AD and its Applications

- **Automatic Differentiation (AD)** is a set of techniques based on the mechanical application of the chain rule to obtain derivatives of a function given as a computer program.

- AD is used in the following areas:
  - Numerical Methods
  - Sensitivity Analysis
  - Design Optimization
  - Data Assimilation & Inverse Problems

[www.autodiff.org](http://www.autodiff.org)
Applications of AD in this Talk

- **Numerical Methods**
  - nonlinear, differential / algebraic equation solving
  - bifurcation analysis
  - nonlinear programming

- **Sensitivity Analysis**
  - first and second order sensitivities

- **Design Optimization**
  - ...
  - under uncertainty

- **Inverse Problems**
  - state estimation (data assimilation)
  - parameter and function estimation
  - model-based optimal experimental design

- **Optimal Control**
  - trajectory optimization
  - dynamic real-time optimization

Topics listed at [www.autodiff.org](http://www.autodiff.org)
Higher-Order Derivatives and AD

- AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations such as additions or elementary functions such as \( \exp() \). By applying the chain rule of derivative calculus repeatedly to these operations, derivatives of arbitrary order can be computed automatically, and accurate to working precision.

- Are higher-order derivatives just a nice-to-have in applications?
  - Nonlinear Programming
  - Optimal Control
  - Dynamic Real-time Optimization
  - Inverse Problems – Model-based Experimental Design
  - Design Optimization under Uncertainty
AD Tools – for the User

- AD tools implement the semantic transformation that systematically applies the chain rule of differential calculus to source code written in various programming languages.
- 29 AD tools listed on www.autodiff.org

<table>
<thead>
<tr>
<th>derivative order</th>
<th># tools</th>
<th># citations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18</td>
<td>296</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>17</td>
</tr>
<tr>
<td>n</td>
<td>5</td>
<td>45</td>
</tr>
</tbody>
</table>

- Higher order derivatives from AD have not reached the level of attention (at least in applications) they deserve!
- Are current tools only supporting expert users?
- Is there sufficient awareness of higher-order AD in applications?
Agenda

- **Introduction**
  - AD for Higher Order Derivatives and their Applications
  - AD Tools for the User

- **Higher-Order Derivatives in Applications**
  - Numerical Methods – Nonlinear Programming
  - Optimal Control
  - Dynamic Real-time Optimization
  - Inverse Problems – Model-Based Experimental Design
  - Design Optimization under Uncertainty

- **Software Tools**

- **Summary and Conclusions**
Agenda

- Introduction
  - AD for Higher Order Derivatives and their Applications
  - AD Tools for the User
- Higher-Order Derivatives in Applications
  - Numerical Methods – Nonlinear Programming
  - Optimal Control
  - Dynamic Real-time Optimization
  - Inverse Problems – Model-Based Experimental Design
  - Design Optimization under Uncertainty
- Software Tools
- Summary and Conclusions
Numerical Optimization Methods

- Why optimize?
- Engineering design means to search a large space of decision variables for an appropriate solution.
- Optimization supports this search effectively.

- Applications often result in Nonlinear Programming problems (NLP):
  \[
  \min_x \phi(x) \\
  \text{s.t. } c_i(x) \geq 0, \ i \in E \cup I
  \]

- Solution algorithms are based on first order necessary (Karush-Kuhn-Tucker) conditions.
Necessary Conditions of Optimality

- Lagrange function

\[ L(x, \lambda) = f(x) - \sum_{i \in A(x)} \lambda_i c_i(x) \]

- first-order necessary (KKT) conditions

\[ \nabla_x L(x^*, \lambda^*) = 0 \]
\[ c_i(x^*) = 0 , \forall i \in E \]
\[ c_i(x^*) \geq 0 , \forall i \in I \]
\[ \lambda^*_i \geq 0 , \forall i \in I \]
\[ \lambda^*_i c_i(x^*) = 0 , \forall i \in E \cup I \]
Hessian of Lagrange Function?

