



UNIVERSITY OF
SASKATCHEWAN

Numerical Simulation Laboratory

Time integration with PETSc

Kevin R. Green, Raymond J. Spiteri

Department of Computer Science
University of Saskatchewan

Math 314 - Department of Mathematics & Statistics
University of Saskatchewan
Fall 2017

Outline

- 1 Lecture 1: Introduction and Installation
- 2 Lecture 2: Data structures and classes overview
- 3 Lecture 3: Data structures and classes continued
- 4 Lecture 4: Solving ODE IVPs with TS
- 5 Lecture 5: Solving BVPs with TS and SNES
- 6 Lecture 6: Solving DAE IVPs with TS

Outline

- 1 Lecture 1: Introduction and Installation
- 2 Lecture 2: Data structures and classes overview
- 3 Lecture 3: Data structures and classes continued
- 4 Lecture 4: Solving ODE IVPs with TS
- 5 Lecture 5: Solving BVPs with TS and SNES
- 6 Lecture 6: Solving DAE IVPs with TS

Source code for example problems

All example codes from these lectures are available in the

Course Materials / PETSc / Examples

directory on Blackboard.

The (updated) pdf of these slides is also available on Blackboard.

PETSc

<https://www.mcs.anl.gov/petsc/>

PETSc (|' pet_si:|) stands for the **P**ortable, **E**xtensible **T**oolkit for **S**cientific computing.

It provides a suite of libraries for the numerical solution of partial differential equations (PDEs) and related problems.

Portable Extensible Toolkit for Scientific computing

- Architecture
 - tightly coupled (e.g. Cray, Blue Gene)
 - loosely coupled such as network of workstations
 - GPU clusters (many vector and sparse matrix kernels)
- Operating systems (Linux, Mac, Windows, BSD, proprietary Unix)
- Any compiler
- Real/complex, single/double/quad precision, 32/64-bit int
- Usable from C, C++, Fortran 77/90, Python, and MATLAB
- Free to everyone (2-clause BSD license), open development
- 10^{12} unknowns, full-machine scalability on Top-10 systems
- Same code runs performantly on a laptop

Portable **Extensible** Toolkit for Scientific computing

Philosophy: Everything has a plugin architecture

- Vectors, Matrices, Coloring/ordering/partitioning algorithms
- Preconditioners, Krylov accelerators
- Nonlinear solvers, Time integrators
- Spatial discretizations/topology

Portable Extensible **Toolkit** for Scientific computing

Algorithms, (parallel) debugging aids, low-overhead profiling.

Composability

- Try new algorithms by choosing from product space and composing existing algorithms (multilevel, domain decomposition, splitting).

Experimentation

- It is **not** possible to pick the best solver *a priori*.
What will deliver best/competitive performance for a given physics, discretization, architecture, and problem size?
- PETSc's response: expose an algebra of composition so new solvers can be created at runtime.
- Important to keep solvers decoupled from physics and discretization because we also experiment with those.

Portable Extensible Toolkit for **Scientific computing**

- Computational Scientists
 - PyLith, Underworld, PFLOTRAN, MOOSE, Proteus, PyClaw, CHASTE
- Algorithm Developers (iterative methods and preconditioning)
- Package Developers
 - SLEPc, TAO, Deal.II, Libmesh, FEniCS, PETSc-FEM, MagPar, OOFEM, FreeCFD, OpenFVM
- Hundreds of tutorial-style examples
- Hyperlinked manual, examples, and manual pages for all routines
- Support from petsc-maint@mcs.anl.gov, petsc-users@mcs.anl.gov

Role of PETSc

Developing parallel, nontrivial PDE solvers that deliver high performance is still difficult and requires months (or even years) of concentrated effort.

*PETSc is a toolkit that can ease these difficulties and reduce the development time, but it is **not** a black-box PDE solver, nor a silver bullet.*

— Barry Smith

PETSc installation

Installing on your own machine (instructions to follow)

Installing on a cluster:

- Don't do it (generally).
- Use pre-built install.
- Consult with computing staff if not available.

Obtaining PETSc source code

- Specific releases, zipped tarballs
<http://www.mcs.anl.gov/petsc/download/>
- Git repository [preferred]
<https://bitbucket.org/petsc/petsc/>

Why is Git preferred?

- Public development repository, you can get *any* development version.
- All releases are just tags: eg. v3.7.6
- Easily rollback changes and releases.

