Physics based preconditioning in BOUT++

Ben Dudson

York Plasma Institute, University of York, UK

benjamin.dudson@york.ac.uk

4th September 2013

Implicit schemes and preconditioning

- Implicit methods used to solve stiff sets of equations. A high-order BDF scheme is used, but as an illustration a first-order scheme (Backwards Euler) is:

$$\frac{\partial \mathbf{f}}{\partial t} = \mathbf{G}\left(\mathbf{f}\right) \qquad\qquad \mathbf{f}^{n+1} \simeq \mathbf{f}^{n} + \Delta t \mathbf{G}\left(\mathbf{f}^{n+1}\right)$$

- Newton-Krylov solvers used to solve this nonlinear system of equations

$$\mathbf{G}\left(\mathbf{f}^{n+1}\right) \simeq \underbrace{\left.\frac{\partial \mathbf{G}}{\partial \mathbf{f}}\right|_{n}}_{\mathbb{J}} \mathbf{f}^{n+1} \qquad \left(\mathbb{I} - \Delta t \mathbb{J}\right)\mathbf{f}^{n+1} \simeq \mathbf{f}^{n}$$

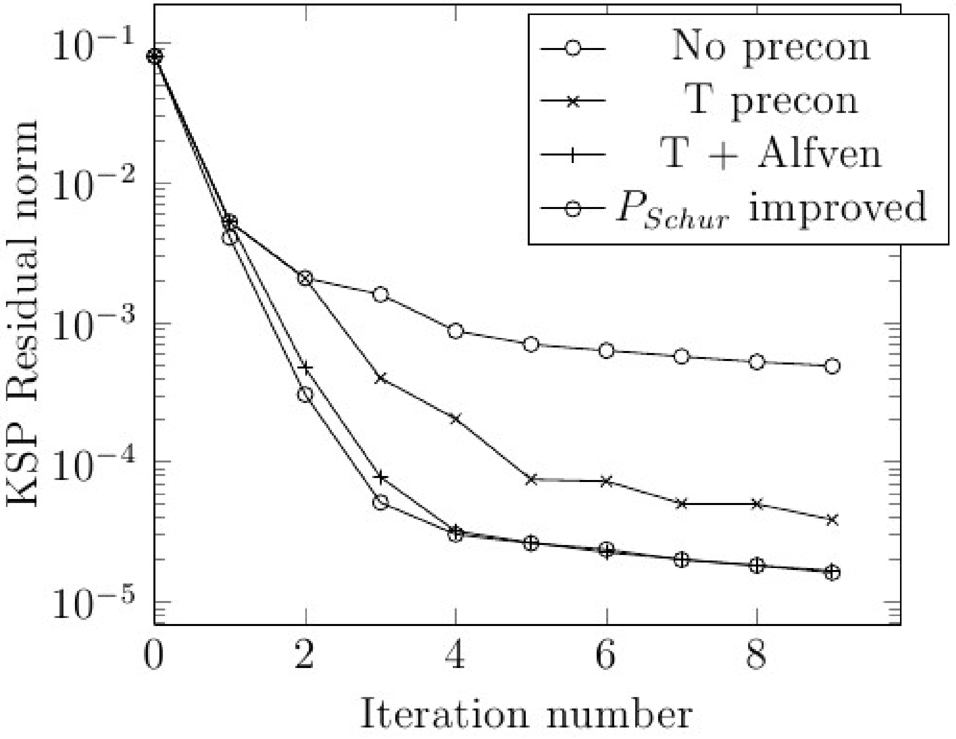
- Typically **f** is ~10 – 100 million variables, so J is a large matrix
- Fortunately we never need to calculate or store J. Instead we use Jacobian Free method:

$$\mathbb{J}\mathbf{v} \simeq \left[\mathbf{G}\left(\mathbf{f}^{n} + \epsilon \mathbf{v}\right) - \mathbf{G}\left(\mathbf{f}^{n}\right)\right] / \epsilon$$

- Fast time scales make this equation more singular and harder to solve → We need a **preconditioner**

