Introduction to Algorithmic Differentiation

Kshitij Kulshreshtha

Universität Paderborn

ESSIM Summer School
Dresden
Schedule

1. Introduction/Motivation/Function Representation/Forward mode
2. Reverse Mode/Higher order Derivatives
3. Implementations/ADOL-C
4. Hand on Exercises (ADOL-C)
Computing Derivatives

Simulation  \[\xrightarrow{\text{Sensitivity Calculation}}\] Optimization
Computing Derivatives

Simulation → Sensitivity → Calculation → Optimization

Data → Input x → User → Modelling → Computer Program → Output y

K. Kulshreshtha
Computing Derivatives
Computing Derivatives

Simulation \[ \frac{\text{Sensitivity}}{\text{Calculation}} \] Optimization

data

user

modelling

computer program

input \( x \)

output \( y \)

theory

differentiation

sensitivity \( \frac{\partial y}{\partial x} \)

Enhanced program

Optimization algorithm

User

Intro to AD

ESSIM Summer School
Computing Derivatives

Given:

Description of functional relation as

- formula $y = F(x)$  $\rightarrow$  explicit expression $y' = F'(x)$
- computer program  $\rightarrow$  ?
Computing Derivatives

Given:

Description of functional relation as
- formula $y = F(x)$  $\rightarrow$ explicit expression $y' = F'(x)$
- computer program  $\rightarrow$ ?

Task:

Computation of derivatives taking
- requirements on exactness
- computational effort

into account
Derivation: What for?
Derivation: What for?

- Optimization:
  - unbounded: \( \min f(x) \), \( f : \mathbb{R}^n \to \mathbb{R} \)
  - bounded: \( \min f(x) \), \( c(x) = 0 \), \( h(x) \leq 0 \),
    \( f : \mathbb{R}^n \to \mathbb{R} \), \( c : \mathbb{R}^n \to \mathbb{R}^m \), \( h : \mathbb{R}^n \to \mathbb{R}^l \)
Derivation: What for?

▶ Optimization:

unbounded: \( \min f(x), \quad f : \mathbb{R}^n \to \mathbb{R} \)

bounded: \( \min f(x), \quad f : \mathbb{R}^n \to \mathbb{R} \)
\( c(x) = 0, \quad c : \mathbb{R}^n \to \mathbb{R}^m \)
\( h(x) \leq 0, \quad h : \mathbb{R}^n \to \mathbb{R}^l \)

▶ Solution of nonlinear equation systems

\( F(x) = 0, \quad F : \mathbb{R}^n \to \mathbb{R}^n \)

Newton method requires \( F'(x) \in \mathbb{R}^{n \times n} \)
Derivation: What for?

- **Optimization:**
  
  **unbounded:** \( \min f(x), \quad f : \mathbb{R}^n \to \mathbb{R} \)
  
  **bounded:** \( \min f(x), \quad f : \mathbb{R}^n \to \mathbb{R} \)

  \[ c(x) = 0, \quad c : \mathbb{R}^n \to \mathbb{R}^m \]

  \[ h(x) \leq 0, \quad h : \mathbb{R}^n \to \mathbb{R}^l \]

- **Solution of nonlinear equation systems**

  \[ F(x) = 0, \quad F : \mathbb{R}^n \to \mathbb{R}^n \]

  Newton method requires \( F'(x) \in \mathbb{R}^{n \times n} \)

- **Simulation of complex procedures**
  
  - system definition
  
  - integration of differential equations
Derivation: What for?

- Optimization:
  - Unbounded: \[ \min f(x), \quad f : \mathbb{R}^n \to \mathbb{R} \]
  - Bounded: \[ \min f(x), \quad c(x) = 0, \quad c : \mathbb{R}^n \to \mathbb{R}^m \]
  - \[ h(x) \leq 0, \quad h : \mathbb{R}^n \to \mathbb{R}^l \]

- Solution of nonlinear equation systems
  \[ F(x) = 0, \quad F : \mathbb{R}^n \to \mathbb{R}^n \]

  Newton method requires \[ F'(x) \in \mathbb{R}^{n \times n} \]

- Simulation of complex procedures
  - System definition
  - Integration of differential equations

- Sensitivity analysis
- Real-time control
Symbolic Differentiation
Symbolic Differentiation

\[ x^2 \Rightarrow 2x \]
Symbolic Differentiation

\[ x^2 \Rightarrow 2x \]
\[ x^p \Rightarrow px^{p-1} \]
Symbolic Differentiation

\[ x^2 \Rightarrow 2x \]
\[ x^p \Rightarrow px^{p-1} \]
\[ \cos(x) \Rightarrow -\sin(x) \]
Symbolic Differentiation

\[ x^2 \Rightarrow 2x \]
\[ x^p \Rightarrow px^{p-1} \]
\[ \cos(x) \Rightarrow -\sin(x) \]
\[ \exp(x) \Rightarrow \exp(x) \]
Symbolic Differentiation

\[ x^2 \Rightarrow 2x \]
\[ x^p \Rightarrow px^{p-1} \]
\[ \cos(x) \Rightarrow -\sin(x) \]
\[ \exp(x) \Rightarrow \exp(x) \]
\[ \log(x) \Rightarrow \frac{1}{x} \]
Symbolic Differentiation

\[ f \quad \gamma \quad \delta \quad \Phi \quad \pm \]
\[ \int \sum x \quad \int y_2 \]
\[ \eta \quad \ldots \quad t_* \quad \partial \]
\[ \beta \]

\[ x^2 \Rightarrow 2x \]
\[ x^p \Rightarrow px^{p-1} \]
\[ \cos(x) \Rightarrow -\sin(x) \]
\[ \exp(x) \Rightarrow \exp(x) \]
\[ \log(x) \Rightarrow \frac{1}{x} \]
\[ g(h(x)) \Rightarrow g'(h(x)) \cdot h'(x) \]
\[ \ldots \]
Symbolic Differentiation

- exact derivatives
  - \( f(x) = \exp(\sin(x^2)) \Rightarrow \)
    \[ f'(x) = \exp(\sin(x^2)) \cdot \cos(x^2) \cdot 2x \]
- \( \min J(x, u) \) such that \( x' = f(x, u) + IC \)
  - reduced formulation: \( J(x, u) \rightarrow \hat{J}(u) \)
  - \( \hat{J}'(u) \) based on symbolic adjoint \( \lambda' = -f_x(x, u)^\top \lambda + TC \)
Symbolic Differentiation

- exact derivatives
  - \( f(x) = \exp(\sin(x^2)) \) \( \Rightarrow \)
    \[
    f'(x) = \exp(\sin(x^2)) \cdot \cos(x^2) \cdot 2x
    \]
- \( \min J(x, u) \) such that \( x' = f(x, u) + \text{IC} \)
  - reduced formulation: \( J(x, u) \rightarrow \hat{J}(u) \)
  - \( \hat{J}'(u) \) 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
read_input_file(argv[1], &code_control);
  code_control.timestep_type = 0; // calculate timestep size like TAU
  // ... 0;
} 

Jan 01, 08 21:46 Seite 30/30 euler2d.c
Gedruckt von Andrea Walther
Dienstag Januar 01, 2008 15/15 euler2d.c
K. Kulshreshtha 8 / 94 Intro to AD ESSIM Summer School

Jan 01, 08 21:46 Seite 30/30 euler2d.c
Gedruckt von Andrea Walther
Dienstag Januar 01, 2008 15/15 euler2d.c

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) /
    (code_control.full_2nd_order - code_control.start_2nd_order))
    weight = 1.0;
  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
", sum_it, residual / first_residual, lift, drag, weight);
  fprintf(conv, "%d %20.10e %20.10e %20.10e
", sum_it+first_step, residual / first_residual, lift, drag);
  sum_it++;
}

Jan 01, 08 21:46 Seite 29/30 euler2d.c
Gedruckt von Andrea Walther
Dienstag Januar 01, 2008 15/15 euler2d.c
K. Kulshreshtha 8 / 94 Intro to AD ESSIM Summer School
Finite Differences

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

$$f(x + h) = f(x) + h' (x) + h^2 f'' (x) / 2 + h^3 f'''(x) / 6 + \cdots$$

$$\Rightarrow f(x + h) \approx f(x) + hf'(x)$$

$$\Rightarrow Df(x) \frac{f(x + h) - f(x)}{h}$$
Finite Differences

**Idea:** Taylor-expansion, \( f : \mathbb{R} \to \mathbb{R} \) smooth then

\[
f(x + h) = f(x) + h'f(x) + h^2f''(x)/2 + h^3f'''(x)/6 + \cdots
\]

\[
\Rightarrow f(x + h) \approx f(x) + hf'(x)
\]

\[
\Rightarrow Df(x) \frac{f(x + h) - f(x)}{h}
\]

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

\( F : \mathbb{R}^n \to \mathbb{R} \Rightarrow OPS(\nabla F(x)) \sim (n + 1)OPS(F(x)) \)
Example 1
Example 1
Example 1
Example 1
Example 1
Example 1
Example 2
Example 2
Example 2

\[ f(x) \]

\[ x = \bar{x}, \bar{x} - h, \bar{x} + h \]
Example 2

\[ f(x) \]

\( x - h \)  \( x \)  \( x + h \)
Example 2
Algorithmic Differentiation (AD)

Main Products:

- Quantitative dependence information (local):
  - Weighted and directed partial derivatives
  - Error and condition number estimates
  - Lipschitz constants, interval enclosures
  - Eigenvalues, Newton steps
Algorithmic Differentiation (AD)

Main Products:

- Quantitative dependence information (local):
  - Weighted and directed partial derivatives
  - Error and condition number estimates . . .
  - Lipschitz constants, interval enclosures . . .
  - Eigenvalues, Newton steps . . .

- Qualitative dependence information (regional):
  - Sparsity structures, degrees of polynomials
  - Ranks, eigenvalue multiplicities . . .
Algorithmic Differentiation (AD)

Main Properties:
- No truncation errors
Algorithmic Differentiation (AD)

Main Properties:

- No truncation errors
- Chain rule applied to numbers
Algorithmic Differentiation (AD)

Main Properties:

- No truncation errors
- Chain rule applied to numbers
- Applicability to "arbitrary programs"
Algorithmic Differentiation (AD)

Main Properties:

- 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.
Algorithmic Differentiation (AD)

Main Properties:

- 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. (Griewank, Walther 09)
The “Hello-World”-Example of AD

\[
y_1 = \tan(\omega t) - \tan(\omega t)
\]

and

\[
y_2 = \gamma \tan(\omega t) - \tan(\omega t)
\]

K. Kulshreshtha 23 / 94 Intro to AD ESSIM Summer School
The “Hello-World”-Example of AD

Lighthouse

\[ y_1 = \tan(\omega t) \quad \gamma - \tan(\omega t) \]

\[ y_2 = \gamma \tan(\omega t) \quad \gamma - \tan(\omega t) \]
The “Hello-World”-Example of AD

\[ y_2 = \gamma y_1 \]

\[ y_2 = \nu \tan(\omega t) - \nu \tan(\omega t) \]

\[ y_2 = \gamma \nu \tan(\omega t) \]
The "Hello-World"-Example of AD

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

Lighthouse

\[ y_2 = \gamma y_1 \]
Automatisation: Computer tolls as Maple, Mathematica, ... 

Maple provides

> y1 := nu * tan(omega * t)/(gamma - tan(omega * t));

\[ y1 := \frac{\nu \tan(\omega t)}{\gamma - \tan(\omega t)} \]

> y2 := gamma * nu * tan(omega * t)/(gamma - tan(omega * t));

\[ y2 := \frac{\gamma \nu \tan(\omega t)}{\gamma - \tan(\omega t)} \]
Automatisation:
Computer tolls as Maple, Mathematica, ... 

