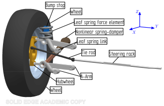
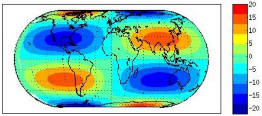
Optimization and sensitivity analysis of time-dependent simulations

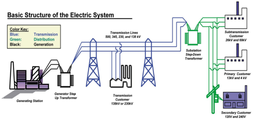
Hong Zhang

Mathematics and Computer Science Division
Argonne National Laboratory

June 16, 2015







What is sensitivity analysis and why is it important?

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

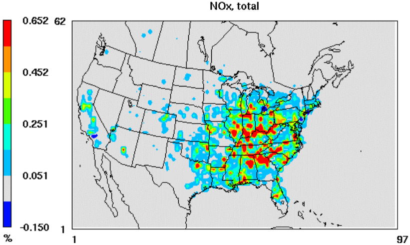


Figure: Air quality sensitivities to emissions of selective chemical species

Can be used to

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

$$G = g(y(t_F)) + \int_{t_0}^{t_F} r(t, y)dt$$

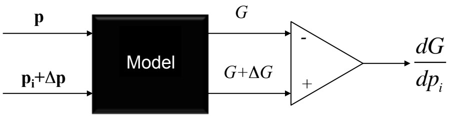
- experimental design
- model reduction

- optimal control
- parameter estimation

- data assimilation
- dynamic constrained optimization

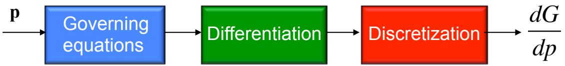
Approaches

(i) Finite difference approach



(ii) Forward approach

Discrete



Continuous

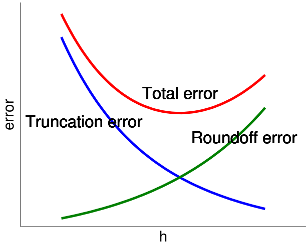
(iii) Adjoint approach

Discrete

Continuous

Finite difference

- Easy to implement
- Inefficient for many parameter case, due to one-at-a-time (OTA)
- Error depends critically on the perturbation value h



Forward approach

Discrete

- Governing equation

$$\mathcal{M}\frac{dy}{dt} = f(t,y), \quad y(t_0) = y_0(p)$$

- Discretization with a time stepping algorithm (e.g. backward Euler)

$$\mathcal{M}y_{n+1} = \mathcal{M}y_n + hf(t_{n+1}, y_{n+1})$$

- Differentiation on parameter such that solution sensitivities $\mathbf{S}_{\ell,n} = dy_n/dp_\ell, 1 \leq \ell \leq m$

$$\mathcal{M}\mathbf{S}_{\ell,n+1} = \mathcal{M}\mathbf{S}_{\ell,n} + h\left(\mathbf{f_y}(t_{n+1}, y_{n+1})\mathbf{S}_{\ell,n+1} + \mathbf{f_p}(t_{n+1}, y_{n+1})\right)$$

Continuous

- Governing equation (same as above)
- Differentiation on parameter such that solution sensitivities $\mathbf{S}_\ell = dy/dp_\ell, 1 \leq \ell \leq m$

$$\mathcal{M}\frac{d\mathbf{S}_\ell}{dt} = \frac{\partial f}{\partial y}(t,y)\mathbf{S}_\ell + \frac{\partial f}{\partial p_\ell}(t,y), \quad \mathbf{S}_\ell(t_0) = \frac{\partial y_0}{\partial p_\ell}$$

- Solving for $\mathbf{S}_\ell$ with the same time stepping algorithm and same step size h gives

$$\mathcal{M}\mathbf{S}_{\ell,n+1} = \mathcal{M}\mathbf{S}_{\ell,n} + h\left(\mathbf{f_y}(t_{n+1}, y_{n+1})\mathbf{S}_{\ell,n+1} + \mathbf{f_p}(t_{n+1}, y_{n+1})\right)$$

Discrete Adjoint approach

Assume the ODE/DAE is integrated with a one-step method (e.g. Euler, Crank-Nicolson, or Runge-Kutta)

$$y_{k+1} = \mathcal{N}_k(y_k), \quad k = 0, \ldots, N-1, \quad y_0 = \gamma(p) \tag{1}$$

The exact objective function $\Psi = g(y(t_F))$ is approximated by $\Psi^d = g(y_N)$. We use the Lagrange multipliers $\lambda_0, \ldots, \lambda_N$ to account for the ODE/DAE constraint

$$\mathcal{L} = \Psi^d - (\lambda_0)^T (y_0 - \gamma) - \sum_{k=0}^{N-1} (\lambda_{k+1})^T (y_{k+1} - \mathcal{N}(y_k)) \tag{2}$$

Discrete adjoint approach (cont.)

