Computing Derivatives of Programs

Algorithmic Differentiation by Example

Andreas Griewank* *Jan Riehme*

* DFG Research Center **Matheon**

Institute for Applied Mathematics
Humboldt Universität zu Berlin
{riehme,griewank}@math.hu-berlin.de

June 1, 2005
MATHEON Workshop – Optimization Software

Introduction

www.autodiff.org

AD - Tools for Fortran and C

- **ADOL-C**, **REVOLVE**: C, C++, Open Source
- **ADIFOR 2.0 / 3.0**: Fortran 77/90/95, Licensed, Closed Source
- **Tapenade**: Fortran 77/90/95, (some) C, free, Closed Source
- **TAF / TAC** (FastOpt GbR):Fortran 77/90/95, (some) C, commercial, maybe free for educational
- **NAGWare Fortran 95**, NAG Ltd., Oxford, UK: AD-enabled version in beta status, not available for the public
- **OpenAd**: Fortran 77/90/95, (some) C, Open Source

Other tools

for Fortran,C, C++ for Matlab for ADA for ...

Introduction – Optimization Programming Scenario

- Nonlinear Optimization needs derivatives, for example:
  - Gradients, Jacobians, Hessians
  - Truncated Newton needs Jacobian-Vector-Products and Vector-Jacobian-Products or Hessian-Vector-Products
- Readily available in **GAMS**/**AMPL**. But how about the real world?
- NLP - solver usually ask for
  - Subroutine to compute function value F
  - Subroutine to evaluate constraints
  - Sparsity patterns of Values, Jacobian, Hessian
- Often there is an interface to provide derivatives
  - Subroutines for gradients, Jacobians-Vector-Products, Hessians-Vector-Products

  Default: Differencing

Use Automatic Differentiation to obtain derivatives

- write a wrapper to plug generated derivative into NLP - interface (**NEOS**)

Main Properties of Automatic Differentiation:

No Truncation Errors !!!!

Chain Rule applied to Numbers

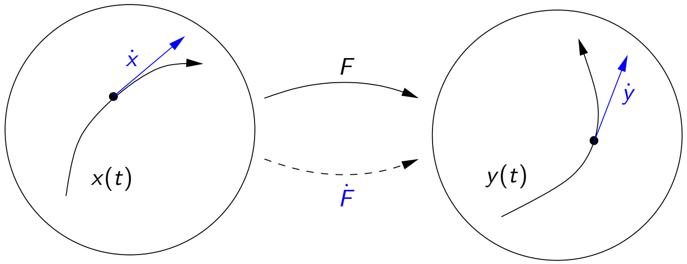
Applicability to "Arbitrary Programs".

A priori bounded and/or adjustable costs:

- Total Operations Count
- Maximal Memory Requirement
- Total Memory Traffic

always relative to original function.

Geometric Interpretation – Forward Mode



F ... Original program F
$\dot{x}$ ... Tangent direction for input x
$\dot{F}$ ... Tangent version of F (generated by Forward Mode AD)
$\dot{y}$ ... Tangent of output y:

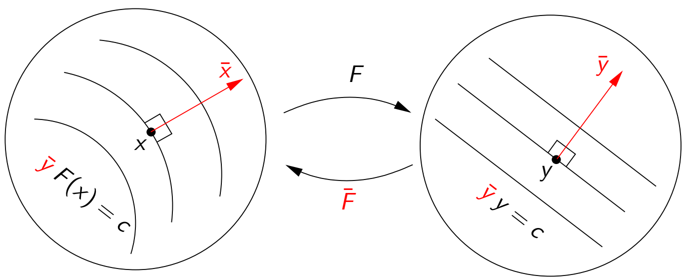
$$\mathbf{\dot{y} = \dot{F}(x, \dot{x}) = F'(x) \cdot \dot{x}}$$

