Automatic differentiation of numerical integration algorithms

after Peter Eberhard, Christian Bischof

Zofia Mączyńska

21. August 2006

Outline

Introduction

We will present the application of automatic differentiation to numerical integration algorithms for ODE's, in particular the ramifications of the fact that AD is applied not only to the **solution** but also to the **solution procedure** itself.

mathematical model of technical systems or natural phenomena

Initial value problem, ODE

For given values of system parameters $p \in \mathbb{R}^h$, find the trajectories $x(t,p), x \in \mathbb{R}^n$ for $t^0 \leq t \leq t^1$, where x is the state vector, t the time. The state is determined by the solution of the initial value problem:

$$\dot{x} = f(x,p,t), x(t = t^0, p) = x^0,$$

where f is the vector of state derivatives, x^0 the initial state, t^0 and t^1 the initial and final time respectively.

The ODE is typically solved by using numerical integration algorithms. In engineering applications one is not only interested in the final state but also in the performance criteria ψ computed from the trajectories $\mathbf{x}$. If optimization procedures are applied on order to choose optimal design variables with respect to certain performance criteria, or if parameter estimation techniques are used to identify model parameters from measurements, then, with the final state

$$x^1 := x(t^1, p)$$

one typically has to compute

$$\frac{\partial x^1}{\partial p^T}.$$

The numerical value of the sensitivities at the final step may depend on the whole history of the system.

- AD tools have been developed to make it possible to augment general Fortran or C codes with statements for the **computation of derivatives in an automated fashion**.
- AD is based on the fact that every computer program employs only **the simplest operations** (addition, multiplication, sin, etc.) whose derivatives are known.
- AD computes the derivatives for the whole programm and then compose them using the **chain rule** of differential calculus.

- AD tools have been developed to make it possible to augment general Fortran or C codes with statements for the **computation of derivatives in an automated fashion**.
- AD is based on the fact that every computer program employs only **the simplest operations** (addition, multiplication, sin, etc.) whose derivatives are known.
- AD computes the derivatives for the whole programm and then compose them using the **chain rule** of differential calculus.

- AD tools have been developed to make it possible to augment general Fortran or C codes with statements for the **computation of derivatives in an automated fashion**.
- AD is based on the fact that every computer program employs only **the simplest operations** (addition, multiplication, sin, etc.) whose derivatives are known.
- AD computes the derivatives for the whole programm and then compose them using the **chain rule** of differential calculus.

Watch out!

AD differentiate not only the solution computed by a programm, but also the algorithm by which the solution is being derived. Thus, the value of the derivative may depend on the algorithm chosen to compute the solution.

PLAN

We will:

1. consider AD of a prototypical integration algorithm and illustrate how different integrators can lead to different values of computed derivatives
2. suggest two approaches to suppress the impact of the solution algorithm on derivatives

PLAN

We will:

1. consider AD of a prototypical integration algorithm and illustrate how different integrators can lead to different values of computed derivatives
2. suggest two approaches to suppress the impact of the solution algorithm on derivatives

Automatic differentiation of prototypical numerical integration algorithms

Algorithms for numerical integration of ODEs:

1. single step
2. multistep
3. extrapolation
4. special (for stiff systems, highly or loosely coupled systems, systems composed of several subsystems, etc.)

Algorithms for numerical integration of ODEs:

1. single step
2. multistep
3. extrapolation
4. special (for stiff systems, highly or loosely coupled systems, systems composed of several subsystems, etc.)

Algorithms for numerical integration of ODEs:

1. single step
2. multistep
3. extrapolation
4. special (for stiff systems, highly or loosely coupled systems, systems composed of several subsystems, etc.)

Algorithms for numerical integration of ODEs:

1. single step
2. multistep
3. extrapolation
4. special (for stiff systems, highly or loosely coupled systems, systems composed of several subsystems, etc.)

