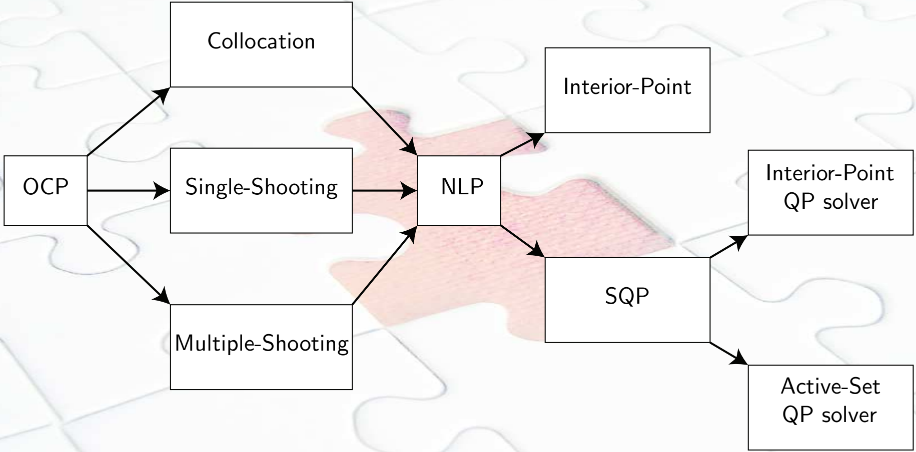
Numerical Optimal Control
Lecture 5: Integrators with Sensitivities

Sébastien Gros

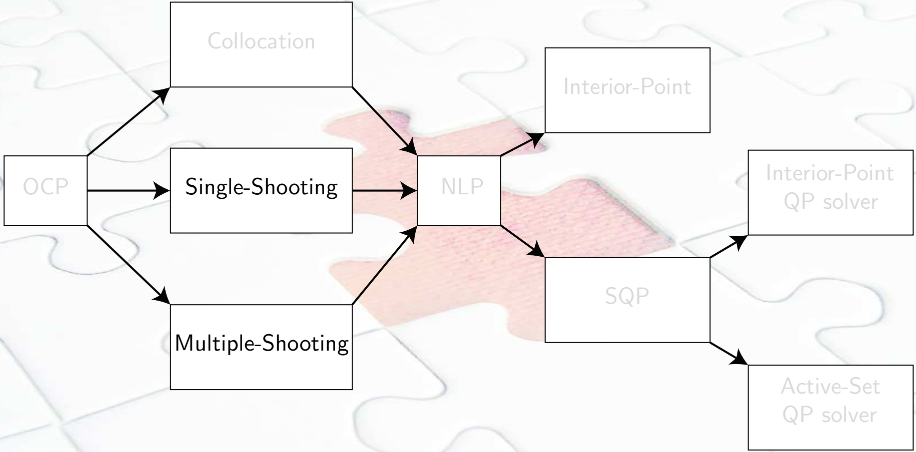
ITK, NTNU

NTNU PhD course

Survival map of Direct Optimal Control



Survival map of Direct Optimal Control



Shooting requires integrators, let's have a look at that...

Outline

Outline

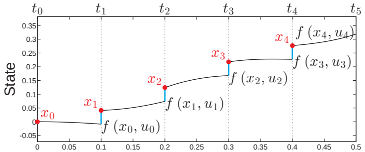
Multiple-Shooting & SQP - Reminder

Multiple shooting yields an NLP:

$$\min_{\mathbf{w}} \quad \Phi(\mathbf{w})$$
$$\text{s.t.} \quad \mathbf{g}(\mathbf{w}) = 0$$

with $\mathbf{w} = \{\mathbf{x}_0, \mathbf{u}_0, ..., \mathbf{x}_{N-1}, \mathbf{u}_{N-1}, \mathbf{x}_N\}$ and

$$\mathbf{g}(\mathbf{w}) = \begin{bmatrix} \mathbf{b}(\mathbf{x}_0, \mathbf{x}_N) \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) - \mathbf{x}_1 \\ \mathbf{f}(\mathbf{x}_1, \mathbf{u}_1) - \mathbf{x}_2 \\ \dots \end{bmatrix}$$



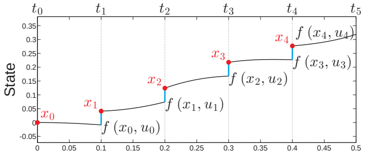
Multiple-Shooting & SQP - Reminder

Multiple shooting yields an NLP:

$$\min_{\mathbf{w}} \quad \Phi(\mathbf{w})$$
$$\text{s.t.} \quad \mathbf{g}(\mathbf{w}) = 0$$

with $\mathbf{w} = \{\mathbf{x}_0, \mathbf{u}_0, ..., \mathbf{x}_{N-1}, \mathbf{u}_{N-1}, \mathbf{x}_N\}$ and

$$\mathbf{g}(\mathbf{w}) = \begin{bmatrix} \mathbf{b}(\mathbf{x}_0, \mathbf{x}_N) \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) - \mathbf{x}_1 \\ \mathbf{f}(\mathbf{x}_1, \mathbf{u}_1) - \mathbf{x}_2 \\ \dots \end{bmatrix}$$



Function $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ is an integration of the ODE over the time interval $[t_k, \ t_{k+1}]$

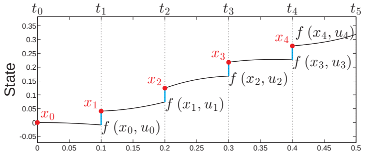
Multiple-Shooting & SQP - Reminder

Multiple shooting yields an NLP:

$$\min_{\mathbf{w}} \quad \Phi(\mathbf{w})$$
$$\text{s.t.} \quad \mathbf{g}(\mathbf{w}) = 0$$

with $\mathbf{w} = \{\mathbf{x}_0, \mathbf{u}_0, \dots, \mathbf{x}_{N-1}, \mathbf{u}_{N-1}, \mathbf{x}_N\}$ and

$$\mathbf{g}(\mathbf{w}) = \begin{bmatrix} \mathbf{b}(\mathbf{x}_0, \mathbf{x}_N) \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) - \mathbf{x}_1 \\ \mathbf{f}(\mathbf{x}_1, \mathbf{u}_1) - \mathbf{x}_2 \\ \dots \end{bmatrix}$$



Function $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ is an integration of the ODE over the time interval $[t_k,\ t_{k+1}]$

SQP, iterate:

1. QP problem (formed at $\mathbf{w}$):

$$\min_{\Delta\mathbf{w}} \quad \frac{1}{2}\Delta\mathbf{w}^\top B\Delta\mathbf{w} + \nabla\Phi^\top\Delta\mathbf{w}$$
$$\text{s.t.} \quad \nabla\mathbf{g}^\top\Delta\mathbf{w} + \mathbf{g} = 0$$

2. $\mathbf{w} \leftarrow \mathbf{w} + \alpha\Delta\mathbf{w}, \quad \alpha \in]0, 1]$

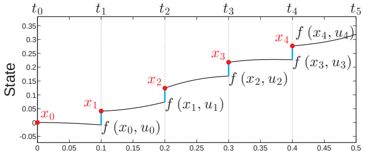
Multiple-Shooting & SQP - Reminder

Multiple shooting yields an NLP:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \Phi(\mathbf{w}) \\ \text{s.t.} \quad & \mathbf{g}(\mathbf{w}) = 0 \end{aligned}$$

with $\mathbf{w} = \{\mathbf{x}_0, \mathbf{u}_0, \dots, \mathbf{x}_{N-1}, \mathbf{u}_{N-1}, \mathbf{x}_N\}$ and

$$\mathbf{g}(\mathbf{w}) = \begin{bmatrix} \mathbf{b}(\mathbf{x}_0, \mathbf{x}_N) \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) - \mathbf{x}_1 \\ \mathbf{f}(\mathbf{x}_1, \mathbf{u}_1) - \mathbf{x}_2 \\ \dots \end{bmatrix}$$



Function $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ is an integration of the ODE over the time interval $[t_k,\ t_{k+1}]$

SQP, iterate:

1. QP problem (formed at $\mathbf{w}$):

$$\begin{aligned} \min_{\Delta\mathbf{w}} \quad & \frac{1}{2}\Delta\mathbf{w}^\top B \Delta\mathbf{w} + \nabla\Phi^\top \Delta\mathbf{w} \\ \text{s.t.} \quad & \nabla\mathbf{g}^\top \Delta\mathbf{w} + \mathbf{g} = 0 \end{aligned}$$

2. $\mathbf{w} \leftarrow \mathbf{w} + \alpha\Delta\mathbf{w}, \quad \alpha \in]0, 1]$

Jacobian $\nabla\mathbf{g}(\mathbf{w})$ requires the linearisation of the integrator, i.e. $\nabla\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$

Constraints Jacobian with Integrators

NLP with multiple-shooting

$$
\begin{aligned}
\min_{\mathbf{w}} \quad & \Phi\left(\mathbf{w}\right) \\
\text{s.t.} \quad & \mathbf{g}\left(\mathbf{w}\right) = \begin{bmatrix} \mathbf{b}\left(\mathbf{x}_0, \mathbf{x}_N\right) \\ \mathbf{f}\left(\mathbf{x}_0, \mathbf{u}_0\right) - \mathbf{x}_1 \\ \mathbf{f}\left(\mathbf{x}_1, \mathbf{u}_1\right) - \mathbf{x}_2 \\ \ldots \end{bmatrix}
\end{aligned}
$$

Constraints Jacobian with Integrators

NLP with multiple-shooting

$$\min_{\mathbf{w}} \quad \Phi(\mathbf{w})$$

$$\text{s.t.} \quad \mathbf{g}(\mathbf{w}) = \begin{bmatrix} \mathbf{b}(\mathbf{x}_0, \mathbf{x}_N) \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) - \mathbf{x}_1 \\ \mathbf{f}(\mathbf{x}_1, \mathbf{u}_1) - \mathbf{x}_2 \\ \ldots \end{bmatrix}$$

where $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ are integrations of the ODE:

$$\dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}(t), \mathbf{u}_k)$$

over the time intervals $[t_k, t_{k+1}]$

Constraints Jacobian with Integrators

NLP with multiple-shooting

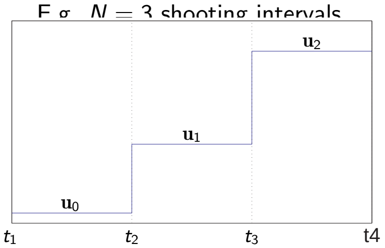
$$\min_{\mathbf{w}} \quad \Phi(\mathbf{w})$$

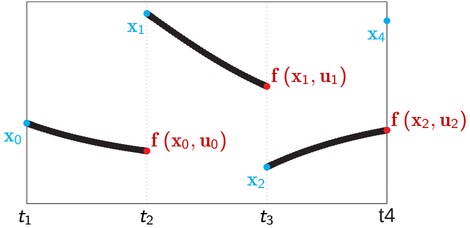
$$\text{s.t.} \quad \mathbf{g}(\mathbf{w}) = \begin{bmatrix} \mathbf{b}(\mathbf{x}_0, \mathbf{x}_N) \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) - \mathbf{x}_1 \\ \mathbf{f}(\mathbf{x}_1, \mathbf{u}_1) - \mathbf{x}_2 \\ \ldots \end{bmatrix}$$

where $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ are integrations of the ODE:

$$\dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}(t), \mathbf{u}_k)$$

over the time intervals $[t_k, t_{k+1}]$





Constraints Jacobian with Integrators

NLP with multiple-shooting

$$\min_{\mathbf{w}} \quad \Phi(\mathbf{w})$$

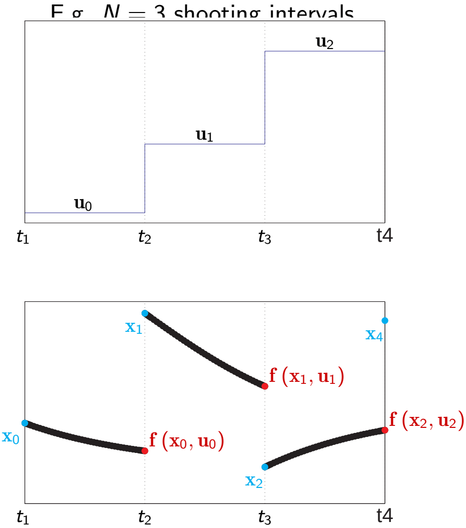
$$\text{s.t.} \quad \mathbf{g}(\mathbf{w}) = \begin{bmatrix} \mathbf{b}(\mathbf{x}_0, \mathbf{x}_N) \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) - \mathbf{x}_1 \\ \mathbf{f}(\mathbf{x}_1, \mathbf{u}_1) - \mathbf{x}_2 \\ \dots \end{bmatrix}$$

where $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ are integrations of the ODE:

$$\dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}(t), \mathbf{u}_k)$$

over the time intervals $[t_k, t_{k+1}]$

NLP solver needs $\nabla_{\mathbf{x}_k}\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ and $\nabla_{\mathbf{u}_k}\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$



Constraints Jacobian with Integrators

NLP with multiple-shooting

$$\min_{\mathbf{w}} \quad \Phi(\mathbf{w})$$

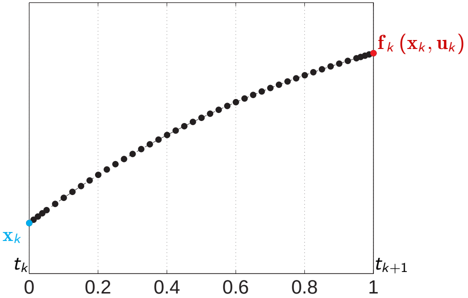
$$\text{s.t.} \quad \mathbf{g}(\mathbf{w}) = \begin{bmatrix} \mathbf{b}(\mathbf{x}_0, \mathbf{x}_N) \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) - \mathbf{x}_1 \\ \mathbf{f}(\mathbf{x}_1, \mathbf{u}_1) - \mathbf{x}_2 \\ \dots \end{bmatrix}$$

where $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ are integrations of the ODE:

$$\dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}(t), \mathbf{u}_k)$$

over the time intervals $[t_k, t_{k+1}]$

NLP solver needs $\nabla_{\mathbf{x}_k}\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ and $\nabla_{\mathbf{u}_k}\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$



Constraints Jacobian with Integrators

NLP with multiple-shooting

$$\min_{\mathbf{w}} \quad \Phi(\mathbf{w})$$

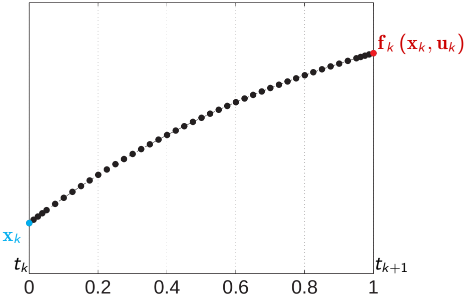
$$\text{s.t.} \quad \mathbf{g}(\mathbf{w}) = \begin{bmatrix} \mathbf{b}(\mathbf{x}_0, \mathbf{x}_N) \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) - \mathbf{x}_1 \\ \mathbf{f}(\mathbf{x}_1, \mathbf{u}_1) - \mathbf{x}_2 \\ \dots \end{bmatrix}$$

where $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ are integrations of the ODE:

$$\dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}(t), \mathbf{u}_k)$$

over the time intervals $[t_k, t_{k+1}]$

NLP solver needs $\nabla_{\mathbf{x}_k}\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ and $\nabla_{\mathbf{u}_k}\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$



How to compute $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ and:

$$\nabla_{\mathbf{x}_k}\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$$

$$\nabla_{\mathbf{u}_k}\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$$

?

Outline

Integrator with sensitivity - Variational approach

Let us define $\mathbf{s}(t)$ the state trajectory over the time interval $[t_k, t_{k+1}]$, i.e.

$$\dot{\mathbf{s}}(t) = \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k) \quad \text{and} \quad \mathbf{s}(t_k) = \mathbf{x}_k$$

then $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}(t_{k+1})$.

Integrator with sensitivity - Variational approach

Let us define $\mathbf{s}(t)$ the state trajectory over the time interval $[t_k, t_{k+1}]$, i.e.

$$\dot{\mathbf{s}}(t) = \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k) \quad \text{and} \quad \mathbf{s}(t_k) = \mathbf{x}_k$$

then $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}(t_{k+1})$.

Define $A(t) = \frac{\partial \mathbf{s}(t)}{\partial \mathbf{x}_k}$, such that $\nabla_{\mathbf{x}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)^\top = A(t_{k+1})$. Moreover:

Integrator with sensitivity - Variational approach

Let us define $\mathbf{s}(t)$ the state trajectory over the time interval $[t_k, t_{k+1}]$, i.e.

$$\dot{\mathbf{s}}(t) = \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k) \quad \text{and} \quad \mathbf{s}(t_k) = \mathbf{x}_k$$

then $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}(t_{k+1})$.

Define $A(t) = \frac{\partial \mathbf{s}(t)}{\partial \mathbf{x}_k}$, such that $\nabla_{\mathbf{x}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)^\top = A(t_{k+1})$. Moreover:

$$\dot{A}(t) = \frac{d}{dt} \frac{\partial \mathbf{s}(t)}{\partial \mathbf{x}_k}$$

Integrator with sensitivity - Variational approach

Let us define $\mathbf{s}(t)$ the state trajectory over the time interval $[t_k, t_{k+1}]$, i.e.

$$\dot{\mathbf{s}}(t) = \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k) \quad \text{and} \quad \mathbf{s}(t_k) = \mathbf{x}_k$$

then $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}(t_{k+1})$.

Define $A(t) = \frac{\partial \mathbf{s}(t)}{\partial \mathbf{x}_k}$, such that $\nabla_{\mathbf{x}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)^\top = A(t_{k+1})$. Moreover:

$$\dot{A}(t) = \frac{d}{dt} \frac{\partial \mathbf{s}(t)}{\partial \mathbf{x}_k} = \frac{\partial \dot{\mathbf{s}}(t)}{\partial \mathbf{x}_k}$$

Integrator with sensitivity - Variational approach

Let us define $\mathbf{s}(t)$ the state trajectory over the time interval $[t_k, t_{k+1}]$, i.e.

$$\dot{\mathbf{s}}(t) = \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k) \quad \text{and} \quad \mathbf{s}(t_k) = \mathbf{x}_k$$

then $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}(t_{k+1})$.

Define $A(t) = \frac{\partial \mathbf{s}(t)}{\partial \mathbf{x}_k}$, such that $\nabla_{\mathbf{x}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)^\top = A(t_{k+1})$. Moreover:

$$\dot{A}(t) = \frac{d}{dt} \frac{\partial \mathbf{s}(t)}{\partial \mathbf{x}_k} = \frac{\partial \dot{\mathbf{s}}(t)}{\partial \mathbf{x}_k} = \frac{\partial \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k)}{\partial \mathbf{x}_k}$$

Integrator with sensitivity - Variational approach

Let us define $\mathbf{s}(t)$ the state trajectory over the time interval $[t_k, t_{k+1}]$, i.e.

$$\dot{\mathbf{s}}(t) = \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k) \quad \text{and} \quad \mathbf{s}(t_k) = \mathbf{x}_k$$

then $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}(t_{k+1})$.

Define $A(t) = \frac{\partial \mathbf{s}(t)}{\partial \mathbf{x}_k}$, such that $\nabla_{\mathbf{x}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)^\top = A(t_{k+1})$. Moreover:

$$\dot{A}(t) = \frac{d}{dt}\frac{\partial \mathbf{s}(t)}{\partial \mathbf{x}_k} = \frac{\partial \dot{\mathbf{s}}(t)}{\partial \mathbf{x}_k} = \frac{\partial \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k)}{\partial \mathbf{x}_k} = \frac{\partial \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k)}{\partial \mathbf{s}(t)}\frac{\partial \mathbf{s}(t)}{\partial \mathbf{x}_k}$$

Integrator with sensitivity - Variational approach

Let us define $\mathrm{s}(t)$ the state trajectory over the time interval $[t_k, t_{k+1}]$, i.e.

$$\dot{\mathrm{s}}(t) = \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k) \quad \text{and} \quad \mathrm{s}(t_k) = \mathbf{x}_k$$

then $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathrm{s}(t_{k+1})$.

