A Novel Computational Framework for the Numerical Solution of Complex Constrained Optimal Control Problems

Anil V. Rao, Ph.D.

Department of Mechanical and Aerospace Engineering
University of Florida
Gainesville, FL 32611-6250

Website: http://www.anilvrao.com
E-mail: anilvrao@gmail.com

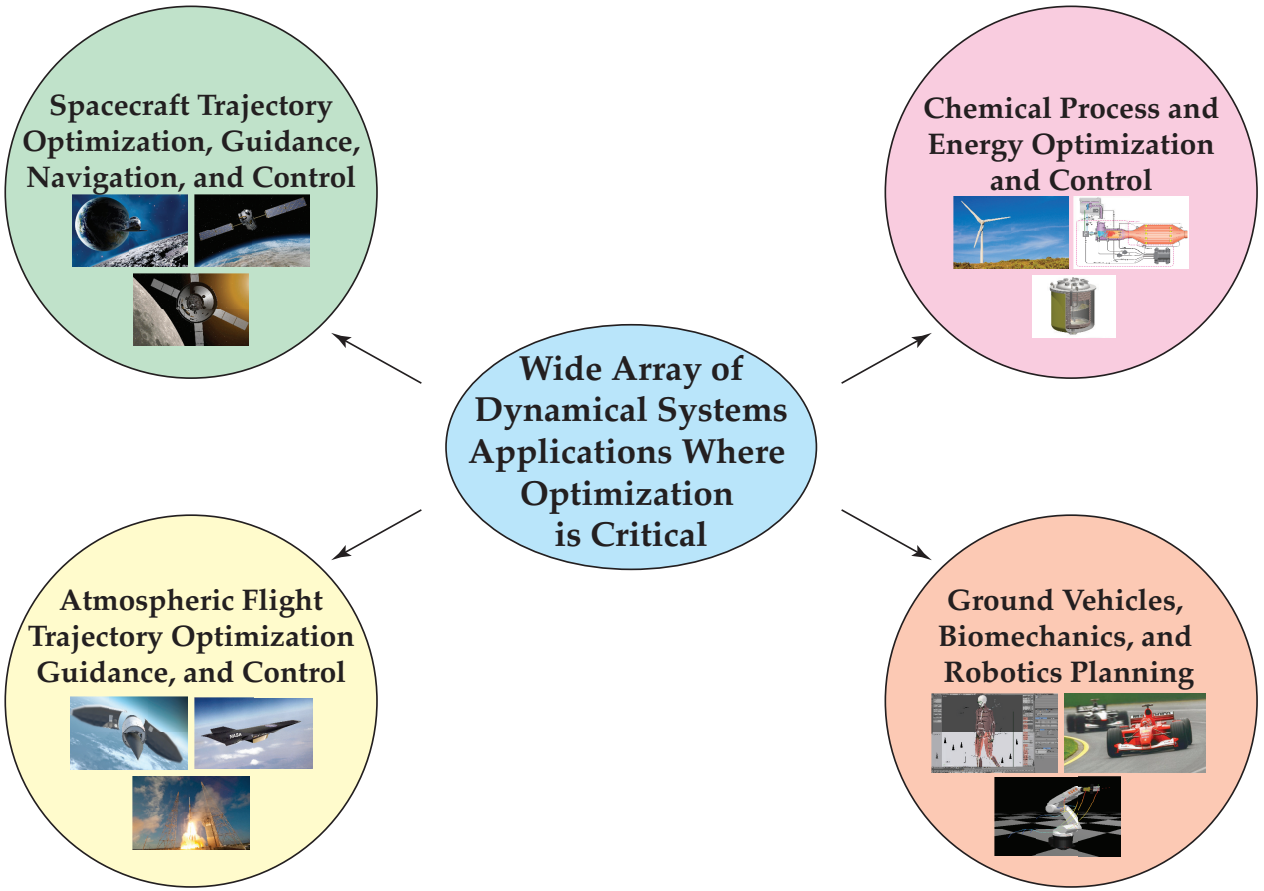
Presentation Given at the CBMS Workshop on Optimal Control
Jackson State University

23 July 2018

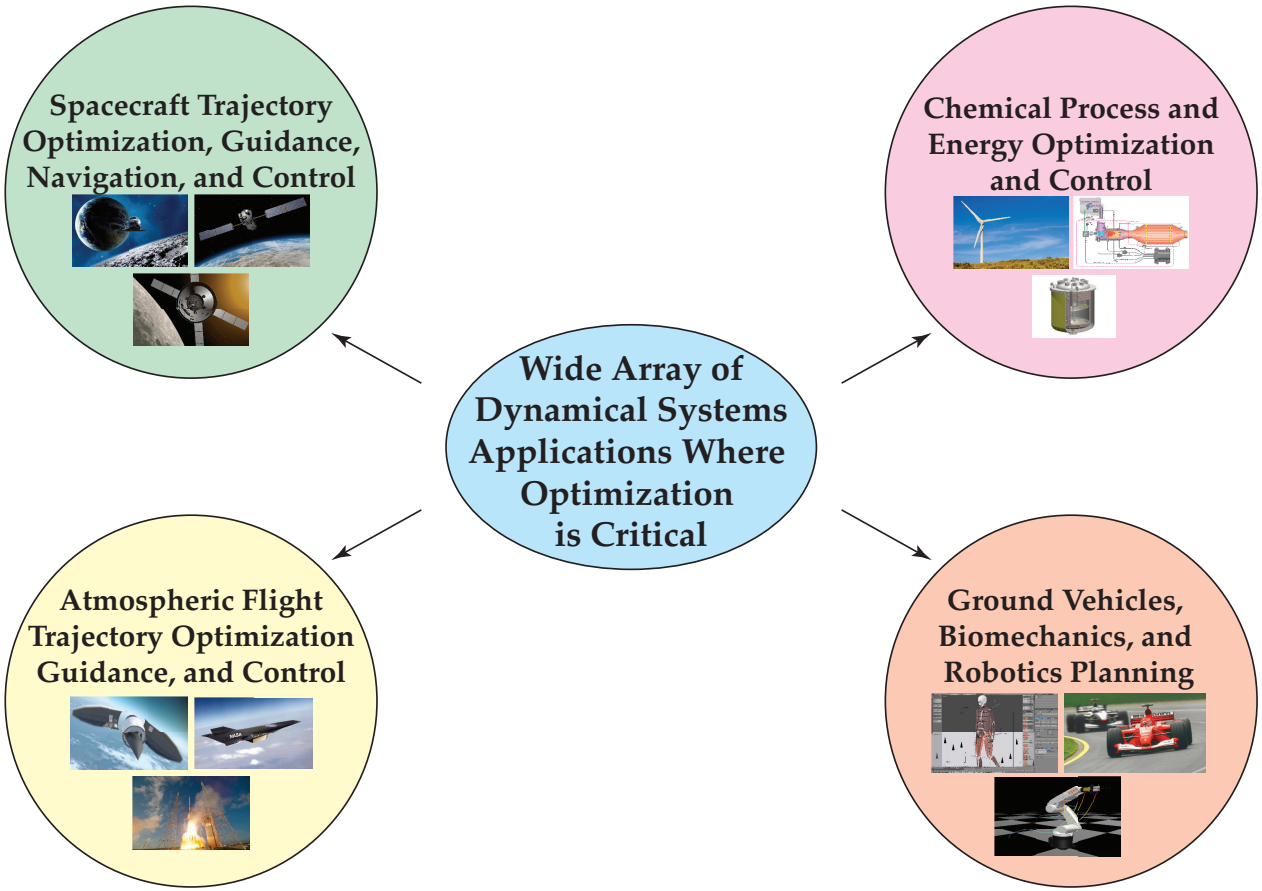
Outline of Presentation

- Motivation and thrust areas of research
- Optimal control
  - Overview of numerical methods for optimal control
  - Rationale for direct collocation methods
- hp-adaptive Gaussian quadrature collocation
  - Use of Gauss quadrature points in integration and interpolation
  - Formulation of direct collocation using hp-adaptive approach
  - Demonstration of advantages of hp-adaptive approach
- Algorithmic differentiation
  - Overview of algorithmic differentiation
  - New research in algorithmic differentiation
  - Demonstration of new algorithmic differentiation method
- Other applications of optimal control framework and software
- Future research, conclusions, and acknowledgments
- Note: important citations appear in magenta

Motivation: Optimization Critical in Controlled Dynamical Systems

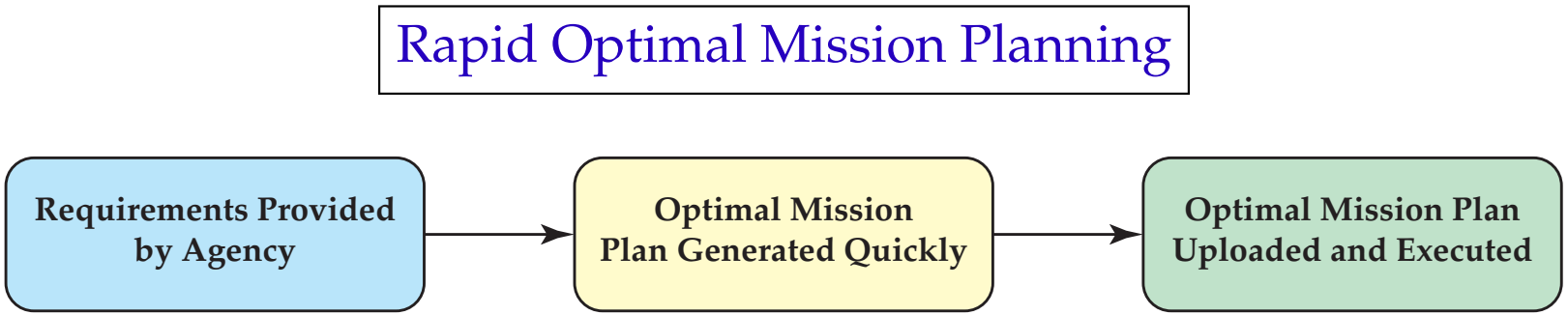


Motivation: Optimization Critical in Controlled Dynamical Systems



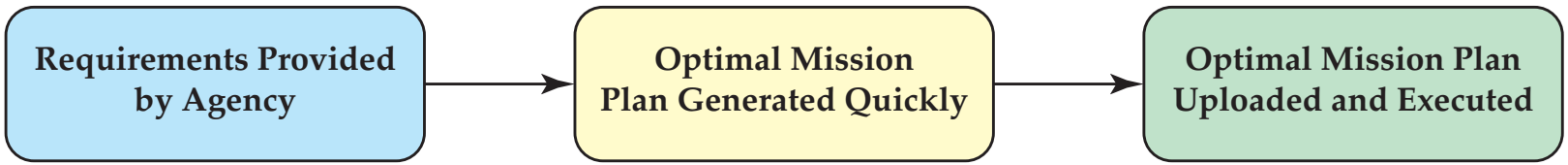
- Optimization of controlled dynamical system $\leftrightarrow$ **optimal control**
- **High-accuracy** solutions must be generated **rapidly**
- **Challenging problems** with **large numbers of constraints**

