UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Algorithmic differentiation: Sensitivity analysis and the computation of adjoints

Andrea Walther
Institut für Mathematik
Universität Paderborn

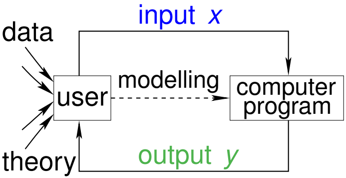
LCCC Workshop on Equation-based Modelling

September 19–21, 2012

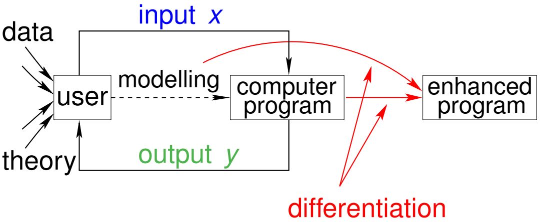
Outline

Computing Derivatives

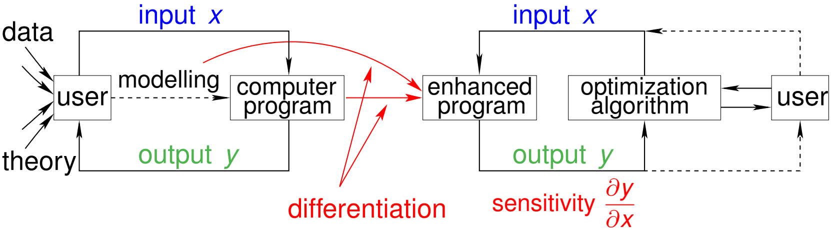
Computing Derivatives



Computing Derivatives



Computing Derivatives



Finite Differences

Idea: Taylor-expansion, $f : \mathbb{R} \to \mathbb{R}$ smooth then

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + h^2 f''(x)/2 + h^3 f'''(x)/6 + \dots \\ \Rightarrow \quad f(x+h) &\approx f(x) + hf'(x) \\ \Rightarrow \quad Df(x) &= \frac{f(x+h) - f(x)}{h} \end{aligned}$$

Finite Differences

Idea: Taylor-expansion, $f : \mathbb{R} \to \mathbb{R}$ smooth then

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + h^2 f''(x)/2 + h^3 f'''(x)/6 + \dots \\ \Rightarrow \quad f(x+h) &\approx f(x) + hf'(x) \\ \Rightarrow \quad Df(x) &= \frac{f(x+h) - f(x)}{h} \end{aligned}$$

- simple derivative calculation (only function evaluations!)
- inexact derivatives
- computation cost often too high

  $F : \mathbb{R}^n \to \mathbb{R} \quad \Rightarrow \quad \mathrm{OPS}(\nabla F(x)) \sim (n+1)\mathrm{OPS}(F(x))$

Analytic Differentiation

- exact derivatives
  - $f(x) = \exp(\sin(x^2)) \Rightarrow$

    $f'(x) = \exp(\sin(x^2)) * \cos(x^2) * 2x$

Analytic Differentiation

- exact derivatives
  - $f(x) \;=\; \exp(\sin(x^2)) \Rightarrow$

    $f'(x) \;=\; \exp(\sin(x^2)) \;*\; \cos(x^2) \;*\; 2x$
  - $\min J(x,u)$ such that $x' = f(x,u) \;+\;$ IC

    reduced formulation: $J(x,u) \to \widehat{J}(u)$

    $\widehat{J}'(u)$ based on symbolic adjoint $\lambda' = -f_x(x,u)^\top \lambda \;+\;$ TC

Analytic Differentiation

- exact derivatives
  - $f(x) = \exp(\sin(x^2)) \Rightarrow$

    $f'(x) = \exp(\sin(x^2)) * \cos(x^2) * 2x$
  - $\min J(x,u)$ such that $x' = f(x,u) + \text{IC}$

    reduced formulation: $J(x,u) \to \widehat{J}(u)$

    $\widehat{J}'(u)$ based on symbolic adjoint $\lambda' = -f_x(x,u)^\top \lambda + \text{TC}$
- cost (common subexpression, implementation)
- legacy code with large number of lines $\Rightarrow$ closed form expression not available
- consistent derivative information?!

Gedruckt von Andrea Walther

Jan 01, 08 21:46 **euler2d.c** Seite 29/30

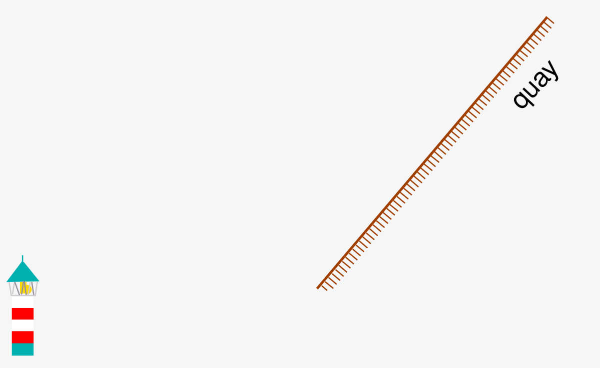
```
  read_input_file(argv[1], &code_control);

  code_control.timestep_type = 0; // calculate timestep size like TAU

  // read in CFD mesh
  read_cfd_mesh(code_control.CFDmesh_name, &gridbase);
  grid[0] = gridbase;

  // remove mesh corner points arizing more than once . . .
  // e.g. for block structured area and at interface between
  // block structured and unstructured area
  remove_double_points( &gridbase, grid);

  // write out mesh in tecplot format
  write_pointdata( name, &(grid[0]));

  // calculate metric of finest grid level
/*     grid[0].xp[ii][1] += 0.00000001; */
  calc_metric(&(grid[0]), &code_control);
  puts("calc_metric ready");

  // create coarse meshes for multigrid, calculate their metric
  // and initialze forcing functions to zero
  for (i = 1; i < code_control.nlevels; i++)
  { create_coarse_mesh(&(grid[i-1]), &(grid[i]));
    init2zero(&(grid[i]), grid[i].force);
  }
  puts("create_coarse_mesh ready");

  // initialize flow field on all grid levels to free stream
  // quantities
  for (i = 0; i < code_control.nlevels; i++)
    init_field(&(grid[i]), &code_control);
  puts("init_field ready");

  // if selected read restart file
  if (code_control.restart == 1)
    read_restart( "restart", grid, &code_control,
                  &first_residual, &first_step);

  // calculate primitive variables for all grid levels and
  // initialize states at the boundary
  for (i = 0; i < code_control.nlevels; i++)
  { cons2prim(&(grid[i]), &code_control);
    init_bdry_states(&(grid[i]));
  }

  // open file for writing convergence history
  conv = fopen("conv.dat", "w");
  fprintf(conv, "title = convergence\n");
  fprintf(conv, "variables = iter, l2res, lift, drag\n");


  level = 0;
  printf("will perform %d steps\n",code_control.nsteps[level]);

  // starting time of computation
  t1 = time(&t1);

  double lift, drag;

  // loop over all multigrid cycles
```