Maple provides

\[
\begin{align*}
\text{> y1} & := \text{nu} \ast \tan(\omega t)/(\gamma - \tan(\omega t))
\end{align*}
\]

\[
\begin{align*}
y1 & := \frac{v \tan(\omega t)}{\gamma - \tan(\omega t)}
\end{align*}
\]

\[
\begin{align*}
\text{> y2} & := \gamma v \ast \text{nu} \ast \tan(\omega t)/(\gamma - \tan(\omega t))
\end{align*}
\]

\[
\begin{align*}
y2 & := \frac{\gamma v \tan(\omega t)}{\gamma - \tan(\omega t)}
\end{align*}
\]

\[
\begin{align*}
\text{> diff(y2,nu)};
\end{align*}
\]

\[
\begin{align*}
\frac{\gamma \tan(\omega t)}{\gamma - \tan(\omega t)}
\end{align*}
\]
Automatisation:
Computer tolls as Maple, Mathematica, . . .

Maple provides

> y1 := nu * tan(omega * t)/(gamma - tan(omega * t));

\[ y1 := \frac{\nu \tan(\omega t)}{\gamma - \tan(\omega t)} \]

> y2 := gamma * nu * tan(omega * t)/(gamma - tan(omega * t));

\[ y2 := \frac{\gamma \nu \tan(\omega t)}{\gamma - \tan(\omega t)} \]

> diff(y2,gamma);

\[ \frac{\nu \tan(\omega t)}{\gamma - \tan(\omega t)} - \frac{\gamma \nu \tan(\omega t)}{(\gamma - \tan(\omega t))^2} \]
Automatisation:
Computer tolls as Maple, Mathematica, . . .

Maple provides

\[
\begin{align*}
\text{y1 := } & \frac{v \tan(\omega t)}{\gamma - \tan(\omega t)} \\
\text{y2 := } & \frac{\gamma v \tan(\omega t)}{\gamma - \tan(\omega t)} \\
\text{diff(y2,omega); } & \frac{v (1 + \tan(\omega t)^2)}{\gamma - \tan(\omega t)} t + \frac{v \tan(\omega t) (1 + \tan(\omega t)^2)}{(\gamma - \tan(\omega t))^2} t
\end{align*}
\]
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 Mode (Lighthouse)

\[
\begin{align*}
V_{-3} &= x_1 = \nu \\
V_{-2} &= x_2 = \gamma \\
V_{-1} &= x_3 = \omega \\
V_0 &= x_4 = t \\
V_1 &= V_{-1} \ast V_0 \\
V_2 &= \tan(V_1) \\
V_3 &= V_{-2} - V_2 \\
V_4 &= V_{-3} \ast V_2 \\
V_5 &= V_4 / V_3 \\
V_6 &= V_5 \ast V_{-2} \\
y_1 &= V_5 \\
y_2 &= V_6
\end{align*}
\]
Forward Mode (Lighthouse)

| \( v_{-3} \) | = | \( x_1 = \nu \) | \( \dot{v}_{-3} \) | = | \( \dot{x}_1 \) |
| \( v_{-2} \) | = | \( x_2 = \gamma \) | \( \dot{v}_{-2} \) | = | \( \dot{x}_2 \) |
| \( v_{-1} \) | = | \( x_3 = \omega \) | \( \dot{v}_{-1} \) | = | \( \dot{x}_3 \) |
| \( v_0 \) | = | \( x_4 = t \) | \( \dot{v}_0 \) | = | \( \dot{x}_4 \) |

\[
\begin{align*}
    v_1 & = v_{-1} \ast v_0 \\
    v_2 & = \tan(v_1) \\
    v_3 & = v_{-2} - v_2 \\
    v_4 & = v_{-3} \ast v_2 \\
    v_5 & = v_4 / v_3 \\
    v_6 & = v_5 \ast v_{-2} \\
    y_1 & = v_5 \\
    y_2 & = v_6
\end{align*}
\]
Forward Mode (Lighthouse)

\[
\begin{align*}
\mathbf{v}_3 &= x_1 = \nu \\
\mathbf{v}_2 &= x_2 = \gamma \\
\mathbf{v}_1 &= x_3 = \omega \\
\mathbf{v}_0 &= x_4 = t
\end{align*}
\]

\[
\begin{align*}
\dot{\mathbf{v}}_3 &= \dot{x}_1 \\
\dot{\mathbf{v}}_2 &= \dot{x}_2 \\
\dot{\mathbf{v}}_1 &= \dot{x}_3 \\
\dot{\mathbf{v}}_0 &= \dot{x}_4
\end{align*}
\]

\[
\begin{align*}
\mathbf{v}_1 &= \mathbf{v}_3 \times \mathbf{v}_0 \\
\mathbf{v}_2 &= \tan(\mathbf{v}_1) \\
\mathbf{v}_3 &= \mathbf{v}_2 - \mathbf{v}_2 \\
\mathbf{v}_4 &= \mathbf{v}_3 \times \mathbf{v}_2 \\
\mathbf{v}_5 &= \mathbf{v}_4 / \mathbf{v}_3 \\
\mathbf{v}_6 &= \mathbf{v}_5 \times \mathbf{v}_2 \\
\dot{\mathbf{v}}_1 &= \dot{\mathbf{v}}_3 \times \mathbf{v}_0 + \mathbf{v}_3 \times \dot{\mathbf{v}}_0
\end{align*}
\]

\[
\begin{align*}
\mathbf{y}_1 &= \mathbf{v}_5 \\
\mathbf{y}_2 &= \mathbf{v}_6
\end{align*}
\]
### Forward Mode (Lighthouse)

<table>
<thead>
<tr>
<th>$v_3$</th>
<th>$x_1 = \nu$</th>
<th>$\dot{v}_3 = \dot{x}_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_2$</td>
<td>$x_2 = \gamma$</td>
<td>$\dot{v}_2 = \dot{x}_2$</td>
</tr>
<tr>
<td>$v_1$</td>
<td>$x_3 = \omega$</td>
<td>$\dot{v}_1 = \dot{x}_3$</td>
</tr>
<tr>
<td>$v_0$</td>
<td>$x_4 = t$</td>
<td>$\dot{v}_0 = \dot{x}_4$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$v_1$</th>
<th>$v_3 * v_0$</th>
<th>$\dot{v}_1 = \dot{v}_3 * v_0 + v_3 * \dot{v}_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_2$</td>
<td>$\tan(v_1)$</td>
<td>$\dot{v}_2 = \dot{v}_1 / \cos(v_1)^2$</td>
</tr>
<tr>
<td>$v_3$</td>
<td>$v_2 - v_2$</td>
<td>$\dot{v}_3 = v_2 - v_2$</td>
</tr>
<tr>
<td>$v_4$</td>
<td>$v_3 * v_2$</td>
<td></td>
</tr>
<tr>
<td>$v_5$</td>
<td>$v_4 / v_3$</td>
<td></td>
</tr>
<tr>
<td>$v_6$</td>
<td>$v_5 * v_2$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$y_1$</th>
<th>$v_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_2$</td>
<td>$v_6$</td>
</tr>
</tbody>
</table>
**Forward Mode (Lighthouse)**

| $v_{-3}$ | $x_1 = \nu$ | $\dot{v}_{-3} = \dot{x}_1$ |
| $v_{-2}$ | $x_2 = \gamma$ | $\dot{v}_{-2} = \dot{x}_2$ |
| $v_{-1}$ | $x_3 = \omega$ | $\dot{v}_{-1} = \dot{x}_3$ |
| $v_0$ | $x_4 = t$ | $\dot{v}_0 = \dot{x}_4$ |
| $v_1$ | $v_{-1} * v_0$ | $\dot{v}_1 = \dot{v}_{-1} * v_0 + v_{-1} * \dot{v}_0$ |
| $v_2$ | $\tan(v_1)$ | $\dot{v}_2 = \dot{v}_1 / \cos(v_1)^2$ |
| $v_3$ | $v_{-2} - v_2$ | $\dot{v}_3 = \dot{v}_{-2} - \dot{v}_2$ |
| $v_4$ | $v_{-3} * v_2$ |
| $v_5$ | $v_4 / v_3$ |
| $v_6$ | $v_5 * v_{-2}$ |
| $y_1$ | $v_5$ |
| $y_2$ | $v_6$ |
### Forward Mode (Lighthouse)

| \( v_{-3} \) | \( x_1 = \nu \) | \( \dot{v}_{-3} \) | \( \dot{x}_1 \) |
|\( v_{-2} \) | \( x_2 = \gamma \) | \( \dot{v}_{-2} \) | \( \dot{x}_2 \) |
|\( v_{-1} \) | \( x_3 = \omega \) | \( \dot{v}_{-1} \) | \( \dot{x}_3 \) |
|\( v_0 \) | \( x_4 = t \) | \( \dot{v}_0 \) | \( \dot{x}_4 \) |

| \( v_1 \) | \( v_{-1} * v_0 \) | \( \dot{v}_1 \) | \( \dot{v}_{-1} * v_0 + v_{-1} * \dot{v}_0 \) |
|\( v_2 \) | \( \tan(v_1) \) | \( \dot{v}_2 \) | \( \dot{v}_1 / \cos(v_1)^2 \) |
|\( v_3 \) | \( v_{-2} - v_2 \) | \( \dot{v}_3 \) | \( \dot{v}_{-2} - \dot{v}_2 \) |
|\( v_4 \) | \( v_{-3} * v_2 \) | \( \dot{v}_4 \) | \( \dot{v}_{-3} * v_2 + v_{-3} * \dot{v}_2 \) |
|\( v_5 \) | \( v_4 / v_3 \) |
|\( v_6 \) | \( v_5 * v_{-2} \) |

\[ y_1 = v_5 \]
\[ y_2 = v_6 \]
# Forward Mode (Lighthouse)