Physics-based preconditioning

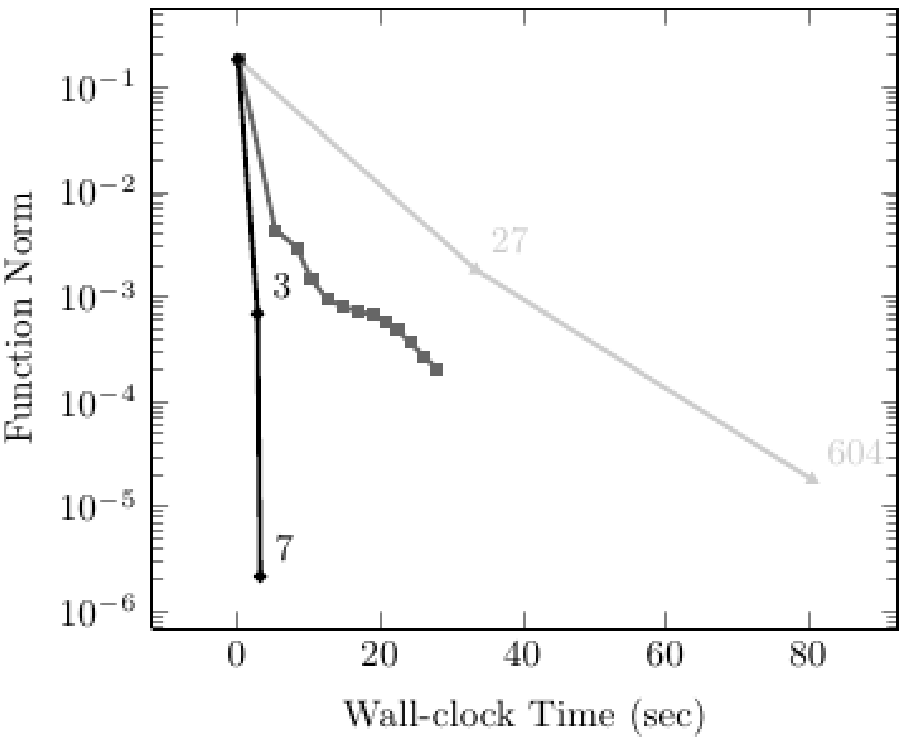
- Typical plasma problems have a wide range of timescales
  → They are "stiff"
- In drift-reduced models these are typically due to shear Alfven waves, and parallel heat conduction
- Assumption of equilibrium flux-surfaces allows reduction of a 3D problem to multiple 1D parabolic solves along field lines
- Can be solved efficiently using FFT + Tridiagonal solve



B Dudson, S Farley, L.C. McInnes
ArXiv: plasm-phys/1209.2054

Physics-based preconditioning

- Typical plasma problems have a wide range of timescales
  → They are "stiff"
- In drift-reduced models these are typically due to shear Alfven waves, and parallel heat conduction
- Assumption of equilibrium flux-surfaces allows reduction of a 3D problem to multiple 1D parabolic solves along field lines
- Can be solved efficiently using FFT + Tridiagonal solve
- For ELM simulations, results in 10 – 100 x speedup



(a) $t = 0$

B Dudson, S Farley, L.C. McInnes
ArXiv: plasm-phys/1209.2054

Preconditioning basics

- To take a timestep with an implicit method, we solve a nonlinear problem using a Newton iteration
- Each iteration requires the solution of a large linear problem

$$A\mathbf{x} = \mathbf{b}$$

- A preconditioner P is an approximate inverse of A which can be applied to the left of the equation:

$$PA\mathbf{x} = P\mathbf{b}$$

or on the right:

$$A\,P\,(P^{-1}\,\mathbf{x}) = \mathbf{b}$$

- So long as P is invertible (non-singular), the result **x** should be independent of choice of P → we can make simplifications in deriving P
- P is chosen to improve the condition number of A, reducing the number of iterations needed to find a solution
- The key is to do this efficiently so that the cost of P is minimised

Preconditioning in BOUT++

Currently supported by the **cvode**, **ida**, and **petsc(>=3.3)** solvers

→ see **examples/test-precon** for simple example

Define a function to calculate P * vector multiply

```
int precon(BoutReal t, BoutReal gamma, BoutReal delta) {
  return 0;
}
```

gamma is (approx.) the timestep (depends on method)
delta only needed for constraints. Ignore here

- System state is stored in variables, as for RHS function
- Input vector is in “time-derivatives” ddt(variables)
- Output vector also in ddt(variables)
  → The above function is the identity operator

Physics based preconditioning

- There are many ways to derive a preconditioner, which can be broadly split into two categories:
  - General, black box methods, which use the structure of the matrix in a generic solver e.g. Jacobi, SOR, GAMG, …
  - Physics based methods, which use some physical insight to simplify the equations solved by the preconditioner, to focus on the fastest timescales

