Enabling Optimization Using Adjoint Solvers

Presented to
ATPESC 2017 Participants

Hong Zhang
Mathematics and Computer Science Division
Argonne National Laboratory

Q Center, St. Charles, IL (USA)
August 7, 2017

U.S. DEPARTMENT OF ENERGY | Office of Science

ATPESC Numerical Software Track

EXASCALE COMPUTING PROJECT

Argonne NATIONAL LABORATORY | BERKELEY LAB | Sandia National Laboratories | Rensselaer | SMU

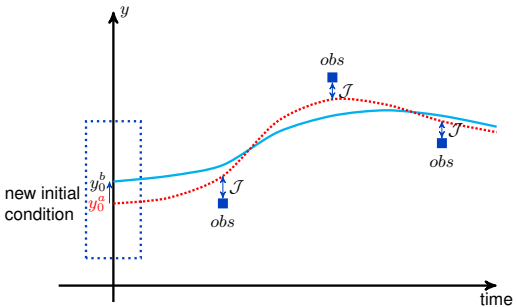
Outline

- Motivation and background
- Existing approaches
  - finite differences
  - automatic differentiation
  - forward and adjoint methods
- Adjoint solvers in PETSc
  - implementation & design
  - checkpointing
  - validation
- Solving dynamic constrained optimization with PETSc/TAO
  - basic usage
  - examples
- PETSc tips and advice
- Takeaways

Motivation: PDE-constrained optimization

$$\min_{y,u} \mathcal{J}(y,u)$$

$$\begin{aligned} s.t. \quad & c(y,u,t) = 0 && \text{(governing PDE)} \\ & g(y,u) = 0 && \text{(equality constraints)} \\ & h(y,u) \leq 0 && \text{(inequality constraints)} \end{aligned}$$



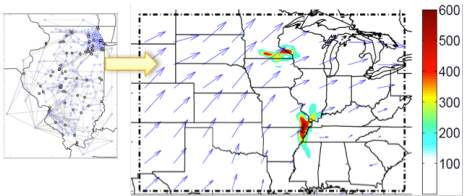
- An example objective function

$$\mathcal{J}(y,u) = \frac{1}{2}\|Qy - d\|^2 + \frac{\alpha}{2}\|L(u - u_{\text{ref}})\|^2$$

  - state variable y, control or design variable u, data d
  - Q is observation operator
  - L is cost functional for design
  - α is tradeoff between cost of design and fitting data
- Gradient-based optimization algorithms require the derivatives (Hessian optionally) for the objective and the constraints

Why do we need sensitivity analysis?

Sensitivity studies can quantify how much model output are affected by changes in model input



Sensitivity of grid operation costs with respect to weather conditions [*Cioaca et.al. 2011*]

Can be used to

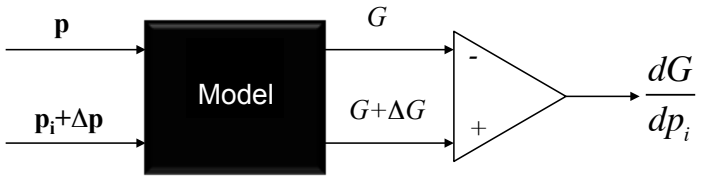
- Identify most influential parameters
- Study dynamical systems (trajectory sensitivities)
- Provide gradients of objective functions

- experimental design
- model reduction

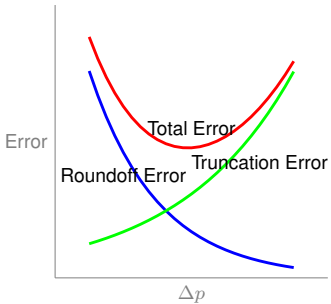
- optimal control
- parameter estimation