Unpacking PETSc

Clone from repository:

```
$ git clone https://bitbucket.org/petsc/petsc.git  
$ git checkout v3.7.6
```

OR

Unpack the archive:

```
$ tar xzf petsc.tar.gz
```

Configuring PETSc

- Set environment variable PETSC_DIR to the installation root directory
- Choose a value for PETSC_ARCH (perhaps c-debug for a standard debug build)
- Run the configuration utility
 - `$PETSC_DIR/configure [--help]`
- May require additional dependencies
 - **Read** the output message if the configure fails, should tell you why, and suggest configure options to help

External libraries

External libraries can be downloaded during the configuration phase.

For example, MUMPS and SuperLU_dist (efficient algorithms for direct solving of linear systems) can be obtained with

```
--download-mumps --download-scalapack
```

and

```
--download-superlu_dist --download-metis --download-parmetis
```

respectively.

Buidling PETSc

After successful configure, run

```
$ make  
$ make test  
$ make streams
```


Enabling PETSc

PETSc requires a couple of environment variables to be set:

- PETSC_DIR - the top level directory of PETSc
- PETSC_ARCH - the *architecture* for which PETSc has been built

Allows multiple ARCH types built on a given system, which can easily be swapped for your specific application code.

ie., having a version configured with debugging flags enabled, and a version without.

Generally on a cluster, PETSC_ARCH is left empty, and only an optimized build is maintained.

Using PETSc on plato.usask.ca

I've built a version of PETSc for use throughout these lectures. Add the following to your `~/.bashrc` file:

```
source /home/krg958/petsc/math314.env
```

This sets up the required environment for compiling PETSc application programs. The programs can then be run as any other MPI programs. eg. with the Slurm job scheduler (template given with examples):

```
$ sbatch submit_template.slurm
```

More info about submitting jobs on plato can be found [here](#)

It is important that you edit at least the following for each submission:

- executable name
- maximum runtime
- maximum RAM

Installing yourself

Exercise 1: Personal installation

Install PETSc v3.7.6 on a machine you use regularly. Set it up with the following properties

- configured with MUMPS with debugging symbols
- `PETSC_ARCH=MUMPS-debug`

Install a second version with

- configured with MUMPS and no debugging symbols
- `PETSC_ARCH=MUMPS-opt`

Testing your PETSc environment

Exercise 2: Testing memory bandwidth of your system

With `PETSC_DIR` and `PETSC_ARCH` set appropriately for your current system:

- Build the `MPIVersion.c` example code.
- Run the `MPIVersion` executable. What kind of memory bandwidth does it show? What does *memory bandwidth* mean?
- Try running with 1, 2, 4, 8, 16, 32 cores.

Recall that running jobs on a cluster should be done by submission with the appropriate job scheduler.

Outline

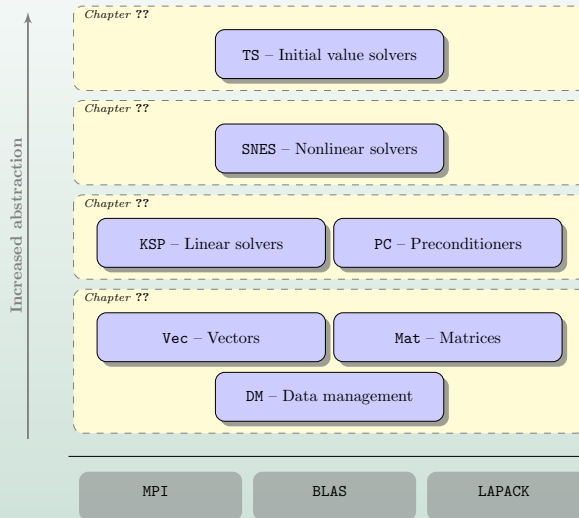
- 1 Lecture 1: Introduction and Installation
- 2 Lecture 2: Data structures and classes overview**
- 3 Lecture 3: Data structures and classes continued
- 4 Lecture 4: Solving ODE IVPs with TS
- 5 Lecture 5: Solving BVPs with TS and SNES
- 6 Lecture 6: Solving DAE IVPs with TS

PETSc data types

Examples.