- Here we will look at a form of preconditioner popularised by L.Chacon (ORNL)

- See talk from 2011 BOUT++ workshop https://bout.llnl.gov/pdf/workshops/2011/talks/Chacon_bout2011.pdf

Recipe for physics-based preconditioning

1) Simplify the equations
2) Calculate Jacobian. Partial derivatives of RHS w.r.t variables
3) Factorise the matrix to be solved
4) Use an approximation to decouple parallel and perpendicular derivatives
5) Implement using the same operators as the time-derivative evaluation. Implemented as another call-back function
6) Tweak, add and remove terms to optimise performance

1D wave example

See **examples/test-precon** and user manual

- Start with a wave equation

$$\frac{\partial u}{\partial t} = \partial_{||} v \qquad \frac{\partial v}{\partial t} = \partial_{||} u$$

(Example used in L.Chacon talk, 2011 workshop)

- Calculate Jacobian (partial derivatives)

$$\mathcal{J} = \left(\begin{array}{cc} \frac{\partial}{\partial u}\frac{\partial u}{\partial t} & \frac{\partial}{\partial v}\frac{\partial u}{\partial t} \\ \frac{\partial}{\partial u}\frac{\partial v}{\partial t} & \frac{\partial}{\partial v}\frac{\partial v}{\partial t} \end{array} \right) = \left(\begin{array}{cc} 0 & \partial_{||} \\ \partial_{||} & 0 \end{array} \right)$$

$$\mathcal{I} - \gamma\mathcal{J} = \left(\begin{array}{cc} 1 & -\gamma\partial_{||} \\ -\gamma\partial_{||} & 1 \end{array} \right)$$

1D wave example

- Block factorise this matrix

$$\begin{pmatrix} \mathbf{E} & \mathbf{U} \\ \mathbf{L} & \mathbf{D} \end{pmatrix}^{-1} = \begin{pmatrix} \mathbf{I} & -\mathbf{E}^{-1}\mathbf{U} \\ 0 & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{E}^{-1} & 0 \\ 0 & \mathbf{P}_{Schur}^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{I} & 0 \\ -\mathbf{L}\mathbf{E}^{-1} & \mathbf{I} \end{pmatrix}$$

$$\mathbf{P}_{Schur} = \mathbf{D} - \mathbf{L}\mathbf{E}^{-1}\mathbf{U}$$

- For this problem, this becomes:

$$\begin{pmatrix} 1 & -\gamma\partial_{||} \\ -\gamma\partial_{||} & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & \gamma\partial_{||} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \left(1 - \gamma^2\partial_{||}^2\right)^{-1} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \gamma\partial_{||} & 1 \end{pmatrix}$$

- These operators can now be implemented in BOUT++

```
int precon(BoutReal t, BoutReal gamma, BoutReal delta) {
}
```

Input and output vector in 'ddt' variables

$$\begin{pmatrix} \mathtt{ddt(u)} \\ \mathtt{ddt(v)} \end{pmatrix}$$

1D wave example

- Apply matrices right to left

$$\begin{pmatrix} \texttt{ddt(u)} \\ \texttt{ddt(v)} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \gamma\partial_{||} & 1 \end{pmatrix} \begin{pmatrix} \texttt{ddt(u)} \\ \texttt{ddt(v)} \end{pmatrix}$$

```
int precon(BoutReal t, BoutReal gamma, BoutReal delta) {
  mesh->communicate(ddt(u));
  //ddt(u) = ddt(u);
  ddt(v) = gamma*Grad_par(ddt(u)) + ddt(v);
```

- Key step is the inversion of P_{schur}, which must be efficient

$$\begin{pmatrix} \texttt{ddt(u)} \\ \texttt{ddt(v)} \end{pmatrix} \leftarrow \begin{pmatrix} 1 & 0 \\ 0 & \left(1 - \gamma^2\partial_{||}^2\right)^{-1} \end{pmatrix} \begin{pmatrix} \texttt{ddt(u)} \\ \texttt{ddt(v)} \end{pmatrix}$$

```
InvertPar *inv; // Parallel inversion class
int physics_init(bool restarting) {
   ...
   inv = InvertPar::Create();
   inv->setCoefA(1.0);
   ...
}
```

```
inv->setCoefB(-SQ(gamma));
ddt(v) = inv->solve(ddt(v));
```

1D wave example

- To use this preconditioner, we need to pass the function pointer to the solver during initialisation

```
int physics_init(bool restarting) {
  solver->setPrecon(precon);
  ...
}
```

- Tell the solver to use the preconditioner in the input options

```
[solver]
type = cvode
use_precon = true
rightprec = false
```

- Currently supported by **cvode** (SUNDIALS) and **petsc** (>=3.3) solvers

1D wave example: Instructions

First we need to compile BOUT++ with SUNDIALS and/or PETSc. See BOUT++ user manual for how to install these packages.