- data assimilation
- dynamic constrained optimization

Computing sensitivities: finite differences

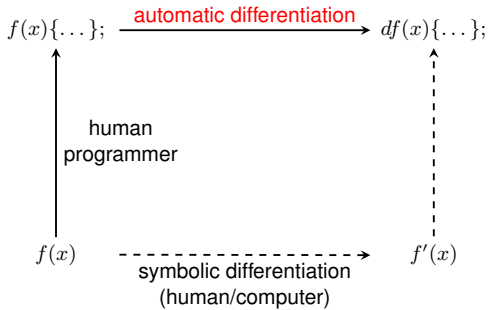


- Easy to implement
- Inefficient for many parameter case, due to one-at-a-time
- Possible to pertube multiple parameters simultaneously by using graph coloring
- Error depends on the perturbation value Δp



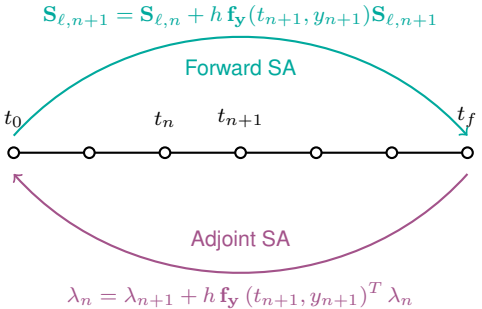
Computing sensitivities: automatic differentiation

- AD can evaluate the sensitivities for an arbitrary sequence of computer codes
- Difficulties of low-level AD
  - pointers
  - dynamic memory
  - directives
  - function calls from external libraries
  - iterative processes (e.g. Newton iteration)
  - non-smooth problems



Forward and adjoint sensitivity analysis (SA) approaches

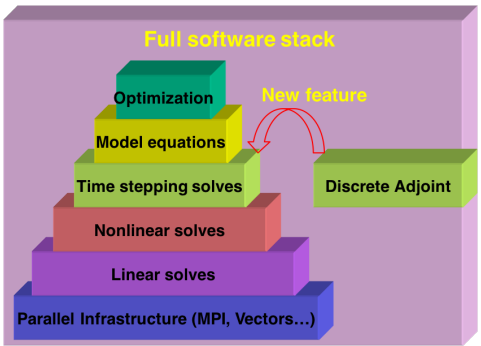
We compute the gradients by differentiating the time stepping algorithm, e.g. backward Euler ($y_{n+1} = y_n + h\,\mathbf{f}(t_{n+1}, y_{n+1})$)



	Forward	Adjoint
Best to use when	# of parameters << # functionals	# of parameters >> # of functionals
Complexity	$\mathcal{O}$ (# of parameters)	$\mathcal{O}$ (# of functionals)
Checkpointing	No	Yes
Implementation	Medium	High
Accuracy	High	High

Adjoint integration with PETSc

- PETSc: open-source numerical library for large-scale parallel computation
  https://www.mcs.anl.gov/petsc/
- $\sim$ 200,000 yearly downloads
- Portability
  - 32/64 bit, real/complex
  - single/double/quad precision
  - tightly/loosely coupled architectures
  - Unix, Linux, MacOS, Windows
  - C, C++, Fortran, Python, MATLAB
  - GPGPUs and support for threads
- Extensibility
  - ParMetis, SuperLU, SuperLU_Dist, MUMPS, HYPRE, UMFPACK, Sundials, Elemental, Scalapack, UMFPack...
- Toolkit
  - sequential and parallel vectors
  - sequential and parallel matrices (AIJ, BAIJ...)
  - **iterative solvers and preconditioners**
  - **parallel nonlinear solvers**
  - **adaptive time stepping (ODE and DAE) solvers**



Other software for adjoints and related functionality

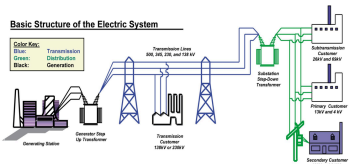
Also available in:

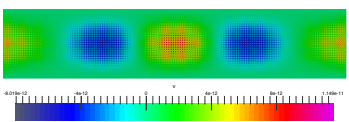
- SUNDIALS
- Trilinos

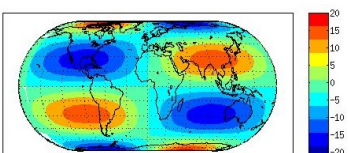
This presentation focuses on experiences in PETSc.