- PetscInt
 - defaults to 32bit integer
 - can be 64bit int if you're dealing with many unknowns (`--with-64-bit-indices`)
- PetscScalar
 - Primary type for your unknowns, depending on configuration can be:
 - Double precision real (default) or complex
 - Single precision
 - Quad precision
- PetscReal
 - Same precision as PetscScalar, but always real

PETSc hierarchy



PetscObject

Every object in PETSc supports a basic interface

Function	Operation
Create()	create the object
Get/SetName()	name the object
Get/SetType()	set the implementation type
Get/SetOptionsPrefix()	set the prefix for all options
SetFromOptions()	customize object from command line
SetUp()	perform other initialization
View()	view the object
Destroy()	cleanup object allocation

Also, all objects support the `-help` option.

Options

Ways to set options:

- Command line
- Filename in the third argument of `PetscInitialize()`
- `~/.petscrc`
- `$PWD/.petscrc`
- `$PWD/petscrc`
- `PetscOptionsInsertFile()`
- `PetscOptionsInsertString()`
- `PETSC_OPTIONS` environment variable
- command line option `-options_file [file]`

Vectors (Vec)

- Extension of arrays.
 - Parallel (global) representations
 - Serial (local) representations
- Think of them just like vectors in linear algebra.
- Used most often to store:
 - solution to your problem
 - residuals - nonlinear and linear
- Can be used to hold location dependent field data for the problem as well.
- Can extract data as arrays for manipulation.

Distributed Vec

Parallel vector stored on 3 processors, as an example

$$\mathbf{u}_{\text{global}} = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} \quad \left\{ \begin{array}{l} \mathbf{u}_{\text{local}}^{(0)} = \begin{bmatrix} u_0 & u_1 & u_2 & \mathbf{u}_{\text{ghost}}^{(0)T} \end{bmatrix}^T \\ \mathbf{u}_{\text{local}}^{(1)} = \begin{bmatrix} u_3 & u_4 & u_5 & \mathbf{u}_{\text{ghost}}^{(1)T} \end{bmatrix}^T \\ \mathbf{u}_{\text{local}}^{(2)} = \begin{bmatrix} u_6 & u_7 & \mathbf{u}_{\text{ghost}}^{(2)T} \end{bmatrix}^T \end{array} \right.$$

Vec operations

Man pages:

http:

[//www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/Vec/index.html](http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/Vec/index.html)

- Things you expect with vectors:
 - axpy, norms, dot products, etc.
- Explicit access to entries:
 - `VecGetArray()`, `VecRestoreArray()`
 - and their *read only* variants
- Parallel communication:
 - assembly, scatter, gather

Matrices (Mat)

What are PETSc matrices?

- Linear operators on finite dimensional vector spaces.

Matrices (Mat)

What are PETSc matrices?

- Linear operators on finite dimensional vector spaces.
- Fundamental objects for storing stiffness matrices and Jacobians
- Each process locally owns a contiguous set of rows
- Supports many data types
 - AIJ, Block AIJ, Symmetric AIJ, Block Diagonal, etc.
- Supports structures for many packages
 - MUMPS, SuperLU, UMFPack, Hypre, Elemental

Matrices (Mat)

What are PETSc matrices?

- Linear operators on finite dimensional vector spaces.
- Fundamental objects for storing stiffness matrices and Jacobians
- Each process locally owns a contiguous set of rows
- Supports many data types
 - AIJ, Block AIJ, Symmetric AIJ, Block Diagonal, etc.
- Supports structures for many packages
 - MUMPS, SuperLU, UMFPack, Hypre, Elemental

Distributed Mat

Nonzero entries of a matrix stored on 3 processors.

1	2	0	3	0	0	4	0	Proc 0
5	6	7	0	0	8	9	0	
0	10	11	12	0	0	13	14	
0	0	15	16	17	0	0	18	Proc 1
19	0	0	20	21	22	23	0	
0	24	0	0	25	26	27	28	
0	29	0	0	0	30	31	32	Proc 2
33	0	0	34	35	0	36	37	

Matrix Polymorphism

The PETSc Mat has a single user interface,

- Matrix assembly: `MatSetValues()`
- Matrix-vector multiplication: `MatMult()`
- Matrix viewing: `MatView()`

but multiple underlying implementations.

- AIJ, Block AIJ, Symmetric Block AIJ,
- Dense, Elemental
- Matrix-Free
- etc.

A matrix is defined by its **interface**, not by its **data structure**.