<table>
<thead>
<tr>
<th>( v_1 )</th>
<th>( v_0 )</th>
<th>( v_2 )</th>
<th>( v_3 )</th>
<th>( v_4 )</th>
<th>( v_5 )</th>
<th>( v_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 = \nu )</td>
<td>( x_4 = t )</td>
<td>( x_2 = \gamma )</td>
<td>( x_3 = \omega )</td>
<td>( \tan(v_1) )</td>
<td>( v_3 = v_2 - v_2 )</td>
<td>( v_5 = v_4/v_3 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \dot{v}_1 )</th>
<th>( \dot{v}_3 )</th>
<th>( \dot{v}_2 )</th>
<th>( \dot{v}_1 )</th>
<th>( \dot{v}_4 )</th>
<th>( \dot{v}_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \dot{\nu} = v_1 )</td>
<td>( \dot{\nu} = v_3 )</td>
<td>( \dot{\gamma} = v_2 )</td>
<td>( \dot{\omega} = v_0 )</td>
<td>( \dot{\omega} = \dot{v}_3 )</td>
<td>( \dot{\omega} = \dot{v}_5 )</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\nu & = v_1 = x_1 = \nu \\
\gamma & = v_2 = x_2 = \gamma \\
\omega & = v_3 = x_3 = \omega \\
t & = v_4 = x_4 = t \\
\end{align*}
\]

\[
\begin{align*}
\dot{\nu} & = \dot{v}_1 = v_1 \\
\dot{\gamma} & = \dot{v}_2 = v_2 \\
\dot{\omega} & = \dot{v}_3 = v_3 \\
\dot{t} & = \dot{v}_4 = v_4 \\
\end{align*}
\]
## Forward Mode (Lighthouse)

| \( \nu_{-3} \) | \( = x_1 = \nu \) | \( \dot{\nu}_{-3} \) | \( = \dot{x}_1 \) |
|\( \nu_{-2} \) | \( = x_2 = \gamma \) | \( \dot{\nu}_{-2} \) | \( = \dot{x}_2 \) |
|\( \nu_{-1} \) | \( = x_3 = \omega \) | \( \dot{\nu}_{-1} \) | \( = \dot{x}_3 \) |
|\( \nu_0 \) | \( = x_4 = t \) | \( \dot{\nu}_0 \) | \( = \dot{x}_4 \) |

| \( \nu_1 \) | \( = \nu_{-1} \times \nu_0 \) | \( \dot{\nu}_1 \) | \( = \dot{\nu}_{-1} \times \nu_0 + \nu_{-1} \times \dot{\nu}_0 \) |
|\( \nu_2 \) | \( = \tan(\nu_1) \) | \( \dot{\nu}_2 \) | \( = \dot{\nu}_1 / \cos(\nu_1)^2 \) |
|\( \nu_3 \) | \( = \nu_{-2} - \nu_2 \) | \( \dot{\nu}_3 \) | \( = \dot{\nu}_{-2} - \dot{\nu}_2 \) |
|\( \nu_4 \) | \( = \nu_{-3} \times \nu_2 \) | \( \dot{\nu}_4 \) | \( = \dot{\nu}_{-3} \times \nu_2 + \nu_{-3} \times \dot{\nu}_2 \) |
|\( \nu_5 \) | \( = \nu_4 / \nu_3 \) | \( \dot{\nu}_5 \) | \( = (\dot{\nu}_4 - \dot{\nu}_3 \times \nu_5) \times (1 / \nu_3) \) |
|\( \nu_6 \) | \( = \nu_5 \times \nu_{-2} \) | \( \dot{\nu}_6 \) | \( = \dot{\nu}_5 \times \nu_{-2} + \nu_5 \times \dot{\nu}_{-2} \) |

| \( y_1 \) | \( = \nu_5 \) |
|\( y_2 \) | \( = \nu_6 \) |
### Forward Mode (Lighthouse)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
<th>Derivative</th>
<th>Derivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbf{v}_{-3} )</td>
<td>( \mathbf{x}_1 = \mathbf{v} )</td>
<td>( \dot{\mathbf{v}}_{-3} = \dot{\mathbf{x}}_1 )</td>
<td></td>
</tr>
<tr>
<td>( \mathbf{v}_{-2} )</td>
<td>( \mathbf{x}_2 = \gamma )</td>
<td>( \dot{\mathbf{v}}_{-2} = \dot{\mathbf{x}}_2 )</td>
<td></td>
</tr>
<tr>
<td>( \mathbf{v}_{-1} )</td>
<td>( \mathbf{x}_3 = \omega )</td>
<td>( \dot{\mathbf{v}}_{-1} = \dot{\mathbf{x}}_3 )</td>
<td></td>
</tr>
<tr>
<td>( \mathbf{v}_0 )</td>
<td>( \mathbf{x}_4 = t )</td>
<td>( \dot{\mathbf{v}}_0 = \dot{\mathbf{x}}_4 )</td>
<td></td>
</tr>
<tr>
<td>( \mathbf{v}_1 )</td>
<td>( \mathbf{v}_{-1} \ast \mathbf{v}_0 )</td>
<td>( \dot{\mathbf{v}}<em>1 = \dot{\mathbf{v}}</em>{-1} \ast \mathbf{v}<em>0 + \mathbf{v}</em>{-1} \ast \dot{\mathbf{v}}_0 )</td>
<td></td>
</tr>
<tr>
<td>( \mathbf{v}_2 )</td>
<td>( \tan(\mathbf{v}_1) )</td>
<td>( \dot{\mathbf{v}}_2 = \dot{\mathbf{v}}_1 / \cos(\mathbf{v}_1)^2 )</td>
<td></td>
</tr>
<tr>
<td>( \mathbf{v}_3 )</td>
<td>( \mathbf{v}_{-2} - \mathbf{v}_2 )</td>
<td>( \dot{\mathbf{v}}<em>3 = \dot{\mathbf{v}}</em>{-2} - \dot{\mathbf{v}}_2 )</td>
<td></td>
</tr>
<tr>
<td>( \mathbf{v}_4 )</td>
<td>( \mathbf{v}_{-3} \ast \mathbf{v}_2 )</td>
<td>( \dot{\mathbf{v}}<em>4 = \dot{\mathbf{v}}</em>{-3} \ast \mathbf{v}<em>2 + \mathbf{v}</em>{-3} \ast \dot{\mathbf{v}}_2 )</td>
<td></td>
</tr>
<tr>
<td>( \mathbf{v}_5 )</td>
<td>( \mathbf{v}_4 / \mathbf{v}_3 )</td>
<td>( \dot{\mathbf{v}}_5 = (\dot{\mathbf{v}}_4 - \dot{\mathbf{v}}_3 \ast \dot{\mathbf{v}}_5) \ast (1 / \mathbf{v}_3) )</td>
<td></td>
</tr>
<tr>
<td>( \mathbf{v}_6 )</td>
<td>( \mathbf{v}<em>5 \ast \mathbf{v}</em>{-2} )</td>
<td>( \dot{\mathbf{v}}_6 = \dot{\mathbf{v}}<em>5 \ast \mathbf{v}</em>{-2} + \mathbf{v}<em>5 \ast \dot{\mathbf{v}}</em>{-2} )</td>
<td></td>
</tr>
<tr>
<td>( \mathbf{y}_1 )</td>
<td>( \mathbf{v}_5 )</td>
<td>( \dot{\mathbf{y}}_1 = \dot{\mathbf{v}}_5 )</td>
<td></td>
</tr>
<tr>
<td>( \mathbf{y}_2 )</td>
<td>( \mathbf{v}_6 )</td>
<td>( \dot{\mathbf{y}}_2 = \dot{\mathbf{v}}_6 )</td>
<td></td>
</tr>
</tbody>
</table>
Complexity (Forward Mode)

<table>
<thead>
<tr>
<th></th>
<th>tang</th>
<th>$c$</th>
<th>$\pm$</th>
<th>$\ast$</th>
<th>$\psi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVES</td>
<td>1 + 1</td>
<td>3 + 3</td>
<td>3 + 3</td>
<td>2 + 2</td>
<td></td>
</tr>
<tr>
<td>ADDS</td>
<td>0</td>
<td>1 + 1</td>
<td>0 + 1</td>
<td>0 + 0</td>
<td></td>
</tr>
<tr>
<td>MULTS</td>
<td>0</td>
<td>0</td>
<td>1 + 2</td>
<td>0 + 1</td>
<td></td>
</tr>
<tr>
<td>NLOPS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1 + 1</td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{OPS}(F'(x)\dot{x}) \leq c \text{ OPS}(F(x))
\]

with $c \in [2, 5/2]$ platform dependent
Reverse Mode AD = Discrete Adjoint

\[ x \equiv y^{\top} F(x) = \nabla_x \langle y^{\top} F(x) \rangle \equiv \bar{F}(x, \bar{y}) \]
Reverse Mode AD = Discrete Adjoint

\[
x \cdot F \bar{y} = c \\
\bar{x} \equiv \bar{y}^\top F'(x) = \nabla_x \langle \bar{y}^\top F(x) \rangle = \bar{F}(x, \bar{y})
\]
Reverse Mode AD = Discrete Adjoint
Reverse Mode AD = Discrete Adjoints

\[
\bar{x} \equiv \bar{y}^T F'(x) = \nabla_x \langle \bar{y}^T F(x) \rangle \equiv \bar{F}(x, \bar{y})
\]
Reverse Mode (Lighthouse)

\[
\begin{align*}
\nu_{-3} &= x_1; & \nu_{-2} &= x_2; & \nu_{-1} &= x_3; & \nu_0 &= x_4; \\
\nu_1 &= \nu_{-1} \times \nu_0; \\
\nu_2 &= \tan(\nu_1); \\
\nu_3 &= \nu_{-2} - \nu_2; \\
\nu_4 &= \nu_{-3} \times \nu_2; \\
\nu_5 &= \nu_4 / \nu_3; \\
\nu_6 &= \nu_5 \times \nu_{-2}; \\
\nu_1 &= \nu_5; & \nu_2 &= \nu_6; \\
\nu_5 &= \bar{\nu}_1; & \nu_6 &= \bar{\nu}_2; \\
\bar{\nu}_5 &= \nu_6 \times \nu_{-2}; & \bar{\nu}_{-2} &= \nu_6 \times \nu_5; \\
\bar{\nu}_4 &= \nu_5 / \nu_3; & \bar{\nu}_3 &= \nu_5 \times \nu_5 / \nu_3; \\
\bar{\nu}_{-3} &= \nu_4 \times \nu_2; & \bar{\nu}_2 &= \nu_4 \times \nu_{-3}; \\
\bar{\nu}_{-2} &= \nu_3; & \bar{\nu}_2 &= \nu_3; \\
\bar{\nu}_1 &= \nu_2 / \cos^2(\nu_1); \\
\bar{\nu}_{-1} &= \nu_1 \times \nu_0; & \bar{\nu}_0 &= \nu_1 \times \nu_{-1}; \\
\bar{x}_4 &= \nu_0; & \bar{x}_3 &= \nu_{-1}; & \bar{x}_2 &= \nu_{-2}; & \bar{x}_1 &= \nu_{-3};
\end{align*}
\]
Complexity (Reverse Mode)

<table>
<thead>
<tr>
<th>grad</th>
<th>$c$</th>
<th>$\pm$</th>
<th>$\ast$</th>
<th>$\psi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVES</td>
<td>$1+1$</td>
<td>$3+6$</td>
<td>$3+8$</td>
<td>$2+5$</td>
</tr>
<tr>
<td>ADDS</td>
<td>$0$</td>
<td>$1+2$</td>
<td>$0+2$</td>
<td>$0+1$</td>
</tr>
<tr>
<td>MULTS</td>
<td>$0$</td>
<td>$0$</td>
<td>$1+2$</td>
<td>$0+1$</td>
</tr>
<tr>
<td>NLOPS</td>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
<td>$1+1$</td>
</tr>
</tbody>
</table>

\[ \text{OPS}(\bar{y}^T F'(x)) \leq c \text{ OPS}(F(x)) \]
\[ \text{MEM}(\bar{y}^T F'(x)) \sim \text{OPS}(F(x)) \]

with $c \in [3, 4]$ platform dependent
Complexity (Reverse Mode)

<table>
<thead>
<tr>
<th>grad</th>
<th>$c$</th>
<th>$\pm$</th>
<th>$\ast$</th>
<th>$\psi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVES</td>
<td>$1 + 1$</td>
<td>$3 + 6$</td>
<td>$3 + 8$</td>
<td>$2 + 5$</td>
</tr>
<tr>
<td>ADDS</td>
<td>$0$</td>
<td>$1 + 2$</td>
<td>$0 + 2$</td>
<td>$0 + 1$</td>
</tr>
<tr>
<td>MULTS</td>
<td>$0$</td>
<td>$0$</td>
<td>$1 + 2$</td>
<td>$0 + 1$</td>
</tr>
<tr>
<td>NLOPS</td>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
<td>$1 + 1$</td>
</tr>
</tbody>
</table>

$$\text{OPS}(\bar{y}^\top F'(x)) \leq c \ \text{OPS}(F(x))$$

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

with $c \in [3, 4]$ platform dependent

**Remarks:**
- Cost for gradient calculation independent of $n$
- Memory requirement may cause problem! $\Rightarrow$ Checkpointing
Algorithmic Differentiation (AD)