Differentiating this function at p and reorganizing yields

$$\frac{d\mathcal{L}}{dp} = (\lambda_0)^T \frac{d\gamma}{dp} - \left(\frac{dg}{dy}(y_N) - (\lambda_N)^T\right) \frac{\partial y_N}{\partial p} - \sum_{k=0}^{N-1} \left((\lambda_k)^T - (\lambda_{k+1})^T \frac{d\mathcal{N}}{dy}(y_k)\right) \frac{\partial y_k}{\partial p} \tag{3}$$

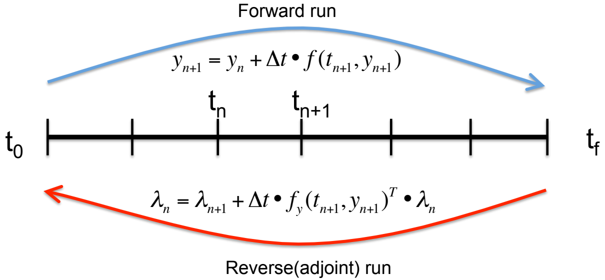
By defining λ to be the solution of the discrete adjoint model

$$\lambda_N = \left(\frac{dg}{dy}(y_N)\right)^T, \quad \lambda_k = \left(\frac{d\mathcal{N}}{dy}(y_k)\right)^T \lambda_{k+1}, \quad k = N-1, \ldots, 0 \tag{4}$$

Then we will have

$$\nabla_p \Psi^d = \left(\frac{d\gamma}{dp}\right)^T \lambda_0$$

Discrete adjoint approach (cont.)



Properties

- The adjoint equation (4) is solved backward in time
- Only one backward run is needed to compute the sensitivities
- Efficient for many parameters and few objective functions
- Need to be derived for the specific time stepping method
- If the simulation problem is nonlinear, the adjoint is linear

Implementation

- The backward run follows the same trajectory
- The Jacobian in the forward run can be reused
- Need to checkpoint the states and time points in the forward run

Continuous adjoint approach

Continuous adjoint equation reads

$$\frac{d\lambda}{dt} = -\mathbf{f_y}^T(t, y)\lambda, \quad \lambda(t_F) = \nabla_y g(t_F)$$

Theoretically adjoint and forward equations can be solved with different time stepping algorithms

Even if solved with the same time stepping algorithm and the same step size, continuous adjoint is inconsistent with discrete adjoint

continuous backward Euler	discrete backward Euler
$\lambda_n = \lambda_{n+1} + (-h)(-\mathbf{f_y}(t_n, y_n))^T \lambda_n$	$\lambda_n = \lambda_{n+1} + h(\mathbf{f_y}(t_{n+1}, y_{n+1}))^T \lambda_n$

Unfortunately the objective function depends on the numerical solution, not the exact solution; this may cause the optimization procedure converge slowly or even not to converge

Make the right choice

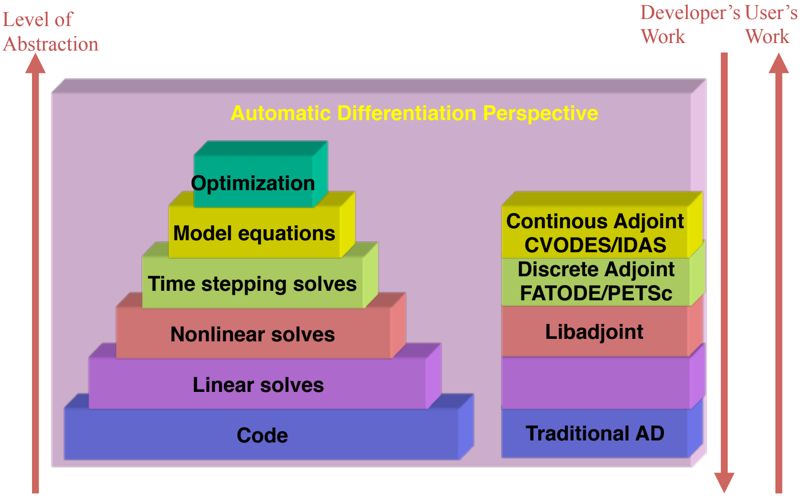
number of parameters $>>$ number of functions	$\Rightarrow$	Adjoint
number of parameters $<<$ number of functions	$\Rightarrow$	Forward
optimization	$\Rightarrow$	Discrete adjoint

Why PETSc and discrete adjoint?

