A multilevel preconditioner for data assimilation with 4D-Var

Alison Ramage*, Kirsty Brown*, Igor Gejadze†

∗ Department of Mathematics and Statistics, University of Strathclyde

† IRSTEA, Montpelier, France

Data assimilation

- Numerical weather prediction is an IVP: given initial conditions, forecast atmospheric evolution. This needs an accurate initial state.
- Data assimilation uses observations and a short range forecast (the background) to update the model state, and determine a "best possible" atmospheric state.
- The update step, which is chosen to minimise a specific cost function, is known as the analysis.
- Other application areas include hydrology, oceanography, environmental science, data analytics, sensor networks. . .

Physical model

- Evolution equation:

$$\begin{aligned} \frac{\partial \varphi(t)}{\partial t} &= F(\varphi(t)) + f(t), \\ \varphi(0) &= u, \end{aligned}$$

$$u \in X, \quad t \in (0,T), \quad f, \varphi \in Y = L_2(0,T;X)$$

true initial state	$\bar{u}$
true state evolution	$\bar{\varphi}$
observation operator	$C_{obs} : Y \to Y_{obs}$
observation error	ξ_o
observations	$\varphi_{obs} = C_{obs}\bar{\varphi} + \xi_o$
background error	ξ_b
background function	$u_b = \bar{u} + \xi_b$

Variational data assimilation problem

- represent model in operator form via control-to-state mapping

$$\varphi = R^{cts}(u)$$

- assume errors ξ_o, ξ_b are normal, unbiased and mutually uncorrelated with positive definite covariance operators

$$V_b(\cdot) = E[\langle \cdot, \xi_b \rangle_X \, \xi_b], \qquad V_o(\cdot) = E[\langle \cdot, \xi_o \rangle_{Y_{obs}} \, \xi_o]$$

- DA problem: find $v \in X$ which minimises

$$J(v) = \frac{1}{2}\langle V_b^{-1} v, v \rangle_X + \frac{1}{2}\langle V_o^{-1} C_{obs} R^{cts}(u) v, C_{obs} R^{cts}(u) v \rangle_{Y_{obs}}$$

Hessian operator

- define associated tangent linear operator

$$R'(u)w = \lim_{\tau \to 0} \frac{R^{cts}(u + \tau w) - R^{cts}(u)}{\tau}, \quad \forall w \in X$$

and adjoint

$$\langle w, R'^{*}(u)w^{*} \rangle_X = \langle R'(u)w, w^{*} \rangle_Y, \quad \forall w \in X\ , \forall w^{*} \in Y$$

- Hessian of DA problem:

$$\mathcal{H}(u) = V_b^{-1} + R'^{*}(u) C_{obs}^{*} V_o^{-1} C_{obs} R'(u)$$

Incremental 4D-Var

- Represent functions using a finite-dimensional basis.
- Rewrite as an unconstrained minimisation problem using Lagrange's method.
- Incremental approach: linearise evolution operator and solve linearised problem iteratively.
- Require a discrete version of the tangent linear model (TLM) and its adjoint.
- Each iteration requires one forward solution of the TLM equations and one backward solution of the adjoint equations.

Hessian matrix

- Hessian of the cost function:

$$\mathcal{H} = V_b^{-1} + R^T C_{obs}^T V_o^{-1} C_{obs} R$$

- Discrete tangent linear operator R and its adjoint.
- $\mathcal{H}$ is often too large to be stored in memory.
- Action of applying $\mathcal{H}$ to a vector is available, but expensive:
    - involves both forward and backward solves with the linearised evolution operator and its adjoint.

Approximating the inverse Hessian

Why approximate $\mathcal{H}^{-1}$?

- $\mathcal{H}^{-1}$ 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.
- $\mathcal{H}^{-1/2}$ can be used in ensemble forecasting.
- $\mathcal{H}^{-1}$, $\mathcal{H}^{-1/2}$ can be used for preconditioning in a Gauss-Newton method (focus of this talk).

AIM: construct a limited-memory approximation to $\mathcal{H}^{-1}$ using only matrix-vector multiplication.

Return to 4D-Var

- Linear system (within a Gauss-Newton method):

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

Hessian of the cost function	$\mathcal{H}(\mathbf{u}_k)$
gradient of the cost function	$G(\mathbf{u}_k)$

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

- Convergence depends on eigenvalues of the Hessian

$$\mathcal{H} = V_b^{-1} + R^T C_{obs}^T V_o^{-1} C_{obs} R.$$

- Evaluating $\mathcal{H}\mathbf{v}$ is very expensive, so we need a good preconditoner.