Differentiation of “computer programs” within machine precision

Basic forms:

- **forward mode:** \( \text{OPS}(F'(x)\dot{x}) \leq c_1 \text{OPS}(F), c_1 \in [2, 5/2] \)
- **reverse mode:** \( \text{OPS}(\bar{y}^\top F'(x)) \leq c_2 \text{OPS}(F), c_2 \in [3, 4] \)
  \( \text{MEM}(\bar{y}^\top F'(x)) \sim \text{OPS}(F) \)
- **combination:** \( \text{OPS}(\bar{y}^\top F''(x)\dot{x}) \leq c_3 \text{OPS}(F), c_3 \in [7, 10] \)
Algorithmic Differentiation (AD)

Differentiation of “computer programs” within machine precision

Basic forms:

forward mode: \[ \text{OPS}(F'(x)\dot{x}) \leq c_1 \text{OPS}(F), c_1 \in [2, 5/2] \]

reverse mode: \[ \text{OPS}(\bar{y}^\top F'(x)) \leq c_2 \text{OPS}(F), c_2 \in [3, 4] \]

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

combination: \[ \text{OPS}(\bar{y}^\top F''(x)\dot{x}) \leq c_3 \text{OPS}(F), c_3 \in [7, 10] \]

Tasks:

Derivatives (of any order) sparsity patterns, differentiation of fixpoint iterations and time-stepping procedures . . .
Algorithmic Differentiation (AD)

Differentiation of “computer programs” within machine precision

Basic forms:

forward mode: \( \text{OPS}(\mathbf{F}'(\mathbf{x})\dot{\mathbf{x}}) \leq c_1 \text{OPS}(\mathbf{F}), c_1 \in [2, 5/2] \)

reverse mode: \( \text{OPS}(\bar{\mathbf{y}}^\top \mathbf{F}'(\mathbf{x})) \leq c_2 \text{OPS}(\mathbf{F}), c_2 \in [3, 4] \)

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

combination: \( \text{OPS}(\bar{\mathbf{y}}^\top \mathbf{F}''(\mathbf{x})\dot{\mathbf{x}}) \leq c_3 \text{OPS}(\mathbf{F}), c_3 \in [7, 10] \)

Tasks:

Derivatives (of any order) sparsity patterns, differentiation of fixpoint iterations and time-stepping procedures . . .

Implementations?
Intermediate Conclusions
Intermediate Conclusions

- Evaluation of derivatives with working accuracy
- Efficient evaluation of derivatives of any order
- Numerous tools available, based on source transformation or operator overloading
Intermediate Conclusions

- Evaluation of derivatives with working accuracy
- Efficient evaluation of derivatives of any order
- Numerous tools available, based on source transformation or operator overloading
- Nondifferentiability only on meager set
- Optimal Jacobi accumulation is (NP-)hard
- Full Jacobians/Hessians often not needed or sparse
Intermediate Conclusions

- Evaluation of derivatives with working accuracy
- Efficient evaluation of derivatives of any order
- Numerous tools available, based on source transformation or operator overloading
- Nondifferentiability only on meager set
- Optimal Jacobi accumulation is (NP-)hard
- Full Jacobians/Hessians often not needed or sparse
- see www.autodiff.org
Forward Mode of AD

\[ \dot{y}(t) = \frac{\partial}{\partial t} F(x(t)) = F'(x(t)) \dot{x}(t) \equiv \dot{F}(x, \dot{x}) \]
Evaluation procedure for:

\[ F(x) = \exp(\sin(x^2)) \]

\[ \frac{d}{dt} F(x) = \exp(\sin(x^2)) \cos(x^2) \cdot 2x \cdot \dot{x} \]

| \( v_0 \) = \( x \) | \( \dot{v}_0 = \dot{x} \) |
| \( v_1 \) = \( v_0^2 \) | \( \dot{v}_1 = 2v_0 \cdot \dot{v}_0 \) |
| \( v_2 \) = \( \sin(v_1) \) | \( \dot{v}_2 = \cos(v_1) \cdot \dot{v}_1 \) |
| \( v_3 \) = \( \exp(v_2) \) | \( \dot{v}_3 = \exp(v_2) \cdot \dot{v}_2 \) |

\( y = v_3 \)

\( \dot{y} = \dot{v}_3 = \frac{d}{dt} F(x) \)
**General:** With \( u_i \equiv (v_j)_{j \prec i} \)

\[ v = \varphi(u_i) \quad \Rightarrow \quad \dot{v} = \varphi'(u_i) \dot{u}_i \]

**Elementary Tangents for Intrinsic Functions**

\[ v = \sin(u) \quad \Rightarrow \quad \dot{v} = \cos(u) \ast \dot{u} \]
\[ v = \sqrt{u} \quad \Rightarrow \quad \dot{v} = 0.5 \ast \dot{u} / v \]
\[ v = \tan(u) \quad \Rightarrow \quad \dot{v} = \dot{u} / \cos^2(u) \]

**Elementary Tangents for Arithmetic Operations**

\[ v = u \pm w \quad \Rightarrow \quad \dot{v} = \dot{u} \pm \dot{w} \]
\[ v = u \ast w \quad \Rightarrow \quad \dot{v} = w \ast \dot{u} + u \ast \dot{w} \]
\[ v = v / w \quad \Rightarrow \quad \dot{v} = (\dot{u} - v \ast \dot{w}) / w \]
Forward Differentiation of Lighthouse Example

\[
\begin{align*}
\nu_{-3} &= x_1 = \nu & \dot{\nu}_{-3} &= \dot{x}_1 \\
\nu_{-2} &= x_2 = \gamma & \dot{\nu}_{-2} &= \dot{x}_2 \\
\nu_{-1} &= x_3 = \omega & \dot{\nu}_{-1} &= \dot{x}_3 \\
\nu_0 &= x_4 = t & \dot{\nu}_0 &= \dot{x}_4 \\
\nu_1 &= \nu_{-1} \ast \nu_0 \\
\nu_2 &= \tan(\nu_1) \\
\nu_3 &= \nu_{-2} - \nu_2 \\
\nu_4 &= \nu_{-3} \ast \nu_2 \\
\nu_5 &= \nu_4 / \nu_3 \\
\nu_6 &= \nu_5 \\
\nu_7 &= \nu_5 \ast \nu_{-2} \\
y_1 &= \nu_6 \\
y_2 &= \nu_7
\end{align*}
\]
## Forward Differentiation of Lighthouse Example

| \( v_{-3} \) | \( = \) | \( x_1 = \nu \) | \( \dot{v}_{-3} \) | \( \equiv \) | \( \dot{x}_1 \) |
| \( v_{-2} \) | \( = \) | \( x_2 = \gamma \) | \( \dot{v}_{-2} \) | \( \equiv \) | \( \dot{x}_2 \) |
| \( v_{-1} \) | \( = \) | \( x_3 = \omega \) | \( \dot{v}_{-1} \) | \( \equiv \) | \( \dot{x}_3 \) |
| \( v_0 \) | \( = \) | \( x_4 = t \) | \( \dot{v}_0 \) | \( \equiv \) | \( \dot{x}_4 \) |
| \( v_1 \) | \( = \) | \( v_{-1} \ast v_0 \) | \( \dot{v}_1 \) | \( = \) | \( \dot{v}_{-1} \ast v_0 + v_{-1} \ast \dot{v}_0 \) |
| \( v_2 \) | \( = \) | \( \tan(v_1) \) | \( \dot{v}_2 \) | \( = \) | \( \dot{v}_1 / \cos(v_1)^2 \) |
| \( v_3 \) | \( = \) | \( v_{-2} - v_2 \) | \( \dot{v}_3 \) | \( = \) | \( \dot{v}_{-2} - \dot{v}_2 \) |
| \( v_4 \) | \( = \) | \( v_{-3} \ast v_2 \) | \( \dot{v}_4 \) | \( = \) | \( v_{-3} \ast v_2 + v_{-3} \ast \dot{v}_2 \) |
| \( v_5 \) | \( = \) | \( v_{4}/v_3 \) | \( \dot{v}_5 \) | \( = \) | \( (\dot{v}_4 - \dot{v}_3 \ast v_5) \ast (1/v_3) \) |
| \( v_6 \) | \( = \) | \( v_5 \) | \( \dot{v}_6 \) | \( = \) | \( \dot{v}_5 \) |
| \( v_7 \) | \( = \) | \( v_{5} \ast v_{-2} \) | \( \dot{v}_7 \) | \( = \) | \( \dot{v}_5 \ast v_{-2} + v_5 \ast \dot{v}_{-2} \) |

\[ y_1 = v_6 \]

\[ y_2 = v_7 \]
Forward Differentiation of Lighthouse Example

| \( v_{-3} \) | \( x_1 = \nu \) | \( \dot{v}_{-3} \) | \( \dot{x}_1 \) |
| \( v_{-2} \) | \( x_2 = \gamma \) | \( \dot{v}_{-2} \) | \( \dot{x}_2 \) |
| \( v_{-1} \) | \( x_3 = \omega \) | \( \dot{v}_{-1} \) | \( \dot{x}_3 \) |
| \( v_0 \) | \( x_4 = t \) | \( \dot{v}_0 \) | \( \dot{x}_4 \) |

\[
\begin{align*}
\dot{v}_1 &= v_{-1} \ast v_0 & \dot{v}_1 &= \dot{v}_{-1} \ast v_0 + \dot{v}_{-1} \ast \dot{v}_0 \\
\dot{v}_2 &= \tan(v_1) & \dot{v}_2 &= \dot{v}_1 / \cos(v_1)^2 \\
\dot{v}_3 &= v_{-2} - v_2 & \dot{v}_3 &= \dot{v}_{-2} - \dot{v}_2 \\
\dot{v}_4 &= v_{-3} \ast v_2 & \dot{v}_4 &= \dot{v}_{-3} \ast v_2 + \dot{v}_{-3} \ast \dot{v}_2 \\
\dot{v}_5 &= v_4 / v_3 & \dot{v}_5 &= \left(\dot{v}_4 - \dot{v}_3 \ast v_5\right) \ast \left(1 / v_3\right) \\
\dot{v}_6 &= v_5 & \dot{v}_6 &= \dot{v}_5 \\
\dot{v}_7 &= v_5 \ast v_{-2} & \dot{v}_7 &= \dot{v}_5 \ast v_{-2} + \dot{v}_5 \ast \dot{v}_{-2} \\
\end{align*}
\]

| \( y_1 \) | \( v_6 \) | \( \dot{y}_1 \) | \( \dot{v}_6 \) |
| \( y_2 \) | \( v_7 \) | \( \dot{y}_2 \) | \( \dot{v}_7 \) |
Define

\[ \dot{\varphi}_i(u_i, \dot{u}_i) \equiv \varphi'_i(u_i) \dot{u}_i \]

\[ \Rightarrow \]

