Linear systems

Vec, Mat, KSP to solve linear systems in PETSc

Ed Bueler, UAF

26 January 2016

preamble: partial differential equations (PDEs) coverage

- I have no chance of "covering" the field of PDEs
  - *I will not try!*
- PETSc is not a magic bullet for problems
- consider alternatives:
  1. FEM packages (FENICs, deal.II, libmesh, firedrake)
     - easier to use, easier for complex geometry
     - *PETSc is often under the hood*
  2. solvers for classes of problems (e.g. clawpack for purely-hyperbolic systems)

PDEs done in this seminar

- the PDEs in this seminar are in three broad classes:
  1. linear elliptic or systems (e.g. Poisson equation, advection-diffusion, Stokes)
  2. nonlinear elliptic (e.g. p-Laplacian, obstacle)
  3. time-dependent of various types (e.g. heat, shallow water, porous medium)
     - will focus on implicit time-stepping on these problems
- these classes are poorly-defined . . . my book is a grab-bag
- generally the case for my choices:

PDE problems generate large linear and nonlinear systems after discretization (e.g. finite difference or finite element)

PDEs generate linear/nonlinear systems

- linear elliptic PDE generates linear system
  - for example: Poisson equation in Chapters 3, 9, and 11

$$-\nabla^2 u = f \qquad \longrightarrow \qquad A\mathbf{u} = \mathbf{b}$$

- linear elliptic PDE generates linear system
  - for example: p-Laplacian equation in Chapter 5

$$-\nabla \cdot \left(|\nabla u|^{p-2} \nabla u \right) = f \qquad \longrightarrow \qquad \mathbf{F}(\mathbf{u}) = 0$$

linear systems from PDEs: big and sparse

One might say that if N is very large, it is probably an approximation to infinity. Trefethen & Bau, page 244

- discretized PDEs generate linear/nonlinear systems as large as we can handle
  - dimensions $N = 10^8$ (or more) are common
- . . . but partial derivatives are *local* interactions
  - so matrix structure is *sparse* with lots of zeros
- job for today: solve small, fixed-size linear system
- job for next week: solve larger linear systems, and transition to PDE-generated systems

recall how to get started with a PETSc example

- check that environment variables are set to a valid PETSc installation:

```
$ echo $PETSC_DIR
$ echo $PETSC_ARCH
```

- compile and run a code:

```
$ cd c/ch2/
$ make vecmatksp
$ ./vecmatksp
$ mpiexec -n 4 ./vecmatksp
```

PETSc object class 1: Vec

- to create, fill, and show a parallel `Vec` called `b` we do this:

```
Vec    b;
int    j[4] = {0, 1, 2, 3};              # global indices
double v[4] = {11.0, 7.0, 5.0, 3.0};     # values

VecCreate(PETSC_COMM_WORLD,&b);
VecSetSizes(b,PETSC_DECIDE,4);
VecSetFromOptions(b);                    # why needed?
VecSetValues(b,4,j,v,INSERT_VALUES);
VecAssemblyBegin(b);                     # why needed?
VecAssemblyEnd(b);
VecView(b,PETSC_VIEWER_STDOUT_WORLD); # print @ stdout
VecDestroy(&b);
```

notice PETSc is object-oriented C . . . sort of

- C is not officially an object-oriented language
- *but* PETSc data types act that way
- we interact with `Vec b` only through function calls
  - internal representations are hidden
  - these functions are essentially "methods" of `Vec`
- however, "inheritance" and other C++ concepts are not in C
- in fact `Vec` is really a pointer to a `struct` . . . but I will *not* look inside

result from VecView(): serial

```
$ ./prog
Vec Object: 1 MPI processes
  type: seq
11.
7.
5.
3.
```