We will illustrate the interplay of the AD and the integration algorithms (single-step algorithm of Euler and Runge-Kutta type with and without stepsize control and a sophisticated multistep intergation algorithm with adaptive stepsize control and interpolation order control.)

single-step algorithm

Time discretization applied to $\dot{x} = f(x, p, t), x(t = t^0, p) = x^0$ leads to a recursive scheme:

$$x_{i+1} = x_i + h_i \dot{\bar{x}}_i$$

$$t_{i+1} = t_i + h_i,$$

where:
i denotes the ith integration step, $x_i := x(t_i)$, h_i is the actual stepsize, $\dot{\bar{x}}$ denotes slope estimation.
Two cases are possible:

- $h_i = h = const$ leads to the Euler scheme
- dynamically adaptive stepsize control (based on local error estimates).

single-step algorithm

Time discretization applied to $\dot{x} = f(x, p, t), x(t = t^0, p) = x^0$ leads to a recursive scheme:

$$x_{i+1} = x_i + h_i \dot{\bar{x}}_i$$

$$t_{i+1} = t_i + h_i,$$

where:
i denotes the ith integration step, $x_i := x(t_i)$, h_i is the actual stepsize, $\dot{\bar{x}}$ denotes slope estimation.
Two cases are possible:

- $h_i = h = const$ leads to the Euler scheme
- dynamically adaptive stepsize control (based on local error estimates).

multistep algorithm

Multistep algorithms additionally use the information from former steps to predict the appropriate stepsizes and slopes.

All variables in

$$x_{i+1} = x_i + h_i \dot{\bar{x}}_i$$

may depend on p.
This leads to:

$$x_{i+1}(p) = x_i(p) + h_i(p) \dot{\bar{x}}_i(p)$$

Differentiating

$$x_{i+1}(p) = x_i(p) + h_i(p)\dot{\bar{x}}_i(p)$$

with respect to p with $\nabla x := \frac{dx}{dp^T}$ gives:

$$\nabla x_{i+1} = \nabla x_i + h_i \nabla \dot{\bar{x}}_i + \nabla h_i \dot{\bar{x}}_i.$$

Computation of the desired derivatives for the state variables

To obtain the desired derivatives we can consider two choices:

- manual transformation
- automatic transformation with AD tools

Computation of the desired derivatives for the state variables

To obtain the desired derivatives we can consider two choices:

- manual transformation
- automatic transformation with AD tools

Differentiating

$$\dot{x} = f(x, p, t)$$

with respect to p we obtain (with $\frac{dt}{dp^T} = 0$):

$$\frac{d}{dp^T}(\dot{x}) = \frac{d}{dp^T}(\frac{dx}{dt}) = \frac{\partial f}{\partial x^T}\frac{dx}{dp^T} + \frac{\partial f}{\partial p^T}$$

Exchanging the order of differentiation we obtain a new ODE for ∇x:

$$\frac{d}{dt}(\frac{dx}{dp^T}) = \frac{d}{dt}(\nabla x) = [\dot{\nabla} x] = \frac{\partial f}{\partial x^T}\nabla x + \frac{\partial f}{\partial p^T},\ \nabla x(t = t^0) = \nabla x^0,$$

which we can integrate alongside our original solution. AD techniques could be employed to compute $\frac{\partial f}{\partial x^T}$ and $\frac{\partial f}{\partial p^T}$ but we would not integrate through the integration algorithm for x.

observation

The stepsize h_i is likely to depend on the parameters p and the AD tool will associate a gradient ∇h_i with h_i. Thus, the update $\nabla x_{i+1} = \nabla x_i + h_i \nabla \dot{\bar{x}}_i + \nabla h_i \dot{\bar{x}}_i$, which will be computed by an AD tool, leads to an inconsistent integrator for ∇x, as the final result depends on the discretization strategy chosen. Hence it is also unlikely that $\nabla x^1 = \nabla x|_{(t_i = t^1)}$ equals the desired $\frac{\partial x^1}{\partial p^T}$.