Jan 01, 08 21:46 **euler2d.c** Seite 30/30

```
  for (it = 0; it < code_control.nsteps[level]; it++)
  { double residual;
    lift = 0.0;
    drag = 0.0;

    // calculate actual weight of gradient needed for reconstruction
    if (sum_it+first_step <= code_control.start_2nd_order)
      weight = 0.0;
    else if (sum_it+first_step < code_control.full_2nd_order)
      weight = (double ) (sum_it+first_step - code_control.start_2nd_order) /
                         (code_control.full_2nd_order - code_control.start_2nd_o
rder);
    else
      weight = 1.0;

    // perform a multigrid cycle on current level
    mg_cycle(grid+level, &code_control, weight, &residual);

    // if current level is finest level, calculate boundary forces
    // (lift and drag)
    if (level == 0)
      calc_forces(grid, &code_control, &lift, &drag);

    // set first l2-residual for normalization, if current cycle is
    // the very first of the computation.
    if ((sum_it + first_step) == 0)
      first_residual = (fabs(residual) > 1.0e-10) ? residual: 1.0;

    // print out convergence information to file and standard output
    printf("IT = %d %20.10e %20.10e %20.10e %4.2f\n",
      sum_it, residual / first_residual, lift, drag, weight);
    fprintf(conv, "%d %20.10e %20.10e %20.10e\n",
      sum_it+first_step, residual / first_residual, lift, drag);
    sum_it++;
  }

  // final time of computation
  t2 = time(&t2);

  // print out time needed for the time loop
  printf ("Zeit : %f\n", difftime(t2, t1));
  last_step = first_step + code_control.nsteps[0] ;

  fclose(conv);

  // map solution from cell centers to vertices
  center2point(grid);

  // write out field solution
  write_eulerdata( "euler.dat", grid, &code_control);

  // write out solution on walls
  write_surf( "euler-surf.dat", grid, &code_control);

  // write restart file
  write_restart( "restart", grid, &code_control,
                 first_residual, last_step);

  return 0;
}
```

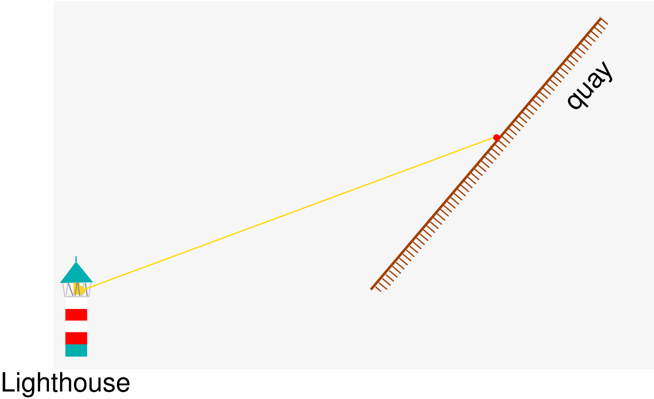
Dienstag Januar 01, 2008 euler2d.c 15/15

The “Hello-World”-Example of AD

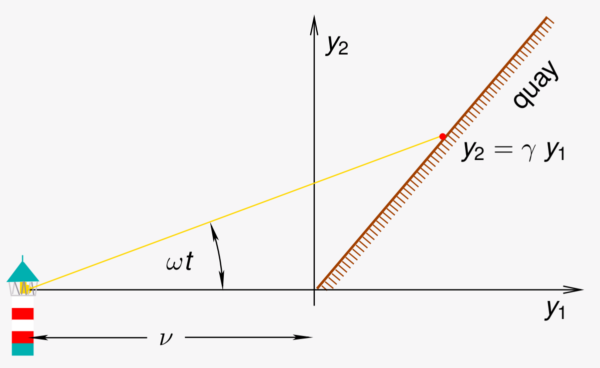


Lighthouse

The “Hello-World”-Example of AD

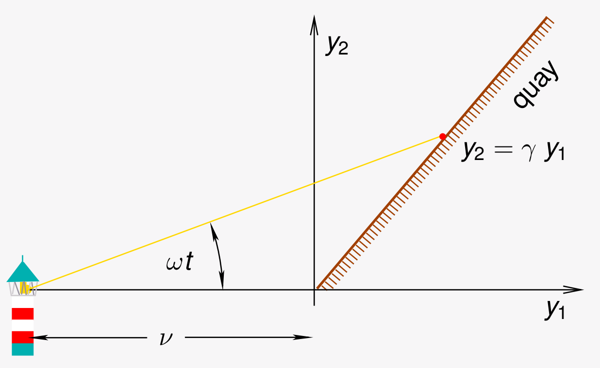


The “Hello-World”-Example of AD



Lighthouse

The “Hello-World”-Example of AD



Lighthouse

$$y_1 = \frac{\nu \tan(\omega\, t)}{\gamma - \tan(\omega\, t)} \quad \text{and} \quad y_2 = \frac{\gamma\, \nu \tan(\omega\, t)}{\gamma - \tan(\omega\, t)}$$

