Adjoints without Tears

William Symes

Rice University

February 2012

Why adjoints

Model space $\mathcal{M}$, data space $\mathcal{D}$

[$\mathcal{M}$, $\mathcal{D}$ open subsets of Hilbert spaces]

Prediction or modeling or "forward" operator $F : \mathcal{M} \rightarrow \mathcal{D}$

Inverse problem: given $d \in \mathcal{D}$ ("observed data"), find $m \in \mathcal{M}$ so that

$$F[m] \simeq d$$

Why adjoints

Least squares formulation: choose x to minimize

$$J[m] = \frac{1}{2}\|F[m] - d\|^2$$

Algorithms: any relative of Newton needs gradient

$$\nabla J[m] = DF[m]^T r, \; r = F[m] - d$$

Note that only the matrix-vector product is needed, not the matrix of $DF[m]^T$ in any basis

Derivatives, adjoints, and time-stepping

Seismic inverse problems - forward map defined via solution of wave equations - time or frequency domain

Finite difference in time:

$$u^{n+1} = u^n + \Delta t \, h[m, u^n], \; n = 0, ...N-1 \; u^0 = 0$$

u = *state vector* = time snap-shot (Cauchy data) of dynamical fields (pressure, velocity, stress,...), $u^n \simeq u(n\Delta t)$.

Discrete time data: $F[m]^n = Su^n$, S = *sampling operator*

NOTE: u^n implicitly function of m.

Derivatives, adjoints, and time-stepping

Differentiation: formal, implicit

$$\delta u^{n+1} = \delta u^n + \Delta t D_m\, h[m, u^n]\delta m + \Delta t D_u\, h[m, u^n]\delta u^n,$$

$$n = 0, ..., N-1;\ \delta u^0 = 0$$

Then $DF[m]\delta m = S\delta u^n$. Define vectors, matrices:

$$U_i = \delta u^i;\ B_i = \Delta t D_m\, h[m, u^i];$$

$$H_{i,i} = I;\ H_{i+1,i} = -\Delta t D_u\, h[m, u^i];\ H_{i,j} = 0,\ i \neq j, j+1$$

Then

$$HU = B\delta m$$

Derivatives, adjoints, and time-stepping

so

$$DF[m]\delta m = SH^{-1}B\delta m$$

whence

$$DF[m]^T r = B^T(H^T)^{-1}S^T r$$

Pull this apart:

$$DF[m]^T r = B^T W, \; H^T W = S^T r$$

$W = (w^0, ..., w^N)^T =$ *adjoint state* vector

Derivatives, adjoints, and time-stepping

picking this apart further, B is a block-column matrix, so

$$B^T W = \sum_{n=0}^{N} B_n w^n \tag{1}$$

also

$$w^n = w^{n+1} + \Delta t D_u h[m, u^n]^T w^{n+1}, i = 0, ..., N-1; \;\; w^N = 0 \tag{2}$$

Leads to *Adjoint State Method*:

- solve equation (2) *backwards* in time, starting with $n = N - 1$;
- in course of iteration, *accumulate* $DF[m]^T r = B^T W$ by adding in successive terms in sum (1).

Derivatives, adjoints, and time-stepping

Notes:

- in seismic imaging literature, equation (1) is (a version of) the *imaging condition*, and equation (2) is the *backpropagation* or *reverse-time* equation
- all of the preceding has complete analogue for continuum problems (differential equations) - see Plessix 2006,...,Chavent 1974 - comes from control theory of PDEs (J.-L. Lions, 1968)
- computations up to this point coded abstractly in IWAVE++, no need to write these loops!!!
- Remaining problem: figure out how to compute derivatives $D_u\, h[m, u^n]\delta u^n$, $B_{n+1}\delta m = D_m\, h[m, u^n]\delta m$ and their adjoints $D_u\, h[m, u^n]^T w^{n+1}$, $B_{n+1}^T w^{n+1} = D_m\, h[m, u^n]^T w^{n+1}$