- gradient based evaluation of KKT conditions requires Hessian of Lagrange function
- first-order BFGS updates of approximate Hessian vs. exact Hessian
  - improved robustness and faster convergence with exact Hessian and suitable NLP solvers
  - warm start facilitated by exact Hessian in repetitive optimization
- check of sufficient conditions of optimality requires projected Hessian of Lagrange function
  - NLP algorithms evaluate symmetrically projected Hessian
  - can be and should be exploited by AD algorithms
Agenda

- **Introduction**
  - AD for Higher Order Derivatives and their Applications
  - AD Tools for the User

- **Higher-Order Derivatives in Applications**
  - Numerical Methods – Nonlinear Programming
  - Optimal Control
  - Dynamic Real-time Optimization
  - Inverse Problems – Model-Based Experimental Design
  - Design Optimization under Uncertainty

- **Software Tools**

- **Summary and Conclusions**
A Simple Optimal Control Problem

\[ \min_{a, t_f} t_f \]
\[ \text{s.t. } f(\dot{x}, \dot{v}, x, v, a) = 0 \]
\[ v(t) \leq v_{\max} \]
\[ x(0), v(0) = 0 \]
\[ v(t_f) = 0, x(t_f) \geq x_{t_f} \]
Optimal Control – Optimization with ODEs

\[
\begin{align*}
\min_{u(t), p_0} & \quad \Phi(x(t_f)) \\
\text{s. t.} & \quad \dot{x}(t) = f(x(t), u(t)), \\
& \quad x(t_0) = x_0(p_0) \in \mathbb{R}^{n_x}, \\
& \quad g(x(t), u(t)) \leq 0 \in \mathbb{R}^{n_g}, \\
& \quad h(x(t_f)) \leq 0 \in \mathbb{R}^{n_h}, \\
& \quad t \in [t_0, t_f],
\end{align*}
\]

- Mayer-type optimal control problem
- fixed final time without loss of generality
Semi-Discretization by Control Vector Parameterization

- discretization of the continuous control $u(t)$ by projection into finite-dimensional function space (e.g. low-order B-splines)

$$u_i(t, p) := \sum_{j=1}^{N_i} p_{ij} \phi_{ij}(t), \quad p_{ij} \in \mathbb{R}, \quad i = 1, \ldots, n_u.$$  

- finite dimensional decision vector $p$

$$p = (p_1, \ldots, p_{n_p})^T \in \mathbb{R}^{n_p}$$

example: piecewise constant approximation of the control
Semi-Discretization Results in NLP

\[
\begin{align*}
\min_p & \quad \Phi(x(t_f)) \\
\ & \quad x(t) = F(t, p) \\
\ & \quad g(x(t_i), u(t_i, p)) \leq 0, \quad i = 1, \ldots, N \\
\ & \quad h(x(t_f)) \leq 0, \\
\ & \quad t \in [t_0, t_f],
\end{align*}
\]

Lagrangian functional

\[
\mathcal{L}(p, \mu, \nu) = \Phi(x(t_f)) + \left( \sum_{k=1}^{N} \mu_k^T g(x(t_k), u(t_k)) \right) + \nu^T h(x(t_f))
\]

How to efficiently determine the Hessian of the Lagrangian?
Composite Adjoint Approach for Hessian Evaluation

(Hannemann & Marquardt, 2008)

- composite adjoint approach is a slight modification of second-order adjoint sensitivity analysis

\[ \frac{d\lambda}{dp_j}^T = -\frac{d\lambda}{dp_j}^T f_x - \lambda^T \left( \frac{d}{dp_j} f_x \right), \quad t \in [t_0, t_k]. \]

\[ \frac{d\lambda}{dp_j}(t_k) = \left( r_{xx} \frac{dx}{dp_j} + r_{xu} \frac{du}{dp_j} \right) \bigg|_{t = t_k}. \]

\( x(t), \lambda(t), \frac{d\lambda}{dp_j}(t) \in \mathbb{R}^{nx}, i = 1, \ldots, n_p \)

\( f : \mathbb{R}^{nx} \times \mathbb{R}^{nu} \rightarrow \mathbb{R}^n, \quad r : \mathbb{R}^{nx} \times \mathbb{R}^{nu} \rightarrow \mathbb{R} \)

(Höyert & Barton, SISC, 2005)

- set of ODEs involving terms with second-order derivatives but partly separable functions
Example: Optimal Control of van der Pol Oscillator (1)

### Total CPU time for optimization (sec)

<table>
<thead>
<tr>
<th>np</th>
<th>BFGS</th>
<th>Hessian</th>
<th>qual. Hessian</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>3.3</td>
<td>2.2</td>
<td>1.3</td>
</tr>
<tr>
<td>50</td>
<td>5.0</td>
<td>4.9</td>
<td>2.8</td>
</tr>
<tr>
<td>100</td>
<td>9.0</td>
<td>10.2</td>
<td>6.6</td>
</tr>
<tr>
<td>250</td>
<td>36.3</td>
<td>32.9</td>
<td>25.3</td>
</tr>
</tbody>
</table>