The automatically differentiated integration algorithm computes $x^1(t^1(p), p)$ (the dependence of t^1 on p results only from the adaptive time discretization). Differentiating with respect to p we obtain:

$$\nabla x^1 = \frac{\partial x^1}{\partial t^1} \nabla t^1 + \frac{\partial x^1}{\partial p^T}$$

The green depend on the time discretization and are computed by the AD generated code. Thus, to arrive at the desired solution $\frac{\partial x^1}{\partial p^T}$, we can pursue one of the following strategies:

$$\nabla x^1 = \frac{\partial x^1}{\partial t^1} \nabla t^1 + \frac{\partial x^1}{\partial p^T}$$

We can pursue one of the following strategies:

- Perform a posteriori error correction:
  The desired derivatives $\frac{\partial x^1}{\partial p^T}$, which don't depend on the time discretization can be computed as:

$$\frac{\partial x^1}{\partial p^T} = \nabla x^1 - f(x^1, p, t^1) \nabla t^1.$$

- Use an integrator with fixed stepsize:
  In this case, we have: $\nabla h_i \equiv 0, \forall i$. Thus $\nabla t \equiv 0$ and hence: $\nabla x^1 \equiv \frac{\partial x^1}{\partial p^T}$. The AD-computed derivative trajectories are the desired ones, and thus, no modification is required for fixed stepsize integration algorithms.

$$\nabla x^1 = \frac{\partial x^1}{\partial t^1} \nabla t^1 + \frac{\partial x^1}{\partial p^T}$$

We can pursue one of the following strategies:

- Perform a posteriori error correction:
  The desired derivatives $\frac{\partial x^1}{\partial p^T}$, which don't depend on the time discretization can be computed as:

$$\frac{\partial x^1}{\partial p^T} = \nabla x^1 - f(x^1, p, t^1) \nabla t^1.$$

- Use an integrator with fixed stepsize:
  In this case, we have: $\nabla h_i \equiv 0, \forall i$. Thus $\nabla t \equiv 0$ and hence: $\nabla x^1 \equiv \frac{\partial x^1}{\partial p^T}$. The AD-computed derivative trajectories are the desired ones, and thus, no modification is required for fixed stepsize integration algorithms.

$$\nabla x^1 = \frac{\partial x^1}{\partial t^1} \nabla t^1 + \frac{\partial x^1}{\partial p^T}$$

We can pursue one of the following strategies:

- Perform a posteriori error correction.
- Use an integrator with fixed stepsize.
- Modify the AD-generated code to enforce $\nabla h_i = 0 \forall i$:
  - For the first step the user must guess the initial step h_0
  - h_0 independent of p, thus $\nabla h_0 = 0$
  - Assume that the correct stepsize h is known in advance for each step. Then $\nabla h_i = 0 \forall i$ and we get the same correct results as for the fixed stepsize algorithm.

$$\nabla x^1 = \frac{\partial x^1}{\partial t^1} \nabla t^1 + \frac{\partial x^1}{\partial p^T}$$

We can pursue one of the following strategies:

- Perform a posteriori error correction.
- Use an integrator with fixed stepsize.
- Modify the AD-generated code to enforce $\nabla h_i = 0 \forall i$:
  - For the first step the user must guess the initial step h_0
  - h_0 independent of p, thus $\nabla h_0 = 0$
  - Assume that the correct stepsize h is known in advance for each step. Then $\nabla h_i = 0 \forall i$ and we get the same correct results as for the fixed stepsize algorithm.

$$\nabla x^1 = \frac{\partial x^1}{\partial t^1} \nabla t^1 + \frac{\partial x^1}{\partial p^T}$$

We can pursue one of the following strategies:

- Perform a posteriori error correction.
- Use an integrator with fixed stepsize.
- Modify the AD-generated code to enforce $\nabla h_i = 0 \forall i$:
  - For the first step the user must guess the initial step h_0
  - h_0 independent of p, thus $\nabla h_0 = 0$
  - Assume that the correct stepsize h is known in advance for each step. Then $\nabla h_i = 0 \forall i$ and we get the same correct results as for the fixed stepsize algorithm.

