Implementation direct solver to allow for better preconditioners in iterative solver

D. Pouw

Supervisors:

C. Vuik
A. van der Ploeg

March 11, 2021

Contents

1 Introduction **4**

2 Composition water flow model **6**
2.1 Physics 6
2.1.1 Navier-Stokes 6
2.1.2 Reynolds decomposition 7
2.1.3 Eddy viscosity 7
2.1.4 Turbulence modeling: Menter turbulence model 8
2.1.5 Turbulence modeling: $k-\omega$ SST Turbulence model . . . 9
2.1.6 Summary 10
2.2 Computational decisions 11
2.2.1 Grids 11
2.2.2 Boundary conditions 12
2.2.3 Discretization 13
2.2.4 Solution procedure 13
2.2.5 Linear system solver and preconditioning 15

3 Analysis of iterative solver PARNASSOS **16**
3.1 GMRES and Preconditioning 16
3.1.1 GMRES: basic algorithm 17
3.1.2 GMRES(m): Restarting GMRES 18
3.1.3 Properties of GMRES 18
3.1.4 Preconditioning 20
3.1.5 Right- and left preconditioned GMRES 20
3.1.6 Original choice preconditioning PARNASSOS 21
3.2 Data generation and storage 23
3.3 Overview solver 25

4 Main goal of the research **26**
4.1 Tests 26
4.2 Test original options preconditioner PARNASSOS 28
4.2.1 Test: 1blockdiag 28
4.2.2 Test: 3blockdiag 28
4.2.3 Test: 7blockdiag 29
4.3 Results 30

5 Complete LU decomposition of the main diagonal blocks **31**
5.1 Theory 31
5.2 Implementation 33
5.3 Scaling the main diagonal blocks of A 34
5.4 Reordering rows and columns to reduce bandwidth 35
5.5 Tests 36

6 $ILU(N)$ decomposition **37**
6.1 Theory . . . 37
6.2 Tests . . . 40
6.3 Results . . . 43

7 Experiment with other direct solver **44**
7.1 Setup idea . . . 44
7.2 Explanation algorithm . . . 45
7.3 Theory . . . 47
7.3.1 Form G_j matrices . . . 47
7.3.2 Computational complexity band matrix multiplications . 48
7.3.3 Computational complexity direct solver . . . 49
7.4 Practice . . . 52

8 Final comparison **54**

9 Summary **59**

10 Propositions for future endeavors **60**

11 Bibliography **61**

12 Appendices **62**
12.1 Appendix A: Starting values tests . . . 62
12.2 Appendix B: Conversion into band matrix . . . 63

1 Introduction

The Netherlands is well known as a trading country with the port in Rotterdam being Europe's largest sea port. The ships that arrive and depart daily sail through the north sea from and to all kinds of ports around the globe. For such ships, decreasing travel time means earning more money. This results in a heavy incentive to have a ship which sails through the waters with much more ease than a ship which does not make use of the water flow.
The research institute MARIN, Maritime Research Institute Netherlands, specializes in creating cleaner, safer and smarter ships which make sustainable use of the sea. At MARIN, the viscous-flow solver PARNASSOS is frequently used to compute the flow of water around ships sailing in still water, without in-coming waves. The ship-generated waves can be computed in order to determine how the form of the ship improves or deters its own travel speed and comfort. With this in mind, PARNASSOS can be used as an analysis tool in ship hull optimization projects. Here follows an example of the water flow around a vessel in PARNASSOS:

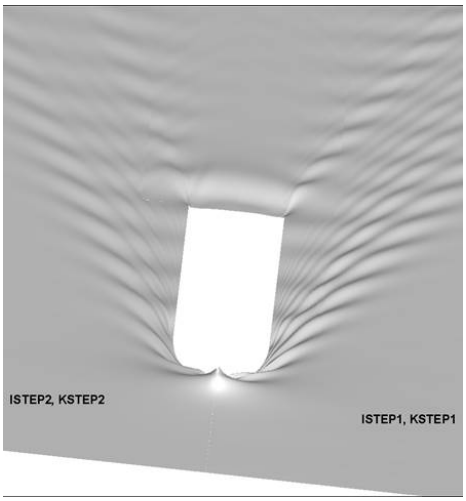
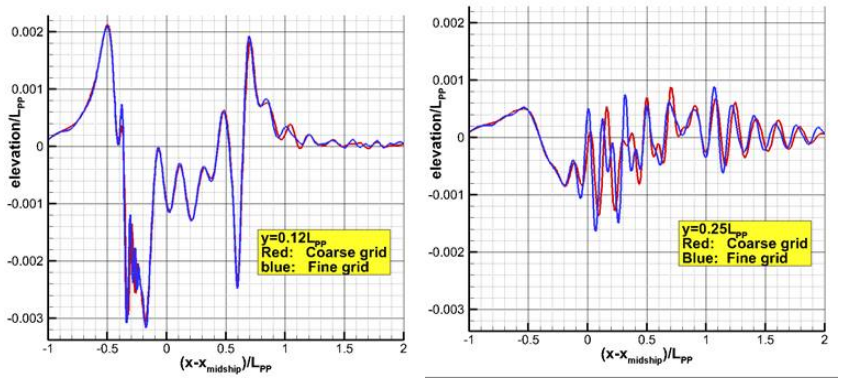


Figure 1: Top view of water flow

One of the aspects of PARNASSOS is the trade-off when choosing how many grid points are necessary for a simulation. When more grid points are needed, the calculation time for the water flow goes up. However, most of the times, a lot of grid points are needed for a more detailed simulation of the water flow. An example is the water flow for the ship above. Both images in Fig. 2 show the wave pattern of the water surface created by the ship, where the blue line comes from having fewer grid points and the red line comes from having more grid points. The left image shows the waves that are closer to the ship and the image on the right shows the waves that are further away from the ship.
It can be seen that having more grid points gives a significant difference in



(a) Water surface close to ship (b) Water surface far from ship.

Figure 2: Wave pattern of the water surface created by the ship

computed wave patterns, especially when going further away from the ship. To counter the fact that more grid points are needed for a more detailed simulation, the program has to work faster in order to keep the time to get the water flow in check. This paper will be about a way to try to improve the part of the program that solves linear equations such that those actions take less time.

The next two main sections will discuss some of the important parts of how PARNASSOS is set up. The first part are the equations behind the physics of water flow. The second main part consists of the grids and how they are chosen to be made, the boundary conditions of the model, the discretizations of the physical equations to make them suitable for the solver and the solution procedure of PARNASSOS.

After the foundation has been laid, the way to improve the solver can be explained in section 4. This section also includes the tests that are done to compare the new options with the previous ones, with the previous options being immediately put to the test.

Sections 5,6 and 7 will discuss the new options for the solver in PARNASSOS.

Finally, all results will be mentioned in a small summary at the end.

2 Composition water flow model

2.1 Physics

In order to correctly model a natural phenomenon, one must know the physical interactions that are significant and the equations that simulate those interactions. For this reason, the next subsections will explain the equations that model water flow and what choices are made to get them.

2.1.1 Navier-Stokes

It is at first assumed for the water flow that the water is an Newtonian fluid. One of the equations in the Navier-Stokes equations is the continuity equation for fluid flow:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0$$

where ρ is the fluid density, t is time and $\mathbf{u}$ is the flow velocity. The flow is incompressible, which means ρ is constant in space and time. This results, when $\rho \neq 0$, in the above equation simplifying to

$$\nabla \cdot \mathbf{u} = 0.$$

After combining the above equation with other equations, the Navier Stokes equations become

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f}$$

where p is the pressure and $\mathbf{f}$ are the external forces. When the variables are transformed to be dimensionless, two constants appear in the equations: The Reynolds number appears in front of the $\nabla^2 \mathbf{u}$ term and the Froude number appears in front of the $\mathbf{f}$ term in the equations:

$$\frac{\partial \mathbf{u}^*}{\partial t^*} + (\mathbf{u}^* \cdot \nabla^*)\mathbf{u}^* = -\nabla^* p^* + \frac{1}{Re}\nabla^{*2}\mathbf{u}^* + \frac{1}{Fr^2}\mathbf{f}^*$$

The Reynolds number, Re, is a dimensionless number defined as

$$Re = \frac{LV_s}{\nu}$$

where $\nu = \frac{\mu}{\rho}$ is the dynamic viscosity of the fluid, L is the characteristic linear dimension of the system and V_s is a characteristic velocity. L can, for example, be the length of the ship from bow to stern and V_s can be the average speed of the ship.

The next assumption made for the water flow is that the flow is stationary, which means that the velocity won't change over time:

$$\frac{\partial \mathbf{u}}{\partial t} = 0 \Leftrightarrow \frac{\partial \mathbf{u}^*}{\partial t^*} = 0$$

Substituting the equality into the Navier-Stokes equations results in the following equations:

$$(\mathbf{u}^* \cdot \nabla^*)\mathbf{u}^* = -\nabla^* p^* + \frac{1}{Re}\nabla^{*2}\mathbf{u}^* + \frac{1}{Fr^2}\mathbf{f}^*$$

If the Reynolds number is large, turbulent flow comes more into play and the equations are more difficult to solve in the regular ways of modeling water flow. One of the approaches is using the Reynolds decomposition.

2.1.2 Reynolds decomposition

The Reynolds decomposition is a technique that splits the pressure p into a time averaged component $\bar{p}$ and a fluctuating component p'. The same is done for each component of the velocity vector $\mathbf{u}$.
For a stationary flow of an incompressible Newtonian fluid, we get the equations [1] (in Cartesian coordinates), where the Einstein summation convention applies to repeated indices:

$$\bar{u_j}\frac{\partial \bar{u_i}}{\partial x_j} = \bar{f_i} - \frac{\partial \bar{p}}{\partial x_i} + \mu\frac{\partial^2 \bar{u_i}}{\partial x_j \partial x_j} - \frac{\partial}{\partial x_j}\left(\overline{u_i' u_j'}\right)$$

where f_i are external forces and μ is the fluid kinematic viscosity. These equations are called the Reynolds-Averaged Navier-Stokes (RANS) equations and the term $\overline{u_i' u_j'}$ in these equations is usually difficult to model.

2.1.3 Eddy viscosity

The term $\overline{u_i' u_j'}$ is called the Reynolds-stress. One of the first ways to model this term is with the concept of eddy viscosity[2]:

$$\overline{u_i' u_j'} = \frac{1}{3}\overline{u_i' u_i'}\delta_{ij} - \nu_t\left(\frac{\partial \bar{u_i}}{\partial x_j} + \frac{\partial \bar{u_j}}{\partial x_i}\right)$$

where ν_t is the eddy viscosity and δ_{ij} is the Kronecker delta function:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

[1] Equations are from section 2.1 of Giancarlo (2009)
[2] Equations are from section 3.1 of Giancarlo (2009)

There are more possible ways to model this term, a few of which are in the paper from Eça & Hoekstra (2000). Two of these are described in the next section.

2.1.4 Turbulence modeling: Menter turbulence model

In PARNASSOS, various turbulence models can be chosen. The variants are described in the paper from Eça & Hoekstra (2000). The turbulence model that is mostly used is the one-equation model from Menter. On occasion, the two equation model $k-\omega$ SST is used. The equations for the following two models are for the dimensionless case.

The one equation model of Menter is as follows:

It comes from the $k-\epsilon$ equation:

$$\frac{\partial(\tilde{\nu_t})}{\partial t}+\sum_{j=1}^{3}\frac{\partial(u_j\tilde{\nu_t})}{\partial x_j}=c_1D_1\tilde{\nu_t}\sqrt{S}+\sum_{j=1}^{3}\frac{\partial}{\partial x_j}\left[(\mu+\sigma\tilde{\nu_t})\frac{\partial\tilde{\nu_t}}{\partial x_j}\right]-c_2E_{1e}$$

The eddy-viscosity in the one-equation model of Menter is as follows:

$$\nu_t=D_2\tilde{\nu_t}$$

with the equations:

$$\begin{aligned}
D_1 &= \frac{\nu_t+\mu}{\tilde{\nu_t}+\mu}\\
E_{1e} &= c_3E_{BB}\tanh\left(\frac{E_{k-\epsilon}}{c_3E_{BB}}\right)\\
D_2 &= 1-\exp\left(-\left(\frac{\tilde{\nu_t}}{A\kappa\mu}\right)^2\right)\\
E_{k-\epsilon} &= {\tilde{\nu_t}}^2\left(\frac{\nabla\sqrt{S}\cdot\nabla\sqrt{S}}{S}\right)\\
E_{BB} &= \nabla\tilde{\nu_t}\cdot\nabla\tilde{\nu_t}
\end{aligned}$$

where S is the strain rate squared and μ is the viscosity of the fluid.

The equation for γ is:

$$\gamma=\frac{\beta}{\beta^*}-\frac{\sigma\kappa^2}{\sqrt{\beta^*}}$$

The model constants are:

$$\begin{array}{ccccc}
c_1=0.144 & , & c_2=1.862 & , & c_3=7\\
\kappa=0.41 & , & \sigma=1 & , & A=13
\end{array}$$

2.1.5 Turbulence modeling: $k-\omega$ SST Turbulence model

In this section, we discuss the $k-\omega$ Shear Stress Transport (SST) turbulence model. The eddy viscosity is defined as follows:

$$\nu_t = \frac{k}{\omega}$$

where k is the turbulent kinetic energy and ω the specific rate of dissipation of the eddies. The quantities k and ω are obtained from the solution of the transport equations

$$\frac{\partial k}{\partial t} + \sum_{j=1}^{3} \frac{\partial (u_j k)}{\partial x_j} = \nu_t S - \beta^* \omega k + \sum_{j=1}^{3} \frac{\partial}{\partial x_j} \left[(\mu + \sigma_k \nu_t) \frac{\partial k}{\partial x_j} \right]$$

and

$$\frac{\partial \omega}{\partial t} + \sum_{j=1}^{3} \frac{\partial (u_j \omega)}{\partial x_j} = \alpha S - \beta \omega^2$$

$$+ \sum_{j=1}^{3} \frac{\partial}{\partial x_j} \left[(\mu + \sigma_\omega \nu_t) \frac{\partial \omega}{\partial x_j} \right] + 2(1 - F_1) \frac{\sigma_{\omega 2}}{\omega} \left(\sum_{j=1}^{3} \frac{\partial k}{\partial x_j} \frac{\partial \omega}{\partial x_j} \right)$$

where S is the strain rate squared. The equation for γ is:

$$\gamma = \frac{\beta}{\beta^*} - \frac{\sigma_\omega \kappa^2}{\sqrt{\beta^*}}$$

The ω equation can be integrated up to the wall. The model constants, which are the constants in the equations without number in subscript and are denoted by ϕ, are obtained from

$$\phi = F_1 \phi_1 + (1 - F_1) \phi_2$$

where the model constants for ϕ_1 are:

$$\begin{array}{ccc} \sigma_{k1} = 0.5 & , & \sigma_{\omega 1} = 0.5 \\ \beta^* = 0.09 & , & \beta_1 = 0.075 \\ \kappa = 0.41 & , & \alpha_1 = 0.5532 \end{array}$$

and the model constants for ϕ_2 are:

$$\begin{array}{ccc} \sigma_{k2} = 1 & , & \sigma_{\omega 2} = 0.826 \\ \beta^* = 0.09 & , & \beta_2 = 0.0828 \\ \kappa = 0.41 & , & \alpha_2 = 0.4404 \end{array}$$

The blending function F_1 is given by

$$F_1 = \tanh(arg^4)$$

with

$$arg = \min\left[\max\left(\frac{\sqrt{k}}{\beta^*\omega d}, \frac{500\mu}{d^2\omega}\right), \frac{4\sigma_{\omega 2}k}{CD_{k\omega}d^2}\right]$$

and

$$CD_{k\omega} = \max\left(2\sigma_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j}, 10^{-20}\right)$$

where d is the distance to the closest surface.