Practical Execution of Forward and Reverse Differentiation

Sourcecode

$y = F(x) \in \mathbb{R}^m$

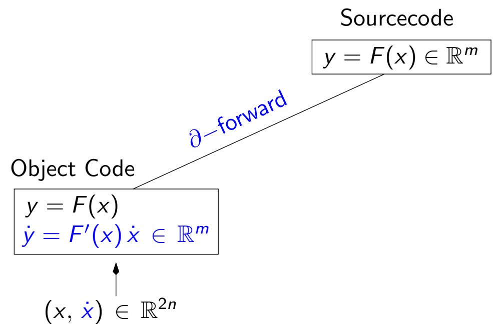
Geometric Interpretation – Reverse Mode



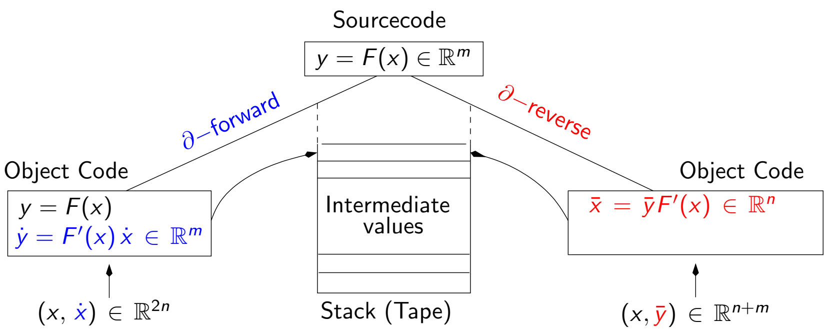
F ... Original program F
$\bar{y}$ ... Adjoint direction for output y
$\bar{F}$ ... Adjoint version of F (generated by Reverse Mode AD)
$\bar{x}$ ... Adjoint of input x:

$$\mathbf{\bar{x} = \bar{F}(x, \bar{y}) = \bar{y} \cdot F'(x)}$$

Practical Execution of Forward and Reverse Differentiation



Practical Execution of Forward and Reverse Differentiation



Derivatives in Optimization Scenario

$$F(x) = \begin{bmatrix} f(x) \\ c(x) \end{bmatrix} \quad : \quad \mathbb{R}^n \mapsto \mathbb{R}^m$$

with Lagrangian function

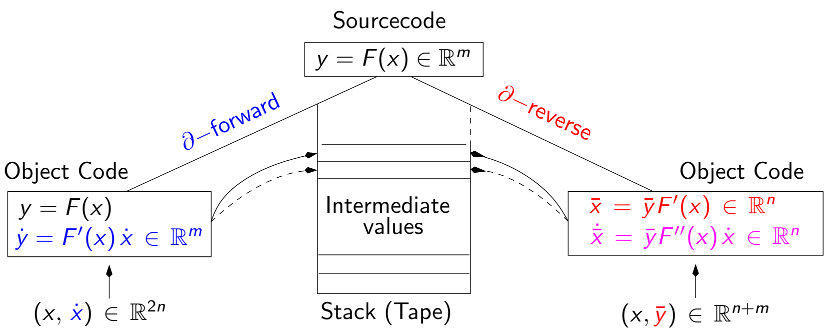
$$L(x) = \lambda^T F(x) \quad \text{for fixed} \quad \lambda \in \mathbb{R}^m$$

Product	Derivative	Cost-Factor
Jac_mat	$F'(x)S \in \mathbb{R}^{m\times p}$	$3 * p$
Jacobian	$F'(x) \in \mathbb{R}^{m\times n}$	$\min\{\#[F'], \#[F'(x)^T]\}$
mat_Jac	$WF'(x) \in \mathbb{R}^{q\times n}$	$5 * q$
gradient	$\nabla L(x) \in \mathbb{R}^n$	5 $\Longleftarrow$
Hess_mat	$\nabla^2 L(x)S \in \mathbb{R}^{n\times p}$	$5 * p$ $\Longleftarrow$
Hessian	$\nabla^2 L(x) \in \mathbb{R}^{n\times n}$	$5 * \#[\nabla^2 L(x)]$