Evaluation Procedure (Lighthouse)

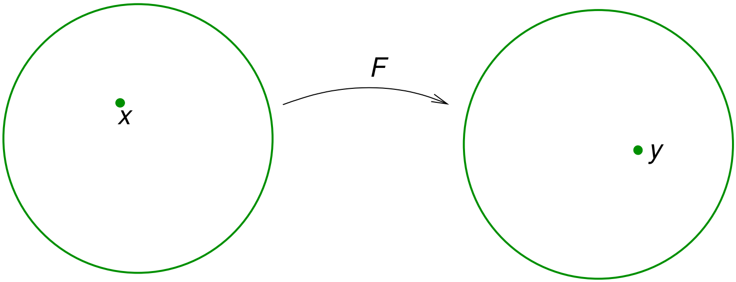
$$y_1 = \frac{\nu \tan(\omega\, t)}{\gamma - \tan(\omega\, t)}$$

$$y_2 = \frac{\gamma\, \nu \tan(\omega\, t)}{\gamma - \tan(\omega\, t)}$$

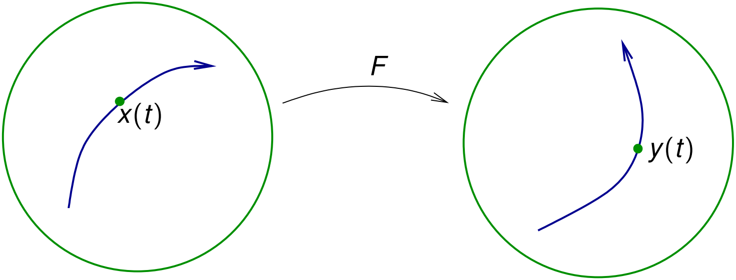
$\Longrightarrow$

$$\begin{array}{lll}
v_{-3} & = x_1 = \nu & \\
v_{-2} & = x_2 = \gamma & \\
v_{-1} & = x_3 = \omega & \\
v_0 & = x_4 = t & \\
\hline
v_1 & = v_{-1} * v_0 & \equiv \varphi_1(v_{-1}, v_0) \\
v_2 & = \tan(v_1) & \equiv \varphi_2(v_1) \\
v_3 & = v_{-2} - v_2 & \equiv \varphi_3(v_{-2}, v_2) \\
v_4 & = v_{-3} * v_2 & \equiv \varphi_4(v_{-3}, v_2) \\
v_5 & = v_4 / v_3 & \equiv \varphi_5(v_4, v_3) \\
v_6 & = v_5 * v_{-2} & \equiv \varphi_6(v_5, v_{-2}) \\
\hline
y_1 & = v_5 & \\
y_2 & = v_6 &
\end{array}$$

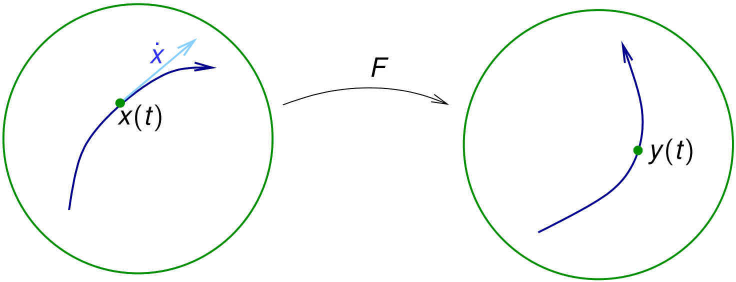
Forward Mode of AD



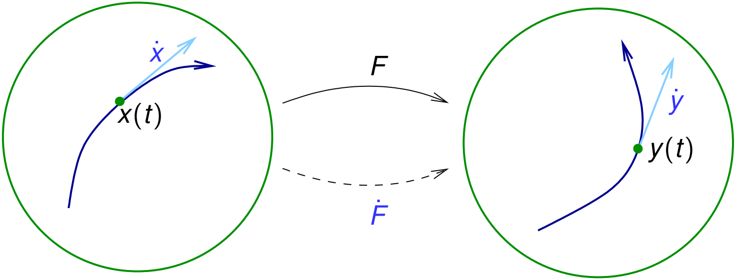
Forward Mode of AD



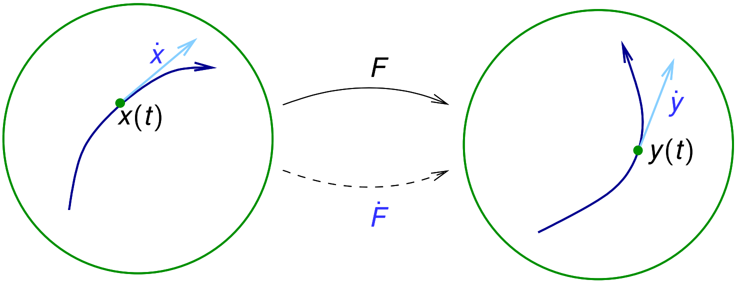
Forward Mode of AD



Forward Mode of AD



Forward Mode of AD



$$\dot{y}(t) \;=\; \frac{\partial}{\partial t} F(x(t)) \;=\; F'(x(t))\,\dot{x}(t) \equiv \; \dot{F}(x, \dot{x})$$

Forward AD (Lighthouse Example)

$$
\begin{array}{lll|lll}
v_{-3} & = & x_1 = \nu & \dot{v}_{-3} & \equiv & \dot{x}_1 \\
v_{-2} & = & x_2 = \gamma & \dot{v}_{-2} & \equiv & \dot{x}_2 \\
v_{-1} & = & x_3 = \omega & \dot{v}_{-1} & \equiv & \dot{x}_3 \\
v_0 & = & x_4 = t & \dot{v}_0 & \equiv & \dot{x}_4 \\
\hline
v_1 & = & v_{-1} * v_0 \\
v_2 & = & \tan(v_1) \\
v_3 & = & v_{-2} - v_2 \\
v_4 & = & v_{-3} * v_2 \\
v_5 & = & v_4 / v_3 \\
v_6 & = & v_5 \\
v_7 & = & v_5 * v_{-2} \\
\hline
y_1 & = & v_6 \\
y_2 & = & v_7
\end{array}
$$