Define $A(t) = \frac{\partial \mathrm{s}(t)}{\partial \mathbf{x}_k}$, such that $\nabla_{\mathbf{x}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)^\top = A(t_{k+1})$. Moreover:

$$\begin{aligned}
\dot{A}(t) &= \frac{d}{dt}\frac{\partial \mathrm{s}(t)}{\partial \mathbf{x}_k} = \frac{\partial \dot{\mathrm{s}}(t)}{\partial \mathbf{x}_k} = \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathbf{x}_k} = \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathrm{s}(t)} \frac{\partial \mathrm{s}(t)}{\partial \mathbf{x}_k} \\
&= \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathrm{s}(t)} A(t)
\end{aligned}$$

Integrator with sensitivity - Variational approach

Let us define $\mathrm{s}(t)$ the state trajectory over the time interval $[t_k, t_{k+1}]$, i.e.

$$\dot{\mathrm{s}}(t) = \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k) \quad \text{and} \quad \mathrm{s}(t_k) = \mathbf{x}_k$$

then $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathrm{s}(t_{k+1})$.

Define $A(t) = \frac{\partial \mathrm{s}(t)}{\partial \mathbf{x}_k}$, such that $\nabla_{\mathbf{x}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)^\top = A(t_{k+1})$. Moreover:

$$\begin{aligned}\dot{A}(t) &= \frac{d}{dt} \frac{\partial \mathrm{s}(t)}{\partial \mathbf{x}_k} = \frac{\partial \dot{\mathrm{s}}(t)}{\partial \mathbf{x}_k} = \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathbf{x}_k} = \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathrm{s}(t)} \frac{\partial \mathrm{s}(t)}{\partial \mathbf{x}_k} \\ &= \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathrm{s}(t)} A(t)\end{aligned}$$

with the initial conditions $A(t_k) = \frac{\partial \mathrm{s}(t_k)}{\partial \mathbf{x}_k} = \frac{\partial \mathbf{x}_k}{\partial \mathbf{x}_k} = I$.

Integrator with sensitivity - Variational approach

Let us define $\mathrm{s}(t)$ the state trajectory over the time interval $[t_k, t_{k+1}]$, i.e.

$$\dot{\mathrm{s}}(t) = \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k) \quad \text{and} \quad \mathrm{s}(t_k) = \mathbf{x}_k$$

then $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathrm{s}(t_{k+1})$.

Define $A(t) = \frac{\partial \mathrm{s}(t)}{\partial \mathbf{x}_k}$, such that $\nabla_{\mathbf{x}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)^\top = A(t_{k+1})$. Moreover:

$$\begin{aligned}
\dot{A}(t) &= \frac{d}{dt}\frac{\partial \mathrm{s}(t)}{\partial \mathbf{x}_k} = \frac{\partial \dot{\mathrm{s}}(t)}{\partial \mathbf{x}_k} = \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathbf{x}_k} = \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathrm{s}(t)}\frac{\partial \mathrm{s}(t)}{\partial \mathbf{x}_k} \\
&= \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathrm{s}(t)} A(t)
\end{aligned}$$

with the initial conditions $A(t_k) = \frac{\partial \mathrm{s}(t_k)}{\partial \mathbf{x}_k} = \frac{\partial \mathbf{x}_k}{\partial \mathbf{x}_k} = I$.

Then

$$\boxed{\begin{array}{rcll}
\nabla_{\mathbf{x}_x} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)^\top &=& A(t_{k+1}) & \text{with} \\
\dot{\mathrm{s}}(t) &=& \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k), & \mathrm{s}(t_k) = \mathbf{x}_k \\
\dot{A}(t) &=& \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathrm{s}(t)} A(t), & A(t_k) = I
\end{array}}$$

Integrator with sensitivity - Variational approach (cont')

Let us define $\mathbf{s}(t)$ the state trajectory over the time interval $[t_k, t_{k+1}]$, i.e.

$$\dot{\mathbf{s}}(t) = \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k) \quad \text{and} \quad \mathbf{s}(t_k) = \mathbf{x}_k$$

then $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}(t_{k+1})$.

Integrator with sensitivity - Variational approach (cont')

Let us define $\mathbf{s}(t)$ the state trajectory over the time interval $[t_k, t_{k+1}]$, i.e.

$$\dot{\mathbf{s}}(t) = \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k) \quad \text{and} \quad \mathbf{s}(t_k) = \mathbf{x}_k$$

then $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}(t_{k+1})$.

Define $B(t) = \frac{\partial \mathbf{s}(t)}{\partial \mathbf{u}_k}$, such that $\nabla_{\mathbf{u}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)^\top = B(t_{k+1})$.

Integrator with sensitivity - Variational approach (cont')

Let us define $\mathbf{s}(t)$ the state trajectory over the time interval $[t_k, t_{k+1}]$, i.e.

$$\dot{\mathbf{s}}(t) = \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k) \quad \text{and} \quad \mathbf{s}(t_k) = \mathbf{x}_k$$

then $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}(t_{k+1})$.

Define $B(t) = \frac{\partial \mathbf{s}(t)}{\partial \mathbf{u}_k}$, such that $\nabla_{\mathbf{u}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)^\top = B(t_{k+1})$. Moreover:

$$\dot{B}(t) = \frac{d}{dt} \frac{\partial \mathbf{s}(t)}{\partial \mathbf{u}_k}$$

Integrator with sensitivity - Variational approach (cont')

Let us define $\mathbf{s}(t)$ the state trajectory over the time interval $[t_k, t_{k+1}]$, i.e.

$$\dot{\mathbf{s}}(t) = \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k) \quad \text{and} \quad \mathbf{s}(t_k) = \mathbf{x}_k$$

then $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}(t_{k+1})$.

Define $B(t) = \frac{\partial \mathbf{s}(t)}{\partial \mathbf{u}_k}$, such that $\nabla_{\mathbf{u}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)^\top = B(t_{k+1})$. Moreover:

$$\dot{B}(t) = \frac{d}{dt} \frac{\partial \mathbf{s}(t)}{\partial \mathbf{u}_k} = \frac{\partial \dot{\mathbf{s}}(t)}{\partial \mathbf{u}_k}$$

Integrator with sensitivity - Variational approach (cont')

Let us define $\mathrm{s}(t)$ the state trajectory over the time interval $[t_k, t_{k+1}]$, i.e.

$$\dot{\mathrm{s}}(t) = \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k) \quad \text{and} \quad \mathrm{s}(t_k) = \mathbf{x}_k$$

then $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathrm{s}(t_{k+1})$.

Define $B(t) = \frac{\partial \mathrm{s}(t)}{\partial \mathbf{u}_k}$, such that $\nabla_{\mathbf{u}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)^\top = B(t_{k+1})$. Moreover:

$$\dot{B}(t) = \frac{d}{dt} \frac{\partial \mathrm{s}(t)}{\partial \mathbf{u}_k} = \frac{\partial \dot{\mathrm{s}}(t)}{\partial \mathbf{u}_k} = \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathbf{u}_k} + \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathrm{s}(t)} \frac{\partial \mathrm{s}(t)}{\partial \mathbf{u}_k}$$

Integrator with sensitivity - Variational approach (cont')

Let us define $\mathrm{s}(t)$ the state trajectory over the time interval $[t_k, t_{k+1}]$, i.e.

$$\dot{\mathrm{s}}(t) = \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k) \quad \text{and} \quad \mathrm{s}(t_k) = \mathbf{x}_k$$

then $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathrm{s}(t_{k+1})$.

Define $B(t) = \frac{\partial \mathrm{s}(t)}{\partial \mathbf{u}_k}$, such that $\nabla_{\mathbf{u}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)^\top = B(t_{k+1})$. Moreover:

$$\begin{aligned}\dot{B}(t) &= \frac{d}{dt}\frac{\partial \mathrm{s}(t)}{\partial \mathbf{u}_k} = \frac{\partial \dot{\mathrm{s}}(t)}{\partial \mathbf{u}_k} = \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathbf{u}_k} + \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathrm{s}(t)}\frac{\partial \mathrm{s}(t)}{\partial \mathbf{u}_k} \\ &= \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathbf{u}_k} + \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathrm{s}(t)} B(t)\end{aligned}$$

Integrator with sensitivity - Variational approach (cont')

Let us define $\mathrm{s}(t)$ the state trajectory over the time interval $[t_k, t_{k+1}]$, i.e.

$$\dot{\mathrm{s}}(t) = \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k) \quad \text{and} \quad \mathrm{s}(t_k) = \mathbf{x}_k$$

then $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathrm{s}(t_{k+1})$.

Define $B(t) = \frac{\partial \mathrm{s}(t)}{\partial \mathbf{u}_k}$, such that $\nabla_{\mathbf{u}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)^\top = B(t_{k+1})$. Moreover:

$$\begin{aligned}\dot{B}(t) &= \frac{d}{dt}\frac{\partial \mathrm{s}(t)}{\partial \mathbf{u}_k} = \frac{\partial \dot{\mathrm{s}}(t)}{\partial \mathbf{u}_k} = \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathbf{u}_k} + \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathrm{s}(t)}\frac{\partial \mathrm{s}(t)}{\partial \mathbf{u}_k} \\ &= \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathbf{u}_k} + \frac{\partial \mathbf{F}(\mathrm{s}(t), \mathbf{u}_k)}{\partial \mathrm{s}(t)} B(t)\end{aligned}$$

with the initial conditions $B(t_k) = 0$

Integrator with sensitivity - Variational approach (cont')

Let us define $\mathbf{s}(t)$ the state trajectory over the time interval $[t_k, t_{k+1}]$, i.e.

$$\dot{\mathbf{s}}(t) = \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k) \quad \text{and} \quad \mathbf{s}(t_k) = \mathbf{x}_k$$

then $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}(t_{k+1})$.

Define $B(t) = \frac{\partial \mathbf{s}(t)}{\partial \mathbf{u}_k}$, such that $\nabla_{\mathbf{u}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)^\top = B(t_{k+1})$. Moreover:

$$\begin{aligned}
\dot{B}(t) = \frac{d}{dt}\frac{\partial \mathbf{s}(t)}{\partial \mathbf{u}_k} = \frac{\partial \dot{\mathbf{s}}(t)}{\partial \mathbf{u}_k} &= \frac{\partial \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k)}{\partial \mathbf{u}_k} + \frac{\partial \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k)}{\partial \mathbf{s}(t)}\frac{\partial \mathbf{s}(t)}{\partial \mathbf{u}_k} \\
&= \frac{\partial \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k)}{\partial \mathbf{u}_k} + \frac{\partial \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k)}{\partial \mathbf{s}(t)} B(t)
\end{aligned}$$

with the initial conditions $B(t_k) = 0$

Then

$$\boxed{\begin{aligned}
\nabla_{\mathbf{u}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)^\top &= B(t_{k+1}) \qquad \text{with} \\
\dot{\mathbf{s}}(t) &= \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k), \quad \mathbf{s}(t_k) = \mathbf{x}_k \\
\dot{B}(t) &= \frac{\partial \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k)}{\partial \mathbf{u}_k} + \frac{\partial \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k)}{\partial \mathbf{s}(t)} B(t), \quad B(t_k) = 0
\end{aligned}}$$

Integrator with sensitivity - Variational approach (cont')

Integrate forward over the time interval $[t_k,\ t_{k+1}]$:

$$
\begin{array}{llll}
\text{State simulation:} & \dot{\mathbf{s}}(t) = \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k), & & \mathbf{s}(t_k) = \mathbf{x}_k \\
\text{State sensitivity:} & \dot{A}(t) = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}(t),\mathbf{u}_k} A(t), & & A(t_k) = I \\
\text{Input sensitivity:} & \dot{B}(t) = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}(t),\mathbf{u}_k} B(t) + \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}(t),\mathbf{u}_k}, & & B(t_k) = 0
\end{array}
$$

Note: read $\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}(t),\mathbf{u}_k} = \frac{\partial \mathbf{F}(\mathbf{s}(t),\mathbf{u}_k)}{\partial \mathbf{s}(t)}$ and $\left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}(t),\mathbf{u}_k} = \frac{\partial \mathbf{F}(\mathbf{s}(t),\mathbf{u}_k)}{\partial \mathbf{u}_k}$

Integrator with sensitivity - Variational approach (cont')

Integrate forward over the time interval $[t_k,\ t_{k+1}]$:

State simulation:	$\dot{\mathbf{s}}(t)$	$=$	$\mathbf{F}(\mathbf{s}(t), \mathbf{u}_k)$,	$\mathbf{s}(t_k) = \mathbf{x}_k$
State sensitivity:	$\dot{A}(t)$	$=$	$\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right\|_{\mathbf{s}(t),\mathbf{u}_k} A(t)$,	$A(t_k) = I$
Input sensitivity:	$\dot{B}(t)$	$=$	$\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right\|_{\mathbf{s}(t),\mathbf{u}_k} B(t) + \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right\|_{\mathbf{s}(t),\mathbf{u}_k}$,	$B(t_k) = 0$

Note: read $\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}(t),\mathbf{u}_k} = \frac{\partial \mathbf{F}(\mathbf{s}(t),\mathbf{u}_k)}{\partial \mathbf{s}(t)}$ and $\left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}(t),\mathbf{u}_k} = \frac{\partial \mathbf{F}(\mathbf{s}(t),\mathbf{u}_k)}{\partial \mathbf{u}_k}$

Integrator evaluation with sensitivities

$$\begin{aligned}
\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) &= \mathbf{s}(t_{k+1}), \\
\nabla_{\mathbf{x}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) &= A(t_{k+1})^\top, \\
\nabla_{\mathbf{u}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) &= B(t_{k+1})^\top,
\end{aligned}$$

Integrator with sensitivity - Variational approach (cont')

Integrate forward over the time interval $[t_k,\ t_{k+1}]$:

$$\begin{array}{llcll} \text{State simulation:} & \dot{\mathbf{s}}(t) & = & \mathbf{F}(\mathbf{s}(t), \mathbf{u}_k), & \mathbf{s}(t_k) = \mathbf{x}_k \\ \text{State sensitivity:} & \dot{A}(t) & = & \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}(t),\mathbf{u}_k} A(t), & A(t_k) = I \\ \text{Input sensitivity:} & \dot{B}(t) & = & \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}(t),\mathbf{u}_k} B(t) + \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}(t),\mathbf{u}_k}, & B(t_k) = 0 \end{array}$$

Note: read $\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}(t),\mathbf{u}_k} = \frac{\partial \mathbf{F}(\mathbf{s}(t),\mathbf{u}_k)}{\partial \mathbf{s}(t)}$ and $\left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}(t),\mathbf{u}_k} = \frac{\partial \mathbf{F}(\mathbf{s}(t),\mathbf{u}_k)}{\partial \mathbf{u}_k}$

Integrator evaluation with sensitivities

$$\begin{aligned} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) &= \mathbf{s}(t_{k+1}), \\ \nabla_{\mathbf{x}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) &= A(t_{k+1})^\top, \\ \nabla_{\mathbf{u}_k} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) &= B(t_{k+1})^\top, \end{aligned}$$

Pros: can use your favourite integrator to handle $\dot{\mathbf{s}}$, $\dot{A}$, $\dot{B}$ jointly

Cons: inexact derivative (because of inexact integration) !!

First differentiate, then discretize (i.e. "integrate")

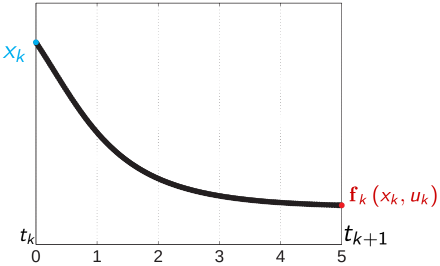
Integrator with sensitivity - Variational approach, example

ODE: $\dot{x} = -\sin(x) + u$

Integrator with sensitivity - Variational approach, example

ODE: $\dot{x} = -\sin(x) + u$

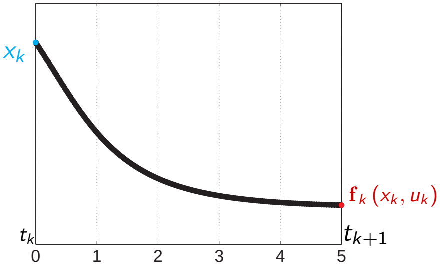
Using ode45.m, integrate the ODE over a time interval $t_{k+1} - t_k = 5\,\mathrm{s}$



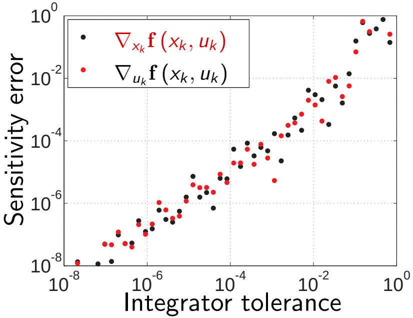
Integrator with sensitivity - Variational approach, example

ODE: $\dot{x} = -\sin(x) + u$

Using ode45.m, integrate the ODE over a time interval $t_{k+1} - t_k = 5\,\mathrm{s}$



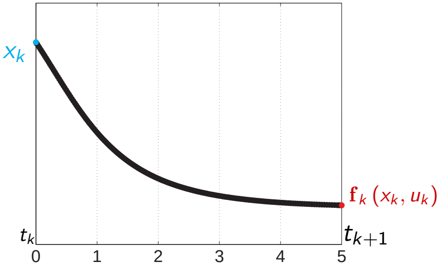
Error in the sensitivities (baseline is numerical differentiation):



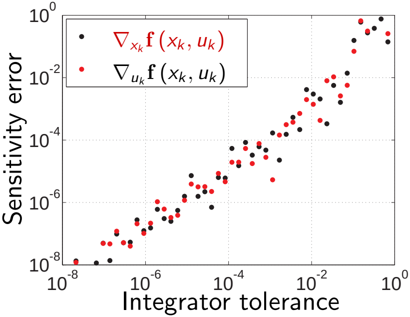
Integrator with sensitivity - Variational approach, example

ODE: $\dot{x} = -\sin(x) + u$

Using ode45.m, integrate the ODE over a time interval $t_{k+1} - t_k = 5\,\mathrm{s}$



Error in the sensitivities (baseline is numerical differentiation):



An NLP solver fed with inaccurate derivatives will struggle when reaching a KKT residual in the range of the derivative inaccuracy. If using a variational approach, **make sure that the integrator tolerance $<<$ tolerance of the NLP solver** (at least 1 order of magnitude)

Outline

Integrator with sensitivity - Algorithmic Differentiation

Key idea: first descritize, then differentiate

Algorithm: Explicit Euler

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k$

for $i = 0 : N-1$ **do**

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$

return $\quad \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}$

Integrator with sensitivity - Algorithmic Differentiation

Key idea: first descritize, then differentiate

Algorithm: Explicit Euler

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

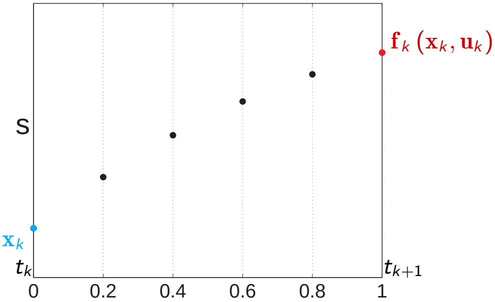
$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k$

for $i = 0 : N-1$ **do**

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}(\mathbf{s}, \mathbf{u}_k)$

return $\quad \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}$

E.g. with $N = 5$, $\Delta t = 1$



Integrator with sensitivity - Algorithmic Differentiation

Key idea: first descritize, then differentiate

Algorithm: Explicit Euler

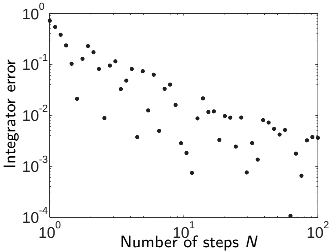
Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k$