where $s \in \mathbb{R}^n$, $S \in \mathbb{R}^{n\times p}$, $W \in \mathbb{R}^{q\times m}$, and

$$\#[A] \equiv \max_i \{nonzeros\,(e_i^T A)\}$$

Practical Execution of Forward and Reverse Differentiation



Baby Example

$$y = [\sin(x_1/x_2) + x_1/x_2 - \exp(x_2)] * [x_1/x_2 - \exp(x_2)]$$

Evaluation of Baby Example with

$$n = \dim(x) = 2 \quad \text{and} \quad m = \dim(y) = 1$$

v_{-1}	=	x_1	=	1.5000		
v_0	=	x_2	=	0.5000		
v_1	=	v_{-1}/v_0	=	1.5000/0.5000	=	3.0000
v_2	=	$\sin(v_1)$	=	sin(3.0000)	=	0.1411
v_3	=	$\exp(v_0)$	=	exp(0.5000)	=	1.6487
v_4	=	$v_1 - v_3$	=	3.0000 − 1.6487	=	1.3513
v_5	=	$v_2 + v_4$	=	0.1411 + 1.3513	=	1.4924
v_6	=	$v_5 * v_4$	=	1.4924 ∗ 1.3513	=	2.0167
y	=	v_6	=	2.0167		

Forward Derived Evaluation Trace of Baby Example

$v_{-1} = x_1$	$= 1.5000$	
$\dot{v}_{-1} = \dot{x}_1$	$= 1.0000$	
$v_0 = x_2$	$= 0.5000$	
$\dot{v}_0 = \dot{x}_2$	$= 0.0000$	
$v_1 = v_{-1}/v_0$	$= 1.5000/0.5000$	$= 3.0000$
$\dot{v}_1 = (\dot{v}_{-1} - v_1 * \dot{v}_0)/v_0$	$= 1.0000/0.5000$	$= 2.0000$
$v_2 = \sin(v_1)$	$= \sin(3.0000)$	$= 0.1411$
$\dot{v}_2 = \cos(v_1) * \dot{v}_1$	$= -0.9900 * 2.0000$	$= -1.9800$
$v_3 = \exp(v_0)$	$= \exp(0.5000)$	$= 1.6487$
$\dot{v}_3 = v_3 * \dot{v}_o$	$= 1.6487 * 0.0000$	$= 0.0000$
$v_4 = v_1 - v_3$	$= 3.0000 - 1.6487$	$= 1.3513$
$\dot{v}_4 = \dot{v}_1 - \dot{v}_3$	$= 2.0000 - 0.0000$	$= 2.0000$
$v_5 = v_2 + v_4$	$= 0.1411 + 1.3513$	$= 1.4924$
$\dot{v}_5 = \dot{v}_2 + \dot{v}_4$	$= -1.9800 + 2.0000$	$= 0.0200$
$v_6 = v_5 * v_4$	$= 1.4924 * 1.3513$	$= 2.0167$
$\dot{v}_6 = \dot{v}_5 * v_4 + v_5 * \dot{v}_4$	$= 0.0200 * 1.3513 + 1.4924 * 2.0000$	$= 3.0118$
$y = v_6$	$= 2.0100$	
$\dot{y} = \dot{v}_6$	$= 3.0110$	

Fortran Example – Tapenade, Forward mode

- Automatic Differentiation of FORTRAN77, F90/F95 partial
- Automatic Differentiation by **Source Transformation**
  - Takes `Fortran` source code
  - Converting to internal representation
  - Augment internal representation with AD-instructions
  - Generate target source code
- Freely available from
  `http://www-sop.inria.fr/tropics/tapenade.html`

```
                   ↓     ↓     ↑
SUBROUTINE baby(  x1,   x2,    y)
```

⇓ Tapenade Forward

```
SUBROUTINE baby_d(  x1,  x1d,  x2,  x2d,  y,  yd)
                    ↑    ↑     ↑    ↑     ↓   ↓
```