Rapid Optimal Mission Planning

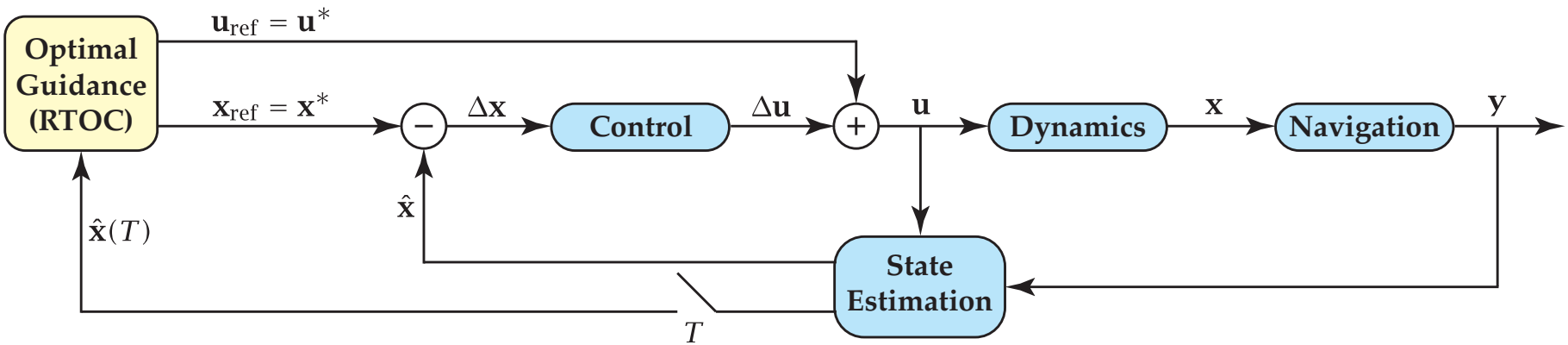


Rapid Optimal Mission Planning

Rapid Optimal Mission Planning

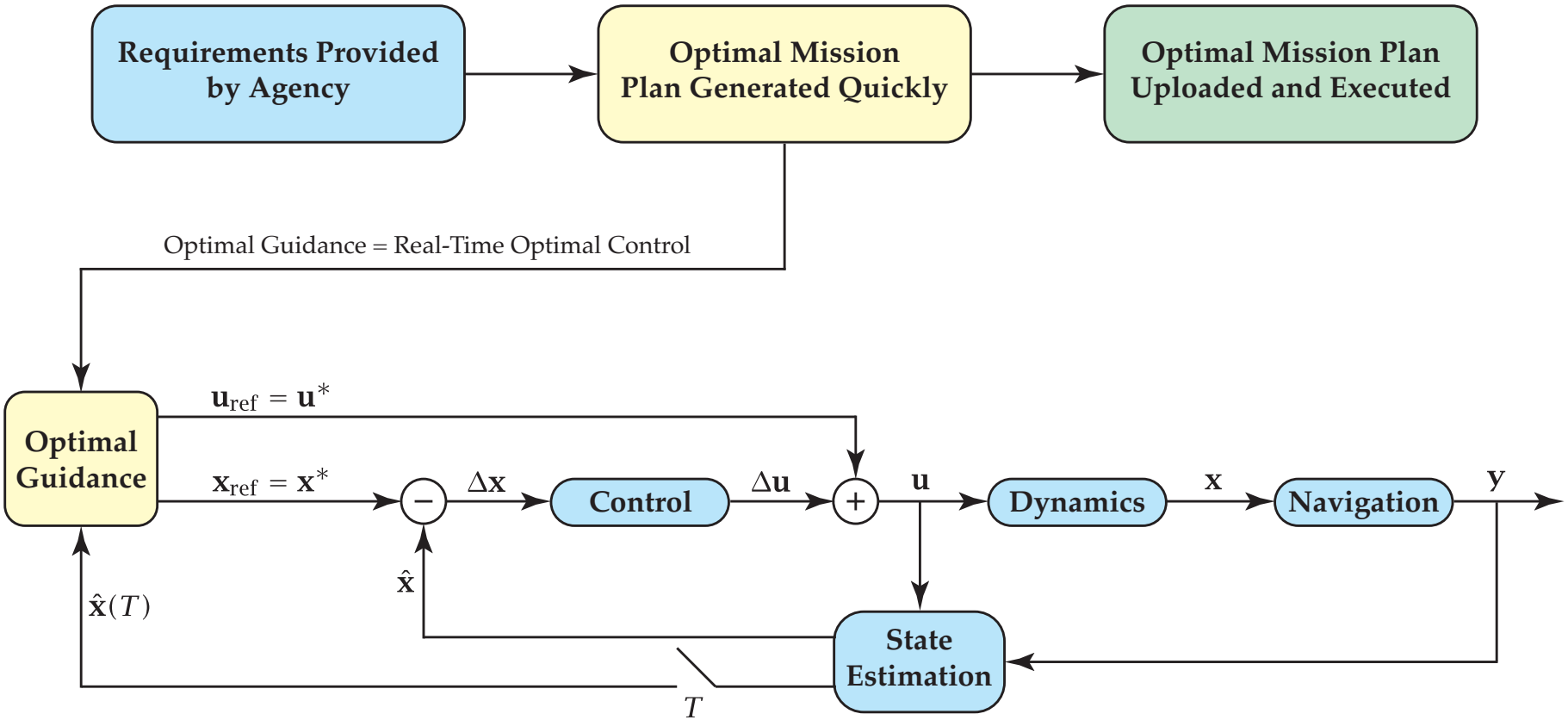


Optimal Guidance⇔Real-Time Optimal Control



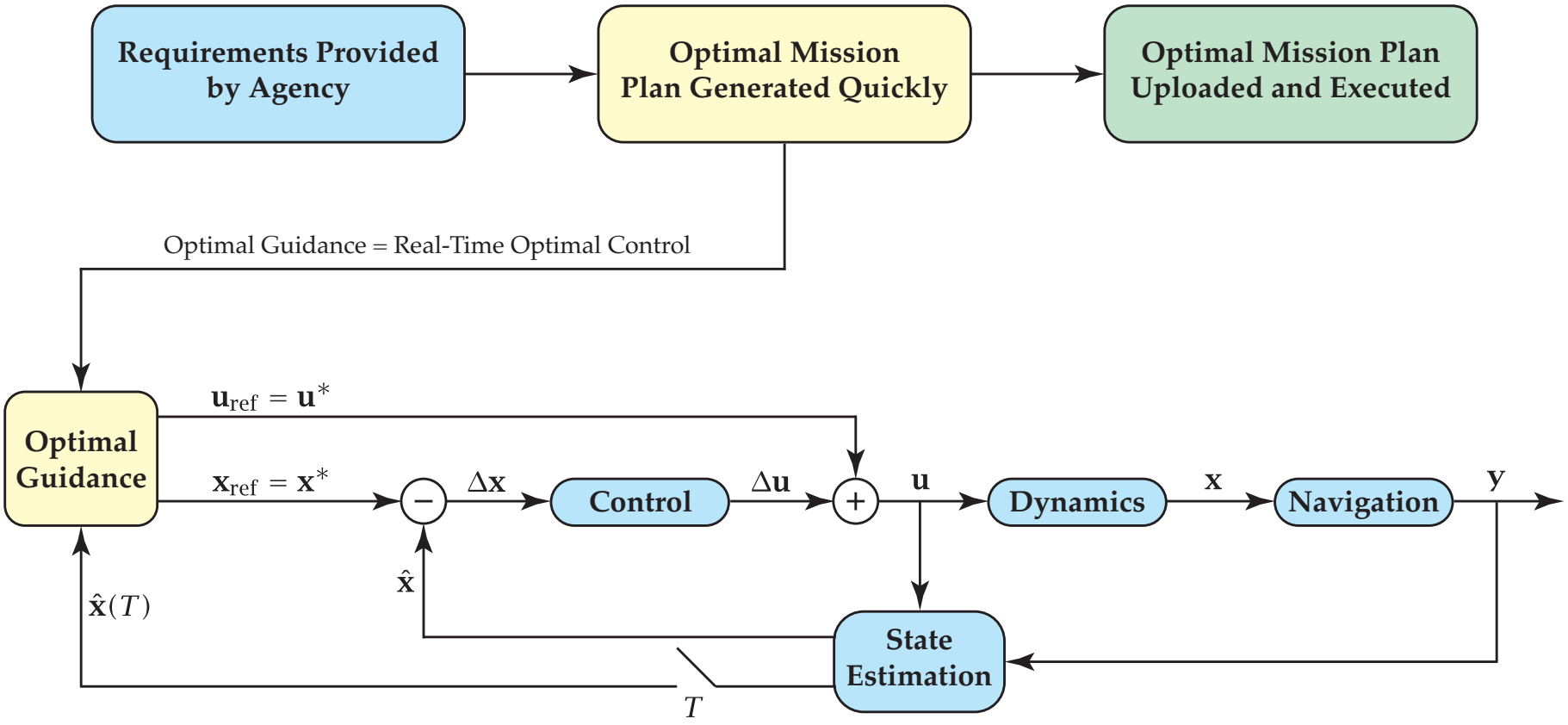
Rapid Optimal Mission Planning

Sufficiently Fast Rapid Optimal Mission Planning $\Leftrightarrow$ Real-Time Optimal Control



