Efficient solvers for incompressible fluid flows in geosciences

A. Amirbekyan[1] L. Gross[2]

(Received 8 October 2008; revised 16 October 2008)

Abstract

Saddle point problems involving large systems of linear equations arise in a wide variety of applications in computational science and engineering. A variety of solvers have been developed for these type of problems typically with specific applications in mind. We focus on saddle point problems as they arise from incompressible fluid flow problems in geosciences. They are characterized through a spatially variable viscosity when modeling temperature dependencies (for example, in Earth mantel convection models) or moving material interfaces (for example, in subduction zones simulation and numerical volcano models). We overview some of the iterative techniques used and discuss suitable preconditioning techniques. We discuss the implementation of the schemes using the python module Escript and compare the efficiency of these schemes when applied to convection models on a parallel computer.

http://anziamj.austms.org.au/ojs/index.php/ANZIAMJ/article/view/1449 gives this article, © Austral. Mathematical Soc. 2008. Published November 3, 2008. ISSN 1446-8735. (Print two pages per sheet of paper.)

Contents

1 Introduction

Finite element and finite difference discretizations of the Navier–Stokes equations for incompressible flow lead to equations of saddle point type, which is a solution of the following operator problem for $u \in V$ and $p \in Q$ with suitable Hilbert spaces V and Q:

$$\begin{bmatrix} A & B \\ B^* & 0 \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix} \tag{1}$$

where A is coercive, self-adjoint linear operator in V, B is a linear operator from Q into V and B^* is the adjoint operator of B. f and g are given elements from V and Q respectively. In most cases, equation (1) is given in the form

$$a(v, u) + b(v, p) = \langle f, v \rangle, \tag{2}$$

$$b(u, q) = \langle g, q \rangle. \tag{3}$$

for all $v \in V$ and $q \in Q$ where $\langle , \rangle$ denotes the scalar product in V and Q. Here $b(u, q) = \langle u, Bq \rangle$ and $a(u, v) = \langle Au, v \rangle$. If b meets the Ladyzhenskya–Babuška–Brezzi[1] (LBB) condition, the linear problem (1) has a unique solution (u, p).

We are particularly looking for suitable methods to solve the Stokes problem with variable viscosity. In this case we have $V = H_0^1(\Omega)^d$ (H_0^1 is an appropriate Sobolev space), $Q = L_0^2(\Omega)$ (set of square integrable L^2-functions on measure space Ω) and

$$a(v, u) = \int_\Omega \eta(v_{i,j}u_{i,j} + v_{i,j}u_{j,i})\, d\Omega \tag{4}$$

$$b(v, q) = \int_\Omega v_{i,i} q\, d\Omega \tag{5}$$

where η is the spatial dependent viscosity with $\eta \geq \eta_{\min} > 0$. Here we use the Einstein summation convention [10] for convenience. This type of problems plays a key role in geoscience applications. For instance, in models for convection in the Earth's mantel the viscosity becomes a function of the temperature T in the form

$$\eta = \eta_0 \exp\left[a\left(\frac{1}{1+T} - \frac{2}{3}\right)\right] \tag{6}$$

where $\eta_0 = 1$ is the viscosity for $T = \frac{1}{2}$. The constant a is called the Arrhenius number. For the case of the Earth a takes the value of 22 [6, 8]. This value for a produces steep viscosity gradients providing a big challange for any solver applied to the saddle point problem (1). Notice that $a = 0$ defines the case of constant viscosity. In this context the external force f is

$$\langle f, v \rangle = \int_\Omega \mathrm{Ra}\, T v_2\, d\Omega \tag{7}$$

[1] This condition essentially says that velosity and pressure spaces cannot be chosen arbitrary, the link between them, known as LBB or *inf-sup* condition, is necessary to guarantee the stability of finite element approximate solution.

where Ra is the Rayleigh number. For the Earth one sets $\mathrm{Ra} = 10^6$. In the case that η is constant, the operator A can be simplified to a multiple of the Laplacean operator. This case has been extensively studied in the past [3, 4, 9]. We investigate how the results for the case of constant viscosity can be applied to the case of spatially variable viscosity.