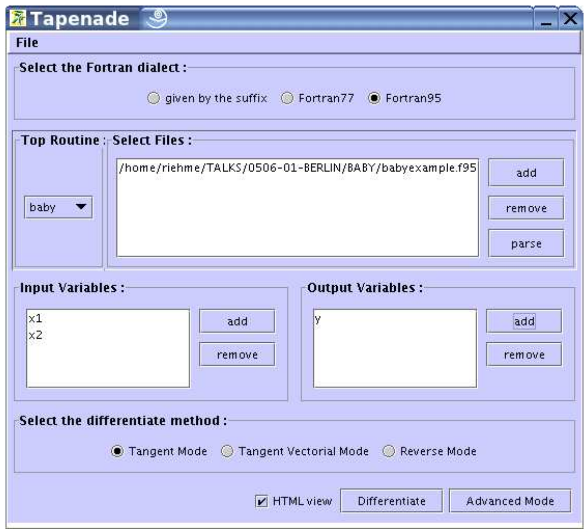
- Driver program needed!

Hurry

Reverse Derived Trace of Baby Example

$v_{-1} = x_1 = 1.5000$
$v_0 = x_2 = 0.5000$
$v_1 = v_{-1}/v_0 = 1.5000/0.5000 = 3.0000$
$v_2 = \sin(v_1) = \sin(3.0000) = 0.1411$
$v_3 = \exp(v_0) = \exp(0.5000) = 1.6487$
$v_4 = v_1 - v_3 = 3.0000 - 1.6487 = 1.3513$
$v_5 = v_2 + v_4 = 0.1411 + 1.3513 = 1.4924$
$v_6 = v_5 * v_4 = 1.4924 * 1.3513 = 2.0167$
$y = v_6 = 2.0167$
$\bar{v}_6 = \bar{y} = 1.0000$
$\bar{v}_5 = \bar{v}_6 * v_4 = 1.0000 * 1.3513 = 1.3513$
$\bar{v}_4 = \bar{v}_6 * v_5 = 1.0000 * 1.4924 = 1.4924$
$\bar{v}_4 = \bar{v}_4 + \bar{v}_5 = 1.4924 + 1.3513 = 2.8437$
$\bar{v}_2 = \bar{v}_5 = 1.3513$
$\bar{v}_3 = -\bar{v}_4 = -2.8437$
$\bar{v}_1 = \bar{v}_4 = 2.8437$
$\bar{v}_0 = \bar{v}_3 * v_3 = -2.8437 * 1.6487 = -4.6884$
$\bar{v}_1 = \bar{v}_1 + \bar{v}_2 * \cos(v_1) = 2.8437 + 1.3513 * (-0.9900) = 1.5059$
$\bar{v}_0 = \bar{v}_0 - \bar{v}_1 * v_1/v_0 = -4.6884 - 1.5059 * 3.000/0.5000 = -13.7239$
$\bar{v}_{-1} = \bar{v}_1/v_0 = 1.5059/0.5000 = 3.0118$
$\bar{x}_2 = \bar{v}_0 = -13.7239$
$\bar{x}_1 = \bar{v}_{-1} = 3.0118$

Fortran Example – Tapenade



Fortran Example – Tapenade, Driver, Forward Mode

```
PROGRAM BABYEXAMPLE_D
  IMPLICIT NONE
  REAL ::  x1, x2, y
  REAL x1d, x2d, yd
  x1 = 1.5
  x2 = 0.5
  x1d = 1.0
  x2d = 0.0
   call baby_d( x1, x1d, x2, x2d, y, yd )
  print *,y, yd
END PROGRAM BABYEXAMPLE_D
```

Output:

```
  2.016647    3.011843
```


Fortran Example – Tapenade, Driver, Forward Mode

```
PROGRAM BABYEXAMPLE_B
  IMPLICIT NONE
  REAL ::  x1, x2, y
  REAL x1b, x2b, yb
  x1 = 1.5
  x2 = 0.5
  yb = 1.0
   call baby_b( x1, x1b, x2, x2b, y, yb )
  print *,x1, x1b
  print *,x2, x2b
END PROGRAM BABYEXAMPLE_B
```