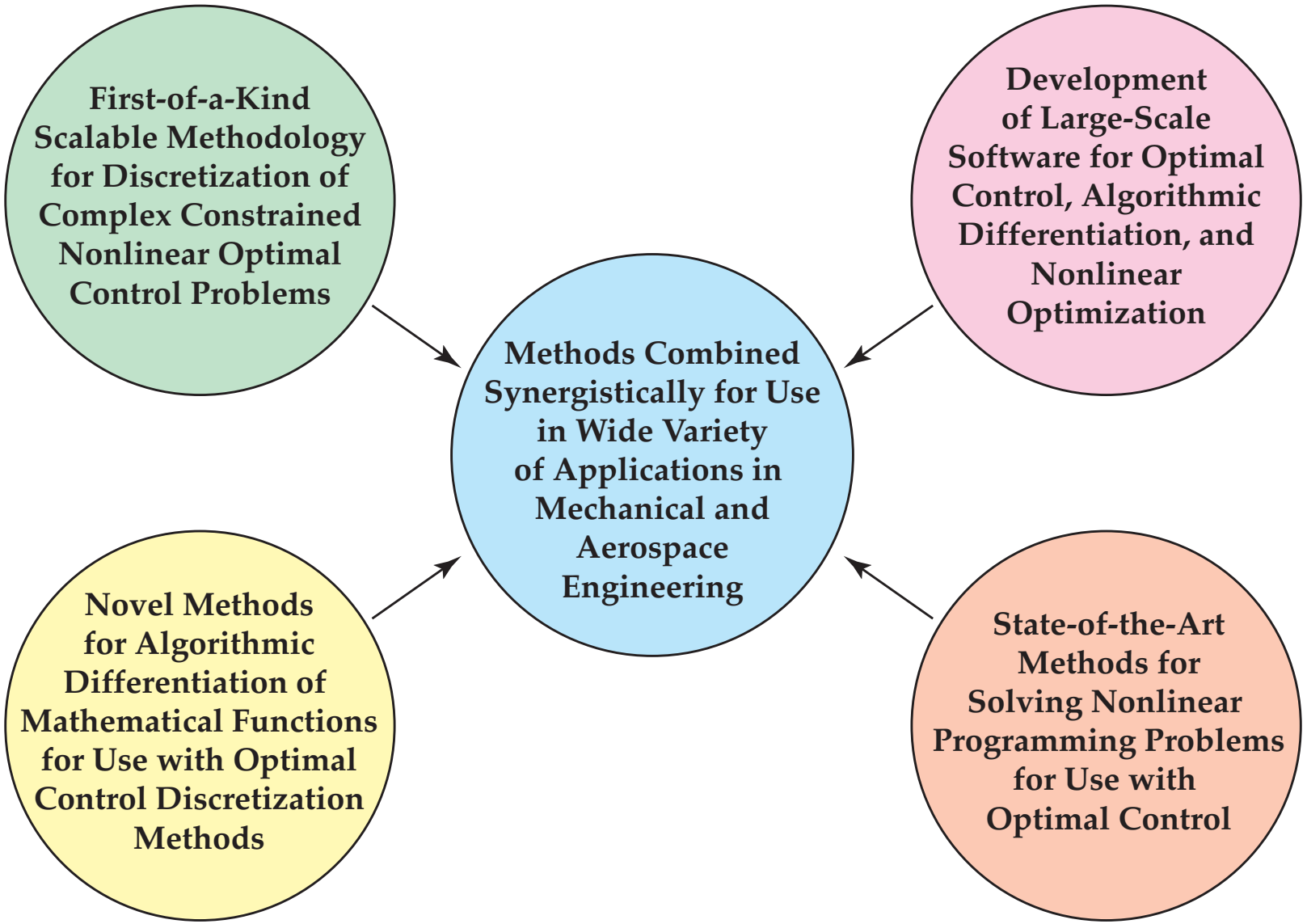
Rapid Optimal Mission Planning

Sufficiently Fast Rapid Optimal Mission Planning $\Leftrightarrow$ Real-Time Optimal Control



Objective: to develop a novel computational framework for rapidly computing high-accuracy solutions to complex constrained nonlinear optimal control problems.

Research Program Thrust Areas



What is an Optimal Control Problem?

- **Definition:** *The goal of an optimal control problem is to determine the state and control of a controlled dynamical system, along with the initial and terminal time, that optimize a specified performance index while satisfying the dynamic constraints, the path constraints, and the boundary conditions.*

*Betts, J. T., *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, SIAM Press, Philadelphia, 2010

What is an Optimal Control Problem?

- **Definition:** *The goal of an optimal control problem is to determine the state and control of a controlled dynamical system, along with the initial and terminal time, that optimize a specified performance index while satisfying the dynamic constraints, the path constraints, and the boundary conditions.*
- Minimize $\mathcal{M}(\mathbf{y}(t_0), t_0, \mathbf{y}(t_f), t_f) + \int_{t_0}^{t_f} \mathcal{L}(\mathbf{y}(t), \mathbf{u}(t), t)dt$ subject to

$$\begin{aligned} \dot{\mathbf{y}}(t) - \mathbf{a}(\mathbf{y}(t), \mathbf{u}(t), t) &= \mathbf{0}, \\ \mathbf{b}(\mathbf{y}(t_0), t_0, \mathbf{y}(t_f), t_f) &= \mathbf{0}, \\ \mathbf{c}(\mathbf{y}(t), \mathbf{u}(t), t) &\leq \mathbf{0}. \end{aligned}$$

*Betts, J. T., *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, SIAM Press, Philadelphia, 2010

What is an Optimal Control Problem?

- **Definition:** *The goal of an optimal control problem is to determine the state and control of a controlled dynamical system, along with the initial and terminal time, that optimize a specified performance index while satisfying the dynamic constraints, the path constraints, and the boundary conditions.*
- Minimize $\mathcal{M}(\mathbf{y}(t_0), t_0, \mathbf{y}(t_f), t_f) + \int_{t_0}^{t_f} \mathcal{L}(\mathbf{y}(t), \mathbf{u}(t), t)dt$ subject to

$$\begin{aligned} \dot{\mathbf{y}}(t) - \mathbf{a}(\mathbf{y}(t), \mathbf{u}(t), t) &= \mathbf{0}, \\ \mathbf{b}(\mathbf{y}(t_0), t_0, \mathbf{y}(t_f), t_f) &= \mathbf{0}, \\ \mathbf{c}(\mathbf{y}(t), \mathbf{u}(t), t) &\leq \mathbf{0}. \end{aligned}$$

- Some key first-order optimality conditions

$$\begin{aligned} (\dot{\mathbf{y}}(t), \dot{\boldsymbol{\lambda}}(t)) &= (\mathcal{H}_{\boldsymbol{\lambda}}^{\mathsf{T}}, -\mathcal{H}_{\mathbf{y}}^{\mathsf{T}}),\ \mathcal{H} = \mathcal{L} + \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{a} - \boldsymbol{\mu}^{\mathsf{T}}\mathbf{c} = \text{Augmented Hamiltonian}, \\ \mathbf{u}^*(t) &= \arg\min_{\mathbf{u}\in\mathcal{U}} \mathcal{H}(\mathbf{y}^*(t), \mathbf{u}(t), \boldsymbol{\lambda}^*(t), t) \rightarrow \text{Minimum Principle}, \\ \boldsymbol{\lambda}^{\mathsf{T}}(t_0) &= -\mathcal{M}_{\mathbf{y}(t_0)} + \boldsymbol{\nu}^{\mathsf{T}}\mathbf{b}_{\mathbf{y}(t_0)}, \\ \boldsymbol{\lambda}^{\mathsf{T}}(t_f) &= \mathcal{M}_{\mathbf{y}(t_f)} - \boldsymbol{\nu}^{\mathsf{T}}\mathbf{b}_{\mathbf{y}(t_f)}. \end{aligned}$$

*Betts, J. T., *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, SIAM Press, Philadelphia, 2010

What is an Optimal Control Problem?

- **Definition:** *The goal of an optimal control problem is to determine the state and control of a controlled dynamical system, along with the initial and terminal time, that optimize a specified performance index while satisfying the dynamic constraints, the path constraints, and the boundary conditions.*
- Minimize $\mathcal{M}(\mathbf{y}(t_0), t_0, \mathbf{y}(t_f), t_f) + \int_{t_0}^{t_f} \mathcal{L}(\mathbf{y}(t), \mathbf{u}(t), t)dt$ subject to

$$\begin{aligned} \dot{\mathbf{y}}(t) - \mathbf{a}(\mathbf{y}(t), \mathbf{u}(t), t) &= \mathbf{0}, \\ \mathbf{b}(\mathbf{y}(t_0), t_0, \mathbf{y}(t_f), t_f) &= \mathbf{0}, \\ \mathbf{c}(\mathbf{y}(t), \mathbf{u}(t), t) &\leq \mathbf{0}. \end{aligned}$$

- Some key first-order optimality conditions

$$\begin{aligned} (\dot{\mathbf{y}}(t), \dot{\boldsymbol{\lambda}}(t)) &= (\mathcal{H}_{\boldsymbol{\lambda}}^{\mathsf{T}}, -\mathcal{H}_{\mathbf{y}}^{\mathsf{T}}),\ \mathcal{H} = \mathcal{L} + \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{a} - \boldsymbol{\mu}^{\mathsf{T}}\mathbf{c} = \text{Augmented Hamiltonian}, \\ \mathbf{u}^*(t) &= \arg\min_{\mathbf{u}\in\mathcal{U}} \mathcal{H}(\mathbf{y}^*(t), \mathbf{u}(t), \boldsymbol{\lambda}^*(t), t) \rightarrow \text{Minimum Principle}, \\ \boldsymbol{\lambda}^{\mathsf{T}}(t_0) &= -\mathcal{M}_{\mathbf{y}(t_0)} + \boldsymbol{\nu}^{\mathsf{T}}\mathbf{b}_{\mathbf{y}(t_0)}, \\ \boldsymbol{\lambda}^{\mathsf{T}}(t_f) &= \mathcal{M}_{\mathbf{y}(t_f)} - \boldsymbol{\nu}^{\mathsf{T}}\mathbf{b}_{\mathbf{y}(t_f)}. \end{aligned}$$

- Betts (2010): "Solving an optimal control problem is not easy."*

*Betts, J. T., *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, SIAM Press, Philadelphia, 2010

Numerical Methods for Optimal Control

Continuous Optimal Control Problem

Numerical Methods

Indirect Methods

- Apply Calculus of Variations
- Derive Hamiltonian Boundary-Value Problem (HBVP)
- Solve Hamiltonian Boundary-Value Problem Numerically

Pros

- Produce Accurate Solutions When They Converge
- Founded on Rigorous Mathematical Theory
- Possible to Verify the Proximity of Solution to True Extremal

Cons

- Hamiltonian Systems Difficult If Not Impossible to Solve
- Require Guesses of Non-Intuitive Costate Boundary Conditions
- Hamiltonian Systems are Unstable

Direct Methods

- Employ Control and/or State and Control Parameterization
- Optimal Control Problem Transcribed to NLP
- NLP Solved Using Established Optimization Software