- A large number of users and applications
- A rich set of time stepping solvers and sophisticated nonlinear/linear solvers
- Motivated by optimization problems
- Comparison with existing tools

	SUNDIALS (LLNL)	FATODE (Virginia Tech)	PETSc-SA (ANL)
start year	$\sim$ 2000	2010	2014
problem type	ODE/DAE	ODE	ODE/DAE
language	C	Fortran/MATLAB	C
time stepping	multistep	Runge-Kutta type	ERK, THETA (Extensible)
adjoint	continuous	discrete	discrete
checkpointing	external+recomputation	in-memory (Extensible)	all external (Extensible)

Another perspective of adjoints from Automatic Differentiation



Adjoint sensitivity in PETSc

- General form of the objective function

$$G = g(y(t_F)) + \int_{t_0}^{t_F} r(t, y) dt$$

- Derived from the extended system

$$\begin{aligned} \dot{y} &= f(t, y) \\ \dot{p} &= 0 \\ \dot{q} &= r(t, y) \end{aligned}$$

- Sensitivity w.r.t. initial values

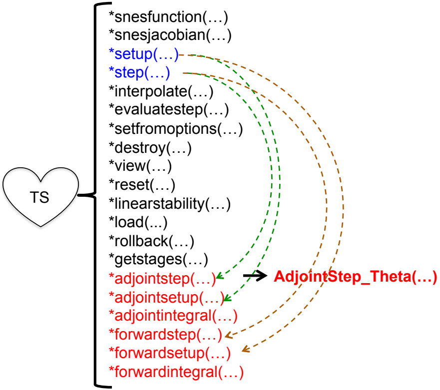
$$\lambda_n = \lambda_{n+1} + h \left(\mathbf{f_y} (t_{n+1}, y_{n+1}) \right)^T \lambda_n + h \left(r_y (t_{n+1}, y_{n+1}) \right)^T$$

$$\mu_n = \mu_{n+1} + h \left(\mathbf{f_p} (t_{n+1}, y_{n+1}) \right)^T \lambda_n + h \left(r_p (t_{n+1}, y_{n+1}) \right)^T$$

- Sensitivity w.r.t. parameters
- Sensitivity of the integrals in the objective function

Adjoint sensitivity in PETSc (cont.)

- Implemented as TS operators
- Add a new object TSTrajectory for checkpointing
- TSTrajectory can also be used for postprocessing



Usage

$$\dot{y} = z$$
$$\dot{z} = \mu\left((1-y^2)z - y\right)$$

```
TSSetSaveTrajectory(ts); //checkpointing
TSSetIFunction(ts,NULL,IFunction,&user);
TSSetIJacobian(ts,A,A,IJacobian,&user);
...
TSSolve(ts,x);
TSSetCostGradients(ts,2,lambda,mup);
TSAdjointSetRHSJacobian(ts,Jacp,RHSJacobianP,&user);
TSAdjointSolve(ts);
```

IFunction: $M\dot{x} - f(x) = \begin{bmatrix} \dot{y} - z \\ \dot{z} - \mu\left((1-y^2)z - y\right) \end{bmatrix}$

IJacobian: $M \cdot \text{shift} - \frac{df}{dx} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \text{shift} - \begin{bmatrix} 0 & 1 \\ \mu(-2\,y\,z - 1) & \mu(1-y^2) \end{bmatrix}$

RHSJacobianP: $\frac{df}{dp} = \begin{bmatrix} 0 \\ ((1-y^2)z - y) \end{bmatrix}$

Forward sensitivity in PETSc

- One solution sensitivity variable $\mathbf{S}_\ell$ corresponds to one parameter

$$\mathcal{M}\mathbf{S}_{\ell,n+1} = \mathcal{M}\mathbf{S}_{\ell,n} + h\left(\left(\mathbf{f_y}(t_{n+1}, y_{n+1})\mathbf{S}_{\ell,n+1} + \mathbf{f_p}(t_{n+1}, y_{n+1})\right)\right) \quad (6)$$

- Initial values are also considered as parameters
- The sensitivities of integral functions

$$q = \int_{t_0}^{t_F} r(t, y, p)dt$$

w.r.t. model parameters can be computed as

$$\frac{\partial q}{\partial p} = \int_{t_0}^{t_F} \left(\frac{\partial r}{\partial y}(t, y, p)\mathbf{S} + \frac{\partial r}{\partial p}(t, y, p)\right) dt$$

