A parallel code to solve for the quasi-steady dynamics of stellar interiors

Pascale Garaud, AMS.

11 March 2008
All stars rotate to some degree. Centrifugal forces and Coriolis forces induced by rotation can

- change the properties of eddies in convective regions
- induce flows in radiative regions

All stars are magnetic to some degree. Lorentz forces induced by the magnetic field can

- change the properties of eddies in convective regions
- induce flows in radiative regions
How does this affect the intrinsic structure and evolution of a star?

How does it affect its observable properties?
The equations governing of the spherically symmetric structure & evolution of the stellar interior are:

\[-\frac{d\bar{p}}{dr} = \bar{\rho}\frac{d\Phi}{dr},\]
\[\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\Phi}{dr}\right) = 4\pi G\bar{\rho},\]
\[\frac{T}{d}dS = \bar{\epsilon} - \frac{1}{r^2}\frac{d}{dr}(r^2F_{\text{rad}} + r^2F_{\text{conv}}),\]
\[\frac{dX}{dt} = f(\bar{\rho}, \bar{T}, X),\]
\[\bar{p} = \bar{p}(\bar{\rho}, \bar{T}, X), \bar{\kappa}_R = \bar{\kappa}_R(\bar{\rho}, \bar{T}, X), \bar{\epsilon} = \bar{\epsilon}(\bar{\rho}, \bar{T}, X)\]

The evolution equations are quasi-linear PDEs for 1 spatial variable and 1 temporal variable.

The structure equations (i.e. the steady-state solution for given composition) require the solution of a two-point boundary value problem (elliptic).
For a given background state \((\bar{\rho}, \bar{T}, \ldots, \text{known})\) the equations governing the 3D anelastic magnetohydrodynamics of the stellar interior are:

\[
\frac{\partial \mathbf{u}}{\partial t} + \bar{\rho} \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p - \rho \nabla \Phi + j \times \mathbf{B} + \nabla \cdot \Pi
\]

\[
\nabla \cdot (\bar{\rho} \mathbf{u}) = 0
\]

\[
\frac{\partial s}{\partial t} + \bar{\rho} \mathbf{u} \cdot \nabla s = \nabla \cdot (k \nabla T)
\]

\[
p = p(\rho, T)
\]

\[
\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{u} \times \mathbf{B}) - \nabla \times (\eta \nabla \times \mathbf{B})
\]

\[
\nabla \cdot \mathbf{B} = 0
\]

These equations are (again) **quasi-linear** PDEs, for 3 spatial variables and 1 temporal variable.

I have been interested in studying the axially symmetric, steady-state solutions of these equations. Finding these solutions requires solving a **2D nonlinear elliptic problem**.
The numerical solution of all of these types of equations (quasi-linear PDEs, or elliptic PDEs) usually requires inverting a very large block-tridiagonal system of linear equations.

In what follows, I will

- show where and how these large block-tridiagonal system of linear equations arise from PDEs

- how these systems are traditionally solved serially, and what problems arise for very large systems.

- show how to solve them numerically using parallel algorithms.
Nonlinear Elliptic Problems
Nonlinear elliptic problems part 1: the basic idea.

Consider the simple nonlinear ODE

\[ u_x = f(x, u) \]

where \( f \) can be any nonlinear function, with boundary conditions

\[ g_A(x_A, u, u_x) = 0 \text{ or } g_B(x_B, u, u_x) = 0 \]

where \( g_A \) (or \( g_B \)) can be any nonlinear functions. A common method of solution is the **Newton-Raphson-Kantorovich method (NRK)**.