2 Uzawa solver

One possible approach, referred to as the Uzawa scheme [1, pp.154–158] for solving the saddle point problem (1) is to eliminate the velocity v from the problem. This is possible if A is invertible. In this case one gets $u = A^{-1}(f - Bp)$ which when inserted into the second equation leads to the problem

$$Sp = B^* A^{-1} f. \tag{8}$$

where $S := B^* A^{-1} B$ is the Schur complement of A. As the Schur complement is symmetric and positive definite, the problem (8) is solved iteratively using preconditioned conjugate gradient (PCG) [7] method with the standard inner product in L^2. Section 3 discusses the question of a suitable preconditioner P_S for the Schur complement. Once p has been calculated, v is recovered in a postprocessor step as $u = A^{-1}(f - Bp)$. The residual r_k in the kth iteration step is

$$r_k = Sp_k - B^* A^{-1} f = B^* A^{-1}(Bp_k - f) = -B^* u_k\,, \tag{9}$$

with $u_k = A^{-1}(f - Bp_k)$. So representing the residual as the pair $r_k = (u_k, -B^* u_k)$, and then inspecting the first component of the current residual when terminating the iteration through a stopping criteria, will save the execution of the postprocessing. When implementing the PCG one needs to provide a function which returns for a given p an increment to the residual. This is calculated in the form

$$\begin{array}{llrcl} \text{(a)} & \text{Solve} & Az & = & Bp\,, \\ \text{(b)} & \text{Solve} & q & = & -B^* z\,, \end{array} \tag{10}$$

where the tuple (z, q) is returned.

3 Preconditioner for Schur complement

In the Stokes equations, the Schur complement operator S can be preconditioned by $1/\eta$, where η is the viscosity [4]. In this investigation a constant viscosity is assumed which allows simplification of the operator A to the Laplacian operator using the incompressibilty condition. The important difference for us is that in convection models η is not a constant, but spatially dependent. Unfortunately, this generalization does not allow us to simplify the operator A to the Laplacian operator using the incompressibilty condition which is a key in the theoretical investigations. The nature of the spatial dependency of η affects the overall performace of the model. In particular, a steep gradient of η will lead to a large deviation from the case of constant viscoscity in which $1/\eta$ provides a suitable preconditioner for S.

In our tests we select the preconditioner $p = P_S q$ of the Schur complement by solving

$$\frac{1}{\tilde{\eta}} p = q \tag{11}$$

where we consider two choices for $\tilde{\eta}$ namely

- spatially dependent viscosity: $\tilde{\eta} = \eta$
- constant average viscosity: $\tilde{\eta} = (\min \eta + \max \eta)/2$

In the case of constant viscosity both cases coincide. Notice that in this case we do not simplify the theoretical operator A to the Laplacian operator.

4 Implementation

We use the *escript* environment to implement the convection code. In this section we briefly outline the basic idea of *escript* [5] and show how to implement the evaluation step for the Schur complement (10). The *escript* module

is designed to implement PDE based models in *python*. It uses PDE based terminology providing an abstraction layer for spatial discretization methods. Its key component is a class used to define steady, linear partial differential equations which are solved using a C/C++ library [2]. The coefficients of the PDE are defined through expressions which are evaluated by the *escript* library. The *escript* library is parallelized for both OpenMP and MPI using the the data distribution used by the underlying PDE solver library.