**General Tangent Procedure**

\[
\begin{align*}
[\mathbf{v}_{i-n}, \mathbf{\dot{v}}_{i-n}] & = [\mathbf{x}_i, \mathbf{\dot{x}}_i] & i = 1, \ldots, n \\
[\mathbf{v}_i, \mathbf{\dot{v}}_i] & = [\varphi_i(u_i), \dot{\varphi}_i(u_i)] & i = 1, \ldots, \ell \\
[\mathbf{y}_{m-i}, \mathbf{\dot{y}}_{m-i}] & = [\mathbf{v}_{l-i}, \mathbf{\dot{v}}_{l-i}] & i = m-1, \ldots, 0
\end{align*}
\]
Complexity:

\[
\begin{array}{cccc}
\text{tang} & c & \pm & \times & \psi \\
\text{MOVES} & 1 + 1 & 3 + 3 & 3 + 3 & 2 + 2 \\
\text{ADDS} & 0 & 1 + 1 & 0 + 1 & 0 + 0 \\
\text{MULTS} & 0 & 0 & 1 + 2 & 0 + 1 \\
\text{NLOPS} & 0 & 0 & 0 & 1 + 1 \\
\end{array}
\]

\[
\text{OPS}(F'(x)\dot{x}) \leq c \text{ OPS}(F(x))
\]

with \( c \in [2, 5/2] \) platform dependent.
Complexity:

<table>
<thead>
<tr>
<th></th>
<th>tang</th>
<th>c</th>
<th>±</th>
<th>*</th>
<th>ψ</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVES</td>
<td>1 + 1</td>
<td>3 + 3</td>
<td>3 + 3</td>
<td>2 + 2</td>
<td></td>
</tr>
<tr>
<td>ADDS</td>
<td>0</td>
<td>1 + 1</td>
<td>0 + 1</td>
<td>0 + 0</td>
<td></td>
</tr>
<tr>
<td>MULTS</td>
<td>0</td>
<td>0</td>
<td>1 + 2</td>
<td>0 + 1</td>
<td></td>
</tr>
<tr>
<td>NLOPS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1 + 1</td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{OPS}(F'(x)\dot{x}) \leq c \text{ OPS}(F(x))
\]

with \(c \in [2, 5/2]\) platform dependent

Remarks:

- How to arrange evaluation of \([v_i, \dot{v}_i]\) optimal?
- Overwrites are no problem
Reverse Mode of AD

\[ F(x) = \nabla_x \langle \bar{y}^\top F(x) \rangle \equiv \bar{F}(x, \bar{y}) \]
Reverse Mode of AD

\[ \bar{y}^\top y = c \]

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

\[ F \]

\[ \bar{x} \]
Reverse Mode of AD

\[ y^T F(x) = c \]

\[ \bar{x} \]

\[ \bar{F} \]

\[ F \]

\[ y^T y = c \]

\[ \bar{y} \]
Reverse Mode of AD

\[ \bar{x}^\top \equiv \bar{y}^\top F'(x) = \nabla_x \langle \bar{y}^\top F(x) \rangle \equiv \bar{F}(x, \bar{y}) \]
Evaluation Procedure:

\[
F(x) = \exp(\sin(x^2))
\]

\[
\bar{y}F'(x) = \bar{y} \ast \exp(\sin(x^2)) \ast \cos(x^2) \ast 2x
\]

\[
\begin{align*}
\nu_0 &= x & \bar{\nu}_3 &= \bar{y} \\
\nu_1 &= \nu_0^2 & \bar{\nu}_2 &= \exp(\nu_2) \ast \bar{\nu}_3 \\
\nu_2 &= \sin(\nu_1) & \bar{\nu}_1 &= \cos(\nu_1) \ast \bar{\nu}_2 \\
\nu_3 &= \exp(\nu_2) & \bar{\nu}_0 &= 2\nu_0 \ast \bar{\nu}_1 \\
y &= \nu_3 & \bar{x} &= \bar{\nu}_0
\end{align*}
\]
Computation of $\bar{y} F'(x)$?

Rewrite general tangent procedure

\[
[v_{i-n}, \dot{v}_{i-n}] = [x_i, \dot{x}_i] \quad i = 1, \ldots, n
\]

\[
[v_i, \dot{v}_i] = [\varphi_i(u_i), \dot{\varphi}_i(u_i)] \quad i = 1, \ldots, l
\]

\[
[y_{m-i}, \dot{y}_{m-i}] = [v_{l-i}, \dot{v}_{l-i}] \quad i = m - 1, \ldots, 0
\]

in matrix-vector notation.
Define

$$c_{ij} \equiv \frac{\partial \varphi_i}{\partial v_j}$$

$$A_i \equiv \begin{bmatrix}
1 & 0 & \cdots & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1 & \cdots & 0 \\
c_{i1-n} & c_{i2-n} & \cdots & c_{i,i-1} & 0 & \cdots & 0 \\
0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \cdots & \cdots & \cdots & \cdots & 1
\end{bmatrix} \in \mathbb{R}^{(n+l) \times (n+l)}$$

$$= l + e_{n+i} [\nabla \varphi_i(u_i) - e_{n+i}]^T \quad i = 1, \ldots, l$$

$$P_n \equiv [l, 0, \ldots, 0] \in \mathbb{R}^{n \times (n+l)}$$,

$$Q_m \equiv [0, \ldots, l] \in \mathbb{R}^{m \times (n+l)}$$.
⇒ Tangent Operation $\dot{v}_i = \varphi_i'(u_i)\dot{u}_i$ equals $\dot{v} = A_i\dot{v}$

⇒ Linear transformation $\dot{y} = F'(x) \dot{x}$ equals

$$\dot{y} = Q_mA_lA_{l-1} \ldots A_2A_1P_n^T \dot{x},$$

with the product representation

$$F'(x) = Q_mA_lA_{l-1} \ldots A_2A_1P_n^T \in \mathbb{R}^{m \times n}.$$ 

⇒ linear transformation $\bar{x} = \bar{y} F'(x)$ (reverse mode) equals

$$\bar{x}^T = P_nA_1^TA_2^T \ldots A_{l-1}^TA_l^TQ_m^T\bar{y}^T,$$
Because of

\[ A_i^T = I + [\nabla \varphi_i(u_i) - e_{n+i}] e_{n+i}^T, \]

product \( A_i^T \bar{v}_j \) with \( \bar{v}_j \in \mathbb{R}^{n+1} \) forms incremental operation:

1. Index \( i \neq j \neq i \) \( \Rightarrow \) \( \bar{v}_j \) is left unchanged
2. Index \( j < i \) \( \Rightarrow \) \( \bar{v}_j \) is incremented by \( \bar{v}_i c_{ij} \)
3. Index \( i \) \( \Rightarrow \) \( \bar{v}_i \) is set to zero

Hence, for computing \( \bar{x} = \bar{y} F'(x) \) one has to

- evaluate all intermediate \( \nu_i \) obtaining partial derivatives \( c_{ij} \)
- increment adjoint values \( \bar{v}_i \) in reverse order.
Corresponding Incremental Adjoint Recursion

\[
\begin{array}{c|c|c}
\bar{V}_i & 0 & i = 1 - n, \ldots, l \\
V_{i-n} & x_i & i = 1, \ldots, n \\
V_i & \varphi_i(V_j)_{j \prec i} & i = 1, \ldots, l \\
Y_{m-i} & V_{l-i} & i = m - 1, \ldots, 0 \\
\bar{V}_{l-i} & \bar{Y}_{m-i} & i = 0, \ldots, m - 1 \\
\bar{V}_j & + & \bar{V}_i \frac{\partial}{\partial V_j} \varphi_i(U_i) \text{ for } j \prec i \quad i = l, \ldots, 1 \\
\bar{x}_i & = & \bar{V}_{i-n} \quad i = n, \ldots, 1 \\
\end{array}
\]

No overwriting ⇒ Zeroing out of the \( \bar{V}_i \) is omitted.
Adjoint Recursion of Lighthouse-Example

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

\[ \bar{v}_5 = \bar{y}_1; \quad \bar{v}_6 = \bar{y}_2; \]
\[ \bar{v}_5 \mathbin{+}= \bar{v}_6 \ast \bar{v}_{-2}; \quad \bar{v}_{-2} \mathbin{+}= \bar{v}_6 \ast \bar{v}_5; \]
\[ \bar{v}_4 \mathbin{+}= \bar{v}_5 / \bar{v}_3; \quad \bar{v}_3 \mathbin{+}= \bar{v}_5 \ast \bar{v}_5 / \bar{v}_3; \]
\[ \bar{v}_{-3} \mathbin{+}= \bar{v}_4 \ast \bar{v}_2; \quad \bar{v}_2 \mathbin{+}= \bar{v}_4 \ast \bar{v}_{-3}; \]
\[ \bar{v}_{-2} \mathbin{+}= \bar{v}_3; \quad \bar{v}_2 \mathbin{-}= \bar{v}_3; \]
\[ \bar{v}_1 \mathbin{+}= \bar{v}_2 / \cos^2(v_1); \]
\[ \bar{v}_{-1} \mathbin{+}= \bar{v}_1 \ast v_0; \quad \bar{v}_0 \mathbin{+}= \bar{v}_1 \ast v_{-1}; \]
\[ \bar{x}_4 = \bar{v}_0; \quad \bar{x}_3 = \bar{v}_{-1}; \quad \bar{x}_2 = \bar{v}_{-2}; \quad \bar{x}_1 = \bar{v}_{-3}; \]
## Reverse Operations of Elementals

<table>
<thead>
<tr>
<th>Elemental</th>
<th>Reverse</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \nu = c )</td>
<td>( \ddot{\nu} = 0 )</td>
</tr>
<tr>
<td>( \nu = u \pm w )</td>
<td>( \ddot{u} \pm= \ddot{\nu}; \quad \ddot{w} \pm= \ddot{\nu}; \quad \ddot{\nu} = 0 )</td>
</tr>
<tr>
<td>( \nu = u \times w )</td>
<td>( \ddot{u} \times= \ddot{\nu} \times w; \quad \ddot{w} \times= \ddot{\nu} \times u; \quad \ddot{\nu} = 0 )</td>
</tr>
<tr>
<td>( \nu = 1/u )</td>
<td>( \ddot{u} \times= (\ddot{\nu} \times v) \times v; \quad \ddot{\nu} = 0 )</td>
</tr>
<tr>
<td>( \nu = \sqrt{u} )</td>
<td>( \ddot{u} \times= 0.5 \times \ddot{v} / v; \quad \ddot{\nu} = 0 )</td>
</tr>
<tr>
<td>( \nu = u^c )</td>
<td>( \ddot{u} \times= (\ddot{\nu} \times c) \times v / u; \quad \ddot{\nu} = 0 )</td>
</tr>
<tr>
<td>( \nu = \exp{(u)} )</td>
<td>( \ddot{u} \times= \ddot{\nu} \times v; \quad \ddot{\nu} = 0 )</td>
</tr>
<tr>
<td>( \nu = \log{(u)} )</td>
<td>( \ddot{u} \times= \ddot{\nu} / u; \quad \ddot{\nu} = 0 )</td>
</tr>
<tr>
<td>( \nu = \sin{(u)} )</td>
<td>( \ddot{u} \times= \ddot{\nu} \times \cos{(u)}; \quad \ddot{\nu} = 0 )</td>
</tr>
</tbody>
</table>
### Complexity:

<table>
<thead>
<tr>
<th></th>
<th>grad</th>
<th>c</th>
<th>±</th>
<th>*</th>
<th>ψ</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVES</td>
<td>1 + 1</td>
<td>3 + 6</td>
<td>3 + 8</td>
<td>2 + 5</td>
<td></td>
</tr>
<tr>
<td>ADDS</td>
<td>0</td>
<td>1 + 2</td>
<td>0 + 2</td>
<td>0 + 1</td>
<td></td>
</tr>
<tr>
<td>MULTS</td>
<td>0</td>
<td>0</td>
<td>1 + 2</td>
<td>0 + 1</td>
<td></td>
</tr>
<tr>
<td>NLOPS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1 + 1</td>
<td></td>
</tr>
</tbody>
</table>

\[ \Rightarrow \quad \text{OPS}(\bar{y}F'(x)) \leq c \text{ OPS}(F(x)) \]

with \( c \in [3, 5] \) platform dependent

**Remark:**

- Overwrites cause problem! See Checkpointing.
Interpretation of adjoint values $\tilde{v}_i$:

- Lagrange multipliers of

\[
\begin{align*}
\text{Min } & \quad \tilde{y}y \\
\text{s.t. } & \quad \varphi(u_i) - v_i = 0 \quad \forall i = 1, \ldots, l \\
& \quad v_{l-i} - y_{m-i} = 0 \quad \forall i = 1, \ldots, m
\end{align*}
\]

- Error propagation factors

\[
\begin{align*}
\tilde{v}_i &= v_i(1 + \varepsilon_i), \quad |\varepsilon_i| \leq \varepsilon \equiv \text{macheps} \\
\tilde{y} \tilde{y} &= \sum_{i=1}^{l} \tilde{v}_i v_i \varepsilon_i + \text{H.O.T} \\
&\leq \varepsilon \sum_{i=1}^{l} |\tilde{v}_i v_i| + \text{H.O.T}
\end{align*}
\]
Vector Modes

How to reduce overhead in forward and reverse mode?

