Norwegian University of Science and Technology
Department of Mathematical Sciences

TMA4205 Numerical Linear Algebra
Fall 2015

Semester project – part 2

Part 2 of the semester project focuses on different approaches for solving the linear algebraic system resulting from the discretization of Stokes equations. Namely, we consider the following boundary value problem:

$$\begin{aligned} -\nabla^2\mathbf{u} + \nabla p &= \mathbf{f}, && \text{in } \Omega = (0, L_x) \times (0, L_y),\\ \nabla\cdot\mathbf{u} &= 0, && \text{in } \Omega,\\ \mathbf{u} &= \mathbf{g}, && \text{on } \partial\Omega, \end{aligned} \tag{1}$$

or in component form

$$\begin{aligned} -(\partial^2_{xx} + \partial^2_{yy})u + \partial_x p &= f_x, && \text{in } \Omega,\\ -(\partial^2_{xx} + \partial^2_{yy})v + \partial_y p &= f_y, && \text{in } \Omega,\\ \partial_x u + \partial_y v &= 0, && \text{in } \Omega,\\ u &= g_x, && \text{on } \partial\Omega,\\ v &= g_y, && \text{on } \partial\Omega. \end{aligned} \tag{2}$$

Physically, the vector function $\mathbf{u} = (u, v)$ and the scalar function p describe the motion of a slow viscous incompressible liquid with a known velocity $\mathbf{g} = (g_x, g_y)$ on the boundary $\partial\Omega$ and with a force $\mathbf{f} = (f_x, f_y)$ acting upon it inside the domain Ω.

Furthermore, owing to the incompressibility of the flow (equation $\nabla\cdot\mathbf{u} = 0$) Gauss–Ostrogradsky theorem implies that

$$0 = \int_\Omega \nabla\cdot\mathbf{u} = \int_{\partial\Omega} \mathbf{n}\cdot\mathbf{u} = \int_{\partial\Omega} \mathbf{n}\cdot\mathbf{g}, \tag{3}$$

where $\mathbf{n}$ is the outwards facing normal on $\partial\Omega$. Thus the solutions only exist when the boundary data $\mathbf{g}$ satisfies the compatibility condition (3), in which case the flow velocity can be determined uniquely from (2), whereas the pressure is only defined up to an arbitrary additive constant.[1]

We will not describe possible discretizations of (2) in great detail. We utilize arguably the simplest possible approach, where the square domain Ω is subdivided into $n_x \times n_y$ small rectangular cells with sides $h_x = L_x/n_x$ and $h_y = L_y/n_y$ as shown in Fig. 1. This leads to $n_x n_y$ p-unknowns, $(n_x - 1)n_y$ u-unknowns, and $n_x(n_y - 1)$ v-unknowns. If we store the unknowns in vectors $U \in$

[1] Note, that the Stokes system only involves derivatives of the pressure, not the function itself.

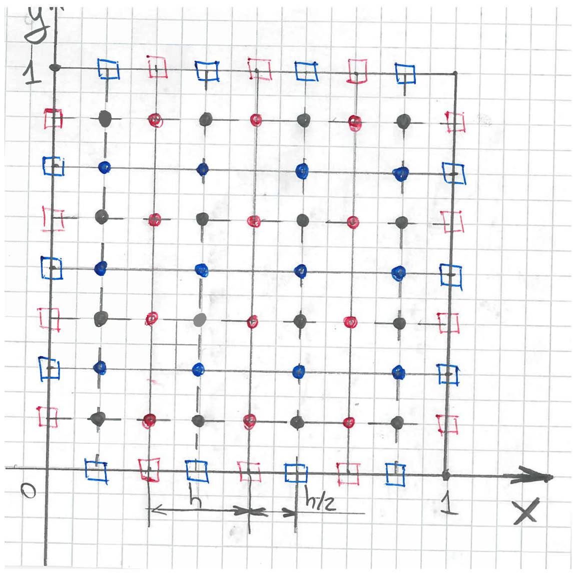

Figure 1: Discretization of the Stokes equations utilized in this project (staggered grid discretization); $L_x = L_y = 1$ and $n_x \times n = 4 \times 4$ cells are shown. Black, red, and blue dots show the locations of p, u, and v unknowns, respectively. Red and blue squares show the locations at which the boundary conditions g_x and g_y are evaluated; red and blue dots is where f_x and respectively f_y is evaluated.