The general form of the PDE in *escript* for an unknown vector valued spatial function u_i is

$$-(A_{ijkl}u_{k,l} + B_{ijk}u_k)_{,j} + C_{ikl}u_{k,l} + D_{ik}u_k = -X_{ij,j} + Y_i \ . \tag{12}$$

The coefficients A, B, C, D, X and Y are functions of their location in the domain, in particular they may depend on solutions of other PDEs, previous time steps or non-linear iteration steps. Moreover, natural boundary conditions are of the form

$$n_j(A_{ijkl}u_{k,l} + B_{ijk}u_k) + d_{ik}u_k = n_jX_{ij} + y_i \, , \tag{13}$$

where y and d are given functions. Notice that A, B and X are already used in the PDE (12). To set values of u_i to r_i on certain locations of the domain one can define constraints of the form

$$u_i = r_i \text{ where } q_i > 0 \ , \tag{14}$$

where q_i is a given function defining a positive value through the locations where the constraint is applied. With these tools it is very straight forward to implement the evaluation of the Schur complement S (10) within the Uzawa scheme. The following *python* function implements this step:

```
from escript import *
from escript.linearPDEs import LinearPDE
def evalS(dom, eta, p):
```

```
    v_pde=LinearPDE(dom)
    id=identityTensor4(dom)
    v_pde.setValue(A=eta*(id+swap_axes(id,1,2)), \
                           X=-p*kronecker(dom))
    p_pde=LinearPDE(dom)
    p_pde.setReductionOn()
    z=v_pde.getSolution()
    p_pde.setValue(D=1, Y=-div(z))
    q=p_pde.getSolution()
    return q
```

This function returns the pressure increment `q`. The class `LinearPDE` provides an interface to the PDE defined by (12)–(14). The corresponding values of PDE coefficients are set via the `setValue` method call. The argument `dom` is an *escript* `Domain` object defining the domain of the PDE including information on the discretization to be used. `eta` is representing the viscosity η which may be constant or a spatial function represented as an *escript* data object. The call of the `setReductionOn` method of `p_pde` switches on the usage of a reduced polynomial order for the pressure approximation as required to meet the requirements of the LBB condition. In a similar fashion one implements the application of the preconditioner P_A from the Schur complement in (11):

```
from escript import *
from escript.linearPDEs import LinearPDE
def evalP_S(dom, eta, p):
   pde=LinearPDE(dom)
   pde.setReductionOn()
   pde.setValue(D=1/eta, Y=p)
   return pde.getSolution()
```

It would be more efficient to keep a copy of the instances of `LinearPDE` class and to reuse them in the `evalS` and `evalP_S` calls. This allows for the

potential reuse of information such a preconditioners. This is implemented using a *python* class. As discussed in Section 2, it is advantageous to represent the residual in PCG using the pair $(\nu, -B^*\nu)$ where ν is the current velocity approximation. In *python* this is implemented using standard PCG code and overloading algebraic operations. In this case `evalS` returns the pair `(z,q)`. Additional to the saddle point problem for pressure and velocity the convection model requires the solution of the time-dependent temperature advection-diffusion problem. In *escript* and *finley* a algebraic flux correction scheme is used which is not discussed here.

5 Experiments

In the first test we investigate the efficiency of the Uszawa scheme when applied to convection problem. As a measure we use the number of inner iteration steps needed to reach a fixed tolerance (see Figure 1). As a stopping criterion we use the L^2 of the diveregence of the velocity relative to its H^1 norm. For solving the convection problems we proceed as follows.

- Initialize all constant parameters.
- For each time step, using the saddle point solver, compute (ν, p).
- Update temperature.[2]
- Go to the next time step.

In the following figures outer iterations refer to the loop in time steps, whereas inner iterations refer to the iterations of the saddle point solver. Table 1 shows the average number of inner iterations required for each step

[2] In the case of variable viscosity this operation will result in an update of viscosity as well.