2.1.6 Summary

In the RANS equations,

$$\bar{u_j}\frac{\partial \bar{u_i}}{\partial x_j} = \bar{f}_i - \frac{\partial \bar{p}}{\partial x_i} + \mu\frac{\partial^2 \bar{u_i}}{\partial x_j \partial x_j} - \frac{\partial}{\partial x_j}\left(\overline{u_i' u_j'}\right)$$

, the difficult to model the Reynolds-stress $\overline{u_i' u_j'}$ will be modeled, see section 2.3. The turbulence model in section 2.4 will be used to model the eddy viscosity with the boundary conditions from section 2.2.2. Then with the continuity equations,

$$\nabla \cdot \mathbf{u} = 0,$$

the flow of water around an object can be modeled.

2.2 Computational decisions

Now that the equations to model water flow are known, the next step is to choose how to approximate the solution to the equations with the current boundary conditions. In the following subsections, the grids, the solution procedure and the discretizations will be explained.

2.2.1 Grids

There are two grids: the physical grid and the computational grid.

The directions in the physical grid are the mainstream direction of the flow of water, ξ ,the direction which is perpendicular to the surface of the ship, η , and the last direction is the one perpendicular to both of the two defined first, ζ. The physical grid is assumed to be body-fitted to the hull of the ship. The following picture shows the physical grid on the starboard side of a ship, where the grid of a constant ξ plane is shown for three different values of ξ.

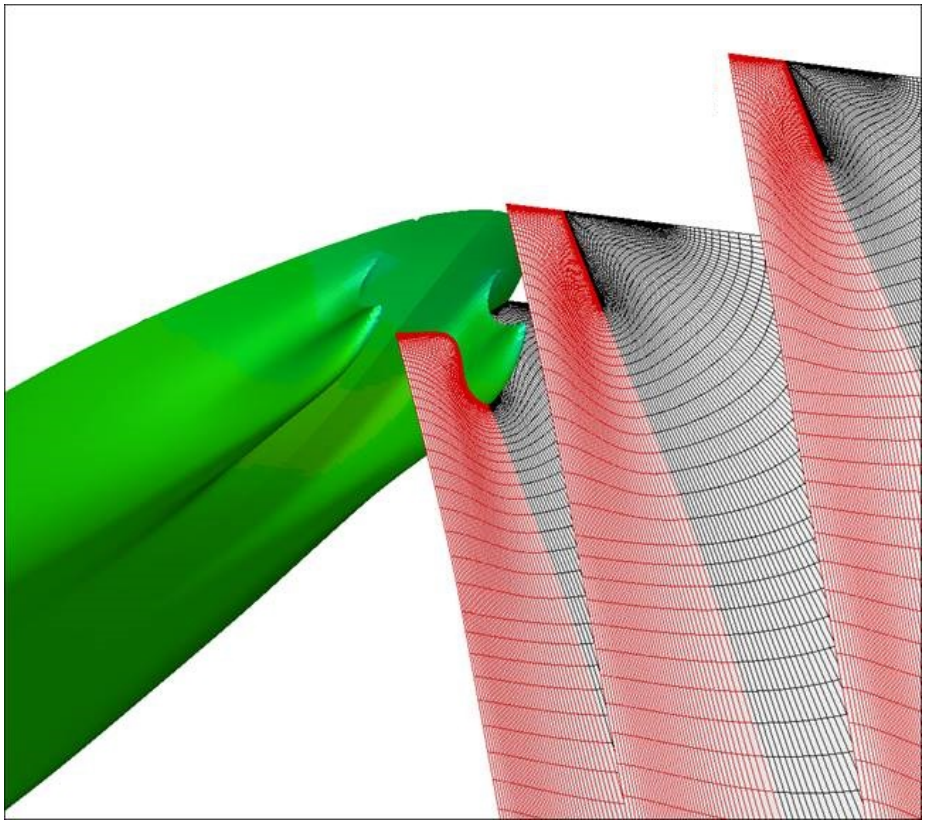
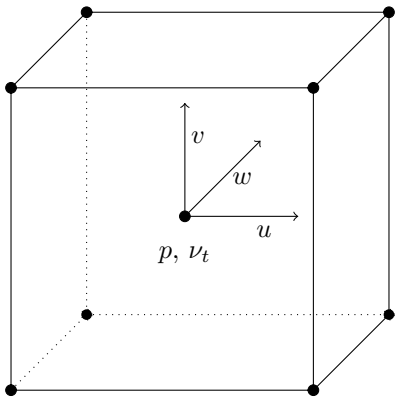


Figure 3: The physical grid for a ship for certain constant planes ξ

Because the ship is symmetric, only the starboard side of the grid needs to be calculated.
The 3D computational grid has points with the coordinates i, j and k which correspond respectively to the directions in the physical grid ξ, η and ζ.



In each point on the grid, we have the following collocated unknowns: The velocities in the directions i, j and k, which are respectively u, v and w, the eddy viscosity in that point, ν_t, and the pressure in that point, p are considered as unknowns.

2.2.2 Boundary conditions

To model the equations for a certain ship, boundary conditions are necessary. An important constant to repeat for the next part is the length of the ship, L. These boundary conditions are for the most part described in the paper of van de Ploeg, Starke & Veldhuis (2013). To describe the boundary conditions, let there be an (x, y, z)-co-ordinate system fixed to the ship with the direction of x positive going towards the aft and the direction of z upward into the sky.

The undisturbed water surface is located at $z = 0$. If the ship's generated waves are not taken into account at $z = 0$, symmetry conditions are imposed. Otherwise, free-surface boundary conditions, as described in the paper of van de Ploeg, Starke & Veldhuis (2013), are imposed.

On the hull of the ship, there are Dirichlet conditions for the velocity components. This follows from the assumption that there is no slip on the hull and that there is no water flowing through the hull.

The inflow boundary is located $0.5L$ in front of the bow and the outflow boundary at $1.5L$ behind the transom. Only the starboard side needs to be taken into account, because of symmetry conditions on the boat itself.

The lateral outer boundary has the form of a quarter of a cylinder with axis $y = 0$, $z = 0$ and radius $1.0L$. At this boundary, tangential velocities and pressure found from a potential-flow computation are imposed. Since that computation already gives for much of the wave pattern good results, these boundary conditions hardly cause any wave reflection, although they are of the Dirichlet type.

The starting conditions are taken as follows: If a one-equation turbulence model is used, then the eddy viscosity on the hull of the ship and on the elliptical boundary is zero. On the other boundaries, similar kind of starting conditions are used in the turbulence models as for mass and momentum equations. For example: everywhere the velocity of the water equal to the average speed of the ship, V_s.

2.2.3 Discretization

To get a good discrete version of the continuous equations, the following choices were made:

- For the numerical evaluation of the grid metric terms, second-order central difference schemes are used
- In the continuity equation, a second-order three-point backward scheme is used for the mainstream derivative and a third-order four-point scheme with a fixed bias in normal and girthwise direction.
- For the derivatives of the velocities that occur in the convective terms, a second-order upwind scheme is used in the streamwise direction and a third-order upwind scheme for the normal and girth-wise direction.
- The direction of the discretization of the pressure gradient, in the momentum conservation equations, is taken in the opposite direction of the direction of the discretization of the pressure gradients, in the continuity equation. This prevents the even-odd decoupling of the pressure.
- All second derivatives in the diffusive terms are discretized by second-order central-difference schemes.

2.2.4 Solution procedure

PARNASSOS uses a marching solution scheme with the planes being roughly perpendicular to the mainstream direction of the flow; This means that the program will work with constant planes, which is the set of points in the physical grid for which ξ is constant, in the direction of the flow of the water. Once updating variables in downstream direction and once updating variables in upstream direction will be called a global iteration step.

In short, the one global iteration step goes as follows:

1. Bow to stern; update per constant plane:
   - First eddy viscosity in each point of the calculation grid
   - Then the velocities and the pressure in each point of the calculation grid
2. Stern to bow; update per constant plane: the pressure in each point of the calculation

One global iteration step will be explained in more detail:
First in each plane, the eddy viscosity will be approximated in every point using the velocity and pressure calculated previously, after which a linear equation is solved to update the velocity and pressure in every point. To be more specific about the calculation, the details are given below.

Let the number of grid nodes in the stream, normal and girthwise direction respectively be NX, NY and NZ. The turbulence model equations are treated as uncoupled from the other equations. The complete discretization can be written as

$$F(u_{ijk}, v_{ijk}, w_{ijk}, p_{ijk}, (\nu_t)_{ijk}) = 0 \tag{1}$$

where F is a vector-valued function with $5 \times NX \times NY \times NZ$ components. When n is the number of global iterations steps, then the calculations of the components in the next global iteration goes as follows:

1. Compute the eddy viscosity $(\nu_t)^n_{ijk}$ approximately from

   $$F(u^{n-1}_{ijk}, v^{n-1}_{ijk}, w^{n-1}_{ijk}, p^{n-1}_{ijk}, (\nu_t)^n_{ijk}) = 0$$

   If a one- or two-equation turbulence model is used, the transport equation(s) are still solved using a plane-by-plane strategy
2. Linearize the RANS-equations, using $(\nu_t)^n_{ijk}$, and compute u^n_{ijk}, v^n_{ijk}, w^n_{ijk} and p^n_{ijk} by solving the resulting linear system.

This calculation will be repeated in every plane from the inflow boundary to the outflow boundary. Next, the method is going through the planes from the stern of the ship to the bow updating the pressure in each point of the plane.

It is possible to solve the variables of multiple planes simultaneously in each global iteration, as described in the paper from van der Ploeg, Eça & Hoekstra (2000) in section 3.1. Let the amount of consecutive grid-planes be g. The resulting coefficient matrix will be g times larger than with only taking one plane each time and contains terms that describe the coupling between all g planes.

Another possibility is to take less grid points in the girthwise direction. The reduction of the amount of grid points results in the ability of faster finding an approximation of the flow around the ship. This approximation can afterwards

be improved when it is used as a starting point with the grid points in the girthwise direction again being NZ.
The value for the reduced amount of grid points in the girthwise direction is named ubk.

2.2.5 Linear system solver and preconditioning

The linear system that is going to be described is the one from step 2 in the global iteration from the previous subsection, which comes from the RANS-equations. The matrix, that contains the coefficients which account for the coupling between the separate ξ = constant planes, is denoted as A. Let the entries of A be grouped in blocks so that all elements multiplying the variables in a ξ = constant plane form a block. Such a block is of size $4 \times NY \times ubk$ and will be represented by one entry $A_{i,l}$, in which i an l are row and column indices.
A has the following penta-diagonal structure following the choices for the discretization of the equations:

$$A = \begin{bmatrix} D_1 & E_1 & F_1 & 0 & 0 & 0 & \cdots \\ C_2 & D_2 & E_2 & F_2 & 0 & 0 & \cdots \\ B_3 & C_3 & D_3 & E_3 & F_3 & 0 & \cdots \\ 0 & B_4 & C_4 & D_4 & E_4 & F_4 & \cdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & 0 & B_g & C_g & D_g \end{bmatrix}$$

As an example, lets assume you take only two constant planes ($g = 2$). The matrix A will be

$$A = \begin{bmatrix} D_1 & E_1 \\ C_2 & D_2 \end{bmatrix}$$

The block D_1 contains all the numbers that represent the connections between the points in plane 1 after discretization. Same for D_2 with plane 2. The block E_1 and C_2 contain the numbers that represent the connections between the points in plane 1 and the points in plane 2.

The linear system is solved with a preconditioned version of GMRES(m), which will be explained in subsection 3.1.

3 Analysis of iterative solver PARNASSOS

The previous section ended with creating a sub-problem in order to get the simulation for the water flow, which is that the linearized equations need to be solved. In order to better explain how PARNASSOS solves this problem, a more rigorous explanation of an integral part of PARNASSOS is needed: the solve routine preconditioned GMRES(m), which is a more specific version of GMRES. Therefore, the following subjects will be discussed:

- What GMRES and preconditioning is
- How the data that is generated in another part of PARNASSOS is structured and what its origins are from physics
- How GMRES is coded to minimize amount of calculations needed

3.1 GMRES and Preconditioning

Given an $n \times n$ matrix A and $n \times 1$ vector b, finding an $n \times 1$ vector x such that $Ax = b$ is a well known and well researched problem. There are mainly two ways to categorize algorithms which solve such problems: iterative or direct. Direct methods give an exact solution for the problem and iterative methods give an approximate solution for the problem. The problem with direct methods is that when the matrix becomes large, it is much slower to solve the problem with a direct method than an iterative method. Combined with the argument that for most problems approximate solutions are good enough for large problems, the mainly used methods are iterative methods. One of those methods is called GMRES.

3.1.1 GMRES: basic algorithm

For the linear problem $Ax = b$, where A is non-singular, The Krylov subspace $\mathcal{K}_m$ is defined for a vector $x_0 \in \mathbb{R}^n$ in the following way, where $r_0 = b - Ax_0$:

$$\mathcal{K}_m(A, x_0) = span(r_0, Ar_0, \ldots, A^{m-1}r_0)$$

Any vector x in $x_0 + \mathcal{K}_m(A, x_0)$ can be written as

$$x = x_0 + V_m y$$

where y is an m-vector and V_m is an $n \times m$ matrix of orthonormal vectors which form a basis of $\mathcal{K}_m(A, x_0)$. To get the solution for $Ax = b$, find a vector y that minimizes

$$||b - Ax||_2 = ||b - A(x_0 + V_m y)||_2$$

When you have the Hessenberg matrix $\bar{H}_m$ for which holds: $AV_m = V_{m+1}\bar{H}_m$, the problem can be reformulated as finding the y_m which minimizes

$$||\beta e_1 - \bar{H}_m y_m||_2$$

where $\beta = ||b - Ax_0||_2$. With the vector y_m the approximate solution $x_m = x_0 + V_m y_m$ can be calculated. The algorithm for this basic version of GMRES is given in algorithm 1.

Algorithm 1 Basic GMRES

1: *Compute* $r_0 = b - Ax_0, \beta := ||r_0||_2$, *and* $v_1 := r_0/\beta$
2: **for** $j = 1, \ldots, m$ **do**
3: *Compute* $w_j := Av_j$
4: **for** $i = 1, \ldots, j$ **do**
5: $h_{ij} := (w_j, v_i)$
6: $w_j := w_j - h_{ij}v_i$
7: $h_{j+1,j} = ||w_j||_2$.*If* $h_{j+1,j} = 0$ *set* $m := j$ *and go to* 9
8: $v_{j+1} = w_j/h_{j+1,j}$
9: *Define the* $(m+1) \times m$ *Hessenberg matrix* $H = \{h_{ij}\}_{1 \le i \le m+1, 1 \le j \le m}$
10: *Compute the minimizer of* $||\beta e_1 - \bar{H}_m y_m||_2$, y_m, *and* $x_m = x_0 + V_m y_m$.

