Static B Matrix Update

Jedi Weekly Meeting
2019-09-11

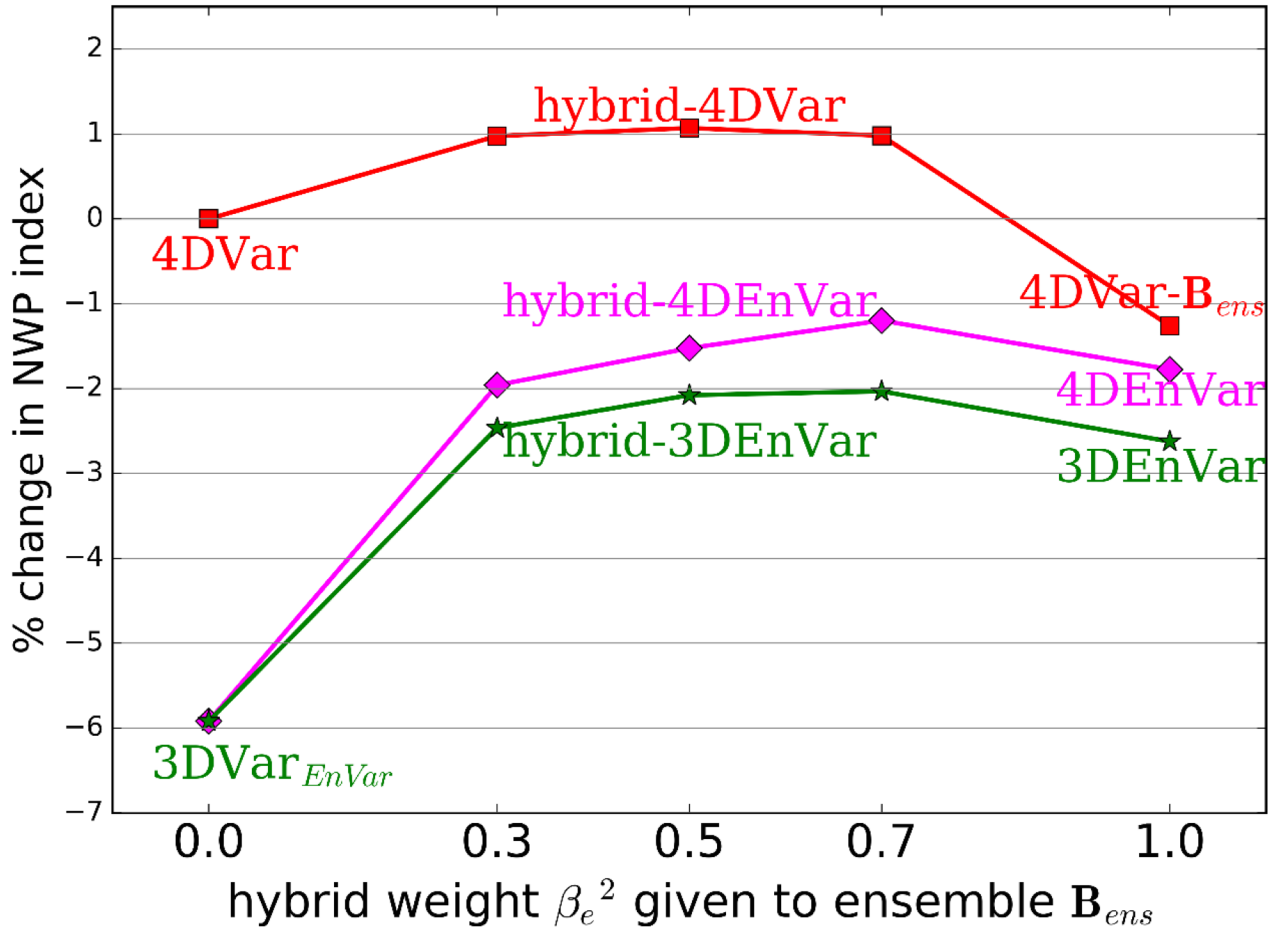
Dan, BJ and Benjamin

Introduction

As shown in Lorenc and Jardak (2018) having a static B matrix gives better performance for the Met Office UM than having a pure ensemble B matrix.

The figure to the right shows a measure of forecast skill with various weights given to the static and ensemble components of the B matrix. The highest skill comes from using a weight of 0.7.

Of course this all relies on having a good static B model.



Static B

B is often written as a product of standard deviation and correlation matrices.

$$B = \Sigma C \Sigma$$

We could attempt to model these multivariate matrices directly using BUMP. However, this is not a sensible approach (Derber & Bouttier, 1999).