How to create matrices

- `MatCreate(MPI_Comm, Mat *)`
- `MatSetSizes(Mat, int m, int n, int M, int N)`
- `MatSetType(Mat, MatType typeName)`
- `MatSetFromOptions(Mat)`
 - Can set the type at runtime
- `MatSetBlockSize(Mat, int bs)`
- `MatXAIJSetPreallocation(Mat,...)`
- `MatSetValues(Mat,...)`
 - `MatSetValuesLocal, MatSetValuesStencil`
 - `MatSetValuesBlocked`
- `MatAssemblyBegin/End`

What can we do with a matrix that doesn't have entries?

Iterative solvers for $Au = b$

- Finding solutions in the Krylov subspace:
 $\{b, Ab, A^2b, A^3b, \dots\}$
- Convergence rate depends on the spectral properties of the matrix
 - Existence of small polynomials $p_n(A) < \epsilon$ where $p_n(0) = 1$.
 - condition number $\kappa(A) = \|A\| \|A^{-1}\| = \sigma_{\max}/\sigma_{\min}$
 - distribution of singular values, spectrum Λ , pseudospectrum Λ_ϵ

Krylov subspace problem (KSP)

- Solving linear systems:

$$Au = b$$

- Various *Krylov* methods:

- GMRES (gmres)
- Conjugate gradient (cg)
- Biconjugate gradient - stabilized (bcgs)
- etc

- Direct methods (external packages recommended):

- LU factorization
- Cholesky factorization

How to set up a linear solver

- `KSPCreate(MPI_Comm, KSP *)`
- `KSPSetOperators(KSP, Mat A, Mat A_p)`
- `KSPSetFromOptions(KSP)`
 - Can set all aspects of solver at runtime
- `KSPGetPC(KSP, PC *)`
 - Explicitly get the preconditioner to manipulate
 - can be set from command line if `KSPSetFromOptions` was used
- `KSPSolve(KSP, Vec b, Vec u)`

Preconditioning?

Definition (Preconditioner)

A *preconditioner* \mathbf{P} is a method for constructing a matrix (just a linear function, not assembled!) $P^{-1} = \mathbf{P}(A, A_p)$ using a matrix A and extra information A_p , such that the spectrum of $P^{-1}A$ (or AP^{-1}) is well-behaved.

PC idea

Idea: improve the conditioning of the Krylov operator

- Left preconditioning

$$(P^{-1}A)u = P^{-1}b$$
$$\{P^{-1}b, (P^{-1}A)P^{-1}b, (P^{-1}A)^2P^{-1}b, \dots\}$$

PC idea

Idea: improve the conditioning of the Krylov operator

- Left preconditioning

$$(P^{-1}A)u = P^{-1}b$$

$$\{P^{-1}b, (P^{-1}A)P^{-1}b, (P^{-1}A)^2P^{-1}b, \dots\}$$

- Right preconditioning

$$(AP^{-1})Pu = b$$

$$\{b, (AP^{-1})b, (AP^{-1})^2b, \dots\}$$

PC idea

Idea: improve the conditioning of the Krylov operator

- Left preconditioning

$$(P^{-1}A)u = P^{-1}b$$

$$\{P^{-1}b, (P^{-1}A)P^{-1}b, (P^{-1}A)^2P^{-1}b, \dots\}$$

- Right preconditioning

$$(AP^{-1})Pu = b$$

$$\{b, (AP^{-1})b, (AP^{-1})^2b, \dots\}$$

- The product $P^{-1}A$ or AP^{-1} is *not* formed.

LU factorization

- Handled as a preconditioner.

LU factorization

- Handled as a preconditioner.
- Logic: LU factorization with back/forward substitution is a *perfect* left preconditioner.

LU factorization

- Handled as a preconditioner.
- Logic: LU factorization with back/forward substitution is a *perfect* left preconditioner.
- PETSc only provides a serial LU preconditioner. . .

LU factorization

- Handled as a preconditioner.
- Logic: LU factorization with back/forward substitution is a *perfect* left preconditioner.
- PETSc only provides a serial LU preconditioner. . .
- But we can use external packages for parallel solves: MUMPS, SuperLU_dist

Vec exercise

Exercise 3: Basic vector routines

- Investigate the source code `examples/Lecture_02/ex1.c`, and the structure of its makefile.
- Set the vector size to 1000 using a command-line option.
- Investigate other options available to this program with the `-help` option.
- Can you get the program to output the line:
Total flops over all processors 32991. ?