Subsection 3.1.1 is mainly based on chapter 6.5 on page 172 from Iterative Methods for Sparse Linear Systems (Saad, 2003)

3.1.2 GMRES(m): Restarting GMRES

When m becomes large, the amount of memory used and the computations become large. On way of solving this problem is by 'restarting' the algorithm after a certain amount of steps. After applying algorithm 1 with the starting vector x_0 and fixed m, the vector x_m is a better approximation for the solution of $Ax = b$ than x_0. Therefore, instead of taking x_0 as starting vector, the restarted version takes x_m as the new starting vector. This restarting of GMRES is done until x_m is sufficiently close to the true solution.

3.1.3 Properties of GMRES

Memory: Lets assume that there are maximal p elements in every row of A. At the end of step m in the algorithm, the following elements are in the memory:

- The elements in the matrix A: np elements
- The elements in the $(m+1) \times m$ Hessenberg matrix: $(m^2+m)/2$ elements
- The elements in the vectors v_j and w_j: $2(m+1)n$ elements
- The starting values r_0 and β: $n+1$ elements

Complexity of calculations: The complexity of the calculations is as follows:

- Letting j go from 1 to m:
  - calculating the vector w_j by multiplying the $n \times n$ matrix A with the $n \times 1$ vector v_j: $2np$ flops
  - Letting i go from 1 to j:
    - inner product between w_j and v_i to get h_{ij}: $2n$ flops
    - updating the vector w_j: $2n$ flops
  - subtotal loop over i: $4jn$ flops
  - Taking the 2-norm of w_j: $2n$ flops
  - calculating v_{j+1}: n flops
- subtotal loop over j: $2mnp + 2(m^2+m)n + 3mn$ flops
- Computing the minimizer y_m: $(m+1)m$ flops
- Computing x_m: $mn + n$ flops

Convergence: There is a corollary known for the convergence of GMRES when the matrix A is diagonalizable [3].

Theorem 1. *Let P_m be all polynomials of degree less than m and let $\sigma = \{\lambda_1, \ldots, \lambda_n\}$ be the eigenvalues of A. Suppose A is diagonalizable so that $A = X\Lambda X^{-1}$, where $\Lambda = diag(\sigma)$ and the columns of X are the eigenvectors of A, and let*

$$\epsilon^{(m)} = \min_{p \in P_m : p(0)=1} \max_{\lambda_i \in \sigma} |p(\lambda_i)|$$

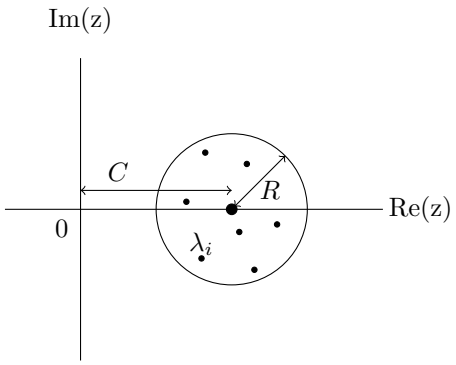
Then the residual norm of the m-th iterate satisfies:

$$||r_m||_2 \leq \kappa_2(X)\epsilon^{(m)}||r_0||_2$$

where $\kappa_2(X) = ||X||_2||X^{-1}||_2$. If furthermore all eigenvalues of A are enclosed in the complex plane in a circle centered at $C \in \mathbb{R}$ with $C > 0$ and having radius R with $C > R$, then

$$\epsilon^{(m)} \leq \left(\frac{R}{C}\right)^m$$

Here is a figure of the complex plane of what is meant with the extra condition to assure the final inequality:



When A is close to being the identity matrix, $C \approx 1$ and $R \approx 0$. This means that the convergence of GMRES is very fast. To change the the linear problem $Ax = b$ in a way that helps with the convergence of GMRES, the idea of preconditioning is well known to work.

[3] Vuik and Lahaye (2017), Scientific Computing (wi4201), Lecture notes TU Delft, Theorem 7.3.1 on page 122

3.1.4 Preconditioning

Let M be an invertible matrix. There are two types of preconditioning techniques: left and right. Left preconditioning is multiplying both sides of the original linear problem $Ax = b$ by the matrix M on the left:

$$M^{-1}Ax = M^{-1}b$$

Because M is invertible, both equations have the same solutions for x. On the other hand, when the matrix M is invertible, there exists a unique vector y such that $y = Mx$:

$$AM^{-1}y = b, \quad y = Mx$$

This is right preconditioning of the original linear problem $Ax = b$.
The matrix AM^{-1} or $M^{-1}A$ can have better properties than the matrix A; for example, M can be made such that the eigenvalues of $M^{-1}A$ are all almost equal to 1 such that GMRES has better convergence with the equation $M^{-1}Ax = M^{-1}b$ than with the equation $Ax = b$. One of the properties that M^{-1} should have is that it must be inexpensive to multiply with a vector.

3.1.5 Right- and left preconditioned GMRES

This version of the GMRES algorithm is used for solving the right preconditioned system:

$$AM^{-1}y = b, \quad x = M^{-1}y$$

Algorithm 2 Right preconditioned GMRES

1: *Compute* $r_0 = b - Ax_0, \beta := ||r_0||_2$, *and* $v_1 := r_0/\beta$
2: **for** $j = 1, \ldots, m$ **do**
3: *Compute* $w_j := AM^{-1}v_j$
4: **for** $i = 1, \ldots, j$ **do**
5: $h_{ij} := (w_j, v_i)$
6: $w_j := w_j - h_{ij}v_i$
7: $h_{j+1,j} = ||w_j||_2$.*If* $h_{j+1,j} = 0$ *set* $m := j$ *and go to* 9
8: $v_{j+1} = w_j/h_{j+1,j}$
9: *Define the* $(m+1) \times m$ *Hessenberg matrix* $H = \{h_{ij}\}_{1 \leq i \leq m+1, 1 \leq j \leq m}$
10: *Compute the minimizer of* $||\beta e_1 - \bar{H}_m y_m||_2$, y_m, *and for* m $x_m = x_0 + M^{-1}V_m y_m$.

The following version of the GMRES algorithm is for solving the left preconditioned system:

$$M^{-1}Ax = M^{-1}b$$

Algorithm 3 Left preconditioned GMRES

1: *Compute* $r_0 = M^{-1}(b - Ax_0), \beta := ||r_0||_2, and\ v_1 := r_0/\beta$
2: **for** $j = 1, \ldots, m$ **do**
3: *Compute* $w_j := M^{-1}Av_j$
4: **for** $i = 1, \ldots, j$ **do**
5: $h_{ij} := (w_j, v_i)$
6: $w_j := w_j - h_{ij}v_i$
7: $h_{j+1,j} = ||w_j||_2$.*If* $h_{j+1,j} = 0$ *set* $m := j$ *and go to* 9
8: $v_{j+1} = w_j/h_{j+1,j}$
9: *Define the* $(m+1) \times m$ *Hessenberg matrix* $H = \{h_{ij}\}_{1 \leq i \leq m+1, 1 \leq j \leq m}$
10: *Compute the minimizer of* $||\beta e_1 - \bar{H_m} y_m||_2$, y_m, *and for* m $x_m = x_0 + V_m y_m$.

3.1.6 Original choice preconditioning PARNASSOS

Let the matrix A be as defined in chapter 2.2.5:

$$A = \begin{bmatrix} D_1 & E_1 & F_1 & 0 & 0 & 0 & \cdots \\ C_2 & D_2 & E_2 & F_2 & 0 & 0 & \cdots \\ B_3 & C_3 & D_3 & E_3 & F_3 & 0 & \cdots \\ 0 & B_4 & C_4 & D_4 & E_4 & F_4 & \cdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & 0 & B_g & C_g & D_g \end{bmatrix}$$

The matrix M is constructed such that it has the block structure

$$M = \begin{bmatrix} M_1 & 0 & 0 & 0 & 0 & 0 & \cdots \\ C_2 & M_2 & 0 & 0 & 0 & 0 & \cdots \\ B_3 & C_3 & M_3 & 0 & 0 & 0 & \cdots \\ 0 & B_4 & C_4 & M_4 & 0 & 0 & \cdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & 0 & B_g & C_g & M_g \end{bmatrix}$$

which will be an approximation of the lower triangular part of the original matrix A:

$$\begin{bmatrix} D_1 & 0 & 0 & 0 & 0 & 0 & \cdots \\ C_2 & D_2 & 0 & 0 & 0 & 0 & \cdots \\ B_3 & C_3 & D_3 & 0 & 0 & 0 & \cdots \\ 0 & B_4 & C_4 & D_4 & 0 & 0 & \cdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & 0 & B_g & C_g & D_g \end{bmatrix}$$

For each i, the approximation M_i is created by first making an incomplete LU decomposition of D_i, which makes M_i of the form $M_i = L_i U_i$. The incomplete

decomposition is constructed on a 4×4 block-level, where every 'entry' in L_i and U_i is a 4×4 block.
The desired requirements are that both the construction and the triangular solves using the matrices L_i and U_i should not cost too many floating-point operations and can be implemented in an efficient way.

Let the entries of D_i be grouped in square blocks of size 4 such that all elements multiplying the velocity components and pressure in a grid point form a block. Such a block will be represented by one entry. The matrix D_i has a block sparsity pattern that corresponds to the following 9-point discretization stencil:

$$\begin{array}{ccccc} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & C_{NW} & C_N & C_{NE} & \cdot \\ \cdot & C_W & C_P & C_E & \cdot \\ \cdot & C_{SW} & C_S & C_{SE} & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{array}$$

The coefficients in C_{NW}, C_{NE}, C_{SW} and C_{SE} are relatively small, and therefore, those blocks are neglected during the incomplete LU decomposition. Furthermore, when fill-in blocks are neglected, this all results in the matrix $L_i + U_i$ having the block sparsity patter that corresponds to the following 5-point stencil:

$$\begin{array}{ccccc} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & U_N & \cdot & \cdot \\ \cdot & L_W & U_P & U_E & \cdot \\ \cdot & \cdot & L_S & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{array}$$

where the diagonal of the matrix L consists of only identity matrices. However, some of the fill-in blocks are not neglected when the option 7blockdiag is chosen for the preconditioner in PARNASSOS. This option is mentioned in section 4.2.3

3.2 Data generation and storage

In order to optimize the way to solve the discretized physical equations, one must first know the matrix that comes out of the discrete equations. After this is known, the best way to solve this linear equation can be chosen.
When looking back at the generated matrix A from section 2.2.5, note that the block elements B_i, C_i, D_i, E_i and F_i are blocks of size $4 \times NY \times ubk$.

$$A = \begin{bmatrix} D_1 & E_1 & F_1 & 0 & 0 & 0 & \cdots \\ C_2 & D_2 & E_2 & F_2 & 0 & 0 & \cdots \\ B_3 & C_3 & D_3 & E_3 & F_3 & 0 & \cdots \\ 0 & B_4 & C_4 & D_4 & E_4 & F_4 & \cdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & 0 & B_g & C_g & D_g \end{bmatrix}$$

From the way PARNASSOS discretizes the physical equations, the blocks B_i, C_i, E_i and F_i are block diagonal matrices where the blocks on the main diagonal are of size 4×4. The D_i blocks can be filled in by the following stencil:

$$\begin{bmatrix} & T & \\ PP & Q & R \\ & S & \end{bmatrix}, \text{ with as positive directions : } \begin{bmatrix} & k & \\ & + & j \\ & & \end{bmatrix}$$

All the elements in the following matrix are 4×4 block matrices. In every 4×4 block matrix, including the 4×4 block matrices on the main diagonals of B_i, C_i, E_i and F_i, the coefficients are from the following equations:

- row 1. momentum equation in main stream direction (ξ, i-coordinate)
- row 2. continuity equation
- row 3. momentum equation in girthwise direction (ζ, k-coordinate)
- row 4. momentum equation in wall-normal direction (η, j-coordinate)

and the unknowns are as follows:

- First column of 4×4 block multiplies U1 (velocity in x-direction).
- Second column of 4×4 block multiplies U2 (velocity in y-direction).
- Third column of 4×4 block multiplies U3 (velocity in z-direction).
- Fourth column of 4×4 block multiplies P (pressure).

The blocks will be noted as $Q_{j,k}$, $PP_{j,k}$, $R_{j,k}$, $S_{j,k}$ and $T_{j,k}$ in the text and in the following matrix as Q, PP, R, S and T with the coefficients left of the matrix in the form (j, k). The actual matrix D_{L1} is a $4 \times NY \times ubk$ block matrix, where j goes from 1 to NY and k goes from 1 to ubk. The following example of the structure, $NY = 4$ and $ubk = 3$

$(1,1)$	Q	R			T							
$(2,1)$	PP	Q	R			T						
$(3,1)$		PP	Q	R			T					
$(4,1)$			PP	Q				T				
$(1,2)$	S				Q	R			T			
$(2,2)$		S			PP	Q	R			T		
$(3,2)$			S			PP	Q	R			T	
$(4,2)$				S			PP	Q				T
$(1,3)$					S				Q	R		
$(2,3)$						S			PP	Q	R	
$(3,3)$							S			PP	Q	R
$(4,3)$								S			PP	Q

Currently, the coefficients of the matrix A are stored per 4×4 blocks, where they are organized by four indices:

1. The nonzero block diagonal the 4×4 block belongs to, for example the block diagonal with only S
2. The plane the variables originate from, which is noted with the index $L1$ in D_{L1} or B_{L1}
3. The index j, which corresponds to the direction that is perpendicular to the surface of the ship, η
4. The index k, which corresponds to the direction that is perpendicular to η and the mainstream direction of the flow of water

3.3 Overview solver

In PARNASSOS, the file where the GMRES(m) is coded into is called *preclinsol.f*. The input of this file contains the following:

- The matrix A of size $4 \cdot g \cdot NY \cdot ubk$ subdivided in 4×4 blocks as described in the previous subsection
- The size of the matrix A, $4 \cdot g \cdot NY \cdot ubk$, that has already been calculated
- The previous approximation to the solution of the linear equation
- The right-hand side of the linear equation that needs to be solved
- Some work memory for the subroutines in the file

Finally, other variables are imported from other files in PARNASSOS that are read and stored by other parts of PARNASSOS that are needed for the calculations. Examples of these are the tolerance threshold for GMRES and at which iteration GMRES needs to be restarted, as stated in section 3.1.2.

