Efficient adjoints and Hessians for optimisation and uncertainty propagation

Thomas Kaminski, Ralf Giering, and Michael Voßbeck

*Fast*Opt

http://www.FastOpt.com

GAMM, Dresden, March 2004

Outline

- FastOpt
- Automatic Differentiation
- TAF
- Applications
- TAC++
- Conclusions

A few facts about *Fast*Opt

- Founded in February 2000 at Hamburg
- By Ralf Giering and Thomas Kaminski
- One more colleague as of July 2003
- Two kinds of business:
  - Develop and provide tools (TAF) for Automatic Differentiation (AD)
  - Carry out Consulting Projects with focus on AD, Inverse Modelling, Optimisation...
- 15 years of Experience in AD

Automatic Differentiation

for details see Griewank (2000) + Talk by Slawig

- Optimisation requires gradient information for existing function code
- Uncertainty propagation requires 2nd derivatives (Hessian) information
- Automatic Differentiation (AD) applies chain rule to function code
- Multiple matrix product can be evaluation in forward or reverse
- Comp. cost of forward mode proportional to number of independents
- Comp. cost of reverse mode proportional to number of dependents
  -> Well suited for optimisation
- AD-derivatives are exact up to rounding error
  (unlike finite difference approximation)
- AD-tools allow to update derivative-based modelling/optimisation systems automatically to each new version of function code
  -> Development cycles are becoming shorter

TAF
Transformation of Algorithms in Fortran

- Source-to-source translator for Fortran-77/95
- Commercial successor of TAMC
- Forward and reverse mode (1st derivatives):
  Tangent linear and adjoint models
- Scalar and vector mode
- Efficient Hessian (2nd derivative) code
  by applying TAF twice (e.g. forward over reverse)
- Command line program with many options
- TAF-Directives are Fortran comments
- Extensive and complex code analyses
  (similar to optimising compilers)
- Generated code is structured and well readable

TAF
More features

- Generation of flexible store/read scheme for required values triggered by TAF init and store directives
- Generation of simple checkpointing scheme (Griewank, 1992) triggered by combination of TAF init and store directives
- Generation of efficient adjoint (Christianson, 1996, 1998) for converging iterations triggered by TAF loop directive
- TAF flow directives for black-box routines,
  or to include user provided derivative code
  (exploit self-adjointness, MPI wrappers, etc...)
- Automatic Sparsity Detection
- Basic support for MPI and OpenMP

TAF
Ongoing Development

- TAF is constantly being adapted to new language standards, next targets:
  - Fortran 2000
  - OpenMP 2
- TAF code analyses are constantly being extended
- TAF algorithms are constantly being improved and adapted to the needs of the users
- FastOpt is giving support for TAF users
- FastOpt is offering consulting for AD and further projects

some larger TAF Derivatives

Model (Who)	Lines	Lang	TLM	ADM	Ckp	HES
NASA/DAO (w. Todling & Lin)	87'000	F90	2.7	6.8	2 lev	-
MOM3 (Galanti & Tziperman)	50'000	F77	Yes	4.6	2 lev	-
MITGCM (ECCO Consortium)	100'000	F77	1.8	5.5	3 lev	11.0/1
BETHY (w. Knorr, Rayner, Scholze)	5'400	F90	1.5	3.6	2 lev	12.5/5
Nav.-Stokes-Solver (Hinze, Slawig)	450	F77	-	2.0	steady	-
NSC2KE (w. Slawig)	2'500	F77	2.4	3.4	steady	9.8/1
HB_AIRFOIL (Thomas & Hall)	8'000	F90	-	3.0		-

- Lines: total number of Fortran lines without comments
- Numbers for TLM and ADM give CPU time for (function + gradient) relative to forward model
- HES format: CPU time for Hessian * n vectors rel. t. forw. model/ n
- 2 (3) level checkpointing costs 1 (2) additional model run(s)

Performance ratios
TAF vs hand coded adjoints

Code	Hand	TAF/TAMC	relativ
EPT (MINPACK-2)	1.5	1.9	-27%
GL1 (MINPACK-2)	1.5	1.7	-13%
GL2 (MINPACK-2)	2.1	1.3	38%
MSA (MINPACK-2)	1.8	1.6	9%
PJB (MINPACK-2)	1.8	2.2	-26%
SSC (MINPACK-2)	1.2	1.1	4%
Nav.-Stokes (Hinze and Slawig)	1.9	2.0	-5%
EULOSOLDO (Cusdin and Müller)	2.4	2.7	-12%
3D NS CFD-code (Cusdin and Müller	1.2	1.2	-4%

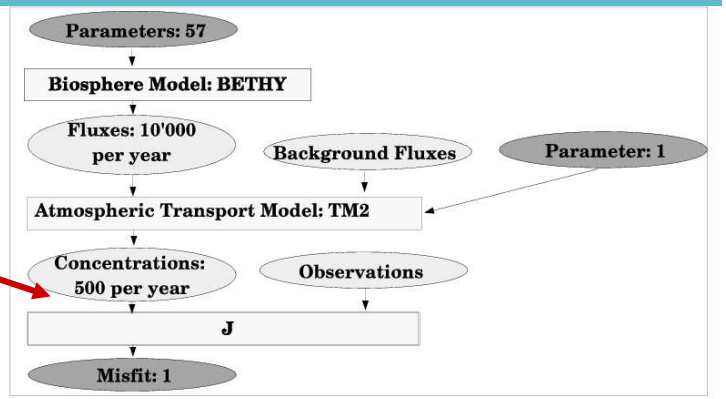
=> Performance of TAF generated adjoints is comparable to that of hand written adjoints