First level preconditioning

- Use the background covariance matrix V_b.
- Projected Hessian:

$$H = (V_b^{1/2})^T \mathcal{H} V_b^{1/2} = I + (V_b^{1/2})^T R^T C_{obs}^T V_o^{-1} C_{obs} R V_b^{1/2}$$

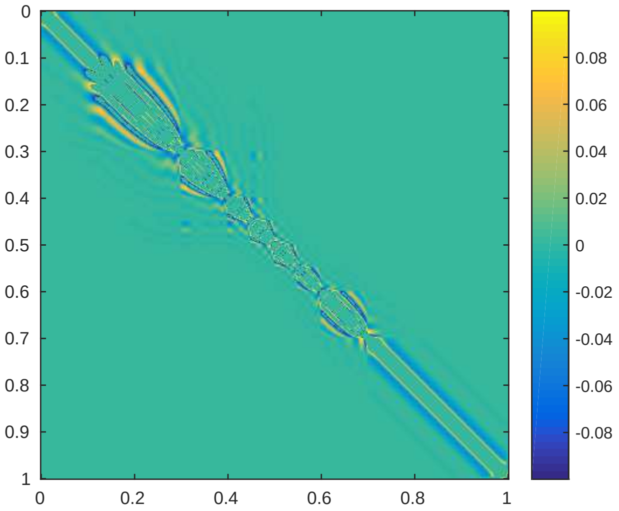
- Easy to recover $\mathcal{H}$ in the original space.
- Eigenvalues of H are usually clustered in a narrow band above one, with few eigenvalues distinct enough to contribute noticeably to the Hessian.

  [Haben et al., Computers & Fluids 46 (2011)]

- This makes H amenable to limited-memory approximation.

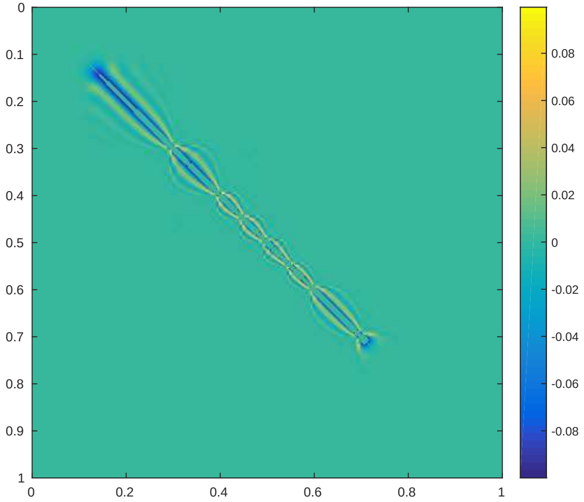
Correlation matrix

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



Preconditioned correlation matrix

- H^{-1} (after first level preconditioning)



Limited-memory approximation

- Find n_e leading eigenvalues and orthonormal eigenvectors using the Lanczos method.

- Construct approximation

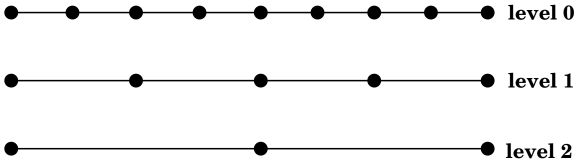
$$H \approx I + \sum_{i=1}^{n_e} (\lambda_i - 1)\mathbf{u}_i\mathbf{u}_i^T$$

- Easy to evaluate matrix powers:

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

Second level preconditioning

- IDEA: Construct a multilevel approximation to H^{-1} based on a sequence of nested grids.
- Discretise evolution equation on a grid with $m + 1$ nodes (level 0) to represent Hessian H_0.
- Grid level k contains $m_k = m/2^k + 1$ nodes.



- Identity matrix I_k on grid level k.