$$\nabla x^1 = \frac{\partial x^1}{\partial t^1} \nabla t^1 + \frac{\partial x^1}{\partial p^T}$$

We can pursue one of the following strategies:

- Perform a posteriori error correction.
- Use an integrator with fixed stepsize.
- Modify the AD-generated code to enforce $\nabla h_i = 0 \forall i$:
  - For the first step the user must guess the initial step h_0
  - h_0 independent of p, thus $\nabla h_0 = 0$
  - Assume that the correct stepsize h is known in advance for each step. Then $\nabla h_i = 0 \forall i$ and we get the same correct results as for the fixed stepsize algorithm.

$$\nabla x^1 = \frac{\partial x^1}{\partial t^1} \nabla t^1 + \frac{\partial x^1}{\partial p^T}$$

We can pursue one of the following strategies:

- Perform a posteriori error correction.
- Use an integrator with fixed stepsize.
- Modify the AD-generated code to enforce $\nabla h_i = 0 \forall i$:
  - For the first step the user must guess the initial step h_0
  - h_0 independent of p, thus $\nabla h_0 = 0$
  - Assume that the correct stepsize h is known in advance for each step. Then $\nabla h_i = 0 \forall i$ and we get the same correct results as for the fixed stepsize algorithm.

$$\nabla x^1 = \frac{\partial x^1}{\partial t^1} \nabla t^1 + \frac{\partial x^1}{\partial p^T}$$

We can pursue one of the following strategies:

- Perform a posteriori error correction.
- Use an integrator with fixed stepsize.
- Modify the AD-generated code to enforce $\nabla h_i = 0 \forall i$:
  - For the first step the user must guess the initial step h_0
  - h_0 independent of p, thus $\nabla h_0 = 0$
  - Assume that the correct stepsize h is known in advance for each step. Then $\nabla h_i = 0 \forall i$ and we get the same correct results as for the fixed stepsize algorithm.

Experimental results with a one-mass oscillator

the simplest multibody system

We consider a horizontal one-mass oscillator. One can derive solutions for both, that state and its gradients, and thus it is a well suited example for us.

mechanical model

- A body mass m can slide on a horizontal ground.
- It is coupled to the wall with a linear spring with spring stiffness c
- the position is described by $y(t)$
- From the Newton's equation one can derive the equation of motion:

$$m\ddot{y} + cy = 0,$$

equivalently, with $x = [y, \dot{y}]^T$ a system of first order ODEs:

$$\dot{x} = [\dot{x}_1, \dot{x}_2]^T = [x_2, -\frac{c}{m}x_1]^T$$

- With the initial condition $x(t = t^0) = x^0 = [0, v^0]^T$, the solution of the ODE is:

$$y(t) = v^0 \sqrt{\frac{m}{c}} sin \sqrt{\frac{c}{m}} t.$$

mechanical model

- A body mass m can slide on a horizontal ground.
- It is coupled to the wall with a linear spring with spring stiffness c
- the position is described by $y(t)$
- From the Newton's equation one can derive the equation of motion:

$$m\ddot{y} + cy = 0,$$

equivalently, with $x = [y, \dot{y}]^T$ a system of first order ODEs:

$$\dot{x} = [\dot{x}_1, \dot{x}_2]^T = [x_2, -\frac{c}{m}x_1]^T$$

- With the initial condition $x(t = t^0) = x^0 = [0, v^0]^T$, the solution of the ODE is:

$$y(t) = v^0 \sqrt{\frac{m}{c}} sin \sqrt{\frac{c}{m}} t.$$

mechanical model

- A body mass m can slide on a horizontal ground.
- It is coupled to the wall with a linear spring with spring stiffness c
- the position is described by $y(t)$
- From the Newton's equation one can derive the equation of motion:

$$m\ddot{y} + cy = 0,$$

equivalently, with $x = [y, \dot{y}]^T$ a system of first order ODEs:

$$\dot{x} = [\dot{x}_1, \dot{x}_2]^T = [x_2, -\frac{c}{m}x_1]^T$$

- With the initial condition $x(t = t^0) = x^0 = [0, v^0]^T$, the solution of the ODE is:

$$y(t) = v^0 \sqrt{\frac{m}{c}} sin \sqrt{\frac{c}{m}} t.$$

mechanical model

- A body mass m can slide on a horizontal ground.
- It is coupled to the wall with a linear spring with spring stiffness c
- the position is described by $y(t)$
- From the Newton's equation one can derive the equation of motion:

$$m\ddot{y} + cy = 0,$$

equivalently, with $x = [y, \dot{y}]^T$ a system of first order ODEs:

$$\dot{x} = [\dot{x}_1, \dot{x}_2]^T = [x_2, -\frac{c}{m}x_1]^T$$

- With the initial condition $x(t = t^0) = x^0 = [0, v^0]^T$, the solution of the ODE is:

$$y(t) = v^0 \sqrt{\frac{m}{c}} sin \sqrt{\frac{c}{m}} t.$$

mechanical model

- A body mass m can slide on a horizontal ground.
- It is coupled to the wall with a linear spring with spring stiffness c
- the position is described by $y(t)$
- From the Newton's equation one can derive the equation of motion:

$$m\ddot{y} + cy = 0,$$

equivalently, with $x = [y, \dot{y}]^T$ a system of first order ODEs:

$$\dot{x} = [\dot{x}_1, \dot{x}_2]^T = [x_2, -\frac{c}{m}x_1]^T$$

- With the initial condition $x(t = t^0) = x^0 = [0, v^0]^T$, the solution of the ODE is:

$$y(t) = v^0 \sqrt{\frac{m}{c}} sin \sqrt{\frac{c}{m}} t.$$

For $m = 1$ and the system parameters $p = [c, v^0]$ we find:

$$y(t) = v^0 \frac{1}{\sqrt{c}} sin\sqrt{c}t,$$

$$\dot{y}(t) = v^0 cos\sqrt{c}t,$$

$$\ddot{y}(t) = -v^0 \sqrt{c} sin\sqrt{c}t.$$

We now define two criteria:

- The criterion ψ_1 contains the position of the body at the time $t^1 = \frac{\pi}{2}$.
  For $p = [1, 0.5]^T$ we have:

$$\frac{d\psi_1}{dp_1} = \frac{d\psi_1}{dc} = \ldots = -0.25, \frac{d\psi_1}{dp_2} = \frac{d\psi_1}{dv^0} = \frac{1}{\sqrt{c}} sin(\sqrt{c}\frac{\pi}{2}) = 1.0.$$

- The criterion ψ_2 integrates the position over the whole interesting simulation time interval $[t^0, t^1]$:

$$\psi_2 = \int_{t^0}^{t^1} y(t)dt = \int_{t^0=0}^{t^1=\frac{\pi}{2}} \frac{v^0}{\sqrt{c}} sin\sqrt{c}t.$$

  Explicit solution for the gradients:

$$\frac{\psi_2}{dp_1} = \frac{\psi_2}{dc} = \ldots = \frac{\pi}{8} - \frac{1}{2}, \frac{\psi_2}{dp_2} = \frac{\psi_2}{dv^0} = \ldots = 1.0.$$

We now define two criteria:

- The criterion ψ_1 contains the position of the body at the time $t^1 = \frac{\pi}{2}$.
  For $p = [1, 0.5]^T$ we have:

$$\frac{d\psi_1}{dp_1} = \frac{d\psi_1}{dc} = \ldots = -0.25, \frac{d\psi_1}{dp_2} = \frac{d\psi_1}{dv^0} = \frac{1}{\sqrt{c}} sin(\sqrt{c}\frac{\pi}{2}) = 1.0.$$

- The criterion ψ_2 integrates the position over the whole interesting simulation time interval $[t^0, t^1]$:

$$\psi_2 = \int_{t^0}^{t^1} y(t)dt = \int_{t^0=0}^{t^1=\frac{\pi}{2}} \frac{v^0}{\sqrt{c}} sin\sqrt{c}t.$$