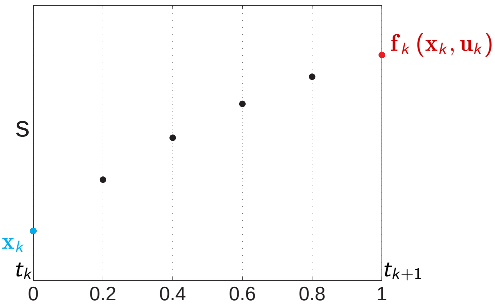
for $i = 0 : N-1$ **do**

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$

return $\quad \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}$



E.g. with $N = 5$, $\Delta t = 1$



Integrator with sensitivity - Algorithmic Differentiation

Key idea: first descritize, then differentiate

Algorithm: Explicit Euler

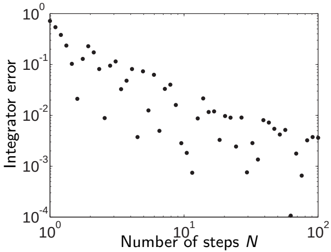
Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k$

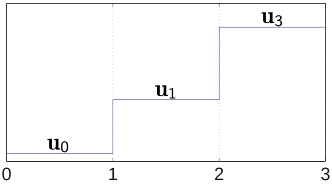
for $i = 0 : N-1$ **do**

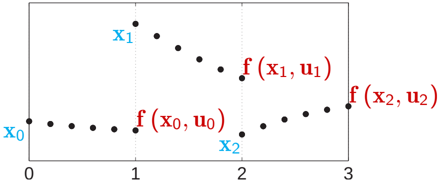
$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$

return $\quad \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}$



E.g. $N = 5$, with 3 shooting intervals





Integrator with sensitivity - Algorithmic Differentiation

Key idea: first descritize, then differentiate

Algorithm: Explicit Euler

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k$

for $i = 0 : N-1$ **do**

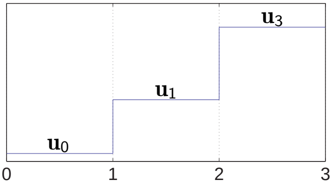
$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$

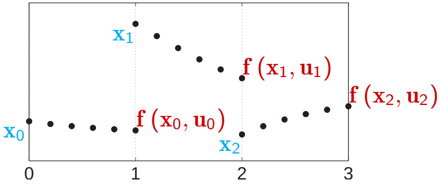
return $\mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}$

Reminder:

$$\mathbf{g}\left(\mathbf{w}\right) = \begin{bmatrix} \mathbf{b}\left(\mathbf{x}_0, \mathbf{x}_N\right) \\ \mathbf{x}_1 - \mathbf{f}\left(\mathbf{x}_0, \mathbf{u}_0\right) \\ \dots \\ \mathbf{x}_N - \mathbf{f}\left(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}\right) \end{bmatrix}$$

E.g. $N = 5$, with 3 shooting intervals





Integrator with sensitivity - Algorithmic Differentiation

Let's differentiate Explicit Euler

Key idea: first descritize, then differentiate

Algorithm: Explicit Euler

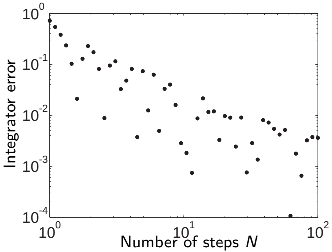
Input: $\mathbf{x}_k,\ \mathbf{u}_k,\ [t_k,\ t_{k+1}],\ N$

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k$

for $i = 0 : N-1$ **do**

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}(\mathbf{s}, \mathbf{u}_k)$

return $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}$



Algorithmic Differentiation

Algorithm: Explicit Euler with forward sensitivities

Input: $\mathbf{x}_k,\ \mathbf{u}_k,\ [t_k,\ t_{k+1}],\ N$

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k$

for $i = 0 : N-1$ **do**

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}(\mathbf{s}, \mathbf{u}_k)$

return

$\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}$

Integrator with sensitivity - Algorithmic Differentiation

Let's differentiate Explicit Euler

Key idea: first descritize, then differentiate

Algorithm: Explicit Euler

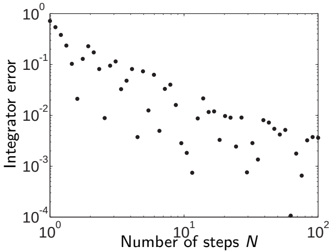
Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k$

for $i = 0 : N-1$ **do**

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}(\mathbf{s}, \mathbf{u}_k)$

return $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}$



Algorithmic Differentiation

Algorithm: Explicit Euler with forward sensitivities

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k, \quad A = I$

for $i = 0 : N-1$ **do**

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}(\mathbf{s}, \mathbf{u}_k)$

return

$\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}$

$$A = \frac{\partial \mathbf{s}}{\partial \mathbf{x}_k}$$

Let's differentiate Explicit Euler

Key idea: first descritize, then differentiate

Algorithm: Explicit Euler

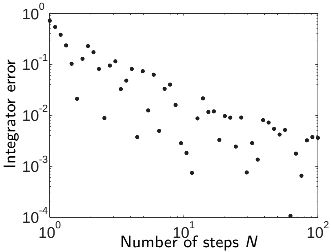
Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k$

for $i = 0 : N-1$ **do**

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$

return $\mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}$



Algorithmic Differentiation

Algorithm: Explicit Euler with forward sensitivities

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k, \quad A = I$

for $i = 0 : N-1$ **do**

$\quad A \leftarrow \left(I + \frac{\Delta t}{N} \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s},\mathbf{u}_k}\right) A$

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$

return

$\mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}$

$$A = \frac{\partial \mathbf{s}}{\partial \mathbf{x}_k}$$

Integrator with sensitivity - Algorithmic Differentiation

Let's differentiate Explicit Euler

Key idea: first descritize, then differentiate

Algorithm: Explicit Euler

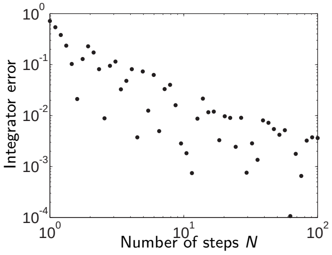
Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k$

for $i = 0 : N-1$ **do**

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$

return $\quad \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}$



Algorithmic Differentiation

Algorithm: Explicit Euler with forward sensitivities

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k, \quad A = I$

for $i = 0 : N-1$ **do**

$\quad A \leftarrow \left(I + \frac{\Delta t}{N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s},\mathbf{u}_k}\right)A$

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$

return

$\mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s},\ \nabla_{\mathbf{x}_k}\mathbf{f}_k = A^\top$

$$A = \frac{\partial \mathbf{s}}{\partial \mathbf{x}_k}$$

Integrator with sensitivity - Algorithmic Differentiation

Let's differentiate Explicit Euler

Key idea: first descritize, then differentiate

Algorithm: Explicit Euler

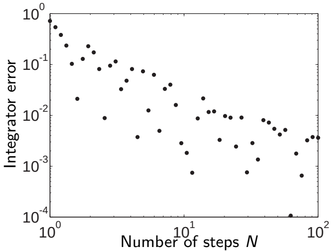
Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k$

for $i = 0 : N-1$ **do**

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}(\mathbf{s}, \mathbf{u}_k)$

return $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}$



Algorithmic Differentiation

Algorithm: Explicit Euler with forward sensitivities

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k, \quad A = I, \quad B = 0$

for $i = 0 : N-1$ **do**

$\quad A \leftarrow \left(I + \frac{\Delta t}{N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s},\mathbf{u}_k}\right) A$

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}(\mathbf{s}, \mathbf{u}_k)$

return

$\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s},\ \nabla_{\mathbf{x}_k}\mathbf{f}_k = A^\top$

$$A = \frac{\partial \mathbf{s}}{\partial \mathbf{x}_k}, \qquad B = \frac{\partial \mathbf{s}}{\partial \mathbf{u}_k}$$

Integrator with sensitivity - Algorithmic Differentiation

Let's differentiate Explicit Euler

Key idea: first descritize, then differentiate

Algorithm: Explicit Euler

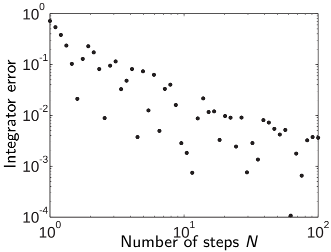
Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k$

for $i = 0 : N-1$ **do**

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$

return $\quad \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}$



Algorithmic Differentiation

Algorithm: Explicit Euler with forward sensitivities

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k, \quad A = I, \quad B = 0$

for $i = 0 : N-1$ **do**

$\quad A \leftarrow \left(I + \frac{\Delta t}{N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s},\mathbf{u}_k}\right) A$

$\quad B \leftarrow \left(I + \frac{\Delta t}{N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s},\mathbf{u}_k}\right) B + \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}_k}\right|_{\mathbf{s},\mathbf{u}_k}$

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$

return

$\quad \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}, \ \nabla_{\mathbf{x}_k}\mathbf{f}_k = A^\top$

$$A = \frac{\partial \mathbf{s}}{\partial \mathbf{x}_k}, \qquad B = \frac{\partial \mathbf{s}}{\partial \mathbf{u}_k}$$

Integrator with sensitivity - Algorithmic Differentiation

Let's differentiate Explicit Euler

Key idea: first descritize, then differentiate

Algorithm: Explicit Euler

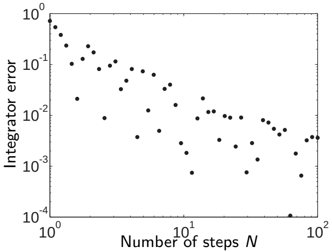
Input: $\mathbf{x}_k,\ \mathbf{u}_k,\ [t_k,\ t_{k+1}],\ N$

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k$

for $i = 0 : N-1$ **do**

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}(\mathbf{s}, \mathbf{u}_k)$

return $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}$



Algorithmic Differentiation

Algorithm: Explicit Euler with forward sensitivities

Input: $\mathbf{x}_k,\ \mathbf{u}_k,\ [t_k,\ t_{k+1}],\ N$

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k, \quad A = I, \quad B = 0$

for $i = 0 : N-1$ **do**

$$A \leftarrow \left(I + \frac{\Delta t}{N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s},\mathbf{u}_k}\right) A$$

$$B \leftarrow \left(I + \frac{\Delta t}{N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s},\mathbf{u}_k}\right) B + \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}_k}\right|_{\mathbf{s},\mathbf{u}_k}$$

$$\mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}(\mathbf{s}, \mathbf{u}_k)$$

return

$\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s},\ \nabla_{\mathbf{x}_k}\mathbf{f}_k = A^\top,\ \nabla_{\mathbf{u}_k}\mathbf{f}_k = B^\top$

$$A = \frac{\partial \mathbf{s}}{\partial \mathbf{x}_k}, \qquad B = \frac{\partial \mathbf{s}}{\partial \mathbf{u}_k}$$

Integrator with sensitivity - Algorithmic Differentiation

Let's differentiate Explicit Euler

Key idea: first descritize, then differentiate

Algorithm: Explicit Euler

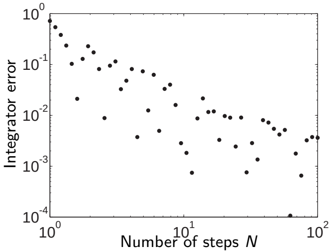
Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k$

for $i = 0 : N-1$ **do**

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}(\mathbf{s}, \mathbf{u}_k)$

return $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}$



Algorithmic Differentiation

Algorithm: Explicit Euler with forward sensitivities

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad \Delta t = t_{k+1} - t_k, \quad A = I, \quad B = 0$

for $i = 0 : N-1$ **do**

$\quad A \leftarrow \left(I + \frac{\Delta t}{N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s},\mathbf{u}_k}\right) A$

$\quad B \leftarrow \left(I + \frac{\Delta t}{N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s},\mathbf{u}_k}\right) B + \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}_k}\right|_{\mathbf{s},\mathbf{u}_k}$

$\quad \mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{N}\mathbf{F}(\mathbf{s}, \mathbf{u}_k)$

return

$\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s},\ \nabla_{\mathbf{x}_k}\mathbf{f}_k = A^\top,\ \nabla_{\mathbf{u}_k}\mathbf{f}_k = B^\top$

Pros: "exact" derivatives, extremely simple code

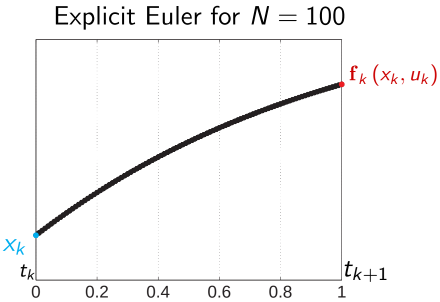
Cons: explicit 1^{st}-order has a poor computational efficiency (flops vs. accuracy)

Integrator with sensitivity - Euler Explicit, example

ODE: $\dot{x} = -\sin(x) + u$

Integrator with sensitivity - Euler Explicit, example

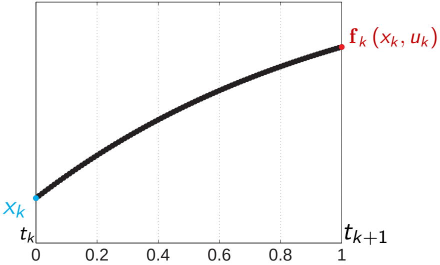
ODE: $\dot{x} = -\sin(x) + u$



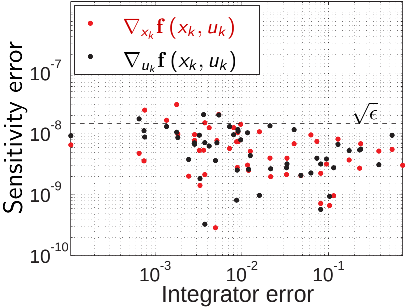
Integrator with sensitivity - Euler Explicit, example

ODE: $\dot{x} = -\sin(x) + u$

Explicit Euler for $N = 100$



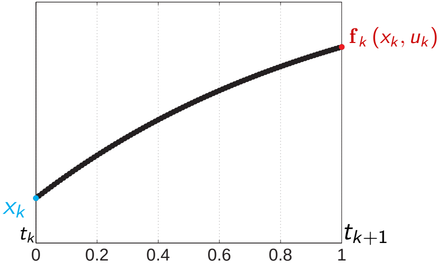
Error in the sensitivities (baseline is numerical differentiation):



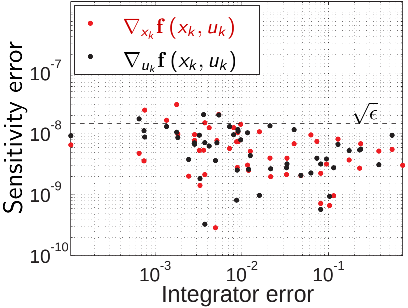
Integrator with sensitivity - Euler Explicit, example

ODE: $\dot{x} = -\sin(x) + u$

Explicit Euler for $N = 100$



Error in the sensitivities (baseline is numerical differentiation):



Algorithmic Differentiation (AD) allows for working with **derivatives at machine precision** (i.e. $\epsilon = 10^{-16}$), regardless of how accurate the integrator is. AD is a generic principle that can be deployed on **any (locally) smooth algorithm**, i.e. one can use it on more efficient integrators than Explicit Euler.

The ERK4 - Explicit Runge-Kutta of 4^{th}-order

Algorithm: ERK4 with sensitivities

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k$

for $i = 0 : N$ **do**

$$\mathbf{k}_1 := \mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$$

$$\mathbf{k}_2 := \mathbf{F}\left(\mathbf{s} + \tfrac{\Delta t}{2N}\mathbf{k}_1, \mathbf{u}_k\right)$$

$$\mathbf{k}_3 := \mathbf{F}\left(\mathbf{s} + \tfrac{\Delta t}{2N}\mathbf{k}_2, \mathbf{u}_k\right)$$

$$\mathbf{k}_4 := \mathbf{F}\left(\mathbf{s} + \tfrac{\Delta t}{N}\mathbf{k}_3, \mathbf{u}_k\right)$$

$$\mathbf{s} \leftarrow \mathbf{s} + \tfrac{\Delta t}{6N}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right)$$

return $\mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}$

The ERK4 - Explicit Runge-Kutta of 4^{th}-order

Algorithm: ERK4 with sensitivities

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k$

for $i = 0 : N$ **do**

$$\mathbf{k}_1 := \mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s},\mathbf{u}_k} \qquad \frac{\partial \mathbf{k}_1}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s},\mathbf{u}_k}$$

$$\mathbf{k}_2 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{2N}\mathbf{k}_1, \mathbf{u}_k\right)$$

$$\mathbf{k}_3 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{2N}\mathbf{k}_2, \mathbf{u}_k\right)$$

$$\mathbf{k}_4 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{N}\mathbf{k}_3, \mathbf{u}_k\right)$$

$$\mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{6N}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right)$$

return $\mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}$

The ERK4 - Explicit Runge-Kutta of 4^{th}-order

Algorithm: ERK4 with sensitivities

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k$

for $i = 0 : N$ **do**

$$\mathbf{k}_1 := \mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s},\mathbf{u}_k} \qquad \frac{\partial \mathbf{k}_1}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s},\mathbf{u}_k}$$

$$\mathbf{k}_2 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{2N}\mathbf{k}_1, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_2}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k} \left(I + \frac{\Delta t}{2N}\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_2}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k} + \frac{\Delta t}{2N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k}\frac{\partial \mathbf{k}_1}{\partial \mathbf{u_k}}$$

$$\mathbf{k}_3 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{2N}\mathbf{k}_2, \mathbf{u}_k\right)$$

$$\mathbf{k}_4 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{N}\mathbf{k}_3, \mathbf{u}_k\right)$$

$$\mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{6N}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right)$$

return $\mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}$

The ERK4 - Explicit Runge-Kutta of 4^{th}-order

Algorithm: ERK4 with sensitivities

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k$

for $i = 0 : N$ **do**

$$\mathbf{k}_1 := \mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s},\mathbf{u}_k} \qquad \frac{\partial \mathbf{k}_1}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s},\mathbf{u}_k}$$

$$\mathbf{k}_2 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{2N}\mathbf{k}_1, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_2}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k} \left(I + \frac{\Delta t}{2N}\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_2}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k} + \frac{\Delta t}{2N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k}\frac{\partial \mathbf{k}_1}{\partial \mathbf{u_k}}$$

$$\mathbf{k}_3 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{2N}\mathbf{k}_2, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_3}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k} \left(I + \frac{\Delta t}{2N}\frac{\partial \mathbf{k}_2}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_3}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k} + \frac{\Delta t}{2N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k}\frac{\partial \mathbf{k}_2}{\partial \mathbf{u_k}}$$

$$\mathbf{k}_4 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{N}\mathbf{k}_3, \mathbf{u}_k\right)$$

$$\mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{6N}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right)$$

return $\mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}$

The ERK4 - Explicit Runge-Kutta of 4^{th}-order

Algorithm: ERK4 with sensitivities

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k$

for $i = 0 : N$ **do**

$$\mathbf{k}_1 := \mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s},\mathbf{u}_k} \qquad \frac{\partial \mathbf{k}_1}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s},\mathbf{u}_k}$$

$$\mathbf{k}_2 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{2N}\mathbf{k}_1, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_2}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k}\left(I + \frac{\Delta t}{2N}\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_2}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k} + \frac{\Delta t}{2N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k}\frac{\partial \mathbf{k}_1}{\partial \mathbf{u_k}}$$

$$\mathbf{k}_3 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{2N}\mathbf{k}_2, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_3}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k}\left(I + \frac{\Delta t}{2N}\frac{\partial \mathbf{k}_2}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_3}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k} + \frac{\Delta t}{2N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k}\frac{\partial \mathbf{k}_2}{\partial \mathbf{u_k}}$$

$$\mathbf{k}_4 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{N}\mathbf{k}_3, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_4}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{N}\mathbf{k}_3,\mathbf{u}_k}\left(I + \frac{\Delta t}{N}\frac{\partial \mathbf{k}_3}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_4}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{N}\mathbf{k}_3,\mathbf{u}_k} + \frac{\Delta t}{N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{N}\mathbf{k}_3,\mathbf{u}_k}\frac{\partial \mathbf{k}_3}{\partial \mathbf{u_k}}$$

$$\mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{6N}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right)$$

return $\mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}$

The ERK4 - Explicit Runge-Kutta of 4^{th}-order

Algorithm: ERK4 with sensitivities

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad A = I$

for $i = 0 : N$ **do**

$$\mathbf{k}_1 := \mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s},\mathbf{u}_k} \qquad \frac{\partial \mathbf{k}_1}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s},\mathbf{u}_k}$$