- 30% reduction in NLP iterations
- Reduced computational effort
- Improved robustness

(Hannemann & Marquardt, 2008)
Example: Optimal Control of van der Pol Oscillator (2)

Scaled CPU time for Jacobian and Hessian evaluation

[\text{msec} / \# \text{parameters}]

<table>
<thead>
<tr>
<th>$n_p$</th>
<th>Jacobian (RK4)</th>
<th>HESS (RK4)</th>
<th>HESS (Euler)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>5.5</td>
<td>23.5</td>
<td>5.5</td>
</tr>
<tr>
<td>50</td>
<td>5.3</td>
<td>23.3</td>
<td>5.2</td>
</tr>
<tr>
<td>100</td>
<td>5.1</td>
<td>22.5</td>
<td>5.0</td>
</tr>
<tr>
<td>250</td>
<td>4.9</td>
<td>22.2</td>
<td>5.0</td>
</tr>
</tbody>
</table>

Jacobian (RK4): forward sensitivities, RK4
HESS (RK4): exact Hessian, discrete composite adjoints, RK4
HESS (Euler): approx. Hessian, discrete composite adjoints, Euler

- Hessian evaluation scales $O(n_p)$
- Cost for high-accuracy approximate Hessian is same as for gradient evaluation!

(Hannemann & Marquardt, 2008)
A Real World Problem

large-scale chemical plant (Shell)
- optimal load change
- 14,000 DAEs, 4 controls, sparse Jacobian
- time horizon >> 24 hrs

adaptive discretization of control #3
- initial guess: 25 pars.
- adaptive parameterization at final solution: 129 pars.
- equivalent equidistant parameterization: 3072 pars.
- CPU time per sensitivity integration (first-order derivative): > 2 h
- total CPU time for (adaptive) optimization: > 1 month

• successful solution of with adaptive discretization (~4 million sensitivity equations)
• non-adaptive solution „impossible“ (~100 million sensitivity equations)
• next challenge: use of second-order derivatives
Agenda

- **Introduction**
  - AD for Higher Order Derivatives and their Applications
  - AD Tools for the User

- **Higher-Order Derivatives in Applications**
  - Numerical Methods – Nonlinear Programming
  - Optimal Control
  - Dynamic Real-time Optimization
  - Inverse Problems – Model-Based Experimental Design
  - Design Optimization under Uncertainty

- **Software Tools**

- **Summary and Conclusions**
Achieve economically optimal plant operation at any time!

Maximize profit

Applications of dynamic real-time optimization
- operation of chemical processes: batch or continuous during transients
- flight management
- and many others
Real-Time Optimization on Receding Horizon

- process and model uncertainties require repetitive solution of similar optimization problems
- fast action requires short cycle times
- large-scale models result in challenging optimization problems
Fast Sensitivity-based Solution Update

- parameterize uncertainty
- exploit sensitivity information of previously solved optimization problem to generate an approximation of the optimal update

Sensitivity system (Fiacco, 1983), invariant active set

\[
\begin{bmatrix}
L_{pp}(\cdot) - g_{p}^{a,T}(\cdot) & 0 \\
g_{p}^{a,T}(\cdot) & 0
\end{bmatrix}
\begin{bmatrix}
p_{\theta} \\
\lambda_{\theta}
\end{bmatrix}
= -
\begin{bmatrix}
L_{p\theta}(\cdot) \\
g_{\theta}(\cdot)
\end{bmatrix}
\]

\[
\Delta p = p(\theta) - p_{0} = p_{\theta}(\theta_{0})\Delta \theta
\]

\[
\Delta \lambda^{a} := \lambda^{a}(\theta) - \lambda_{0}^{a}(\theta_{0})\Delta \theta
\]

\[
\Delta \lambda^{na} := \lambda^{na}(\theta) - \lambda_{0}^{na} = 0
\]

Changing active set (Ganesh & Biegler, 1987)

\[
\min_{\Delta z} 0.5 \Delta p^{T} L_{pp}^{ref} \Delta p + \Delta \theta^{T} L_{p\theta}^{ref} \Delta p
\]

s.t. \[
g_{p}^{ref} \Delta p \geq -g_{\theta}^{ref} \Delta \theta + g^{ref}
\]