The use of a matrix M as the preconditioner can be broken up into two parts. The first part is preparing the matrix M for repeated use by transforming the matrix M into a form that will be less computationally expensive than before the transformation. The other part is the repeated use of the new form of the matrix M.
The code in *preclinsol.f* is structured as follows to solve the linear equation $Ax = b$:

1. The preconditioner M, that is based on the matrix A, is calculated
2. The preconditioned version of the rhs, $M^{-1}b$, is calculated if the left preconditioned version of GMRES is chosen
3. The repeated version of GMRES, GMRES(m), is applied with chosen parameters
4. The preconditioned solution, x, is calculated if the right preconditioned version of GMRES is chosen

4 Main goal of the research

As mentioned in the intro, the main goal of this research is to make the solver faster. One of the ways to do this is to increase the convergence of GMRES by using a different preconditioner as the one mentioned before in section 3.1.6. If the full matrix that is used in GMRES is noted as the following:

$$A = \begin{bmatrix} D_1 & E_1 & F_1 & 0 & 0 & 0 & \cdots \\ C_2 & D_2 & E_2 & F_2 & 0 & 0 & \cdots \\ B_3 & C_3 & D_3 & E_3 & F_3 & 0 & \cdots \\ 0 & B_4 & C_4 & D_4 & E_4 & F_4 & \cdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & 0 & B_g & C_g & D_g \end{bmatrix}$$

, then the current objective is to get the following matrix M as preconditioner

$$M = \begin{bmatrix} D_1 & 0 & 0 & 0 & 0 & 0 & \cdots \\ C_2 & D_2 & 0 & 0 & 0 & 0 & \cdots \\ B_3 & C_3 & D_3 & 0 & 0 & 0 & \cdots \\ 0 & B_4 & C_4 & D_4 & 0 & 0 & \cdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & 0 & B_g & C_g & D_g \end{bmatrix}$$

The main difference between this preconditioner M and the one in section 3.1.6 is that in this case on the main diagonal the matrices are not altered. This may bring $M^{-1}A$ closer to being the identity matrix than the matrix M in section 3.1.6, which will help with the convergence of GMRES. The downside is that to solve $M^{-1}A$ in an efficient way, linear equations of the form $D_{L1}x = y$ have to be solved. This means that direct solvers need to be implemented. The first main section will explore the LU decomposition as an direct solver for the linear equation $D_{L1}x = y$. The second main section will let M go back to the form in section 3.1.6, where the blocks D_i are replaced with matrices M_i and explore an alteration to the LU decomposition, namely the $ILU(N)$ decomposition that is more general than the ILU already used. The final main section describes a different option for a direct solver which is specifically made for the type of matrices that are similar to D_{L1}.

In order to compare and determine which option as preconditioner is superior, the next section will describe the test that is done to all options.

4.1 Tests

To compare the different options as preconditioner in PARNASSOS, they are separated into different batches first. The batches are as follows:

1. The original options for preconditioner of PARNASSOS; done in the original file, the one least modified to still work and without LAPACK

2. The LU decomposition as direct solver for preconditioner for the blocks on the main diagonal

3. The $ILU(N)$ decomposition with multiple N values

In each of the batch, the options are compared with each other and the best option(s) are taken from the batch and compared with the others who were the best. What it means to be the best is a combination of how long one global iteration step takes with the preconditioner and the convergence of one global iteration step. The decisions will be discussed in the section after each test.

One of the decisions that can be said beforehand is that the amount of planes doesn't matter when comparing different preconditioners as long as the amount of planes is constant during the comparison. The amount of planes, g, can be taken larger to increase the global iteration speed of PARNASSOS with the cost of larger linear equations to solve in GMRES due to an increased amount of couplings between ξ constant planes. Changing only the preconditioners influences the convergence speed of the GMRES that solves those linear equations, not other parts of PARNASSOS. The only reason that the test is done for multiple values of g is to see of the difference in convergence speed between different preconditioners is consistent over multiple values of g.

The residual comes out of the values that are computed in the current iteration filled in into the function F in 1. closer to 0 means converged more.

The starting values for the tests are mentioned in the appendix in section 12.1.

The starting values for the model comes from an actual ship form, not just some arbitrary block, after 300 global iteration steps have been done with 7blockdiag as preconditioner.

Without abbreviations, the tables get unnecessarily large. Therefore, the following abbreviations have been used in the tables that contain the results:

- ttotsol: time triangular solves (applying preconditioner)
- timeprec: time construction of L and U (creating preconditioner)
- ttotmv: time for matrix-vector multiplications
- timegm: total time linear solvers
- ttotddotp: total time inner products
- tcoeff: time for construction Jacobean
- tturb: time for turbulence model
- exectm: total execution time

- nummatvec: total number of matvecs
- nnewton: total number of newton iterations

4.2 Test original options preconditioner PARNASSOS

4.2.1 Test: 1blockdiag

The next tests are done with the option 1blockdiag as preconditioner in PARNASSOS. The option 1blockdiag is choosing block jacobi as preconditioner, which is well parallelizable, but does not have a good iterative convergence.

amount of planes (g)	2	3	4	5
ttotsol	0.00	0.00	0.00	0.00
timeprec	1.97	1.94	1.98	2.00
ttotmv	176.41	231.13	258.52	310.28
timegm	401.58	487.57	555.06	638.97
ttotddotp	142.83	160.69	180.58	200.35
tcoeff	14.84	14.83	14.41	14.60
tturb	8.71	8.73	8.72	8.95
execm	430.14	515.97	582.98	667.29
nummatvec	81955	60486	46661	43312
nnewton	360	240	180	144
	maximum residual of equations			
ξ-momentum	$5.423 \cdot 10^{-3}$	$6.812 \cdot 10^{-3}$	$7.821 \cdot 10^{-3}$	$9.045 \cdot 10^{-3}$
continuity	$2.968 \cdot 10^{-3}$	$3.866 \cdot 10^{-3}$	$3.758 \cdot 10^{-3}$	$5.557 \cdot 10^{-3}$
ζ-momentum	$1.898 \cdot 10^{-3}$	$2.023 \cdot 10^{-3}$	$2.722 \cdot 10^{-3}$	$2.596 \cdot 10^{-3}$
η-momentum	$1.428 \cdot 10^{-3}$	$1.551 \cdot 10^{-3}$	$1.993 \cdot 10^{-3}$	$2.001 \cdot 10^{-3}$

Table 1: Results for tests with 1blockdiag with total amount of planes varying from 2 to 5

4.2.2 Test: 3blockdiag

The next tests are done with the option 3blockdiag as preconditioner in PARNASSOS. The option 3blockdiag is having a block-tridiagonal preconditioner which corresponds to the Succesive Line Overrelaxation method SLOR. The Succesive Overrelaxation method is explained in more detail in section 5.3.3 of C. Vuik and D.J.P. Lahaye (2017). This option is moderately parallelizable and has moderate iterative convergence.

amount of planes (g)	2	3	4	5
ttotsol	12.84	23.12	30.93	38.25
timeprec	2.08	2.18	2.18	2.17
ttotmv	29.34	44.01	56.02	67.08
timegm	74.33	112.73	144.08	175.00
ttotddotp	19.06	27.16	34.02	40.65
tcoeff	14.69	14.43	14.55	14.33
tturb	8.56	8.67	8.84	8.89
exectm	102.55	140.64	173.12	202.89
nummatvec	13963	11742	10403	9592
nnewton	360	240	180	144
	maximum residual of equations			
ξ-momentum	$5.474 \cdot 10^{-3}$	$6.896 \cdot 10^{-3}$	$7.921 \cdot 10^{-3}$	$9.125 \cdot 10^{-3}$
continuity	$2.968 \cdot 10^{-3}$	$3.924 \cdot 10^{-3}$	$3.827 \cdot 10^{-3}$	$5.632 \cdot 10^{-3}$
ζ-momentum	$2.122 \cdot 10^{-3}$	$2.173 \cdot 10^{-3}$	$2.996 \cdot 10^{-3}$	$2.771 \cdot 10^{-3}$
η-momentum	$1.428 \cdot 10^{-3}$	$1.549 \cdot 10^{-3}$	$1.997 \cdot 10^{-3}$	$2.004 \cdot 10^{-3}$

Table 2: Results for tests with 3blockdiag with total amount of planes varying from 2 to 5

4.2.3 Test: 7blockdiag

The next tests are done with the option 7blockdiag as preconditioner in PARNASSOS. The option 7blockdiag is an optimized version of ILU(1) as described at the end of section 6.1.This option is not parallelizable, but has a good iterative convergence.

amount of planes (g)	2	3	4	5
ttotsol	8.07	11.21	12.64	16.00
timeprec	3.22	3.35	3.36	3.38
ttotmv	7.63	10.85	12.39	15.90
timegm	22.03	30.30	33.84	43.13
ttotddotp	1.87	3.00	3.24	4.80
tcoeff	14.75	14.75	14.42	14.38
tturb	8.62	8.69	8.75	8.90
exectm	50.37	58.63	61.74	71.23
nummatvec	3588	2944	2333	2304
nnewton	360	240	180	144
	maximum residual of equations			
ξ-momentum	$5.465 \cdot 10^{-3}$	$6.880 \cdot 10^{-3}$	$7.873 \cdot 10^{-3}$	$9.093 \cdot 10^{-3}$
continuity	$2.968 \cdot 10^{-3}$	$3.915 \cdot 10^{-3}$	$3.792 \cdot 10^{-3}$	$5.614 \cdot 10^{-3}$
ζ-momentum	$2.123 \cdot 10^{-3}$	$2.173 \cdot 10^{-3}$	$2.999 \cdot 10^{-3}$	$2.775 \cdot 10^{-3}$
η-momentum	$1.428 \cdot 10^{-3}$	$1.549 \cdot 10^{-3}$	$1.997 \cdot 10^{-3}$	$2.002 \cdot 10^{-3}$

Table 3: Results for tests with 7blockdiag with total amount of planes varying from 2 to 5

4.3 Results

When comparing the three options above for a constant g, the maximum residual of the equations are relatively very close to each other. This is not the case for the total execution time: 1blockdiag as preconditioner takes for a chosen g around $3.5\times$ more time to complete a global iteration step when compared to 3blockdiag. When compared to 7blockdiag, 7blockdiag is around $2\times$ faster than 3blockdiag for $g = 2$ and for $g = 5$ that ratio is almost up to 2.9. Therefore, the best choice for preconditioner for this batch is 7blockdiag.

The option 1blockdiag has less elements from the full LU decomposition than 3blockdiag, which in turn has less elements from the full LU decomposition than 7blockdiag. They also have relatively decreasing amount of total execution time and decreasing value of nummatvec, which shows a decreasing amount of repeated GMRES use. Therefore, from these results, the best option to first take a look at is implementing the full LU decomposition in the options for preconditioners.

5 Complete LU decomposition of the main diagonal blocks

5.1 Theory

Finding an LU decomposition of the matrix A is defined as finding an upper triangular matrix U and a lower triangular matrix L such that $A = LU$. For a unique decomposition, the entries on the diagonal of the matrix U or the matrix L have to be all equal to one.

$$L = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & \cdots & 1 \end{bmatrix}, U = \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ 0 & u_{2,2} & \cdots & u_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{n,n} \end{bmatrix}$$

It is not always possible to get an LU decomposition. To always get an LU decomposition when having a square matrix A, need to permute the rows of A. This is called LU factorization with partial pivoting. Having the LU decomposition of a matrix A is great for exactly calculating the solution of the linear equation. By first calculating the solution y of the equation

$$Ly = b$$

and afterwards calculating the solution x of the equation

$$Ux = y$$

the solution x is more easily calculated than immediately trying to find the solution for the original linear equation.

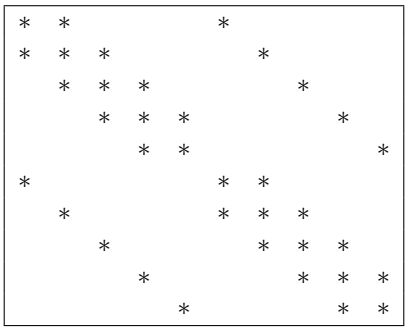
The algorithm for calculating the LU factorization of a matrix A without pivoting, where the lower triagnular part of A becomes the matrix L and the upper triangular part becomes U is the following:

Algorithm 4 Calculate LU decomposition

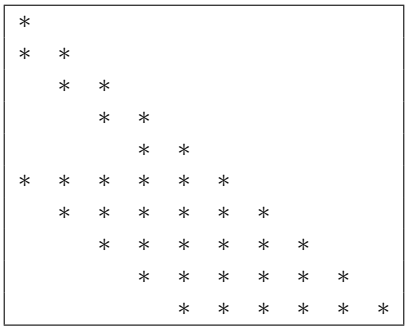
1: **for** $k = 1, \ldots, n-1$ **do**
2: **if** A(k,k) $= 0$ **then**
3: *quit algorithm (breakdown due to pivot)*
4: **else**
5: **for** $i = k+1, \ldots, n$ **do**
6: $L(i,k) = A(i,k)/A(k,k)$
7: $A(i,k) = L(i,k)$
8: **for** $j = k+1, \ldots, n$ **do**
9: $A(i,j) = A(i,j) - L(i,k)A(k,j)$

One of the things to keep in mind with getting an LU decomposition is that when not all elements of A are nonzero, that the places where nonzero elements

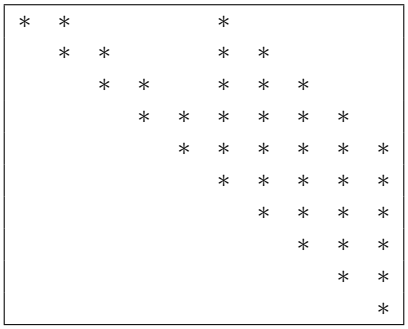
can appear in L and U do not overlap with the placing of nonzero elements in the original matrix A. This is called fill-in. To say that an matrix A has a small amount of nonzero elements in comparison to its dimensions is normally stated as A being sparse.
An example of this fill-in is when A is created from a 5-point stencil with the original grid being 5×2. The matrix has at the following places elements not equal to zero:



The matrix L will have the following places which can have elements not equal to zero:



and U has the following places which can have elements not equal to zero:



5.2 Implementation

One of the ways to implement the LU decomposition into the code of *preclinsol.f* is by using the library LAPACK [4], which is a library containing direct solvers. From the way the matrices D_{L1} are structured, it can be surmised that the best way to store the matrices D_{L1} is in band matrix form. For example, the following matrix Z with one superdiagonal and two subdiagonals,

$$Z = \begin{bmatrix} z_{11} & z_{12} & 0 & 0 & 0 & 0 \\ z_{21} & z_{22} & z_{23} & 0 & 0 & 0 \\ z_{31} & z_{32} & z_{33} & z_{34} & 0 & 0 \\ 0 & z_{42} & z_{43} & z_{44} & z_{45} & 0 \\ 0 & 0 & z_{53} & z_{54} & z_{55} & z_{56} \\ 0 & 0 & 0 & z_{64} & z_{65} & z_{66} \end{bmatrix}$$

transforms into

$$Z' = \begin{bmatrix} * & z_{12} & z_{23} & z_{34} & z_{45} & z_{56} \\ z_{11} & z_{22} & z_{33} & z_{44} & z_{55} & z_{66} \\ z_{21} & z_{32} & z_{43} & z_{54} & z_{65} & * \\ z_{31} & z_{42} & z_{53} & z_{64} & * & * \end{bmatrix}$$

The code that transforms the current way the variables are stored into the band form to use in LAPACK will be in the appendix in section 12.2.

The function from LAPACK that performs the LU decomposition on a band matrix is called DGBTRF. The abbreviation DGBTRF stands for Double precision General Band matrix Triangular matrix Factorization. The function DGBTRF computes an LU factorization of a real valued m-by-n band matrix using partial pivoting with row interchanges.