Applications of PETSc adjoint solvers

- The PETSc adjoint solvers have been used:
  - In a bundle method for transient security constrained economic dispatch (TSCED) problem [Francois et.al. 2017]
  - To estimate generator inertias during dynamic transient [Petra et.al. 2017]
  - With libmesh (finite element methods) for parallel solution of Navier-Stokes problems
  - With spectral element methods for CFD optimal control (ongoing)
  - In variational methods for data assimilation (ongoing)
- The programming languages for these applications range from Julia, Python, C++ and C

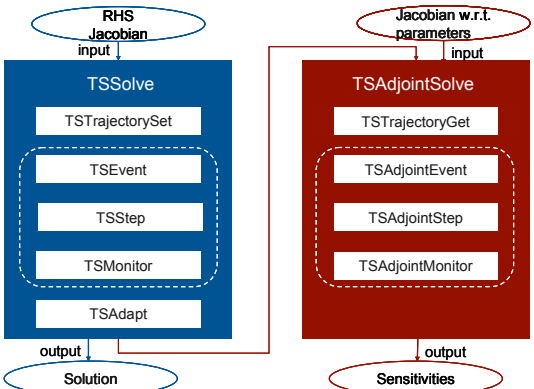






Design goals and implementation

- Minimize intrusion
- Reuse functionalities (implemented in PETSc or provided by users)
- Aim for general-purpose solutions



Optimal checkpointing

- Minimize the number of recomputations and the number of reads/writes by using the **revolve** library of Griewank and Walther
  - **Revolve** is designed as a top-level controller for time stepping
  - TSTrajectory consults **revolve** about when to store/restore/recompute
- Incorporate a variety of single-level and two-level schemes for offline and online checkpointing
  - existing algorithms work great for RAM only checkpointing
  - optimal extension for RAM+disk (work in progress)

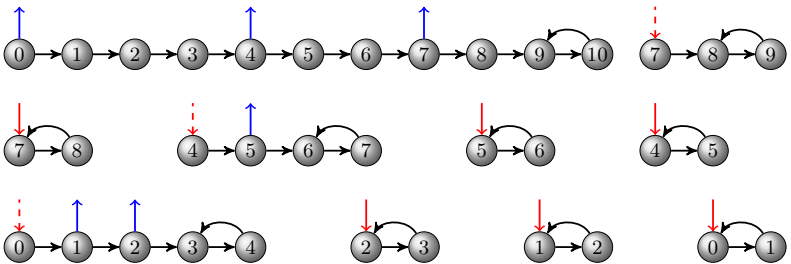
An optimal schedule given 3 allowable checkpoints in RAM:

blue arrow: store a checkpoint

red arrow: restore a checkpoint

black arrow: a step

circle: solution



Validating Jacobian and sensitivity

- PETSc and TAO (optimization component in PETSc) can test hand-coded Jacobian and gradients against finite difference approximations
- Jacobian test: `-snes_type test`

```
Norm of matrix ratio 2.83894e-08, difference 1.08067e-05 (user-defined state)
Norm of matrix ratio 3.36163e-08, difference 1.31068e-05 (constant state -1.0)
Norm of matrix ratio 3.33553e-08, difference 1.3005e-05 (constant state 1.0)
```

- Gradient test: `-tao_type test -tao_test_gradient`

```
||fd|| 0.168434, ||hc|| = 1.18456, angle cosine = (fd'hc)/||fd||||hc|| = 0.987391
2-norm ||fd-hc||/max(||hc||,||fd||) = 0.859896, difference ||fd-hc|| = 1.01859
max-norm ||fd-hc||/max(||hc||,||fd||) = 0.853218, difference ||fd-hc|| = 0.311475
```

- `-snes_test_display` and `-tao_test_display` can show the differences element-wisely

Solving dynamic constrained optimization I

User needs to provide routines that

- Initialize the variable vector (optional)
- Set the variable bounds (for bounded optimization)
- Compute the objective function value
- Compute the gradient
- Compute the Hessian matrix (optional) for Newton methods