Forward AD (Lighthouse Example)

$$
\begin{array}{lllllll}
v_{-3} & = & x_1 = \nu & & \dot{v}_{-3} & \equiv & \dot{x}_1 \\
v_{-2} & = & x_2 = \gamma & & \dot{v}_{-2} & \equiv & \dot{x}_2 \\
v_{-1} & = & x_3 = \omega & & \dot{v}_{-1} & \equiv & \dot{x}_3 \\
v_0 & = & x_4 = t & & \dot{v}_0 & \equiv & \dot{x}_4 \\
\hline
v_1 & = & v_{-1} * v_0 & & \dot{v}_1 & = & \dot{v}_{-1} * v_0 + v_{-1} * \dot{v}_0 \\
v_2 & = & \tan(v_1) & & \dot{v}_2 & = & \dot{v}_1 / \cos(v_1)^2 \\
v_3 & = & v_{-2} - v_2 & & \dot{v}_3 & = & \dot{v}_{-2} - \dot{v}_2 \\
v_4 & = & v_{-3} * v_2 & & \dot{v}_4 & = & \dot{v}_{-3} * v_2 + v_{-3} * \dot{v}_2 \\
v_5 & = & v_4 / v_3 & & \dot{v}_5 & = & (\dot{v}_4 - \dot{v}_3 * v_5) * (1 / v_3) \\
v_6 & = & v_5 & & \dot{v}_6 & = & \dot{v}_5 \\
v_7 & = & v_5 * v_{-2} & & \dot{v}_7 & = & \dot{v}_5 * v_{-2} + v_5 * \dot{v}_{-2} \\
\hline
y_1 & = & v_6 & & & & \\
y_2 & = & v_7 & & & &
\end{array}
$$

Forward AD (Lighthouse Example)

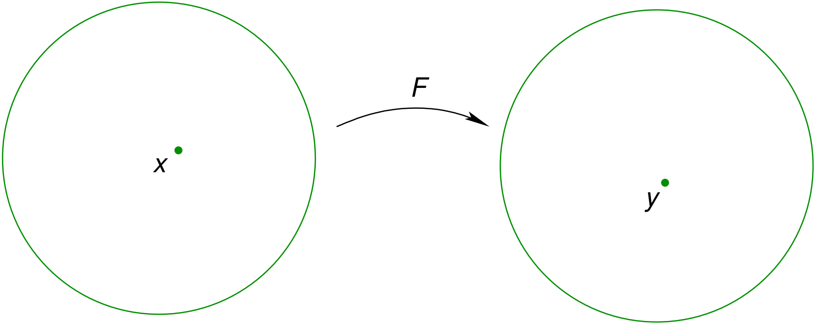
$$
\begin{array}{lllll}
v_{-3} & = & x_1 = \nu & \dot{v}_{-3} & \equiv \dot{x}_1 \\
v_{-2} & = & x_2 = \gamma & \dot{v}_{-2} & \equiv \dot{x}_2 \\
v_{-1} & = & x_3 = \omega & \dot{v}_{-1} & \equiv \dot{x}_3 \\
v_0 & = & x_4 = t & \dot{v}_0 & \equiv \dot{x}_4 \\
\hline
v_1 & = & v_{-1} * v_0 & \dot{v}_1 & = \dot{v}_{-1} * v_0 + v_{-1} * \dot{v}_0 \\
v_2 & = & \tan(v_1) & \dot{v}_2 & = \dot{v}_1 / \cos(v_1)^2 \\
v_3 & = & v_{-2} - v_2 & \dot{v}_3 & = \dot{v}_{-2} - \dot{v}_2 \\
v_4 & = & v_{-3} * v_2 & \dot{v}_4 & = \dot{v}_{-3} * v_2 + v_{-3} * \dot{v}_2 \\
v_5 & = & v_4 / v_3 & \dot{v}_5 & = (\dot{v}_4 - \dot{v}_3 * v_5) * (1 / v_3) \\
v_6 & = & v_5 & \dot{v}_6 & = \dot{v}_5 \\
v_7 & = & v_5 * v_{-2} & \dot{v}_7 & = \dot{v}_5 * v_{-2} + v_5 * \dot{v}_{-2} \\
\hline
y_1 & = & v_6 & \dot{y}_1 & = \dot{v}_6 \\
y_2 & = & v_7 & \dot{y}_2 & = \dot{v}_7
\end{array}
$$

... and the real code

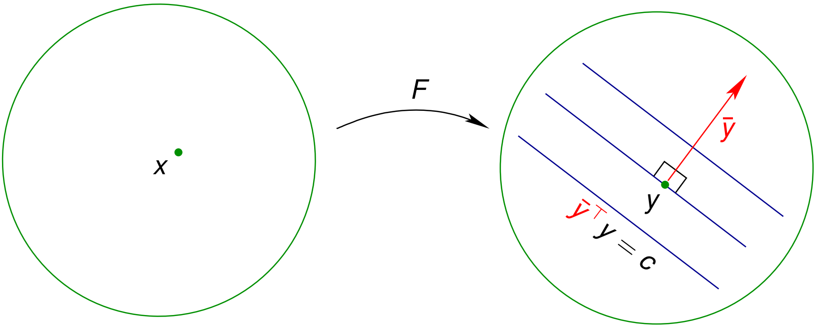
```
void d1_f(double* x, double* d1_x, double* y, double* d1_y)
//$ad indep x d1_x
//$ad dep y d1_y
 {
   double v[2];                double d1_v[2];
   double w1_0 = 0;            double d1_w1_0 = 0;
   ...
   double w1_5 = 0;            double d1_w1_5 = 0;

   d1_w1_0 = d1_x[2];          w1_0 = x[2];
   d1_w1_1 = d1_x[3];          w1_1 = x[3];
   d1_w1_2 = w1_1*d1_w1_0 + w1_0*d1_w1_1;
   w1_2 = w1_0*w1_1;
   d1_w1_3 = 1/(cos(w1_2)*cos(w1_2)) * d1_w1_2;
   w1_3 = tan(w1_2);
   ...
```

using dcc 1.0 (U. Naumann, RWTH Aachen)