**The idea behind NRK:**

(1) Discretize the ODE: let \( u_n = u(x_n) \), with \( n = 1, \ldots, N \) then

\[
\frac{u_{n+1} - u_n}{x_{n+1} - x_n} = \frac{1}{2} [f(x_{n+1}, u_{n+1}) + f(x_n, u_n)]
\]

The \((N - 1)\) discretized equations are second-order accurate half-way between the meshpoints:
(2) Write \( u_n = \overline{u}_n + \delta u_n \) where \( \overline{u}_n \) is a “guess” for the real solution \( u_n \). Taylor-expand the discretized equations assuming \( \delta u_n \) is small:

\[
\frac{\overline{u}_{n+1} + \delta u_{n+1} - \overline{u}_n - \delta u_n}{\Delta x_n} = \frac{1}{2} \left[ f(x_{n+1}, \overline{u}_{n+1}) + \delta u_{n+1} \frac{\partial f}{\partial u}|_{x_{n+1}, \overline{u}_{n+1}} + f(x_n, \overline{u}_n) + \delta u_n \frac{\partial f}{\partial u}|_{x_n, \overline{u}_n} \right]
\]

where \( \Delta x_n = x_{n+1} - x_n \).

(3) Also write the boundary condition in the same form: (for example, if the boundary condition is at the first boundary)

\[
g_A(x_1, \overline{u}_1) + \delta u_1 \frac{\partial g_A}{\partial u}|_{x_1, \overline{u}_1} = 0
\]
This yields an $N \times N$ linear system for the “errors” $\delta u_n$:

$$
\begin{pmatrix}
G_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A_1 & B_1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & A_2 & B_2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & A_3 & B_3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & A_4 & B_4 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & A_5 & B_5 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & A_6 & B_6 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & A_7 & B_7
\end{pmatrix}
\begin{pmatrix}
\delta u_1 \\
\delta u_2 \\
\delta u_3 \\
\delta u_4 \\
\delta u_5 \\
\delta u_6 \\
\delta u_7 \\
\delta u_8
\end{pmatrix} =
\begin{pmatrix}
-g_A \\
C_1 \\
C_2 \\
C_3 \\
C_4 \\
C_5 \\
C_6 \\
C_7
\end{pmatrix}
$$

where

$$
G_1 = \left. \frac{\partial g_A}{\partial u} \right|_{x_1, \overline{u}_1} \quad \text{and} \quad -g_A = -g_A(x_1, \overline{u}_1)
$$

$$
A_n = -1 - \frac{\Delta x_n}{2} \left. \frac{\partial f}{\partial u} \right|_{x_n, \overline{u}_n}
$$

$$
B_n = -1 + \frac{\Delta x_n}{2} \left. \frac{\partial f}{\partial u} \right|_{x_{n+1}, \overline{u}_{n+1}}
$$

$$
C_n = - (\overline{u}_n + \overline{u}_{n+1}) + \frac{\Delta x_n}{2} \left[ f(x_n, \overline{u}_n) + f(x_{n+1}, \overline{u}_{n+1}) \right]
$$
(5) Once calculated, the solution of this linear system $\delta u_n$ is added on to the initial guess:

$$\bar{u} := \bar{u} + \delta u$$

to provide a “better” approximation to the real solution.

(6) The method can be iterated upon many times until the desired accuracy is achieved: a driver for the method would then read

Initialize guess.
do iter = 1,nitermax

- Solve for $\delta u$
- $\bar{u} := \bar{u} + \delta u$
- Evaluate the norm of $\delta u$ to estimate the error
- If error is small enough, exit loop.

enddo
Nonlinear elliptic problems part 2: systems of ODEs.

The method described earlier can naturally be extended to systems of coupled nonlinear ODEs as well. Given the interval \((x_A, x_B)\), consider the system of \(I\) nonlinear equations

\[
\frac{\partial u^i}{\partial x} = f^i(x, u) \text{ for } i = 1, \ldots, I
\]

where \(u = (u^1, u^2, \ldots, u^I)\), and the functions \(f^i\) can be any nonlinear function. The boundary conditions are split between the two boundaries, with

\[
g^k_A(x_A, u) = 0 \text{ for } k = 1, \ldots, k_A
\]
\[
g^k_B(x_B, u) = 0 \text{ for } k = k_{A+1}, \ldots, I
\]

where the functions \(g^k_A\) and \(g^k_B\) can be any nonlinear functions.