  Explicit solution for the gradients:

$$\frac{\psi_2}{dp_1} = \frac{\psi_2}{dc} = \ldots = \frac{\pi}{8} - \frac{1}{2}, \frac{\psi_2}{dp_2} = \frac{\psi_2}{dv^0} = \ldots = 1.0.$$

Single-step integration without stepsize control

- We investigate and compare three similar integration schemes: Euler scheme, the Heun algorithm, and the 4th order Runge-Kutta algorithm.
- **Result:**
  - Only minor differences between the reference gradient and the AD gradient exists
  - The relative error in the criteria is about the size of the error on the gradients
- **Explanation:** As expected AD of single-step integration algorithms without stepsize control leads to the desired results without any need for user modification. The differnties can be explained by the choice of the stepsize and the algorithm, no additional errors are introduced in the AD procedure.

Single-step integration without stepsize control

- We investigate and compare three similar integration schemes: Euler scheme, the Heun algorithm, and the 4th order Runge-Kutta algorithm.
- **Result:**
  - Only minor differences between the reference gradient and the AD gradient exists
  - The relative error in the criteria is about the size of the error on the gradients
- **Explanation:** As expected AD of single-step integration algorithms without stepsize control leads to the desired results without any need for user modification. The differnties can be explained by the choice of the stepsize and the algorithm, no additional errors are introduced in the AD procedure.

Single-step integration without stepsize control

- We investigate and compare three similar integration schemes: Euler scheme, the Heun algorithm, and the 4th order Runge-Kutta algorithm.
- **Result:**
  - Only minor differences between the reference gradient and the AD gradient exists
  - The relative error in the criteria is about the size of the error on the gradients
- **Explanation:** As expected AD of single-step integration algorithms without stepsize control leads to the desired results without any need for user modification. The differnties can be explained by the choice of the stepsize and the algorithm, no additional errors are introduced in the AD procedure.

Single-step integration without stepsize control

- We investigate and compare three similar integration schemes: Euler scheme, the Heun algorithm, and the 4th order Runge-Kutta algorithm.
- **Result:**
  - Only minor differences between the reference gradient and the AD gradient exists
  - The relative error in the criteria is about the size of the error on the gradients
- **Explanation:** As expected AD of single-step integration algorithms without stepsize control leads to the desired results without any need for user modification. The differnties can be explained by the choice of the stepsize and the algorithm, no additional errors are introduced in the AD procedure.

Single-step integration without stepsize control

- We investigate and compare three similar integration schemes: Euler scheme, the Heun algorithm, and the 4th order Runge-Kutta algorithm.
- **Result:**
  - Only minor differences between the reference gradient and the AD gradient exists
  - The relative error in the criteria is about the size of the error on the gradients
- **Explanation:** As expected AD of single-step integration algorithms without stepsize control leads to the desired results without any need for user modification. The differnties can be explained by the choice of the stepsize and the algorithm, no additional errors are introduced in the AD procedure.

Single step integration algorithms with adaptive stepsize control

We consider a mixed 4th and 5th order Runge-Kutta algorithm with stepsize control.
The error Δ is of magnitude h^5, thus we can estimate the required stepsize $\bar{h}$ from the desired error bound $\bar{\Delta}$, the actual stepsize h and the actual error Δ:

$$\frac{\bar{h}^5}{\bar{\Delta}} \approx \frac{h^5}{\Delta} \Rightarrow \bar{h} \approx h \sqrt[5]{\frac{\bar{\Delta}}{\Delta}}$$

If $\bar{h} > h$, the actual stepsize was too big and the step has to be repeated with decreased stepsize until the the error estimate is acceptable. Otherwise the next step is computed and the integration proceeds.

- The actual stepsize h_i and the actual time t_i depend n the state x_i and therefore on the system parameters p, so the AD tool will compute the gradients $\nabla h_i = \frac{dh_i}{dp^T}$ and $\nabla t_i = \frac{dt_i}{dp^T}$.
- We then employ the relation $\frac{\partial x^1}{\partial p^T} = \nabla x^1 - f(x^1, p, t^1)\nabla t^1$ to compute (at every time step) the desired $\frac{\partial x}{\partial p^T}$ from ∇x and ∇t.