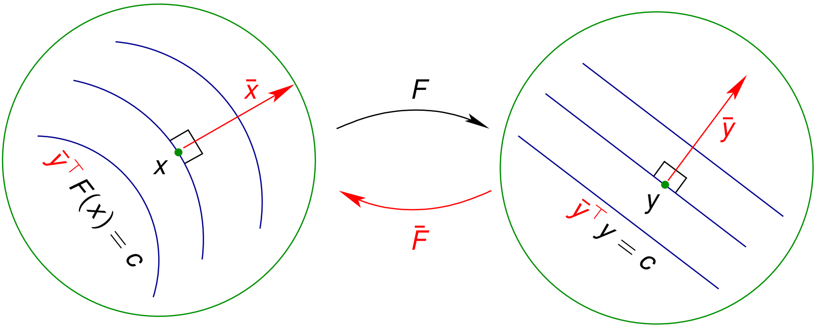
Reverse Mode of AD



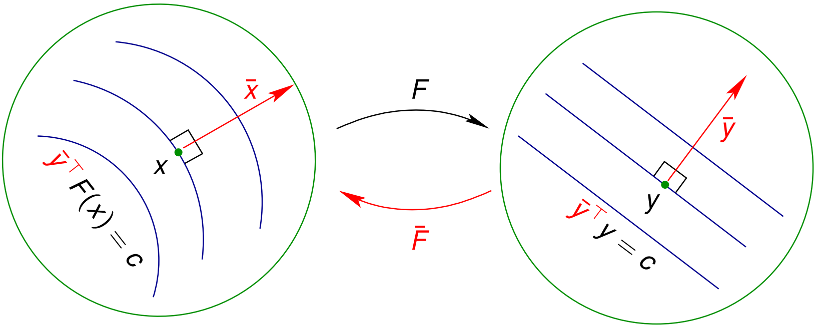
Reverse Mode of AD



Reverse Mode of AD



Reverse Mode of AD



$$\bar{x}^\top \equiv \bar{y}^\top F'(x) = \nabla_x \langle \bar{y}^\top F(x) \rangle \equiv \bar{F}(x, \bar{y})$$

Reverse Mode (Lighthouse)

$$v_{-3} = x_1; \quad v_{-2} = x_2; \quad v_{-1} = x_3; \quad v_0 = x_4;$$
$$v_1 = v_{-1} * v_0;$$
$$v_2 = \tan(v_1);$$
$$v_3 = v_{-2} - v_2;$$
$$v_4 = v_{-3} * v_2;$$
$$v_5 = v_4 / v_3;$$
$$v_6 = v_5 * v_{-2};$$
$$y_1 = v_5; \quad y_2 = v_6;$$

$$\bar{v}_5 = \bar{y}_1; \quad \bar{v}_6 = \bar{y}_2;$$
$$\bar{v}_5 \mathrel{+}= \bar{v}_6 * v_{-2}; \quad \bar{v}_{-2} \mathrel{+}= \bar{v}_6 * v_5;$$
$$\bar{v}_4 \mathrel{+}= \bar{v}_5 / v_3; \quad \bar{v}_3 \mathrel{-}= \bar{v}_5 * v_5 / v_3;$$
$$\bar{v}_{-3} \mathrel{+}= \bar{v}_4 * v_2; \quad \bar{v}_2 \mathrel{+}= \bar{v}_4 * v_{-3};$$
$$\bar{v}_{-2} \mathrel{+}= \bar{v}_3; \; \bar{v}_2 \mathrel{-}= \bar{v}_3;$$
$$\bar{v}_1 \mathrel{+}= \bar{v}_2 / \cos^2(v_1);$$
$$\bar{v}_{-1} \mathrel{+}= \bar{v}_1 * v_0; \; \bar{v}_0 \mathrel{+}= \bar{v}_1 * v_{-1};$$
$$\bar{x}_4 = \bar{v}_0; \quad \bar{x}_3 = \bar{v}_{-1}; \quad \bar{x}_2 = \bar{v}_{-2}; \quad \bar{x}_1 = \bar{v}_{-3};$$

... and the real code generated by dcc 1.0

```
void b1_f(int& bmode_1, double* x, double* b1_x, double* y, double* b1_y)
//$ad indep x b1_x b1_y
//$ad dep y b1_x
 { double v[2];          double b1_v[2];
   double w1_0 = 0;      double b1_w1_0 = 0;      ...
   double w1_5 = 0;      double b1_w1_5 = 0;
   int save_cs_c = 0;    save_cs_c = cs_c;
   if (bmode_1==1) { // augmented forward section
    cs[cs_c] = 0;          cs_c = cs_c+1;
    fds[fds_c] = v[0];     fds_c = fds_c+1;    v[0] = tan(x[2]*x[3]);
    ...
    fds[fds_c] = y[1];     fds_c = fds_c+1;    y[1] = x[1]*y[0];
    while (cs_c>save_cs_c) {    // reverse section
       cs_c = cs_c-1;
       if (cs[cs_c]==0) {
          fds_c = fds_c-1;        y[1] = fds[fds_c];
          w1_0 = x[1];            w1_1 = y[0];     w1_2 = w1_0*w1_1;
          b1_w1_2 = b1_y[1];  b1_y[1] = 0; // adjoint assignment
          b1_w1_0 = w1_1*b1_w1_2;    b1_w1_1 = w1_0*b1_w1_2;
          b1_y[0] = b1_y[0]+b1_w1_1;   b1_x[1] = b1_x[1]+b1_w1_0;     ...
```

AD Tools

Fortran 77 (90): (mainly source transformation)

- Tapenade (INRIA, F)
- AD in the compiler (NAG, RWTH Aachen, Univ. Hertfordshire)
- …

AD Tools

Fortran 77 (90): (mainly source transformation)

- Tapenade (INRIA, F)
- AD in the compiler (NAG, RWTH Aachen, Univ. Hertfordshire)
- ...