As before, we discretize the equations upon the mesh, and consider that for each dependent variable \(u_i\) at each meshpoint \(x_n\), \(u^i_n = u^i(x_n)\) and write them as the sum of a “guess” plus and “error”:

\[
u^i_n = \bar{u}^i_n + \delta u^i_n
\]
This time, the errors $\delta u^i_n$ are the solution of the **block-tridiagonal** linear system

$$
\begin{pmatrix}
G_A & 0 & 0 & 0 & 0 & 0 & 0 \\
A_1 & B_1 & 0 & 0 & 0 & 0 & 0 \\
0 & A_2 & B_2 & 0 & 0 & 0 & 0 \\
0 & 0 & A_3 & B_3 & 0 & 0 & 0 \\
0 & 0 & 0 & A_4 & B_4 & 0 & 0 \\
0 & 0 & 0 & 0 & A_5 & B_5 & 0 \\
0 & 0 & 0 & 0 & 0 & A_6 & B_6 \\
0 & 0 & 0 & 0 & 0 & 0 & A_7 & B_7 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & G_B
\end{pmatrix}
\begin{pmatrix}
\delta u^1 \\
\delta u^2 \\
\delta u^3 \\
\delta u^4 \\
\delta u^5 \\
\delta u^6 \\
\delta u^7 \\
\delta u^8 \\
\delta u^9 \\
\delta u^{10}
\end{pmatrix} =
\begin{pmatrix}
-g_A \\
C_1 \\
C_2 \\
C_3 \\
C_4 \\
C_5 \\
C_6 \\
C_7 \\
-g_B
\end{pmatrix}
$$

where the **matrices** $G_A$, $G_B$, $A_n$, $B_n$, have coeffs:

$$(G_A)_{ij} = \frac{\partial g_A^i}{\partial u^j} \bigg|_{x_1, \bar{u}_1}$$

for $i = 1, ..., k_A$, $j = 1, .., I$

$$(G_B)_{ij} = \frac{\partial g_B^{i+k_A}}{\partial u^j} \bigg|_{x_N, \bar{u}_N}$$

for $i = 1, ..., I - k_A$, $j = 1, .., I$

$$(A_n)_{ij} = -\delta_{ij} - \frac{\Delta x_n}{2} \frac{\partial f^i}{\partial u^j} \bigg|_{x_n, \bar{u}_n}$$

$$(B_n)_{ij} = -\delta_{ij} + \frac{\Delta x_n}{2} \frac{\partial f^i}{\partial u^j} \bigg|_{x_{n+1}, \bar{u}_{n+1}}$$

and the **vectors** $g_A$, $g_B$, and $C_n$ are

$$(g_A)_i = -g_A^i(x_1, \bar{u}_1) \text{ for } i = 1, .., k_A$$

$$(g_B)_i = -g_B^{i+k_A}(x_N, \bar{u}_N) \text{ for } i = 1, ..., I - k_A, j = 1, .., I$$

$$(C_n)_i = -(\bar{u}_n^i + \bar{u}_{n+1}) + \frac{\Delta x_n}{2} \left[ f^i(x_n, \bar{u}_n) + f^i(x_{n+1}, \bar{u}_{n+1}) \right]$$
Note that the matrix is indeed block-tridiagonal: to see this, note that the first block-line only has \( k_A \) lines and the last one only has \( k_B \) lines. Emphasize the tridiagonal nature of the problem by looking is as

Each of the main blocks \( A_n \) or \( B_n \) is \( I \times I \). There are \((N-1)\) such block-lines. The total size of the matrix is \( IN \times IN \rightarrow \text{could be huge!} \)
Nonlinear elliptic problems part 3: from a small set of 2D PDEs to a large set of 1D ODEs.

Consider now for example the 2D, quadratically nonlinear elliptic problem

$$uu_x = u_{xx} + u_{yy}$$

on the unit square ($x \in [0, 1]$, $y \in [0, 1]$) with zero Dirichlet condition ($u(x, y) = 0$ on the boundary).