$$\mathbf{k}_2 := \mathbf{F}\left(\mathbf{s} + \tfrac{\Delta t}{2N}\mathbf{k}_1, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_2}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k} \left(I + \frac{\Delta t}{2N}\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_2}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k} + \frac{\Delta t}{2N} \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k} \frac{\partial \mathbf{k}_1}{\partial \mathbf{u_k}}$$

$$\mathbf{k}_3 := \mathbf{F}\left(\mathbf{s} + \tfrac{\Delta t}{2N}\mathbf{k}_2, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_3}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k} \left(I + \frac{\Delta t}{2N}\frac{\partial \mathbf{k}_2}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_3}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k} + \frac{\Delta t}{2N} \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k} \frac{\partial \mathbf{k}_2}{\partial \mathbf{u_k}}$$

$$\mathbf{k}_4 := \mathbf{F}\left(\mathbf{s} + \tfrac{\Delta t}{N}\mathbf{k}_3, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_4}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{N}\mathbf{k}_3,\mathbf{u}_k} \left(I + \frac{\Delta t}{N}\frac{\partial \mathbf{k}_3}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_4}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{N}\mathbf{k}_3,\mathbf{u}_k} + \frac{\Delta t}{N} \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{N}\mathbf{k}_3,\mathbf{u}_k} \frac{\partial \mathbf{k}_3}{\partial \mathbf{u_k}}$$

$$\mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{6N}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right)$$

$$M \leftarrow I + \frac{\Delta t}{6N}\left(\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}} + 2\frac{\partial \mathbf{k}_2}{\partial \mathbf{s}} + 2\frac{\partial \mathbf{k}_3}{\partial \mathbf{s}} + \frac{\partial \mathbf{k}_4}{\partial \mathbf{s}}\right)$$

return $\quad \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}$

The ERK4 - Explicit Runge-Kutta of 4^{th}-order

Algorithm: ERK4 with sensitivities

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad A = I, \quad B = 0$

for $i = 0 : N$ **do**

$$\mathbf{k}_1 := \mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s},\mathbf{u}_k} \qquad \frac{\partial \mathbf{k}_1}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s},\mathbf{u}_k}$$

$$\mathbf{k}_2 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{2N}\mathbf{k}_1, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_2}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k}\left(I + \frac{\Delta t}{2N}\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_2}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k} + \frac{\Delta t}{2N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k}\frac{\partial \mathbf{k}_1}{\partial \mathbf{u_k}}$$

$$\mathbf{k}_3 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{2N}\mathbf{k}_2, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_3}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k}\left(I + \frac{\Delta t}{2N}\frac{\partial \mathbf{k}_2}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_3}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k} + \frac{\Delta t}{2N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k}\frac{\partial \mathbf{k}_2}{\partial \mathbf{u_k}}$$

$$\mathbf{k}_4 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{N}\mathbf{k}_3, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_4}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{N}\mathbf{k}_3,\mathbf{u}_k}\left(I + \frac{\Delta t}{N}\frac{\partial \mathbf{k}_3}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_4}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{N}\mathbf{k}_3,\mathbf{u}_k} + \frac{\Delta t}{N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{N}\mathbf{k}_3,\mathbf{u}_k}\frac{\partial \mathbf{k}_3}{\partial \mathbf{u_k}}$$

$$\mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{6N}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right)$$

$$M \leftarrow I + \frac{\Delta t}{6N}\left(\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}} + 2\frac{\partial \mathbf{k}_2}{\partial \mathbf{s}} + 2\frac{\partial \mathbf{k}_3}{\partial \mathbf{s}} + \frac{\partial \mathbf{k}_4}{\partial \mathbf{s}}\right)$$

$$A \leftarrow MA$$

return $\quad \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}, \quad \nabla_{\mathbf{x}_k}\mathbf{f}_k\left(\mathbf{x}_k, \mathbf{u}_k\right) = A^\top,$

The ERK4 - Explicit Runge-Kutta of 4^{th}-order

Algorithm: ERK4 with sensitivities

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad A = I, \quad B = 0$

for $i = 0 : N$ **do**

$$\mathbf{k}_1 := \mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s},\mathbf{u}_k} \qquad \frac{\partial \mathbf{k}_1}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s},\mathbf{u}_k}$$

$$\mathbf{k}_2 := \mathbf{F}\left(\mathbf{s} + \tfrac{\Delta t}{2N}\mathbf{k}_1, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_2}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k}\left(I + \frac{\Delta t}{2N}\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_2}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k} + \frac{\Delta t}{2N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k}\frac{\partial \mathbf{k}_1}{\partial \mathbf{u_k}}$$

$$\mathbf{k}_3 := \mathbf{F}\left(\mathbf{s} + \tfrac{\Delta t}{2N}\mathbf{k}_2, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_3}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k}\left(I + \frac{\Delta t}{2N}\frac{\partial \mathbf{k}_2}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_3}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k} + \frac{\Delta t}{2N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k}\frac{\partial \mathbf{k}_2}{\partial \mathbf{u_k}}$$

$$\mathbf{k}_4 := \mathbf{F}\left(\mathbf{s} + \tfrac{\Delta t}{N}\mathbf{k}_3, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_4}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{N}\mathbf{k}_3,\mathbf{u}_k}\left(I + \frac{\Delta t}{N}\frac{\partial \mathbf{k}_3}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_4}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{N}\mathbf{k}_3,\mathbf{u}_k} + \frac{\Delta t}{N}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{N}\mathbf{k}_3,\mathbf{u}_k}\frac{\partial \mathbf{k}_3}{\partial \mathbf{u_k}}$$

$$\mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{6N}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right)$$

$$M \leftarrow I + \frac{\Delta t}{6N}\left(\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}} + 2\frac{\partial \mathbf{k}_2}{\partial \mathbf{s}} + 2\frac{\partial \mathbf{k}_3}{\partial \mathbf{s}} + \frac{\partial \mathbf{k}_4}{\partial \mathbf{s}}\right)$$

$$A \leftarrow MA$$

$$B \leftarrow MB + \frac{\Delta t}{6N}\left(\frac{\partial \mathbf{k}_1}{\partial \mathbf{u}} + 2\frac{\partial \mathbf{k}_2}{\partial \mathbf{u}} + 2\frac{\partial \mathbf{k}_3}{\partial \mathbf{u}} + \frac{\partial \mathbf{k}_4}{\partial \mathbf{u}}\right)$$

return $\quad \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}, \quad \nabla_{\mathbf{x}_k}\mathbf{f}_k\left(\mathbf{x}_k, \mathbf{u}_k\right) = A^\top,$

The ERK4 - Explicit Runge-Kutta of 4^{th}-order

Algorithm: ERK4 with sensitivities

Input: $\mathbf{x}_k$, $\mathbf{u}_k$, $[t_k,\ t_{k+1}]$, N

$\mathbf{s} = \mathbf{x}_k, \quad A = I, \quad B = 0$

for $i = 0 : N$ **do**

$$\mathbf{k}_1 := \mathbf{F}\left(\mathbf{s}, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s},\mathbf{u}_k} \qquad \frac{\partial \mathbf{k}_1}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s},\mathbf{u}_k}$$

$$\mathbf{k}_2 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{2N}\mathbf{k}_1, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_2}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k} \left(I + \frac{\Delta t}{2N}\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_2}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k} + \frac{\Delta t}{2N} \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_1,\mathbf{u}_k} \frac{\partial \mathbf{k}_1}{\partial \mathbf{u_k}}$$

$$\mathbf{k}_3 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{2N}\mathbf{k}_2, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_3}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k} \left(I + \frac{\Delta t}{2N}\frac{\partial \mathbf{k}_2}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_3}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k} + \frac{\Delta t}{2N} \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{2N}\mathbf{k}_2,\mathbf{u}_k} \frac{\partial \mathbf{k}_2}{\partial \mathbf{u_k}}$$

$$\mathbf{k}_4 := \mathbf{F}\left(\mathbf{s} + \frac{\Delta t}{N}\mathbf{k}_3, \mathbf{u}_k\right)$$

$$\frac{\partial \mathbf{k}_4}{\partial \mathbf{s}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{N}\mathbf{k}_3,\mathbf{u}_k} \left(I + \frac{\Delta t}{N}\frac{\partial \mathbf{k}_3}{\partial \mathbf{s}}\right) \qquad \frac{\partial \mathbf{k}_4}{\partial \mathbf{u_k}} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\mathbf{s}+\frac{\Delta t}{N}\mathbf{k}_3,\mathbf{u}_k} + \frac{\Delta t}{N} \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\mathbf{s}+\frac{\Delta t}{N}\mathbf{k}_3,\mathbf{u}_k} \frac{\partial \mathbf{k}_3}{\partial \mathbf{u_k}}$$

$$\mathbf{s} \leftarrow \mathbf{s} + \frac{\Delta t}{6N}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right)$$

$$M \leftarrow I + \frac{\Delta t}{6N}\left(\frac{\partial \mathbf{k}_1}{\partial \mathbf{s}} + 2\frac{\partial \mathbf{k}_2}{\partial \mathbf{s}} + 2\frac{\partial \mathbf{k}_3}{\partial \mathbf{s}} + \frac{\partial \mathbf{k}_4}{\partial \mathbf{s}}\right)$$

$$A \leftarrow MA$$

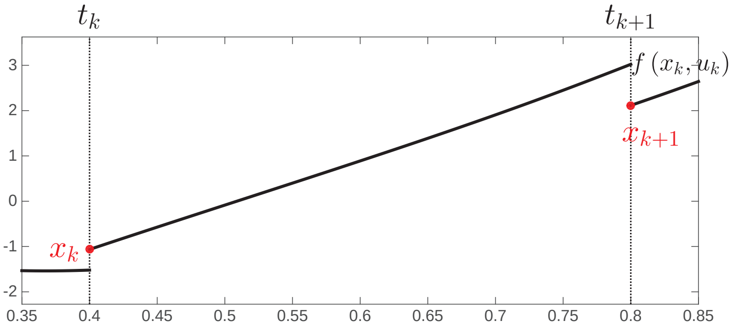
$$B \leftarrow MB + \frac{\Delta t}{6N}\left(\frac{\partial \mathbf{k}_1}{\partial \mathbf{u}} + 2\frac{\partial \mathbf{k}_2}{\partial \mathbf{u}} + 2\frac{\partial \mathbf{k}_3}{\partial \mathbf{u}} + \frac{\partial \mathbf{k}_4}{\partial \mathbf{u}}\right)$$

return $\quad \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}, \quad \nabla_{\mathbf{x}_k}\mathbf{f}_k\left(\mathbf{x}_k, \mathbf{u}_k\right) = A^\top, \quad \nabla_{\mathbf{u}_k}\mathbf{f}_k\left(\mathbf{x}_k, \mathbf{u}_k\right) = B^\top$

Outline

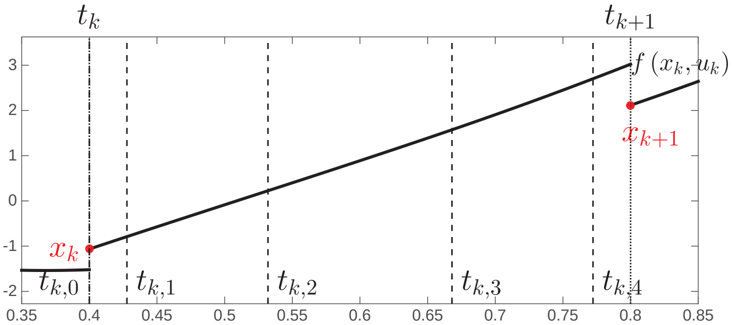
Collocation methods - key idea

Approximate state trajectory $\mathbf{s}(t)$ via polynomials (order K)



Approximate state trajectory $\mathbf{s}(t)$ via polynomials (order K)

- Pick $K+1$ time nodes $t_{k,i} \in [t_k, t_{k+1}]$

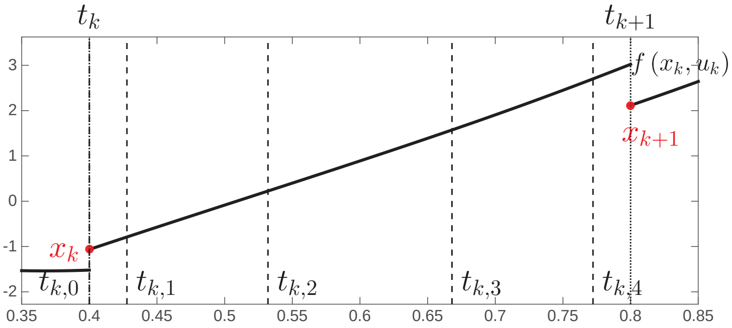


Collocation methods - key idea

Approximate state trajectory $\mathbf{s}(t)$ via polynomials (order K)

- Pick $K+1$ time nodes $t_{k,i} \in [t_k, t_{k+1}]$
- Approximate on each interval $[t_k, t_{k+1}]$:

$$\mathbf{s}(\boldsymbol{\theta}_k, t) = \sum_{i=0}^{K} \underbrace{\boldsymbol{\theta}_{k,i}}_{\text{parameters}} \cdot \underbrace{P_{k,i}(t)}_{\text{polynomials}}$$



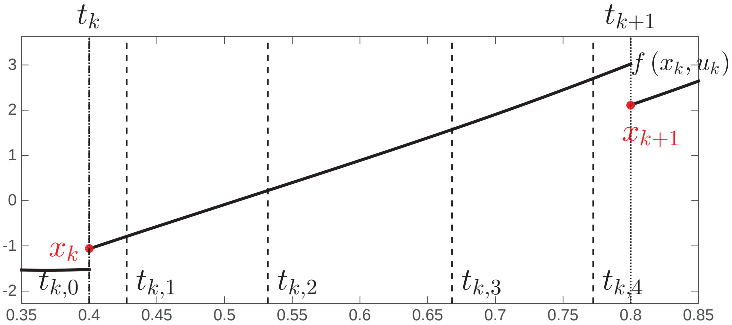
Collocation methods - key idea

Approximate state trajectory $\mathrm{s}(t)$ via polynomials (order K)

- Pick $K+1$ time nodes $t_{k,i} \in [t_k, t_{k+1}]$
- Approximate on each interval $[t_k, t_{k+1}]$:

$$\mathrm{s}(\boldsymbol{\theta}_k, t) = \sum_{i=0}^{K} \underbrace{\boldsymbol{\theta}_{k,i}}_{\text{parameters}} \cdot \underbrace{P_{k,i}(t)}_{\text{polynomials}}$$

- Pick the $P_{k,i}(t)$ so that $\mathrm{s}(\boldsymbol{\theta}_k, t_{k,i}) = \boldsymbol{\theta}_{k,i}$,



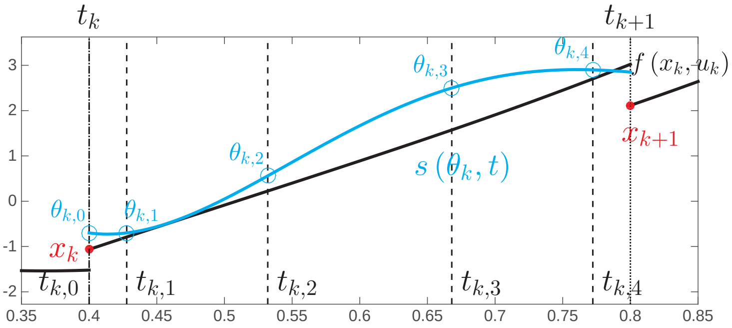
Collocation methods - key idea

Approximate state trajectory $\mathrm{s}(t)$ via polynomials (order K)

- Pick $K+1$ time nodes $t_{k,i} \in [t_k, t_{k+1}]$
- Approximate on each interval $[t_k, t_{k+1}]$:

$$\mathrm{s}(\boldsymbol{\theta}_k, t) = \sum_{i=0}^{K} \underbrace{\boldsymbol{\theta}_{k,i}}_{\text{parameters}} \cdot \underbrace{P_{k,i}(t)}_{\text{polynomials}}$$

- Pick the $P_{k,i}(t)$ so that $\mathrm{s}(\boldsymbol{\theta}_k, t_{k,i}) = \boldsymbol{\theta}_{k,i}$,



Collocation methods - key idea

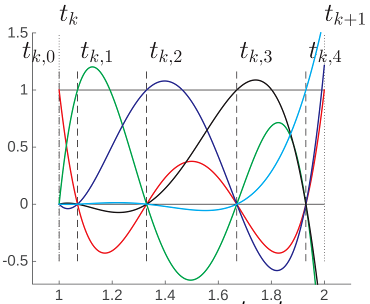
Approximate state trajectory $\mathrm{s}(t)$ via polynomials (order K)

- Pick $K+1$ time nodes $t_{k,i} \in [t_k, t_{k+1}]$
- Approximate on each interval $[t_k, t_{k+1}]$:

$$\mathrm{s}(\boldsymbol{\theta}_k, t) = \sum_{i=0}^{K} \underbrace{\boldsymbol{\theta}_{k,i}}_{\text{parameters}} \cdot \underbrace{P_{k,i}(t)}_{\text{polynomials}}$$

- Pick the $P_{k,i}(t)$ so that $\mathrm{s}(\boldsymbol{\theta}_k, t_{k,i}) = \boldsymbol{\theta}_{k,i}$,

e.g. Lagrange Polynomials ($K = 4$)



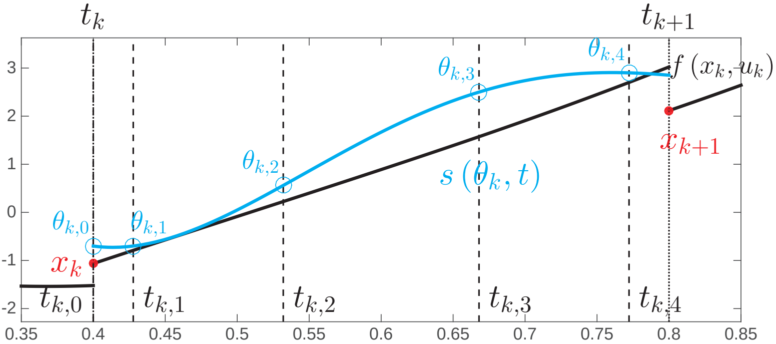
$$P_{k,i}(t) = \Pi_{j=0, j\neq i}^{K} \frac{t - t_{k,j}}{t_{k,i} - t_{k,j}}$$

Collocation methods - how to interpolate ?

On each interval $[t_k, t_{k+1}]$, approximate $\dot{\mathbf{s}} = \mathbf{F}(\mathbf{s}, \mathbf{u}_k)$ using

$$\mathbf{s}(\boldsymbol{\theta}_k, t) = \sum_{i=0}^{K} \underbrace{\boldsymbol{\theta}_{k,i}}_{\text{parameters}} \cdot \underbrace{P_{k,i}(t)}_{\text{polynomials}} \qquad \text{with} \qquad \mathbf{s}(\boldsymbol{\theta}_k, t_{k,i}) = \boldsymbol{\theta}_{k,i}$$

Note: we have $K+1$ degrees of freedom per state.



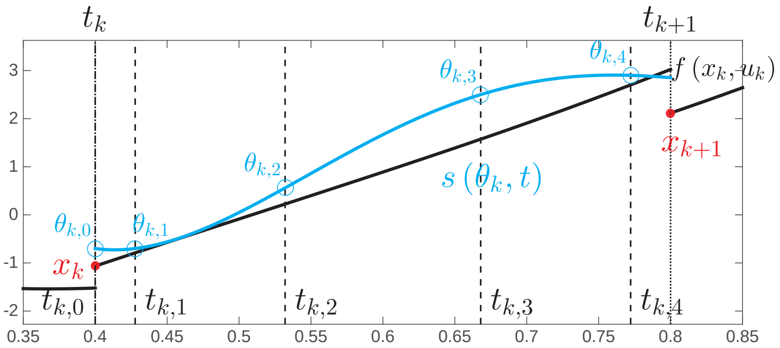
Collocation methods - how to interpolate ?