TAO application

```
Tao tao; /* TAO Optimization solver */
UserContext user; /* user-defined structure */
Vec x; /* solution vector */

PetscInitialize(&argc,&argv,0,0);
TaoCreate(PETSC_COMM_WORLD,&tao);
TaoSetType(tao,TAOBLMVM);
TaoSetInitialVector(tao,x);
TaoSetObjectiveRoutine(tao,MyFunction,&user);
TaoSetGradientRoutine(tao,MyGradient,&user);
// The above two routines can be replaced with
// TaoSetObjectiveAndGradientRoutine()
TaoSetFromOptions(tao);
TaoSolve(tao);
```

Solving dynamic constrained optimization II

To compute the gradients using TSAdjoint, one needs to provide routines that are normally needed by TS such as the right-hand-side function and the Jacobian and an additional Jacobian w.r.t parameters if gradients to the parameters are desired.

The terminal conditions (initial values for adjoint variables) for the adjoint sensitivity variables must be set properly.

Using PETSc Adjoint solver for gradients

```
PetscErrorCode MyGradient(TaoSolver tao,Vec x,
                          Vec g,void *user){
TS ts;
Vec u;

TSCreate(PETSC_COMM_WORLD,&ts);
TSSetType(ts,TSCN);
TSSetIFunction(ts,NULL,MyIFunction,user);
TSSetIJacobian(ts,A,A,MyIJacobian,user);
TSSetDuration(ts,PETSC_DEFAULT,ftime);
TSSetExactFinalTime(ts,TS_EXACTFINALTIME_MATCHSTEP);
TSSetInitialTimeStep(ts,0.0,0.1);
TSSetSaveTrajectory(ts);
...Initialize u or system parameters with x...
TSSolve(ts,u);
...Set terminal conditions for lambda and mu...
TSSetCostGradients(ts,1,lambda,mu);CHKERRQ(ierr);
TSAdjointSolve();
...Compute g from lambda and mu...
}
```

Examples for SA and optimization

PETSc has some examples included in the source folder `src/ts/exmaples` that you can follow to build your own applications.

Equation	Source	Application
Van der pol	ex20adj.c	adjoint SA
	ex20fwd.c	forward SA
	ex20opt_ic.c	optimization over initial conditions
	ex20opt_p.c	optimization over parameters
Hybrid system with switching	hybrid/ex1adj.c	adjoint SA
	hybrid/ex1fwd.c	forward SA
Power grid stability	power_grid/stability_9bus/ex9busadj.c	adjoint SA
	power_grid/stability_9bus/ex9busopt.c	optimization over parameters
Generator swing equation with discontinuities	power_grid/ex3adj.c	adjoint SA
	power_grid/ex3fwd.c	forward SA
	power_grid/ex3opt.c	optimization using adjoint
	power_grid/ex3opt_fd.c	optimization using FD approximation
	power_grid/ex3opt_fwd.c	optimization using forward SA
Diffusion-reaction PDE	advection-diffusion-reaction/ex5adj.c	adjoint SA

* Highlighted examples will be demonstrated in the hands-on session

PETSc tips and advice

- Jacobian can be efficiently approximated using finite difference with coloring (`-snes_fd_coloring`); particularly convenient via `DMDA`
- Most of the difficulties stem from mistakes in the hand-coded Jacobian function; make sure to validate it carefully
- Use direct solvers such as SuperLU and MUMPS for best accuracy (but not scalability) of the gradients
- Use `-tao_monitor` `-ts_monitor` `-ts_adjoint_monitor` `-snes_monitor` `-log_view` for monitoring the solver behavior and profiling the performance
- `-malloc_hbw` allows us to do the computation using MCDRAM and checkpointing using DRAM on Intel's Knights Landing processors (Argonne's Theta, NERSC's Cori)
- Check the user manual and the website for more information, and ask questions on the mailing lists

Takeaways

- Adjoint as an enabling technology for optimization
- PETSc offers discrete adjoint solvers that take advantage of highly developed PETSc infrastructure: MPI, parallel vectors, domain decomposition, linear/nonlinear solvers
- Requires minimal user input, and reuses information provided for the forward simulation
- PETSc and TAO help you rapidly develop parallel code for dynamic constrained optimization
- Advanced checkpointing, transparent to the user
- Validation for Jacobian and gradients using finite differences

Thank you!