KSP direct solver exercise

Exercise 4: Simple linear solve

- Looking at the code in `examples/Lecture_02/ex2.c`, determine:
 - Where the matrix values are set.
 - Where the solver type is set.
 - How the error is computed.
- What are the default KSP type and PC type?
 - *Don't forget about `-help`*

Exercise 5: Solve with LU factorization

- Use the options:
`-pc_type lu -pc_factor_mat_solver_package mumps`
to solve using LU factorization.
- Look at `-ksp_view` to see what info this gives you.

Message passing interface (MPI)

Typically, user does not call MPI functions directly.

At most, `MPI_Comm_size()` and `MPI_Comm_rank()` can be used. All other MPI communication functions are wrapped within PETSc functions.

However, some degree of understanding how MPI works is needed to create PETSc code that can scale well.

MPI communicators in PETSc

- PETSc objects need an MPI_Comm in their constructor
 - PETSC_COMM_SELF for serial objects
 - PETSC_COMM_WORLD common, but *not* required
- Can split communicators, spawn processes on new communicators, etc
- Operations are one of
 - Not Collective: *ie.*, VecGetLocalSize(), MatSetValues()
 - Logically Collective: *ie.*, KSPSetType()
 - checked when running in debug mode
 - Neighbor-wise Collective: VecScatterBegin(), MatMult()
 - Point-to-point communication between two processes
 - Collective: VecNorm(), MatAssemblyBegin(), KSPCreate()
 - Global communication, synchronous
- Deadlock if some process doesn't participate (e.g. wrong order)

Outline

- 1 Lecture 1: Introduction and Installation
- 2 Lecture 2: Data structures and classes overview
- 3 Lecture 3: Data structures and classes continued**
- 4 Lecture 4: Solving ODE IVPs with TS
- 5 Lecture 5: Solving BVPs with TS and SNES
- 6 Lecture 6: Solving DAE IVPs with TS

Structured nonlinear equation solver (SNES)

- Standard form of a nonlinear system

$$F(u) = 0$$

- Newton iteration

$$\text{Solve: } J(u)w = -F(u)$$

$$\text{Update: } u^+ \leftarrow u + w$$

- Quadratically convergent near a root: $|u^{n+1} - u^*| \in \mathcal{O}(|u^n - u^*|^2)$
- *Quasi-Newton* methods can use a different $J(u)$

How to set up a SNES I

- `SNESCreate(SNES *)`
- The SNES interface is based upon callback functions
 - `FormFunction(...)`, set by `SNESSetFunction()`
 - `FormJacobian(...)`, set by `SNESSetJacobian()`
- When PETSc needs to evaluate F and J ,
 - Solver calls the **user's** function
 - User function gets application state through the `ctx` variable
 - cast to/from a void pointer (see upcoming exercise)
 - PETSc *never* sees application data
- `SNESSetFromOptions(SNES)`
`SNESSolve(SNES, Vec b, Vec u)`

How to set up a SNES II

The user provided function which calculates the **nonlinear residual** has signature

```
PetscErrorCode (*func)(SNES snes, Vec u, Vec r, void *ctx)
```

u: The current solution

r: The residual

ctx: The user context passed to `SNESSetFunction()`

- Use this to pass application information, e.g. physical constants

How to set up a SNES III

Similar story for the function to evaluate Jacobian:

```
PetscErrorCode (*func)(SNES snes,Vec u,Mat J,  
                      Mat Jpre,void *ctx)
```

u: The current solution

J: The Jacobian

Jpre: The Jacobian preconditioning matrix (possibly J itself)

ctx: The user context passed to `SNESSetFunction()`

- Use this to pass application information, e.g. physical constants

How to set up a SNES III

Similar story for the function to evaluate Jacobian:

```
PetscErrorCode (*func)(SNES snes,Vec u,Mat J,
                      Mat Jpre,void *ctx)
```

u: The current solution

J: The Jacobian

Jpre: The Jacobian preconditioning matrix (possibly J itself)

ctx: The user context passed to `SNESSetFunction()`

- Use this to pass application information, e.g. physical constants

Alternatively, you can use

- a built-in sparse finite difference approximation (“coloring”)

SNES global strategies

Newton's method can diverge if not close to a solution.

SNES global strategies

Newton's method can diverge if not close to a solution.

Line search (`newtonls`):

- 1 Update **direction** determined by linear solve in Newton method
- 2 Update **size** chosen by maximizing decrease in nonlinear residual in the update direction

SNES global strategies

Newton's method can diverge if not close to a solution.