One possible way of solving the system is to expand one of the two directions in Fourier modes, say the $x-$direction for example: let

$$u(x, y) = \sum_{m=1, M} u_m(y) \sin(m\pi x)$$

(the sum is truncated at finite $M$ since numerically we will only be able to solve the problem for a finite number of modes).

The governing nonlinear PDE is then reduced to the large coupled set of nonlinear second-order ODEs:

$$\frac{d^2u_m}{d^2y} - m^2\pi^2 u_m = \sum_{k=1, M} \frac{k\pi}{2} u_k(u_{m-k} + u_{m+k} - u_{k-m})$$

(where by definition $u_k = 0$ if $k$ is greater than $M$ or smaller than 1).
By defining the new functions

\[ U_m = u_m \]
\[ U_{M+m} = \frac{du_m}{dy} \]

we end up with a set of \(2M\) coupled nonlinear first-order ODEs

\[
\frac{dU_m}{dy} = U_{m+M} \\
\frac{dU_{M+m}}{dy} = m^2 \pi^2 U_m + \sum_{k=1,M} k \pi \frac{U_k(U_{m-k} + U_{m+k} - U_{k-m})}{2}
\]

which is exactly in the form studied in part 2, with the \(2M\) boundary conditions split between the two boundaries (at \(y = 0\) and \(y = 1\)) as

\[ U_m(0) = 0 \text{ and } U_m(1) = 0 \]

If we wanted to study the solutions to this equation with, for example,

- 500 meshpoints in the \(y\) direction
- 100 spectral modes

we would need to invert a \(100,000 \times 100,000\) block-tridiagonal matrix (where there are 500 block-lines each containing blocks of size \(200 \times 200\)).
Nonlinear elliptic problems part 4: my own code

My aim is to find 2D (axially symmetric) solutions of the following set of quadratically nonlinear PDEs, in a spherical geometry:

\[
\begin{align*}
\bar{\rho} u \cdot \nabla u &= -\nabla p - \rho \nabla \Phi + j \times B + \nabla \cdot \Pi \\
\nabla \cdot (\bar{\rho} u) &= 0 \\
\bar{\rho} T u \cdot \nabla s &= \nabla \cdot (k \nabla T) \\
p &= p(\rho, T) \\
\nabla \times (u \times B) &= \nabla \times (\eta \nabla \times B) \\
\nabla \cdot B &= 0
\end{align*}
\]

The unknown variables are the thermodynamic quantities \( \rho, T, p \), the velocity vector \( u = (u_r, u_\theta, u_\phi) \), and the magnetic field vector \( B = (B_r, B_\theta, B_\phi) \). Each can depend on the radius \( r \) and the co-latitude \( \theta \).

This set of PDEs (there are actually 10 equations, some algebraic, some first order, some second order) can be expanded upon a set of \( M \) Chebishev functions in \( \theta \), and results in a system of \( 15M \) equations (some first-order, some algebraic).

My high-resolution simulations typically have 2000 mesh-points in \( r \), and up to 100 spectral modes. Then

- each block of the main matrix is \( 1,500 \times 1,500 \)
- there are 2000 block-lines
- the total size of the matrix would be \( 3,000,000 \times 3,000,000 \)
Serial solution of block-tridiagonal systems
(see Numerical Recipes for example)
Let’s consider the following system, generated from 4 equations and 4 meshpoints, with 3 boundary conditions at the first point and 1 at the last one:
The serial algorithm consists in the following steps:

- Perform a pivoted Gauss-Jordan elimination of the first block (to transform it to the identity), **update** the rest of the matrix as well as the RHS vector (not shown here) and **store** the entries $S$ for future backsubstitution.

\[
\begin{pmatrix}
1 & 0 & 0 & S \\
0 & 1 & 0 & S \\
0 & 0 & 1 & S \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
X & X & X & X \\
\end{pmatrix}
\]
• Zero the block below the first block (and perform the corresponding ops on the rest of the matrix and RHS). Note that this doesn’t affect the stored entries.