$\mathbb{R}^{(n_x-1)n_y}$, $V \in \mathbb{R}^{n_x(n_y-1)}$, $P \in \mathbb{R}^{n_x n_y}$ as follows:

$$U \approx \begin{pmatrix} u(h_x, h_y/2) \\ u(2h_x, h_y/2) \\ \vdots \\ u((n_x-1)h_x, h_y/2) \\ u(h_x, 3h_y/2) \\ \vdots \\ u((n_x-1)h_x, (n_y-1/2)h_y) \end{pmatrix}, \quad V \approx \begin{pmatrix} v(h_x/2, h_y) \\ v(3h_x/2, h_y) \\ \vdots \\ v((n_x-1/2)h_x, h_y) \\ v(h_x/2, 2h_y) \\ \vdots \\ v((n_x-1/2)h_x, (n_y-1)h_y) \end{pmatrix},$$

$$P \approx \begin{pmatrix} p(h_x/2, h_y/2) \\ p(3h_x/2, h_y/2) \\ \vdots \\ p((n_x-1/2)h_x, h_y/2) \\ p(h_x/2, 3h_y/2) \\ \vdots \\ p((n_x-1/2)h_x, (n_y-1/2)h_y) \end{pmatrix},$$

and similarly for the equation numbers, then the resulting linear algebraic system has a block structure

$$\underbrace{\begin{pmatrix} A_x & 0 & B_x \\ 0 & A_y & B_y \\ B_x^{\mathrm{T}} & B_y^{\mathrm{T}} & 0 \end{pmatrix}}_{=:\mathcal{A}} \begin{pmatrix} U \\ V \\ P \end{pmatrix} = \begin{pmatrix} F_x \\ F_y \\ G \end{pmatrix}, \tag{4}$$

where $A_x \in \mathbb{R}^{(n_x-1)n_y \times (n_x-1)n_y}$ and $A_y \in \mathbb{R}^{n_x(n_y-1) \times n_x(n_y-1)}$ are symmetric positive definite matrices resulting from the discretization of $-(\partial^2_{xx} + \partial^2_{yy})u$ and $-(\partial^2_{xx} + \partial^2_{yy})v$; $B_x \in \mathbb{R}^{(n_x-1)n_y \times n_x n_y}$ and

$B_y \in \mathbb{R}^{n_x(n_y-1)\times n_x n_y}$ are matrices resulting from the discretization of $\partial_x p$ and $\partial_y p$. Note that the matrix on the left hand side of (4) is singular as pressures P are only determined up to a constant.

Matlab/Octave code `stokes.m` for generating the left and the right hand side for (4) is available for downloading from the course's wiki page. *In addition to answering the questions described in this document, you have to verify and document the correctness of each solution approach; see Appendix A for an example.*

$\boxed{1}$ **Unpreconditioned CG iteration in the pressure space.**

In this part, we consider the following approach for solving (4). First, we express U and V in terms of P from the first and second block-equations of (4): $U = A_x^{-1}(F_x - B_x P)$, $V = A_y^{-1}(F_y - B_y P)$. We then substitute these expressions into the last equation to obtain a smaller system for the pressure unknowns only:

$$\underbrace{[B_x^{\mathrm{T}} A_x^{-1} B_x + B_y^{\mathrm{T}} A_y^{-1} B_y]}_{=:S} P = B_x^{\mathrm{T}} A_x^{-1} F_x + B_y^{\mathrm{T}} A_y^{-1} F_y - G. \tag{5}$$

a) Given that both A_x and A_y are symmetric and positive definite (SPD), show that S is symmetric and positive semi-definite.