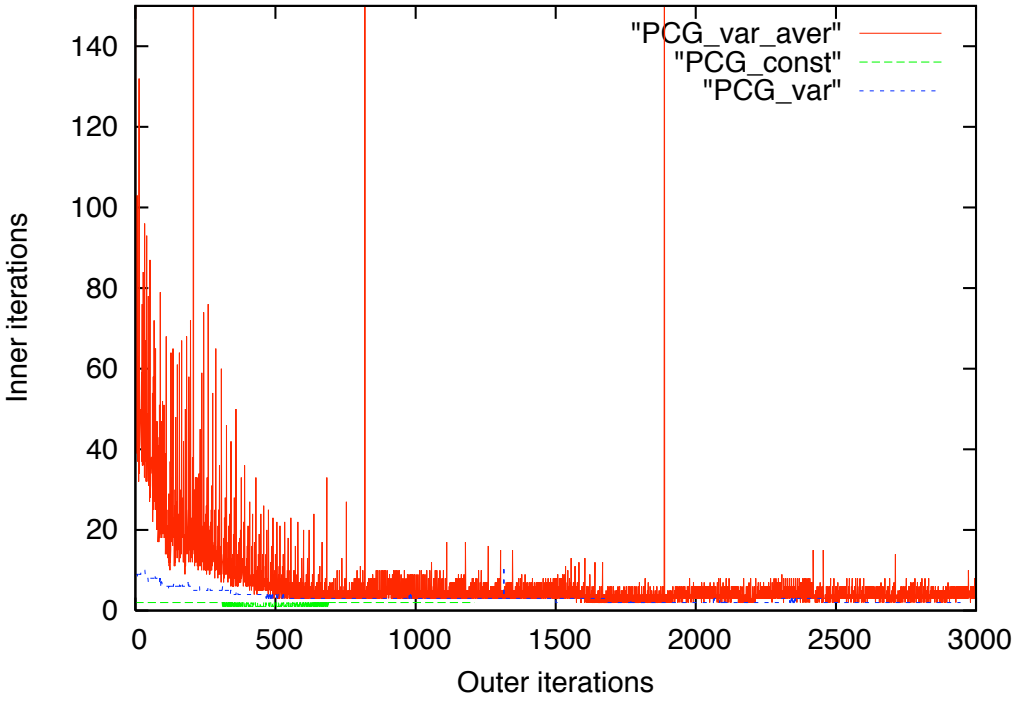

FIGURE 1: Comparison of inner iterations for the cases of constant viscosity (PCG_const), spatially dependent viscosity (A=22), but with constant average viscosity preconditioning (PCG_var_aver) and spatially dependent viscosity, with spatially dependent preconditioning (PCG_var).

TABLE 1: Average number of inner iterations with 95% confidence intervals.

PCG spatially dependent viscosity	5 ± 3
PCG constant average viscosity	22 ± 50
PCG constant viscosity	2 ± 1

of saddle point solver. We compared only the first 500 outer iteration steps, because after that usually each outer iteration requires two inner iterations.

These experiments clearly confirm that using $1/\eta$ as an approximation for the Schur complement is a very effective way to achive fast convergancy in iterative solvers for saddle point problems. Note that in both cases for constant and variable viscosities we achived very fast convergence. We tested this approach also with other solvers, such as GMRES, and we got similar convergence. However, for the case of average viscosity preconditioning we observed slow convergence (see Figure 1 and Table 1).

In Section 2 each inner iteration involves finding a solution of $\mathsf{Az} = \mathsf{Bp}$, where A sparse positive definite matrix. Once this solution is found, we use it for finding the velocity and pressure. One can use any suitable solver for solving $\mathsf{Az} = \mathsf{Bp}$; we use PCG solver. But the important question is how exact do we need to solve $\mathsf{Az} = \mathsf{Bp}$ for our whole saddle point solver in order to converge in each step? Thus, in the next experiments we observed the behavior of the saddle point solver depending on the sub-tolerance factor for $\mathsf{Az} = \mathsf{Bp}$.