- compute first- and second-order derivatives \( L_{pp}, L_{p\theta}, f_{p}, g_{p}, g_{\theta} \)
- solve QP for fast update
- re-iterate if necessary

L: Lagrange function
f: objective function
g: constraints
p: discretized controls
\( \theta \): uncertain parameter
Example – Real-time Optimization of Semi-batch Reactor

Optimization problem

\[
\begin{align*}
\max_{F_B, T_w} & \quad \text{yield}(t_f) \\
\text{s.t.} & \quad 0 \leq F_B(t) \leq 5.784 \ \text{kg/sec} \\
& \quad 20 \leq T_w(t) \leq 100 \ \text{°C} \\
& \quad 60 \leq T_r(t) \leq 90 \ \text{°C} \\
& \quad 0 \leq V(t_f) \leq 5 \ \text{m}^3
\end{align*}
\]

Uncertainties

\[
k_i = a_i \exp\left(\frac{b_i}{T_r + 273.15}\right), \quad i = 1, \ldots, 3
\]

\[
b_{1,0} = 6666.7 \ \text{sec}^{-1} \quad +10 \%
\]

\[
T_{in} = [35 - 10a(t)] \ \text{°C} \quad \theta := [T_{in}, b_1]
\]

Maximize product P at end of batch!

(Würth, Hannemann, Kadam and Marquardt, 2008)
Semi-batch Reactor Example – Fast Solution Update

Control variables

Constrained variables

(Würth, Hannemann, Kadam and Marquardt, 2008)
Semi-batch Reactor Example – Computational Performance

<table>
<thead>
<tr>
<th>Time</th>
<th>31.25</th>
<th>62.5</th>
<th>93.75</th>
<th>125</th>
<th>156.25</th>
<th>187.5</th>
<th>218.75</th>
</tr>
</thead>
<tbody>
<tr>
<td># NLP(^t)</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPU(s)</td>
<td>6.25</td>
<td>1.29</td>
<td>1.13</td>
<td>1.11</td>
<td>1.09</td>
<td>1.06</td>
<td>1.04</td>
</tr>
<tr>
<td>Obj.</td>
<td>1266.7</td>
<td>1265.9</td>
<td>1262.5</td>
<td>1248.9</td>
<td>1204.1</td>
<td>1128</td>
<td>1083.2</td>
</tr>
</tbody>
</table>

Fast updates with second-order derivatives

<table>
<thead>
<tr>
<th>Time</th>
<th>31.25</th>
<th>62.5</th>
<th>93.75</th>
<th>125</th>
<th>156.25</th>
<th>187.5</th>
<th>218.75</th>
</tr>
</thead>
<tbody>
<tr>
<td># NLP(^t)</td>
<td>21</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>CPU(s)</td>
<td>9.58</td>
<td>3.49</td>
<td>6.09</td>
<td>3.53</td>
<td>3.78</td>
<td>4.25</td>
<td>3.89</td>
</tr>
<tr>
<td>Obj.</td>
<td>1266.8</td>
<td>1265.9</td>
<td>1262.5</td>
<td>1249</td>
<td>1204.9</td>
<td>1130.4</td>
<td>1083.9</td>
</tr>
</tbody>
</table>

Rigorous solution (only first-order derivatives)

Exact Hessian facilitates favorable solution update!

(Würth, Hannemann, Kadam and Marquardt, 2008)
Agenda

- **Introduction**
  - AD for Higher Order Derivatives and their Applications
  - AD Tools for the User

- **Higher-Order Derivatives in Applications**
  - Numerical Methods – Nonlinear Programming
  - Optimal Control
  - Dynamic Real-time Optimization
  - Inverse Problems – Model-Based Experimental Design
  - Design Optimization under Uncertainty

- **Software Tools**

- **Summary and Conclusions**
Model-Based Experimental Design

... a model-assisted methodology for model building from experiments!

**Optimal Experimental Design (OED) – Key Idea**

Choose exp. conditions $\mathbf{x}$ such that information on parameter $\theta$ is maximized.