S is not positive definite; in fact it is singular and $\ker S = \ker[B_x^{\mathrm{T}}, B_y^{\mathrm{T}}]^{\mathrm{T}} = \operatorname{span}\langle e\rangle$, where $e \in \mathbb{R}^{n_x n_y}$ is a vector of all ones (you do not have to prove this; simply use this as a given fact in the project). This is yet another manifestation of the fact that pressures are only determined up to a constant.

b) Show that the system $SP = b$ admits a solution if and only if $e^{\mathrm{T}} b = 0$.

c) Show that $S_\alpha := S + \alpha e e^{\mathrm{T}}$ is SPD (hence non-singular) for any $\alpha > 0$. Further, for any $b \in \mathbb{R}^{n_x n_y}$ such that $e^{\mathrm{T}} b = 0$ show that $S_\alpha P = b \implies SP = b$.

d) Implement a matrix-vector product routine $P \mapsto S_\alpha P$, which can be used inside a CG iteration. *Do not form A_x^{-1}, A_y^{-1}, S, ee^{T}, or S_α!*[2] You should pre-compute Cholesky factorizations of A_x, A_y and perform backward-forward substitutions when necessary.
Note: the choice of α affects the condition number of S_α. From here on use $\alpha = 1/(nxny)^2$ in the implementation.
Present numerical evidence that the condition number of S_α remains bounded with respect to the mesh refinement by studying the dependence of the number of unpreconditioned CG iterations needed to solve the linear system with this matrix in the left hand side.

For fine discretizations of Stokes equations the accurate computation of a matrix-vector product $P \mapsto SP$, which requires solving auxiliary algebraic systems with matrices A_x and A_y at every iteration becomes a major computational burden. Therefore, an alternative approach is desirable.

$\boxed{2}$ **Block-preconditioned FGMRES iteration.**

[2] I repeat: do not form A_x^{-1}, A_y^{-1}, S, ee^{T}, or S_α!

a) Show that solving the linear system in $\boxed{1}$ **c), d)** is equivalent to solving the following modification of (4):

$$\underbrace{\begin{pmatrix} A_x & 0 & B_x \\ 0 & A_y & B_y \\ B_x^{\mathrm{T}} & B_y^{\mathrm{T}} & -\alpha e e^{\mathrm{T}} \end{pmatrix}}_{=:\mathscr{A}_\alpha} \begin{pmatrix} U \\ V \\ P \end{pmatrix} = \begin{pmatrix} F_x \\ F_y \\ G \end{pmatrix}. \tag{6}$$

b) Compute a symbolic block-LU factorization of the matrix $\mathscr{A}_\alpha$ defined in (6), that is, find the unknown blocks in the representation

$$\mathscr{A}_\alpha = \underbrace{\begin{pmatrix} I & 0 & 0 \\ L_{21} & I & 0 \\ L_{31} & L_{32} & I \end{pmatrix}}_{=:\mathscr{L}} \underbrace{\begin{pmatrix} U_{11} & U_{21} & U_{31} \\ 0 & U_{22} & U_{32} \\ 0 & 0 & U_{33} \end{pmatrix}}_{=:\mathscr{U}} \tag{7}$$

The block-triangluar matrix $\mathscr{U}$ is an ideal right preconditioner for GMRES, which is establieshed in the following few steps:

c) Determine the spectrum of the preconditioned matrix $\mathscr{A}_\alpha \mathscr{U}^{-1}$.

d) Show the following equality for the preconditioned matrix $\mathscr{A}_\alpha \mathscr{U}^{-1}$:

$$(\mathscr{A}_\alpha \mathscr{U}^{-1} - I)^2 = 0. \tag{8}$$

e) Show that GMRES applied to $\mathscr{A}_\alpha \mathscr{U}^{-1}$ converges in at most 2 iterations.

f) Describe, in detail, the computation of the matrix-vector product $\mathscr{U}^{-1} W$ for some given block vector $W = [W_U^{\mathrm{T}}, W_V^{\mathrm{T}}, W_P^{\mathrm{T}}]^{\mathrm{T}}$. Allowed "elementary steps" in the description are matrix-vector multiplications involving S_α and blocks of $\mathscr{A}_\alpha$, or linear system solves involving these matrices/blocks.

We will now replace the ideal preconditioner $\mathscr{U}$ with a more practical version. In particular, we replace all inverse matrices involved in the computation found in the previous step with a suitable preconditioner. For example, in $\boxed{1}$ **d)** we have learned that S_α may be relatively efficiently preconditioned with the identity matrix. We will use an incomplete Cholesky preconditioner for A_x, A_y (`doc ichol` in Matlab).