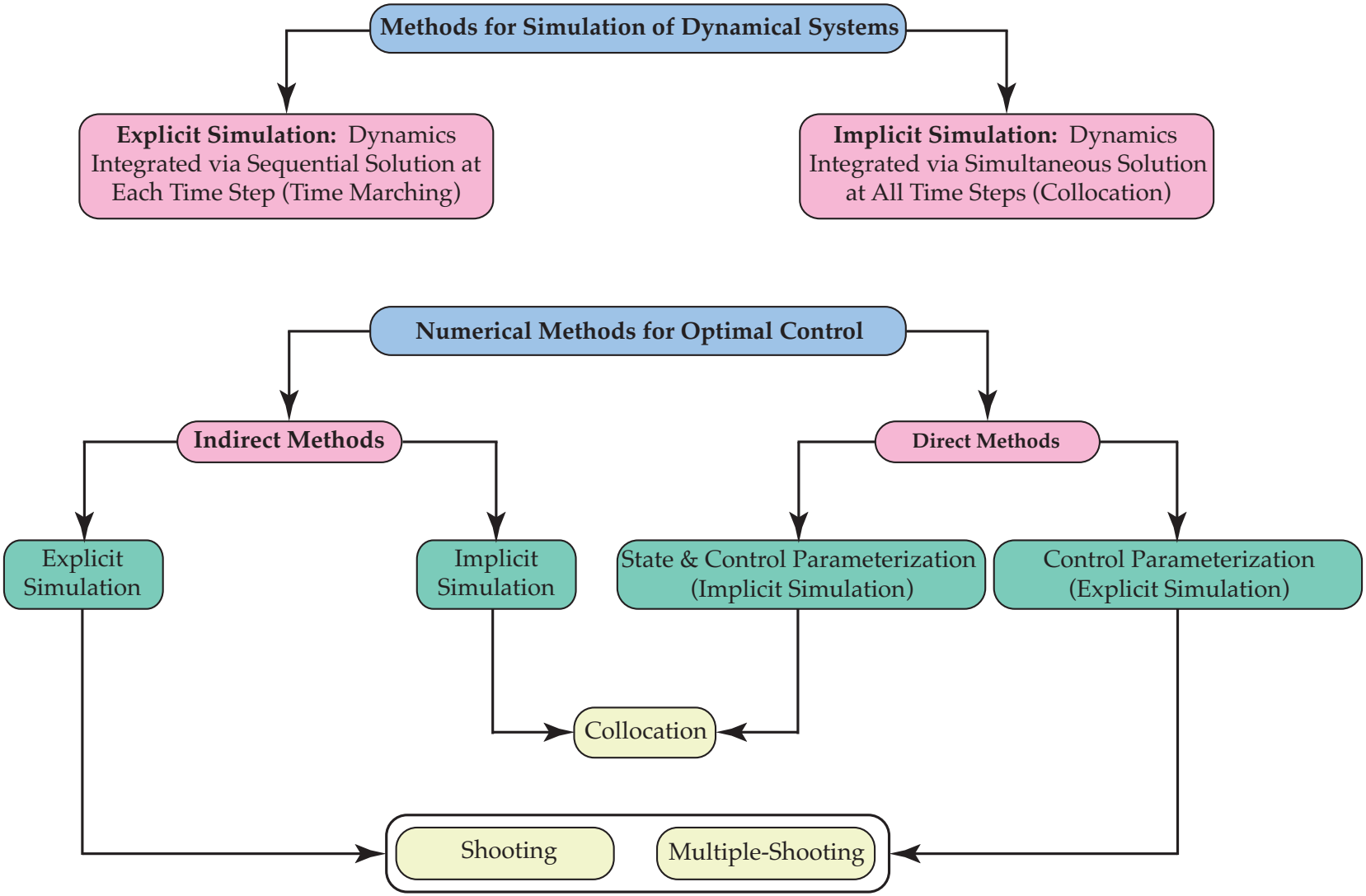
Pros

- No Need to Derive Complicated Optimality Conditions
- Some Methods Enable Straightforward Problem Modifications
- Allows for General Problem Formulation (Method-Dependent)

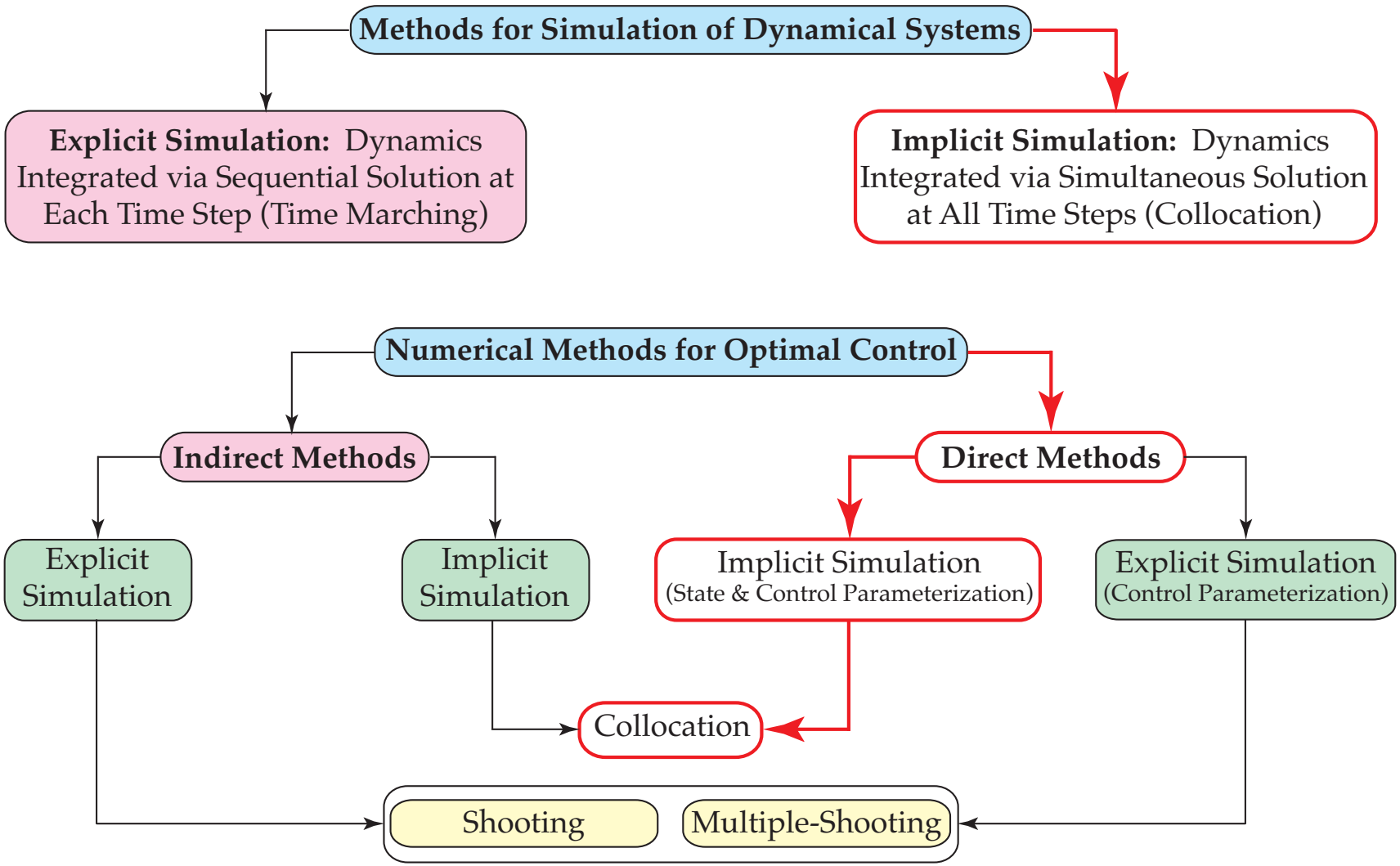
Cons

- ***Traditional Approaches*** are Not Mathematically Rigorous
- ***Traditional Approaches*** Produce Low-Accuracy Solutions
- ***Traditional Approaches*** are Computationally Inefficient

Breakdown of Indirect and Direct Methods



Breakdown of Indirect and Direct Methods



- Direct collocation ↔ implicit simulation ↔ transcription to large NLP

Why Direct Collocation for Optimal Control?

- Issues with indirect and direct time-marching (explicit simulation)
  - Indirect
    - Hamiltonian systems unstable in either forward or reverse time
    - Need to know switching structure in path constraints and control a priori
    - Cannot pose a general problem formulation
  - Direct
    - Path constraints cannot be included
    - Low-accuracy solutions even with no path constraints
    - Essentially, need to know solution before solving problem

Why Direct Collocation for Optimal Control?

- Issues with indirect and direct time-marching (explicit simulation)
  - Indirect
    - Hamiltonian systems unstable in either forward or reverse time
    - Need to know switching structure in path constraints and control a priori
    - Cannot pose a general problem formulation
  - Direct
    - Path constraints cannot be included
    - Low-accuracy solutions even with no path constraints
    - Essentially, need to know solution before solving problem
- Issues with indirect collocation (implicit simulation)
  - Like indirect and direct time-marching, cannot handle path constraints
  - Unable to solve problems where control is discontinuous

Why Direct Collocation for Optimal Control?

- Issues with indirect and direct time-marching (explicit simulation)
  - Indirect
    - Hamiltonian systems unstable in either forward or reverse time
    - Need to know switching structure in path constraints and control a priori
    - Cannot pose a general problem formulation
  - Direct
    - Path constraints cannot be included
    - Low-accuracy solutions even with no path constraints
    - Essentially, need to know solution before solving problem
- Issues with indirect collocation (implicit simulation)
  - Like indirect and direct time-marching, cannot handle path constraints
  - Unable to solve problems where control is discontinuous
- Advantages of direct collocation over time-marching
  - Leads to large *sparse* NLPs which can be solved efficiently
  - Inclusion of path constraints straightforward
  - Allows for highly general problem formulation

Example of (Low-Accuracy) Direct Collocation: Euler Forward

- Euler Forward using explicit simulation: $(y_{k+1} - y_k)/h = f(y_k, u_k, t_k)$
- Rewriting in *collocation* or *implicit simulation* form gives

$$\left.\begin{array}{rcl} y_1 - y_0 - hf(y_0, u_0, t_0) & = & 0 \\ y_2 - y_1 - hf(y_1, u_1, t_1) & = & 0 \\ & \vdots & \\ y_M - y_{M-1} - hf(y_{M-1}, u_{M-1}, t_{M-1}) & = & 0 \end{array}\right\} \begin{array}{c}\text{Equality Constraints} \\ \text{Solved Simultaneously for} \\ (y_0, \ldots, y_M) \\ (u_0, \ldots, u_{M-1})\end{array}$$