Maximize curvature of parameter estimation objective,

$$\max_{\mathbf{x}} f(\mathbf{x}) = \frac{\partial^2 \phi(\mathbf{x})}{\partial \theta^2}$$

with

$$\frac{\partial^2 \phi(\mathbf{x})}{\partial \theta^2} = F_{\theta} \approx V_{\theta}^{-1}$$
OED for Best Parameter Precision (1)

Mathematical model

\[ f(x, \theta, t) = 0, \quad x \in \Omega, \theta \in \mathbb{R}^{n_p}, n_p \geq 1 \]
\[ y = h(x), \quad y \in \mathbb{R}^{n_{resp}} \]

OED for improved parameter precision: "size" of \( |V_\theta| \) → min

\[ V_\theta \approx \sigma^2 \left( (\partial \phi/\partial \theta)(\partial \phi/\partial \theta)^T \right)^{-1} \]

Different metrics for \( |V_\theta| \)

- D – optimality: min det \( V_\theta \)
- E – optimality: min \( \lambda_{max} \)
- A – optimality: min \( tr(V_\theta) \)
OED for Best Parameter Precision (2)

\[
\min_\theta |\sigma^2 \left( \left( \frac{\partial \phi}{\partial \theta} \right) \left( \frac{\partial \phi}{\partial \theta} \right)^T \right)^{-1} |
\]

s.t. model equations

\[ \frac{\partial y}{\partial \theta} \]

numerical solution requires higher order derivatives!

first-order gradient-based NLP solver \(\leftarrow\) second-order derivatives
second-order gradient-based NLP solver \(\leftarrow\) third-order derivatives

\[ \frac{\partial y}{\partial \theta} \] may involve sensitivities of the solution of a set of parametric evolution equations (ODE or PDE)
Example – Diffusion Modeling Using Spectroscopic Data

- analysis of one-dimensional diffusion
- simultaneous measurement of all mole fractions
- nonlinear calibration
- high resolution ($\Delta t = 10 \text{ s}$, $\Delta z = 20 \mu\text{m}$)
- measurement error: statistical $\leq 0.2 \text{ mol-\%}$, systematic $\leq 0.5 \text{ mol-\%}$

Experimental Design Issues

- what mixture volume ratio?
- where to measure?
- how long to measure?
- which mixture compositions?

optimal experimental design for diffusion coefficient measurements

Optimal Experimental Conditions - Visualization

example: acetone-benzene-methanol

measurements at the wall, i.e., restricted diffusion experiments
• unequal volume of both phases, actual value depending on molar volume
• optimal experiment duration depends on cell size → optimal Fourier number

OED-Problem for 1D Multi-Component Diffusion

\[ \zeta(D, d) := \left[ \frac{\det(F_D(d))}{\det(F_D(d^*))} \right]^{1/n_p} \rightarrow \max \]

subject to diffusion model:

\[ \frac{\partial c_i}{\partial t} = \frac{\partial}{\partial z} \left( \sum_{k=1}^{n_c-1} D_{ik} \frac{\partial c_k}{\partial z} \right), \quad i = 1, \ldots, n_c - 1, \]

\[ x_i = \frac{c_i}{c_t}, \quad i = 1, \ldots, n_c, \]

\[ 0 = c_t - \sum_{i=1}^{n_c} c_i, \]

\[ \frac{1}{c_t} = \sum_{i=1}^{n_c} x_i V_i^0(T) \]

... at least 2\textsuperscript{nd}, better 3\textsuperscript{rd} order sensitivities of solution to PDE model

Agenda

- **Introduction**
  - AD for Higher Order Derivatives and their Applications
  - AD Tools for the User

- **Higher-Order Derivatives in Applications**
  - Numerical Methods – Nonlinear Programming
  - Optimal Control
  - Dynamic Real-time Optimization
  - Inverse Problems – Model-Based Experimental Design
  - Design Optimization under Uncertainty

- **Software Tools**

- **Summary and Conclusions**
Design Optimization – Constructive Nonlinear Dynamics

Chemical processes are complex:

- one unit operation is already complex
- plant is combination of many unit operations
- feedback and recycling increases complexity

Complexity and nonlinearities can lead to unexpected process behavior:

- ammonia synthesis on industrial scale since 1916
- >80 years later still surprising behavior

Include dynamic properties and uncertainty in the process design!

reactor temperature of an industrial ammonia synthesis:

Include dynamic properties and uncertainty in the process design!
Simple Example

Semi-batch bioreactor

\[ F, S_F \]

\[ \begin{align*} 
S, & \quad \text{Substrate conc.} \\
X, & \quad \text{Biomass density} 
\end{align*} \]

Process model of the bioreactor

\[ \begin{align*} 
\frac{dX}{dt} &= -\frac{F}{V} X + \mu(S)X, \\
\frac{dS}{dt} &= \frac{F}{V} (S_F - S) - \sigma(S)X, \\
\mu(S) &= kS \exp(-S / K), \\
\sigma(S) &= \frac{\mu(S)}{a + bS} 
\end{align*} \]