Instead the general practice is to model these operators univariately by transforming unbalanced model variables into balanced variables. This makes cross covariances zero.

$$B = K_1 \ldots K_N \underbrace{\Sigma C \Sigma}_{\text{BUMP}} \underbrace{K_n^\top \ldots K_1^\top}_{\text{BUMP + Models}}$$

There is typically a series of *K* operators handling horizontal (Dan and BJ) and vertical balance (Benjamin).

In most operational centers the variables stream function and velocity potential are used. They have have convenient balance characteristics but introduce mathematical complexity.

Stream function and velocity potential

First compute vorticity and divergence

$$\zeta = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$$

$$D = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$$

Then solve the inverse Laplacian (Poisson equation):

$$\psi = \nabla^{-2}\zeta$$

$$\chi = \nabla^{-2}D$$

This is not easy to solve on the kinds of grids used in modern models, e.g. cubed sphere or icosahedral hexagons.

Stream function and velocity potential

Getting from stream function and velocity potential to winds is easy:

$$u = \frac{\partial \psi}{\partial y} + \frac{\partial \chi}{\partial x}$$

$$v = -\frac{\partial \psi}{\partial x} + \frac{\partial \chi}{\partial y}$$

In Cartesian coordinates

Thankfully with linear balance this is and its adjoint is all we need during the minimization.

- We still need to convert perturbations to stream function in order to construct the correlation model but efficiency is not a primary concern since it's done offline.

- In the future we would like to support a nonlinear balance method and that would require solving the Poisson equation (and a Helmholtz equation) during the minimization and having an adjoint of the solvers.

Approach

- Interpolate to a latitude/longitude, Gaussian or spectral grid and use existing static B matrix operators.
  - 😁 Easy
  - ⚠ These solvers typically suffer from scalability issues.
  - ⚠ Additional damping from interpolation.
- Interpolate to lat/lon etc just for the variable transforms.
  - 😁 Easy and can use existing tools e.g. NCL.
  - ⚠ Scalability concerns.
- Use sophisticated external library e.g. FEniCS, Trilinos, PETSc/Tao to perform the transforms.
  - 😁 Fast, well coded and easy to use
  - ⚠ Adds dependency on massive library outside of our control.
  - ⚠ Use e.g. Python APIs making integration into JEDI tricky.
  - ⚠ May lack the adjoint (but some are close to self adjoint).
- Build our own solver
  - 😁 Flexibility to evolve ourselves, easier to maintain. Easier use in the minimization.
  - 😁 Applications centralized in JEDI.
  - ⚠ Potentially much more work.

Divide and conquer

We have decided to try three approaches, divided between JEDI modeling groups.

The MPAS-JEDI group are looking into using FEniCS (native grid) and NCL (Gaussian grid) libraries and developing an offline tool for converting perturbations to stream function and velocity potential. At this stage there is no plan to integrate these libraries into the JEDI infrastructure directly.

The FV3-JEDI group are looking at developing our own online solver that can be integrated into JEDI and potentially be useful in the longer term for nonlinear balance. The Met Office group are also looking into repurposing an existing online solver.

The idea is that the approaches will allow us to make progress in both the short and long term and since the problem is quite complex have a way of comparing any inhouse methods with the trusty libraries.

Meanwhile Benjamin has developed the ability to compute vertical regressions using BUMP with a regional averaging to mimic the zonal averaging used for lat-lon grids.

Finite Element Mesh Poisson Solver (FEMPS)

Originally developed by John Thuburn at the University of Exeter in the UK.

The Poisson problem is discretized in the form: $D_2 M^{-1} \bar{D}_1 L\psi = \zeta$

D are incidence matrices mostly describing grid connectivity. L and M are mass matrices. M has a non-sparse inverse making the problem tricky to solve. Instead of trying to solve in the above form it solves:

$$D_2 \hat{M}^{-1} \bar{D}_1 L\psi = \zeta$$

Where $\hat{M}^{-1}$ is a sparse approximation. Since it solves the approximate problem an outer loop is performed that corrects the guess for $M^{-1} \bar{D}_1 L\psi$ with a multigrid solver to correct ψ.

FEMPS

- The code from John has been extensively refactored and modernized to make it compatible with JEDI and has been placed in its own repository. It has been interfaced to into fv3-jedi in the branch feature/femps. Once fully working the femps repository will be transferred to the JCSDA GitHub.

- Unit testing added that checks $\psi \rightarrow \zeta \rightarrow \psi$ for an analytical function.

- The trickiest part of the setup is to build the connectivity matrices. This has now been completed for the FV3 grid but would need to be completed for other grids. I've added some plotting utilities that can be used to make sure the grid is structured in the arrays correctly and with the neighbours set properly.

- Once the grid structure is set all of the operators for the solver are precomputed in the variable transform constructor. This makes the actual solver step very fast.