- Leads to a large sparse nonlinear programming problem. Minimize

$$\left.\begin{array}{lrcl} \text{Minimize} & J & = & \mathcal{M}(y_0, t_0, y_M, t_M) + h\sum_{k=1}^{M-1} \mathcal{L}[y_i, u_i, t_i] \\ \text{subject to} & & & \\ & 0 & = & y_1 - y_0 - hf(y_0, u_0, t_0), \\ & 0 & = & y_2 - y_1 - hf(y_1, u_1, t_1), \\ & & \vdots & \\ & 0 & = & y_M - y_{M-1} - hf(y_{M-1}, u_{M-1}, t_{M-1}), \\ & \mathbf{0} & = & \mathbf{b}(y_0, t_0, y_M, t_M) \end{array}\right\} \begin{array}{l}\text{NLP:} \\ \\ \begin{array}{lrcl} \text{Minimize} & f(\mathbf{z}) & & \\ \text{subject to} & \mathbf{h}(\mathbf{z}) & = & \mathbf{0}, \\ & \mathbf{h}(\mathbf{z}) & \leq & \mathbf{0} \end{array}\end{array}$$

Collocation: A Non-Intuitive Huge Step Forward

- Let n_y, n_u, and n_c be size of state, control, and path constraints
- As M gets large
  - NLP variables $\approx M(n_y + n_u)$; NLP constraints $\approx N n_y$
  - Suppose $M = 1000$, $n_y = 6$, and $n_u = 3 \longrightarrow M(n_y + n_u) \approx 9000$
- This is a *large* NLP
  - Thousands to tens of thousands variables and constraints
  - NLP using direct collocation is *huge*
- Upon first glance, looks like problem is now much harder
  - Note, however, that collocation methods are *sparse* systems
  - Sparse $\leftrightarrow$ large percentage of NLP derivatives are *zero*
  - Advanced NLP solvers designed to solve *large sparse* problems
- Sparsity $\leftrightarrow$ can efficiently solve direct collocation NLP
- Result: direct implicit simulation $\leftrightarrow$ direct collocation is preferred

Traditional Collocation Methods: h and p Methods

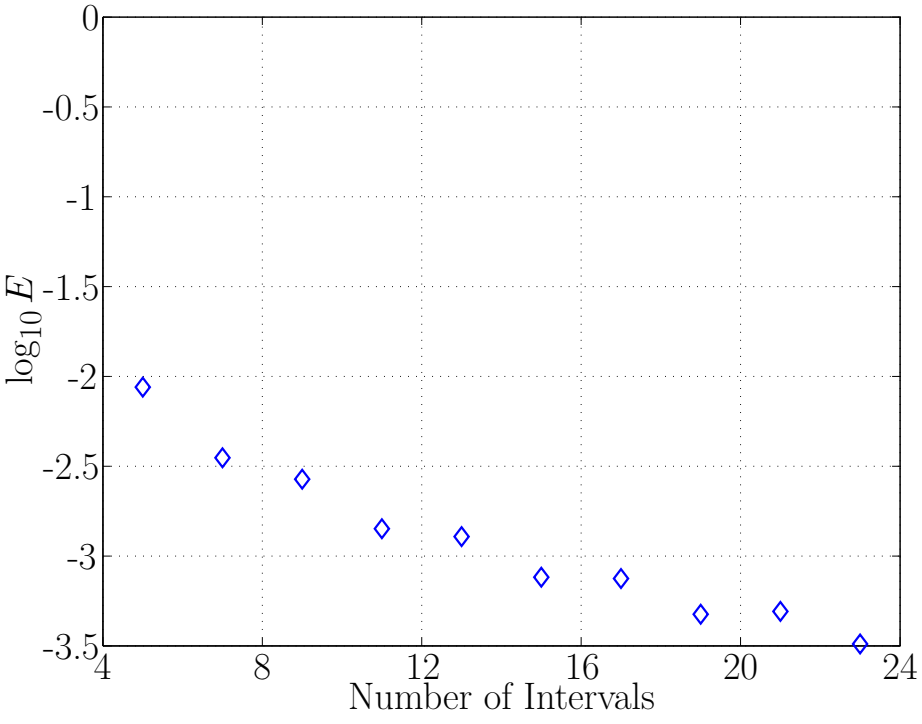
Method	Order	For High Accuracy	NLP	Efficiency
h	Fixed	Extremely Fine Mesh	Huge NLP	Extremely Inefficient
p	Variable	Unreasonably Large Degree	Large Dense NLP	Intractable

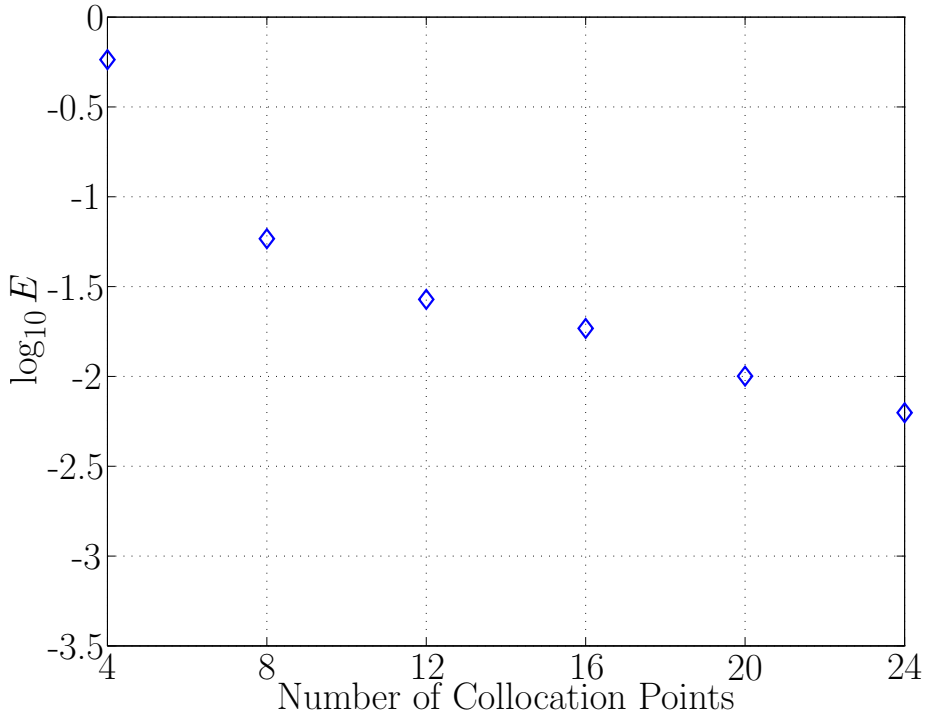
Traditional Collocation Methods: h and p Methods

Method	Order	For High Accuracy	NLP	Efficiency
h	Fixed	Extremely Fine Mesh	Huge NLP	Extremely Inefficient
p	Variable	Unreasonably Large Degree	Large Dense NLP	Intractable

Example: slow convergence of h and p methods

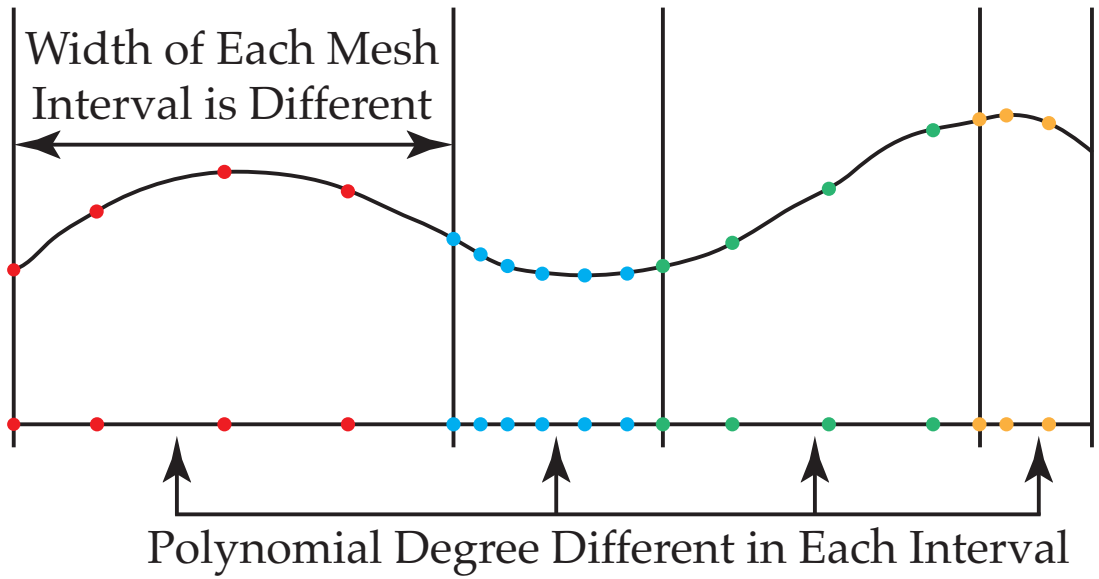
$$\frac{dy}{d\tau} = \begin{cases} 0, & -1 \leq \tau < -0.5 \\ \pi\cos(\pi\tau), & -0.5 \leq \tau \leq +0.5 \\ 0, & +0.5 < \tau \leq +1 \end{cases} \quad y(-1) = y_0, \quad y(\tau) = \begin{cases} y_0, \\ y_0 + 1 + \sin(\pi\tau) \\ y_{20} + 2 \end{cases}$$





hp-Adaptive Methods: New Approach to Collocation

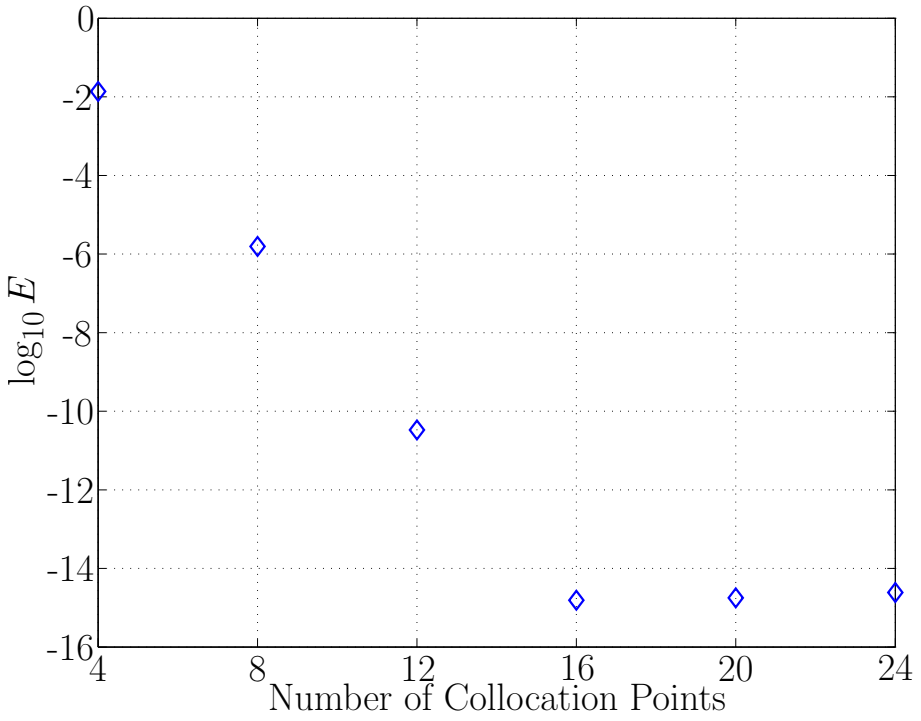
- Gain fast convergence in regions where solution is smooth
- Refine solution in nonsmooth or rapidly changing segments
- Benefits
  - Overcomes intractability of a p method
  - Significantly smaller NLP when compared with an h method
  - Computationally efficient because NLP remains sparse



