Automatically generated tangent and adjoint C codes

Michael Voßbeck, Ralf Giering, and Thomas Kaminski

Copy of presentation at http://www.FastOpt.com

Outline

- Motivation
- AD-Tool
- Applications
  - Roeflux
  - 2streamsRT
  - TAU-ij
  - GasNetOpt
- Performance
- Summary

Motivation

- Reverse mode AD for C/C++ only implemented as operator overloading,
  e.g. CppAD (Bell) or ADOL-C (Griewank et al., 1996) -> Walther
- Only available Source-to-Source tool for C/C++:
  ADIC (Bischof et al., 1997; Hofland et al, 2002), so far restricted to forward mode
- This talk:
  Demonstrate feasibility of reverse mode source-to-source transformation for ANSI-C by applying our AD-Tool to four different C codes

AD-Tool

- **Applies same philosophy as TAF (Giering and Kaminski, 1998)**
- **Command-line tool**
- **Two options so far (`-f`/`-r` : forward/reverse mode)**
- **Uses (simplified) implementations of a subset of TAF algorithms (e.g. ERA, Giering and Kaminski, 2002)**
- **No activity analysis yet (all floating point variables are treated as active)**
- **No AD directives (e.g. TAF STORE directive) yet**
- **Function code has to be preprocessed (e.g. by cpp)**
- **Currently generated adjoint / tangent code operates in pure mode, i.e. evaluates gradient but not function (full mode is easy to add)**

*Fast***Opt**

Roeflux

(Provided by Paul Cusdin, Jens-Dominik Mueller)

- **Roe Solver (1997) of CFD-code EULSOLDO (Cusdin and Mueller, 2003)**
- **Compute the gradient of subroutine `flux` in forward/reverse mode**
- **8 independent variables (each 4 components of left and right cell values)**
- **1 dependent variable (the sum over the four components of the residual)**
- **Use of FastOpt standard driver for scalar-valued functions to run the code**

Roeflux

- C code is generated by f2c from original Fortran 77 version (140 lines without comments)
- Generated C code contains basic arithmetics, intrinsic function calls (`sqrt`), control flow elements (`for`, `if`, `conditional-expression`)
- f2c also inserts simple pointer arithmetics (to mimic fortran array indexing)

```
/* Subroutine */ int flux(doublereal *ql, doublereal *qr,
    doublereal *sinal, doublereal *cosal, doublereal *ds,
    doublereal *r__,doublereal *lambda)
{...
    /* Parameter adjustments */
    --r__;
    --qr;
    --ql;
    /* Function Body */
    qln[0] = ql[2] * *cosal + ql[3] * *sinal;
    qln[1] = ql[3] * *cosal - ql[2] * *sinal;
```

Roeflux

- AD-Tool normalises “nasty” expressions (w.r.t. AD) (e.g. comma-expression)

```
#define abs(x) ((x) >= 0 ? (x) : -(x))
/*      Absolute eigenvalues, acoustic waves with entropy fix.
    */
l[0] = (d__1 = uhat - ahat, abs(d__1));
```

```
d__1 = uhat -ahat;
l[0] = (d__1 >= 0 ? d__1 : -d__1);
```

- Association by name, interface of adjoint routine

```
/* Subroutine */ int model_(integer *n, doublereal *x,
    doublereal *fc)
{ ... }
```

```
void model__ad( integer *n, doublereal *x, doublereal *x_ad,
    doublereal *fc, doublereal *fc_ad )
{ ... }
```

2streamsRT

(Collaboration with B. Pinty, N. Gobron, J.-L. Widlowski, T. Lavergne, M. Verstraete, JRC, Ispra)

- **Simplified (one-dimensional) radiative transfer (RT) model for efficient retrieval of vegetation canopy properties from remote sensing data (Pinty et al., JGR, 2004)**
- **Model has to be calibrated (parameter estimation problem)**
- **Requires gradient of scalar-valued misfit function with respect to three independent variables (parameters)**
- **Function code essentially consists of basic numerical computations with intrinsic function calls (`exp`, `cos`, `sqrt`) and comprises 56 lines**
- **Generated adjoint code has a length of 215 lines (with one declaration / statement per line)**

2streamsRT

- **Recomputation of required variables in the adjoint code (ERA):**

```
secnd_term1 = (1. - k*mu0)*(alpha2 + k*gamma3)*expktau;
secnd_term2 = (1. + k*mu0)*(alpha2 - k*gamma3)/expktau;
secnd_term3 = 2. * k * (gamma3 - alpha2*mu0)*tmp3;
tmp = (w0 * first_term * (secnd_term1 - secnd_term2 -secnd_term3));
*AlbBS = (float)tmp;
if (are_equals(ksquare,0.)) first_term = 1.;
secnd_term1 = (1.+k*mu0)*(alpha1+k*gamma4)*expktau;
secnd_term2 = (1.-k*mu0)*(alpha1-k*gamma4)/expktau;
```

```
/* RECOMP============== begin */
first_term=(1.00000000-ksquare*mu0*mu0)*((k+gamma1)*expktau+(k-
gamma1)/expktau);
first_term=1.00000000/first_term;
secnd_term1=(1.00000000-k*mu0)*(alpha2+k*gamma3)*expktau;
secnd_term2=(1.00000000+k*mu0)*(alpha2-k*gamma3)/expktau;
secnd_term3=2.00000000*k*(gamma3-alpha2*mu0)*tmp3;
/* RECOMP============== end */
w0_ad+=tmp_ad*(first_term*(secnd_term1-secnd_term2-secnd_term3));
first_term_ad+=tmp_ad*(w0*(secnd_term1-secnd_term2-secnd_term3));
```