\[
\begin{pmatrix}
1 & 0 & 0 & S \\
0 & 1 & 0 & S \\
0 & 0 & 1 & S \\
0 & 0 & 0 & X & X & X & X & X \\
0 & 0 & 0 & X & X & X & X & X \\
0 & 0 & 0 & X & X & X & X & X \\
0 & 0 & 0 & X & X & X & X & X \\
0 & 0 & 0 & X & X & X & X & X \\
0 & 0 & 0 & X & X & X & X & X \\
0 & 0 & 0 & X & X & X & X & X \\
0 & 0 & 0 & X & X & X & X & X \\
0 & 0 & 0 & X & X & X & X & X \\
0 & 0 & 0 & X & X & X & X & X \\
0 & 0 & 0 & X & X & X & X & X \\
0 & 0 & 0 & X & X & X & X & X \\
0 & 0 & 0 & X & X & X & X & X \\
0 & 0 & 0 & X & X & X & X & X \\
0 & 0 & 0 & X & X & X & X & X \\
0 & 0 & 0 & X & X & X & X & X \\
0 & 0 & 0 & X & X & X & X & X \\
0 & 0 & 0 & X & X & X & X & X \\
\end{pmatrix}
\]
• Perform a Gauss-Jordan elimination on the diagonal block (to transform it to the identity), update the rest of the matrix (note that this doesn’t affect the previously stored block nor the zeroed block), update the RHS vector and store the entries marked $S$.

\[
\begin{pmatrix}
1 & 0 & 0 & S \\
0 & 1 & 0 & S \\
0 & 0 & 1 & S \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & S \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & S \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & S \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & S \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X \\
X & X & X & X & X & X & X & X
• Continue iterating the algorithm: zero the block below the diagonal, Gauss-Jordan the diagonal block, store the entries. Make sure to keep track of RHS. At the end, the transformed matrix is

$$\begin{pmatrix}
1 & 0 & 0 & S \\
0 & 1 & 0 & S \\
0 & 0 & 1 & S \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & S \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & S \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & S \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & S \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & S \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & S \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

and the problem is ready for back-substitution.
Note that

- Each of the main blocks is dense: this means that they take a memory space of $I \times I \times 8$ bytes $= 18$MB (for $I = 1,5000$ in double-precision). Reading all of the non-zero entries of the matrix in memory in one go can be impossible (cf. in my high-resolution case).

- The largest stored array contains the entries $S$ (which must be stored before back-substitution). This array is of size $(I \times k_B \times N \times 8)$GB for double-precision.

- The pros of the algorithm:
  - Pivoting makes the algorithm stable.
  - The matrix doesn’t need to be read ahead of time. Each block-line can be read just before it is manipulated.

- The cons of the algorithm:
  - It is inherently serial (cannot be parallelized as such)
  - If the largest-stored array is larger than available memory, huge loss of performance (cf. in my code, for high resolution $I \times k_B \times N \times 8 = 16$GB $> most$ processor memory).
Parallel solutions of block-tridiagonal systems (see Golub & Ortega for example)
Let’s first consider a simple tridiagonal system, and write out the equations explicitly (let’s assume for example there are $N$ equations, where $N$ is odd):

\[
\begin{align*}
a_{11}x_1 + a_{12}x_2 &= b_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\
a_{32}x_2 + a_{33}x_3 + a_{34}x_4 &= b_3 \\
a_{43}x_3 + a_{44}x_4 + a_{45}x_5 &= b_4 \\
&\quad\cdots
\end{align*}
\]

\[
\begin{align*}
a_{N-1,N-2}x_{N-2} + a_{N-1,N-1}x_{N-1} + a_{N-1,N}x_N &= b_{N-1} \\
a_{N,N-1}x_{N-1} + a_{NN}x_N &= b_N
\end{align*}
\]

The standard Gaussian Elimination consists in solving for $x_1$ in the first equation, substituting into the second one, then solve for $x_2$, then substitute that into the third one, etc...