Table 2 shows how tolerance affects the first time step (of the outer loop). The tests for the next 100 steps show similar figures. We notice here that for different mesh sizes the tolerance dependancy is similar. Thus, for presenting the elapsed time versus tolerance graph we show only one of them, for instance, for the mesh $16 \times 64 \times 16$ (see Figure 2). If the sub-tolerance exceeds the value 0.1 then the inner iterations do not converge any more.

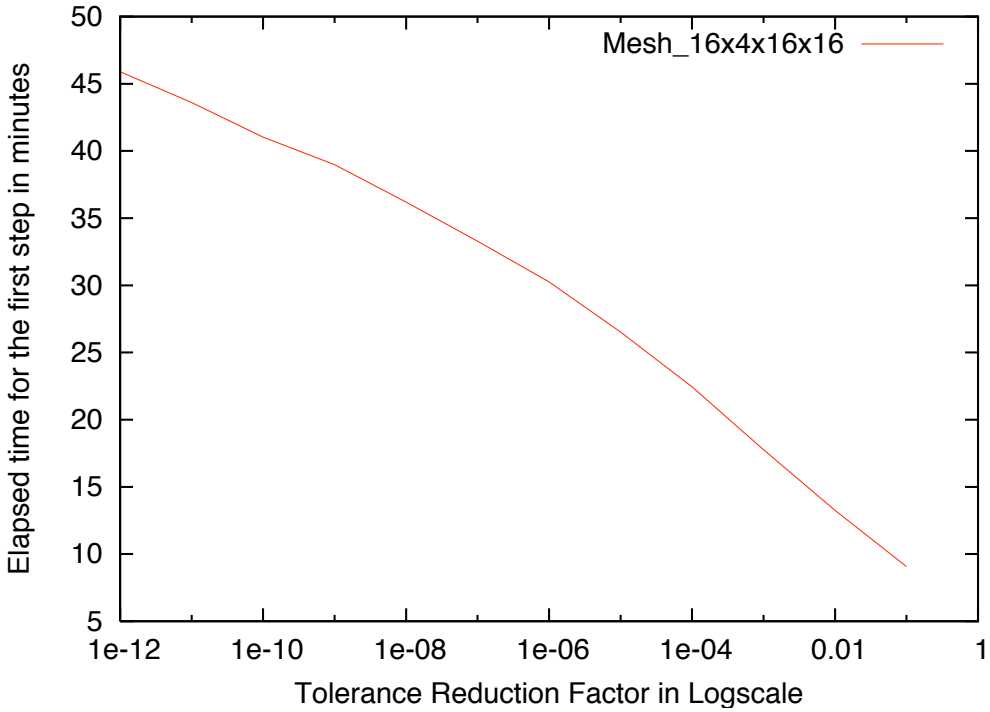

FIGURE 2: Elapsed time versus tolerance.

TABLE 2: Elapsed time in seconds for the first time step for various meshes.

Sub-tolerance	$8 \times 32 \times 8$	$16 \times 64 \times 16$	$24 \times 96 \times 24$
10^{-1}	59.41	544.68	2447.81
10^{-2}	50.23	795.20	3402.54
10^{-3}	75.23	1066.35	5218.70
10^{-4}	93.39	1346.93	6480.15
10^{-5}	109.17	1591.44	7823.06
10^{-6}	121.87	1814.94	8882.58
10^{-7}	133.55	1997.24	9755.85
10^{-8}	144.25	2171.93	10604.85
10^{-9}	153.92	2338.95	11382.37

TABLE 3: Average number of inner iterations with 95% confidence intervals.

PCG spatially dependent viscosity (A=22)	5 ± 3
PCG spatially dependent viscosity (A=11)	4 ± 2
PCG constant viscosity (A=0)	2 ± 1

The next experiment investigated how the Arrhenius number affects the convergence behavior of the solver. We have already presented the cases where Arrhenius number $\mathtt{a} = 0$ (constant viscosity) and $\mathtt{a} = 22$ (variable viscosity). The interesting question will be how $\mathtt{a} = 11$ affects the solver? Table 3 presents the experimental results. Here again we compare only the first 500 outer iteration steps.

