UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Scientific Computing (COC)

Lab 2: Data Classes in PETSc

Year 2021-2022

Contents

Introduction

This lab unit consists of 3 sessions and deals with the data classes available in PETSc. The following table lists the starting material to carry out the work for each section:

Session 1	Vectors `Vec`	`prac2a.c`, `prac2b.c`
Session 2	Matrices `Mat`	`prac2c.c`
Session 3	Ordering and partitioning	-

The programs must include the header files with the definition of the used classes. For example, in the first session we will have to include `petscvec.h`.

1 Vectors `Vec`

In the first session of this unit, we restrict ourselves to work with vectors only.

1.1 Basic usage

In this subsection, we are going to work with the provided example `prac2a.c`. This example computes the integral of a certain function in the interval $[0, 1]$. For this, it discretizes the interval with 1 million points. Modify the code so that the number of points can be specified in the command line, by using function `PetscOptionsGetInt()` in a similar way as was done in the previous session.

The program approximates the integral by means of the *trapezoidal rule*, as illustrated in Figure 1, that is, adding the areas of the trapezoids defined by two consecutive mesh points x_k and x_{k+1}, and the respective

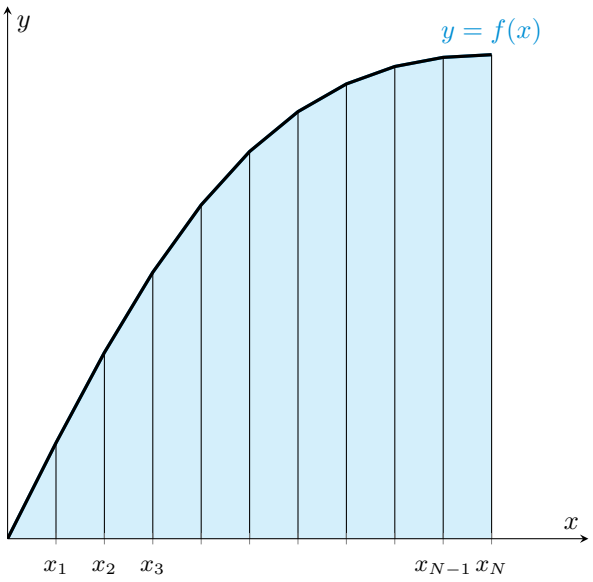


Figure 1: Geometric interpretation of an integral.

values of the function at those points, $f(x_k)$ and $f(x_{k+1})$. For an interval $[a, b]$ divided in N subintervals, the expression for the trapezoidal rule is

$$\int_a^b f(x)dx \approx h\left(\frac{1}{2}f(x_0) + f(x_1) + f(x_2) + \ldots + f(x_{N-1}) + \frac{1}{2}f(x_N)\right),$$

where $x_0 = a$, $x_N = b$ and $h = (b - a)/N$. In the program, this formula is implemented as

$$h\sum_{i=0}^{N} f(x_i) - h\big(0.5f(x_0) + 0.5f(x_N)\big).$$

Observe how vector `xend` is created. Which values will it have? To check it, insert the following line after the `VecAssemblyEnd()` operation:

```
ierr = VecView(xend,NULL);CHKERRQ(ierr);
```

1.2 Gram-Schmidt Orthogonalization

The code of Figure 2 shows the implementation in PETSc of the classical Gram-Schmidt algorithm. To invoke the function two arrays of `k` vectors must be passed, which are previously allocated with `VecDuplicateVecs()`.

The objective of the `prac2b.c` program is to initialize a set of vectors `v` with random values, and then call function `CGS` that implements the classical Gram-Schmidt algorithm to obtain a set of orthonormal vectors `q`. The generation of random vectors can be done as shown in Figure 3.

The program is incomplete, you must carry out the following changes:

- Enable it to receive the number of vectors `k` in the command line, as well as their dimension `n` (use `PetscOptionsGetInt()`).
- Create the arrays of vectors `v` and `q` by means of `VecDuplicateVecs()`.
- Destroy these vectors with `VecDestroyVecs()` when they are no longer necessary.

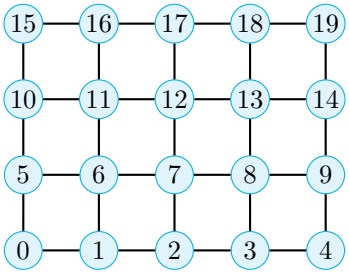
```
PetscErrorCode CGS(PetscInt k,Vec *v,Vec *q)
{
  PetscErrorCode ierr;
  PetscScalar    *R;
  PetscReal      norm;
  PetscInt       i,j;
  Vec            w;

  PetscFunctionBegin;
  ierr = VecDuplicate(v[0],&w);CHKERRQ(ierr);
  ierr = PetscMalloc1(k*k,&R);CHKERRQ(ierr);
  ierr = PetscArrayzero(R,k*k);CHKERRQ(ierr);
  for (j=0;j<k;j++) {
    ierr = VecCopy(v[j],w);CHKERRQ(ierr);
    for (i=0;i<j;i++) {
      ierr = VecDot(v[j],q[i],&R[i+j*k]);CHKERRQ(ierr);
      ierr = VecAXPY(w,-R[i+j*k],q[i]);CHKERRQ(ierr);
    }
    ierr = VecNorm(w,NORM_2,&norm);CHKERRQ(ierr);
    R[j+j*k] = norm;
    ierr = VecCopy(w,q[j]);CHKERRQ(ierr);
    ierr = VecScale(q[j],1.0/R[j+j*k]);CHKERRQ(ierr);
  }
  ierr = PetscFree(R);CHKERRQ(ierr);
  ierr = VecDestroy(&w);CHKERRQ(ierr);
  PetscFunctionReturn(0);
}
```

