Introduction to PETSc

First steps

Loïc Gouarin

Laboratoire de Mathématiques d'Orsay

May 13-15, 2013

Outline

What is PETSc?

Portable, Extensible Toolkit for Scientific Computation

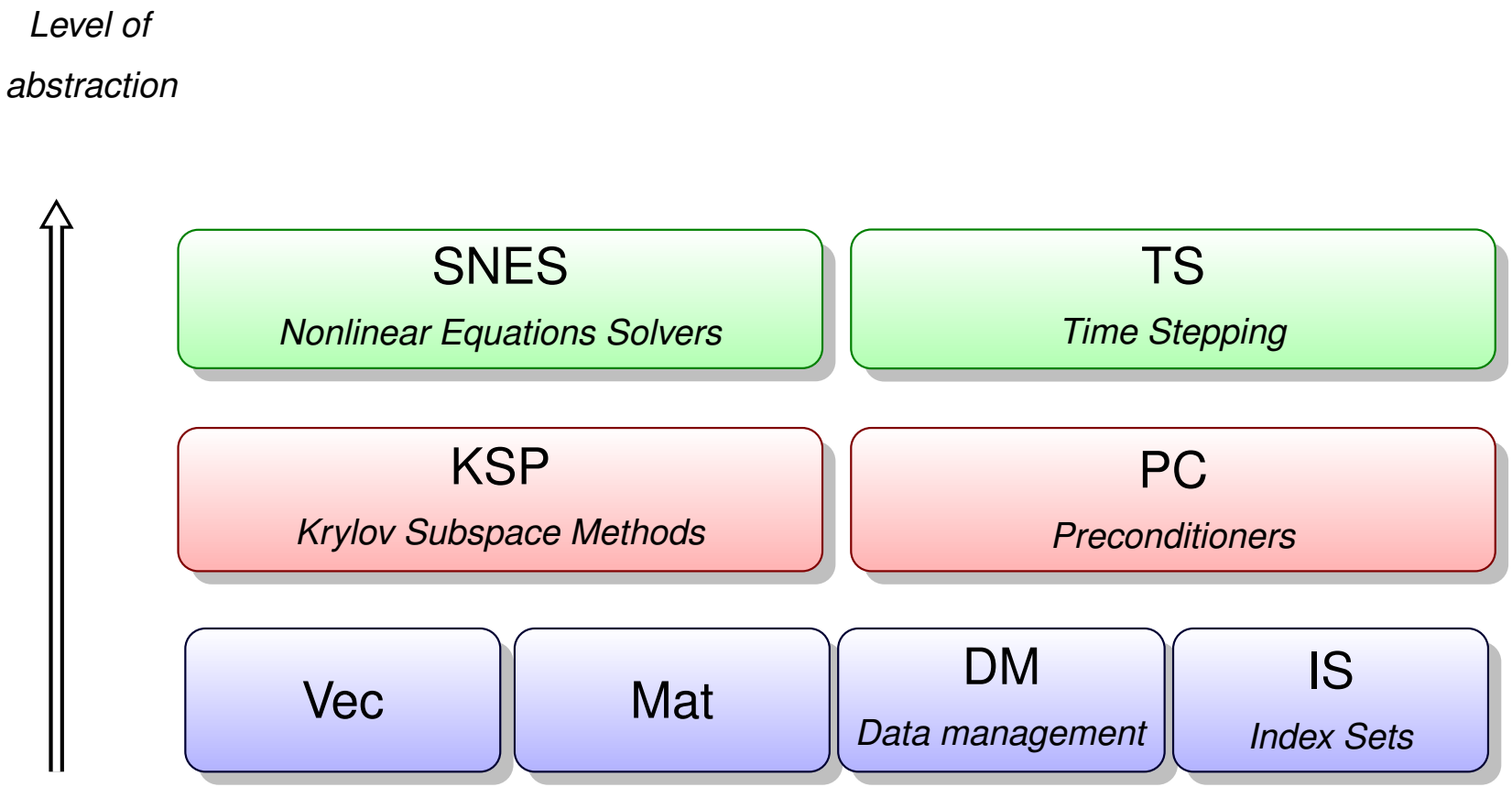
PETSc history

- begun September 1991
- Over 60,000 downloads since 1995 (version 2)
- Currently 400 per month
- 12 active developers