Nonlinear program

\[ \begin{align*} 
\text{max} & \quad \Phi \\
F, & \quad S_F \\
\text{subject to} & \\
- & \text{steady state operating point} \\
- & \text{bounds on } S_F \\
0.3 & \leq S_F \leq 1.0 \text{ kmol m}^{-3} 
\end{align*} \]
Stability at the Design Optimum

Unstable optimum not operable!
Detection of Stability Boundary

- detection of stability boundaries by evaluating the eigenvalues
- stability boundary separates parameter space \((S_F, F)\)
- stability boundary defined only implicitly
Normal Vector Method

- model uncertainties
  \[ \alpha \in [\alpha^{(0)} \pm \Delta \alpha] \]
  parameters not known exactly, drifting parameters
- closest critical points is along the normal vector direction \( r \)
  (Dobson, 1993)
- normal vector constraints for optimization problem
  (Mönnigmann & Marquardt, 2002)

Normal vector is one-dimensional regardless of the number of parameters

complexity grows only linear with the number of parameters
Design Optimization – Problem Formulation

**Bioreactor**

\[ F, S_F \quad \text{max } \Phi \]

**Nonlinear program**

\[
\begin{align*}
\text{max } & \Phi \\
F, S_F & \\
\text{subject to} & \\
- \text{steady state operating point} & \\
- \text{bounds on } S_F & 0.3 \leq S_F \leq 1.0 \text{ kmol m}^{-3} \\
- \text{normal vector constraints to stability boundary} & \\
\end{align*}
\]

**Consider systematically**

- desired dynamic behavior
- process uncertainties

**Critical manifolds**

- stability boundaries
- eigenvalue bounds
- feasibility constraints
Design Optimization with Normal Vector Constraints

$$\max_{x^{(eq)}, p, \alpha^{(eq)}, l^{(i)}} \phi(x^{(eq)}, p, \alpha^{(eq)}) \quad \text{objective}$$

$$0 = f(x^{(eq)}, p, \alpha^{(eq)}),$$

$$0 = G^{(i)}(x^{(i)}, p, \alpha^{(i)}, r^{(i)}),$$

$$\alpha^{(i)} = \alpha^{(eq)} + l^{(i)} r^{(i)},$$

$$l^{(i)} \| r^{(i)} \| \geq \sqrt{n_p},$$

$$i = 1, \ldots, I$$

steady state \((x^{(eq)}, p, \alpha^{(eq)})\)

normal vector \(r^{(i)}\) for critical manifold \(i\)

distance between operating point and critical manifold \(i\)

**Types of critical manifolds:**

- stability boundaries
- feasibility constraints
- performance constraints (eigenvalue sectors)
- higher codimension bifurcations (cusp, isola, ...)

| AACHENER VERFAHRENSTECHNIK | Higher-Order Derivatives in Engineering Applications, AD 2008, August 11 - 15 | 44 |
Normal Vector Equation Systems

**Saddle-node bifurcation**

**augmented system**
\[ 0 = f \]
\[ 0 = f_x v \]
\[ 0 = v^T v - 1 \]
\[ 0 = r - f^T f \alpha \]

**Grazing bifurcation**

**augmented system**
\[ \dot{x} = f(x(t), \alpha, t), \quad x(0) = x_0 \]
\[ 0 = h(x(t), \alpha, t) \]
\[ 0 = h_x x + h_\alpha \]
\[ \dot{x}_\alpha = f_x x_\alpha + f_\alpha, \quad x_\alpha(0) = 0 \]
\[ 0 = h_x x_\alpha + h_\alpha - r \]

**Hopf bifurcation**

**augmented system**
\[ 0 = f \]
\[ 0 = f_x w^{(1)} + \omega w^{(2)} \]
\[ 0 = v^{(1)}T w^{(1)} + v^{(2)}T w^{(2)} - 1 \]
\[ 0 = v^{(1)}T w^{(2)} - v^{(2)}T w^{(1)} \]
\[ 0 = f_x T u + v^{(1)}T f_{xx} w^{(1)} + v^{(2)}T f_{xx} w^{(2)} \]
\[ 0 = r - f_\alpha u + v^{(1)}T f_{x\alpha} w^{(1)} + v^{(2)}T f_{x\alpha} w^{(2)} \]

**Normal vector constraints require first and second order derivatives.**

**Gradient based NLP solver requires at least second/third order derivatives**
Time-Domain Constraints

Eigenvalue bounds guarantee asymptotic behavior

In the presence of disturbances the transient process behavior has to be considered.