The function from LAPACK which applies the LU decomposition to solve the linear system is called DGBTRS. The abbreviation DGBTRS stands for Double precision General Band matrix Triangular matrix Solver. DGBTRS solves a system of linear equations with a general band matrix using the LU factorization computed by DGBTRF.
The following instructions are applied to all matrices D_{L1}, which is only done once whenever the full matrix A is changed:

1. The matrix D_{L1} is transformed in its band matrix form and stored in a variable called MB.

2. The function DGBTRF is applied to the band version of the matrix D_{L1} in MB. The function returns the transformed version of the matrix D'_{L1}, that contains the LU factorization of D_{L1}, and a vector ipiv which contains the permutations made.

[4] http://www.netlib.org/lapack/explore-html/

3. The band version of matrix D_{L1} is automatically replaced by the matrix D'_{L1} by the function DGBTRF. The vector ipiv is stored in a separate vector called IPIV.

When the equation $D_{L1}x = b$ needs to be solved for x, one only needs to apply the function DGBTRS to D'_{L1}, with the corresponding permutation vector ipiv, and the vector b to get the vector x.
To check if the functions DGBTRF and DGBTRS are correctly implemented on the matrices D_{L1} and to check how precise the solutions from those functions are when used on D_{L1}, the following test is performed for each fixed $L1$ between 1 and g with the matrices D_{L1} created from a given model:

1. Create the following $4 \cdot NY \cdot ubk \times 1$ vector v: $\mathrm{v}(k) = k$
2. Multiply v with the matrix D_{L1} to create v2: $\mathrm{v2} = D_{L1} \cdot \mathrm{v}$
3. Solve the equation $D_{L1} \cdot x = \mathrm{v2}$ by using DGBTRF and DGBTRS

The expected outcome should be $||x - \mathrm{v}||_2$ being of the order of the machine precision. This is not the case. After a few tries checking what went wrong in the code, two major problems occurred: one programming mistake, that is ultimately fixed, and the problem that the matrices D_{L1} all have condition numbers of the order 10^{23}. This means the linear problem $D_{L1}x = b$ is ill conditioned and must be transformed into a linear problem that is well-conditioned.

5.3 Scaling the main diagonal blocks of A

To solve this problem of the matrices D_{L1} being ill-conditioned, the matrix Q containing only the 4×4 blocks $Q_{j,k}$ on its diagonal to scale the matrix A. This will be called the diagonal block scaling of the matrix A. The entire linear system becomes $Q^{-1}Ax = Q^{-1}b$, which results in each of the linear equations to solve with D_{L1} becoming $Q^{-1}D_{L1}x = Q^{-1}b'$. Each $Q_{j,k}$ block is inverted using Cramer's rule, which is doable for 4×4 block matrices. This results in the condition numbers of the matrices D_{L1} becoming of the order 10^5, which means the linear problems $Q^{-1}D_{L1}x = Q^{-1}b$ are well-conditioned. After doing the test at the previous section for $L1$ between 1 and 4 with the diagonally scaled matrix A, the resulting vectors all have absolute errors of order 10^{-9} and relative errors of order 10^{-13}.

The previous test was in order to test if the functions DGBTRF and DGBTRS solved the linear equations $D_{L1}x = b$. The following test that is done determines if the entire matrix M, as stated in the beginning of section 4, is correctly implemented in the code:

1. Create the following $4 \cdot NY \cdot ubk \cdot g \times 1$ vector v: $\mathrm{v}(k) = k$.
2. Multiply v with the matrix M to create v2: $\mathrm{v2} = M \cdot V$.

3. Apply the diagonal block scaling to both matrix M and vector v2.

4. Solve the equation $(Q^{-1}M)x = Q^{-1}\text{v2}$ by using forward substitution with blocks of size $4 \cdot NY \cdot ubk$.

    - When solving the equations for the $4 \cdot NY \cdot ubk$ square blocks on the diagonal, use DGBTRF and DGBTRS.

The expected outcome should be $||x - \text{v}||_2$ being of the order of the machine precision.

After doing the test as laid out above, vector x and vector v are partitioned into g parts of $4 \cdot NY \cdot ubk$ consecutive elements, which are related to planes. When comparing each of the parts of x and v, which relate to the same plane, the absolute error and relative error is calculated and the maximum of those numbers is taken. The maximum of absolute errors for each plane is between $3.3 \cdot 10^{-9}$ and $4.5 \cdot 10^{-8}$, for which the maximum of absolute errors increases for each plane, and the maximum of relative errors is around $4 \cdot 10^{-13}$ or around $6 \cdot 10^{-13}$.

5.4 Reordering rows and columns to reduce bandwidth

The upper and lower bandwidth of the matrices D_{L1} are $4 \cdot (NY + 1) - 1$. We try to switch the coordinates of the 4×4 blocks from (j, k) to (k, j) in order to reduce the bandwidth in the block matrices D_k when $NY > ubk$. The upper and lower bandwidth of the D_{L1} matrices becomes $4 * (ubk + 1) - 1$. From here on out, the bandwidth of the matrix will be called NY when the original form of D_{L1} is mentioned and the bandwidth of the matrix will be called ubk when the coordinates in the matrices D_{L1} are switched.
If Z is the permutation matrix that switches (j, k) around with (k, j), then $Z^T = Z^{-1}$. Because Z only switches distinct pairs of rows, the permutation matrix can be written as the product of disjoint transpositions, from which follows that Z has the property $Z^{-1} = Z$. The equation $D_k x = y$ can become: $ZD_kZu = Zy$ with $u = Zx$.

The condition numbers of the matrices D_{L1} with bandwidth ubk, are between $1.4 \cdot 10^5$ and $1.7 \cdot 10^5$. The order of the condition numbers are reasonable.
After doing the test from previous chapter on the entire matrix M, with the matrices D_{L1} having bandwidth ubk when solving the equation for the square blocks on the diagonal, the maximum absolute error is between $5.4 \cdot 10^{-7}$ and $8.4 \cdot 10^{-6}$, for which the absolute maximum error increases for each plane, and the maximum relative error is between $8.2 \cdot 10^{-11}$ and $1.3 \cdot 10^{-10}$.

Even though the relative error is of a relatively larger order when the bandwidth is ubk instead of NY, this is not a big problem. The relative error with bandwidth ubk is small enough and the matrix is only being used as a preconditioner. The CPU operator time drastically decreases with bandwidth NZ, thus

even though the relative error is a bit larger than with bandwidth NY, the direct solver is better when using the version with bandwidth *ubk*.

5.5 Tests

The next tests are done with the LU used as component in the preconditioner M in PARNASSOS to allow for whole blocks on the main diagonal of M. This version is called block_fulli in a new version of the file *preclinsol.f*

amount of planes (g)	2	3	4	5
ttotsol	186.13	185.88	203.15	248.36
timeprec	544.17	539.99	537.93	537.84
ttotmv	4.73	5.43	6.41	8.03
timegm	756.58	752.52	769.09	816.32
ttotddotp	0.89	0.94	1.09	1.48
tcoeff	14.87	15.18	14.74	14.68
tturb	8.84	8.95	9.19	9.57
exectm	786.56	781.82	798.34	846.05
nummatvec	2160	1441	1178	1147
nnewton	360	240	180	144
	maximum residual of equations			
ξ-momentum	$5.473 \cdot 10^{-3}$	$6.881 \cdot 10^{-3}$	$7.887 \cdot 10^{-3}$	$9.101 \cdot 10^{-3}$
continuity	$2.968 \cdot 10^{-3}$	$3.916 \cdot 10^{-3}$	$3.803 \cdot 10^{-3}$	$5.608 \cdot 10^{-3}$
ζ-momentum	$2.122 \cdot 10^{-3}$	$2.172 \cdot 10^{-3}$	$2.998 \cdot 10^{-3}$	$2.776 \cdot 10^{-3}$
η-momentum	$1.427 \cdot 10^{-3}$	$1.549 \cdot 10^{-3}$	$1.997 \cdot 10^{-3}$	$2.004 \cdot 10^{-3}$

Table 4: Results for tests with block_fulli with total amount of planes varying from 2 to 5

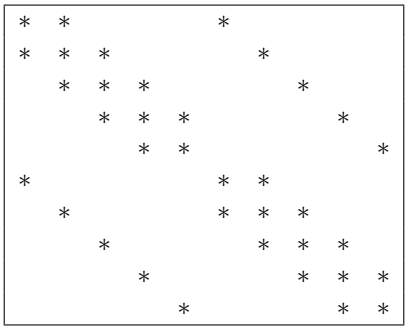
Because this is the only option in this batch, this is the best one. Therefore, the comparison will only be made with the other best options from the other batches.

6 $ILU(N)$ decomposition

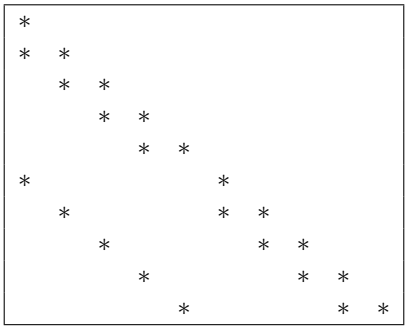
Because the fill in of the LU decomposition of a sparse matrix A can create an enormous memory requirement and more nonzero elements to work with, a version of LU decomposition was created named $ILU(N)$. The main idea behind $ILU(N)$ is wanting to find an LU decomposition of the form $LU = A + R$, in which the elements of the residual matrix R are small.

6.1 Theory

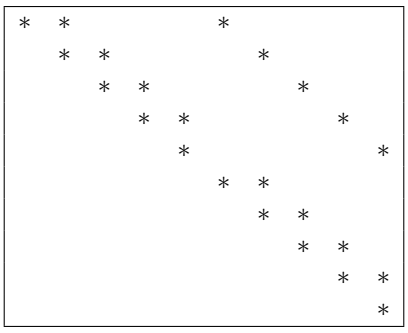
$ILU(0)$ has the requirement that the matrices L and U in $LU = A+R$ have only nonzero elements where there are nonzero elements in the matrix A. When going back to the example of an matrix A which is created from a 5-point stencil with the original grid being 5×2. The matrix has at the following places elements not equal to zero:



The matrix L of the $ILU(0)$ decomposition will have the following places which can have elements not equal to zero:



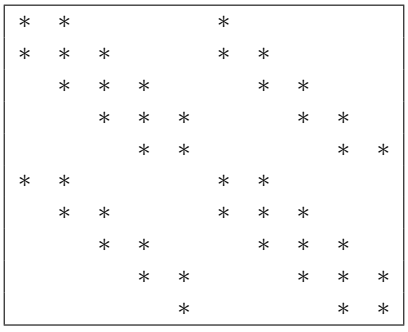
and U has the following places which can have elements not equal to zero:



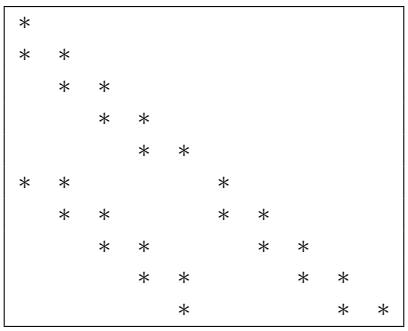
Because the amount of fill in is zero, the matrices L and U together have the same memory requirements as the matrix A. This means the amount of nonzero elements in the calculations is much smaller than with an LU decomposition. However, the $ILU(0)$ decomposition is in most cases not a great approximation of the matrix A. Therefore a trade-off can be made with the amount of fill in and how accurate the $ILU(N)$ decomposition is to the actual LU decomposition of the matrix A.

The other $ILU(N)$ are defined recursively from $ILU(0)$. Define L_N and U_N as the upper and lower triangular matrices from the $ILU(N)$ decomposition of the matrix A and let $A_N = L_N U_N$ for all N

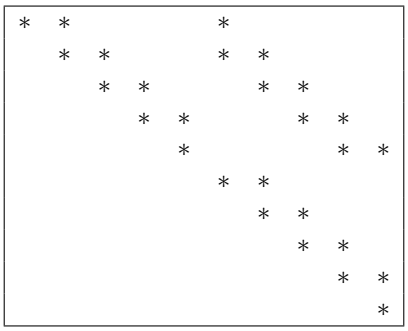
To get L_1 and U_1 for $ILU(1)$, allow for nonzero elements in de L_1 and U_1 to be in the same places as the nonzero elements of A_0. Then with those restrictions for the fill in in mind, minimize the elements of R_1 in $R_1 = L_1 U_1 - A$. In the example given before, the matrix $A_0 = L_0 U_0$ have the nonzero elements at the following places:



L_1 and U_1 have nonzero elements only on the spots where A_0 has nonzero elements. Therefore the nonzero elements of L_1 are in the places:



and U_1 has the following places which can have elements not equal to zero:



The rest of the restrictions for L_N and U_N are of the same form: L_N and U_N can only have nonzero elements on the places where $A_{N-1} = L_{N-1}U_{N-1}$ has nonzero elements.

For band matrices formed by a 5-point stencil, the places of the matrix U_N from $ILU(N)$ where nonzero elements can occur is the transpose of the places of the matrix L_N where nonzero matrices can occur. The following picture shows at what order the fill in occurs in L_N if the iterative process is followed, where the numbers note when the element is possibly nonzero at.

0													
0	0												
	0	0											
		0	0										
			0	0									
				0	0								
					0	0							
0	1	2	3	4	5	6	0						
	0	1	2	3	4	5	0	0					
		0	1	2	3	4	3	0	0				
			0	1	2	3	4	3	0	0			
				0	1	2	5	4	3	0	0		
					0	1	6	5	4	3	0	0	
						0	7	6	5	4	3	0	0

This figure shows that at first, the fill-in comes inwards with diagonal lines from the outer diagonal. However, at and after $N = 3$, the fill in also comes with diagonal lines outwards from the main diagonal. In order to keep things simple, it is possible for a certain N that the allowed fill-in is at most N extra diagonals from the outer diagonal inwards and at most $N - 2$ extra diagonals from the inner diagonals outward.

In the current version of $ILU(N)$, the condition for the allowed fill-in at a certain N is even simpler: N extra diagonals from the outer diagonal inwards and at most N extra diagonals from the inner diagonals outward.