Usage

```
TSSetIFunction(ts,NULL,IFunction,&user);
TSSetIJacobian(ts,A,A,IJacobian,&user);
TSSetForwardSensitivities(ts,3,sensi);
TSForwardSetRHSJacobianP(ts,jacp,RHSJacobianP,&user);
…
TSSolve(ts,x);
```

Application in power system

$$M\dot{x} = f(t, x, y, p), \quad x(t_0) = I_{x0}(p) \qquad \text{(Machine ODEs)}$$
$$0 = g(t, x, y, p), \quad y(t_0) = I_{y0}(p) \qquad \text{(Network algebraic equations)}$$

- $x \rightarrow$ machine dynamic variables
- $y \rightarrow$ network + machine algebraic variables
- g_y is invertible (semi-explicit index-1 DAE)

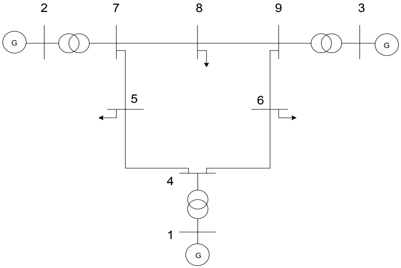


Figure: 9 bus problem

Application in power system (cont.)

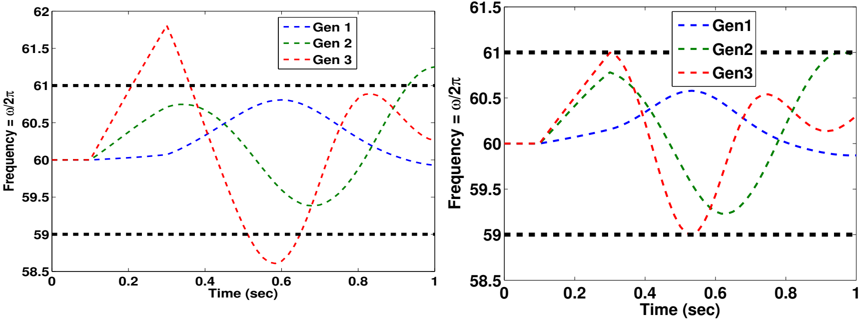
Dynamics security constrained Optimal Power Flow problem needs to consider a dynamic constraint aggregation

$$H(x(p,t), y(p,t)) = \int_0^T h(x(p,t), y(p,t))\, dt \le \rho$$

An example of $H(x,y)$: Generator frequency, $\omega \subset x$, deviation

$$H(x,y) = \int_0^T \left[\max\left(0, \omega(t) - \omega^+, \omega^- - \omega(t)\right)\right]^\eta \, dt$$

Computing partial of the dynamic constraint, H_p, was difficult!



Results

Basic settings

	dof.	No. of parameters	No. of functions
9 bus	54	24	3
118 bus	884	344	54

CPU time comparison

	forward	adjoint	simulation
9 bus	3.82 s (7.3x)	1.80 s (3.5x)	0.52 s (1x)
118 bus	2132.61 s (630.9x)	29.86 s (8.8x)	3.38 s (1x)

Forward approach is very costly

$$\frac{\partial q}{\partial p} = \int_{t_0}^{t_F} \left(\frac{\partial r}{\partial y}(t, y, p)\mathbf{S} + \frac{\partial r}{\partial p}(t, y, p) \right) dt$$

Sensitivity analysis for hybrid systems

The dynamic behavior of many systems may include discrete-event dynamics, switching action and jump phenomena. Such nonlinear nonsmooth hybrid systems can be complicated.

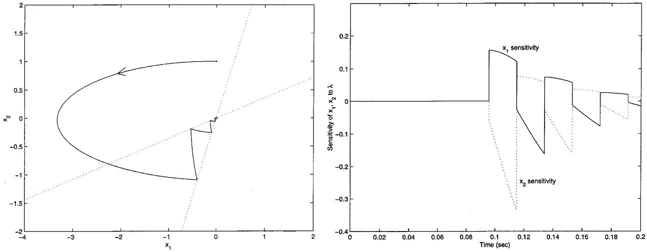
Example [Hiskens et. al. 2000]

$$\dot{x} = A_i x$$

where the matrix A_i changes from

$$A_1 = \begin{bmatrix} 1 & -100 \\ 10 & 1 \end{bmatrix} \quad \text{to} \quad A_2 = \begin{bmatrix} 1 & 10 \\ -100 & 1 \end{bmatrix}$$

when $x_2 = 2.75x_1$ and from A_2 to A_1 when $x_2 = 0.36x_1$. Initially $x_0 = [0 \quad 1]^T$ and $i = 1$.



