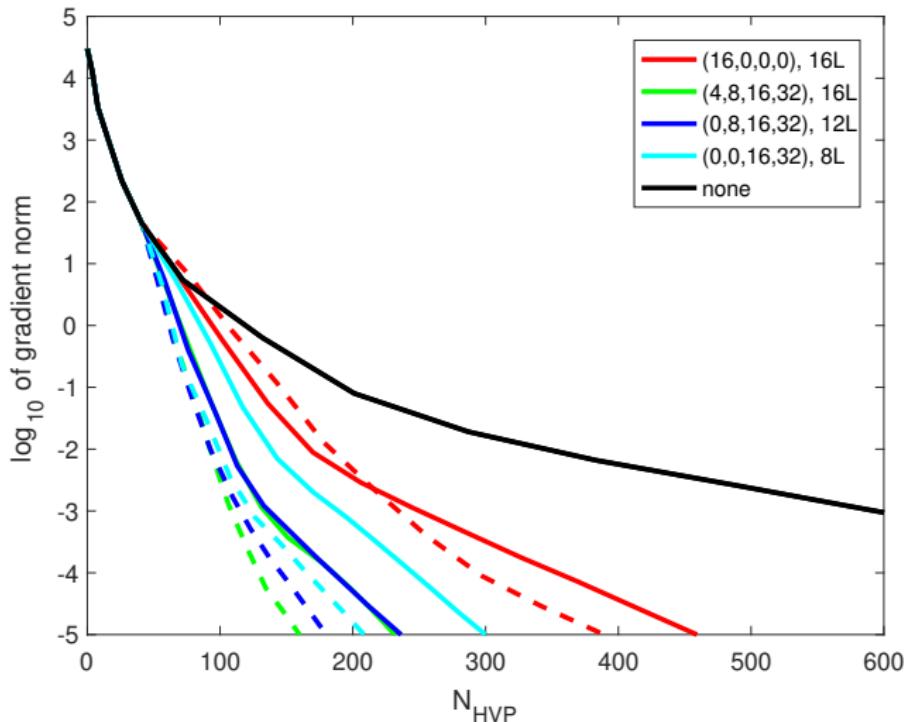
$$H_k^s \approx I + \sum_{i=1}^{n_k} (\lambda_i - 1) \mathbf{u}_i \mathbf{u}_i^T$$

- ② Assemble these to form H_a .
- ③ Apply MLalg to H_a based on a fixed N_e .

- Advantage:
 - local Hessians cheaper to compute.
- Disadvantages:
 - additional user-specified parameter(s) n_k needed;
 - more memory required as local Hessians must also be stored.

Sample costs including building preconditioner

- Local Hessians with 8 eigenvalues at level 0 (solid lines) or level 1 (dashed lines).



Practical approach: Version 2

- ① Approximate each local Hessian H_k^s by applying **MLalg** to local inverse Hessians based on $N_{e,k}$.
- ② Assemble these to form reduced-memory Hessian H_a^{rm} .
- ③ Use **MLalg** again on H_a^{rm} based on N_e .

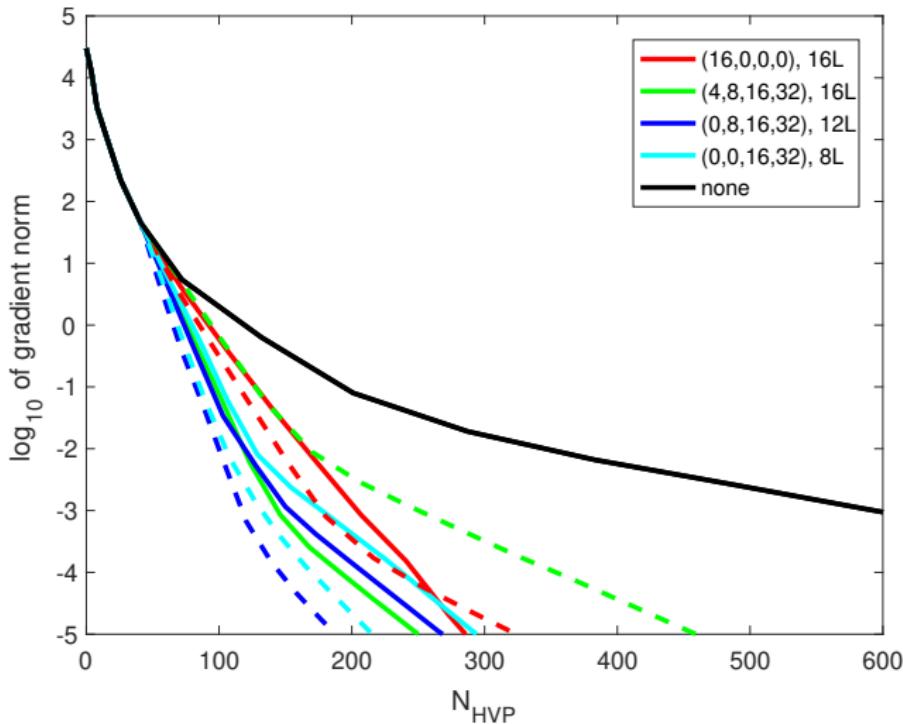
Practical approach: Version 2

- ① Approximate each local Hessian H_k^s by applying **MLalg** to local inverse Hessians based on $N_{e,k}$.
- ② Assemble these to form reduced-memory Hessian H_a^{rm} .
- ③ Use **MLalg** again on H_a^{rm} based on N_e .

- Advantage:
 - requires **less memory** than Version 1.
- Disadvantage:
 - **additional user-specified parameter(s)** $N_{e,k}$ needed.

Version 2: cost including building preconditioner

- Local Hessians with 8 eigenvalues at level 0 (solid lines) or level 1 (dashed lines) with $N_{e,k} = (8, 4, 0, 0)$ ML approx.



Concluding remarks

- Algorithm based solely on repeated use of **Lanczos** at each level (for building limited-memory approximations).
- Difficult to identify the **correct number of eigenvalues** to use at each level: good rule of thumb available but analysis would be better!
- Full algorithm may not always be practical, but we have developed practical implementations based on **Hessian decompositions**.
- Also works well for other configurations (e.g. moving sensors, different initial conditions).
- Potential for extension to higher dimensions and other applications.

Brown, Gejadze & Ramage SISC 38(5), 2016
A Multilevel Approach for Computing the Limited-Memory Hessian and its Inverse in Variational Data Assimilation,

It is sometimes nice in Scotland . . .