Status

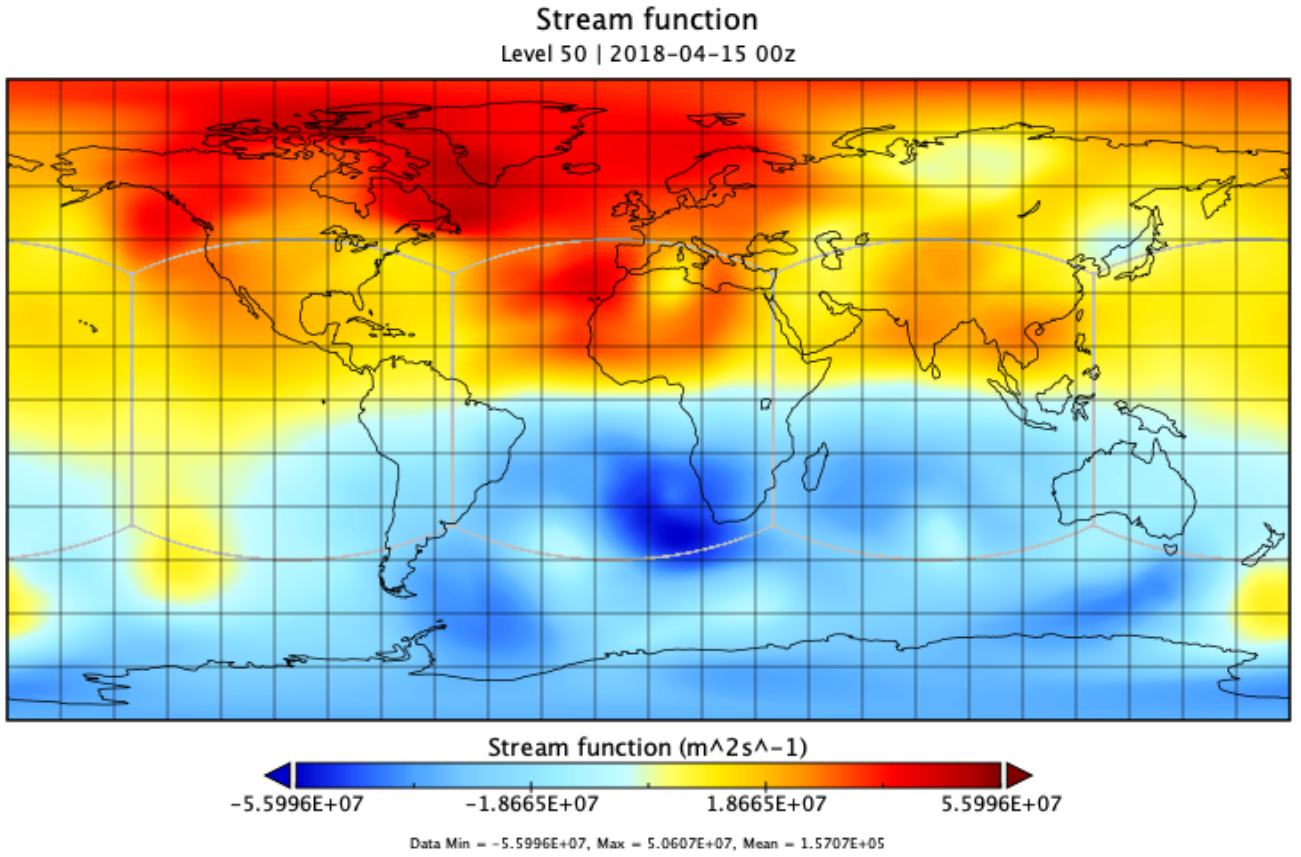
Code is running in fv3-jedi but not working quite properly yet. Magnitude of stream function looks OK, but structure is a bit unrealistic and winds are not correctly recovered with the inverse transform.

Need to look at the A-Grid to A-Grid derivatives.

Timings:

- $u, v \rightarrow \psi, \chi$: ~0.91 seconds per model level per variable. Operator building is ~4.1 seconds.

- $\psi, \chi \rightarrow u, v$: ~0.00008 seconds per level per variable



Ongoing issues

- Not typical to perform horizontal gradients over two grid boxes from A-Grid to A-grid. D-Grid and C-Grid winds may give higher accuracy and utilize existing routines, but would add complexity elsewhere in the code.

- The code runs on a single PE and has no openMP. Domain decomposition is not straightforward due to the multigrid approach. For a 5 grid C96 (100km) hierarchy the coarsest grid (216 grid points, 1600km) is ~0.29% of all grid points. Need to do profiling to figure out where to divert MPI tasks, though not a priority at the moment. For now we can parallelize by level and member.

- Helpful to use resolutions that divide nicely, C90 is not good for example.

- Restriction and prolongation operators for the multigrid solver are only coded for cubed-sphere, icosahedral hexagons and diamond grids. This covers most of the JEDI groups but wont cover e.g. stretched grid versions of those models. ATLAS could be helpful.