6 Conclusion

Approximations and preconditioning techniques is one of the main techniques to speed up iterative solvers. For earth mantel convection problems where we observe variable viscosity, we were able to show empirically that $1/\eta$ can be used as a suitable preconditioner for the Schur complement. We also presented a way of implemetation using the Python based PDE modeling environment *Escript*.

Acknowledgements This work uses of infrastructure provided through the Auscope National Collaborative Research Infrastructure Strategy with funding from the Australian Commonwealth, the Queensland State Goverment and the University of Queensland.

References

[1] K. Arrow, L. Hurwicz, and H. Uzawa. *Studies in linear and nonlinear programming*. Stanford University Press. Stanford, CA, 1958. C192

[2] M. Davies, L. Gross, and H.-B. Muhlhaus. Scripting high performance earth systems simulations on the sgi altix 3700. In *Proceedings of the 7th International Conference on High Performance Computing and Grid in the Asia Pacific Region, Tokyo, Japan*, pages 244–251. IEEE, 2004. doi:10.1109/HPCASIA.2004.1324041 C194

[3] A. de Niet and W. Wubs. Two preconditioners for saddle point problems in fluid flows. *International Journal for Numerical Methods in Fluids*, 54:355–377, 2007. Published online 19 December 2006 in Wiley InterScience. doi:10.1002/fld.1401 C192

[4] H. Elman and D. Silvester. Fast nonsymmetric iterations and preconditioning for Navier-Stokes equations. *SIAM Journal on Scientific Computing.*, 17(1):33–46, 1996. doi:10.1137/0917004 C192, C193

[5] L. Gross, L. Bourgouin, A.J. Hale, and H.-B. Mhlhaus. Interface modeling in incompressible media using level sets in escript. *Physics of The Earth and Planetary Interiors*, 163(1–4):23–34, 2007. doi:10.1016/j.physletb.2003.10.071 C193

[6] M. Kameyama, A. Kageyama, and T. Sato. Multigrid iterative algorithm using pseudo-compressibility for three-dimensional mantle convection with strongly variable viscosity. *Journal Computational Physics*, 206(1):162–181, 2005. doi:10.1016/j.jcp.2004.11.030 C191

[7] A. V. Knyazev. A preconditioned conjugate gradient method for eigenvalue problems and its implementation in a subspace. In *International Ser. Numerical Mathematics, v. 96, Eigenwertaufgaben in Natur- und Ingenieurwissenschaften und ihre numerische Behandlung, Oberwolfach, 1990.*, pages 143–154. Birkhauser Basel, 1991. C192

[8] L. Moresi, F. Dufour, and H.-B. Muehlhaus. Mantle convection modeling with viscoelastic/brittle lithosphere: Numerical methodology and plate tectonic modeling. *Journal on Pure and Applied Geophysics*, 159(10):2335–2356, 2002. doi:10.1007/s00024-002-8738-3 C191

[9] D. Silvester, H. Elman, D. Kay, and A. Wathen. Efficient preconditioning of the linearized Navier-Stokes equations for incompressible flow. *J. Comput. Appl. Math.*, 128(1-2):261–279, 2001. Numerical analysis 2000, Vol. VII, Partial differential equations. doi:10.1016/S0377-0427(00)00515-X C192

[10] W Weisstein, E. Einstein summation. From MathWorld—A Wolfram Web Resource. `http://mathworld.wolfram.com/EinsteinSummation.html` C191

Author addresses

1. **A. Amirbekyan**, Earth System Sciences Computational Center, The University of Queensland, AUSTRALIA.
mailto:artak@uq.edu.au

2. **L. Gross**, Earth System Sciences Computational Center, The University of Queensland, AUSTRALIA.
mailto:l.gross@uq.edu.au