On each interval $[t_k, t_{k+1}]$, approximate $\dot{\mathbf{s}} = \mathbf{F}(\mathbf{s}, \mathbf{u}_k)$ using

$$\mathbf{s}(\boldsymbol{\theta}_k, t) = \sum_{i=0}^{K} \underbrace{\boldsymbol{\theta}_{k,i}}_{\text{parameters}} \cdot \underbrace{P_{k,i}(t)}_{\text{polynomials}} \qquad \text{with} \qquad \mathbf{s}(\boldsymbol{\theta}_k, t_{k,i}) = \boldsymbol{\theta}_{k,i}$$

Note: we have $K+1$ degrees of freedom per state. Collocation uses the constraints:

$$\mathbf{s}\left(\boldsymbol{\theta}_k, t_k\right) = \mathbf{x}_k, \qquad \text{(note that } \mathbf{x}_k \text{ is coming from the NLP !!)}$$



Collocation methods - how to interpolate ?

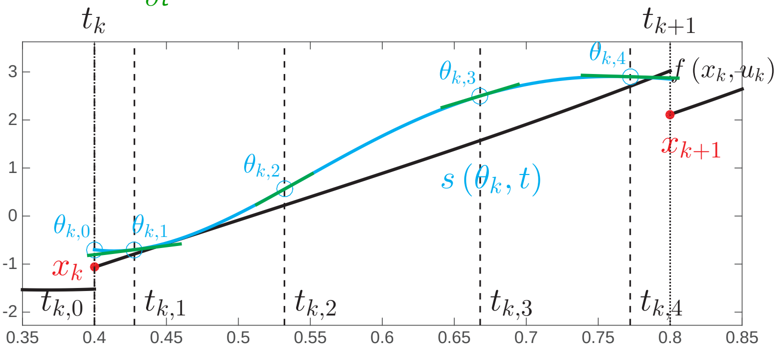
On each interval $[t_k, t_{k+1}]$, approximate $\dot{\mathbf{s}} = \mathbf{F}(\mathbf{s}, \mathbf{u}_k)$ using

$$\mathbf{s}(\boldsymbol{\theta}_k, t) = \sum_{i=0}^{K} \underbrace{\boldsymbol{\theta}_{k,i}}_{\text{parameters}} \cdot \underbrace{P_{k,i}(t)}_{\text{polynomials}} \qquad \text{with} \qquad \mathbf{s}(\boldsymbol{\theta}_k, t_{k,i}) = \boldsymbol{\theta}_{k,i}$$

Note: we have $K+1$ degrees of freedom per state. Collocation uses the constraints:

$$\mathbf{s}(\boldsymbol{\theta}_k, t_k) = \mathbf{x}_k, \qquad \text{(note that } \mathbf{x}_k \text{ is coming from the NLP !!)}$$

$$\frac{\partial}{\partial t}\mathbf{s}(\boldsymbol{\theta}_k, t_{k,i}) = \mathbf{F}(\mathbf{s}(\boldsymbol{\theta}_k, t_{k,i}), \mathbf{u}_k), \qquad i = 1, ..., K$$



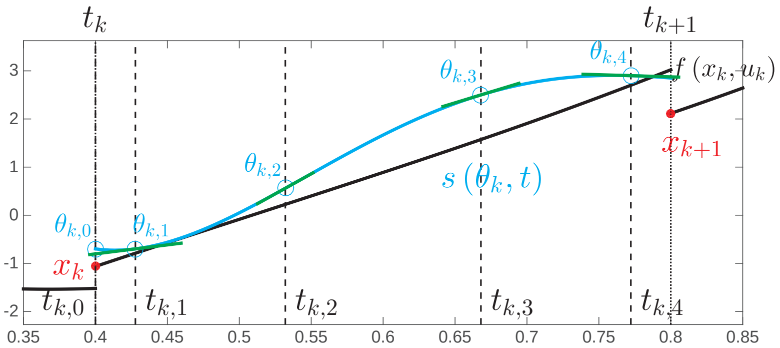
Collocation methods - Implementation

Collocation uses the constraints:

$$\mathbf{s}\left(\boldsymbol{\theta}_k, t_k\right) = \mathbf{x}_k$$

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t_{k,i}\right) = \mathbf{F}\left(\mathbf{s}\left(\boldsymbol{\theta}_k, t_{k,i}\right), \mathbf{u}_k\right),$$

with $i = 1, ..., K$.



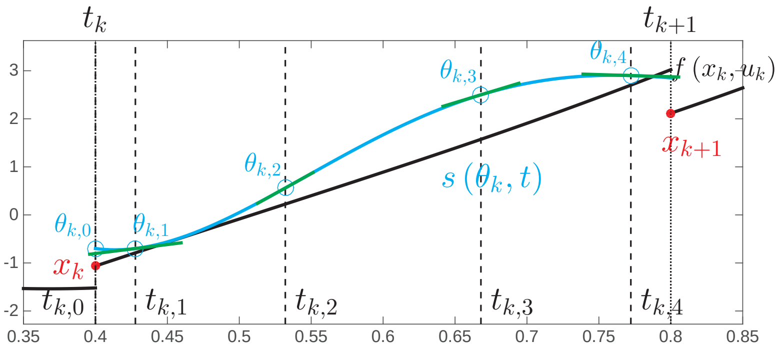
Collocation methods - Implementation

Collocation uses the constraints:

$$\mathbf{s}\left(\boldsymbol{\theta}_k, t_k\right) = \mathbf{x}_k$$

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t_{k,i}\right) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right),$$

with $i = 1, ..., K$.



Collocation methods - Implementation

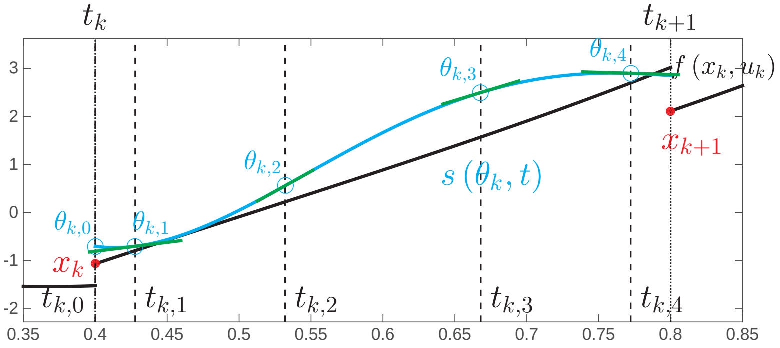
Collocation uses the constraints:

$$\mathbf{s}\left(\boldsymbol{\theta}_k, t_k\right) = \mathbf{x}_k$$

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t_{k,i}\right) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right),$$

with $i = 1, ..., K$. Note:

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t\right) = \sum_{j=0}^{K} \boldsymbol{\theta}_{k,j}\dot{P}_{k,j}(t)$$



Collocation methods - Implementation

Collocation uses the constraints:

$$\mathbf{s}\left(\boldsymbol{\theta}_k, t_k\right) = \mathbf{x}_k$$

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t_{k,i}\right) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right),$$

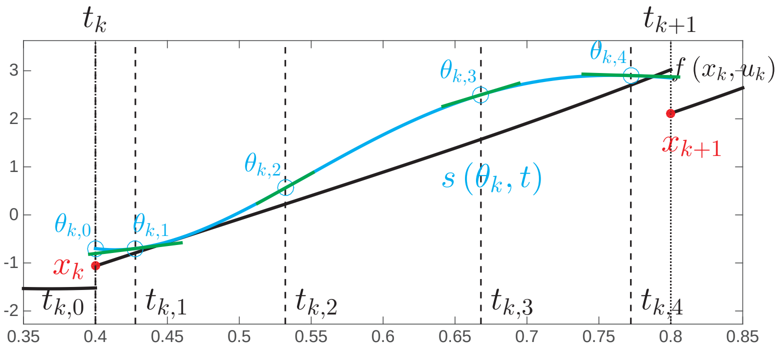
with $i = 1, ..., K$. Note:

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t\right) = \sum_{j=0}^{K} \boldsymbol{\theta}_{k,j}\dot{P}_{k,j}(t)$$

Solve for $\boldsymbol{\theta}_{k,i}$ using Newton

$$\sum_{i=0}^{K} \boldsymbol{\theta}_{k,i} P_{k,i}(t_k) = \mathbf{x}_k$$

$$\sum_{j=0}^{K} \boldsymbol{\theta}_{k,j}\dot{P}_{k,j}(t_{k,i}) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right), \quad i = 1, ..., K$$



Collocation methods - Implementation

Collocation uses the constraints:

$$\mathbf{s}\left(\boldsymbol{\theta}_k, t_k\right) = \mathbf{x}_k$$

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t_{k,i}\right) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right),$$

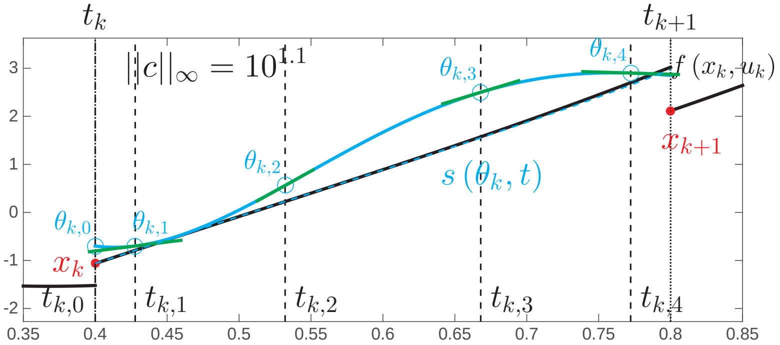
with $i = 1, ..., K$. Note:

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t\right) = \sum_{j=0}^{K} \boldsymbol{\theta}_{k,j}\dot{P}_{k,j}(t)$$

Solve for $\boldsymbol{\theta}_{k,i}$ using Newton

$$\sum_{i=0}^{K} \boldsymbol{\theta}_{k,i} P_{k,i}(t_k) = \mathbf{x}_k$$

$$\sum_{j=0}^{K} \boldsymbol{\theta}_{k,j}\dot{P}_{k,j}(t_{k,i}) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right), \quad i = 1, ..., K$$



Collocation methods - Implementation

Collocation uses the constraints:

$$\mathbf{s}\left(\boldsymbol{\theta}_k, t_k\right) = \mathbf{x}_k$$

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t_{k,i}\right) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right),$$

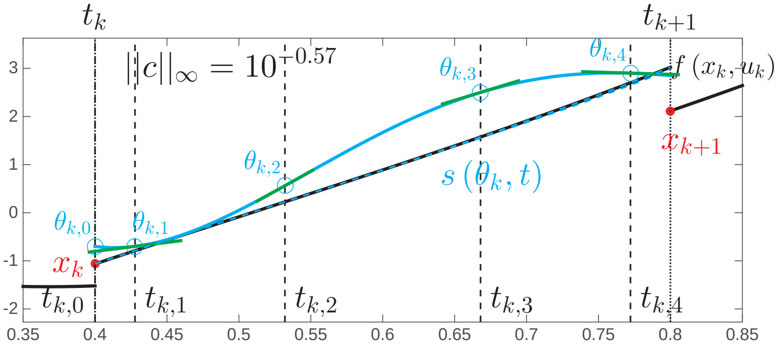
with $i = 1, ..., K$. Note:

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t\right) = \sum_{j=0}^{K} \boldsymbol{\theta}_{k,j}\dot{P}_{k,j}(t)$$

Solve for $\boldsymbol{\theta}_{k,i}$ using Newton

$$\sum_{i=0}^{K} \boldsymbol{\theta}_{k,i} P_{k,i}(t_k) = \mathbf{x}_k$$

$$\sum_{j=0}^{K} \boldsymbol{\theta}_{k,j}\dot{P}_{k,j}(t_{k,i}) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right), \quad i = 1, ..., K$$



Collocation methods - Implementation

Collocation uses the constraints:

$$\mathbf{s}\left(\boldsymbol{\theta}_k, t_k\right) = \mathbf{x}_k$$

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t_{k,i}\right) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right),$$

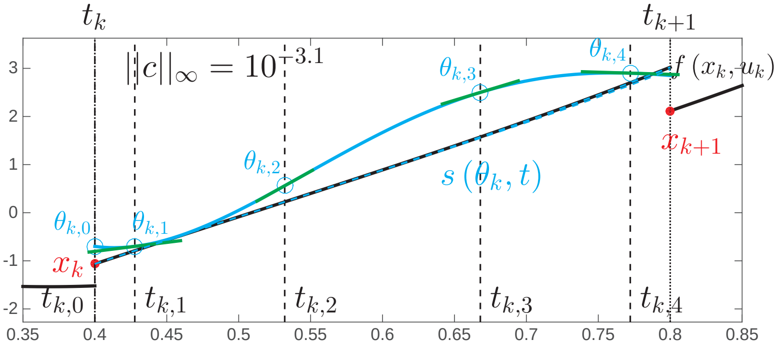
with $i = 1, ..., K$. Note:

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t\right) = \sum_{j=0}^{K} \boldsymbol{\theta}_{k,j}\dot{P}_{k,j}(t)$$

Solve for $\boldsymbol{\theta}_{k,i}$ using Newton

$$\sum_{i=0}^{K} \boldsymbol{\theta}_{k,i}P_{k,i}(t_k) = \mathbf{x}_k$$

$$\sum_{j=0}^{K} \boldsymbol{\theta}_{k,j}\dot{P}_{k,j}(t_{k,i}) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right), \quad i = 1, ..., K$$



Collocation methods - Implementation

Collocation uses the constraints:

$$\mathbf{s}\left(\boldsymbol{\theta}_k, t_k\right) = \mathbf{x}_k$$

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t_{k,i}\right) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right),$$

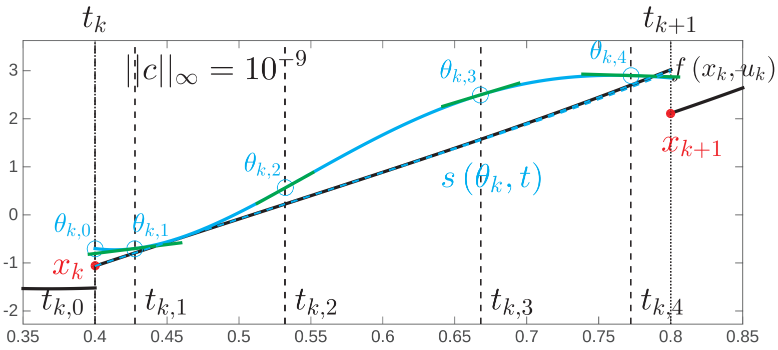
with $i = 1, ..., K$. Note:

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t\right) = \sum_{j=0}^{K} \boldsymbol{\theta}_{k,j}\dot{P}_{k,j}(t)$$

Solve for $\boldsymbol{\theta}_{k,i}$ using Newton

$$\sum_{i=0}^{K} \boldsymbol{\theta}_{k,i} P_{k,i}(t_k) = \mathbf{x}_k$$

$$\sum_{j=0}^{K} \boldsymbol{\theta}_{k,j}\dot{P}_{k,j}(t_{k,i}) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right), \quad i = 1, ..., K$$



Collocation methods - Implementation

Collocation uses the constraints:

$$\mathbf{s}\left(\boldsymbol{\theta}_k, t_k\right) = \mathbf{x}_k$$

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t_{k,i}\right) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right),$$

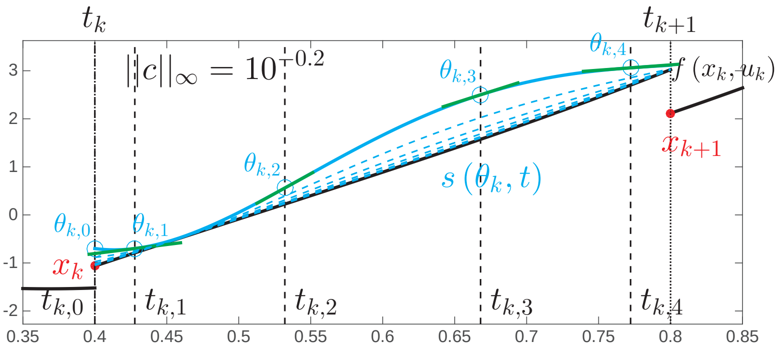
with $i = 1, ..., K$. Note:

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t\right) = \sum_{j=0}^{K} \boldsymbol{\theta}_{k,j} \dot{P}_{k,j}(t)$$

Solve for $\boldsymbol{\theta}_{k,i}$ using Newton

$$\sum_{i=0}^{K} \boldsymbol{\theta}_{k,i} P_{k,i}(t_k) = \mathbf{x}_k$$

$$\sum_{j=0}^{K} \boldsymbol{\theta}_{k,j} \dot{P}_{k,j}(t_{k,i}) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right), \quad i = 1, ..., K$$



Collocation methods - Implementation

Collocation uses the constraints:

$$\mathbf{s}\left(\boldsymbol{\theta}_k, t_k\right) = \mathbf{x}_k$$

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t_{k,i}\right) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right),$$

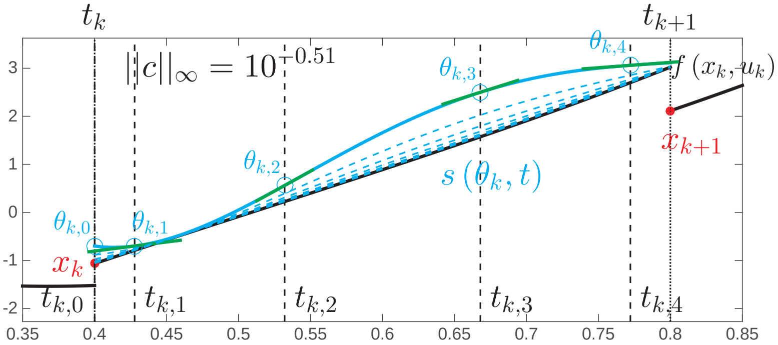
with $i = 1, ..., K$. Note:

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t\right) = \sum_{j=0}^{K} \boldsymbol{\theta}_{k,j} \dot{P}_{k,j}(t)$$

Solve for $\boldsymbol{\theta}_{k,i}$ using Newton

$$\sum_{i=0}^{K} \boldsymbol{\theta}_{k,i} P_{k,i}(t_k) = \mathbf{x}_k$$

$$\sum_{j=0}^{K} \boldsymbol{\theta}_{k,j} \dot{P}_{k,j}(t_{k,i}) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right), \quad i = 1, ..., K$$



Collocation methods - Implementation

Collocation uses the constraints:

$$\mathbf{s}\left(\boldsymbol{\theta}_k, t_k\right) = \mathbf{x}_k$$

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t_{k,i}\right) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right),$$

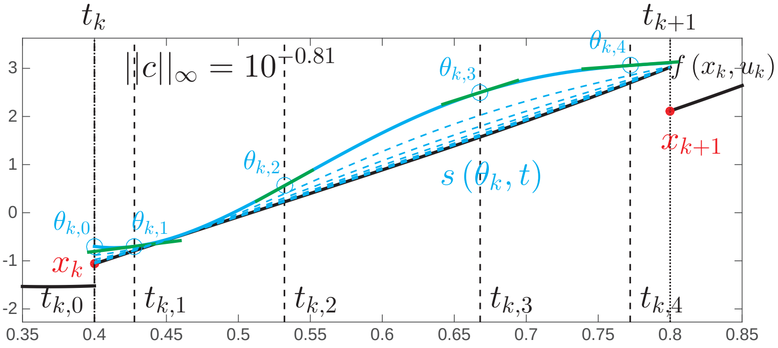
with $i = 1, ..., K$. Note:

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t\right) = \sum_{j=0}^{K} \boldsymbol{\theta}_{k,j}\dot{P}_{k,j}(t)$$

Solve for $\boldsymbol{\theta}_{k,i}$ using Newton

$$\sum_{i=0}^{K} \boldsymbol{\theta}_{k,i} P_{k,i}(t_k) = \mathbf{x}_k$$

$$\sum_{j=0}^{K} \boldsymbol{\theta}_{k,j}\dot{P}_{k,j}(t_{k,i}) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right), \quad i = 1, ..., K$$



Collocation methods - Implementation

Collocation uses the constraints:

$$\mathbf{s}\left(\boldsymbol{\theta}_k, t_k\right) = \mathbf{x}_k$$

$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t_{k,i}\right) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right),$$

with $i = 1, ..., K$. Note:

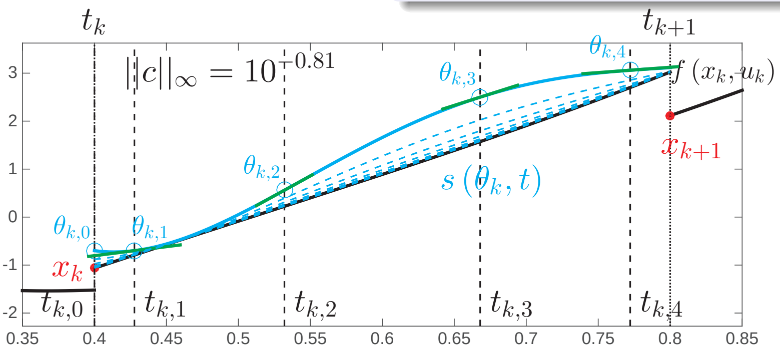
$$\frac{\partial}{\partial t}\mathbf{s}\left(\boldsymbol{\theta}_k, t\right) = \sum_{j=0}^{K} \boldsymbol{\theta}_{k,j}\dot{P}_{k,j}(t)$$

Shooting constraints

$$\underbrace{\mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right)}_{=\mathbf{s}(\boldsymbol{\theta}_k, t_k)} - \mathbf{x}_{k+1} = 0$$

becomes:

$$\sum_{j=0}^{K} \boldsymbol{\theta}_{k,j} P_{k,j}(t_{k+1}) - \mathbf{x}_{k+1} = 0$$



Collocation - Sensitivity

Collocation constraints...

$$\sum_{i=0}^{K} \boldsymbol{\theta}_{k,i} P_{k,i}(t_k) = \mathbf{x}_k$$

$$\sum_{j=0}^{K} \boldsymbol{\theta}_{k,j} \dot{P}_{k,j}(t_{k,i}) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right), \quad i = 1, ..., K$$

Collocation - Sensitivity

Collocation constraints...

$$\sum_{i=0}^{K} \boldsymbol{\theta}_{k,i} P_{k,i}(t_k) = \mathbf{x}_k$$

$$\sum_{j=0}^{K} \boldsymbol{\theta}_{k,j} \dot{P}_{k,j}(t_{k,i}) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right), \quad i = 1, ..., K$$

... can be seen as

$$\mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right) = 0$$

Collocation - Sensitivity

Collocation constraints...

$$\sum_{i=0}^{K} \boldsymbol{\theta}_{k,i} P_{k,i}(t_k) = \mathbf{x}_k$$

$$\sum_{j=0}^{K} \boldsymbol{\theta}_{k,j} \dot{P}_{k,j}(t_{k,i}) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right), \quad i = 1, ..., K$$

... can be seen as

$$\mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right) = 0$$

Solved by iterating:

$$\Delta\boldsymbol{\theta}_k = -\frac{\partial \mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right)}{\partial \boldsymbol{\theta}_k}^{-1} \mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right)$$

Collocation - Sensitivity

Collocation constraints...

$$\sum_{i=0}^{K} \boldsymbol{\theta}_{k,i} P_{k,i}(t_k) = \mathbf{x}_k$$

$$\sum_{j=0}^{K} \boldsymbol{\theta}_{k,j} \dot{P}_{k,j}(t_{k,i}) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right), \quad i = 1, ..., K$$

Integrator function given by:

$$\mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}\left(\boldsymbol{\theta}_k, t_{k+1}\right)$$

... can be seen as

$$\mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right) = 0$$

Solved by iterating:

$$\Delta\boldsymbol{\theta}_k = -\frac{\partial \mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right)}{\partial \boldsymbol{\theta}_k}^{-1} \mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right)$$

Collocation - Sensitivity

Collocation constraints...

$$\sum_{i=0}^{K} \boldsymbol{\theta}_{k,i} P_{k,i}(t_k) = \mathbf{x}_k$$

$$\sum_{j=0}^{K} \boldsymbol{\theta}_{k,j} \dot{P}_{k,j}(t_{k,i}) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right), \quad i = 1, ..., K$$

... can be seen as

$$\mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right) = 0$$

Solved by iterating:

$$\Delta\boldsymbol{\theta}_k = -\frac{\partial \mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right)}{\partial \boldsymbol{\theta}_k}^{-1} \mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right)$$

Integrator function given by:

$$\mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}\left(\boldsymbol{\theta}_k, t_{k+1}\right)$$

Get sensitivities using:

$$\frac{\partial \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right)}{\partial \mathbf{x}_k} = \frac{\partial \mathbf{s}\left(\boldsymbol{\theta}_k, t_{k+1}\right)}{\partial \boldsymbol{\theta}_k} \frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{x}_k}, \qquad \frac{\partial \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right)}{\partial \mathbf{u}_k} = \frac{\partial \mathbf{s}\left(\boldsymbol{\theta}_k, t_{k+1}\right)}{\partial \boldsymbol{\theta}_k} \frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{u}_k}$$

Collocation - Sensitivity

Collocation constraints...

$$\sum_{i=0}^{K} \boldsymbol{\theta}_{k,i} P_{k,i}(t_k) = \mathbf{x}_k$$

$$\sum_{j=0}^{K} \boldsymbol{\theta}_{k,j} \dot{P}_{k,j}(t_{k,i}) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right), \quad i = 1, ..., K$$

... can be seen as

$$\mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right) = 0$$

Solved by iterating:

$$\Delta\boldsymbol{\theta}_k = -\frac{\partial \mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right)}{\partial \boldsymbol{\theta}_k}^{-1} \mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right)$$

Integrator function given by:

$$\mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}\left(\boldsymbol{\theta}_k, t_{k+1}\right)$$

Get sensitivities using:

$$\frac{\partial \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right)}{\partial \mathbf{x}_k} = \frac{\partial \mathbf{s}\left(\boldsymbol{\theta}_k, t_{k+1}\right)}{\partial \boldsymbol{\theta}_k}\frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{x}_k}, \qquad \frac{\partial \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right)}{\partial \mathbf{u}_k} = \frac{\partial \mathbf{s}\left(\boldsymbol{\theta}_k, t_{k+1}\right)}{\partial \boldsymbol{\theta}_k}\frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{u}_k}$$

Implicit function theorem states that

$$\frac{\partial \mathbf{c}}{\partial \boldsymbol{\theta}_k}\frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{x}_k} + \frac{\partial \mathbf{c}}{\partial \mathbf{x}_k} = 0, \qquad \frac{\partial \mathbf{c}}{\partial \boldsymbol{\theta}_k}\frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{u}_k} + \frac{\partial \mathbf{c}}{\partial \mathbf{u}_k} = 0$$

Collocation - Sensitivity

Collocation constraints...

$$\sum_{i=0}^{K} \boldsymbol{\theta}_{k,i} P_{k,i}(t_k) = \mathbf{x}_k$$

$$\sum_{j=0}^{K} \boldsymbol{\theta}_{k,j} \dot{P}_{k,j}(t_{k,i}) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right), \quad i = 1, ..., K$$

... can be seen as

$$\mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right) = 0$$

Solved by iterating:

$$\Delta\boldsymbol{\theta}_k = -\frac{\partial \mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right)}{\partial \boldsymbol{\theta}_k}^{-1} \mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right)$$

Integrator function given by:

$$\mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}\left(\boldsymbol{\theta}_k, t_{k+1}\right)$$

Get sensitivities using:

$$\frac{\partial \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right)}{\partial \mathbf{x}_k} = \frac{\partial \mathbf{s}\left(\boldsymbol{\theta}_k, t_{k+1}\right)}{\partial \boldsymbol{\theta}_k} \frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{x}_k}, \qquad \frac{\partial \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right)}{\partial \mathbf{u}_k} = \frac{\partial \mathbf{s}\left(\boldsymbol{\theta}_k, t_{k+1}\right)}{\partial \boldsymbol{\theta}_k} \frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{u}_k}$$

Implicit function theorem states that

$$\frac{\partial \mathbf{c}}{\partial \boldsymbol{\theta}_k} \frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{x}_k} + \frac{\partial \mathbf{c}}{\partial \mathbf{x}_k} = 0, \qquad \frac{\partial \mathbf{c}}{\partial \boldsymbol{\theta}_k} \frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{u}_k} + \frac{\partial \mathbf{c}}{\partial \mathbf{u}_k} = 0$$

Hence:

$$\frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{x}_k} = -\frac{\partial \mathbf{c}}{\partial \boldsymbol{\theta}_k}^{-1} \frac{\partial \mathbf{c}}{\partial \mathbf{x}_k}, \qquad \frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{u}_k} = -\frac{\partial \mathbf{c}}{\partial \boldsymbol{\theta}_k}^{-1} \frac{\partial \mathbf{c}}{\partial \mathbf{u}_k}$$

Collocation - Sensitivity

Collocation constraints...

$$\sum_{i=0}^{K} \boldsymbol{\theta}_{k,i} P_{k,i}(t_k) = \mathbf{x}_k$$

$$\sum_{j=0}^{K} \boldsymbol{\theta}_{k,j} \dot{P}_{k,j}(t_{k,i}) = \mathbf{F}\left(\boldsymbol{\theta}_{k,i}, \mathbf{u}_k\right), \quad i = 1, ..., K$$

... can be seen as

$$\mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right) = 0$$

Solved by iterating:

$$\Delta\boldsymbol{\theta}_k = -\frac{\partial \mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right)}{\partial \boldsymbol{\theta}_k}^{-1} \mathbf{c}\left(\mathbf{x}_k, \boldsymbol{\theta}_k, \mathbf{u}_k\right)$$

Integrator function given by:

$$\mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) = \mathbf{s}\left(\boldsymbol{\theta}_k, t_{k+1}\right)$$

Get sensitivities using:

$$\frac{\partial \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right)}{\partial \mathbf{x}_k} = \frac{\partial \mathbf{s}\left(\boldsymbol{\theta}_k, t_{k+1}\right)}{\partial \boldsymbol{\theta}_k} \frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{x}_k}, \qquad \frac{\partial \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right)}{\partial \mathbf{u}_k} = \frac{\partial \mathbf{s}\left(\boldsymbol{\theta}_k, t_{k+1}\right)}{\partial \boldsymbol{\theta}_k} \frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{u}_k}$$

Implicit function theorem states that

$$\frac{\partial \mathbf{c}}{\partial \boldsymbol{\theta}_k} \frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{x}_k} + \frac{\partial \mathbf{c}}{\partial \mathbf{x}_k} = 0, \qquad \frac{\partial \mathbf{c}}{\partial \boldsymbol{\theta}_k} \frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{u}_k} + \frac{\partial \mathbf{c}}{\partial \mathbf{u}_k} = 0$$

Hence:

$$\frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{x}_k} = -\frac{\partial \mathbf{c}}{\partial \boldsymbol{\theta}_k}^{-1} \frac{\partial \mathbf{c}}{\partial \mathbf{x}_k}, \qquad \frac{\partial \boldsymbol{\theta}_k}{\partial \mathbf{u}_k} = -\frac{\partial \mathbf{c}}{\partial \boldsymbol{\theta}_k}^{-1} \frac{\partial \mathbf{c}}{\partial \mathbf{u}_k}$$

Note that $\frac{\partial \mathbf{c}}{\partial \boldsymbol{\theta}_k}^{-1}$ is computed in the Newton iteration, i.e. it comes for free !!

Outline

Forward sensitivity (forward AD)

Consider $\mathbf{f}(\mathbf{x}, \mathbf{u})$ given by

Algorithm: function $\mathbf{f}(\mathbf{x}, \mathbf{u})$

Input: $\mathbf{x} \in \mathbb{R}^n, \ \mathbf{u} \in \mathbb{R}^m$
$\mathbf{s} = \mathbf{x}$
for $i = 1{:}N$ **do**
    $\mathbf{s} \leftarrow \boldsymbol{\xi}(\mathbf{s}, \mathbf{u})$
return $\mathbf{f}(\mathbf{x}, \mathbf{u}) = \mathbf{s}$

Note: e.g. $\boldsymbol{\xi}$ *reads as* $\mathbf{s} + \Delta t\mathbf{F}$ *in Euler*

Forward sensitivity (forward AD)

Consider $\mathbf{f}(\mathbf{x}, \mathbf{u})$ given by

Algorithm: function $\mathbf{f}(\mathbf{x}, \mathbf{u})$

Input: $\mathbf{x} \in \mathbb{R}^n,\ \mathbf{u} \in \mathbb{R}^m$
$\mathbf{s} = \mathbf{x}$
for $i = 1{:}N$ **do**
  $\mathbf{s} \leftarrow \boldsymbol{\xi}(\mathbf{s}, \mathbf{u})$
return $\mathbf{f}(\mathbf{x}, \mathbf{u}) = \mathbf{s}$

Note: e.g. $\boldsymbol{\xi}$ *reads as* $\mathbf{s} + \Delta t \mathbf{F}$ *in Euler*

Forward sensitivity computation:

Algorithm: function $\mathbf{f}(\mathbf{x}, \mathbf{u})$ with forward sensitivity

Input: $\mathbf{x} \in \mathbb{R}^n,\ \mathbf{u} \in \mathbb{R}^m$
$\mathbf{s} = \mathbf{x},\ A = I,\ B = 0$
for $i = 1 : N$ **do**
  $A \leftarrow \frac{\partial \boldsymbol{\xi}(\mathbf{s},\mathbf{u})}{\partial \mathbf{s}} A$
  $B \leftarrow \frac{\partial \boldsymbol{\xi}(\mathbf{s},\mathbf{u})}{\partial \mathbf{s}} B + \frac{\partial \boldsymbol{\xi}(\mathbf{s},\mathbf{u})}{\partial \mathbf{u}}$
  $\mathbf{s} \leftarrow \boldsymbol{\zeta}(\mathbf{s}, \mathbf{u})$
return $\mathbf{f}(\mathbf{x}, \mathbf{u}) = \mathbf{s},\ \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}} = A,\ \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}} = B$

Forward sensitivity (forward AD)

Consider $\mathbf{f}(\mathbf{x}, \mathbf{u})$ given by

Algorithm: function $\mathbf{f}(\mathbf{x}, \mathbf{u})$

Input: $\mathbf{x} \in \mathbb{R}^n, \ \mathbf{u} \in \mathbb{R}^m$
$\mathbf{s} = \mathbf{x}$
for $i = 1{:}N$ **do**
  $\mathbf{s} \leftarrow \boldsymbol{\xi}(\mathbf{s}, \mathbf{u})$
return $\mathbf{f}(\mathbf{x}, \mathbf{u}) = \mathbf{s}$

Note: e.g. $\boldsymbol{\xi}$ *reads as* $\mathbf{s} + \Delta t \mathbf{F}$ *in Euler*

Forward sensitivity computation:

Algorithm: function $\mathbf{f}(\mathbf{x}, \mathbf{u})$ with forward sensitivity

Input: $\mathbf{x} \in \mathbb{R}^n, \ \mathbf{u} \in \mathbb{R}^m$
$\mathbf{s} = \mathbf{x}, \ A = I, \ B = 0$
for $i = 1 : N$ **do**
  $A \leftarrow \frac{\partial \boldsymbol{\xi}(\mathbf{s},\mathbf{u})}{\partial \mathbf{s}} A$
  $B \leftarrow \frac{\partial \boldsymbol{\xi}(\mathbf{s},\mathbf{u})}{\partial \mathbf{s}} B + \frac{\partial \boldsymbol{\xi}(\mathbf{s},\mathbf{u})}{\partial \mathbf{u}}$
  $\mathbf{s} \leftarrow \boldsymbol{\zeta}(\mathbf{s}, \mathbf{u})$
return $\mathbf{f}(\mathbf{x}, \mathbf{u}) = \mathbf{s}, \ \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}} = A, \ \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}} = B$

- $A \in \mathbb{R}^{n \times n}, \ B \in \mathbb{R}^{n \times m}$, often dense
- Memory footprint $n(n+m)$ floating point numbers
- N dense matrix-matrix multiplications to build A, complexity Nn^3
- N dense matrix-matrix multiplications to build B, complexity $Nn^2 m$
- Total complexity $Nn^2(n+m)$

Forward sensitivity (forward AD)

Consider $\mathbf{f}(\mathbf{x},\mathbf{u})$ given by

Algorithm: function $\mathbf{f}(\mathbf{x},\mathbf{u})$

Input: $\mathbf{x} \in \mathbb{R}^n, \ \mathbf{u} \in \mathbb{R}^m$
$\mathbf{s} = \mathbf{x}$
for $i = 1{:}N$ **do**
  $\mathbf{s} \leftarrow \boldsymbol{\xi}(\mathbf{s},\mathbf{u})$
return $\mathbf{f}(\mathbf{x},\mathbf{u}) = \mathbf{s}$

Note: e.g. $\boldsymbol{\xi}$ *reads as* $\mathbf{s} + \Delta t \mathbf{F}$ *in Euler*

Forward sensitivity computation:

Algorithm: function $\mathbf{f}(\mathbf{x},\mathbf{u})$ with forward sensitivity

Input: $\mathbf{x} \in \mathbb{R}^n, \ \mathbf{u} \in \mathbb{R}^m$
$\mathbf{s} = \mathbf{x}, \ A = I, \ B = 0$
for $i = 1 : N$ **do**
  $A \leftarrow \frac{\partial \boldsymbol{\xi}(\mathbf{s},\mathbf{u})}{\partial \mathbf{s}} A$
  $B \leftarrow \frac{\partial \boldsymbol{\xi}(\mathbf{s},\mathbf{u})}{\partial \mathbf{s}} B + \frac{\partial \boldsymbol{\xi}(\mathbf{s},\mathbf{u})}{\partial \mathbf{u}}$
  $\mathbf{s} \leftarrow \boldsymbol{\zeta}(\mathbf{s},\mathbf{u})$
return $\mathbf{f}(\mathbf{x},\mathbf{u}) = \mathbf{s}, \ \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}} = A, \ \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}} = B$

- $A \in \mathbb{R}^{n\times n}, \ B \in \mathbb{R}^{n\times m}$, often dense
- Memory footprint $n(n+m)$ floating point numbers
- N dense matrix-matrix multiplications to build A, complexity Nn^3
- N dense matrix-matrix multiplications to build B, complexity Nn^2m
- Total complexity $Nn^2(n+m)$

What if we have a function $\zeta : \mathbb{R}^n \rightarrow \mathbb{R}$, and care only about $\frac{\partial \zeta(\mathbf{f}(\mathbf{x},\mathbf{u}))}{\partial \mathbf{x}} \in \mathbb{R}^n$, $\frac{\partial \zeta(\mathbf{f}(\mathbf{x},\mathbf{u}))}{\partial \mathbf{u}} \in \mathbb{R}^m$?

Adjoint-mode AD (reverse mode) - key idea

Consider $T(\mathbf{x}, \mathbf{u})$ given by

Algorithm: function $T(\mathbf{x}, \mathbf{u})$

Input: $\mathbf{s} \in \mathbb{R}^n, \quad \mathbf{u} \in \mathbb{R}^m$

$\mathbf{s}_0 = \mathbf{x}$

for $i = 1{:}N$ **do**

$\quad \mathbf{s}_i = \boldsymbol{\xi}(\mathbf{s}_{i-1}, \mathbf{u})$

return $T(\mathbf{x}, \mathbf{u}) = \zeta(\mathbf{s}_N)$

Can we compute $\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}$, $\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}}$ more efficiently than by doing forward sensitivity ? Yes, use the adjoint mode.

Adjoint-mode AD (reverse mode) - key idea

Consider $T(\mathbf{x}, \mathbf{u})$ given by

Algorithm: function $T(\mathbf{x}, \mathbf{u})$

Input: $\mathbf{s} \in \mathbb{R}^n, \quad \mathbf{u} \in \mathbb{R}^m$
$\mathbf{s}_0 = \mathbf{x}$
for $i = 1{:}N$ **do**
  $\mathbf{s}_i = \boldsymbol{\xi}(\mathbf{s}_{i-1}, \mathbf{u})$
return $T(\mathbf{x}, \mathbf{u}) = \zeta(\mathbf{s}_N)$

Define $\boldsymbol{\lambda}_i^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_i}$

Can we compute $\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}$, $\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}}$ more efficiently than by doing forward sensitivity ? Yes, use the adjoint mode.