hp-Adaptive Methods: New Approach to Collocation

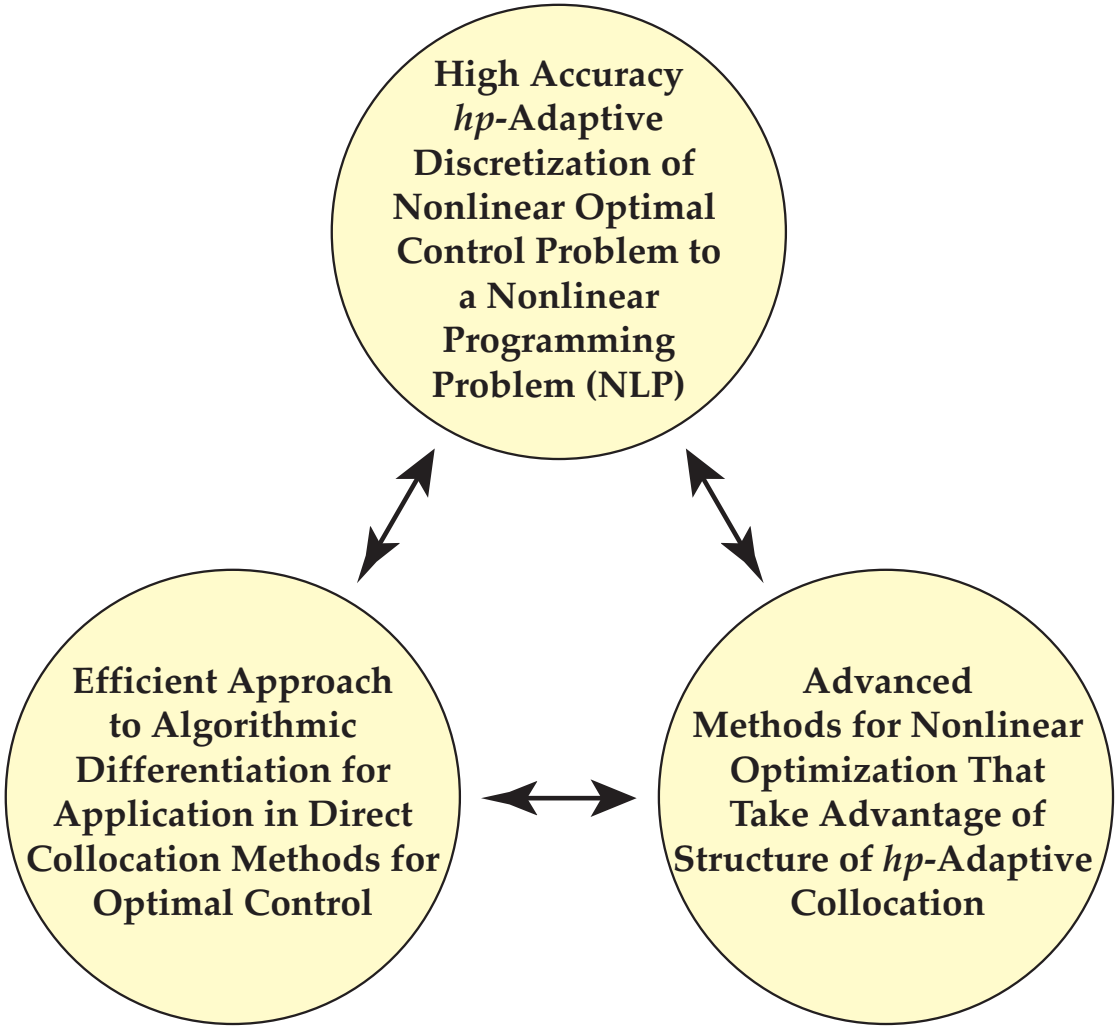
Behavior	Achieving High Accuracy	Convergence Rate	Mesh
Smooth	Increase Polynomial Degree	Exponential	Small
Nonsmooth/Fast	Refine Mesh Selectively	Piecewise Exponential	Small

Convergence rate of example ODE using *hp* method

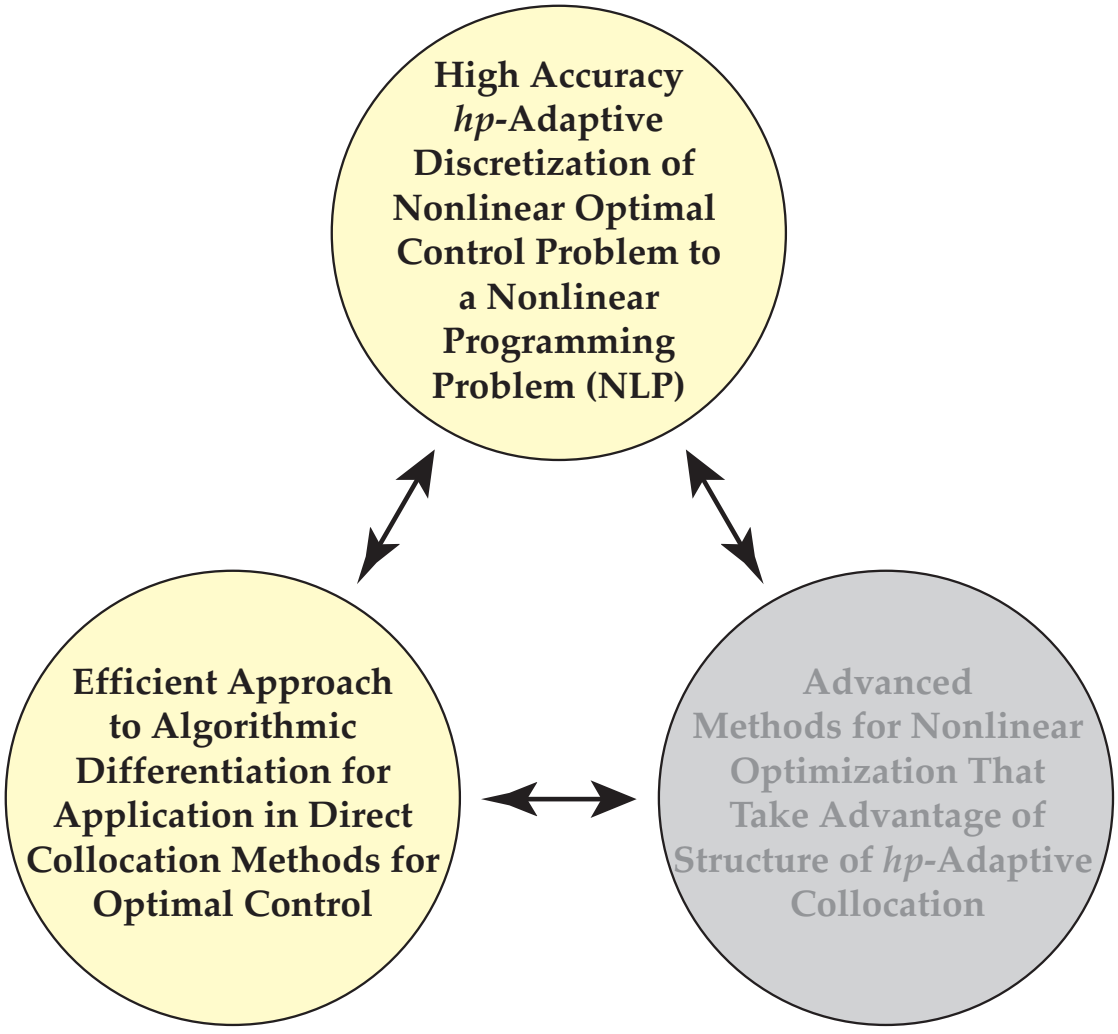


This talk: *novel computational framework using hp-adaptive methods*

Components of Optimal Control Framework



Components of Optimal Control Framework



- Benchmark aircraft optimal control problem used as demonstration

Quadrature Approximation and Interpolation

- Optimal control: need to integrate and interpolate functions
- Quadrature: approximation of an integral
  - Function sampled at support points $(\tau_1, \ldots, \tau_N)$
  - Use a weighted sum of these points (quadrature weights)

$$\int_{-1}^{+1} f(\tau) d\tau = \sum_{i=1}^{N} w_i f(\tau_i)$$

- Construct interpolant given discrete points $(\tau_1, \ldots, \tau_N)$

$$f(\tau) \approx F(\tau) = \sum_{i=1}^{N} c_i \psi_i(\tau), \quad \psi_i(\tau) = \text{trial or basis functions}$$