- The actual stepsize h_i and the actual time t_i depend n the state x_i and therefore on the system parameters p, so the AD tool will compute the gradients $\nabla h_i = \frac{dh_i}{dp^T}$ and $\nabla t_i = \frac{dt_i}{dp^T}$.
- We then employ the relation $\frac{\partial x^1}{\partial p^T} = \nabla x^1 - f(x^1, p, t^1)\nabla t^1$ to compute (at every time step) the desired $\frac{\partial x}{\partial p^T}$ from ∇x and ∇t.

multistep integration algorithm (Shampine-Gordon)

In a multistep algorithm the information already available from previous steps is used to predict further steps. (The integration algorithm consists of about 900 lines of code and therefore the manual modification of the code to ensure $\nabla h_i = 0$ is not a reliable approach, the a posteriori error correction is used instead and it leads to correct results.

Application to a technical system

- We consider a robot which consists of 7 bodies, has 5 degrees of freedom and is described by 10 ODEs.
- We investigate the sensitivity of the position of the end effector at the final time with respect to the disturbances in several system parameters $p = [F_{1,z}, L, t_{end}, m_2, I_{3zz}]^T$ (driving force, geometrical length, final time, mass, a component of the inertia tensor).
- The results are verified by a using the **adjoint variable method (AVM)** and (very costly) finite-difference approximations with adaptive order control.

- We consider a robot which consists of 7 bodies, has 5 degrees of freedom and is described by 10 ODEs.
- We investigate the sensitivity of the position of the end effector at the final time with respect to the disturbances in several system parameters $p = [F_{1,z}, L, t_{end}, m_2, I_{3zz}]^T$ (driving force, geometrical length, final time, mass, a component of the inertia tensor).
- The results are verified by a using the **adjoint variable method (AVM)** and (very costly) finite-difference approximations with adaptive order control.

- We consider a robot which consists of 7 bodies, has 5 degrees of freedom and is described by 10 ODEs.
- We investigate the sensitivity of the position of the end effector at the final time with respect to the disturbances in several system parameters $p = [F_{1,z}, L, t_{end}, m_2, I_{3zz}]^T$ (driving force, geometrical length, final time, mass, a component of the inertia tensor).
- The results are verified by a using the **adjoint variable method (AVM)** and (very costly) finite-difference approximations with adaptive order control.

The reference criterion and reference gradient obtained using AVM with integration error tolerances near machine accuracy is for the component $\frac{\partial \psi}{\partial p_1}$ as follows:

$$\psi = 4.136636, \frac{\partial \psi}{\partial p_1} = 0.0186126.$$

- If the relative and the absolute error bounds for the Shampine-Gordon integration algorithms are chosen as $relerr = abserr = 10^{-7}$, we get the following errors in the component $\frac{\partial \psi}{\partial p_1}$:

  AVM $relerr = 8.06 \cdot 10^{-7}, abserr = 1.5 \cdot 10^{-8}$

  AD + corrections $relerr = 2.31 \cdot 10^{-7}, abserr = 4.3 \cdot 10^{-8}$.

- Both gradients are sufficiently accurate and both methods can be used for the sensitivity analysis of a multibody system.

- If the relative and the absolute error bounds for the Shampine-Gordon integration algorithms are chosen as $relerr = abserr = 10^{-7}$, we get the following errors in the component $\frac{\partial \psi}{\partial p_1}$:

  AVM $relerr = 8.06 \cdot 10^{-7}, abserr = 1.5 \cdot 10^{-8}$

  AD + corrections $relerr = 2.31 \cdot 10^{-7}, abserr = 4.3 \cdot 10^{-8}$.

- Both gradients are sufficiently accurate and both methods can be used for the sensitivity analysis of a multibody system.