Instead Gaussian Elimination, let’s perform a cyclic reduction. First, let’s take all odd equations and write all of the odd variables in terms of the even ones:

\[
\begin{align*}
x_1 &= \frac{b_1 - a_{12}x_2}{a_{11}} \\
x_3 &= \frac{b_3 - a_{32}x_2 - a_{34}x_4}{a_{33}} \\
&\quad\cdots
\end{align*}
\]

\[
x_N = \frac{b_N - a_{N,N-1}x_{N-1}}{a_{N,N}}
\]

Then if we substitute these into the remaining even equations we get a new system of equations for the even variables only $\{x_2, x_4, \ldots, x_{N-1}\}$. 

25
In terms of matrix representation, these operations can be viewed as:

\[
\begin{pmatrix}
X & X \\
X & X & X \\
X & X & X & X \\
X & X & X & X & X \\
X & X & X & X & X \\
X & X & X & X & X \\
X & X & X & X & X \\
X & X & X \\
X & X
\end{pmatrix}
\Rightarrow
\begin{pmatrix}
1 & S \\
0 & X & 0 & X \\
S & 1 & S \\
X & 0 & X & 0 & X \\
S & 1 & S \\
X & 0 & X & 0 & X \\
S & 1 & S \\
X & 0 & X & 0 \\
S & 1
\end{pmatrix}
\]

(with the equivalent operations on the RHS).

In the example above, the original \((9 \times 9)\) system has been reduced to a \((4 \times 4)\) system.

The idea is to continue reducing the size of the system in the same way: in each cycle roughly a half of the variables are eliminated.
If one starts with a system that is exactly of size \((2^N - 1, 2^N - 1)\), the cyclic reduction can be done exactly \(N - 1\) times to yield the central variable \(x_{2^{N-1}}\), after which back-substitution can start:

**Example:**

- Start with \(N = 4\), 15 variables:
  \[ \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}\}. \]

- First cycle of reduction, down to 7 variables:
  \[ \{x_2, x_4, x_6, x_8, x_{10}, x_{12}, x_{14}\}. \]

- Second cycle of reduction, down to 3 variables:
  \[ \{x_4, x_8, x_{12}\}. \]

- Third cycle of reduction, down to 1 variable:
  \[ \{x_8\}. \]

- Start cycle of backsubstitution: solve for \(x_8\)

- First cycle of backsubstitution: solve for \(\{x_4, x_{12}\}\)

- Second cycle of backsubstitution: solve for \(\{x_2, x_6, x_{10}, x_{14}\}\).

- Third cycle of backsubstitution: solve for all remaining variables.
Of course, exactly the same technique can apply for block-tridiagonal system:

- the operation (for example)

\[ x_1 = \frac{b_1 - a_{12}x_2}{a_{11}} \]

is merely replaced by the matrix operation

\[ X_1 = A_{11}^{-1} (B_1 - A_{12}X_2) \]

so that the matrix-view of the first cycle of reduction becomes

\[
\begin{pmatrix}
I & S \\
0 & X & 0 & X \\
S & I & S \\
X & 0 & X & 0 & X \\
S & I & S \\
X & 0 & X & 0 & X \\
S & I & S \\
X & 0 & X & 0 & X \\
S & I
\end{pmatrix}
\]

- After the last reduction cycle only 1 block is left containing a linear system for the “central variables” (recall, if we start with \(2^N - 1\) variables, \(c = 2^{N-1}\))

\[ \tilde{A}_{cc}X_c = \tilde{B}_c \]

That last linear system can be solved for the vector \(X_c\), and backsubstitution begins.
The advantage is that within one cycle, all operations are **local**, i.e. they can all be done at the same time, and only require information from 3 lines (or block-lines) at a time.