6.2 Tests

The next tests are done with the ILU used as component in the preconditioner M in PARNASSOS for N ranging from 0 to 3.

amount of planes (g)	2	3	4	5
ttotsol	7.17	10.98	12.79	16.67
timeprec	3.84	3.99	4.02	4.03
ttotmv	9.82	14.57	16.26	21.25
timegm	25.85	37.51	42.03	55.52
ttotddotp	3.10	4.90	5.38	8.12
tcoeff	15.08	14.60	14.84	14.45
tturb	8.62	8.73	8.75	8.92
exectm	56.11	65.84	70.73	83.75
nummatvec	4659	3921	3028	3089
nnewton	360	240	180	144
	maximum residual of equations			
ξ-momentum	$5.457 \cdot 10^{-3}$	$6.885 \cdot 10^{-3}$	$7.813 \cdot 10^{-3}$	$9.114 \cdot 10^{-3}$
continuity	$2.968 \cdot 10^{-3}$	$3.925 \cdot 10^{-3}$	$3.744 \cdot 10^{-3}$	$5.655 \cdot 10^{-3}$
ζ-momentum	$2.122 \cdot 10^{-3}$	$2.172 \cdot 10^{-3}$	$2.998 \cdot 10^{-3}$	$2.782 \cdot 10^{-3}$
η-momentum	$1.427 \cdot 10^{-3}$	$1.550 \cdot 10^{-3}$	$1.995 \cdot 10^{-3}$	$2.005 \cdot 10^{-3}$

Table 5: Results for tests with $ILU(0)$ with total amount of planes varying from 2 to 5

amount of planes (g)	2	3	4	5
ttotsol	10.87	13.56	15.18	19.48
timeprec	7.63	7.55	7.54	7.57
ttotmv	7.56	11.08	12.59	16.20
timegm	29.46	37.19	41.30	52.37
ttotddotp	2.02	2.98	3.49	5.26
tcoeff	15.07	14.55	14.56	14.63
tturb	8.72	8.69	8.90	8.89
exectm	58.52	65.38	69.74	80.88
nummatvec	3588	2944	2333	2304
nnewton	360	240	180	144
	maximum residual of equations			
ξ-momentum	$5.465 \cdot 10^{-3}$	$6.880 \cdot 10^{-3}$	$7.873 \cdot 10^{-3}$	$9.093 \cdot 10^{-3}$
continuity	$2.968 \cdot 10^{-3}$	$3.915 \cdot 10^{-3}$	$3.792 \cdot 10^{-3}$	$5.614 \cdot 10^{-3}$
ζ-momentum	$2.123 \cdot 10^{-3}$	$2.173 \cdot 10^{-3}$	$2.999 \cdot 10^{-3}$	$2.775 \cdot 10^{-3}$
η-momentum	$1.428 \cdot 10^{-3}$	$1.549 \cdot 10^{-3}$	$1.997 \cdot 10^{-3}$	$2.002 \cdot 10^{-3}$

Table 6: Results for tests with $ILU(1)$ with total amount of planes varying from 2 to 5

amount of planes (g)	2	3	4	5
ttotsol	15.62	21.43	23.50	30.07
timeprec	10.92	11.40	11.40	11.30
ttotmv	8.24	12.10	13.98	18.50
timegm	38.75	51.70	56.16	70.51
ttotddotp	2.38	3.96	4.26	6.31
tcoeff	14.77	14.77	14.75	14.59
tturb	8.62	8.74	8.81	8.89
exectm	67.30	80.15	84.94	99.10
nummatvec	3913	3273	2600	2622
nnewton	360	240	180	144
	maximum residual of equations			
ξ-momentum	$5.465 \cdot 10^{-3}$	$6.883 \cdot 10^{-3}$	$7.870 \cdot 10^{-3}$	$9.092 \cdot 10^{-3}$
continuity	$2.968 \cdot 10^{-3}$	$3.920 \cdot 10^{-3}$	$3.792 \cdot 10^{-3}$	$5.614 \cdot 10^{-3}$
ζ-momentum	$2.124 \cdot 10^{-3}$	$2.172 \cdot 10^{-3}$	$2.999 \cdot 10^{-3}$	$2.776 \cdot 10^{-3}$
η-momentum	$1.427 \cdot 10^{-3}$	$1.548 \cdot 10^{-3}$	$1.999 \cdot 10^{-3}$	$2.001 \cdot 10^{-3}$

Table 7: Results for tests with $ILU(2)$ with total amount of planes varying from 2 to 5

amount of planes (g)	2	3	4	5
ttotsol	21.41	27.24	30.06	37.85
timeprec	17.97	18.84	19.45	19.12
ttotmv	8.87	12.50	14.25	18.48
timegm	52.44	64.85	70.92	86.17
ttotddotp	2.54	3.84	4.28	6.58
tcoeff	15.22	14.95	14.72	14.58
tturb	9.02	8.85	8.90	9.00
exectm	82.08	93.73	99.75	114.91
nummatvec	3930	3306	2592	2636
nnewton	360	240	180	144
	maximum residual of equations			
ξ-momentum	$5.467 \cdot 10^{-3}$	$6.882 \cdot 10^{-3}$	$7.869 \cdot 10^{-3}$	$9.093 \cdot 10^{-3}$
continuity	$2.968 \cdot 10^{-3}$	$3.919 \cdot 10^{-3}$	$3.791 \cdot 10^{-3}$	$5.618 \cdot 10^{-3}$
ζ-momentum	$2.124 \cdot 10^{-3}$	$2.172 \cdot 10^{-3}$	$3.000 \cdot 10^{-3}$	$2.775 \cdot 10^{-3}$
η-momentum	$1.427 \cdot 10^{-3}$	$1.548 \cdot 10^{-3}$	$1.996 \cdot 10^{-3}$	$2.005 \cdot 10^{-3}$

Table 8: Results for tests with $ILU(3)$ with total amount of planes varying from 2 to 5

6.3 Results

Comparing $ILU(0)$ with $ILU(1)$, the latter has used a fewer amount of matrix vector multiplications with the original matrix on which M is based for every constant amount of planes than the former. This means that with $ILU(1)$, the preconditioner GMRES converges more quickly than with $ILU(0)$. due to the increased amount of matrix vector multiplications, $ILU(0)$ spends much more time on those kinds of operations than $ILU(1)$. However, $ILU(0)$ has fewer nonzero elements, which results in a reduced amount of time spend on solving the linear equation $Mx = b$ and creating the preconditioner than $ILU(1)$. As a result, the total execution time for one global iteration step with $ILU(0)$ or $ILU(1)$ is comparable.

When comparing $ILU(1)$ with $ILU(2)$ and $ILU(3)$ for a constant amount of planes, the increasing amount of elements used seems not to outweigh a potential for becoming a better preconditioner. In fact, for all given g, increasing N will increase the amount of matrix vector multiplications, which means that GMRES with $ILU(2)$ as preconditioner converges slower than with $ILU(1)$ and $ILU(3)$ converges as good as $ILU(2)$ for the value *nummatvec* is almost the same. The increased total execution time seems to come from the fact that they let GMRES converge less and there are more nonzero elements.

Therefore, $ILU(2)$ and $ILU(3)$ are worse as a preconditioner for GMRES than $ILU(0)$ and $ILU(1)$. As for the difference between $ILU(0)$ and $ILU(1)$, $ILU(0)$ seems to be slightly better for $g = 2, 3$ and 4, while being a bit worse at $g = 5$. Therefore, both will be further compared at the end.

7 Experiment with other direct solver

In this section, a different direct solver is explored. This direct solver is specifically made for matrices with a 5-point stencil on a rectangular grid.

7.1 Setup idea

The matrix that is considered is the matrix M as stated in section 4, specifically the matrices D_{L1} which structure of the nonzero elements are explained in 3.2. Because M is going to be used as a preconditioner for A, slightly changing the elements in D_{L1} should not matter that much.

The reason that this is stated is that the block matrices containing $T_{j,k}$ are not invertible, which is necessary to create the form necessary for the direct solver. However, if ϵ times the 4×4 identity matrix is added or subtracted to each of the $T_{j,k}$ blocks, they become invertible if $-\epsilon$ is not an eigenvalue of an $T_{j,k}$ block. After row scaling the matrix D_{L1} with matrices $T_{j,k} + \epsilon I$, the matrix D_{L1} has the following form:

$$\begin{bmatrix} A_1 & I & & & \\ B_2 & A_2 & I & & \\ & \ddots & \ddots & \ddots & \\ & & B_{ubk-1} & A_{ubk-1} & I \\ & & & B_{ubk} & A_{ubk} \end{bmatrix}$$

where A_i are tridiagonal matrices and B_i are block diagonal matrices. This form is the one necessary in order to use the following direct solver to take the inverse of the slightly altered matrix M and multiply it with matrix A.

One remark about ϵ that has to be made is that there is a trade-off that happens. On one hand, ϵ wants to be taken small such that the altered matrices D_{L1} are still very close to being the original matrices D_{L1}. This will help with the convergence of GMRES. On the other hand, ϵ can be taken larger such that taking the inverse of the matrices $T_{j,k} + \epsilon I$ have smaller elements which helps with reducing round-off errors.

7.2 Explanation algorithm

Let the block-tridiagonal linear equation $Mx = y$ be of the form

$$\begin{bmatrix} A_1 & I & & & & \\ B_2 & A_2 & I & & & \\ & B_3 & A_3 & I & & \\ & & \ddots & \ddots & \ddots & \\ & & & B_{m-1} & A_{m-1} & I \\ & & & & B_m & A_m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{m-1} \\ x_m \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{m-1} \\ y_m \end{bmatrix}$$

.

The idea for the direct solver is to solve this equation in the following way:

First some Gaussian elimination steps: by multiplying the first equation to the left by A_2 and subtracting it from the second row, the linear equation becomes:

$$\begin{bmatrix} A_1 & I & & & & \\ B_2 - A_2A_1 & O & I & & & \\ & B_3 & A_3 & I & & \\ & & \ddots & \ddots & \ddots & \\ & & & B_{m-1} & A_{m-1} & I \\ & & & & B_m & A_m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{m-1} \\ x_m \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 - A_2y_1 \\ y_3 \\ \vdots \\ y_{m-1} \\ y_m \end{bmatrix}$$

.

Now the second row only has a nonzero block element in the first column and the identity in the third column. To shorten the notation, let $G_2 = B_2 - A_2A_1$ and $v_2 = y_2 - A_2y_1$. Then for consistency, note $G_1 = A_1$ and $v_1 = y_1$.

Next is multiplying the first row by B_3, multiplying the second row by A_3 and subtracting both from the third row. The equation becomes:

$$\begin{bmatrix} G_1 & I & & & & \\ G_2 & & I & & & \\ G_3 & & & I & & \\ & \ddots & \ddots & \ddots & & \\ & & B_{m-1} & A_{m-1} & I & \\ & & & B_m & A_m & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{m-1} \\ x_m \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ y_3 - A_3v_2 - B_3v_1 \\ \vdots \\ y_{m-1} \\ y_m \end{bmatrix}$$

.

This process is repeated until the last row is reached. The recurrence relation for G_k and v_k for $k \geq 3$ are:

$$\begin{aligned} G_k &= -A_kG_{k-1} - B_kG_{k-2} \\ v_k &= y_k - A_kv_{k-1} - B_kv_{k-2} \end{aligned}$$

The final linear system is:

$$\begin{bmatrix} G_1 & I & & & & \\ G_2 & & I & & & \\ G_3 & & & I & & \\ \vdots & & \ddots & \ddots & \ddots & \\ G_{m-1} & & & & & I \\ G_m & & & & & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{m-1} \\ x_m \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{m-1} \\ v_m \end{bmatrix}$$

.
The final block row gives a linear equation with only x_1:

$$G_m x_1 = v_m$$

Solving this equation gives x_1.

One way to calculate x_k without the use of v_k and G_k with $k < m$ after calculating x_1 is by returning to the original form:

$$\begin{bmatrix} A_1 & I & & & & \\ B_2 & A_2 & I & & & \\ & B_3 & A_3 & I & & \\ & & \ddots & \ddots & \ddots & \\ & & & B_{m-1} & A_{m-1} & I \\ & & & & B_m & A_m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{m-1} \\ x_m \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{m-1} \\ y_m \end{bmatrix}$$

.
The first equation goes as follows:

$$A_1 x_1 + x_2 = y_1$$

By using the fact that x_1 is known, x_2 can be calculated:

$$x_2 = y_1 - A_1 x_1$$

From the second row:

$$B_2 x_1 + A_2 x_2 + x_3 = y_2$$

By using the fact that x_1 and x_2 are known, x_3 can be calculated:

$$x_3 = y_2 - A_2 x_2 - B_2 x_1$$

In this way, the other vectors x_k can be calculated recursively by using the two previous solutions x_{k-1} and x_{k-2}:

$$x_k = y_{k-1} - A_{k-1} x_{k-1} - B_{k-1} x_{k-2}$$

When the linear equation $Mx = y$ needs to be solved for different y, the matrix G_m only needs to be calculated once. Then only the recursion steps need to be

taken with the vector y to get the vector v_m in order to get the full equation $G_m x_1 = v_m$. The rest of the calculations can be done by using the coefficients from the matrix M.

7.3 Theory

In the following sections, the starting matrix M is of the form:

$$\begin{bmatrix} A_1 & I & & & \\ B_2 & A_2 & I & & \\ & \ddots & \ddots & \ddots & \\ & & B_{m-1} & A_{m-1} & I \\ & & & B_m & A_m \end{bmatrix}$$

Where the $n \times n$ matrices A_i are tridiagonal block matrices and the $n \times n$ matrices B_i are diagonal block matrices. The blocks are of size 4×4.

Next up, the $n \times n$ matrices G_i that are mentioned in the previous section are made in the following way: $G_1 = A_1$, $G_2 = B_2 - A_2 A_1$ and $G_j = -A_j G_{j-1} - B_j G_{j-2}$, $\forall j = 2, \ldots, m$.

In the following sections, the steps are explained in order to arrive at the computational complexity of the algorithm.

7.3.1 Form G_j matrices

To calculate the complexity, the maximal size of the block matrices G_j have to be known which will come from some theory about band matrices. First multiplication of band matrices:

Proposition 1. *Let C_1 be a block band matrix with upper bandwidth k_1 and lower bandwidth l_1 and C_2 be a block band matrix with upper bandwidth k_2 and lower bandwidth l_2. Then $C_1 C_2$ is a block band matrix with at most upper bandwidth $k_1 + k_2$ and lower bandwidth $l_1 + l_2$.*

Proof. To get an element $(C_1 C_2)_{i,j}$, row i of C_1 is multiplied with column j of C_2. Row i of C_1 has nonzero elements from index $\max(1, i - l_1)$ to $\min(n, i + k_1)$. Column j of C_2 has nonzero elements from index $\max(1, j - k_2)$ to $\min(n, j + l_2)$. Multiplying row i of C_1 with column j of C_2 has a chance to become nonzero if two elements are nonzero at the same index, but must be zero if there are no two elements that are nonzero at the same index.
Overlap of the nonzero part of the row and column happens when $i + k_1 \leq j - k_2$ and $j + l_2 \geq i - l_1 \Leftrightarrow j \leq i + k_1 + k_2$ and $j \geq i - l_1 - l_2$
Overlap of the nonzero part of the row and column never happens when $i + k_1 < j - k_2$ or $j + l_2 < i - l_1 \Leftrightarrow j < i - l_1 - l_2$ or $j > i + k_1 + k_2$
Therefore $C_1 C_2$ is a block band matrix with at most upper bandwidth $k_1 + k_2$ and lower bandwidth $l_1 + l_2$. □

Proposition 2. *With the assumptions made about A_i and B_i at the end of section 7.2, the matrices G_j are block band matrices with bandwidth at most j.*

Proof. Begin case $j = 1$: $G_1 = A_1$ with A_1 being a block band matrix with bandwidth 1.
Begin case $j = 2$: B_2 has bandwidth 0. A_1 and A_2 have bandwidth 1, therefore A_1A_2 has at most bandwidth $1 + 1 = 2$. $G_2 = -B_2 - A_1A_2$ thus G_2 has at most bandwidth $\max(0, 2) = 2$. Induction hypothesis: G_{j-1} and G_{j-2} are block band matrices with at most bandwidth $j - 1$ and $j - 2$ respectively.
B_j has bandwidth 0 and by IH G_{j-2} has at most bandwidth $j - 2$, therefore B_jG_{j-2} has at most bandwidth $j - 2$.
A_j has bandwidth 1 and by IH G_{j-1} has at most bandwidth $j - 1$, therefore A_jG_{j-1} has at most bandwidth $1 + j - 1 = j$.
$G_j = -A_jG_{j-1} - B_jG_{j-2}$ thus G_j has at most bandwidth $\max(j, j-2) = j$. □

7.3.2 Computational complexity band matrix multiplications

Now that it is known that each of the matrices G_j are block band matrices with bandwidth at most its index, some theory is needed for matrix multiplication with G_j. Specifically, as seen in the recurrence relation to get G_j from G_{j-1} and G_{j-2}, the computational complexity of multiplying a tridiagonal matrix A_j with the band matrix G_{j-1} is needed.