Rules for differentiating stencils

A typical time-stepping rule: (2,4) staggered grid pressure update in 1D, $p_j^n \simeq p(n\Delta t, j\Delta x)$ etc.:

$$p_j^{n+1} = p_j^n + (c_1(v_{j+1/2}^{n+1/2} - v_{j-1/2}^{n+1/2}) + c_2(v_{j+3/2}^{n+1/2} - v_{j-3/2}^{n+1/2})) * \kappa_j$$

c_1, c_2 are scheme constants. Velocity grid offset by 1/2 cell in space and time from pressure grid.

Part of $u^{n+1} = u^n + \Delta t\, h[m, u^n]$, where $u^n = (p^n, v^{n+1/2})$, $m = (\kappa, \rho)$

Rules for differentiating stencils

Translation into code: update form - only one time level of p, v retained ($\kappa \sim \mathtt{mp}$) - *stencil in update form*

```
p[j] += mp[j] * (c1*(v[j]-v[j-1])+
                 c2*(v[j+1]-v[j-2]));
```

Differentiate using formal Leibnitz rule: `v, mp` are indep. variables, gives *perturbation stencil*

```
dp[j] += mp[j] * (c1*(dv[j]-dv[j-1])+
                  c2*(dv[j+1]-dv[j-2])) +
         dmp[j] * (c1*(v[j]-v[j-1])+
                  c2*(v[j+1]-v[j-2]));
```

RICE

Rules for adjoint stencils

- identify input perturbation variables on the RHS, as opposed to constants

```
dp[j] += mp[j] * (c1*(dv[j]-dv[j-1])+
                  c2*(dv[j+1]-dv[j-2])) +
        dmp[j] * (c1*(v[j]-v[j-1])+
                  c2*(v[j+1]-v[j-2]));
```

Rules for adjoint stencils

- for each input perturbation variable, add an update rule with the same multiplier and the output variable to obtain adjoint stencil:

```
dv[j  ] +=  mp[j]*c1*dp[j];
dv[j-1] += -mp[j]*c1*dp[j];
dv[j+1] +=  mp[j]*c2*dp[j];
dv[j-2] += -mp[j]*c2*dp[j];
dmp[j ] +=  dp[j] * (c1*(v[j]-v[j-1])+
                     c2*(v[j+1]-v[j-2]));
```

Important note: loop limits (on `j`) should be *same* as in reference stencil

Rules for adjoint stencils

- rewrite non-increment update (=) as increment (+=), apply preceding rule

Example: Dirichlet boundary condition stencil for (2,4) scheme

```
dp[-1] = - dp[1];
dp[0 ]= 0;
```

rewrite in increment form:

```
dp[-1] += -dp[-1] - dp[1];
dp[0 ] += -dp[0];
```

so adjoint is

```
dp[1 ] += -dp[-1];
dp[0 ] += -dp[0];
dp[-1] += -dp[-1] ;
```

(can translate back to assignment form - careful about order!)

RICE

Rules for adjoint stencils

- reverse the order of blocks of independent updates

Example: in 1D (2,4) scheme, natural order of pressure perturbation array update is

- update interior nodes (1, 2,....) of `dp` using perturbation stencil;
- update boundary nodes (-1, 0) of `dp` using Dirichlet stencil

In adjoint pressure perturbation update, reverse order:

- update boundary nodes (-1, 0, 1) of `dp` using adjoint Dirichlet stencil
- update `dv`, `dmp` at whatever node indices occur in adjoint stencil loop

Rules for adjoint stencils

Exercise for reader: apply these rules to matrix multiplication (after all, a stencil is a compact representation of a sparse matrix multiply!) and verify that in fact they produce a correct algorithm for multiplication by the transpose matrix.