Adjoint-mode AD (reverse mode) - key idea

Consider $T(\mathbf{x}, \mathbf{u})$ given by

Algorithm: function $T(\mathbf{x}, \mathbf{u})$

Input: $\mathbf{s} \in \mathbb{R}^n$, $\quad \mathbf{u} \in \mathbb{R}^m$
$\mathbf{s}_0 = \mathbf{x}$
for $i = 1{:}N$ **do**
$\quad \mathbf{s}_i = \boldsymbol{\xi}(\mathbf{s}_{i-1}, \mathbf{u})$
return $T(\mathbf{x}, \mathbf{u}) = \zeta(\mathbf{s}_N)$

Can we compute $\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}$, $\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}}$ more efficiently than by doing forward sensitivity ? Yes, use the adjoint mode.

Define $\boldsymbol{\lambda}_i^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_i}$, then

$$\boldsymbol{\lambda}_N^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_N}$$

Adjoint-mode AD (reverse mode) - key idea

Consider $T(\mathbf{x}, \mathbf{u})$ given by

Algorithm: function $T(\mathbf{x}, \mathbf{u})$

Input: $\mathbf{s} \in \mathbb{R}^n, \quad \mathbf{u} \in \mathbb{R}^m$

$\mathbf{s}_0 = \mathbf{x}$

for $i = 1{:}N$ **do**

$\quad \mathbf{s}_i = \boldsymbol{\xi}(\mathbf{s}_{i-1}, \mathbf{u})$

return $T(\mathbf{x}, \mathbf{u}) = \zeta(\mathbf{s}_N)$

Can we compute $\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}$, $\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}}$ more efficiently than by doing forward sensitivity ? Yes, use the adjoint mode.

Define $\boldsymbol{\lambda}_i^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_i}$, then

$$\boldsymbol{\lambda}_N^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_N}$$

$$\boldsymbol{\lambda}_{i-1}^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_{i-1}} = \underbrace{\frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_i}} \cdot \underbrace{\frac{\partial \mathbf{s}_i}{\partial \mathbf{s}_{i-1}}}$$

Adjoint-mode AD (reverse mode) - key idea

Consider $T(\mathbf{x}, \mathbf{u})$ given by

Algorithm: function $T(\mathbf{x}, \mathbf{u})$

Input: $\mathbf{s} \in \mathbb{R}^n, \quad \mathbf{u} \in \mathbb{R}^m$

$\mathbf{s}_0 = \mathbf{x}$

for $i = 1{:}N$ **do**

$\quad \mathbf{s}_i = \boldsymbol{\xi}(\mathbf{s}_{i-1}, \mathbf{u})$

return $T(\mathbf{x}, \mathbf{u}) = \zeta(\mathbf{s}_N)$

Can we compute $\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}, \frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}}$ more efficiently than by doing forward sensitivity ? Yes, use the adjoint mode.

Define $\boldsymbol{\lambda}_i^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_i}$, then

$$\boldsymbol{\lambda}_N^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_N}$$

$$\boldsymbol{\lambda}_{i-1}^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_{i-1}} = \underbrace{\frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_i}}_{\boldsymbol{\lambda}_i^\top} \cdot \underbrace{\frac{\partial \mathbf{s}_i}{\partial \mathbf{s}_{i-1}}}$$

Adjoint-mode AD (reverse mode) - key idea

Consider $T(\mathbf{x}, \mathbf{u})$ given by

Algorithm: function $T(\mathbf{x}, \mathbf{u})$

Input: $\mathbf{s} \in \mathbb{R}^n, \quad \mathbf{u} \in \mathbb{R}^m$

$\mathbf{s}_0 = \mathbf{x}$

for $i = 1{:}N$ **do**

$\quad \mathbf{s}_i = \boldsymbol{\xi}(\mathbf{s}_{i-1}, \mathbf{u})$

return $T(\mathbf{x}, \mathbf{u}) = \zeta(\mathbf{s}_N)$

Can we compute $\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}$, $\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}}$ more efficiently than by doing forward sensitivity ? Yes, use the adjoint mode.

Define $\boldsymbol{\lambda}_i^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_i}$, then

$$\boldsymbol{\lambda}_N^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_N}$$

$$\boldsymbol{\lambda}_{i-1}^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_{i-1}} = \underbrace{\frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_i}}_{\boldsymbol{\lambda}_i^\top} \cdot \underbrace{\frac{\partial \mathbf{s}_i}{\partial \mathbf{s}_{i-1}}}_{\frac{\partial \boldsymbol{\xi}(\mathbf{s}_{i-1},\mathbf{u})}{\partial \mathbf{s}_{i-1}}}$$

Adjoint-mode AD (reverse mode) - key idea

Consider $T(\mathbf{x},\mathbf{u})$ given by

Algorithm: function $T(\mathbf{x},\mathbf{u})$

Input: $\mathbf{s} \in \mathbb{R}^n, \quad \mathbf{u} \in \mathbb{R}^m$

$\mathbf{s}_0 = \mathbf{x}$

for $i = 1{:}N$ **do**

$\quad \mathbf{s}_i = \boldsymbol{\xi}(\mathbf{s}_{i-1},\mathbf{u})$

return $T(\mathbf{x},\mathbf{u}) = \zeta(\mathbf{s}_N)$

Can we compute $\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}, \frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}}$ more efficiently than by doing forward sensitivity ? Yes, use the adjoint mode.

Define $\boldsymbol{\lambda}_i^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_i}$, then

$$\boldsymbol{\lambda}_N^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_N}$$

$$\boldsymbol{\lambda}_{i-1}^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_{i-1}} = \underbrace{\frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_i}}_{\boldsymbol{\lambda}_i^\top} \cdot \underbrace{\frac{\partial \mathbf{s}_i}{\partial \mathbf{s}_{i-1}}}_{\frac{\partial \boldsymbol{\xi}(\mathbf{s}_{i-1},\mathbf{u})}{\partial \mathbf{s}_{i-1}}}$$

Moreover

$$\frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{u}} = \sum_{i=1}^{N} \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_i}\frac{\partial \mathbf{s}_i}{\partial \mathbf{u}} =$$

$$\sum_{i=1}^{N} \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_i}\frac{\partial \boldsymbol{\xi}(\mathbf{s}_{i-1},\mathbf{u})}{\partial \mathbf{u}} = \sum_{i=1}^{N} \boldsymbol{\lambda}_i{}^\top \frac{\partial \boldsymbol{\xi}(\mathbf{s}_{i-1},\mathbf{u})}{\partial \mathbf{u}}$$

Adjoint-mode AD (reverse mode) - key idea

Consider $T(\mathbf{x},\mathbf{u})$ given by

Algorithm: function $T(\mathbf{x},\mathbf{u})$

Input: $\mathbf{s} \in \mathbb{R}^n, \quad \mathbf{u} \in \mathbb{R}^m$
$\mathbf{s}_0 = \mathbf{x}$
for $i = 1{:}N$ **do**
  $\mathbf{s}_i = \boldsymbol{\xi}(\mathbf{s}_{i-1},\mathbf{u})$
return $T(\mathbf{x},\mathbf{u}) = \zeta(\mathbf{s}_N)$

Can we compute $\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}$, $\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}}$ more efficiently than by doing forward sensitivity ? Yes, use the adjoint mode.

Define $\boldsymbol{\lambda}_i^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_i}$, then

$$\boldsymbol{\lambda}_N^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_N}$$

$$\boldsymbol{\lambda}_{i-1}^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_{i-1}} = \underbrace{\frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_i}}_{\boldsymbol{\lambda}_i^\top} \cdot \underbrace{\frac{\partial \mathbf{s}_i}{\partial \mathbf{s}_{i-1}}}_{\frac{\partial \boldsymbol{\xi}(\mathbf{s}_{i-1},\mathbf{u})}{\partial \mathbf{s}_{i-1}}}$$

Moreover

$$\frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{u}} = \sum_{i=1}^{N} \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_i}\frac{\partial \mathbf{s}_i}{\partial \mathbf{u}} =$$

$$\sum_{i=1}^{N} \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_i}\frac{\partial \boldsymbol{\xi}(\mathbf{s}_{i-1},\mathbf{u})}{\partial \mathbf{u}} = \sum_{i=1}^{N} \boldsymbol{\lambda}_i{}^\top \frac{\partial \boldsymbol{\xi}(\mathbf{s}_{i-1},\mathbf{u})}{\partial \mathbf{u}}$$

$$\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}} = \boldsymbol{\lambda}_0{}^\top$$

$$\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}} = \sum_{i=1}^{N} \boldsymbol{\lambda}_i{}^\top \frac{\partial \boldsymbol{\xi}(\mathbf{s}_{i-1},\mathbf{u})}{\partial \mathbf{u}}$$

with

$$\boldsymbol{\lambda}_i^\top = \boldsymbol{\lambda}_{i+1}^\top \frac{\partial \boldsymbol{\xi}(\mathbf{s}_i,\mathbf{u})}{\partial \mathbf{s}_i}$$

$$\boldsymbol{\lambda}_N^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_N}$$

Adjoint-mode AD (reverse mode) - Implementation

Consider $T(\mathbf{x}, \mathbf{u})$ given by

Algorithm: function $T(\mathbf{x}, \mathbf{u})$

Input: $\mathbf{x} \in \mathbb{R}^n, \ \mathbf{u} \in \mathbb{R}^m$

$\mathbf{s}_0 = \mathbf{x}$

for $i = 1{:}N$ **do**

  $\mathbf{s}_i = \boldsymbol{\xi}(\mathbf{s}_{i-1}, \mathbf{u})$

return

  $T(\mathbf{x}, \mathbf{u}) = \zeta(\mathbf{s}_N), \ \mathbf{s}_{0,\dots,N}$

$$\frac{\partial T(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} = \boldsymbol{\lambda}_0{}^\top$$

$$\frac{\partial T(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} = \sum_{i=1}^{N} \boldsymbol{\lambda}_i{}^\top \frac{\partial \boldsymbol{\xi}(\mathbf{s}_{i-1}, \mathbf{u})}{\partial \mathbf{u}}$$

with

$$\boldsymbol{\lambda}_i^\top = \frac{\partial \boldsymbol{\xi}(\mathbf{s}_i, \mathbf{u})}{\partial \mathbf{s}_i} \boldsymbol{\lambda}_{i+1}^\top$$

$$\boldsymbol{\lambda}_N^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_N}$$

Adjoint-mode AD (reverse mode) - Implementation

Consider $T(\mathbf{x},\mathbf{u})$ given by

Algorithm: function $T(\mathbf{x},\mathbf{u})$

Input: $\mathbf{x} \in \mathbb{R}^n, \; \mathbf{u} \in \mathbb{R}^m$
$\mathbf{s}_0 = \mathbf{x}$
for $i = 1{:}N$ **do**
  $\mathbf{s}_i = \boldsymbol{\xi}(\mathbf{s}_{i-1},\mathbf{u})$
return
  $T(\mathbf{x},\mathbf{u}) = \zeta(\mathbf{s}_N), \; \mathbf{s}_{0,\dots,N}$

Algorithm: adjoint-mode sensitivity of function $T(\mathbf{x},\mathbf{u})$

Input: $\mathbf{s}_{0,\dots,N}, \; \mathbf{u}$
$\boldsymbol{\lambda}_N = \nabla\zeta(\mathbf{s}_N), \; \boldsymbol{\sigma} = 0$
for $i = N\text{-}1{:}0$ **do**
  $\boldsymbol{\sigma} = \boldsymbol{\sigma} + \nabla_{\mathbf{u}}\boldsymbol{\xi}(\mathbf{s}_i,\mathbf{u})\,\boldsymbol{\lambda}_{i+1}$
  $\boldsymbol{\lambda}_i = \nabla_{\mathbf{s}_i}\boldsymbol{\xi}(\mathbf{s}_i,\mathbf{u})\,\boldsymbol{\lambda}_{i+1}$
return $\nabla_{\mathbf{x}} T(\mathbf{x},\mathbf{u}) = \boldsymbol{\lambda}_0, \quad \nabla_{\mathbf{u}} T(\mathbf{x},\mathbf{u}) = \boldsymbol{\sigma}$

$$\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}} = \boldsymbol{\lambda}_0{}^\top$$
$$\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}} = \sum_{i=1}^{N} \boldsymbol{\lambda}_i{}^\top \frac{\partial \boldsymbol{\xi}(\mathbf{s}_{i-1},\mathbf{u})}{\partial \mathbf{u}}$$

with

$$\boldsymbol{\lambda}_i^\top = \frac{\partial \boldsymbol{\xi}(\mathbf{s}_i,\mathbf{u})}{\partial \mathbf{s}_i}\boldsymbol{\lambda}_{i+1}^\top$$
$$\boldsymbol{\lambda}_N^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_N}$$

Adjoint-mode AD (reverse mode) - Implementation

Consider $T(\mathbf{x}, \mathbf{u})$ given by

Algorithm: function $T(\mathbf{x}, \mathbf{u})$

Input: $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{u} \in \mathbb{R}^m$
$\mathbf{s}_0 = \mathbf{x}$
for $i = 1{:}N$ **do**
  $\mathbf{s}_i = \boldsymbol{\xi}(\mathbf{s}_{i-1}, \mathbf{u})$
return
  $T(\mathbf{x}, \mathbf{u}) = \zeta(\mathbf{s}_N)$, $\mathbf{s}_{0,\dots,N}$

Algorithm: adjoint-mode sensitivity of function $T(\mathbf{x}, \mathbf{u})$

Input: $\mathbf{s}_{0,\dots,N}$, $\mathbf{u}$
$\boldsymbol{\lambda} = \nabla\zeta(\mathbf{s}_N)$, $\boldsymbol{\sigma} = 0$
for $i = N\text{-}1{:}0$ **do**
  $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma} + \nabla_{\mathbf{u}}\boldsymbol{\xi}(\mathbf{s}_i, \mathbf{u})\boldsymbol{\lambda}$
  $\boldsymbol{\lambda} \leftarrow \nabla_{\mathbf{s}_i}\boldsymbol{\xi}(\mathbf{s}_i, \mathbf{u})\boldsymbol{\lambda}$
return $\nabla_{\mathbf{x}} T(\mathbf{x}, \mathbf{u}) = \boldsymbol{\lambda}$, $\nabla_{\mathbf{u}} T(\mathbf{x}, \mathbf{u}) = \boldsymbol{\sigma}$

$$\frac{\partial T(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} = \boldsymbol{\lambda}_0{}^\top$$

$$\frac{\partial T(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} = \sum_{i=1}^{N} \boldsymbol{\lambda}_i{}^\top \frac{\partial \boldsymbol{\xi}(\mathbf{s}_{i-1}, \mathbf{u})}{\partial \mathbf{u}}$$

with

$$\boldsymbol{\lambda}_i^\top = \frac{\partial \boldsymbol{\xi}(\mathbf{s}_i, \mathbf{u})}{\partial \mathbf{s}_i}\boldsymbol{\lambda}_{i+1}^\top$$

$$\boldsymbol{\lambda}_N^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_N}$$

Adjoint-mode AD (reverse mode) - Implementation

Consider $T(\mathbf{x},\mathbf{u})$ given by

Algorithm: function $T(\mathbf{x},\mathbf{u})$

Input: $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{u} \in \mathbb{R}^m$
$\mathbf{s}_0 = \mathbf{x}$
for $i = 1{:}N$ **do**
  $\mathbf{s}_i = \boldsymbol{\xi}(\mathbf{s}_{i-1},\mathbf{u})$
return
  $T(\mathbf{x},\mathbf{u}) = \zeta(\mathbf{s}_N)$, $\mathbf{s}_{0,\dots,N}$

Algorithm: adjoint-mode sensitivity of function $T(\mathbf{x},\mathbf{u})$

Input: $\mathbf{s}_{0,\dots,N}$, $\mathbf{u}$
$\boldsymbol{\lambda} = \nabla\zeta(\mathbf{s}_N)$, $\boldsymbol{\sigma} = 0$
for $i = N\text{-}1{:}0$ **do**
  $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma} + \nabla_{\mathbf{u}}\boldsymbol{\xi}(\mathbf{s}_i,\mathbf{u})\,\boldsymbol{\lambda}$
  $\boldsymbol{\lambda} \leftarrow \nabla_{\mathbf{s}_i}\boldsymbol{\xi}(\mathbf{s}_i,\mathbf{u})\,\boldsymbol{\lambda}$
return $\nabla_{\mathbf{x}} T(\mathbf{x},\mathbf{u}) = \boldsymbol{\lambda}$, $\nabla_{\mathbf{u}} T(\mathbf{x},\mathbf{u}) = \boldsymbol{\sigma}$

- Two passes algorithm !! First do the forward sweep (compute $\mathbf{s}_{0,\dots,N}$), then do the backward sweep (compute $\boldsymbol{\lambda}$, $\boldsymbol{\sigma}$)
- Forward sweep needs to store $\mathbf{s}_{0,\dots,N}$, memory footprint Nn.
- Forward sweep inexpensive
- Backward sweep requires N matrix-vector multiplication to build $\boldsymbol{\lambda}$, complexity n^2
- Backward sweep requires N matrix-vector multiplication to build $\boldsymbol{\sigma}$, complexity np
- Total complexity $Nn(n+p)$, i.e. <u>n times less than the forward mode</u>

$$\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}} = \boldsymbol{\lambda}_0^\top$$

$$\frac{\partial T(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}} = \sum_{i=1}^{N} \boldsymbol{\lambda}_i^\top \frac{\partial \boldsymbol{\xi}(\mathbf{s}_{i-1},\mathbf{u})}{\partial \mathbf{u}}$$

with

$$\boldsymbol{\lambda}_i^\top = \frac{\partial \boldsymbol{\xi}(\mathbf{s}_i,\mathbf{u})}{\partial \mathbf{s}_i}\boldsymbol{\lambda}_{i+1}^\top$$

$$\boldsymbol{\lambda}_N^\top = \frac{\partial \zeta(\mathbf{s}_N)}{\partial \mathbf{s}_N}$$

Outline

Why do we need the 2^{nd}-order sensitivities of integrators ?

NLP with multiple-shooting

$$\min_{\mathbf{w}} \quad \Phi(\mathbf{w})$$

$$\text{s.t.} \quad \mathbf{g}(\mathbf{w}) = \begin{bmatrix} \mathbf{x}_0 - \bar{\mathbf{x}}_0 \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) - \mathbf{x}_1 \\ \mathbf{f}(\mathbf{x}_1, \mathbf{u}_1) - \mathbf{x}_2 \\ \dots \end{bmatrix}$$

with

$$\mathbf{w} = \{\mathbf{x}_0, \mathbf{u}_0, \dots, \mathbf{x}_{N-1}, \mathbf{u}_{N-1}, \mathbf{x}_N\}$$

Why do we need the 2^{nd}-order sensitivities of integrators ?

NLP with multiple-shooting

$$\begin{aligned} \min_{\mathbf{w}} \quad & \Phi(\mathbf{w}) \\ \text{s.t.} \quad & \mathbf{g}(\mathbf{w}) = \begin{bmatrix} \mathbf{x}_0 - \bar{\mathbf{x}}_0 \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) - \mathbf{x}_1 \\ \mathbf{f}(\mathbf{x}_1, \mathbf{u}_1) - \mathbf{x}_2 \\ \dots \end{bmatrix} \end{aligned}$$

with

$$\mathbf{w} = \{\mathbf{x}_0, \mathbf{u}_0, \dots, \mathbf{x}_{N-1}, \mathbf{u}_{N-1}, \mathbf{x}_N\}$$

SQP iterates:

$$\begin{aligned} \min_{\Delta\mathbf{w}} \quad & \frac{1}{2}\Delta\mathbf{w}^\top B \Delta\mathbf{w} + \nabla\Phi^\top \Delta\mathbf{w} \\ \text{s.t.} \quad & \nabla\mathbf{g}^\top \Delta\mathbf{w} + \mathbf{g} = 0 \end{aligned}$$

where $B = \nabla^2_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \boldsymbol{\lambda})$

Why do we need the 2^{nd}-order sensitivities of integrators ?