Propagate a bundle of vectors instead of a single vector!

Forward Mode:

Compute \( \dot{Y} = F'(x)\dot{X} \in \mathbb{R}^{m \times p} \) for \( \dot{X} \in \mathbb{R}^{n \times p} \)

instead of

\( \dot{y} = F'(x)\dot{x} \in \mathbb{R}^{m} \) for \( \dot{x} \in \mathbb{R}^{n} \)

- Replace \( \dot{v}_j \in \mathbb{R} \) by \( \dot{V}_j \in \mathbb{R}^{p} \) in tangent procedure
- Replace \( \dot{z} \in \mathbb{R}^{(l-m)} \) by \( \dot{Z} \in \mathbb{R}^{(l-m) \times p} \)
- Everything else remains unchanged
Vector Tangents:

\[ v = u \pm w \Rightarrow \dot{V} = \dot{U} \pm \dot{W} \]
\[ v = u \ast w \Rightarrow \dot{V} = w \ast \dot{U} + u \ast \dot{W} \]
\[ v = \sin(u) \Rightarrow \dot{V} = \cos(u) \ast \dot{U} \]

\[ \ldots \]

Complexity:

<table>
<thead>
<tr>
<th></th>
<th>( c )</th>
<th>( \pm )</th>
<th>( \ast )</th>
<th>( \psi )</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVES</td>
<td>( 1 + p )</td>
<td>( 3 + 3p )</td>
<td>( 3 + 3p )</td>
<td>( 2 + 2p )</td>
</tr>
<tr>
<td>ADDS</td>
<td>( 0 )</td>
<td>( 1 + p )</td>
<td>( 0 + p )</td>
<td>( 0 + 0 )</td>
</tr>
<tr>
<td>MULTS</td>
<td>( 0 )</td>
<td>( 0 )</td>
<td>( 1 + 2p )</td>
<td>( 0 + p )</td>
</tr>
<tr>
<td>NLOPS</td>
<td>( 0 )</td>
<td>( 0 )</td>
<td>( 0 )</td>
<td>( 2 )</td>
</tr>
</tbody>
</table>

\[ \Rightarrow \text{OPS}(F'(x) \dot{X}) \leq c \text{ OPS}(F(x)) \]

with \( c \in [1 + p, 1 + 1.5p] \)
Applications:

- Computation of full Jacobian
  Set $X = I_n$ ⇒

  $\text{OPS}(F'(x)) \leq (1 + 1.5n)\text{OPS}(F(x))$

  Compare with scalar mode, i.e. $F'(x)e_i, i = 1, \ldots, n$:

  $\text{OP}(f'(x)) \leq 2.5n\text{OPS}(F(x))$

  ⇒ Remarkable run-time savings can be obtained

- Computation of sparse Jacobians
Reverse mode:

Compute $\tilde{X} = \tilde{Y}F'(x) \in \mathbb{R}^{q \times n}$ for $\tilde{Y} \in \mathbb{R}^{q \times m}$ instead of

$$\tilde{x} = \tilde{y}F'(x) \in \mathbb{R}^{n} \text{ for } \tilde{y} \in \mathbb{R}^{m}$$

Implementation:

- Replace $\tilde{v}_i \in \mathbb{R}$ by $\tilde{V}_i \in \mathbb{R}^q$ in adjoint recursion
- Everything else remains unchanged
Complexity:

<table>
<thead>
<tr>
<th>gradq</th>
<th>c</th>
<th>±</th>
<th>*</th>
<th>ψ</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVES</td>
<td>1 + q</td>
<td>3 + 6q</td>
<td>5 + 6q</td>
<td>3 + 4q</td>
</tr>
<tr>
<td>ADDS</td>
<td>0</td>
<td>1 + 2q</td>
<td>0 + 2q</td>
<td>0 + q</td>
</tr>
<tr>
<td>MULTS</td>
<td>0</td>
<td>0</td>
<td>1 + 2q</td>
<td>0 + q</td>
</tr>
<tr>
<td>NLOPS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

⇒ \( \text{OPS}(\bar{Y}F'(x)) \leq k \text{OPS}(F(x)) \)

with \( c \in [1 + 2q, 1.5 + 2.5q] \)

Remarks:
- Computation of full Jacobian
- Remarkable run-time savings can be obtained
- Application: e.g. matrix compression for sparse Jacobians
General Evaluation Procedure:

\[
\begin{align*}
v_{i-n} &= x_i & \text{for } i = 1, \ldots, n \\
v_i &= \varphi_i(v_j)_{j \prec i} & \text{for } i = 1, \ldots, l \\
y_{m-i} &= v_{l-i} & \text{for } i = m - 1, \ldots, 0
\end{align*}
\]

where

- \( v_i, i \leq 0 \), are the independents
- \( v_i, i \geq l - m + 1 \), are the dependents
- \( \varphi_i \in \Phi \) is elemental function, \( \Phi = \text{set of elemental functions} \)
- \( j \prec i \) is dependence relation: \( v_i \) depends directly on \( v_j \)
<table>
<thead>
<tr>
<th></th>
<th>Essential</th>
<th>Optional</th>
<th>Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Φ</strong></td>
<td><strong>Essential</strong></td>
<td><strong>Optional</strong></td>
<td><strong>Vector</strong></td>
</tr>
<tr>
<td>Smooth</td>
<td>$u + v$, $u \star v$</td>
<td>$u - v$, $u/v$</td>
<td>$\sum_k u_k \star v_k$</td>
</tr>
<tr>
<td></td>
<td>$-u$, $c$</td>
<td>$c \star u$, $c \pm u$</td>
<td>$\sum_k c_k \star u_k$</td>
</tr>
<tr>
<td></td>
<td>$\text{rec}(u) \equiv 1/u$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\exp(u)$, $\log(u)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\sin(u)$, $\cos(u)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$</td>
<td>u</td>
<td>^c$, $c &gt; 1$</td>
</tr>
<tr>
<td>Lipschitz</td>
<td>$\text{abs}(u) \equiv</td>
<td>u</td>
<td>$</td>
</tr>
<tr>
<td></td>
<td>$</td>
<td>u, v</td>
<td>\equiv \sqrt{u^2 + v^2}$</td>
</tr>
<tr>
<td>General</td>
<td>$\text{heav}(u)$</td>
<td>$\text{sign}(u)$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(u &gt; 0)?v, w$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Representation as computational graph

Computational graph of lighthouse example
Representation as extended system $E(x; v) = 0$

- Define

$$u_i \equiv (v_j)_{j<i} \quad \text{for } i = 1 - n, \ldots, l$$
$$\varphi_i(u_i) \equiv x_{i+n} \quad \text{for } i = 1 - n, \ldots, 0$$
$$c_{ij} \equiv \frac{\partial \varphi_i}{\partial v_j} \quad \text{called elemental partials}$$

- Assume dependents variables as mutually independent

Then one has that

- $i < 1$ or $j > l - m$ implies $c_{ij} \equiv 0$
- Evaluation procedure equivalent to nonlinear system

$$0 = E(x; v) \equiv (\varphi_i(u_i) - v_i)_{i=1-n,\ldots,l}$$
Topics to keep in mind:

- **Composite Differentiation (overall differentiability):** Here it is assumed that all $\varphi_i$ are differentiable at respective argument.

- **Overwriting $=\text{Shared Allocation of } v_i \text{ and } v_j$**
  Not considered here, but:
  There are ways to handle them.

- **Aliasing of Arguments and Results:**
  Not considered here, but:
  Again there are ways to handle them.
Main Results:

- Gradients are cheap $\sim$ function costs
- Costs grow only quadratically in degree
Main Results:

- Gradients are cheap \( \sim \) function costs
- Costs grow only quadratically in degree
- Nondifferentiability only on meager set
Main Results:

- Gradients are cheap \( \sim \) function costs
- Costs grow only quadratically in degree
- Nondifferentiability only on meager set
- Optimal Jacobi-accumulation is (NP-)hard
Main Results:

- Gradients are cheap \(\sim\) function costs
- Costs grow only quadratically in degree
- Nondifferentiability only on meager set
- Optimal Jacobi-accumulation is (NP-)hard
- Jacobi-accumulation is often unnecessary
Higher Order Derivatives

Second derivatives:

Use forward and reverse differentiation together!

\[
y = F(x) \quad \text{reverse differentiation} \quad \bar{x} = \bar{y}F'(x)
\]

\[
\bar{x} = \bar{y}F'(x) \quad \text{forward differentiation} \quad \dot{x} = \bar{y}F''(x)\dot{x} + \dot{y}F'(x)
\]

For scalar-valued function \( F \), \( \bar{y} = 1 \), \( \dot{y} = 0 \), and \( \dot{x} \) one has

\[
\dot{x} = F''(x)\dot{x} = \text{Hessian-vector product}
\]
Implementation

Sourcecode

\[ y = F(x) \in \mathbb{R}^m \]
Implementation

Sourcecode

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

Object Code

\[
\begin{align*}
    y &= F(x) \\
    \dot{y} &= F'(x) \dot{x} \in \mathbb{R}^m \\
    (x, \dot{x}) &\in \mathbb{R}^{2n}
\end{align*}
\]
Implementation

Sourcecode

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

Object Code

\[ y = F(x) \]
\[ \dot{y} = F'(x) \dot{x} \in \mathbb{R}^m \]

Intermediate values

Stack (Tape)

\[ (x, \dot{x}) \in \mathbb{R}^{2n} \]

\[ (x, \bar{y}) \in \mathbb{R}^{n+m} \]

\[ \bar{x} = \bar{y}F'(x) \in \mathbb{R}^n \]
Implementation

Sourcecode

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

Intermediate values

Stack (Tape)

Object Code

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

$\bar{y} = \bar{y}F'(x) \in \mathbb{R}^n$

$\dot{\bar{y}} = \bar{y}F''(x) \dot{x} \in \mathbb{R}^n$

$\bar{x} = (x, \bar{y}) \in \mathbb{R}^{n+m}$

$\bar{x} = (x, \bar{y}) \in \mathbb{R}^{n+m}$

$\partial -$ forward

$\partial -$ reverse

$(x, \dot{x}) \in \mathbb{R}^{2n}$
Higher derivatives: Think in Taylor polynomials
Let $x$ be the path

$$x(t) \equiv \sum_{j=0}^{d} x_j t^j : \mathbb{R} \mapsto \mathbb{R} \quad \text{with} \quad x_j = \frac{1}{j!} \frac{\partial^j}{\partial t^j} x(t) \bigg|_{t=0}$$

Hence

- $x_j$ is scaled derivative at $t = 0$.
- $x_1 = \text{tangent}, x_2 = \text{curvature}$.