Grid transfers with "correction"

- Grid transfer based on piecewise cubic splines:
  - Restriction matrix R_c^f from $k = f$ to $k = c$.
  - Prolongation matrix P_f^c from $k = c$ to $k = f$.
- Construct new operators which transfer a matrix between a coarse grid level c and a fine grid level f.
  - From coarse to fine:

$$M_{c \to f} = P_f^c (M_c - I_c) R_c^f + I_f$$

  - From fine to coarse:

$$M_{f \to c} = R_c^f (M_f - I_f) P_f^c + I_c$$

Outline of multilevel concept

Given a symmetric positive definite operator H_0 available on the finest grid level in matrix-vector product form:

1. restrict H_0 to the coarsest grid level (denoted by $H_{0\to k}$);
2. use a local preconditioner to improve the eigenvalue distribution;
3. build a limited-memory approximation to its inverse square root using the Lanczos method (which will form the basis of the local preconditioner at the next level up);
4. move up one grid level and repeat.

Multilevel algorithm to approximate H^{-1}

- Represent H_0 at a given level (k, say):

$$H_{0\to k} = R_k^0(H_0 - I_0)P_0^k + I_k$$

- Use local preconditioner to improve eigenvalue spectrum:

$$\tilde{H}_{0\to k} = (B_k^{k+1})^T H_{0\to k} B_k^{k+1}$$

- Find n_k eigenvalues/eigenvectors of $\tilde{H}_{0\to k}$ using the Lanczos method.

- Approximate $\tilde{H}_{0\to k}^{-1/2}$:

$$\tilde{H}_{0\to k}^{-1/2} \approx I_k + \sum_{i=1}^{n_k} \left(\frac{1}{\sqrt{\lambda_i}} - 1 \right) \mathbf{u}_i \mathbf{u}_i^T$$

Local preconditioners

- B_k^{k+1}, constructed on level $k+1$ and applied on level k.
- On coarsest grid, level $k+1$ does not exist so set $B_k^{k+1} = I_k$.
- For other levels, construct preconditioners recursively:

$$B_k^{k+1} = \left[B_{k+1}^{k+2}\tilde{H}_{0\to k+1}^{-1/2}\right]_{\to k}, \quad {B_k^{k+1}}^T = \left[\tilde{H}_{0\to k+1}^{-1/2}{B_{k+1}^{k+2}}^T\right]_{\to k}$$

- Example: 4 levels $(k = 0, 1, 2, 3)$

$$\begin{aligned}
&B_4^5 = I_4,\\
&B_3^4 = [\tilde{H}_{0\to 4}^{-1/2}]_{\to 3},\\
&B_2^3 = [[\tilde{H}_{0\to 4}^{-1/2}]_{\to 3}\tilde{H}_{0\to 3}^{-1/2}]_{\to 2},\\
&B_1^2 = [[[\tilde{H}_{0\to 4}^{-1/2}]_{\to 3}\tilde{H}_{0\to 3}^{-1/2}]_{\to 2}\tilde{H}_{0\to 2}^{-1/2}]_{\to 1},\\
&B_0^1 = [[[[\tilde{H}_{0\to 4}^{-1/2}]_{\to 3}\tilde{H}_{0\to 3}^{-1/2}]_{\to 2}\tilde{H}_{0\to 2}^{-1/2}]_{\to 1}\tilde{H}_{0\to 1}^{-1/2}]_{\to 0}.
\end{aligned}$$

Finest level

- We already have H_0, so precondition to obtain

$$\tilde{H}_0 = {B_0^1}^T H_0 B_0^1.$$

- Find n_0 eigenvalues/eigenvectors of $\tilde{H}_0$ using the Lanczos method.

- Approximate $\tilde{H}_0^{-1}$:

$$\tilde{H}_0^{-1} \approx I_k + \sum_{i=1}^{n_0} \left(\frac{1}{\lambda_i} - 1 \right) \mathbf{u}_i \mathbf{u}_i^T$$

- Recover projected inverse Hessian using

$$H_0^{-1} = B_0^1 \tilde{H}_0^{-1} {B_0^1}^T$$

Notes

- The main cost comes from using the Lanczos method, which involves the Hessian vector product $H_0\mathbf{v}$.
- All matrices involved are in limited-memory form, and are never actually constructed: instead they are applied using multiplication with eigenvalues and eigenvectors only.
- The remaining problem is how to choose the number of eigenvalues used in the limited-memory approximation on each level

$$N_e = (n_0, n_1, \ldots, n_c)$$

to balance accuracy and cost.