Let C in the next two algorithms be an tridiagonal block matrix with blocks of size 4×4 and D be an $n \times n$ block band matrix with bandwidth k with blocks of size 4×4. Then an algorithm to calculate $W = CD$ goes as follows, where the indices correspond to the row and column index in the matrix:

Algorithm 5 Calculate tridiagonal matrix multiplication $W = CD$

```
1: Define the n × n band matrix W := 0
2: for i = 1, ..., n do
3:     for l = max(1, i − 1), ..., min(n, i + 1) do
4:         for j = max(1, l − k), ..., min(n, l + k) do
5:             Compute W_{i,j} := W_{i,j} + C_{i,l} · D_{l,j}
```

Multiplying two 4×4 matrices with each other have computational complexity $2 \cdot 4^3$. Therefore, the computational complexity of this algorithm is $2 \cdot 4^3 \cdot 3 \cdot (2k + 1) \cdot n$.

Now for block matrix vector multiplication with a $n \times n$ tridiagonal block matrix C with blocks of size 4×4 and a $n \times 1$ block vector v with sections of size 4.

Algorithm 6 Calculate tridiagonal matrix vector multiplication $w = Cv$

1: *Define the* $n \times 1$ *vector* $w := 0$
2: **for** $i = 1, \ldots, n$ **do**
3: **for** $j = \max(1, i-1), \ldots, \min(n, i+1)$ **do**
4: *Compute* $w_i := w_i + C_{i,j} \cdot v_j$

Multiplying a 4×4 matrix with a 4×1 vector has computational complexity $2 \cdot 4^2$. Thus, the computational complexity of this algorithm is $2 \cdot 4^2 \cdot 3 \cdot n$.

Finally, it is necessary to know the computational complexity of the $n \times n$ diagonal block matrix C multiplication with a $n \times n$ block band matrix D with bandwidth k and block size 4×4.

Algorithm 7 Calculate diagonal block matrix multiplication $W = CD$

1: *Define the* $n \times n$ *band matrix* $W := 0$
2: **for** $i = 1, \ldots, n$ **do**
3: **for** $j = \max(1, i-k), \ldots, \min(n, i+k)$ **do**
4: *Compute* $W_{i,j} := W_{i,j} + C_{i,l} \cdot D_{l,j}$

The computational complexity of this algorithm is $2 \cdot 4^3 \cdot (2k+1) \cdot n$.

7.3.3 Computational complexity direct solver

The following comes from Scientific Computing (wi4201) lecture notes (Vuik and Lahaye (2017)), section 7.3.1 between theorem 4.9.1 and example 4.9.2: Let C be an $n \times n$ band matrix with upper bandwidth k and lower bandwidth l. Then the factorization stage to get an LU decomposition without pivoting requires $2kln$ flops. The forward and backward triangular solve costs $2nl$ and $2nk$ flops, respectively.

Thus for an $n \times n$ band matrix with bandwidth k, the LU decomposition requires $2k^2n$ flops and the forward and backward triangular solve both cost $2nk$ flops.

The previous fact was for normal band matrices. For a block band matrix with blocks of size 4×4 holds that the element multiplication takes 4^3 flops and adding two 4×4 matrices takes 4^2 flops. Therefore, when taking that into account, the LU decomposition of an $n \times n$ block band matrix with bandwidth k and block size 4×4 requires at most $4^3 \cdot 2k^2n$ flops and the forward and backward triangular solve both cost at most $4^3 \cdot 2nk$ flops.

Now that all the necessary computational complexities have been calculated, the final two algorithms can be given. The first algorithm calculates G_m and calculates its LU decomposition. The second algorithm solves the equation

$Mx = y$ with the calculated matrix G_m. As stated in section 7.2, The matrices G_j are created by the following recurrence relation:

$$G_j \quad = \quad -A_j G_{j-1} - B_j G_{j-2}$$

with $G_1 = A_1$ and $G_2 = B_2 - A_2 A_1$.

Algorithm 8 Preparation to get G_m

1: *Define* $Z_1 := A_1$
2: *Compute* $Z_2 := B_2 - A_2 A_1$
3: *Compute* $Z_3 := -A_3 Z_2 - B_3 Z_1$
4: **for** $j = 4, \ldots, m$ **do**
5: *Let* Z_1 *become* Z_2
6: *Let* Z_2 *become* Z_3
7: *Compute* $Z_3 := -A_j Z_2 - B_j Z_1$

The output of the algorithm is the matrix G_m. Memory requirements for the algorithm are:

- The $n \times n$ tridiagonal block matrices A_1 through A_m
- The $n \times n$ diagonal block matrices B_1 through B_m
- The $n \times n$ band matrices Z_1, Z_2 and Z_3

Complexity of calculations: The complexity of the calculations is as follows:

- Computing $-A_2 A_1$ and adding B_2: $2 \cdot 4^3 \cdot 3 \cdot n + 2 \cdot 4^3 \cdot n$ flops
- calculating $-A_3 Z_2$: $2 \cdot 4^3 \cdot 5 \cdot n$ flops
- calculating $-B_3 Z_1$: $2 \cdot 4^3 \cdot 3 \cdot n$ flops
- Letting j go from 4 to m:
  - calculating $-A_j Z_2$: $2 \cdot 4^3 \cdot (2(j-1)-1) \cdot n$ flops
  - calculating $-B_j Z_1$: $2 \cdot 4^3 \cdot (2(j-2)-1) \cdot n$ flops
  - adding previous two to each other can be done at the same time as calculating $-B_j Z_1$

Calculate order of total amount of flops:

$$\begin{aligned} &2 \cdot 4^3 \cdot \left(3n + n + 5n + 3n + \textstyle\sum_{j=4}^{m} \left((2(j-1) + 1 + 2(j-2) + 1) \cdot n\right)\right) \\ &= 2 \cdot 4^3 \cdot n \cdot \left(\textstyle\sum_{j=1}^{m-1} \left(2j + 1 + 2(j-1) + 1\right)\right) = n \cdot \textstyle\sum_{j=1}^{m-1} (4j) \\ &= 2 \cdot 4^3 \cdot 4n \cdot \frac{m(m-1)}{2} = 4^3 \cdot 2^2 (m^2 - m)n = 4^4 (m^2 - m)n \end{aligned}$$

Therefore, the algorithm to get G_m has a computational complexity of the order $4^4(m^2 - m)n$.

To get the LU decomposition of G_m, at most $4^3 \cdot 2m^2n$ flops are needed for G_m is a block band matrix with bandwidth m.

Now follows the final algorithm to solve the equation $Mx = y$ when the LU decomposition of G_m has been calculated.

Algorithm 9 Solving the linear equation

1: *Define* $v_1 := y_1$
2: *Compute* $v_2 := y_2 - A_2v_1$
3: *Compute* $v_3 := y_3 - A_3v_2 - B_3v_1$
4: **for** $j = 4, \ldots, k$ **do**
5: *Let* v_1 *become* v_2
6: *Let* v_2 *become* v_3
7: *Compute* $v_3 := y_j - A_jv_2 - B_jv_1$
8: *Compute* $x_1 := G_m^{-1}v_3$
9: *Compute* $x_2 := y_1 - A_1x_1$
10: **for** $j = 3, \ldots, m$ **do**
11: *Compute* $x_j := y_{j-1} - A_{j-1}x_{j-1} - B_{j-1}x_{j-2}$

The output of the algorithm is the block vector x. Memory requirements for the algorithm are:

- The block matrices A_1 through A_m
- The block matrices B_1 through B_m
- The LU decomposition of the block matrix G_m
- The $n \times 1$ size block vector y
- The n size block vectors v_1, v_2 and v_3

Complexity of calculations: The complexity of the calculations is as follows:

- Computing $-A_2v_1$ and adding y_2: $2 \cdot 4^2 \cdot 3 \cdot n + 2 \cdot 4^2 \cdot n$ flops
- calculating $-A_3v_2$: $2 \cdot 4^2 \cdot 3 \cdot n$ flops
- calculating $-B_3v_1$: $2 \cdot 4^2 \cdot n$ flops
- adding previous two to each other to y_3: $2 \cdot 4^2 \cdot n$ flops
- Letting j go from 4 to m:
  - calculating $-A_jv_2$: $2 \cdot 4^2 \cdot 3 \cdot n$ flops
  - calculating $-B_jv_1$: $2 \cdot 4^2 \cdot n$ flops

  - adding previous two to each other to y_3: $2 \cdot 4^2 \cdot n$ flops
- Computing G_m^{-1} by using LU decomposition: $4^3 \cdot 2mn$ flops
- Computing $-A_1 x_1$ and adding to y_1: $2 \cdot 4^2 \cdot 3 \cdot n + 2 \cdot 4^2 \cdot n$ flops
- Letting j go from 3 to m:
  - calculating $-A_{j-1} x_{j-1}$: $2 \cdot 4^2 \cdot 3 \cdot n$ flops
  - calculating $-B_{j-1} x_{j-2}$: $2 \cdot 4^2 \cdot n$ flops
  - adding previous two to each other to y_{j-1}: $2 \cdot 4^2 \cdot n$ flops

Calculate order of total amount of flops:

$$\begin{aligned} &2 \cdot 4^2 \cdot \left(4n + 5n + \textstyle\sum_{j=4}^{m}(3n+n+n) + 4mn + 3n + n + \sum_{j=3}^{m}(3n+n+n)\right) \\ &\quad = 2 \cdot 4^2 \cdot \left(4n + \textstyle\sum_{j=3}^{m}(5n) + 4mn + 4n + \sum_{j=3}^{m}(5n)\right) \\ &\quad = 2 \cdot 4^2 \cdot (4n + 5(m-2)n + 4mn + 4n + 5(m-2)n) \\ &\quad = 2 \cdot 4^2 \cdot (14mn - 12n) = 4^3 \cdot (7mn - 6n) \end{aligned}$$

Therefore, the algorithm to solve $Mx = y$ when having the LU decomposition of G_m has a computational complexity of the order $4^3 \cdot (7mn - 6n)$.

The LU decomposition of the $mn \times mn$ block matrix M with bandwidth m and block size 4×4 takes $4^3 \cdot 2(m)^2 \cdot mn = 4^3 \cdot 2m^3 n$ and the forwards and backwards solve costs $4^2 \cdot 2m^2 n$ flops. This means that the new solver is an order m faster in both ways than applying the LU decomposition directly to the matrix M.

7.4 Practice

However, in practice, problems appear when phenomena occur that are not worked out in theory before. One of these ideas is computer precision. In theory, every calculation can be done with infinite precision. In Practice, computers have finite precision.

The example that will be given have the same places nonzeros as in the matrices D_{L1}. The matrices are as follows when compared to the structure of the matrix D_{L1} seen in 3.2:

- On the places with S are 4×4 identity matrices multiplied by 3
- On the places with PP are 4×4 identity matrices multiplied by 2
- On the places with Q are 4×4 identity matrices multiplied by -10
- On the places with R are 4×4 identity matrices multiplied by 2

- On the places with T are 4×4 identity matrices

This matrix is diagonally dominant, which means that it has an inverse. And the places where there are T are identity matrices, thus the solver can be applied to ithe matrix.

For the test, let v be the vector where the element is the same as the index. The algorithm will try to solve $D_{L1}x = b$ with D_{L1} the matrix as mentioned above and b the matrix D_{L1} multiplied with the vector v. The resulting vector x should be such that $||x - \mathrm{v}||_2$ being of the order of the machine precision.

The code is written in MATLAB.

For $NY = 30$ and $ubk = 10$, the error $||x - \mathrm{v}||_2$ is $7.7268 \cdot 10^{-4}$. However, when $NY = 121$ and $ubk = 85$, which is in the later cases of the PARNASSOS program, $||x - \mathrm{v}||_2$ is equal to $1.9839 \cdot 10^{82}$. When $NY = 121$ and $ubk = 10$, $||x - \mathrm{v}||_2$ is equal to 0.0120. Lastly, with $NY = 30$ and $ubk = 85$, $||x - \mathrm{v}||_2$ is equal to $3.1858 \cdot 10^{82}$

From these tests, it can be seen that the greatest influence in the precision of the program is how large ubk is. The condition number of G_{ubk} is not easily controlled in size, which means that the equation $G_m x_1 = v_m$ is ill-conditioned.

This means that this problem must be solved if it occurs before this solver is potentially feasible.