Now:
Consider $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$, $d$ times continuously differentiable

$$y(t) \equiv F(x(t)) : \mathbb{R} \mapsto \mathbb{R}^m \quad \text{is smooth (chain-rule)}$$
There exist Taylor coefficients $y_j \in \mathbb{R}^m$, $0 \leq j \leq d$, with

$$y(t) = \sum_{j=0}^{d} y_j t^j + O(t^{d+1}).$$

Using the chain-rule, one obtains:

$$y_0 = F(x_0)$$
$$y_1 = F'(x_0)x_1$$
$$y_2 = F'(x_0)x_2 + \frac{1}{2} F''(x_0)x_1x_1$$
$$y_3 = F'(x_0)x_3 + F''(x_0)x_1x_2 + \frac{1}{6} F'''(x_0)x_1x_1x_1$$

\[ \cdots \]

⇒ directional derivatives of high order (forward mode)

$$\text{OPS}(x_0, \ldots, x_d \rightarrow y_0, \ldots, y_d) \sim d^2 \text{OPS}(x_0 \rightarrow y_0)$$

using Taylor arithmetic
Cross Country / Preaccumulation

Consider again Jacobian of the extended system $E(x; v)$:

$$E'(x; v) = \begin{bmatrix} -I & 0 & 0 \\ B & L - I & 0 \\ R & T & -1 \end{bmatrix}$$

with $F'(x) = R + T(I - L)^{-1}B = \text{Schur complement}$

- Forward mode: Row-Block-Elimination of $T$ by $L - I$
- Reverse mode: Column-Block-Elimination of $B$ by $L - I$
- Cross country: Apply other elimination order
Another Interpretation

Elimination in Jacobian of extended system =
Vertex elimination in computational graph:

\[
\begin{align*}
4 \ast & \rightarrow 1 + \\
1 \ast & \rightarrow 4 +
\end{align*}
\]
Another Interpretation

Elimination in Jacobian of extended system =
Vertex elimination in computational graph:

Example for forward and reverse mode:
AD by Source Transformation

Code for Simulation ⇒ Code for Derivatives
AD by Source Transformation

Code for Simulation ⇒ Code for Derivatives

func.src ⇒
1. Lexical analysis
2. Syntax analysis
3. Semantic analysis
4. Static data flow analyses (Symbol table, Error handler)
5. AD !!!
6. Code optimization
7. Unparsing

⇒ func&der.drc
Remarks

Pros:
- Make use of compiler optimization !!
- Derivative code is "readable"

Cons:
- Requires complete compiler
  ⇒ activity analysis, new language features
- Flexibility
Speelpenning’s Example

Speelpenning’s function $f(x) = \prod_{i=1}^{n} x_i$ coded in Fortran:

```fortran
SUBROUTINE speelpenning(x, y)
DOUBLE PRECISION x(10), y
INTEGER i

y = 1.0
DO i = 1, 10
    y = y * x(i)
END DO
END
```
Forward Mode Differentiation

Requirements for AD tool:

▶ determine active variables
▶ associate derivative objects
▶ generate differentiated statements
▶ generate differentiated subroutines if required
Forward Mode Differentiation

```
SUBROUTINE SPEELPENNING_D(x, xd, y, yd
DOUBLE PRECISION x(10), xd(10), y, yd
INTEGER i

y=1.0
yd=0.0
DO i=1,10
  y=yd=yd*x(i)+y*xd(i)
  y=y*x(i)
END DO
END
```
Reverse Mode Differentiation

Requirements for AD tool:

- determine active variables
- associate derivative objects
- generate adjoint statements
- determine values to be recorded
- reverse control flow
- generate ”recording” part of the code
Reverse Mode Differentiation

SUBROUTINE SPEELPENNING_B(x, xb, y, yb)
DOUBLE PRECISION x(10), xb(10), y, yb
INTEGER i

y = 1.0
DO i = 1, 10
   CALL PUSHREAL8(y)
   y = y * x(i)
END DO
CALL PUSHINTEGER4(i - 1)
CALL POPINTEGER4(adTo)
DO i = adTo, 1, -1
   CALL POPREAL8(y)
   xb(i) = xb(i) + y * yb
   yb = x(i) * yb
END DO
yb = 0.0
END
Source Transformation Tools

**Fortran:**
- **AMPL**: Bell Lab, USA, forward + reverse
- **NAG**: RWTH Aachen, Germany + NAG, forward + reverse
- **TAF**: FastOpt, Germany, forward + reverse
- **Tapenade**: INRIA, France, forward + reverse
- **OpenAD**: ANL, USA, RWTH Aachen, Germany, forward

All tools provide at most second derivatives!
Goal: Get AD into the compiler
Source Transformation Tools

**Fortran:**

<table>
<thead>
<tr>
<th>Tool</th>
<th>Location</th>
<th>Derivative Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMPL</td>
<td>Bell Lab, USA</td>
<td>forward + reverse</td>
</tr>
<tr>
<td>NAG</td>
<td>RWTH Aachen, Germany + NAG</td>
<td>forward + reverse</td>
</tr>
<tr>
<td>TAF</td>
<td>FastOpt, Germany</td>
<td>forward + reverse</td>
</tr>
<tr>
<td>Tapenade</td>
<td>INRIA, France</td>
<td>forward + reverse</td>
</tr>
<tr>
<td>OpenAD</td>
<td>ANL, USA, RWTH Aachen, Germany</td>
<td>forward</td>
</tr>
</tbody>
</table>

... 

All tools provide at most second derivatives!

Goal: Get AD into the compiler

**Matlab:**

<table>
<thead>
<tr>
<th>Tool</th>
<th>Location</th>
<th>Derivative Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADIMAT</td>
<td>RWTH Aachen, Germany</td>
<td>forward</td>
</tr>
<tr>
<td>MAD (in TOMS)</td>
<td>Univ. Shrivenham, GB</td>
<td>forward</td>
</tr>
</tbody>
</table>

...
AD by Operator Overloading

Implementation strategies:

▶ New variable type for active variables, i.e., for independents, dependents, intermediates (!)
▶ Derivative information included as
  ▶ field in composite structures
  ▶ generation of internal function representation

Principle: New class definition

```csharp
type ad_double = class
    VAL : double;
    ???
end;
```
AD by Field in Composite Structure

Idea for forward mode:

type  ad_double = class
       VAL : double;
       DER : array[1..p] of double;
end;

All operations are overloaded, e.g., multiplication:
operator *(U,V : ad_double) W : ad_double;
i : integer;
begin
   W.VAL := U.VAL * V.VAL;
   for i := 1 to p do
end;
AD by Field in Composite Structure

Idea for forward mode:

define type ad_double = class
  VAL : double;
  DER : array[1..p] of double;
end;

All operations are overloaded, e.g., multiplication:

operator * (U,V : ad_double) W : ad_double;
  i : integer;
begin
  W.VAL := U.VAL * V.VAL;
  for i := 1 to p do
    W.DER[i] := U.VAL*V.DER[i] + U.DER[i]*V.VAL;
end;
Remarks:

- Implements basic rules of differentiations for
  - each arithmetic operation
  - each intrinsic function
- "Easy" to realize
- tapeless forward mode of ADOL-C
Remarks:

▶ Implements basic rules of differentiations for
  ▶ each arithmetic operation
  ▶ each intrinsic function
▶ “Easy” to realize
tapeless forward mode of ADOL-C

Idea for reverse mode: Graph in “core”
Need of backward pointer for calculating adjoints ⇒ new node type:

.. .

<table>
<thead>
<tr>
<th>VAL</th>
<th>IND</th>
</tr>
</thead>
<tbody>
<tr>
<td>DER</td>
<td>ADJ</td>
</tr>
</tbody>
</table>
\[ W := U \times V \] generates
W := U * V generates

Data dependency in opposite direction !!
Remarks:

- Calculation of derivatives only possible after function evaluation at same point
- Complete computational graph is kept in “core”
  Amount of internal storage may cause problems
- Access on nodes largely at random

Conclusion:

Derivative calculation in active class OK for forward mode but not practicable for reverse mode on large problems!!
Remarks:

- Calculation of derivatives only possible after function evaluation at same point
- Complete computational graph is kept in “core”
  Amount of internal storage may cause problems
- Access on nodes largely at random

Conclusion:

Derivative calculation in active class OK for forward mode but not practicable for reverse mode on large problems!!
AD by Internal Representation

Idea:

Class adouble {
    VAL : double;
    IND : integer;
}
AD by Internal Representation

Idea:

```c
Class adouble {
    VAL : double;
    IND : integer;
}
```

⇒ $W = U \times V$ causes recording onto three tapes:

- $INDEXCOUNT += 1$
- $W.IND = INDEXCOUNT$
- $W.VAL = U.VAL \times V.VAL$
- $W.VAL \rightarrow VALUE-TAPE$
- "MULT" $\rightarrow OPERATION-TAPE$
- $(U.IND, V.IND, W.IND) \rightarrow INDEX-TAPE$
Remarks for Tape-based Derivatives

- Mechanical application of chain-rule based on the tapes
- Data access on tapes strictly sequential
- Tapes can be reused for function / derivative calculation at different points
- Several tapes for different control flows depending on active variables
General Remarks for Operator Overloading

**Pros:**
- Only choice for C/C++
- Great flexibility with respect to mode/order
- “Simple” to maintain

**Cons:**
- Compiler optimization may be lost
- Only object code for derivatives
Operator Overloading Tools

C/C++:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Institution</th>
<th>Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADO1</td>
<td>Cranfield Uni., GB</td>
<td>forward + reverse</td>
</tr>
<tr>
<td>ADOL-C</td>
<td>Uni. Paderborn, Germany</td>
<td>forward + reverse</td>
</tr>
<tr>
<td>CppAD</td>
<td>Uni. Washington, USA</td>
<td>forward + reverse</td>
</tr>
<tr>
<td>FADBAD</td>
<td>TU Denmark</td>
<td>forward + reverse</td>
</tr>
<tr>
<td>TADIFF</td>
<td>TU Denmark</td>
<td>forward</td>
</tr>
</tbody>
</table>

...
# Operator Overloading Tools

## C/C++:
- **ADO1**: Cranfield Uni., GB  forward + reverse
- **ADOL-C**: Uni. Paderborn, Germany  forward + reverse
- **CppAD**: Uni. Washington, USA  forward + reverse
- **FADBAD**: TU Denmark  forward + reverse
- **TADIFF**: TU Denmark  forward

## Matlab:
- **ADMAT**: Cayuga Research Associates, USA  forward + reverse

...
Conclusions

- Evaluation of derivatives with working accuracy

  - Forward mode: \( \text{OPS}(F'(x)\dot{x}) \leq c \text{OPS}(F), \quad c \in [2, 5/2] \)
  
  - Reverse mode: \( \text{OPS}(\bar{y}^\top F'(x)) \leq c \text{OPS}(F), \quad c \in [3, 4] \)

\[ \Rightarrow \text{Gradients are cheap} \sim \text{Function Costs!!} \]