a vector in serial: picture

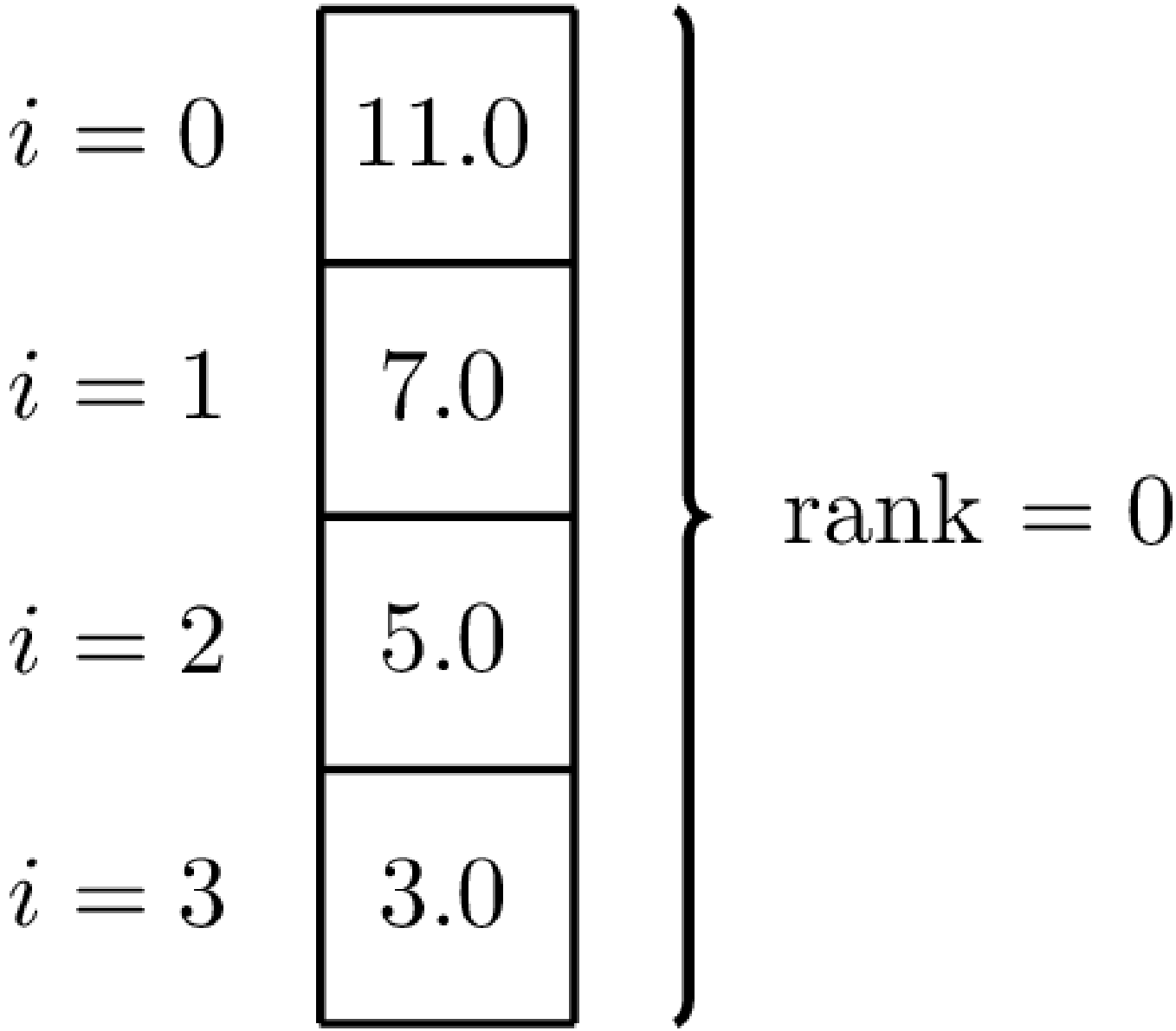

Figure 1:layout of Vec in serial

result from VecView(): parallel

- `VecSetFromOptions()` allows the type of the `Vec` to be set to different defaults at runtime
- thus we see this on 2 processes

```
$ mpiexec -n 2 ./prog
Vec Object: 2 MPI processes
  type: mpi
Process [0]
11.
7.
Process [1]
5.
3.
```