The resulting approximation to $\mathscr{U}^{-1}$ will be called $\tilde{\mathscr{U}}^{-1}$ in the subsequent discussion.

g) Implement the matrix-vector multiplication with $\tilde{\mathscr{U}}^{-1}$ without forming any inverse or dense matrices. Use the implemented subroutine as the right preconditioner in FGMRES applied to (6).[3] Verify your implementation. Study the efficiency of this preconditioning strategy on a sequence of refined meshes: compare the number of unpreconditioned/preconditioned FGMRES iterations needed to obtain some desired accuracy. Compare with the approach taken in $\boxed{1}$. Note: you may need to experiment with different drop tolerances for the incomplete Cholesky preconditioner/restart parameter of FGMRES to obtain a reasonable convergence speed.

h) One could try to obtain further improvements to the preconditioner constructed in the previous step by using a (very) approximate incomplete Cholesky preconditioned CG solve in place of a plain incomplete Cholesky preconditioner for the blocks A_x, A_y. Implement this version of the preconditioner (you can use `pcg` available in Matlab). Does this strategy decrease the overall solution time, when compared with the previous step?

[3] Two implementations of FGMRES, modified Gram-Schmidt (`fgmres_mgs.m`) and Householder reflections (`fgmres_h.m`) based, are available for downloading from the course's wiki page.

We will now eliminate the last non-scalable part from our block-preconditioning strategy, that is, the incomplete Cholesky based approximation to A_x^{-1}, A_y^{-1}.

$\boxed{3}$ Geometric multigrid preconditioner for the Laplacian of velocity.

a) Implement a geometric multigrid V-cycle for solving the linear systems $A_x U = b_u$, $A_y V = b_v$. (One can use the same implementation for both systems, by replacing the roles of x and y and reordering the unknowns/equations.) As input, the program should take the initial guess, the right-hand side, the number of levels (or grids), the number of pre-smoothings, and the number of post-smoothings. Use the direct solver (backslash) to solve the problem on the coarsest level; under-relaxed Jacobi iteration as a smoother; and linear interpolation to transfer information between grids. *Note: in the selected discretization scheme the right hand side of the algebraic system is formed as* $h_x h_y \mathbf{f}(x_i, y_j)$*, where* $\mathbf{f}(x_i, y_j)$ *is a given function. Therefore if you use injection as the coarsening operator, the coarsened residual* $r2h$ *should be further multiplied with* $4 = (2hx \times 2hy)/(hx \times hy)$. You should not explicitly form the matrices representing the inter-grid transfers.

It is advantageous to break the program down into several parts. The main routine could for instance look like this:

```
function U = mgv_u(U0, rhs, Lx,Ly, nx,ny, nu1,nu2, level, max_level)
%
% U = mgv_u(U0, rhs, N, nu1, nu2, level, max_level) performs one
% multigrid V-cycle for the system A_x U = rhs
%
% input:  U0        - initial guess
% rhs       - right-hand side
% Lx, Ly    - dimensions of the domain
% nx, ny    - U0 and rhs are vectors of length (nx-1)ny
% nu1       - number of pre-smoothings
% nu2       - number of post-smoothings
% level     - current level
% max_level - total number of levels
%
% output: U         - numerical solution

   % FORM A_x (nx, ny, Lx, Ly)
   if level == max_level
      % direct  solve
      U = A_x \ rhs;
   else
      % pre-smooth, e.g.
      U   = jacobi(A_x, U0, rhs, 2 / 3, nu1);
      rh  = rhs - A_x * U;
      r2h = restriction_u(rh, nx,ny);
      e2h = mgv_u(zeros((nx/2-1)*(ny/2),1), r2h, ...
                  Lx, Ly, nx/2, ny/2, nu1, nu2, level + 1, ...
                  max_level);
      eh  = interpolation_u(e2h, nx/2,ny/2);
```

```
    U   = U + eh;
    U   = jacobi(A_x, U, rhs, 2 / 3, nu2);
  end
```

Before utilizing this algorithm within a block-preconditioner framework of [2] it is a good idea to make sure that the multigrid implementation works as expected both as a solver and a preconditioner for the Laplace problem. You can use the script `laplace_uv.m` (analogous to `stokes.m`) for verification purposes.

b) Utilize one or several multigrid V-cycles as a preconditioner for A_x, A_y in the block-preconditioning strategy of [2]. That is, replace the incomplete Cholesky-based preconditioner with the one based on multigrid. Thus $z_u = A_x^{-1} r_u$ is approximated as

```
z_u = zeros((nx-1)*ny, 1);
for j=1:Niter,
  z_u = mgv_u(z_u, r_u, Lx, Ly, nx, ny, ...
            nu1, nu2, 1, max_level);
end
```

Verify the implementation of this strategy. Compare its performance with that of the preconditioner used in [2] **g)**–**h)**.