Output:

```
  1.500000      3.011843
 0.5000000    -13.72396
```


Fortran Example – Tapenade, Reverse mode

```
                     ↓     ↓      ↑
SUBROUTINE baby(  x1,   x2,     y)
```

⇓ Tapenade Reverse

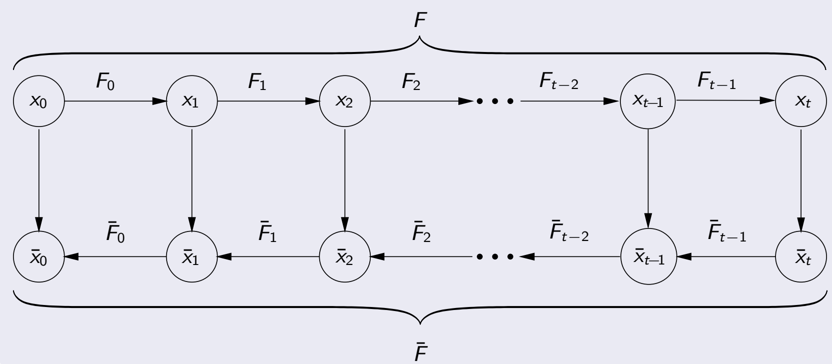
```
SUBROUTINE baby_b(  x1,  x1b,  x2,  x2b,   y,   yb)
                    ↑     ↓     ↑     ↓    X     ↑
```

- Driver program needed!
- !! Function value y not computed in adjoint mode !! (Tapenade specific)

Hurry

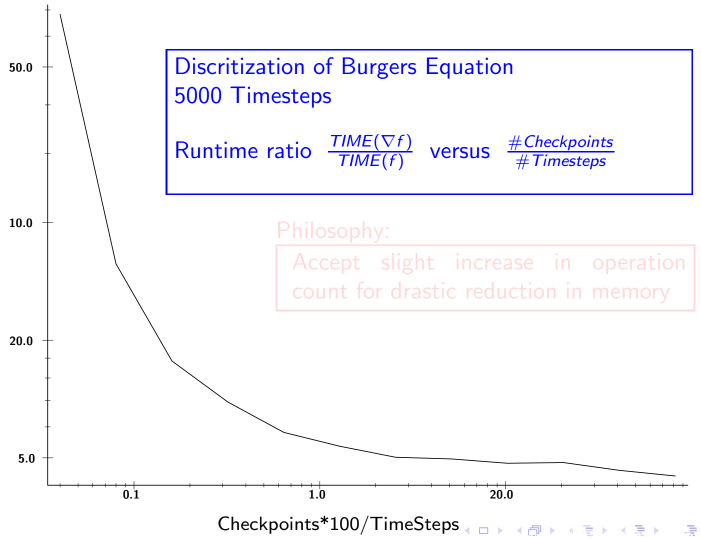
Checkpointing – Evolutions, time dependent problem

Adjoint calculation



Hurry

Checkpointing – Runtime-Memory-Tradeoff



Discritization of Burgers Equation
5000 Timesteps

Runtime ratio $\frac{TIME(\nabla f)}{TIME(f)}$ versus $\frac{\#Checkpoints}{\#Timesteps}$

Philosophy:

Accept slight increase in operation count for drastic reduction in memory

Checkpointing – Implementation

ADOL-C – C++-tool, based on Overloading

- Full language support (work is done at runtime)
- records function evaluation on a several `TAPE`s
- Activate Sourcecode – replace `double` by active datatype `adouble`
- special syntax for tape creation, initialization of independents, etc.
- Freely available from `http://www.math.tu-dresden.de/~adol-c/`