C/C++: (mainly operator overloading)

- ADOL-C (Univ. Paderborn)
- CppAD (Univ. Washington, USA)
- ...

AD Tools

Fortran 77 (90): (mainly source transformation)

- Tapenade (INRIA, F)
- AD in the compiler (NAG, RWTH Aachen, Univ. Hertfordshire)
- ...

C/C++: (mainly operator overloading)

- ADOL-C (Univ. Paderborn)
- CppAD (Univ. Washington, USA)
- ...

Matlab: Adimat, MAD, ...
Modelica: ADModelica by Atya Elsheikh und Wolfgang Wiechert (!)

AD Tools

Fortran 77 (90): (mainly source transformation)

- Tapenade (INRIA, F)
- AD in the compiler (NAG, RWTH Aachen, Univ. Hertfordshire)
- ...

C/C++: (mainly operator overloading)

- ADOL-C (Univ. Paderborn)
- CppAD (Univ. Washington, USA)
- ...

Matlab: Adimat, MAD, ...
Modelica: ADModelica by Atya Elsheikh und Wolfgang Wiechert (!)

see www.autodiff.org, (Griewank, Walther 2008), (Naumann 2012) for more tools and literature

Conclusions: Basic AD

- Evaluation of derivatives with working accuracy (Griewank, Kulshreshtha, Walther 2012)
- Forward mode: $\text{OPS}(F'(x)\dot{x}) \leq c\,\text{OPS}(F), \quad c \in [2, 5/2]$

  Reverse mode: $\text{OPS}(\bar{y}^\top F'(x)) \leq c\,\text{OPS}(F), \quad c \in [3, 4]$

  $\text{MEM}(\bar{y}^\top F'(x)) \sim \text{OPS}(F),$

⟹ Gradients are cheap $\sim$ Function Costs!!

Conclusions: Basic AD

- Evaluation of derivatives with working accuracy (Griewank, Kulshreshtha, Walther 2012)
- Forward mode: $\mathrm{OPS}(F'(x)\dot{x}) \leq c\,\mathrm{OPS}(F), \quad c \in [2, 5/2]$

  Reverse mode: $\mathrm{OPS}(\bar{y}^\top F'(x)) \leq c\,\mathrm{OPS}(F), \quad c \in [3, 4]$

  $\mathrm{MEM}(\bar{y}^\top F'(x)) \sim \mathrm{OPS}(F),$

⟹ Gradients are cheap $\sim$ Function Costs!!

- Combination: $\mathrm{OPS}(\bar{y}^\top F''(x)\dot{x}) \leq c\,\mathrm{OPS}(F), \quad c \in [7, 10]$
- Cost of higher derivatives grows quadratically in the degree
- Nondifferentiability only on meager set
- Full Jacobians/Hessians often not needed or sparse

Conclusions: Basic AD

- Evaluation of derivatives with working accuracy (Griewank, Kulshreshtha, Walther 2012)
- Forward mode: $\mathrm{OPS}(F'(x)\dot{x}) \leq c\,\mathrm{OPS}(F), \quad c \in [2, 5/2]$
  Reverse mode: $\mathrm{OPS}(\bar{y}^\top F'(x)) \leq c\,\mathrm{OPS}(F), \quad c \in [3, 4]$
  $\mathrm{MEM}(\bar{y}^\top F'(x)) \sim \mathrm{OPS}(F),$

➞ Gradients are cheap $\sim$ Function Costs!!

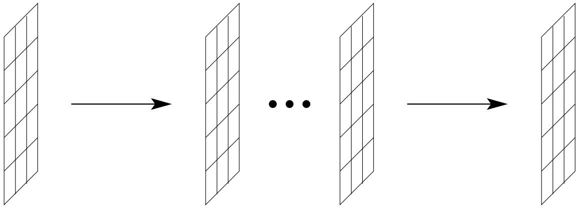
- Combination: $\mathrm{OPS}(\bar{y}^\top F''(x)\dot{x}) \leq c\,\mathrm{OPS}(F), \quad c \in [7, 10]$
- Cost of higher derivatives grows quadratically in the degree
- Nondifferentiability only on meager set
- Full Jacobians/Hessians often not needed or sparse

Questions: Structure Exploitation!!
Time-stepping, sparsity, fixed point iteration, ...

Automatic Differentiation by Overloading in C++

- **ADOL-C version 2.3**
- available at COIN-OR since May 2009
- interface to ColPack (Purdue University) and Ipopt (COIN-OR)
- recent developments
  - improved computation of sparsity pattern for Hessians
  - handling of MPI-parallel codes
  - handling of GPU-parallel codes
- future plans
  - generalized derivatives for nonsmooth functions
  - ...

Calculating Adjoints

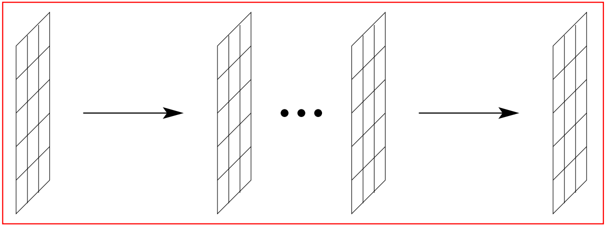


Integration of forward solution:

$$y_{i+1} = F_i(y_i, u_i), \qquad i = 1, \ldots, l$$

Integration of adjoint $\bar{y}_{i-1} = \bar{F}_i(\bar{y}_i, \bar{u}_i, y_i),\ i = l, \ldots, 1$?