A Verification

We consider a so-called Kovasznay flow benchmark:

```
function [u,v,p,fx,fy]=kovasznay
lambda  = -1;
u    = @(x,y) 1-exp(lambda*(x-0.5)).*cos(2*pi*y);
v    = @(x,y) (lambda/2/pi)*exp(lambda*(x-0.5)).*sin(2*pi*y);
p    = @(x,y) 0.5*(exp(2*lambda*(x-0.5)));
d2u  = @(x,y) (-lambda^2 + 4*pi^2)*exp(lambda*(x-0.5)).*cos(2*pi*y);
d2v  = @(x,y) (lambda^3/2/pi - 2*pi*lambda)*exp(lambda*(x-0.5)).*sin(2*pi*y);
px   = @(x,y) lambda*exp(2*lambda*(x-0.5));
py   = @(x,y) zeros(size(x)).*zeros(size(y));
fx   = @(x,y) -d2u(x,y) + px(x,y);
fy   = @(x,y) -d2v(x,y) + py(x,y);
```

The values of u, v on the boundary are then used as Dirichlet boundary conditions on the flow. The problem is solved on a sequence of refined meshes and the error between the solution to the discretized system and the analytical solution evaluated at grid points is measured and recorded. For our discretization, if the linear algebraic system is solved accurately enough, the error in velocity approximation should decay as $O(h^2)$ and in the pressure component as $O(h)$, see Fig. 2.

For more details see `stokes.m`

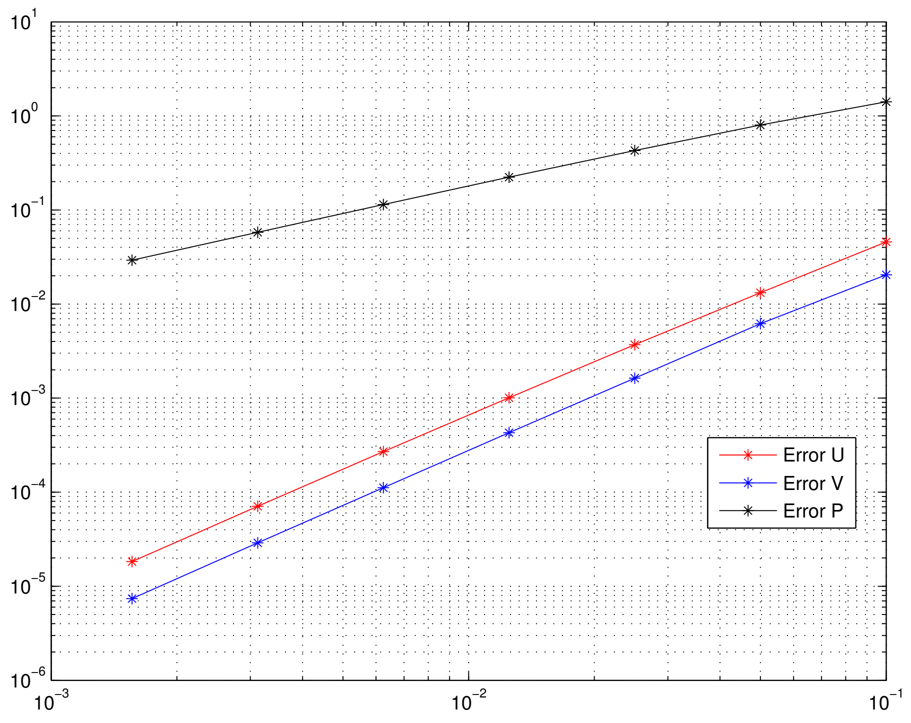

Figure 2: Discretization error decay as a function of grid size.