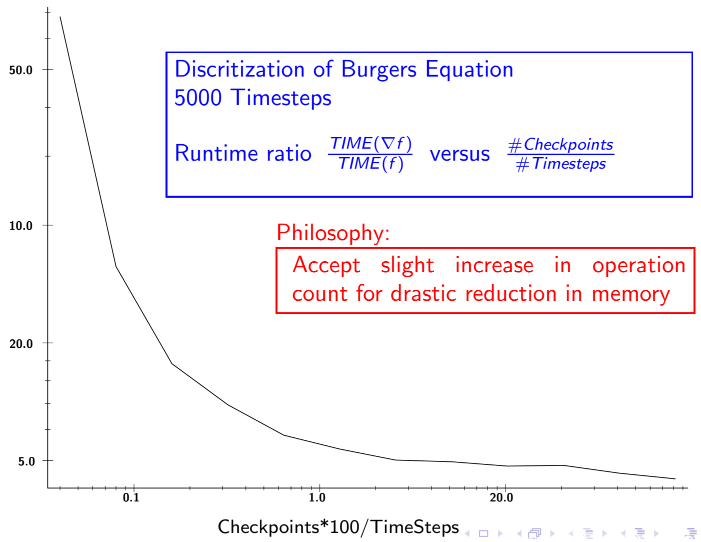
Question: Where to place checkpoints optimally?

REVOLVE – Program-Reversals for (pseudo)- Time-stepping Procedures

- Freely available from `http://www.math.tu-dresden.de/wir/project/revolve/`
- **revolve** tells what to do next inside main loop:
  - Store / restore state
  - Advance k states
  - Adjoint state

  } provide user routines for these steps!

Checkpointing – Runtime-Memory-Tradeoff



Discritization of Burgers Equation
5000 Timesteps

Runtime ratio $\frac{TIME(\nabla f)}{TIME(f)}$ versus $\frac{\#Checkpoints}{\#Timesteps}$

Philosophy:

Accept slight increase in operation count for drastic reduction in memory

Checkpointing – Implementation

ADOL-C – C++-tool, based on Overloading

- Full language support (work is done at runtime)
- records function evaluation on a several `TAPE`s
- Activate Sourcecode – replace `double` by active datatype `adouble`
- special syntax for tape creation, initialization of independents, etc.
- Freely available from `http://www.math.tu-dresden.de/~adol-c/`

Question: Where to place checkpoints optimally?

REVOLVE – Program-Reversals for (pseudo)- Time-stepping Procedures

- Freely available from `http://www.math.tu-dresden.de/wir/project/revolve/`
- **revolve** tells what to do next inside main loop:
  - Store / restore state
  - Advance k states
  - Adjoint state

  } provide user routines for these steps!

Software for Automatic Differentiation

www.autodiff.org

AD - Tools for Fortran and C

- **ADOL-C**, **REVOLVE**: C, C++, Open Source
- **ADIFOR 2.0 / 3.0**: Fortran 77/90/95, Licensed, Closed Source
- **Tapenade**: Fortran 77/90/95, (some) C, free, Closed Source
- **TAF / TAC** (FastOpt GbR):Fortran 77/90/95, (some) C, commercial, maybe free for educational
- **NAGWare Fortran 95**, NAG Ltd., Oxford, UK: AD-enabled version in beta status, not available for the public
- **OpenAd**: Fortran 77/90/95, (some) C, Open Source

Other tools

for Fortran,C, C++ for Matlab for ADA for ...

Thank you!

Bibliography

- A. Griewank: *Evaluating Derivatives: principles and techniques of algorithmic differentiation*. SIAM, Frontiers in Applied Mathematics, Number 19, 2000.
- A. Griewank and A. Walther: *Revolve: An Implementation of Checkpointing for the Reverse or Adjoint Mode of Computational Differentiation*. ACM Trans. Math. Software 26, 2000, 19–45.
- A. Griewank and A. Walther: *Applying the Checkpointing Routine treeverse to Discretizations of Burgers' Equation*. Lect. Notes Comput. Sci.and Engin. 8: H.-J. Bungartz, F. Durst, C. Zenger, (eds.), High Performance Scientific and Engineering Computing, Springer Berlin Heidelberg, 1999.