8 Final comparison

The best options from all the batches are going to be repeated here again: The option 7blockdiag, the one with the LU decomposition and the ones with $ILU(0)$ and $ILU(1)$

amount of planes (g)	2	3	4	5
ttotsol	8.07	11.21	12.64	16.00
timeprec	3.22	3.35	3.36	3.38
ttotmv	7.63	10.85	12.39	15.90
timegm	22.03	30.30	33.84	43.13
ttotddotp	1.87	3.00	3.24	4.80
tcoeff	14.75	14.75	14.42	14.38
tturb	8.62	8.69	8.75	8.90
execttm	50.37	58.63	61.74	71.23
nummatvec	3588	2944	2333	2304
nnewton	360	240	180	144
	maximum residual of equations			
ξ-momentum	$5.465 \cdot 10^{-3}$	$6.880 \cdot 10^{-3}$	$7.873 \cdot 10^{-3}$	$9.093 \cdot 10^{-3}$
continuity	$2.968 \cdot 10^{-3}$	$3.915 \cdot 10^{-3}$	$3.792 \cdot 10^{-3}$	$5.614 \cdot 10^{-3}$
ζ-momentum	$2.123 \cdot 10^{-3}$	$2.173 \cdot 10^{-3}$	$2.999 \cdot 10^{-3}$	$2.775 \cdot 10^{-3}$
η-momentum	$1.428 \cdot 10^{-3}$	$1.549 \cdot 10^{-3}$	$1.997 \cdot 10^{-3}$	$2.002 \cdot 10^{-3}$

Table 9: Results for tests with 7blockdiag with total amount of planes varying from 2 to 5

amount of planes (g)	2	3	4	5
ttotsol	186.13	185.88	203.15	248.36
timeprec	544.17	539.99	537.93	537.84
ttotmv	4.73	5.43	6.41	8.03
timegm	756.58	752.52	769.09	816.32
ttotddotp	0.89	0.94	1.09	1.48
tcoeff	14.87	15.18	14.74	14.68
tturb	8.84	8.95	9.19	9.57
exectm	786.56	781.82	798.34	846.05
nummatvec	2160	1441	1178	1147
nnewton	360	240	180	144
	maximum residual of equations			
ξ-momentum	$5.473 \cdot 10^{-3}$	$6.881 \cdot 10^{-3}$	$7.887 \cdot 10^{-3}$	$9.101 \cdot 10^{-3}$
continuity	$2.968 \cdot 10^{-3}$	$3.916 \cdot 10^{-3}$	$3.803 \cdot 10^{-3}$	$5.608 \cdot 10^{-3}$
ζ-momentum	$2.122 \cdot 10^{-3}$	$2.172 \cdot 10^{-3}$	$2.998 \cdot 10^{-3}$	$2.776 \cdot 10^{-3}$
η-momentum	$1.427 \cdot 10^{-3}$	$1.549 \cdot 10^{-3}$	$1.997 \cdot 10^{-3}$	$2.004 \cdot 10^{-3}$

Table 10: Results for tests with block_fulli with total amount of planes varying from 2 to 5

amount of planes (g)	2	3	4	5
ttotsol	7.17	10.98	12.79	16.67
timeprec	3.84	3.99	4.02	4.03
ttotmv	9.82	14.57	16.26	21.25
timegm	25.85	37.51	42.03	55.52
ttotddotp	3.10	4.90	5.38	8.12
tcoeff	15.08	14.60	14.84	14.45
tturb	8.62	8.73	8.75	8.92
exectm	56.11	65.84	70.73	83.75
nummatvec	4659	3921	3028	3089
nnewton	360	240	180	144
	maximum residual of equations			
ξ-momentum	$5.457 \cdot 10^{-3}$	$6.885 \cdot 10^{-3}$	$7.813 \cdot 10^{-3}$	$9.114 \cdot 10^{-3}$
continuity	$2.968 \cdot 10^{-3}$	$3.925 \cdot 10^{-3}$	$3.744 \cdot 10^{-3}$	$5.655 \cdot 10^{-3}$
ζ-momentum	$2.122 \cdot 10^{-3}$	$2.172 \cdot 10^{-3}$	$2.998 \cdot 10^{-3}$	$2.782 \cdot 10^{-3}$
η-momentum	$1.427 \cdot 10^{-3}$	$1.550 \cdot 10^{-3}$	$1.995 \cdot 10^{-3}$	$2.005 \cdot 10^{-3}$

Table 11: Results for tests with $ILU(0)$ with total amount of planes varying from 2 to 5

amount of planes (g)	2	3	4	5
ttotsol	10.87	13.56	15.18	19.48
timeprec	7.63	7.55	7.54	7.57
ttotmv	7.56	11.08	12.59	16.20
timegm	29.46	37.19	41.30	52.37
ttotddotp	2.02	2.98	3.49	5.26
tcoeff	15.07	14.55	14.56	14.63
tturb	8.72	8.69	8.90	8.89
exectm	58.52	65.38	69.74	80.88
nummatvec	3588	2944	2333	2304
nnewton	360	240	180	144
	maximum residual of equations			
ξ-momentum	$5.465 \cdot 10^{-3}$	$6.880 \cdot 10^{-3}$	$7.873 \cdot 10^{-3}$	$9.093 \cdot 10^{-3}$
continuity	$2.968 \cdot 10^{-3}$	$3.915 \cdot 10^{-3}$	$3.792 \cdot 10^{-3}$	$5.614 \cdot 10^{-3}$
ζ-momentum	$2.123 \cdot 10^{-3}$	$2.173 \cdot 10^{-3}$	$2.999 \cdot 10^{-3}$	$2.775 \cdot 10^{-3}$
η-momentum	$1.428 \cdot 10^{-3}$	$1.549 \cdot 10^{-3}$	$1.997 \cdot 10^{-3}$	$2.002 \cdot 10^{-3}$

Table 12: Results for tests with $ILU(1)$ with total amount of planes varying from 2 to 5

The first similarity that appears is the amount of matrix vector multiplications, nummatvecs, in $ILU(1)$ is the same as in the option with 7blockdiag. This is because the options are in theory the same. However, $ILU(1)$ comes from the implementation of the more general case $ILU(N)$ and 7blockdiag is truly focused on making the $ILU(1)$ is good as possible.

When comparing the values for nummatvec of 7blockdiag and the preconditioner with the LU decomposition, The value from the latter is around half that of the former. This means that for the convergence of GMRES, the preconditioner with the LU decomposition is much better than the option 7blockdiag. However, the amount of fill in that is created in the process of an LU decomposition gives a heavy burden in the time it takes to make calculations with the preconditioner, as shown by the huge amount of time it takes to create the preconditioner, timeprec, and the amount of time the program takes to solve the equation with the preconditioner, ttotsol. Therefore, the option 7blockdiag far outperforms the option with the LU decomposition due to the huge burden of the fill in when compared to the increase in the convergence of GMRES.

When the new options are compared with the older option of 7blockdiag, the total execution time of one global iteration step has unfortunately not improved. The best performing preconditioner of these four is thus still 7blockdiag, followed by $ILU(0)$ or $ILU(1)$ and as last the preconditioner that used the LU decomposition.

The final graphs will contain the results of a test done for 500 global iteration steps from a blank start to finish of the best options for $g = 5$ with the same starting conditions as mentioned in section 12.1: 7blockdiag is displayed in figure 4 and $ILU(1)$ is displayed in figure 5. *dp* stands for the maximal absolute change in pressure before and after the global iteration. Same for *du*, *dv* and *dw* as change in velocity in the direction stated in section 2.2.1.

The spikes at 100, 200 and 300 global iterations can be explained by the fact that on those occasions, the computations went from a course grid to a more smoother grid. This results in enormous errors on the new nodes which are quickly sorted out.

Both figures are identical until global iteration 300, which makes sense due to the fact that 7blockdiag is a specialized version of $ILU(1)$. After which they slightly deviate from each other. This deviation most likely occurred due to the fact that while testing, an error occured at iteration 320 with $ILU(1)$. However, the starting conditions were saved and the test could be restarted at 320. This deviation still doesn not change the fact that the same convergence pattern can be seen in both figures after global iteration 330.

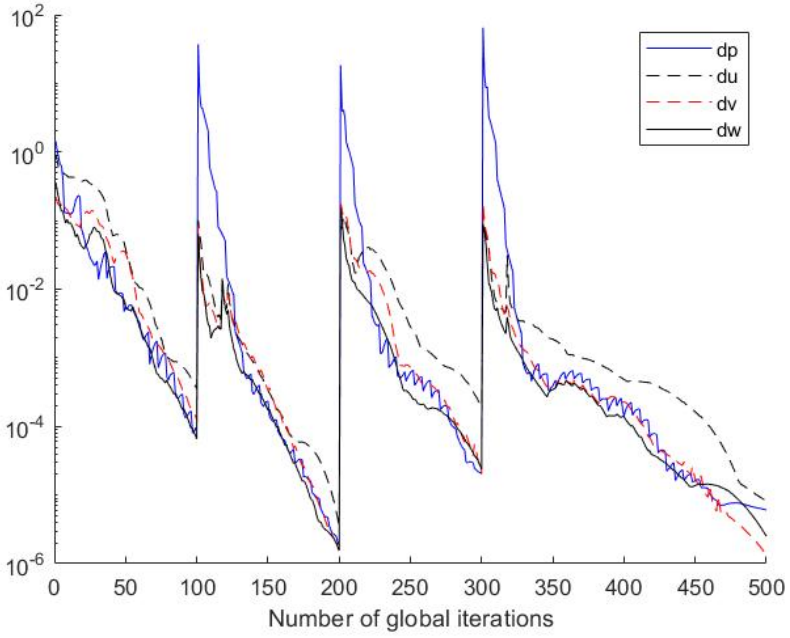


Figure 4: 500 global iterations with 7blockdiag

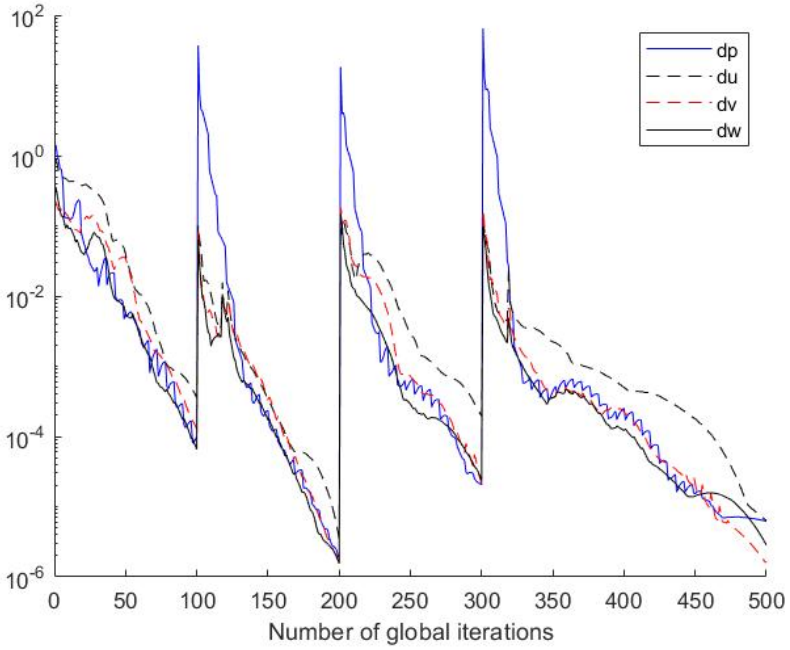


Figure 5: 500 global iterations with $ILU(1)$

9 Summary

This report tries to improve the preconditioner used in GMRES to decrease the execution time of PARNASSOS. To this end, a direct solver called the LU factorization is introduced to try and expand the possibilities for preconditioners in GMRES. This option has been tested against the older options for preconditioners of PARNASSOS, one of which is called 7blockdiag, in order to see if the new method improved the convergence of GMRES while keeping the extra time to create the preconditioner to a minimum. This, unfortunately, did not turn out to be a success.

The next option tried to reduce the amount of extra nonzero elements in comparison with the LU decomposition when creating the preconditioner by using $ILU(N)$. This option turns out to be much better than the LU decomposition. However, the option 7blockdiag is a specialized version of $ILU(1)$, therefore it is no shock to find that $ILU(1)$ is a worse option for preconditioner than 7blockdiag.

In the final option, a direct solver is created to solve linear equations with matrices that are similar to the matrices of interest on the main block diagonal. Although from the theory it can be concluded that the computational complexity of the new direct solver is lower than with the LU decomposition, one of the problems that the algorithm faces in practice is that a linear equation that needs to be solved in the algorithm could become ill conditioned when the full matrix becomes too large. Therefore, the solver can not yet be used in practice.

The conclusion from this report is that no preconditioner has been found that is better than the older options for a preconditioner in PARNASSOS

10 Propositions for future endeavors

This section will mention four ideas that can be explored in the future to ultimately achieve the original goal of this research: to create a better option for a preconditioner in PARNASSOS.

- The preconditioner option that used the LU decomposition was much better as a preconditioner for GMRES than the option 7blockdiag. Unfortunately, the amount of time that it took to perform the LU decomposition was much larger than the amount of time it spared with less GMRES iterations. Therefore, it is possible that a GPU implementation of the LU decomposition can improve the option as a new preconditioner.

- The new solver was in theory faster than the LU decomposition for large matrices, but in practice it couldn't solve for large matrices due to the equation $G_m x_1 = v_m$ becoming ill-conditioner. Therefore, a possibility is to try and solve for this problem while still keeping the improved computational complexity.

- As mentioned in section 6.3, $ILU(2)$ is a worse preconditioner than $ILU(1)$ and $ILU(3)$ is a worse preconditioner than $ILU(3)$. This flies in the face of the whole idea of $ILU(N)$, namely allow for more fill in to get a better preconditioner. On of the possible reasons is that this version of $ILU(N)$ is a too simplified version of what is described in section 6.1. Therefore, it is possible that with the new condition for $ILU(N)$ as allowing at most $N-2$ extra diagonals from the inner diagonals outward, that this version of $ILU(N)$ will lead to a better preconditioner when used.

- The file which contained the code for $ILU(N)$ may still have some hidden errors within them, thus an option is to go through the code one more time.

11 Bibliography

A, Giancarlo (2009). Reynolds-Averaged Navier-Stokes Equations for Turbulence Modeling. *Applied Mechanics Reviews, 62*

L. Eça and M. Hoekstra . Numerical Prediction of Scale Effects in Ship Stern Flows with Eddy-Viscosity Turbulence Models. In *Twenty-Third Symposium on Naval Hydrodynamics Applications to Ship Flow and Hull Form Design, Osaka, Japan,* September 2000.

Van der Ploeg, A, Starke, B and Veldhuis, C. Optimization of a chemical tanker with free-surface viscous flow computations. In the conference of *practical design of ships and other floating structures (PRADS), Changwon City, Korea,* October 2013.

Van der Ploeg, A, Hoekstra, M and Eça, L (2000). Combining accuracy and efficiency with robustness in ship stern flow computation. In *Twenty-Third Symposium on Naval Hydrodynamics Applications to Ship Flow and Hull Form Design, Osaka, Japan,* September 2000.

Saad, Y (2003). Iterative methods for sparse linear systems (2).

C. Vuik and D.J.P. Lahaye (2017). Scientific Computing (wi4201). Retrieved from http://ta.twi.tudelft.nl/nw/users/vuik/wi4201/

12 Appendices

12.1 Appendix A: Starting values tests

The *nosd_m_111.ini* file, which is one of the files that contain the starting values for the model, has a section called general. The following values are the ones used for the tests:

- MAXSWEEP: 2000
- IstepSweep: 99
- EpsGlobal : $5 \cdot 10^{-6}$
- Tolgmr: $2 \cdot 10^{-3}$
- maxlingmr: 40
- minlingmr: 5 (only exception 1 with LU when $g = 1$)
- ISTEPSweep: 49
- start_undist: 1
- begincomp: 0
- endcomp: 0

12.2 Appendix B: Conversion into band matrix

The following code is to insert a matrix D_{L1} for a certain constant $L1$ in its band form of LAPACK, which is stored in the matrix MB:

```
MB(:,:,L1)=0.d0
DO k=1,ubk
DO j=1,NY
DO column=1,N
          columnQ = N*NY*(k-1)+N*(j-1) + column
          MB(diagQ+1-column:diagQ+N-column,columnQ,L1)
&         = Q(1:N,column,j,k,L1)
          IF (k.GT.1) THEN
             MB(diagS+1-column:diagS+N-column,columnQ-N*NY,L1)
&            = S(1:N,column,j,k,L1)
          ENDIF
          IF (j.GT.1) THEN
             MB(diagPP+1-column:diagPP+N-column,columnQ-N,L1)
&            = PP(1:N,column,j,k,L1)
          ENDIF
          IF (k.LT.ubk) THEN
             MB(diagT+1-column:diagT+N-column,columnQ+N*NY,L1)
&            = T(1:N,column,j,k,L1)
          ENDIF
          IF (j.LT.NY) THEN
             MB(diagR+1-column:diagR+N-column,columnQ+N,L1)
&            = R(1:N,column,j,k,L1)
          END IF
ENDDO
ENDDO
ENDDO
```

With the constants

```
bw = N*(NY+1)-1
diagQ = 2*bw+1
diagS = diagQ+N*NY
diagPP = diagQ+N
diagR = diagQ-N
diagT = diagQ-N*NY
```

Where bw is the bandwidth of each of the matrices D_{L1}.