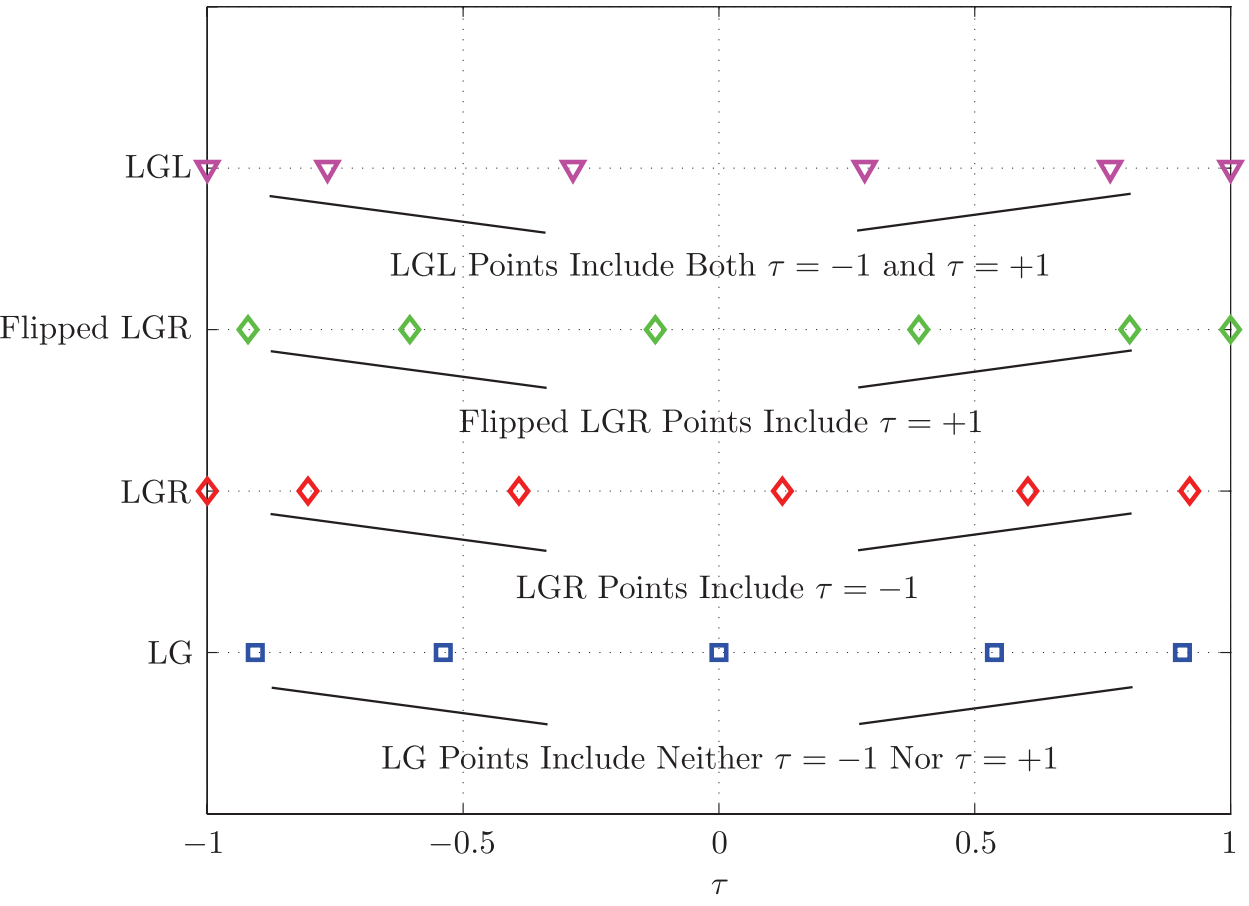
- This framework: integration and interpolation performed at *Gaussian quadrature points*

Basis of hp-Adaptive Methods: Gaussian Quadrature

- Want to *minimize* quadrature error $E(p) = \int_{-1}^{+1} f(\tau)d\tau - \sum_{k=1}^{N} w_k f(\tau_k)$
- Three types of Gaussian quadrature: LG, LGR, and LGL

Basis of *hp*-Adaptive Methods: Gaussian Quadrature

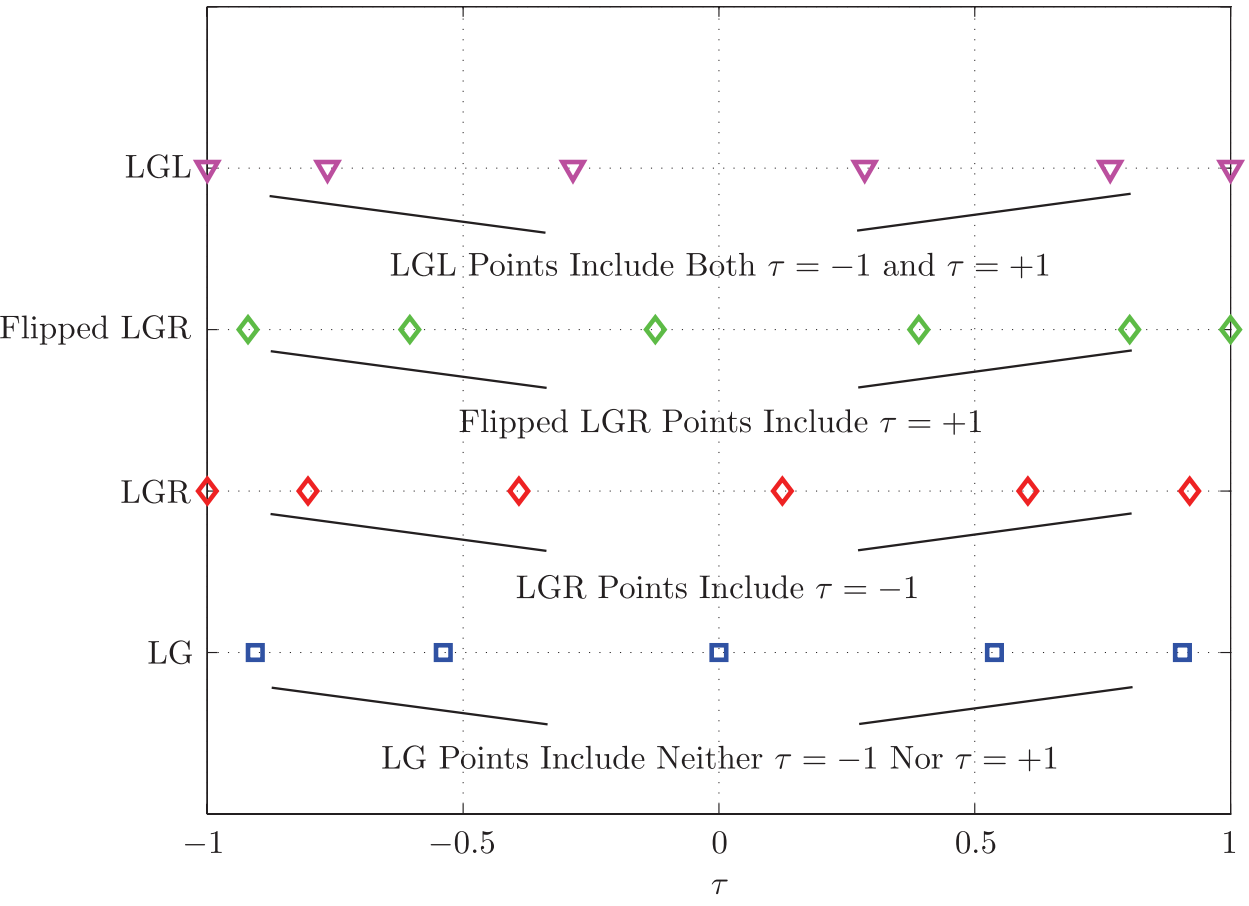
- Want to *minimize* quadrature error $E(p) = \int_{-1}^{+1} f(\tau)d\tau - \sum_{k=1}^{N} w_k f(\tau_k)$
- Three types of Gaussian quadrature: LG, LGR, and LGL



- LG: Roots of $P_N(\tau)$
- LGR: Roots of $P_N(\tau) + P_{N-1}(\tau)$
- LGL: Roots of $\dot{P}_{N-1}(\tau) \oplus -1$ and $+1$
- $P_N(\tau)$ = N^{th} Degree Legendre Polynomial

Basis of *hp*-Adaptive Methods: Gaussian Quadrature

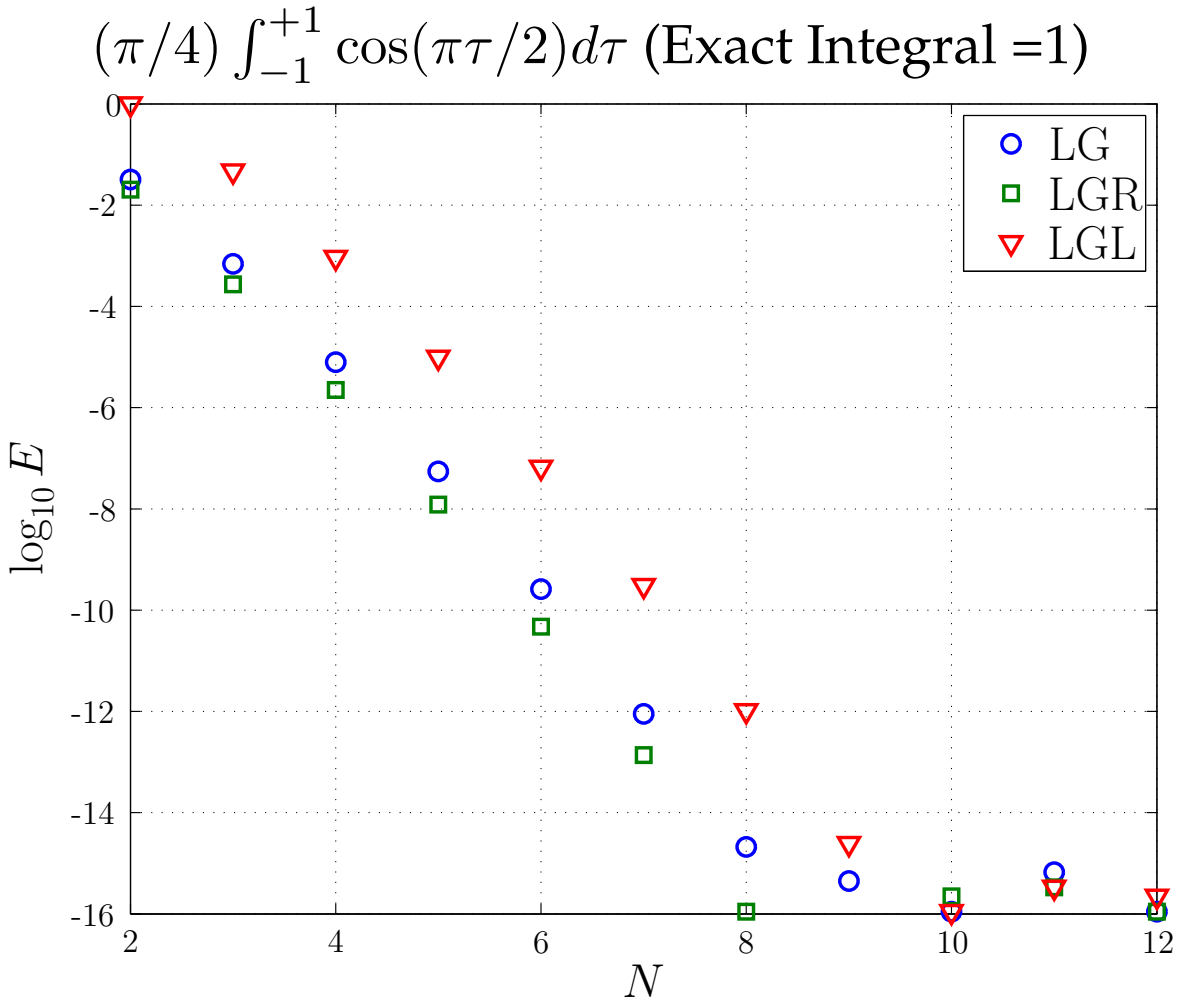
- Want to *minimize* quadrature error $E(p) = \int_{-1}^{+1} f(\tau)d\tau - \sum_{k=1}^{N} w_k f(\tau_k)$
- Three types of Gaussian quadrature: LG, LGR, and LGL