Definition of new types of critical manifolds are necessary.
Critical Manifolds for Time-Domain Constraints

- **Grazing Bifurcation** \((x)\) (Nordmark 1991)

  trajectory touches state constraint

\[ x(t^g,p_1) = x_{\text{max}} \]

- critical manifold in the parameter space
- separates region in which constraint is violated from region in which constraint is not violated
Normal Vector Equation Systems

Grazing bifurcation

augmented system

\[ x = \varphi(x(t_0), t_0, p, \alpha, t) \]
\[ 0 = h(x(t), \alpha, t) \]
\[ 0 = h_x x + h_t \]
\[ x_\alpha = \varphi_x(x(t_0), t_0, p, \alpha, t) \]
\[ 0 = h_x x_\alpha + h_\alpha - r \]

Endpoint constraints

augmented system

\[ x = \varphi(x(t_0), t_0, p, \alpha, t) \]
\[ 0 = h(x(t), \alpha, t) \]
\[ 0 = t - t^e \]
\[ x_\alpha = \varphi_x(x(t_0), t_0, p, \alpha, t) \]
\[ 0 = h_x x_\alpha + h_\alpha - r \]

Normal vector equation systems for time-domain constraints

- flux of nonlinear systems
- first-order sensitivities of flux w.r.t. uncertain parameters
- gradients of constraints require second-order sensitivities of the flux w.r.t. parameters \( \alpha \) and \( p \)
Agenda

- **Introduction**
  - AD for Higher Order Derivatives and their Applications
  - AD Tools for the User

- **Higher-Order Derivatives in Applications**
  - Numerical Methods – Nonlinear Programming
  - Optimal Control
  - Dynamic Real-time Optimization
  - Inverse Problems – Model-Based Experimental Design
  - Design Optimization under Uncertainty

- **Software Tools**
  - Normal Vector Approach
  - Optimal Control

- **Summary and Conclusions**
Augmented process model involves higher order derivatives of process model equations or flows (normal vector constraints)

- process model is coded in MAPLE (Monagan et al. 2000)

- normal vector constraints (1st and 2nd derivatives, SN, Hopf) are calculated by forward AD algorithm implemented in MAPLE to augment process model to result in design optimization problem

- first-order derivatives (gradient) for numerical solution of NLP calculated by automatic differentiation with ADIFOR (Bischof et al. 1998), generation of second-order derivatives (Hessian) not possible

- ... a work-around for higher-order derivatives based on AD

- optimization, evaluation of test functions along the linear path between initial guess and optimal end point
Agenda

- **Introduction**
  - AD for Higher Order Derivatives and their Applications
  - AD Tools for the User

- **Higher-Order Derivatives in Applications**
  - Numerical Methods – Nonlinear Programming
  - Optimal Control
  - Dynamic Real-time Optimization
  - Inverse Problems – Model-Based Experimental Design
  - Design Optimization under Uncertainty

- **Software Tools**
  - Normal Vector Approach
  - Optimal Control

- **Summary and Conclusions**
A Typical Optimization Problem …

- requires to code a (e.g. DAE) model
  \[ M \dot{x} = F(x(t), u(t), p, t), \quad t \in [t_0, t_f], \]
  \[ 0 = x(t_0) - x_0 \]

- with path and end point constraints
  \[ 0 \geq P(x, u, p, t), \quad t \in [t_0, t_f], \]
  \[ 0 \geq E(x(t_f)) \]

- and an objective function
  \[ \min_{p,u} \Phi(x(u, p, t_f)) \]

- by means of an advanced modeling and simulation environment.
DyOS – Dynamic Optimization Software Environment

- commercial modeling and simulation environment for process engineering
- high-level equation- / object-oriented modeling language
- model compiler generates semi-discretized DAE model and Jacobian (by AD)
- ESO interface to export right hand side and Jacobian values
- interfaces to native numerical solution engines
- ESO interfaces to native numerical solution engines

---

Modelica/CapeML/ADiCape

- Dynamic link library
- CAPE-ML
- Translation
- Automatic differentiation

---

gPROMS

- process model
- CAPE-OPEN compliant software interface
- CORBA Object Bus

---

DyOS

- initial trajectory
- grid refinement
- stopping criterion
- optimal trajectory

---

SetVariables
GetResiduals
Translation
Modelica-model
Automatic differentiation
CAPE-ML
Translation
Modelica-model
Dynamic link library

---

Process Systems Enterprise Ltd.

