UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Scientific Computing (COC)

Lab 3: Linear Systems in PETSc

Year 2021-2022

Contents

Introduction

This lab unit consists of 3 sessions and in each of them we will employ PETSc with a different strategy to solve a system of linear equations

$$Ax = b. \tag{1}$$

The next table lists the starting material to carry out each of the sections:

Session 1	Direct methods	`prac3a.c`
Session 2	Iterative methods	`prac3b.c`
Session 3	Preconditioners	-

1 Direct methods

The program `prac3a.c` loads a matrix A from a binary file and solves the system (1) with the vector $b = [1, 1, \ldots, 1]^T$. For this, it creates a `KSP` object as discussed in class.

Exercise. Carry out the following steps:

1. Modify the program so that it selects the LU factorization, using `KSPSetType()` and `PCSetType()`.
2. Add the necessary code so that it computes and prints the residual norm $\|b - Ax\|_2$.
3. Add the necessary code so that it prints the time spent in the `KSPSolve()` operation.
4. Try the program with the following matrices: `dawson5`, `nasasrb`, and `af23560`.

1.1 Fill-in

To know the details of the algorithm being employed, you can use the `-ksp_view` option, for instance:

```
$ ./prac3a -f /labos/asignaturas/ETSINF/coc/bwm200.petsc -ksp_view
KSP Object: 1 MPI processes
  type: preonly
  maximum iterations=10000, initial guess is zero
  tolerances:  relative=1e-05, absolute=1e-50, divergence=10000
  left preconditioning
  using NONE norm type for convergence test
PC Object: 1 MPI processes
  type: lu
    LU: out-of-place factorization
    tolerance for zero pivot 2.22045e-14
    matrix ordering: nd
    factor fill ratio given 5, needed 2.00754
      Factored matrix follows:
        Mat Object: 1 MPI processes
          type: seqaij
          rows=200, cols=200
          package used to perform factorization: petsc
          total: nonzeros=1598, allocated nonzeros=1598
          total number of mallocs used during MatSetValues calls =0
            not using I-node routines
  linear system matrix = precond matrix:
  Mat Object: 1 MPI processes
    type: seqaij
    rows=200, cols=200
    total: nonzeros=796, allocated nonzeros=796
    total number of mallocs used during MatSetValues calls =0
      not using I-node routines
```

In this subsection, we are going to experiment with different orderings. The example is using `matrix ordering: nd`, that is, *nested dissection*. For this ordering, it states that the fill ratio has been `2.00754`, that is, the number of nonzero elements after the factorization is about twice as much as the original one.

Exercise. The available orderings are the ones discussed in the previous lab unit (`MatOrdering`). Repeat the executions indicating different orderings, in particular: `natural`, `rcm`, `nd`, and `qmd`. To avoid modifying the code and recompiling every time, the ordering can be set by adding the command-line option:

```
    -pc_factor_mat_ordering_type nd
```

In each case, write down the fill ratio and observe how there is a correlation between it and the computing time. Usually, the natural ordering is always the worst one. Observe also that the RCM method hardly reduces fill-in (despite minimizing the bandwidth).

1.2 Cholesky

The first two matrices (`dawson5` and `nasasrb`) are symmetric, and hence it is possible to use the Cholesky method (remember that in PETSc the `cholesky` solver may be either Choleksy or LDL^T depending on the matrix being positive-definite or not).

Exercise. Compare the executions of LU and Cholesky. Take into account that for a fair comparison, both methods must use the same ordering (note that the default ordering in LU is `nd` while in Cholesky the default is `natural`). The theoretical cost of Cholesky ($\frac{1}{3}n^3$) is half that of LU ($\frac{2}{3}n^3$). However, it is not faster because the implementation of Cholesky is less optimized.

2 Iterative methods

In this session we are going to solve systems of linear equations with iterative methods.