TAU-ij

(Collaboration with N. Gauger, R. Heinrich, N. Kroll, DLR, Braunschweig)

- **TAU is the DLR's industrial aerodynamics solver for unstructured grids**
- **TAU-ij is a simplified version of TAU (Euler)**
- **Did not generate the adjoint of the whole model but selected a representative routine (`calc_inner_fluxes_mapsp,` 129 lines)**
- **Driver computes the gradient of a scalar (the energy of the flux within a particular cell of the mesh) with respect to 8 independents (state parameters of the neighbouring cell)**

TAU-ij

- **Code essentially operates on a user-defined C structure type (`mesh`) with many fields being pointers to multidimensional arrays**
- **For each (active) structured type, the AD-Tool generates a corresponding adjoint structured type**
- **Adjoint structured type contains the adjoints of the active components from the original structured type**
- **This approach avoids memory allocation for variables that are not required**

TAU-ij

```
#define NPRIM 8
typedef struct mesh
{ ...
  int    ninnerfaces;        // number of inner faces
  int    nboundaryfaces;     // number of boundary faces
  int    (*fpi)[2];          // the 2 neighbours of an inner face
  ...
  double (*li_states)[NPRIM];  // left and
  double (*ri_states)[NPRIM];  // right state of inner faces
  double (*prim)[NPRIM];      // primitive variables
  double (*res)[NCONS];       // residual
  ...
 } mesh;
```

```
typedef struct mesh_ad
{ ...
  double (*li_states_ad)[8];
  double (*ri_states_ad)[8];
  double (*prim_ad)[8];
  double (*res_ad)[3];
  ...
} mesh_ad;
```

TAU-ij

```
mesh grid;
...
void calc_inner_fluxes_mapsp(mesh *grid)
{
  double (*res )[NCONS] = grid->res ;
  double (*l_states)[NPRIM] = grid->li_states;
  double (*r_states)[NPRIM] = grid->ri_states;
...
}
```

```
mesh grid;
mesh grid_ad;
...
void calc_inner_fluxes_mapsp_ad( mesh *grid, mesh_ad *grid_ad )
{
  double (*res)[5];
  double (*l_states)[8];
  ...
}
```

GasNetOpt

(Provided by Marc Steinbach, ZIB, Berlin)

- **GasNetOpt optimises the Load Distribution in public Gas Networks (Ehrhardt and Steinbach, 2005)**
- **Most of the complexity arises from pipelines (hyperbolic PDE for gas dynamics)**
- **Model is implemented in C++ (with use of `namespace` and `templates`).**
- **Ehrhardt and Steinbach use hand coding/ADOL-C for gradient and Hessian computation**
- **Converted two simple routines `Reynolds_Number` and `lambda_Hofer` to pure C code**
- **Compute the derivative of the pipe friction with respect to 4 independents in reverse mode**

GasNetOpt

(Provided by Marc Steinbach, ZIB, Berlin)

```
namespace gas {
{  template<typename Real>
   inline Real
   Reynolds_Number(Real const M, Real const D, Real const eta)
   {
     static Real const Re_c = 2320.0; // critical value for laminar flow
     Real const C = D * eta * Real(M_PI_4L);
     Real Re = M / C;
     if (Re < Real(3) * Re_c) {
       // Replace Re(M) by unique cubic polynomial p(M) on (0, 3 * C * Re_c)
       // having p(0) = Re_c, p'(0) = 0, and a C2-junction at 3 * C * Re_c:
       Real const x = Re / (Real(3) * Re_c);
       Re = Re_c + Re_c * (Real(3) - x) * x * x;
    ...
```

```
void Reynolds_Number_ad( const Real M, Real *M_ad, const Real D, Real *D_ad, const
    Real eta, Real *eta_ad, Real *Re, Real *Re_ad )
{ ...
  C=D*eta*(Real )0.78539816L;
  *Re=M/C;
  if( *Re < 3.00000000*Re_c )
  {
    Real x;
    Real x_ad = 0.00000000;    x=*Re/((Real )3*Re_c);
    x_ad+=*Re_ad*((Re_c*-1*x+Re_c*((Real )3-x))*x+Re_c*((Real )3-x)*x);
    *Re_ad=0;
    *Re_ad+=x_ad*(1/((Real )3*Re_c));
    x_ad=0;
  }
  *M_ad+=*Re_ad*(1/C); ...
```

FastOpt

Performance

- **Test environment such that function code runs as fast as possible (here: icc with options `-O3 -ipo -tpp6`)**

Model	#lines of code	FUNC [s]	TLM / FUNC	ADM/ FUNC
Roeflux (Cusdin, Mueller)	140	6.2E-7	3.3	3.9
2streams (Pinty et al.)	56	4.0E-7	2.5	2.7
TAU-ij (Gauger et al.)	129	3.0E-3	--	2.6
GasNetOpt (Steinbach)	25	2.4E-7	2.4	2.7
Roeflux; F77, TAF	105	5.1E-7	--	2.9

- **Generated code is efficient**
- **Comparison to TAF (Roeflux) indicates some scope for improvement in terms of performance of the generated code**

Summary

- Have demonstrated feasibility of reverse mode source-to-source transformation of C code by building first reverse mode source-to-source transformation tool and applying it to four different codes
- Generated code is efficient (for Roeflux faster than operator overloading)
- AD-Tool serves as starting point for design of TAC++, the TAF equivalent for C/C++
- AD-Tool is valuable already as it can handle some real-life applications and can support hand coders of C/C++ adjoints
- Extension of functionality will be demand-driven, i.e. from application to application
- TAF experience was very helpful and further development will benefit from well proved TAF concepts

Thanks for your attention

More Info:

- **http://FastOpt.com**

*Fast*Opt