PETSc includes

- Linear system solvers (sparse/dense, iterative/direct)
- Nonlinear system solvers
- Tools for distributed matrices
- Support for profiling, debugging, graphical output

PETSc components



Supporting languages

- C/C++
- Fortran
- Python
- Matlab (sequential only)

Interface with external softwares

- FFTW,
- Hypre,
- MUMPS,
- ParMeTiS,
- SuperLU,
- UMFPACK,
- ...

How do I get help?

- Website : http://www.mcs.anl.gov/petsc
- FAQ
- Mailing Lists
  - For configure and installation questions :
    petsc-maint@mcs.anl.gov
  - For PETSc users :
    petsc-users@mcs.anl.gov

Configuration step

In the following, we will use the PETSc Release Version 3.3.
You can download the last version here.

- Extract the compressed file
- Set `$PETSC_DIR` to the installation root directory
- Run the configuration utility
  1. `$PETSC_DIR/configure`
  2. `$PETSC_DIR/configure -help`
  3. `$PETSC_DIR/configure --with-debugging=0`
- Many other examples can be found here
- Configuration files are in `$PETSC_DIR/PETSC_ARCH/conf`

Configuration step

- You can easily reconfigure with the same options
  `./$PETSC_ARCH/conf/reconfigure-$PETSC_ARCH.py`
- You can add other options
  `./$PETSC_ARCH/conf/reconfigure-$PETSC_ARCH.py --download-petsc4py`
- You can maintain several different configurations
  `./configure -PETSC_ARCH=linux-fast -with-debugging=0`
- All configuration information is in the logfile
  `./$PETSC_ARCH/conf/configure.log`
  ALWAYS send this file with bug reports

Building step

- Classical build with

```
cd $PETSC_DIR
make -j xx
make install
make test
```

- You can build multiple configurations
  `PETSC_ARCH=linux-fast make`
  Libraries are in `$PETSC_DIR/$PETSC_ARCH/lib`
- All build information is in the logfile
  `./$PETSC_ARCH/conf/make.log`
  ALWAYS send this file with bug reports

Exercise 1

1. Download, extract and configure PETSc
2. Build and install PETSc
3. Test your installation with `make test`
4. Install a no debug version using `--with-debugging=0`
5. Reconfigure with `--download-mumps`

C example

```c
#include "petscvec.h"

int main(int argc, char **argv){
  Vec x;

  PetscInitialize(&argc, &argv, NULL, NULL);

  VecCreateSeq(PETSC_COMM_SELF, 100, &x);
  VecSet(x, 1.);

  PetscFinalize();
  return 0;
}
```

Fortran example

```
      program main
      implicit none

#include"finclude/petscsys.h"
#include"finclude/petscvec.h"

      PetscErrorCode ierr
      Vec x

      call PetscInitialize(PETSC_NULL_CHARACTER, ierr)
      call VecCreateSeq(PETSC_COMM_SELF, 100, x, ierr)
      call VecSet(x, 1., ierr)
      call PetscFinalize(ierr)
      end program main
```

Python example

```
import petsc4py.PETSc as petsc

x = petsc.Vec()
x.createSeq(100)
x.set(1.)
```

In C/C++

```
int PetscInitialize(int *argc,char ***argv,
                    char *file,char *help)
```

- `argc` and `argv`: command line arguments.
- `file`: alternative name for the PETSc options file `.petscrc`.
- `help`: optional character string printed if option `-help` is used.