Summary of algorithm

- use $N_e = (n_0, n_1, \dots, n_c)$ eigenvalues at each level

```
$[\Lambda, \mathcal{U}] = mlevd(H_0, N_e)$
for   $k = k_c, k_c - 1, \dots, 0$
    compute by the Lanczos method
    and store in memory
        $\{\lambda_k^i, U_k^i\},\ i = 1, \dots, n_k$ of $\tilde{H}_{0\to k}$
    using preconditioner $B_k^{k+1}$
end
```

- storage:

$$\begin{aligned}\Lambda &= \left[\lambda_{k_c}^1, \dots, \lambda_{k_c}^{n_{k_c}}, \lambda_{k_c-1}^1, \dots, \lambda_{k_c-1}^{n_{k_c-1}}, \dots, \lambda_0^1, \dots, \lambda_0^{n_0}\right], \\ \mathcal{U} &= \left[U_{k_c}^1, \dots, U_{k_c}^{n_{k_c}}, U_{k_c-1}^1, \dots, U_{k_c-1}^{n_{k_c-1}}, \dots, U_0^1, \dots, U_0^{n_0}\right].\end{aligned}$$

Example

- Test using 1D Burgers' equation with initial condition

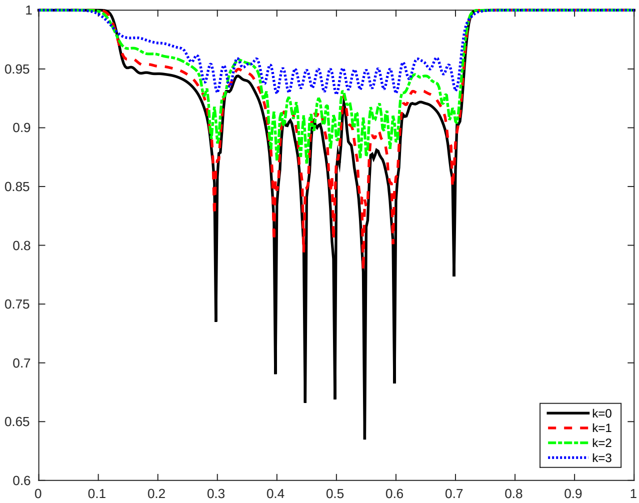
$$f(x) = 0.1 + 0.35\left[1 + \sin\left(4\pi x + \frac{3\pi}{2}\right)\right], \qquad 0 < x < 1$$

- 1D uniform grid with 7 sensors located at 0.3, 0.4, 0.45, 0.5, 0.55, 0.6, and 0.7 in $[0, 1]$.

- Multilevel preconditioning with four grid levels:

k	0	1	2	3
grid points	401	201	101	51

Diagonal of H^{-1}



Assessing approximation accuracy

- Riemannian distance:

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

- Compare eigenvalues of H^{-1} and $\tilde{H}^{-1}$ on the finest grid level $k = 0$ using

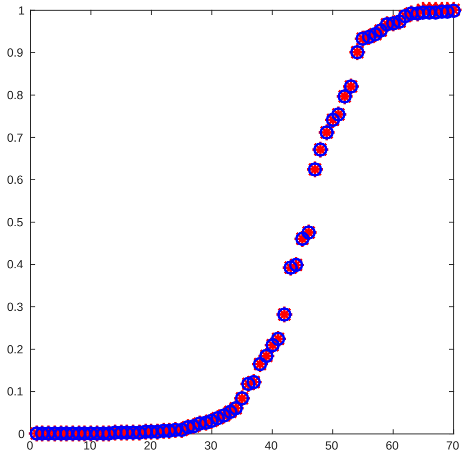
$$D = \frac{\delta(H^{-1}, \tilde{H}^{-1})}{\delta(H^{-1}, I)}$$

- Vary number of eigenvalues chosen on each grid level

$$N_e = (n_0, n_1, n_2, n_3)$$

Eigenvalues of the inverse Hessian

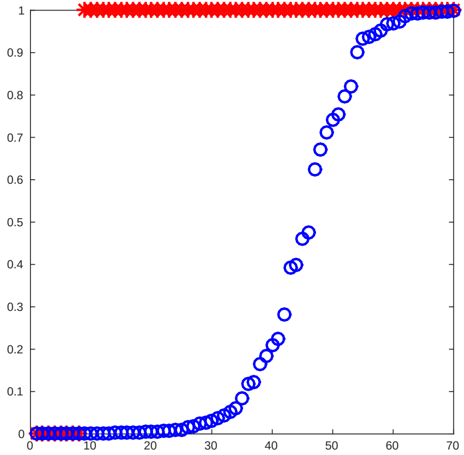
- Exact (blue circles), approximated (red stars)