Calculating Adjoints



Integration of forward solution:

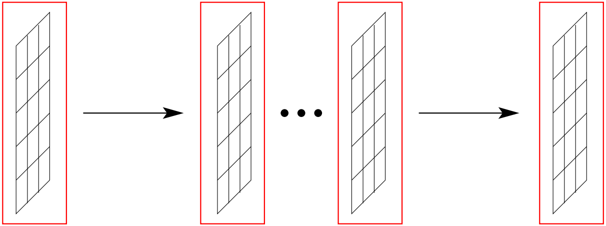
$$y_{i+1} = F_i(y_i, u_i), \qquad i = 1, \ldots, l$$

Integration of adjoint $\bar{y}_{i-1} = \bar{F}_i(\bar{y}_i, \bar{u}_i, y_i),\ i = l, \ldots, 1?$

"Black-Box"-approach, e.g. using AD

Memory requirement?? Computing time ??

Calculating Adjoints



Integration of forward solution:

$$y_{i+1} = F_i(y_i, u_i), \qquad i = 1, \ldots, l$$

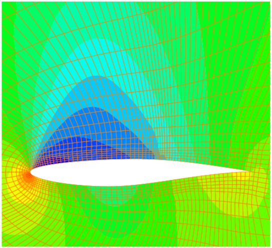
Integration of adjoint $\bar{y}_{i-1} = \bar{F}_i(\bar{y}_i, \bar{u}_i, y_i),\ i = l, \ldots, 1$?

Time Structure Exploitation

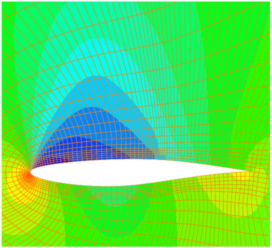
Memory requirement?? Computing time ?? Adjoint ??

Pseudo Time-dependent Problems

- Example:
  Shape Optimization
  in Aerodynamics
- Target: Minimize drag



Pseudo Time-dependent Problems



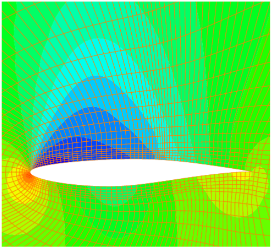
- Example:
  Shape Optimization
  in Aerodynamics
- Target: Minimize drag

Approaches:

- Exploitation of fixed point structure
  $\Rightarrow$ reverse accumulation of gradient (Christianson 1991)
  $\Rightarrow$ TIME(gradient)/TIME(target function) < 9
  (Gauger, Walther, Özkaya, Moldenhauer 2012)

Pseudo Time-dependent Problems

- Example:
  Shape Optimization
  in Aerodynamics
- Target: Minimize drag

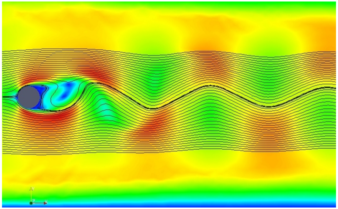


Approaches:

- Exploitation of fixed point structure
  $\Rightarrow$ reverse accumulation of gradient (Christianson 1991)
  $\Rightarrow$ TIME(gradient)/TIME(target function) < 9
  (Gauger, Walther, Özkaya, Moldenhauer 2012)
- One-Shot Optimization
  $\Rightarrow$ again adjoint of only one time step required
  N. Gauger, A. Griewank, E. Özkaya

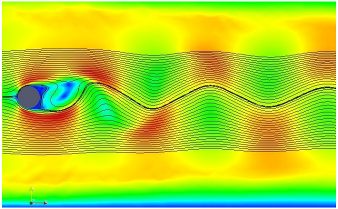
Real Time-dependent Problems

- Example:
  Transient flows
- Target: Minimize drag/turbulence



Real Time-dependent Problems

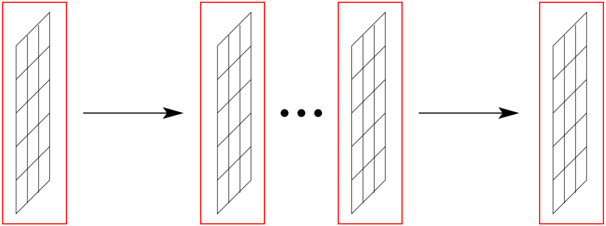
- Example:
  Transient flows
- Target: Minimize drag/turbulence



Approaches: Checkpointing in all variations, adjoint of one time step

- PDE-based optimization: Windowing
  Berggren, Meidner, Vexler, . . .
- Binomial Checkpointing
  Griewank, Walther, Sternberg, Stumm, Moin, . . .
- in general for AD: subroutine oriented checkpointing
  OpenAD, Tapenade

Calculating Adjoints II



Integration of forward solution:

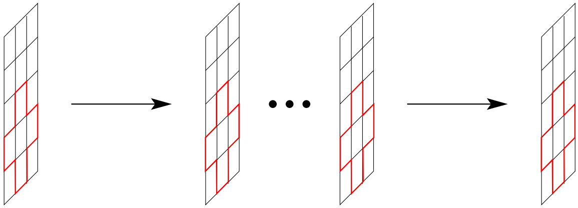
$$y_{i+1} = F_i(y_i, u_i), \qquad i = 1, \ldots, l$$

Integration of adjoint $\bar{y}_{i-1} = \bar{F}_i(\bar{y}_i, \bar{u}_i, y_i),\; i = l, \ldots, 1$?

Time Structure Exploitation

Memory requirement?? Computing time ?? Adjoint ??

Calculating Adjoints II



Integration of forward solution:

$$y_{i+1} = F_i(y_i, u_i), \qquad i = 1, \ldots, l$$

Integration of adjoint $\bar{y}_{i-1} = \bar{F}_i(\bar{y}_i, \bar{u}_i, y_i),\ i = l, \ldots, 1$?

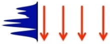
Time and Space Structure Exploitation

Memory requirement?? Computing time ?? Adjoint ??