NLP with multiple-shooting

$$\min_{\mathbf{w}} \quad \Phi(\mathbf{w})$$

$$\text{s.t.} \quad \mathbf{g}(\mathbf{w}) = \begin{bmatrix} \mathbf{x}_0 - \bar{\mathbf{x}}_0 \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) - \mathbf{x}_1 \\ \mathbf{f}(\mathbf{x}_1, \mathbf{u}_1) - \mathbf{x}_2 \\ \dots \end{bmatrix}$$

with

$$\mathbf{w} = \{\mathbf{x}_0, \mathbf{u}_0, \dots, \mathbf{x}_{N-1}, \mathbf{u}_{N-1}, \mathbf{x}_N\}$$

Reminder:

$$\mathcal{L}(\mathbf{w}, \boldsymbol{\lambda}) = \phi(\mathbf{w}) + \sum_{k=0}^{N-1} \boldsymbol{\lambda}_{k+1}{}^\top (\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) - \mathbf{x}_{k+1}) + \boldsymbol{\lambda}_0{}^\top (\mathbf{x}_0 - \bar{\mathbf{x}}_0)$$

SQP iterates:

$$\min_{\Delta\mathbf{w}} \quad \frac{1}{2}\Delta\mathbf{w}^\top B \Delta\mathbf{w} + \nabla\Phi^\top \Delta\mathbf{w}$$

$$\text{s.t.} \quad \nabla\mathbf{g}^\top \Delta\mathbf{w} + \mathbf{g} = 0$$

where $B = \nabla^2_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \boldsymbol{\lambda})$

Why do we need the 2$^{\text{nd}}$-order sensitivities of integrators ?

NLP with multiple-shooting

$$\min_{\mathbf{w}} \quad \Phi(\mathbf{w})$$

$$\text{s.t.} \quad \mathbf{g}(\mathbf{w}) = \begin{bmatrix} \mathbf{x}_0 - \bar{\mathbf{x}}_0 \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) - \mathbf{x}_1 \\ \mathbf{f}(\mathbf{x}_1, \mathbf{u}_1) - \mathbf{x}_2 \\ \ldots \end{bmatrix}$$

with

$$\mathbf{w} = \{\mathbf{x}_0, \mathbf{u}_0, \ldots, \mathbf{x}_{N-1}, \mathbf{u}_{N-1}, \mathbf{x}_N\}$$

SQP iterates:

$$\min_{\Delta\mathbf{w}} \quad \frac{1}{2}\Delta\mathbf{w}^\top B \Delta\mathbf{w} + \nabla\Phi^\top \Delta\mathbf{w}$$

$$\text{s.t.} \quad \nabla\mathbf{g}^\top \Delta\mathbf{w} + \mathbf{g} = 0$$

where $B = \nabla^2_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \boldsymbol{\lambda})$

Reminder:

$$\mathcal{L}(\mathbf{w}, \boldsymbol{\lambda}) = \phi(\mathbf{w}) + \sum_{k=0}^{N-1} \boldsymbol{\lambda}_{k+1}{}^\top (\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) - \mathbf{x}_{k+1}) + \boldsymbol{\lambda}_0{}^\top (\mathbf{x}_0 - \bar{\mathbf{x}}_0)$$

Then

$$\nabla^2_{\mathbf{w}_k} \mathcal{L}(\mathbf{w}, \boldsymbol{\lambda}) = \nabla^2_{\mathbf{w}_k} \phi(\mathbf{w}) + \sum_{k=0}^{N-1} \nabla^2_{\mathbf{w}_k} \left(\boldsymbol{\lambda}_{k+1}{}^\top \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)\right)$$

with $\mathbf{w}_k = \{\mathbf{x}_k, \mathbf{u}_k\}, \;\; k = 0, \ldots, N-1$

Why do we need the 2$^{\text{nd}}$-order sensitivities of integrators ?

NLP with multiple-shooting

$$\min_{\mathbf{w}} \quad \Phi(\mathbf{w})$$

$$\text{s.t.} \quad \mathbf{g}(\mathbf{w}) = \begin{bmatrix} \mathbf{x}_0 - \bar{\mathbf{x}}_0 \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) - \mathbf{x}_1 \\ \mathbf{f}(\mathbf{x}_1, \mathbf{u}_1) - \mathbf{x}_2 \\ \cdots \end{bmatrix}$$

with

$$\mathbf{w} = \{\mathbf{x}_0, \mathbf{u}_0, \ldots, \mathbf{x}_{N-1}, \mathbf{u}_{N-1}, \mathbf{x}_N\}$$

SQP iterates:

$$\min_{\Delta\mathbf{w}} \quad \frac{1}{2}\Delta\mathbf{w}^\top B \Delta\mathbf{w} + \nabla\Phi^\top \Delta\mathbf{w}$$

$$\text{s.t.} \quad \nabla\mathbf{g}^\top \Delta\mathbf{w} + \mathbf{g} = 0$$

where $B = \nabla^2_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \boldsymbol{\lambda})$

Reminder:

$$\mathcal{L}(\mathbf{w}, \boldsymbol{\lambda}) = \phi(\mathbf{w}) + \sum_{k=0}^{N-1} \boldsymbol{\lambda}_{k+1}{}^\top \left(\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) - \mathbf{x}_{k+1}\right) + \boldsymbol{\lambda}_0{}^\top (\mathbf{x}_0 - \bar{\mathbf{x}}_0)$$

Then

$$\nabla^2_{\mathbf{w}_k} \mathcal{L}(\mathbf{w}, \boldsymbol{\lambda}) = \nabla^2_{\mathbf{w}_k} \phi(\mathbf{w}) + \sum_{k=0}^{N-1} \nabla^2_{\mathbf{w}_k} \left(\boldsymbol{\lambda}_{k+1}{}^\top \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)\right)$$

with $\mathbf{w}_k = \{\mathbf{x}_k, \mathbf{u}_k\}$, $k = 0, \ldots, N-1$

To deploy an SQP method with exact Hessian, we need to compute

$$\nabla^2_{\mathbf{w}_k} \left(\boldsymbol{\lambda}_{k+1}{}^\top \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)\right)$$

for $k = 0, \ldots, N-1$!!

Second-order integrator sensitivity

To make it simple we ignore the input $\mathbf{u}$. Let's look at:

$$T(\mathbf{x}, \boldsymbol{\mu}) = \boldsymbol{\mu}^\top \mathbf{f}(\mathbf{x}) \quad (\in \mathbb{R})$$

We can compute the sensitivity of T using the adjoint mode, then:

$$\nabla_{\mathbf{x}} T(\mathbf{x}, \boldsymbol{\mu}) = \boldsymbol{\lambda}$$

and

$$\nabla_{\mathbf{x}}^2 T(\mathbf{x}, \boldsymbol{\mu}) = \nabla_{\mathbf{x}} \boldsymbol{\lambda}$$

Second-order integrator sensitivity

To make it simple we ignore the input $\mathbf{u}$. Let's look at:

$$T(\mathbf{x}, \boldsymbol{\mu}) = \boldsymbol{\mu}^\top \mathbf{f}(\mathbf{x}) \quad (\in \mathbb{R})$$

We can compute the sensitivity of T using the adjoint mode, then:

$$\nabla_{\mathbf{x}} T(\mathbf{x}, \boldsymbol{\mu}) = \boldsymbol{\lambda}$$

and

$$\nabla^2_{\mathbf{x}} T(\mathbf{x}, \boldsymbol{\mu}) = \nabla_{\mathbf{x}} \boldsymbol{\lambda}$$

Algorithm: Forward AD

Input: $\mathbf{x}$

$\mathbf{s}_0 = \mathbf{x}$, $A_0 = I$

for i = *0:N-1* **do**

$$A_{i+1} = \frac{\partial \boldsymbol{\xi}(\mathbf{s}_i)}{\partial \mathbf{s}_i} A_i$$

$$\mathbf{s}_{i+1} = \boldsymbol{\xi}(\mathbf{s}_i)$$

return $\mathbf{s}_{0,\ldots,N}$, $A_{0,\ldots,N}$

Second-order integrator sensitivity

To make it simple we ignore the input $\mathbf{u}$. Let's look at:

$$T\left(\mathbf{x}, \boldsymbol{\mu}\right) = \boldsymbol{\mu}^\top \mathbf{f}\left(\mathbf{x}\right) \quad (\in \mathbb{R})$$

We can compute the sensitivity of T using the adjoint mode, then:

$$\nabla_{\mathbf{x}} T\left(\mathbf{x}, \boldsymbol{\mu}\right) = \boldsymbol{\lambda}$$

and

$$\nabla^2_{\mathbf{x}} T\left(\mathbf{x}, \boldsymbol{\mu}\right) = \nabla_{\mathbf{x}} \boldsymbol{\lambda}$$

Algorithm: Forward AD

Input: $\mathbf{x}$
$\mathbf{s}_0 = \mathbf{x}$, $A_0 = I$
for $i = 0{:}N\text{-}1$ **do**
$\quad A_{i+1} = \frac{\partial \boldsymbol{\xi}(\mathbf{s}_i)}{\partial \mathbf{s}_i} A_i$
$\quad \mathbf{s}_{i+1} = \boldsymbol{\xi}\left(\mathbf{s}_i\right)$
return $\mathbf{s}_{0,\ldots,N}$, $A_{0,\ldots,N}$

Reminder (here $\zeta\left(\mathbf{x}\right) = \boldsymbol{\mu}^\top \mathbf{x}$):

Algorithm: Adjoint-mode AD

Input: $\mathbf{s}_{0,\ldots,N}$, $\boldsymbol{\mu}$
$\boldsymbol{\lambda} = \nabla \zeta\left(\mathbf{s}_N\right) = \boldsymbol{\mu}$
for $i = N\text{-}1{:}0$ **do**
$\quad \boldsymbol{\lambda} \leftarrow \nabla_{\mathbf{s}_i} \boldsymbol{\xi}\left(\mathbf{s}_i\right) \boldsymbol{\lambda}$
return $\boldsymbol{\lambda}$

Second-order integrator sensitivity

To make it simple we ignore the input $\mathbf{u}$. Let's look at:

$$T(\mathbf{x}, \boldsymbol{\mu}) = \boldsymbol{\mu}^\top \mathbf{f}(\mathbf{x}) \quad (\in \mathbb{R})$$

We can compute the sensitivity of T using the adjoint mode, then:

$$\nabla_{\mathbf{x}} T(\mathbf{x}, \boldsymbol{\mu}) = \boldsymbol{\lambda}$$

and

$$\nabla^2_{\mathbf{x}} T(\mathbf{x}, \boldsymbol{\mu}) = \nabla_{\mathbf{x}} \boldsymbol{\lambda}$$

Algorithm: Forward AD

Input: $\mathbf{x}$
$\mathbf{s}_0 = \mathbf{x}$, $A_0 = I$
for i = *0:N-1* **do**
$\quad A_{i+1} = \frac{\partial \boldsymbol{\xi}(\mathbf{s}_i)}{\partial \mathbf{s}_i} A_i$
$\quad \mathbf{s}_{i+1} = \boldsymbol{\xi}(\mathbf{s}_i)$
return $\mathbf{s}_{0,\ldots,N}$, $A_{0,\ldots,N}$

Reminder (here $\zeta(\mathbf{x}) = \boldsymbol{\mu}^\top \mathbf{x}$):

Algorithm: Adjoint-mode AD

Input: $\mathbf{s}_{0,\ldots,N}$, $\boldsymbol{\mu}$
$\boldsymbol{\lambda} = \nabla \zeta(\mathbf{s}_N) = \boldsymbol{\mu}$
for i = *N-1:0* **do**
$\quad \boldsymbol{\lambda} \leftarrow \nabla_{\mathbf{s}_i} \boldsymbol{\xi}(\mathbf{s}_i) \boldsymbol{\lambda}$
return $\boldsymbol{\lambda}$

Key idea: perform forward sensitivity on the adjoint-mode algorithm (note: it runs backward in time !!)

Second-order integrator sensitivity

To make it simple we ignore the input $\mathbf{u}$. Let's look at:

$$T(\mathbf{x}, \boldsymbol{\mu}) = \boldsymbol{\mu}^\top \mathbf{f}(\mathbf{x}) \quad (\in \mathbb{R})$$

We can compute the sensitivity of T using the adjoint mode, then:

$$\nabla_{\mathbf{x}} T(\mathbf{x}, \boldsymbol{\mu}) = \boldsymbol{\lambda}$$

and

$$\nabla^2_{\mathbf{x}} T(\mathbf{x}, \boldsymbol{\mu}) = \nabla_{\mathbf{x}} \boldsymbol{\lambda}$$

Algorithm: Forward Over Adjoints

Input: $\mathbf{s}_{0,\dots,N}$, $A_{0,\dots,N-1}$, $\boldsymbol{\mu}$

$\boldsymbol{\lambda}_N = \boldsymbol{\mu}$, $H_N = 0$

for i = *N-1:0* **do**

- $H_i = \nabla^2_{\mathbf{s}_i}\left(\boldsymbol{\lambda}_{i+1}{}^\top \boldsymbol{\xi}(\mathbf{s}_i)\right) A_i + \nabla_{\mathbf{s}_i}\boldsymbol{\xi}(\mathbf{s}_i) H_{i+1}$
- $\boldsymbol{\lambda}_i = \nabla_{\mathbf{s}_i}\boldsymbol{\xi}(\mathbf{s}_i)\boldsymbol{\lambda}_{i+1}$

return $\nabla^2_{\mathbf{x}} T = H_0$

Algorithm: Forward AD

Input: $\mathbf{x}$

$\mathbf{s}_0 = \mathbf{x}$, $A_0 = I$

for i = *0:N-1* **do**

- $A_{i+1} = \frac{\partial \boldsymbol{\xi}(\mathbf{s}_i)}{\partial \mathbf{s}_i} A_i$
- $\mathbf{s}_{i+1} = \boldsymbol{\xi}(\mathbf{s}_i)$

return $\mathbf{s}_{0,\dots,N}$, $A_{0,\dots,N}$

Reminder (here $\zeta(\mathbf{x}) = \boldsymbol{\mu}^\top \mathbf{x}$):

Algorithm: Adjoint-mode AD

Input: $\mathbf{s}_{0,\dots,N}$, $\boldsymbol{\mu}$

$\boldsymbol{\lambda} = \nabla \zeta(\mathbf{s}_N) = \boldsymbol{\mu}$

for i = *N-1:0* **do**

- $\boldsymbol{\lambda} \leftarrow \nabla_{\mathbf{s}_i}\boldsymbol{\xi}(\mathbf{s}_i)\boldsymbol{\lambda}$

return $\boldsymbol{\lambda}$

Key idea: perform forward sensitivity on the adjoint-mode algorithm (note: it runs backward in time !!)

Second-order integrator sensitivity

To make it simple we ignore the input $\mathbf{u}$. Let's look at:

$$T(\mathbf{x}, \boldsymbol{\mu}) = \boldsymbol{\mu}^\top \mathbf{f}(\mathbf{x}) \quad (\in \mathbb{R})$$

We can compute the sensitivity of T using the adjoint mode, then:

$$\nabla_{\mathbf{x}} T(\mathbf{x}, \boldsymbol{\mu}) = \boldsymbol{\lambda}$$

and

$$\nabla^2_{\mathbf{x}} T(\mathbf{x}, \boldsymbol{\mu}) = \nabla_{\mathbf{x}} \boldsymbol{\lambda}$$

Algorithm: Forward Over Adjoints (F.O.A.)

Input: $\mathbf{s}_{0,\dots,N}$, $A_{0,\dots,N-1}$ and $\boldsymbol{\mu}$
$\boldsymbol{\lambda} = \boldsymbol{\mu}$, $H = 0$
for $i = N\text{-}1{:}0$ **do**
- $H = \nabla^2_{\mathbf{s}_i}\left(\boldsymbol{\lambda}^\top \boldsymbol{\xi}(\mathbf{s}_i)\right) A_i + \nabla_{\mathbf{s}_i} \boldsymbol{\xi}(\mathbf{s}_i) H$
- $\boldsymbol{\lambda} = \nabla_{\mathbf{s}_i} \boldsymbol{\xi}(\mathbf{s}_i) \boldsymbol{\lambda}$

return $\nabla^2_{\mathbf{x}} T = H$

Algorithm: Forward AD

Input: $\mathbf{x}$
$\mathbf{s}_0 = \mathbf{x}$, $A_0 = I$
for $i = 0{:}N\text{-}1$ **do**
- $A_{i+1} = \frac{\partial \boldsymbol{\xi}(\mathbf{s}_i)}{\partial \mathbf{s}_i} A_i$
- $\mathbf{s}_{i+1} = \boldsymbol{\xi}(\mathbf{s}_i)$

return $\mathbf{s}_{0,\dots,N}$, $A_{0,\dots,N}$

Reminder (here $\zeta(\mathbf{x}) = \boldsymbol{\mu}^\top \mathbf{x}$):

Algorithm: Adjoint-mode AD

Input: $\mathbf{s}_{0,\dots,N}$, $\boldsymbol{\mu}$
$\boldsymbol{\lambda} = \nabla \zeta(\mathbf{s}_N) = \boldsymbol{\mu}$
for $i = N\text{-}1{:}0$ **do**
- $\boldsymbol{\lambda} \leftarrow \nabla_{\mathbf{s}_i} \boldsymbol{\xi}(\mathbf{s}_i) \boldsymbol{\lambda}$

return $\boldsymbol{\lambda}$

Key idea: perform forward sensitivity on the adjoint-mode algorithm (note: it runs backward in time !!)

Second-order integrator sensitivity

To make it simple we ignore the input $\mathbf{u}$. Let's look at:

$$T(\mathbf{x}, \boldsymbol{\mu}) = \boldsymbol{\mu}^\top \mathbf{f}(\mathbf{x}) \quad (\in \mathbb{R})$$

We can compute the sensitivity of T using the adjoint mode, then:

$$\nabla_{\mathbf{x}} T(\mathbf{x}, \boldsymbol{\mu}) = \boldsymbol{\lambda}$$

and

$$\nabla^2_{\mathbf{x}} T(\mathbf{x}, \boldsymbol{\mu}) = \nabla_{\mathbf{x}} \boldsymbol{\lambda}$$

Algorithm: Forward Over Adjoints (F.O.A.)

Input: $\mathbf{s}_{0,\ldots,N}$, $A_{0,\ldots,N-1}$ and $\boldsymbol{\mu}$

$\boldsymbol{\lambda} = \boldsymbol{\mu}$, $H = 0$

for $i = N\text{-}1\text{:}0$ **do**

$\quad H = \nabla^2_{\mathbf{s}_i}\left(\boldsymbol{\lambda}^\top \boldsymbol{\xi}(\mathbf{s}_i)\right) A_i + \nabla_{\mathbf{s}_i} \boldsymbol{\xi}(\mathbf{s}_i) H$

$\quad \boldsymbol{\lambda} = \nabla_{\mathbf{s}_i} \boldsymbol{\xi}(\mathbf{s}_i) \boldsymbol{\lambda}$

return $\nabla^2_{\mathbf{x}} T = H$

- Forward AD pass, store $\mathbf{s}_{0,\ldots,N}$, $A_{0,\ldots,N}$
- F.O.A pass, return $\nabla^2_{\mathbf{x}} T(\mathbf{x}, \boldsymbol{\lambda})$
- Lower computational complexity possible with reformulation (CDC 2013) !!

Algorithm: Forward AD

Input: $\mathbf{x}$

$\mathbf{s}_0 = \mathbf{x}$, $A_0 = I$

for $i = 0\text{:}N\text{-}1$ **do**

$\quad A_{i+1} = \frac{\partial \boldsymbol{\xi}(\mathbf{s}_i)}{\partial \mathbf{s}_i} A_i$

$\quad \mathbf{s}_{i+1} = \boldsymbol{\xi}(\mathbf{s}_i)$

return $\mathbf{s}_{0,\ldots,N}$, $A_{0,\ldots,N}$

Reminder (here $\zeta(\mathbf{x}) = \boldsymbol{\mu}^\top \mathbf{x}$):

Algorithm: Adjoint-mode AD

Input: $\mathbf{s}_{0,\ldots,N}$, $\boldsymbol{\mu}$

$\boldsymbol{\lambda} = \nabla \zeta(\mathbf{s}_N) = \boldsymbol{\mu}$

for $i = N\text{-}1\text{:}0$ **do**

$\quad \boldsymbol{\lambda} \leftarrow \nabla_{\mathbf{s}_i} \boldsymbol{\xi}(\mathbf{s}_i) \boldsymbol{\lambda}$

return $\boldsymbol{\lambda}$

Key idea: perform forward sensitivity on the adjoint-mode algorithm (note: it runs backward in time !!)