2.1 Stationary methods

The program `prac3b.c` solves the Poisson equation $-\nabla^2 u = f$ using the finite-difference method on a square regular mesh with $n+1$ subintervals in each side. The f function is constant and the boundary conditions are $u = 0$ on all sides of the domain.

Exercise. We are going to solve the system of equations $Ax = b$ by means of iterative methods. Modify the program so that it prints the residual norm and the number of iterations that the method has carried out. The code should also call `KSPGetConvergedReason()` to check that the method has finished correctly and print a warning otherwise (alternatively, you can run with the command-line option `-ksp_converged_reason`).

Exercise. For a size $n = 50$, compare the following methods (use a textual or graphical monitor to observe the convergence):

1. Richardson. Try with the default value of α. Is convergence achieved? Next, try with a value smaller than 1, for instance $\alpha = 0.2$. Try also with the command-line option `-ksp_richardson_self_scale` so that it uses an optimal value of α computed automatically.

2. Jacobi. Remember that in PETSc the Jacobi method is available as a combination of Richardson (with $\alpha = 1$) with the `jacobi` preconditioner.

3. SOR. Execute SOR with $\omega = 1$ (default value), which is equivalent to the Gauss-Seidel method. Is the number of iterations reduced with respect to Jacobi? Next, try incrementing ω with values between 1 and 2. The number of iterations should decrease, until reaching a certain value in which they increase again.
   Note: due to how SOR is implemented in PETSc, it will not compute the residual norm at each step and therefore the method will not stop until the maximum number of iterations is reached. This can be avoided simply by adding a monitor (which forces computation of the residual norm).

Note: in all executions, it is convenient to use the `-ksp_view` command-line option to make sure that the intended options have been set correctly.

Exercise. For some of the previous methods, run with a relative tolerance of 10^{-11} and check that the obtained residual norm is reduced consistently.

2.2 Krylov methods

In this section we will use Krylov methods without preconditioner (we will study preconditioners in the last session of this lab unit). In order to solve without a preconditioner we must use the option:

```
-pc_type none
```

Exercise. Try with the different Krylov methods studied in class with the previous problem. Is the number of iterations reduced with respect to the stationary methods? Which is the method that requires less iterations in this case?

Activate the graphical monitor to observe how the different methods converge. Notice that convergence is more irregular than in the case of the stationary methods.

Exercise. An iterative method doing less iterations does not imply that it is faster. Submit parallel executions of a larger problem, for example $n = 150$ with 4 processes. Check the time for `KSPSolve()`.

Exercise. Some Krylov methods have tunable options. Try modifying the `restart` parameter in GMRES, and observe how the number of iterations change. Does the execution time also get reduced?

3 Preconditioners

In this session we come back to the matrices loaded from a file. You can use for instance `finan512`, `shallow_water1`, and `bwm200`. Use the program from the first session, `prac3a.c`. We will now obtain the solution computed with preconditioned Krylov methods.

In order to carry out the comparison, we will use the Bi-CGStab method combined with several preconditioners. In all cases, use the `-log_view` option to see which is the cost of each step of the resolution. In particular, we will focus our attention on the operations `KSPSolve()`, `PCSetUp()` and `PCApply()`. Roughly, the first one will be equal to the sum of the other two (plus the `MatMult()` operations and other). Establish the relationship between the number of times that the `PCApply()` operation has been called with the number of iterations of the iterative method.

Exercise. For each of the three matrices, compare the number of necessary iterations with the `jacobi`, `sor`, and `ilu` preconditioners. In the last two cases, try several values of the ω parameter (`sor`) or the level of fill (`ilu`). Is there a correlation between the execution time and the number of iterations of the method?

Exercise. Solve the problems with the `bjacobi` preconditioner for different number of blocks of the preconditioner, from 1 up to 16. Observe how the effectiveness of the preconditioner gets reduced, since the number of iterations grow. Execute in parallel with different number of processes (`bjacobi` with one block per process) and analyze the obtained speedup. [Note: use a case whose execution time is significant.]