**Cyclic reduction is not a serial process, and can be done with minimal inter-processor communication.**
Inter-processor communication is only necessary for the block-lines at the boundaries between the processors.
A typical algorithm works in the following way

- performs as many rounds of reductions as necessary until there are only 1 block-line to manipulate left in each processor.

- the reduction then continues across processors. At each round half of the processors stop working until the central block-solution is found in the central processor.

- then back-substitution begins.
An example of the work flow, for a \((127 \times 127)\) block-tridiagonal system, working on 8 processors would be

<table>
<thead>
<tr>
<th>(P_1)</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_4)</th>
<th>(x_8)</th>
<th>(x_8)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(x_4)</td>
<td>(x_8)</td>
<td>(x_{12})</td>
<td>(x_{16})</td>
<td>(x_{16})</td>
</tr>
<tr>
<td>(x_{16})</td>
<td>(x_{16})</td>
<td>(x_{16})</td>
<td>(x_{16})</td>
<td>(x_{16})</td>
<td>(x_{16})</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(P_2)</th>
<th>(x_{17})</th>
<th>(x_{18})</th>
<th>(x_{20})</th>
<th>(x_{24})</th>
<th>(x_{24})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(x_{20})</td>
<td>(x_{24})</td>
<td>(x_{28})</td>
<td>(x_{32})</td>
<td>(x_{32})</td>
</tr>
<tr>
<td>(x_{32})</td>
<td>(x_{32})</td>
<td>(x_{32})</td>
<td>(x_{32})</td>
<td>(x_{32})</td>
<td>(x_{32})</td>
</tr>
</tbody>
</table>

| \(P_3\) | \(x_{33}\) | \(x_{34}\) | \(x_{36}\) |
|---------|---------|---------|
|         | \(x_{36}\) | \(x_{40}\) | \(x_{40}\) |
| \(x_{48}\) | \(x_{48}\) | \(x_{48}\) | \(x_{48}\) |

| \(P_4\) | \(x_{49}\) | \(x_{50}\) | \(x_{52}\) |
|---------|---------|---------|
|         | \(x_{52}\) | \(x_{56}\) | \(x_{56}\) |
| \(x_{64}\) | \(x_{64}\) | \(x_{64}\) | \(x_{64}\) |

| \(P_5\) | \(x_{65}\) | \(x_{66}\) | \(x_{68}\) |
|---------|---------|---------|
|         | \(x_{68}\) | \(x_{72}\) | \(x_{72}\) |
| \(x_{80}\) | \(x_{80}\) | \(x_{80}\) | \(x_{80}\) |

| \(P_6\) | \(x_{81}\) | \(x_{82}\) | \(x_{84}\) |
|---------|---------|---------|
|         | \(x_{84}\) | \(x_{88}\) | \(x_{88}\) |
| \(x_{96}\) | \(x_{96}\) | \(x_{96}\) | \(x_{96}\) |

| \(P_7\) | \(x_{97}\) | \(x_{98}\) | \(x_{100}\) |
|---------|---------|---------|
|         | \(x_{100}\) | \(x_{104}\) | \(x_{104}\) |
| \(x_{112}\) | \(x_{112}\) | \(x_{112}\) | \(x_{112}\) |

| \(P_8\) | \(x_{113}\) | \(x_{114}\) | \(x_{116}\) |
|---------|---------|---------|
|         | \(x_{116}\) | \(x_{120}\) | \(x_{120}\) |
| \(x_{127}\) | \(x_{126}\) | \(x_{124}\) | \(x_{124}\) |
Memory & Scaling

(1) Memory requirements:

- In this algorithm, all of the block-lines in one processor have to be read before the start of the back-substitution.

- A similar number of block-lines must be stored for later back-substitution

- This implies that the total memory requirement per processor is $O(I \times I \times (N/N_P) \times 8)$GB, where $N_P$ is the number of processors.

(2) Scaling performance:

- At the $k^{th}$ reduction cycle, $2^{N-k}$ blocks are inverted.

- At the $k^{th}$ backsubstitution cycle, $2^k$ blocks back-substituted.