$$N_e = (64, 0, 0, 0)$$
$$D = 2.98e-4$$

Eigenvalues of the inverse Hessian

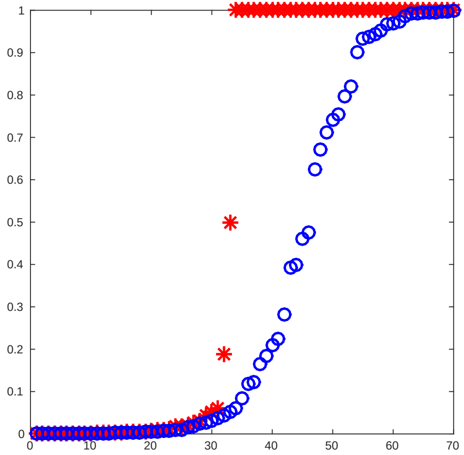
- Exact (blue circles), approximated (red stars)



$$N_e = (8,0,0,0)$$
$$D = 7.71e-1$$

Eigenvalues of the inverse Hessian

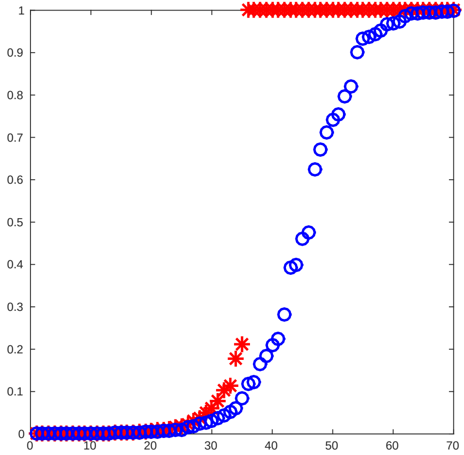
- Exact (blue circles), approximated (red stars)



$$N_e = (0, 6, 13, 14)$$
$$D = 3.95e - 1$$

Eigenvalues of the inverse Hessian

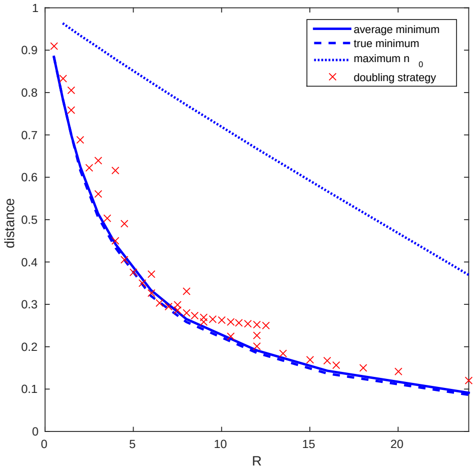
- Exact (blue circles), approximated (red stars)



$$N_e = (0, 0, 29, 6)$$
$$D = 3.39e - 1$$

Fixed memory ratio

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



PCG iteration for one Newton step

- measurement units
  - memory: length of vector on finest grid L
  - cost: cost of HVP on finest grid M

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

Practical versions

- Algorithm 1: apply mlevd to H.
  - Still needs a lot of resources for realistic problems.

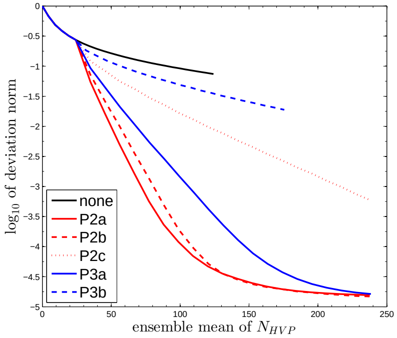
- Algorithm 2: assemble local Hessians for each sensor to form H_g, then apply mlevd to H_g.
  - Local Hessians could be computed at a coarser grid level l (including discretisation of the tangent linear model). ✓
  - Local Hessians cheaper to compute (use n_l eigenvalues), and can be computed in parallel. ✓
  - Need to store additional local Hessians. $\times$

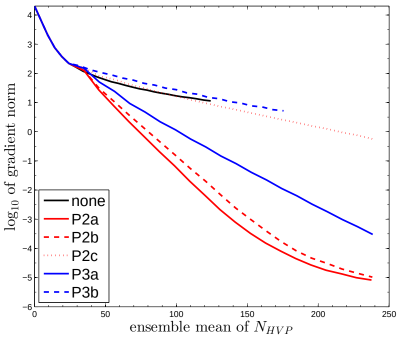
- Algorithm 3: apply mlevd to local inverse Hessians and assemble to form H_g^{rm}, then apply mlevd to H_g^{rm}.
  - Needs less storage than Algorithm 2. ✓
  - Can use N_e^l eigenvalues for local Hessians. ✓
  - Introduces an additional layer of approximation. $\times$