Optimisation for Nanooptics

Cooperation with T. Meier, M. Reichelt, Dep. Physik, Uni Paderborn

Generic configuration:

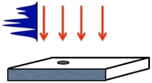


$\longleftarrow$ adaptable light puls $E(t)$

Optimisation for Nanooptics

Cooperation with T. Meier, M. Reichelt, Dep. Physik, Uni Paderborn

Generic configuration:



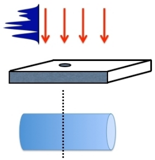
$\longleftarrow$ adaptable light puls $E(t)$

$\longleftarrow$ metal aperture with air hole

Optimisation for Nanooptics

Cooperation with T. Meier, M. Reichelt, Dep. Physik, Uni Paderborn

Generic configuration:



$\longleftarrow$ adaptable light puls $E(t)$

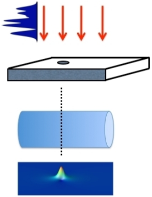
$\longleftarrow$ metal aperture with air hole

$\longleftarrow$ quantum wire

Optimisation for Nanooptics

Cooperation with T. Meier, M. Reichelt, Dep. Physik, Uni Paderborn

Generic configuration:



$\longleftarrow$ adaptable light puls $E(t)$

$\longleftarrow$ metal aperture with air hole

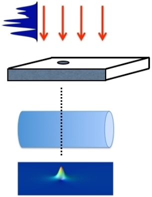
$\longleftarrow$ quantum wire

$\longleftarrow$ electron distribution?

Optimisation for Nanooptics

Cooperation with T. Meier, M. Reichelt, Dep. Physik, Uni Paderborn

Generic configuration:



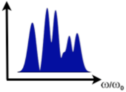
$\longleftarrow$ adaptable light puls $E(t)$

$\longleftarrow$ metal aperture with air hole

$\longleftarrow$ quantum wire

$\longleftarrow$ electron distribution?

Light puls:

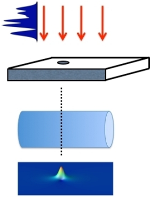


$$\text{with } E(t) = \sum A_i \exp\left(-\left(\frac{t-t_i}{\Delta t_i}\right)^2\right)\cos(\omega_i t + \phi_i)$$

Optimisation for Nanooptics

Cooperation with T. Meier, M. Reichelt, Dep. Physik, Uni Paderborn

Generic configuration:



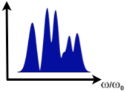
$\longleftarrow$ adaptable light puls $E(t)$

$\longleftarrow$ metal aperture with air hole

$\longleftarrow$ quantum wire

$\longleftarrow$ electron distribution?

Light puls:



$$\text{with } E(t) = \sum A_i \exp\left(-\left(\frac{t-t_i}{\Delta t_i}\right)^2\right)\cos(\omega_i t + \phi_i)$$

Parameter: A_i, ϕ_i $\quad\Rightarrow$ 60!

Nanooptics: Optimisation

So far: Genetic algorithms

Now: L-BFGS and efficient gradient computation

- AD coupled with hand-coded adjoints
- Checkpointing (160 000 time steps!!)

$\Rightarrow$ TIME(gradient)/TIME(target function) $<$ 7 despite of checkpointing!

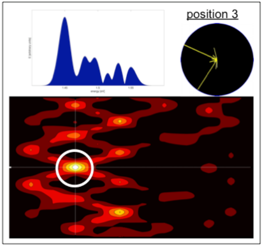
Nanooptics: Optimisation

So far: Genetic algorithms

Now: L-BFGS and efficient gradient computation

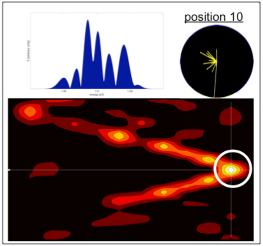
- AD coupled with hand-coded adjoints
- Checkpointing (160 000 time steps!!)

$\Rightarrow$ TIME(gradient)/TIME(target function) $<$ 7 despite of checkpointing!

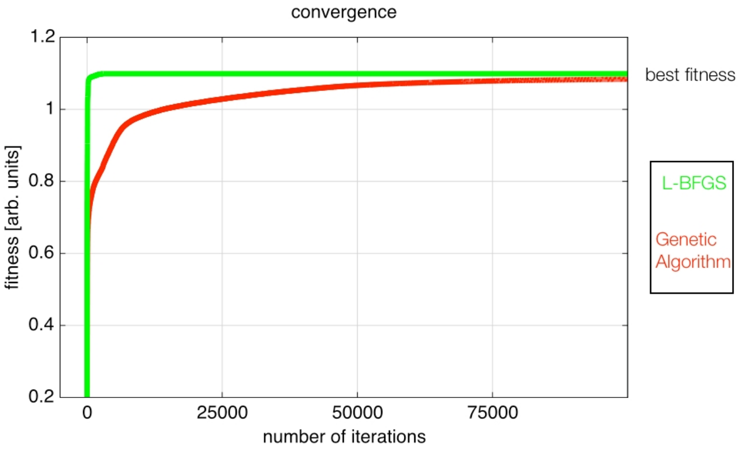


excite
•at **same** position
•at **same** time
•with **same** energy

optimize
•for **same** t_{opt}
•**different** positions



Nanooptics: Comparison



(Walther, Reichelt, Meier 2011)

Conclusions

- Basics of Algorithmic Differentiation
  - Efficient evaluation of derivatives with working accuracy
  - Discrete Analogons of sensitivity and adjoint equation
  - Theory for basic modes complete, advanced AD?

Conclusions

- Basics of Algorithmic Differentiation
  - Efficient evaluation of derivatives with working accuracy
  - Discrete Analogons of sensitivity and adjoint equation
  - Theory for basic modes complete, advanced AD?
- Structure exploitation indispensable

Conclusions

- Basics of Algorithmic Differentiation
  - Efficient evaluation of derivatives with working accuracy
  - Discrete Analogons of sensitivity and adjoint equation
  - Theory for basic modes complete, advanced AD?
- Structure exploitation indispensable
- Consistent adjoint information? Efficient implementation?

  Suitable combination of continuous and discrete approach!