A semi-serious example

`adjexpl.c`

code for forward, linearized, and adjoint operators, (2,4) staggered grid scheme for 2D acoustics.

Includes *dot product test* for components (individual stencils operators composing $D_u h(m,u)$, $D_m h(m,u)$, sampling operator S), and full program ($DF[m]\delta m$, $DF[m]^T r$).

Will be posted with talk slides

A semi-serious example

Dot product test evaluates alleged adjoint pair of operators A, B: compare

$$p_1 = (Ax)^T y \text{ and } p_2 = x^T B y$$

for random x, y. $B =$ acceptable approximation to $A^T \Rightarrow$ $p_1 = p_2 + \epsilon$, $\epsilon =$ low multiple (10^2) of `macheps`. Practically: adjoint coding error $\Rightarrow p_1, p_2$ not that close (discretization vs. roundoff).

`adjexpl.c` includes dot product test - passes, you try it!

A more serious example: IWAVE++

IWAVE = framework for regular grid FD modeling with high accuracy schemes, boundary conditions, standard data formats, loop and task level parallelism via MPI & OpenMP

- started as SEAM QC code (Fehler & Keliher 2012), released SEG 09, v2.0 coming 12Q2
- common services - malloc, MPI comm, i/o, job control, etc.
- apps - staggered grid acoustics, isotropic elasticity, more to come

IWAVE++ = imaging and inversion framework based on IWAVE, RVL = Rice Vector Library = optimization and linear algebra services

RICE

A more serious example: IWAVE++

RVL provides dot product etc., IWAVE++ includes middleware interface

AND handles time-loop aspects of adjoint state (checkpointing)

[demo]

A more serious example: IWAVE++

What IWAVE++::asg++ must deal with that adjexpl.c ignores:

- PML boundary conditions - no big deal, just more arrays, equations
- trace sampling (S) includes spline interpolation onto output time grid - internal, archival time steps generally not same - adjoint sampling implemented in IWAVE trace i/o
- material parameter array loads/saves (part of B) involves grid extension, index shifts - may involve stencil (eg. shifted density grids) hence communication - IWAVE grid i/o uses `MPI_Reduce`
- parallelism - column orientation of adjoint $\Rightarrow$ *different communication pattern* than forward modeling

A more serious example: IWAVE++

Domain decomposition & adjoint loops: schematic forward stencil

$$
\begin{aligned}
y_1 \quad &+= \quad a_1 * x_1 + \ldots \\
y_2 \quad &+= \quad b_1 * x_1 + \ldots
\end{aligned}
$$

Blue = domain 1; green = domain 2

IWAVE: every field variable *belongs* to one domain ("computational"), is shared with other domains via ghost cells

x_1 is in *send buffer* for its domain 1, x_1 is in receive buffer of domain 2

Forward outputs y_1, y_2 each fully updated in proper computational domain.

A more serious example: IWAVE++

Adjoint:

$$\begin{aligned} x_1 &\mathrel{+}= a_1 * y_1 + \dots \\ x_1 &\mathrel{+}= b_1 * y_2 + \dots \end{aligned}$$

Update for x_1 *partially complete* in each domain - neither is correct!

Solution:

- make temp copy of send buffer (contains variables belonging to domain, to be shared with neighbor domains)
- *reverse* communication pattern of forward: copy receive buffers from each neighbor into send buffer (`MPI_SendRecv`), add to temp copy
- after all neighbors visited, copy temp buffer into send buffer - now all variables belonging to domain are fully updated

[demo]

Conclusion

- machine precision adjoints on large scale feasible "without tears", avoid coupling of optimization, simulator accuracy control
- simple rules produce correct time-step code for both serial and parallel adjoint state
- abstract interface code (IWAVE++) can provide convenient and portable time loop services
- IWAVE++ release to TRIP sponsors 12Q2

Thanks to: IWAVE team (Igor, Tanya, Xin, Dong, Marco), NSF, sponsors of TRIP

RICE