a vector in parallel: picture

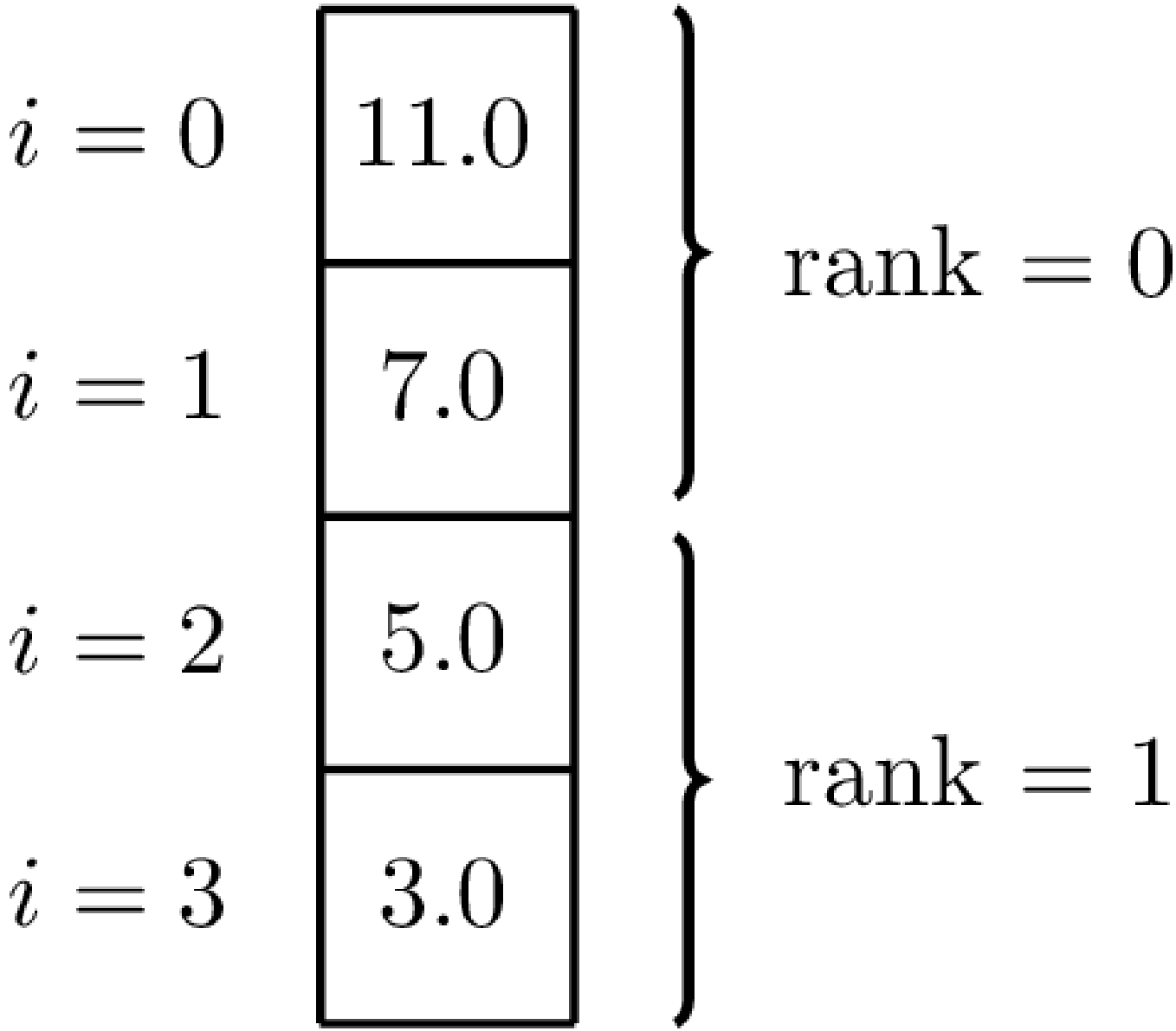

Figure 2:layout of Vec over two processes

why VecAssemblyBegin()/End()?

- recall:

```
VecAssemblyBegin(b);
VecAssemblyEnd(b);
```

- why is this needed?
- because `VecSetValues()` can be done *in parallel*
- entries of the `Vec` computed on one process will be moved "automatically" by this `VecAssembly` process to the process where they are owned

PETSc object class 2: Mat

- to create, show, and destroy a `Mat` called `A` we do this:

```
Mat     A;
MatCreate(PETSC_COMM_WORLD,&A);
MatSetSizes(A,PETSC_DECIDE,PETSC_DECIDE,4,4);
MatSetFromOptions(A);
MatSetUp(A);

[  CODE TO FILL ENTRIES  ]

MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);
MatView(A,PETSC_VIEWER_STDOUT_WORLD);
MatDestroy(&A);
```

PETSc object class 2: Mat

- to fill, here is one way:

```
int    j[4] = {0, 1, 2, 3};        # column indices
i = 0;                             # row index
v[0] = 1.0;  v[1] = 2.0;  v[2] = 3.0;
MatSetValues(A,1,&i,3,j,v,INSERT_VALUES);
i = 1;
v[0] = 2.0;  v[1] = 1.0;  v[2] = -2.0;  v[3] = -3.0;
MatSetValues(A,1,&i,4,j,v,INSERT_VALUES);
...
```

- ugly and tedious compared to Matlab's

```
A = [1.0 2.0 3.0 0.0; ...]
```

- *but* PETSc grid/mesh support will automate such `Mat` entry-filling in PDE examples

MatView() result

```
$ mpiexec -n 2 ./prog
Mat Object: 2 MPI processes
  type: mpiaij
row 0: (0, 1.)  (1, 2.)  (2, 3.)
row 1: (0, 2.)  (1, 1.)  (2, -2.)  (3, -3.)
row 2: (0, -1.)  (1, 1.)  (2, 1.)  (3, 0.)
row 3: (1, 1.)  (2, 1.)  (3, -1.)
```

- i.e. shown as sparse storage

a matrix in parallel: picture of MPIAIJ

	$j=0$	$j=1$	$j=2$	$j=3$	
$i=0$	1.0	2.0	3.0		$\text{rank}=0$
$i=1$	2.0	1.0	−2.0	−3.0	
$i=2$	−1.0	1.0	1.0	0.0	$\text{rank}=1$
$i=3$		1.0	1.0	−1.0	

Figure 3:layout of Mat over two processes

matrix in parallel

- by default, parallel PETSc matrices (MPIAIJ) have each processor own a block of rows
- thus $\mathbf{w} = A\mathbf{v}$ requires communication to put $\mathbf{v}$ on the right process, but then the result $\mathbf{w}$ is already there

full code for solving linear system

- do

```
$ cd c/ch2/
$ make vecmatksp
$ ./vecmatksp
$ mpiexec -n 2 ./vecmatksp -vec_view -mat_view
```

- look at `vecmatksp.c` code
  - note all the error-checking clutter
  - clearly: need to explain "`KSP`"
  - which requires some attention to notation, definitions, and numerical linear algebra
  - *next week*