- This implies that the longest stage of the algorithm are the first reduction cycle and the last backsubstitution cycle.

- Though processors are idle in the later part of the reduction/first part of the backsubstitution, that stage lasts a very small percentage of the time it takes for the first cycle → expect the algorithm to scale well, with minimal inter-communication, provided the number of processors is (much) smaller than the total number of block-lines.
Three catches:

Catch # 1:

The total number of FP operations can be shown to be roughly twice that of serial Gaussian elimination.

→ Maximum speedup for cyclic reduction compared with Gaussian Elimination is $p/2$ for $p$ processors.

Catch # 2:

Optimal performance is only achieved for a set number of block-lines (and number of processors).

→ This limits the choice of the number of mesh-points for two-point BVP.

Catch # 3 (the biggy):

It is not possible to pivot the whole matrix properly using cyclic reduction (only individual blocks can be pivoted).

→ Cyclic reduction can be unstable.

→ This is a notorious problem for two-point boundary value problems containing “diffusion” operators.
The sweeping reduction

To go beyond the intrinsic stability problem of cyclic reduction, we try to combine the idea of the serial and the parallel algorithm:

→ Perform a sweeping pivoted Gaussian Elimination within each individual processor.
Step 1: By performing a sweeping Gaussian Elimination from the top block-line in each processor down to the last, the block-tridiagonal matrix is first transformed into the following form (example of 12 block-lines split between three processors):

\[
\begin{bmatrix}
I & S \\
0 & I & S \\
0 & 0 & I & S \\
0 & 0 & I & X \\
S & I & S \\
S & 0 & I & S \\
S & 0 & 0 & I & S \\
X & 0 & 0 & I & X \\
S & I & S \\
S & 0 & I & S \\
S & 0 & 0 & I & S \\
X & 0 & 0 & I & X \\
\end{bmatrix}
\]
**Step 2:** At the end of the sweeping step, each process sends its very last block-row to the following process. Thus the blocks are redistributed as

$$
\begin{bmatrix}
I & S \\
0 & I & S \\
0 & I & S \\
\end{bmatrix}
\begin{bmatrix}
0 & I & X \\
S & I & S \\
S & 0 & I & S \\
0 & I & S \\
\end{bmatrix}
\begin{bmatrix}
X & 0 & I & X \\
S & I & S \\
S & 0 & I & S \\
0 & I & S \\
X & 0 & I \\
\end{bmatrix}
$$
**Step 3:** Each process can then continue eliminating undesirable variables from its top block-row by Gaussian elimination with the successive block-rows below, until the following form is achieved:

\[
\begin{bmatrix}
I & S \\
0 & I & S \\
0 & 0 & I & S \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & X & 0 & 0 & 0 & X \\
S & I & S \\
S & 0 & I & S \\
S & 0 & 0 & I & S \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
X & 0 & X & 0 & 0 & 0 & X \\
S & I & S \\
S & 0 & I & S \\
S & 0 & I & S \\
X & 0 & 0 & I \\
\end{bmatrix}
\]

At this point, the top-rows in each process can be combined into a much-reduced block tri-diagonal system. Solution from here on proceeds with a cyclic reduction of the remaining blocks across processes, followed by a back-substitution step.
What have we gained?

- **Stability:** through pivoting.

- **Speedup:** there is only a *single* step of inter-processor communication before the cross-processor reduction starts (instead of $k$ stages in the original algorithm) and similarly after the cross-processor back-substitution has ended.

- **Versatility:** the number of meshpoints is not constrained anymore and it is easy to load-balance the number of block-lines per processor.

Did we lose anything? **NO!**

- It’s easy to show that the total number of FP Ops is exactly the same as cyclic reduction

**YAY!**
Performance analysis

The following plot shows the time taken for 1 iteration of the NRK algorithm, working on 900 simultaneous coupled nonlinear ODEs for 2000 meshpoints (diamonds) and for 1000 meshpoints (stars).
Pretty pictures...