For now, using SUNDIALS and PETSc already installed on Hopper

1) Log into Hopper

2) Run workshop configuration script:

```
cd BOUT-2.0
source configure.workshop
```

1D wave example: Instructions

First we need to compile BOUT++ with SUNDIALS and/or PETSc. See BOUT++ user manual for how to install these packages.

For now, using SUNDIALS and PETSc already installed on Hopper

```
Configuration summary

  FACETS support: no
  PETSc support: yes (version 3.3, release = 1)
  PETSc has SUNDIALS support: no
  IDA support: yes
  CVODE support: yes
  NetCDF support: yes
  Parallel-NetCDF support: no
  PDB support: no
  Hypre support: no
  MUMPS support: yes
```

→ make

1D wave example: Instructions

First we need to compile BOUT++ with SUNDIALS and/or PETSc

```
source configure.workshop
```

Re-compile the BOUT++ library

```
make
```

Change to the test-precon directory, compile and run

```
cd examples/test-precon
make
```

```
1.000e+01        115       7.70e-01    -5.4    0.0   15.9    1.8   87.7
|  Step 1 of 10. Elapsed 0:00:00.0 ETA 0:00:06.9
CVODE: nsteps 42, nfevals 66, nniters 65, npevals 0, nliters 74
    -> Newton iterations per step: 1.547619e+00
    -> Linear iterations per Newton iteration: 1.138462e+00
    -> Preconditioner evaluations per Newton: 0.000000e+00
```

1D wave example: Instructions

First we need to compile BOUT++ with SUNDIALS and/or PETSc

```
./configure --with-sundials --with-petsc
```

Re-compile the BOUT++ library

```
make
```

Change to the test-precon directory, compile and run

```
cd examples/test-precon
make
```

Try turning on and off preconditioning in BOUT.inp options:

```
[solver]
type = cvode              # Need CVODE or PETSc
use_precon = true         # <----
```

Using PETSc for diagnostics

- One of the nice features of PETSc is its extensive monitoring capabilities, which help in optimising preconditioners
- First set the solver type to petsc, either in BOUT.inp or command line

```
[solver]
type = petsc
```

```
solver:type=petsc
```

- PETSc options can then be set on the command line e.g. to select the theta method with = 0.5 (i.e. Crank-Nicholson)