Figure 2: Implementation of the classical Gram-Schmidt algorithm.

```
  PetscRandom rnd;
  ierr = PetscRandomCreate(PETSC_COMM_WORLD,&rnd);CHKERRQ(ierr);
  ierr = PetscRandomSetFromOptions(rnd);CHKERRQ(ierr);
  for (i=0;i<k;i++) {
    ierr = VecSetRandom(v[i],rnd);CHKERRQ(ierr);
  }
  ierr = PetscRandomDestroy(&rnd);CHKERRQ(ierr);
```

Figure 3: Generation of random vectors.




```
ierr = MatGetOwnershipRange(A,&Istart,&Iend);CHKERRQ(ierr);
for (k=Istart;k<Iend;k++) {
  i = k/n; j = k-i*n;
  MatSetValue(A,k,k,4.0,INSERT_VALUES);
  if (i>0) MatSetValue(A,k,k-n,-1.0,INSERT_VALUES);
  if (i<m-1) MatSetValue(A,k,k+n,-1.0,INSERT_VALUES);
  if (j>0) MatSetValue(A,k,k-1,-1.0,INSERT_VALUES);
  if (j<n-1) MatSetValue(A,k,k+1,-1.0,INSERT_VALUES);
}
```

Figure 4: Example of mesh of 4×5 points (left) and a code that inserts the coefficients (right). All nodes are interior nodes (boundary nodes are not shown).

Do the following exercises:

1. Implement a function that prints the scalar products of each q_i with all the rest, to check the orthogonality. The result must be the identity matrix. Take into account that rounding errors will make that instead of an exact zero, the product of two orthogonal vectors will give a number of order ε, the machine precision ($\varepsilon \approx 2.2 \cdot 10^{-16}$ in double precision).
2. Modify function `CGS` so that it uses multi-vector operations, `VecMDot()` and `VecMAXPY()`. **Hint**: note that you cannot do a sign change in `VecMAXPY()` as it was done with `VecAXPY()`, a possible alternative is to change the sign explicitly by overwriting the appropriate elements of `R` with a loop.
3. Compare the parallel execution times of both versions, for sufficiently large values of `k` and `n`.

2 Matrices `Mat`

In this session we are going to work with matrices created in a program as well as loaded from a file.

2.1 Matrix assembly

As discussed in theory, the Poisson equation

$$-\nabla^2 u = f \tag{1}$$

can be discretized by means of the finite-difference method, resulting in a system of linear equations $Ax = b$. The pattern and values of matrix A will depend on the number of points in the mesh, as well as the used ordering. Suppose a mesh of `n` $\times$ `m` points with natural ordering, as illustrated in Figure 4.

The figure shows on the mesh the global indices corresponding to each point. The code shown in the figure inserts the coefficients in the appropriate positions of the matrix. For each global index `k` the code computes the coordinates of row `i` and column `j` within the mesh (using integer division), and checks whether such point has four neighbors or less.

Exercise. Complete the `prac2c.c` program so that it gets the values of `n` and `m` via the command line (by default their value must be 20), creates a matrix object `Mat` with `m*n` rows and `m*n` columns, and inserts the coefficients according to the code of Figure 4 (include also the matrix assembly, that was omitted in the figure). Use the different forms of `-mat_view` to obtain information about the matrix. Note: the graphical options such as `-mat_view draw` can only be used when running directly on the front-end and in this case it is recommended to add the option `-draw_pause -1` so that the plot is displayed until the window is clicked with the mouse.

Exercise. In the previous program, create two vectors with `MatCreateVecs()` and fill one of them to all 1's. Later, multiply it by the matrix and show the resulting vector. Interpret the shown values.

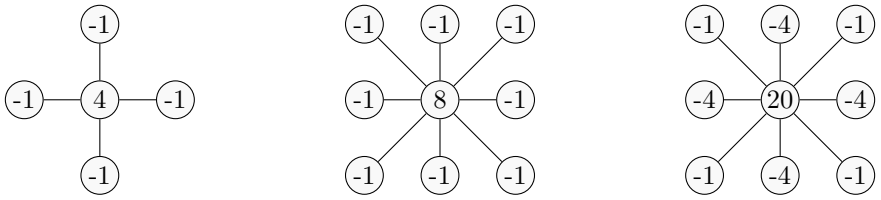


Figure 5: Discretization rules (*stencil*) corresponding to the 5-point formula (left) and with 9 points (center and right).