Line search (`newtonls`):

- ① Update **direction** determined by linear solve in Newton method
- ② Update **size** chosen by maximizing decrease in nonlinear residual in the update direction

Trust region (`newtontr`):

- ① Trust region **size** chosen based on how well a local approximation matches F
- ② Update **direction** chosen by maximizing the decrease in nonlinear residual within the trust region

Timestepping (TS)

Typically applied to *method of lines* discretizations of PDEs.

PETSc permits 2 forms for initial value problems

- 1 Standard ODE IVP form:

$$\dot{u} = F(t, u)$$

- 2 DAE and IMEX methods:

$$G(t, u, \dot{u}) = F(t, u)$$

$$J_{\alpha} = \alpha G_{\dot{u}} + G_u$$

Single Step interface so user can control their own time loop, or one-shot Solve to produce a solution at some desired final time.

Some TS methods

TSTHETA Theta methods

TSALPHA Alpha methods

TSRK Runge-Kutta methods

TSARKIMEX 2-Additive Runge-Kutta methods

TSBDF Backward differentiation methods

TSROSW Rosenbrock-W methods

TSSSP Strong stability methods (Total variation diminishing)

TSGL General linear methods - multi-stage/multi-step

TSSUNDIALS SUNDIALS (external) ode integrators

How to set up a TS I

- `TSCreate(TS *)`
- User provides functions for:
 - Explicit rhs - `TSSetRHSFunction()`
 - Implicit lhs - `TSSetIFunction()`, `TSSetIJacobian()`
- `TSSetFromOptions(TS)`
- `TSSolve(TS, Vec u)`

How to set up a TS II

Function signatures:

- `FormRHSFunction(ts,t,u,F,void *ctx);`
- `FormIFunction(ts,t,u,udot,G,void *ctx);`
- `FormIJacobian(ts,t,u,udot,alpha,J,J_p,void *ctx);`

How to set up a TS II

Function signatures:

- `FormRHSFunction(ts,t,u,F,void *ctx);`
- `FormIFunction(ts,t,u,udot,G,void *ctx);`
- `FormIJacobian(ts,t,u,udot,alpha,J,J_p,void *ctx);`

Can also supply:

- `FormRHSJacobian(ts,t,u,F,J,J_p,void *ctx);`

to use fully implicit methods.

SNES exercise

Exercise 6: Simple nonlinear solve

- Investigate the code in `examples/Lecture_03/ex1.c` (note this is a serial example!), determine:
 - How the residual is evaluated.
 - How the Jacobian is evaluated.
- What are the default methods employed?
- Can you get the program to output the reason for convergence?
- Can you monitor the linear system solves as well?

TS exercise

Exercise 7: DAE benchmark problems for TS

- Investigate the source code in `examples/Lecture_03/ex8.c`, determine:
 - What problems are being solved?
 - How do you select a problem from command line?
- What types of time integration methods can you use, what is the default? (`-ts_view`)
- Run the program with LU factorization for the linear system solves.

Outline

- 1 Lecture 1: Introduction and Installation
- 2 Lecture 2: Data structures and classes overview
- 3 Lecture 3: Data structures and classes continued
- 4 Lecture 4: Solving ODE IVPs with TS**
- 5 Lecture 5: Solving BVPs with TS and SNES
- 6 Lecture 6: Solving DAE IVPs with TS

ODE TS example problem

Forced, damped oscillator:

$$m\ddot{x} + c\dot{x} + kx = f(t), \quad x(0) = x_0, \quad \dot{x}(0) = 0$$

written as a first order system

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{f}(t),$$

with

$$\mathbf{x} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 0 & 1 \\ -k/m & -c/m \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} 0 \\ f(t)/m \end{bmatrix}$$

Feeling out the code

Build the example:

```
cd examples/Lecture_04 && make fdo1
```

Start by discovering defaults and available options:

```
./fdo1 -help -ts_view > options_and_defaults
```

User provided option to output solution during time integration:

```
-monitor_solution
```

(Can be easily visualized with your tool of choice, ie., gnuplot)

Problem struct

Problem specific data stored in one of these.

```
typedef struct _Parameters Parameters;  
struct _Parameters {  
    PetscReal    m,c,k;    // eqn parameters  
    PetscScalar F,omega; // forcing parameters  
    PetscScalar x0,xp0;    // initial conditions  
};
```

Created in main program as

```
Parameters* params;  
ierr = PetscMalloc(sizeof(Parameters), &params); CHKERRQ(ierr);
```

Problem cli options

In main program,

```
ierr = PetscOptionsBegin(PETSC_COMM_WORLD, NULL, "Title", "Man"); CHKERRQ(ierr);  
{  
    /* ... */  
}  
ierr = PetscOptionsEnd(); CHKERRQ(ierr);
```

nicely annotates the `-help` with your problem-specific options.