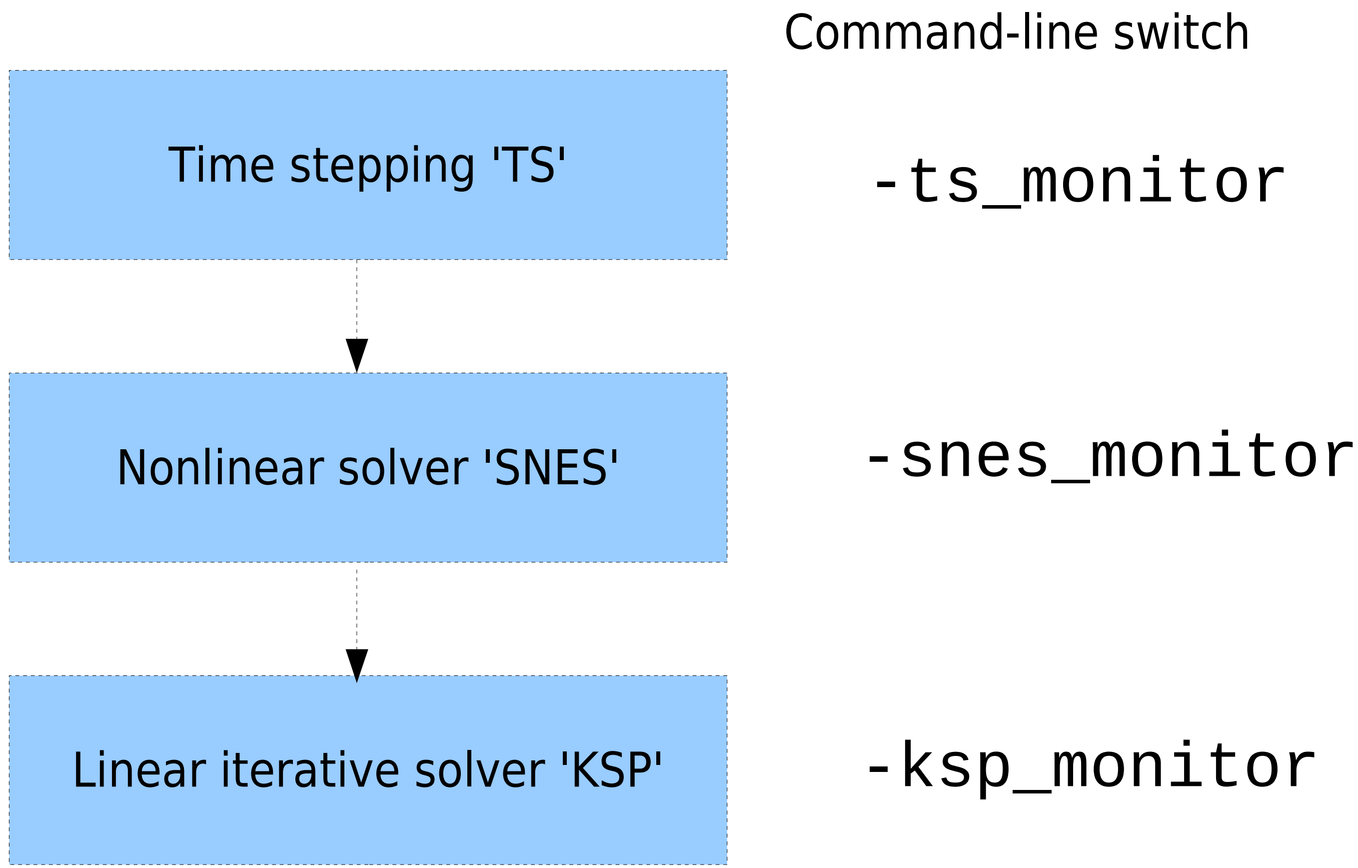
```
-ts_type theta -ts_theta_theta 0.5
```

- Fixed timestep method using BOUT++ output timestep

```
timestep=10
```

Using PETSc for diagnostics

- Monitoring can be enabled for various components of PETSc



1D wave example: Instructions (2)

Modify the BOUT.inp file to use PETSc time stepping solver

```
[solver]
type = petsc
use_precon = true
```

Run with command-line options

```
-ts_type theta -ts_theta_theta 0.5 -{ksp,snes,ts}_monitor
```

```
0.000e+00        1      1.08e-01   1091.8  300.8   65.4  217.0  -1575.0
|  Step 1 of 50. Elapsed 0:00:00.0 ETA 0:00:05.3 Wall 3:59:60.00 TS dt 1000 time 0
    0 SNES Function norm 5.814140098646e-05
      0 KSP Residual norm 5.814140098646e-05
      1 KSP Residual norm 5.814111317072e-05
      2 KSP Residual norm 3.183140943212e-05
      3 KSP Residual norm 3.183140939101e-05
      4 KSP Residual norm 2.372086517703e-05
```

Reconnect-2field example

- A 3D slab forced reconnection problem

- Contains shear Alfven wave with short timescales relative to long timescale of reconnection process → Benefits from preconditioning

$$
\begin{aligned}
\frac{\partial A_{||}}{\partial t} &= -\frac{1}{\hat{\beta}}\nabla_{||}\phi - \frac{1}{\hat{\beta}}\eta j_{||} \\
\frac{\partial U}{\partial t} &= -\mathbf{v}_{E\times B}\cdot\nabla U + B_0^2\nabla_{||}\left(\frac{J_{||}+J_{||0}}{B_0}\right)
\end{aligned}
$$

$$
U = \frac{1}{B_0}\nabla_\perp^2\phi \qquad j_{||} = -\nabla_\perp^2 A_{||}
$$

- Contains basic physics present in most plasma problems of interest → this same preconditioner can be applied to many models, including elm-pb 3-field model