- Combination: \( \text{OPS}(\bar{y}^\top, F''(x)\dot{x}) \leq c \text{OPS}(F), \quad c \in [7, 10] \)

- Cost of higher derivatives grows quadratically in the degree
Conclusions

- Evaluation of derivatives with working accuracy
- Efficient evaluation of derivatives of any order
- Numerous tools available, based on source transformation or operator overloading
Conclusions

- Evaluation of derivatives with working accuracy
- Efficient evaluation of derivatives of any order
- Numerous tools available, based on source transformation or operator overloading
- Nondifferentiability only on meager set
- Optimal Jacobi accumulation is (NP-)hard
- Full Jacobians/Hessians often not needed or sparse
Conclusions

- Evaluation of derivatives with working accuracy
- Efficient evaluation of derivatives of any order
- Numerous tools available, based on source transformation or operator overloading
- Nondifferentiability only on meager set
- Optimal Jacobi accumulation is (NP-)hard
- Full Jacobians/Hessians often not needed or sparse
- see www.autodiff.org
Automatic differentiation by overloading in C++

ADOL-C version 2.1

- reorganization of taping
  tape dependent information kept in separate structures
- different differentiation contexts ⇒
  - documented external function facility
  - documented fixpoint iteration facility
  - documented checkpointing facility based on revolve
- documented parallelization of derivative calculation
- coupled with ColPack for exploitation of sparsity
- available at COIN-OR since May 2009
Source Code Modifications

- Include needed header-files
  easiest way: #include "adolc.h"
Source Code Modifications

- Include needed header-files
easiest way: #include “adolc.h”
- Define region that has to be differentiated:

  trace_on(tag, keep);
  ...
  trace_off(file);

  Start of active section and its end
Source Code Modifications

- Include needed header-files
  easiest way: `#include “adolc.h”`
- Define region that has to be differentiated:

  ```c
  trace_on(tag, keep);
  ...
  trace_off(file);
  ```

  Start of active section and its end

- Mark independents and dependents in active section:

  ```c
  xa <<= xp;
  ...
  ya >>= yp;
  ```

  mark and initialize independents and dependents in calculations
Source Code Modifications

- Include needed header-files
  easiest way: `#include “adolc.h”`
- Define region that has to be differentiated:
  ```c
  trace_on(tag,keep);  // Start of active section
  ...
  trace_off(file);     // and its end
  ```
- Mark independents and dependents in active section:
  ```c
  xa <<= xp;  // mark and initialize independents
  ...
  ya >>= yp;  // mark dependents
  ```
- Declare all active variables of type adouble
Source Code Modifications

- Include needed header-files
  easiest way: `#include “adolc.h”`
- Define region that has to be differentiated:
  ```
  trace_on(tag,keep);
  ...
  trace_off(file);
  ```
  Start of active section and its end
- Mark independents and dependents in active section:
  ```
  xa <<= xp;  // mark and initialize independents
  ...
  ya >>= yp;  // mark dependents
  ```
- Declare all active variables of type adouble
- Calculate derivative objects after `trace_off(file)`
Evaluation Procedure (Lighthouse)

\[ y_1 = \frac{\nu \tan(\omega t)}{\gamma - \tan(\omega t)} \]
\[ y_2 = \frac{\gamma \nu \tan(\omega t)}{\gamma - \tan(\omega t)} \]

\[ v_3 = x_1 = \nu \]
\[ v_2 = x_2 = \gamma \]
\[ v_1 = x_3 = \omega \]
\[ v_0 = x_4 = t \]

\[ v_1 = v_{-1} \ast v_0 \equiv \varphi_1(v_{-1}, v_0) \]
\[ v_2 = \tan(v_1) \equiv \varphi_2(v_1) \]
\[ v_3 = v_{-2} - v_2 \equiv \varphi_3(v_{-2}, v_2) \]
\[ v_4 = v_{-3} \ast v_2 \equiv \varphi_4(v_{-3}, v_2) \]
\[ v_5 = v_4 / v_3 \equiv \varphi_5(v_4, v_3) \]
\[ v_6 = v_5 \ast v_{-2} \equiv \varphi_6(v_5, v_{-2}) \]

\[ y_1 = v_5 \]
\[ y_2 = v_6 \]
Model Generation with ADOL-C
Model Generation with ADOL-C

```c
double x1, x2, x3, x4;
double v1, v2, ...
double y1, y2;

x1 = 3.7; x2 = 0.7;
x3 = 0.5; x4 = 0.5;

v1 = x3*x4;
v2 = tan(v1);
v3 = x2-v2;
v4 = x1*v2;
v5 = v4/v3;
v6 = v5*x2

y1 = v5; y2 = v6
```
Model Generation with ADOL-C

```c
adouble x1, x2, x3, x4;
adouble v1, v2, ...
double y1, y2;

trace_on(tag);
x1 <= 3.7; x2 <= 0.7;
x3 <= 0.5; x4 <= 0.5;

v1 = x3*x4;
v2 = tan(v1);
v3 = x2-v2;
v4 = x1*v2;
v5 = v4/v3;
v6 = v5*x2

v5 >= y1; v6 >= y2;
trace_off();
```
Model Generation with ADOL-C

```
adouble x1, x2, x3, x4;
adouble v1, v2, ...
double y1, y2;

trace_on(tag);
x1 <= 3.7; x2 <= 0.7;
x3 <= 0.5; x4 <= 0.5;

v1 = x3*x4;
v2 = tan(v1);
v3 = x2-v2;
v4 = x1*v2;
v5 = v4/v3;
v6 = v5*x2

v5 >>= y1; v6 >>= y2;
trace_off();
```

ADOL-C tape

```
trace_on, tag
<<=, x1, 3.7
<<=, x2, 0.7
<<=, x3, 0.5
<<=, x4, 0.5
*, x3, x4, v1
tan, v1, v2
-, x2, v2, v3
*, x1, v2, v4
/, v4, v3, v5
*, v5, x2, v6
>>=, v5, y1
>>=, v6, y2
trace_on, tag
```
Easy-to-use routines

```c
int gradient (tag, n, x[n], g[n]):
    tag = tape number
    n = # indeps
    x[n] = values of indeps
    g[n] = ∇f(x)
```

```c
int jacobian (tag, m, n, x[n], J[m][n]):
    tag = tape number
    m = # deps
    n = # indeps
    x[n] = values of indeps
    J[m][n] = F'(x)
```

```c
int hessian (tag, n, x[n], H[n][n]):
    tag = tape number
    n = # indeps
    x[n] = values of indeps
    H[n][n] = ∇^2 f(x)
```
Easy-to-use routines

int gradient (tag,n,x[n],g[n]):

- tag = tape number
- n = # indeps
- x[n] = values of indeps
- g[n] = $\nabla f(x)$

int jacobian (tag,m,n,x[n],J[m][n]):

- tag = tape number
- m = # deps
- n = # indeps
- x[n] = values of indeps
- J[m][n] = $F'(x)$
Easy-to-use routines

int gradient (tag,n,x[n],g[n]):
  tag       = tape number
  n         = # indeps
  x[n]      = values of indeps
  g[n]      = $\nabla f(x)$

int jacobian (tag,m,n,x[n],J[m][n]):
  tag       = tape number
  m         = # deps
  n         = # indeps
  x[n]      = values of indeps
  J[m][n]   = $F'(x)$

int hessian (tag,n,x[n],H[n][n]):
  tag       = tape number
  n         = # indeps
  x[n]      = values of indeps
  H[n][n]   = $\nabla^2 f(x)$
Further drivers for optimization:

- `vec_jac(tag,m,n,repeat,x[n],u[m],z[n])` Computes $z = u^T F'(x)$
- `jac_vec(tag,m,n,x[n],v[n],z[n])` Computes $z = F'(x)v$
- `hess_vec(tag,n,x[n],v[n],z[n])` Computes $z = \nabla^2 f(x)v$
- `lagra_hess_vec(tag,n,m,x[n],v[n],u[m],h[n])` Computes $h = u^T F''(x)v$
  Extension to $u^T F''(x)V$ available
- `jac_solv(tag,n,x[n],b[n], sparse, mode)` Computes $w$ with $F'(x)w = b$ and store result in $b$
Drivers for sparse Derivative Matrices

**Jacobian:**

```c
sparse_jac(tag,m,n,repeat,x,&nnz,&row_ind,&col_ind,
    &values,options);
```

Hessian:

```c
sparse_hess(tag,n,repeat,x,&nnz,&row_ind,&col_ind,
    &values,options);
```

graph coloring routines by ColPack (Gebrehemedin, Pothen)
Drivers for sparse Derivative Matrices

**Jacobian:**

```c
sparse_jac(tag, m, n, repeat, x, &nnz, &row_ind, &col_ind, &values, options);
jac_pat(tag, m, n, x, JP, options);
generate_seed_jac(m, n, JP, &seed, &p, option);
```
Drivers for sparse Derivative Matrices

**Jacobian:**

```c
sparse_jac(tag, m, n, repeat, x, &nnz, &row_ind, &col_ind, &values, options);
jac_pat(tag, m, n, x, JP, options);
generate_seed_jac(m, n, JP, &seed, &p, option);
```

**Hessian:**

```c
sparse_hess(tag, n, repeat, x, &nnz, &row_ind, &col_ind, &values, options);
```
Drivers for sparse Derivative Matrices

**Jacobian:**

\[
\text{sparse_jac}(\text{tag}, m, n, \text{repeat}, x, &\text{nnz}, &\text{row_ind}, &\text{col_ind}, \\
&\text{values}, \text{options})
\]

\[
\text{jac_pat}(\text{tag}, m, n, x, \text{JP}, \text{options})
\]

\[
\text{generate_seed_jac}(m, n, \text{JP}, &\text{seed}, &\text{p}, \text{option})
\]

**Hessian:**

\[
\text{sparse_hess}(\text{tag}, n, \text{repeat}, x, &\text{nnz}, &\text{row_ind}, &\text{col_ind}, \\
&\text{values}, \text{options})
\]

\[
\text{hess_pat}(\text{tag}, n, x, \text{HP}, \text{options})
\]

\[
\text{generate_seed_hess}(n, \text{HP}, &\text{seed}, &\text{p}, \text{option})
\]

graph coloring routines by ColPack (Gebrehemedin, Pothen)
Advanced taping techniques

ADOL-C tape (basic)

Part A          Part B          Part C
Advanced taping techniques

ADOL-C tape (basic)

Part A  Part B  Part C

ADOL-C tape (nested)
Advanced taping techniques

ADOL-C tape (basic)

Part A          Part B          Part C

ADOL-C tape (nested)

External differentiated functions
Differentiation of fixpoint iterations

ADOL-C tape (basic)

- initialization
- iteration
- evaluation
Differentiation of fixpoint iterations

ADOL–C tape (basic)

 initialization  iteration  evaluation

ADOL–C tape (nested)

ADOL–C function: fp_iterate()
Differentiation of fixpoint iterations

ADOL–C tape (basic)

- initialization
- iteration
- evaluation

reverse evaluation

repeated subtape evaluation
Summary and Outlook

- AD for C and C++ ⇒ derivatives of any order
- Easy-to-use routines for standard derivative objects, optimization, ODEs, tensors, . . .
- Randomly accessed memory of original magnitude
- Sequential access of data on tape, i.e., array or file
- Wide range of application:
  - aerodynamic, chemical engineering, medicine, network opt., . . .

Several ideas for improvement:
- runtime activity, parallelization, front ends, . . .

Contact:
http://www.coin-or.org/projects/ADOL-C.xml