- If the relative and the absolute error bounds for the Shampine-Gordon integration algorithms are chosen as $relerr = abserr = 10^{-7}$, we get the following errors in the component $\frac{\partial \psi}{\partial p_1}$:

  AVM $relerr = 8.06 \cdot 10^{-7}, abserr = 1.5 \cdot 10^{-8}$

  AD + corrections $relerr = 2.31 \cdot 10^{-7}, abserr = 4.3 \cdot 10^{-8}$.

- Both gradients are sufficiently accurate and both methods can be used for the sensitivity analysis of a multibody system.

- If the relative and the absolute error bounds for the Shampine-Gordon integration algorithms are chosen as $relerr = abserr = 10^{-7}$, we get the following errors in the component $\frac{\partial \psi}{\partial p_1}$:

  AVM $relerr = 8.06 \cdot 10^{-7}, abserr = 1.5 \cdot 10^{-8}$

  AD + corrections $relerr = 2.31 \cdot 10^{-7}, abserr = 4.3 \cdot 10^{-8}$.
- Both gradients are sufficiently accurate and both methods can be used for the sensitivity analysis of a multibody system.

- In general:
  - the computation of the adjoint variable gradients is more efficient, but they are based on a hand coded, highly optimized algorithm whose implementation took man-years
  - AD-generated code is fairly simple to create and requires (including the result verification) much less time, at the expense of a less efficient execution.
  - However the time verification of the gradients can be very high and unless the algorithm and the implementations are well understood, one should check the results carefully. This is due to the fact that AD differentiate the algorithm without any knowledge of the mathematics that underlie the algorithm.

- In general:
  - the computation of the adjoint variable gradients is more efficient, but they are based on a hand coded, highly optimized algorithm whose implementation took man-years
  - AD-generated code is fairly simple to create and requires (including the result verification) much less time, at the expense of a less efficient execution.
  - However the time verification of the gradients can be very high and unless the algorithm and the implementations are well understood, one should check the results carefully. This is due to the fact that AD differentiate the algorithm without any knowledge of the mathematics that underlie the algorithm.

- In general:
  - the computation of the adjoint variable gradients is more efficient, but they are based on a hand coded, highly optimized algorithm whose implementation took man-years
  - AD-generated code is fairly simple to create and requires (including the result verification) much less time, at the expense of a less efficient execution.
  - However the time verification of the gradients can be very high and unless the algorithm and the implementations are well understood, one should check the results carefully. This is due to the fact that AD differentiate the algorithm without any knowledge of the mathematics that underlie the algorithm.

- In general:
  - the computation of the adjoint variable gradients is more efficient, but they are based on a hand coded, highly optimized algorithm whose implementation took man-years
  - AD-generated code is fairly simple to create and requires (including the result verification) much less time, at the expense of a less efficient execution.
  - However the time verification of the gradients can be very high and unless the algorithm and the implementations are well understood, one should check the results carefully. This is due to the fact that AD differentiate the algorithm without any knowledge of the mathematics that underlie the algorithm.

Conclusions

- The numerical behavior of the criteria **and** the gradient computation must be studied carefully. It is not obvious that the stepsize control is determined only by the state variables required for the criteria computation; and, therefore, the errors introduced in the state variables for the gradients may be bigger than the prescribed error bounds for the state variables for the criteria.
- The application of plan AD often yields the right results **nevertheless** the inclusion of expert knowledge can highly improve the performance and numerical behavior. (Example: if the differentiation of the stepsize control is switched off, we can compute the correct gradients more efficiently)

- The numerical behavior of the criteria **and** the gradient computation must be studied carefully. It is not obvious that the stepsize control is determined only by the state variables required for the criteria computation; and, therefore, the errors introduced in the state variables for the gradients may be bigger than the prescribed error bounds for the state variables for the criteria.
- The application of plan AD often yields the right results **nevertheless** the inclusion of expert knowledge can highly improve the performance and numerical behavior. (Example: if the differentiation of the stepsize control is switched off, we can compute the correct gradients more efficiently)