`PetscInitialize()` automatically calls `MPI_Init()`

- `PETSC_COMM_WORLD`
- `PETSC_COMM_SELF`

In Fortran

```
call PetscInitialize(character(*) file,
                     integer ierr);
```

- Do exactly the same as the C version.
- Command line arguments are read by getarg function.

In C/C++

```c
int PetscFinalize();
```

In Fortran

```
call PetscFinalize(integer ierr)
```

Use `CHKERRQ` to verify the return codes of a PETSc routine.

In C/C++

```
ierr = PetscXXX(...);CHKERRQ(ierr);
```

In Fortran

```
call PetscXXX(...., ierr)
CHKERRQ(ierr)
```

Check error on all PETSc routines and yours can save a lot of debugging time!!

```
#undef __FUNCT__
#define __FUNCT__ "myfunction"
PetscErrorCode myfunction(...){
  PetscErrorCode ierr;
  PetscFunctionBegin;

  ...

  PetscFunctionReturn(0);
}
```

How to build your project?

using makefile

```
ALL: test

CFLAGS      = -I${PETSC_DIR}/include
FFLAGS      = -I${PETSC_DIR}/include/finclude
SOURCESC    = main.c
SOURCESF    =
OBJ         = $(SOURCESC:.c=.o)
CLEANFILES = ${OBJ} test

include ${PETSC_DIR}/conf/variables
include ${PETSC_DIR}/conf/rules

test: ${OBJ} chkopts
        -${CLINKER} -o test ${OBJ} ${PETSC_SYS_LIB}
```

How to build your project?

using Cmake

You need some cmake files to be able to find PETSc on your system.

Jed Brown gives some useful cmake modules here.

For PETSc, you need

- `ResolvCompilerPaths.cmake`
- `FindPackageMultipass.cmake`
- `CorrectWindowsPaths.cmake`
- `FindPETSc.cmake`

How to build your project?

using Cmake

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)

set(CMAKE_MODULE_PATH "${CMAKE_SOURCE_DIR}/cmake/"
    ${CMAKE_MODULE_PATH})

FIND_PACKAGE(PETSc)

include_directories(${PETSC_INCLUDES})

ADD_EXECUTABLE(test main.c)
TARGET_LINK_LIBRARIES(test ${PETSC_LIBRARIES})
```

PETSc gives some options at runtime to customize your code.

The available options of your project are visible using `-help` command line.

- `-vec_view`
- `-mat_view`
- `-ksp_type`
- ...

`-option_left` indicates all user options that are not used during the execution.

Create your own options

```
PetscOptionsHasName(char *pre,char *name,
                    PetscBool *flg);

PetscOptionsGetInt(char *pre,char *name,int *value,
                   PetscBool *flg);

PetscOptionsGetReal(char *pre,char *name,double *value,
                    PetscBool *flg);

PetscOptionsGetString(char *pre,char *name,char *value,
                      int maxlen,PetscBool *flg);

...
```

Create your own options

An example

```
#include "petsc.h"

int main(int argc, char **argv){
  PetscBool flg;
  ...
  ierr = PetscOptionsHasName(PETSC_NULL, "-myoption",
                             &flg);CHKERRQ(ierr);
  if (flg)
    PetscPrintf(PETSC_COMM_SELF, "-myoption is set\n");
  ...
}
```

Exercise 2

1. Extract the tarball `myProject.tar.gz`.
2. Choose your language.
3. Create the associated `makefile` or `CMakeLists.txt`.
4. Run this example.

Exercise 3

1. Could you find the bugs ?
2. Set error checking in the source code.

Exercise 4

1. What are the available options ?
2. Create an option to change vector dimension at runtime.
3. Create 2 options to print `x1` and `x2` with options `-x1_view` and `-x2_view`.
   Use `VecView` function to view `x1` and `x2`.

References

1. PETSc documentation
   http://www.mcs.anl.gov/petsc/documentation/index.html
2. PETSc tutorial
   http://www.mcs.anl.gov/petsc/documentation/tutorials/index.html