Bischof and Marquardt groups

Marquardt group
Modelica (www.modelica.org)
- open source, high-level equation-oriented modeling language for DAE systems

CapeML (von Wedel, 2002)
- XML-based model representation for model exchange
- structured, equation-oriented models; follows systems approach

DyOS – Dynamic Optimization Software Environment

---

Marquardt group

Bischof and Marquardt groups
A Simple Example (Car Dynamics)

Modelica code

```modelica
decision a; model car
Real velo;
Real dist;
Real time;
Real parameter accel;
Real parameter alpha;
Real constant pi;

equation
  der(dist) = velo;
  der(time) = 1.0;
  der(velo) = 4/pi * arctan(accel) - alpha*power(velo,2);
end car;
```

CapeML code fragment

```xml
<Equation>
  <BalancedEquation myID="V-0">
    <Expression>
      <Term>
        <Factor>
          <FunctionCall fcn.name="der">
            <Expression>
              <Term>
                <Factor>
                  <VariableOccurrence definition="V-car-dist"/>
                </Factor>
              </Term>
            </Expression>
          </FunctionCall>
        </Factor>
        <Factor>
          <FunctionCall fcn.name="der">
            <Expression>
              <Term>
                <Factor>
                  <VariableOccurrence definition="V-car-dist"/>
                </Factor>
              </Term>
            </Expression>
          </FunctionCall>
        </Factor>
      </Expression>
    </Expression>
    </BalancedEquation>
  </Equation>
```
Modelica (www.modelica.org)
  - open source, high-level equation-/object-oriented modeling language for DAE systems

CapeML (von Wedel, 2002)
  - XML based model representation for model exchange
  - structured, equation-oriented models, follows systems approach

ADiCape (Bischof et al., 2005)
  - model transformation from Modelica to CapeML (via XSLT)
  - AD applied to CapeML code
  - extended ESO interface for values of right hand side, Jacobian, Jacobian*vector, Hessian & symmetric projection of Hessian
Agenda

- **Introduction**
  - AD for Higher Order Derivatives and their Applications
  - AD Tools for the User

- **Higher-Order Derivatives in Applications**
  - Numerical Methods – Nonlinear Programming
  - Optimal Control
  - Dynamic Real-time Optimization
  - Inverse Problems – Model-Based Experimental Design
  - Design Optimization under Uncertainty

- **Software Tools**
  - Normal Vector Approach
  - Optimal Control

- **Summary and Conclusions**
Summary

- **higher-order derivatives are more than a nice-to-have**
  - NLP solvers are more efficient if exact Hessian is available
  - many applications result in NLP involving first- or second-order derivates of
    - functions (right hand sides of models) or
    - flux (solution of evolution equations)
- **models often large-scale involving complex nonlinear terms**
  - (manual) symbolic differentiation and implementation impractical
  - higher-order AD is highly desirable, but users are not aware of existing capabilities
- **models are coded in high-level modeling languages**
  - AD of computer programs is therefore often not appropriate
  - AD technology has to be built on modeling languages in addition to programming languages
Lessons Learned – Rough Software Architecture

Computational Engineer

modeling environment A
neutral model representation (XML-based?)
extended neutral model representation (derivatives, additional equations involving derivatives)
efficient model implementation (executable)

modeling environment Z

model implementation
model translation
model extension & AD
code generation

numerical solvers
Conclusions – We Want Your Expertise!

- easy-to-use tool for computational systems engineering applications should support
  - high-level modeling languages
  - facilitate model extensions prior to AD (e.g. symbolic manipulation)
  - derivatives of functions and solutions of evolution equations up to order 4
  - „generalized seeding“ to define terms involving derivatives
  - replacement of recurrent terms
  - code generation for efficient numerical evaluation
  - standard interfaces to numerical solvers (e.g. CAPE-OPEN)
Conclusions – We Want Your Expertise!

- Run-time efficiency is a decisive issue for technology acceptance
  - Large-scale, strongly (non-polynomial) nonlinear models
  - Computational complexity should not only grow with higher-order derivatives
  - Model development requires extremely short processing times to facilitate interactive work process

- Cooperation between problem owners and AD technology providers is highly desirable!

Thank You!