Reconnect-2field example

Follow same procedure as for 1D wave example

1) Simplify equations

$$\frac{\partial A_{||}}{\partial t} = -\frac{1}{\hat{\beta}}\nabla_{||}\phi \qquad \frac{\partial U}{\partial t} = B_0^2\nabla_{||}\left(\frac{J_{||}}{B_0}\right)$$

$$U = \frac{1}{B}\nabla_\perp^2\phi \qquad J_{||} = -\nabla_\perp^2 A_{||}$$

2) Calculate Jacobian analytically

$$\left(\begin{array}{c} A_{||} \\ U \end{array}\right)$$

$$\mathcal{J} = \left(\begin{array}{cc} \frac{\partial}{\partial A_{||}}\frac{\partial A_{||}}{\partial t} & \frac{\partial}{\partial U}\frac{\partial A_{||}}{\partial t} \\ \frac{\partial}{\partial A_{||}}\frac{\partial U}{\partial t} & \frac{\partial}{\partial U}\frac{\partial U}{\partial t} \end{array}\right)$$

$$= \left(\begin{array}{cc} 0 & -\nabla_{||}\frac{1}{\hat{\beta}}\nabla_\perp^{-2}B_0\cdot \\ -B_0^2\nabla_{||}\frac{1}{B_0}\nabla_\perp^2\cdot & 0 \end{array}\right)$$

Reconnect-2field example

Follow same procedure as for 1D wave example

$$\begin{pmatrix} A_{||} \\ U \end{pmatrix}$$

1) Simplify equations
2) Calculate Jacobian analytically
3) Factorise

$$\mathcal{I} - \gamma\mathcal{J} = \begin{pmatrix} I & \gamma\nabla_{||}\nabla_{\perp}^{-2}B_0\cdot \\ \gamma B_0^2\nabla_{||}\frac{1}{B_0}\nabla_{\perp}^2\cdot & I \end{pmatrix}$$

$$= \begin{pmatrix} E & U \\ L & D \end{pmatrix}$$

$$(\mathcal{I} - \gamma\mathcal{J})^{-1} = \begin{pmatrix} I & \gamma\nabla_{||}\frac{1}{\hat{\beta}}\nabla_{\perp}^{-2}B_0\cdot \\ 0 & I \end{pmatrix}\begin{pmatrix} I & 0 \\ 0 & P_{schur} \end{pmatrix}\begin{pmatrix} I & 0 \\ -\gamma B_0^2\nabla_{||}\frac{1}{B_0}\nabla_{\perp}^2\cdot & I \end{pmatrix}$$

$$P_{Schur} = I - \gamma B_0^2\nabla_{||}\frac{1}{B_0}\nabla_{\perp}^2\gamma\nabla_{||}\frac{1}{\hat{\beta}}\nabla_{\perp}^{-2}B_0\cdot$$

Reconnect-2field example

$$\begin{pmatrix} A_{||} \\ U \end{pmatrix}$$

Follow same procedure as for 1D wave example

1) Simplify equations
2) Calculate Jacobian analytically
3) Factorise
4) Simplify to decouple parallel and perpendicular

$$P_{Schur} = I - \gamma B_0^2 \nabla_{||} \frac{1}{B_0} \nabla_\perp^2 \gamma \nabla_{||} \frac{1}{\hat{\beta}} \nabla_\perp^{-2} B_0.$$

$$P_{Schur} \simeq I - \gamma^2 \frac{B_0^2}{\hat{\beta}} \nabla_{||}^2.$$

Can be solved using InvertPar solver: $A + B\nabla_{||}^2$

Reconnect-2field example

$$\begin{pmatrix} A_{||} \\ U \end{pmatrix}$$

Follow same procedure as for 1D wave example

1) Simplify equations
2) Calculate Jacobian analytically
3) Factorise
4) Simplify to decouple parallel and perpendicular
5) Implement in BOUT++

$$P_{Schur} \simeq I - \gamma^2 \frac{B_0^2}{\hat{\beta}} \nabla_{||}^2 \cdot$$

```
InvertPar *inv;

inv = InvertPar::Create();
inv->setCoefA(1.0);

inv->setCoefB(-SQ(gamma*Bxy)/beta_hat);
ddt(Upar) = inv->solve(Upar1);
```

Exercises

1) Run the **test-precon** example for wave equation using petsc solver with and without preconditioner

2) Vary the timestep, and test the effectiveness of the preconditioner. Note the damping of the wave once timesteps become large
   → A common effect of implicit timestepping methods on unresolved timescales

3) Try the **reconnect-2field** test case, with and without preconditioning, using cvode and petsc solvers

4) Try preconditioning options in **elm-pb** example

5) Try adding a preconditioner for the diffusion test case **examples/conduction**