Set defaults and get cli options in this region, ie.,

```
params->m = 1;  
ierr      = PetscOptionsReal("-m", "Mass parameter", "", params->m, &(params->m), NULL); CHKERRQ(ierr);
```

Problem function evaluation

```
#undef __FUNCT__
#define __FUNCT__ "RHSFunction"
static PetscErrorCode RHSFunction(TS ts,PetscReal t,Vec X,Vec F,void *ctx)
{
    PetscErrorCode    ierr;
    Parameters*       params = (Parameters*)(ctx);
    PetscScalar        *f;
    const PetscScalar *x;

    PetscFunctionBeginUser;
    ierr = VecGetArrayRead(X,&x);CHKERRQ(ierr);
    ierr = VecGetArray(F,&f);CHKERRQ(ierr);
    f[0] = x[1];
    f[1] = (-params->k*x[0] - params->c*x[1] + params->c*forcing_function(t,params)) / params->h;
    ierr = VecRestoreArrayRead(X,&x);CHKERRQ(ierr);
    ierr = VecRestoreArray(F,&f);CHKERRQ(ierr);
    PetscFunctionReturn(0);
}
```

Problem Jacobian evaluation

Similar to function evaluation.

```
#undef __FUNCT__
#define __FUNCT__ "RHSJacobian"
static PetscErrorCode RHSJacobian(TS ts,PetscReal t,Vec X,Mat A,Mat B,void *ctx)
{
    PetscErrorCode    ierr;
    Parameters*       params = (Parameters*)(ctx);
    PetscInt           rowcol[] = {0,1};
    PetscScalar        J[2][2];
    const PetscScalar *x;

    PetscFunctionBeginUser;
    /* ... */
    PetscFunctionReturn(0);
}
```


TS setup

Getting everything into a TS object.

```
ierr = TSCreate(PETSC_COMM_WORLD,&ts);CHKERRQ(ierr);  
ierr = TSSetProblemType(ts,TS_LINEAR);CHKERRQ(ierr);  
ierr = TSSetRHSFunction(ts,NULL,RHSFunction,params);CHKERRQ(ierr);  
ierr = TSSetRHSJacobian(ts,A,A,RHSJacobian,params);CHKERRQ(ierr);  
ierr = TSSetDuration(ts,maxsteps,final_time);CHKERRQ(ierr);  
ierr = TSSetExactFinalTime(ts,TS_EXACTFINALTIME_STEPOVER);CHKERRQ(ierr);  
ierr = TSSetMaxStepRejections(ts,10);CHKERRQ(ierr);
```

ODE examples

For running examples, use $c = 10001$, $k = 10000$.

Exercise 8: Code modification

Modify the `fdo1.c` example code to the following:

- The default method is BDF3.
- BDF adaptivity is on by default.
- The custom monitor function outputs the stepsize at each time step.

ODE examples

Exercise 9: Adaptive step sizes

Solving the fdo problem with `-ts_type bdf -ts_order 3`:

- Adjust the relative and absolute tolerances for the adaptation, note how this affects the time step sizes.
- Set the relative and absolute tolerances to be 10 times tighter for the derivative \dot{x} than for the position x . (This has to be done within the code!)

Outline

- 1 Lecture 1: Introduction and Installation
- 2 Lecture 2: Data structures and classes overview
- 3 Lecture 3: Data structures and classes continued
- 4 Lecture 4: Solving ODE IVPs with TS
- 5 Lecture 5: Solving BVPs with TS and SNES**
- 6 Lecture 6: Solving DAE IVPs with TS

Outline

- 1 Lecture 1: Introduction and Installation
- 2 Lecture 2: Data structures and classes overview
- 3 Lecture 3: Data structures and classes continued
- 4 Lecture 4: Solving ODE IVPs with TS
- 5 Lecture 5: Solving BVPs with TS and SNES
- 6 Lecture 6: Solving DAE IVPs with TS**