Table 1: Test matrices in PETSc's binary format. The value κ is an approximation of the condition number.

Name	Size	Symm	κ	Description
af23560	23,560	no	$2 \cdot 10^4$	NACA 0012 airfoil
bcircuit	68,902	no	$8 \cdot 10^4$	circuit with parasitics
bwm200	200	no	$2 \cdot 10^3$	transport interaction of chemical solutions
copter2	55,476	yes (indef)	$5 \cdot 10^5$	helicopter rotor blade
dawson5	51,537	yes (indef)	$7 \cdot 10^5$	flap, part of actuator system on airplane
finan512	74,752	yes	$3 \cdot 10^1$	portfolio optimization
ford2	100,196	yes	$3 \cdot 10^8$	car surface mesh
Ga3As3H12	61,349	yes	$1 \cdot 10^7$	quantum chemistry problem
ibm_matrix_2	51,448	no	$3 \cdot 10^8$	semiconductor device
nasasrb	54,870	yes	–	shuttle rocket booster
pkustk13	94,893	yes	$7 \cdot 10^6$	pky symmetric stiffness, machine element
rajat26	51,032	no	$2 \cdot 10^8$	circuit simulation
shallow_water1	81,920	yes	3.5	shallow water modelling
ship_003	121,728	yes	–	ship structure
struct3	53,570	yes	$1 \cdot 10^8$	finite element matrix
xenon1	48,600	no	–	complex zeolite, sodalite crystals

The employed discretization is the commonly used 5-point formula, represented in Figure 5 (left). To obtain a higher-order approximation, it is necessary to add more points and modify the coefficients.

Exercise. Modify the previous program so that it uses any of the 9-point rules shown in Figure 5. Compare the statistics of the matrix with the previous case.

Exercise. Execute the program modifying the storage format. Select `baij`, and observe how when showing the matrix we can see some zeros if the block size is larger than 1 (this can be changed with option `-mat_block_size <bs>`). Try also the `sbaij` type and observe the difference.

2.2 Loading from a file

In the directory `$HOME/asigDSIC/ETSINF/coc` of the `kahan` cluster there are several sparse matrices stored in PETSc's binary format. The matrices belong to the SuiteSparse matrix collection[1], coming from real applications, and their properties are summarized in Table 1.

The objective of this subsection is to load some of these matrices from a PETSc program to be able to obtain information from them. Figure 6 shows the required calls to initialize a matrix loaded from a file.

Exercise. From the program of the previous subsection, create a new program that loads the matrix from a file, in such a way that the name of the file can be specified at the command line (`PetscOptionsGetString`).

[1] `https://sparse.tamu.edu`.

```
char        filename[PETSC_MAX_PATH_LEN];
PetscViewer viewer;
ierr = PetscViewerBinaryOpen(PETSC_COMM_WORLD,filename,FILE_MODE_READ,&viewer);CHKERRQ(ierr);
ierr = MatCreate(PETSC_COMM_WORLD,&A);CHKERRQ(ierr);
ierr = MatSetFromOptions(A);CHKERRQ(ierr);
ierr = MatLoad(A,viewer);CHKERRQ(ierr);
ierr = PetscViewerDestroy(&viewer);CHKERRQ(ierr);
```

Figure 6: Fragment of code that loads a matrix from a PETSc binary file.

Add the necessary instructions so that the program performs several matrix-vector products and shows the elapsed time. Execute the program for different number of processes and study the achieved speedup. Do you think that parallel performance depends on the matrix pattern?

3 Ordering and partitioning

In this session we explore PETSc's functionality related to the different orderings studied in class.

Exercise. Modify the program from the previous subsection so that we can specify an ordering. The program must invoke `MatGetOrdering()` with the indicated type of ordering and permute the matrix with `MatPermute()`. Observe the nonzero pattern of the matrix in each case for several matrices from directory `$HOME/asigDSIC/ETSINF/coc`. To see the pattern of the permuted matrix we have two options:

- Save the permuted matrix in a file and load it later with the same program indicating the option `-mat_view draw`.
- Execute with option `-mat_view_ordering draw` (in this case, PETSc calls `MatPermute()` internally and the permuted matrix is visualized).

Exercise. Modify the program so that it carries out the partitioning of the adjacency graph and redistributes the matrix according to the new ordering, following the steps indicated in class. If a small test case is run on `kahan`'s front-end with a few processes, then it will be possible to visualize how the matrix pattern changes and how the nonzero elements tend to cluster around the diagonal blocks.

Perform a parallel performance study to compare the cases when the partitioning is used or not. For this, use several matrices from directory `$HOME/asigDSIC/ETSINF/coc`. As in the previous session, perform several matrix-vector products to do the comparison. In this case, instead of measuring the times directly, use the `-log_view` option to extract the execution statistics.