Jump condition

$$\begin{aligned} y^{(1)}(t_0) &= \theta(p) \\ \dot{y}^{(1)} &= \mathbf{f}^{(1)}(t, y^{(1)}), \quad t \in [t_0, \tau] \\ \gamma(y^{(1)}(\tau)) &= 0 \\ \dot{y}^{(2)} &= \mathbf{f}^{(2)}(t, y^{(2)}), \quad t \in (\tau, t_F] \end{aligned}$$

- The states are continuous at the junction time

$$y^{(2)}(\tau) = y^{(1)}(\tau)$$

- $\mathbf{f}^{(1)}$, $\mathbf{f}^{(2)}$, γ are $\mathcal{C}^1$
- Transversality condition must be satisfied

$$\frac{d\gamma}{dy}(\tau)\mathbf{f}^{(1)}(\tau, y^{(1)}(\tau)) \neq 0$$

Jump condition for discrete adjoint

$$\lambda^{(1)}_{N^{(1)}} = \left(\mathbf{I} + \left(\frac{\partial y^{(2)}_{N^{(1)}}}{\partial t} - \frac{\partial y^{(1)}_{N^{(1)}}}{\partial t} \right) \frac{\frac{d\gamma}{dy}(y^{(1)}_{N^{(1)}})}{\frac{d\gamma}{dy}(y^{(1)}_{N^{(1)}}) \cdot \frac{\partial y^{(1)}_{N^{(1)}}}{\partial t}} \right)^T \cdot \lambda^{(2)}_{N^{(1)}}$$

Use event monitor

- Event detection in PETSc EventFunction(...)

```
PetscErrorCode EventFunction(TS ts,PetscReal t,Vec U,PetscScalar *fvalue,void *ctx)
{ AppCtx          *actx=(AppCtx*)ctx;
  const PetscScalar *u;
  …
  VecGetArrayRead(U,&u);
  if (actx->mode == 1) { fvalue[0] = u[1]-actx->lambda1*u[0];
  }else if (actx->mode == 2) { fvalue[0] = u[1]-actx->lambda2*u[0];}
  VecRestoreArrayRead(U,&u);
  …
}
```

- Event handling in PETSc PostEventFunction(...)

```
PetscErrorCode PostEventFunction(TS ts,PetscInt nevents,PetscInt
event_list[],PetscReal t,Vec U,PetscBool forwardsolve,void* ctx)
{ AppCtx        *actx=(AppCtx*)ctx;
  …
  if (!forwardsolve) {ShiftGradients(ts,U,actx); }
  if (actx->mode == 1) { actx->mode = 2;
  } else if (actx->mode == 2) {actx->mode = 1;}
  …
}
```

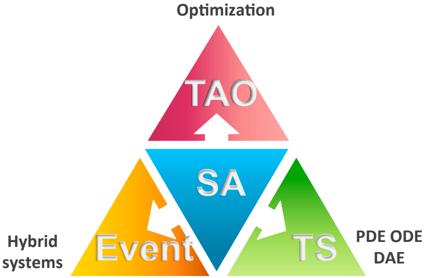
- Works seamlessly with sensitivity analysis

Ongoing and future work

- Use ADIC to generate Jacobians (in a matrix-free manner) automatically; use the matrix type MATSHELL and overload the matrix-vector multiplication operator
- Interface with libMesh (a framework for solving PDEs using arbitrary unstructured mesh in parallel) to enable more applications
- Extend to more advanced time-stepping algorithms
- Develop heterogeneous checkpointing schemes

Summary

- Developed forward and discrete adjoint sensitivity analysis in PETSc
- Established the theory of discrete adjoint for hybrid systems
- Explored the application in power system
- Successful application requires to incorporate multiple components



Theoretical methods are now sufficiently advanced so that it is intellectually dishonest to perform modeling without sensitivity analysis.

— Charles E. Kolb (Herschel Rabitz, 1989, Science)

Thank you!

Acknowledgment:
I am thankful to Shri for the help on the power system examples, to Kamil for the help on deriving the jump condition of hybrid systems, to Krishna and Paul for the help on ADIC, to Satish and Barry for the help on PETSc.