Iteration counts

Preconditioner	Algorithm	N_e	l	n_l	N_e^l
P2a	2	(200,0,0,0)	1	8	-
P2b	2	(0,8,16,32)	1	8	-
P2c	2	(0,4,8,16)	1	8	-
P3a	3	(0,8,16,32)	1	8	(0,0,8,0)
P3b	3	(0,8,16,32)	2	8	(0,0,0,8)

Number of Hessian vector products





Conclusions and next steps

- Similar results with other configurations (e.g. moving sensors, different initial conditions).
- Multilevel preconditioning looks promising for constructing a good limited-memory approximation to H^{-1}.
- The balance between restrictions on memory/cost limitations may vary between particular applications.
- Identifying globally appropriate values for (n_0, n_1, n_2, n_3) and other parameters is tricky.

- Next steps:
  - extension to higher dimensions;
  - applying the evolution model on coarser grids;
  - application to other sensor networks.

Please join us in June!

27th Biennial Numerical Analysis Conference
University of Strathclyde, Glasgow, Scotland
June 27th-30th 2017

`http://numericalanalysisconference.org.uk/`

Hessian decomposition

- partition domain into subregions and compute local Hessians H^l such that

$$H(u) = I + \sum_{l=1}^{L} (H^l(u) - I)$$

- fewer eigenvalues required for limited-memory representation of each H^l
- local Hessians can be computed in parallel
- H^l need not be computed at finest grid level:

$$H_k(u_k) = I_k + \sum_{l=1}^{L} (H_k^l(u_k) - I_k)$$

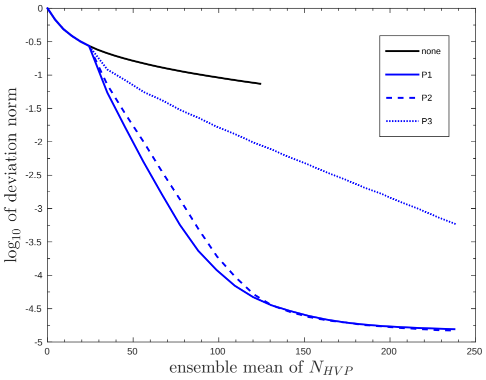
- could run local rather than global model

Practical approach: version 1

- Compute limited-memory approximations to local sensor-based Hessians on level l using n_l eigenpairs.
- Assemble these to form H_a, then apply mlevd to H_a based on a fixed N_e.
- Local Hessians cheaper to compute.
- Additional user-specified parameter(s) l, n_l needed.
- More memory required as local Hessians must also be stored.

Numerical results

Preconditioner	N_e	l	n_l
P1	(200,0,0,0)	1	8
P2	(0,8,16,32)	1	8
P3	(0,4,8,16)	1	8



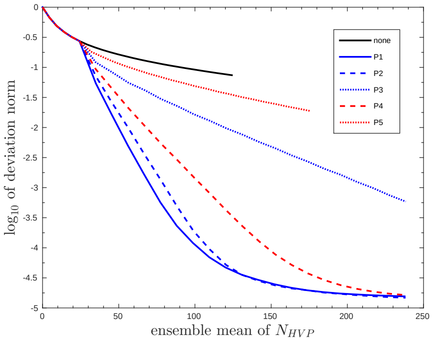
log$_{10}$(error) vs number of HVP

Practical approach: version 2

- Can reduce memory requirements further by using a multilevel approximation of each limited-memory local Hessian on level l using n_l eigenpairs.
- Approximate local Hessians by applying mlevd to local inverse Hessians based on N_e^l.
- Assemble these to form a reduced-memory assembled Hessian H_a^{rm}.
- Use mlevd again on H_a^{rm} based on N_e.

Numerical results

Preconditioner	N_e	l	n_l	N_e^l
P1	(200,0,0,0)	1	8	-
P2	(0,8,16,32)	1	8	-
P3	(0,4,8,16)	1	8	-
P4	(0,8,16,32)	1	8	(0,0,8,0)
P5	(0,8,16,32)	2	8	(0,0,0,8)



$\log_{10}$(error) vs number of HVP