Terrestrial Vegetation Modelling (http://CCDAS.org)

BETHY coupled to TM2

Collaboration with Knorr, Scholze, Widmann (Max-Planck, Germany) and Rayner (CSIRO, Australia)

- Terrestrial vegetation model (BETHY, Knorr 2000) simulates CO2 fluxes to atmosphere
- Model has 58 unkown parameters m ("design variables")
- Atmospheric CO2 is observed at global networks
- Use atmospheric CO2 to constrain parameters (calibration)
- Put mitfit to observations and other information in scalar objective function J
- Use gradient algorithm to minimise objective function- > Need adjoint
- Use error in measurements (and model) to infer uncertainties in parameters (covariance C_m) - > Need Hessian
- Use parameter uncertainties to derive uncertainties for predictions - > Need Jacobian

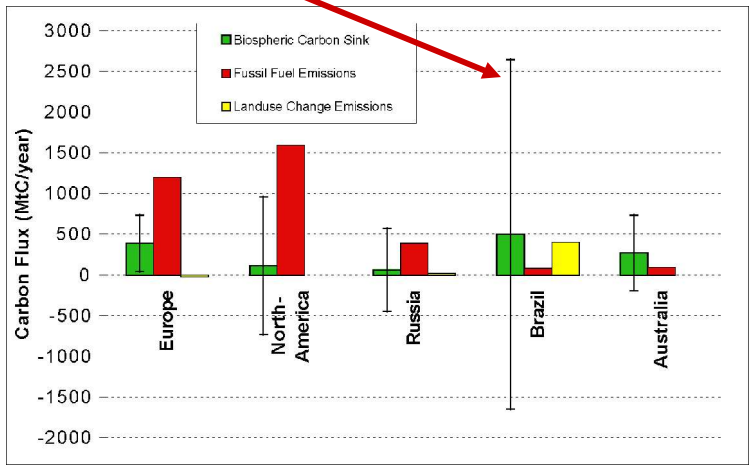


$$C_m \approx \left\{ \frac{\partial^2 J(m_{opt})}{\partial m_{i,i}^2} \right\}^{-1}$$

$$C_y = \left(\frac{\partial y_i(m_{opt})}{\partial m_j} \right) C_m \left(\frac{\partial y_i(m_{opt})}{\partial m_j} \right)^T$$

BETHY:
5400 lines of Fortran 90 code, e.g. Modules, dynamic memory allocation, derived types
intrinsics like reshape, matmul, transpose

AD of BETHY with TAF by FastOpt:
- Generated adjoint, Jacobian, and Hessian, and sparsity detection code
- Inserting of 38 TAF directives for storing, support of code analysis



- First system to optimise arbitrary number of vegetation parameters and with rigorous handling of uncertainty
- Model is continuously developed further, TAF is integral part of modelling system, updates derivative code

MIT global ocean model
http://mitgcm.org

ECCO state estimation: problem size

- Dimensionality:
  - grid @ 1° × 1° resolution: $n_x \cdot n_y \cdot n_z = 360 \cdot 160 \cdot 23$ — $1,324,800$
  - model state: 17 3D + 2 2D fields — $\sim 2 \cdot 10^7$
  - timesteps: 10 years @ 1-hour time step — $87,600$
  - control vector — $\sim 1 \cdot 10^8$
    - initial temperature (T), salinity (S)
    - time-dependent surface forcing (every 2 days)
  - cost function: observational elements: — $\sim 1 \cdot 10^8$
- Computational size:
  - 60 processors (15 nodes) @ 512MB per proc.
  - I/O: 10 GB input, 35GB output
  - time: 59 hours per iteration @ 60 processors
- What we would ideally want:
  - 1/10° × 1/10° resol., 1000 years, full model error covariance ...

Slide: Courtesy Patrick Heimbach, MIT

- **Fast and reliable gradient (10^8 components)**
- **Scientific applications infeasible without efficient adjoint code**
- **After each update of MITgcm, the derivative code is updated by TAF (automated process)**

TAC++
Transformation of Algorithms in C++

- Source-to-source translator for C/C(++)
- Same approach as TAF
- Completed basic feasibility study (Vossbeck et al., 2004)
- Generated adjoint of Roe solver
  - from EULSOLDO by Müller (1991)
  - 129 statements in C code (by f2c from Fortran-77)
  - C code contains simple constructs
  - Performance comparable to TAF generated code
- Functionality will be extended "model by model“““"

Conclusions

- TAF generates efficient tangent linear, adjoint, and Hessian code
- TAF is an essential component in a number of modeling systems
- First feasibility study for TAC++ completed

More info:
http://www.FastOpt.com