- LG: Roots of $P_N(\tau)$
- LGR: Roots of $P_N(\tau) + P_{N-1}(\tau)$
- LGL: Roots of $\dot{P}_{N-1}(\tau) \oplus -1$ and $+1$
- $P_N(\tau)$ = N^{th} Degree Legendre Polynomial

- Question: why use Gaussian quadrature points for collocation?
- Answer: (1) exponential convergence; (2) avoids Runge phenomenon

Exponential Convergence of Gauss Quadrature



Lagrange Interpolation Using Gauss Quadrature Points

- Approximate $f(\tau)$ by $F(\tau)$ using Lagrange interpolation:

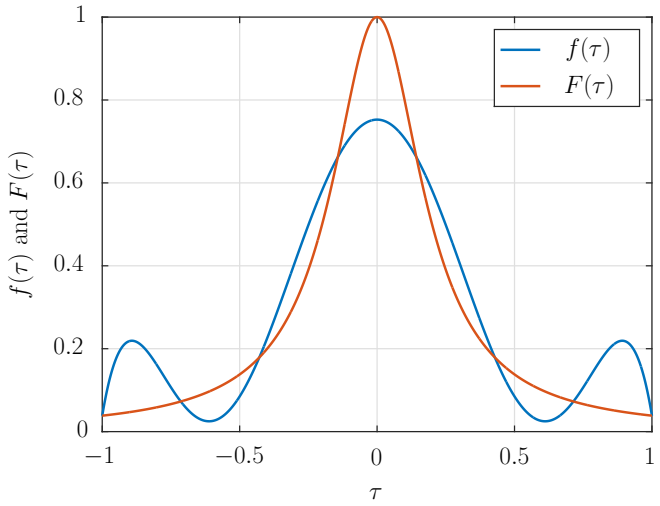
$$f(\tau) \approx F(\tau) = \sum_{i=1}^{N} c_i L_i(\tau), \; L_i(\tau) = \prod_{\substack{k=1 \\ k \neq i}}^{N} \frac{\tau - \tau_k}{\tau_i - \tau_k},$$

$$L_i(\tau_k) = \begin{cases} 1, & i = k \\ 0, & i \neq k \end{cases} \rightarrow \begin{array}{l} \text{isolation} \\ \text{property} \end{array}$$

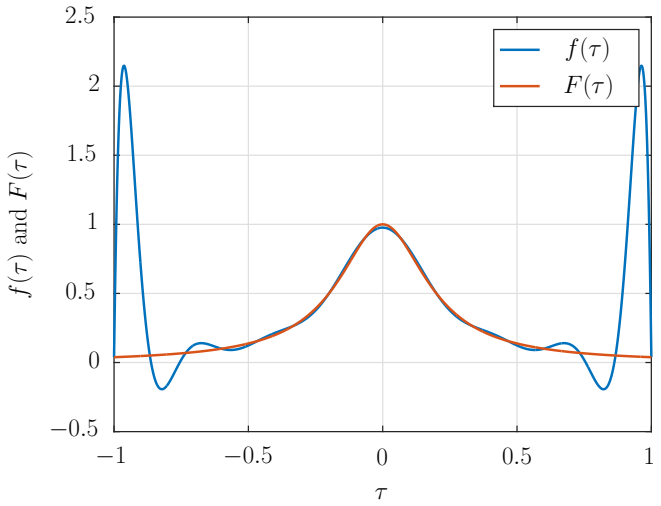
  where $\tau_k \in [-1, +1], \; (k = 1, \ldots, N)$ are the *support points*
- Note: isolation property $\rightarrow c_i = F(\tau_i)$
- Could use any support points (e.g., uniformly-spaced)
- Gauss quadrature support points highly advantageous
  - Quadrature converges exponentially (already discussed)
  - Runge phenomenon eliminated
- Demonstration on function $f(\tau) = 1/(1 + 25\tau^2)$

Lagrange Interpolation of $f(\tau) = 1/(1 + 25\tau^2)$

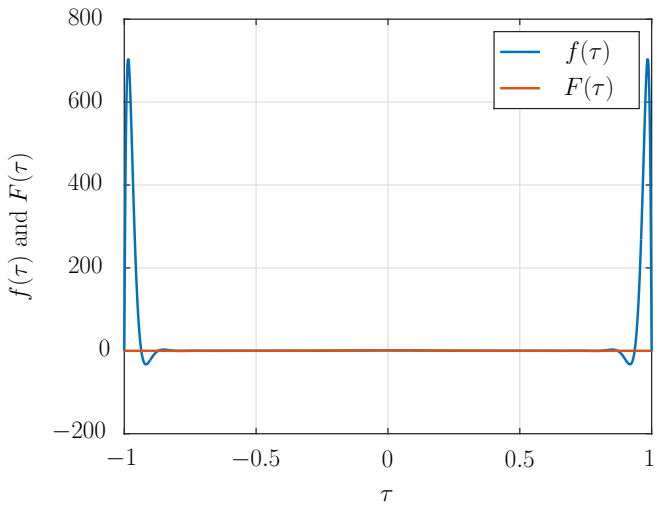
8 Uniformly-Spaced Points



16 Uniformly-Spaced Points

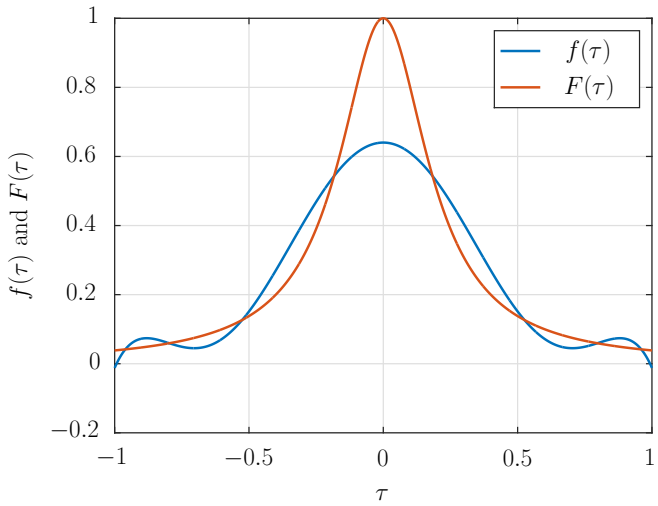


32 Uniformly-Spaced Points

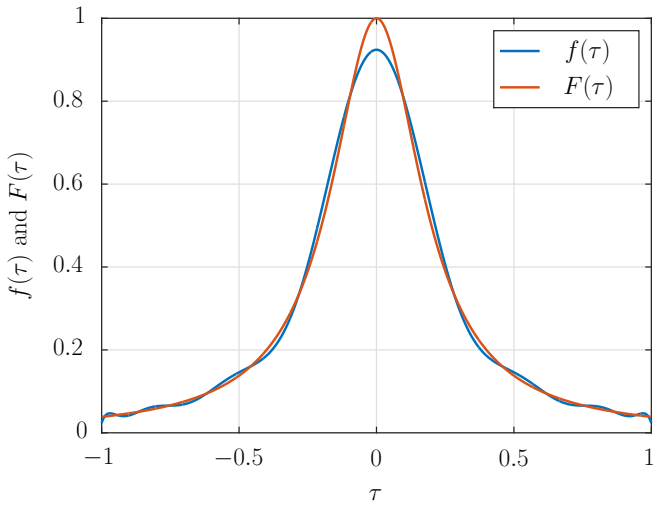


Lagrange Interpolation of $f(\tau) = 1/(1 + 25\tau^2)$

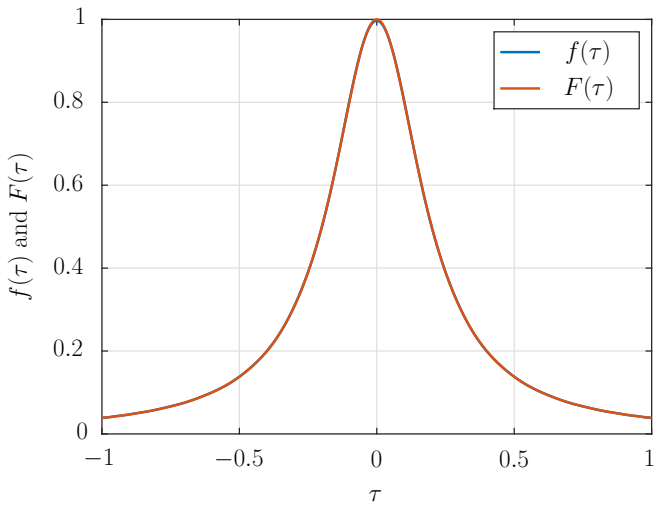
8 LG Points



16 LG Points

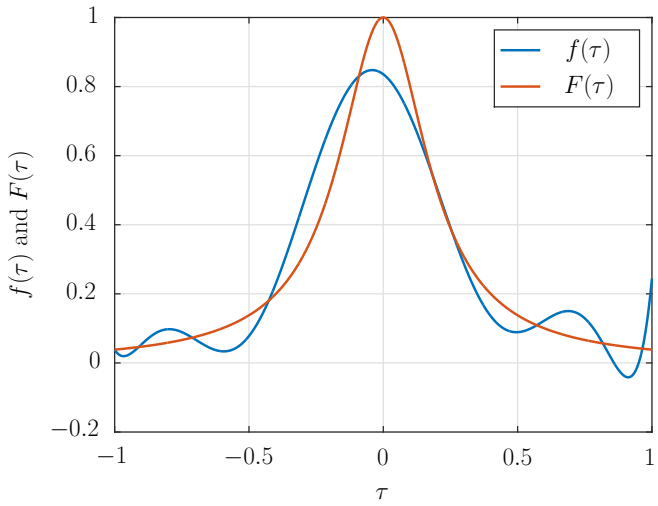


32 LG Points

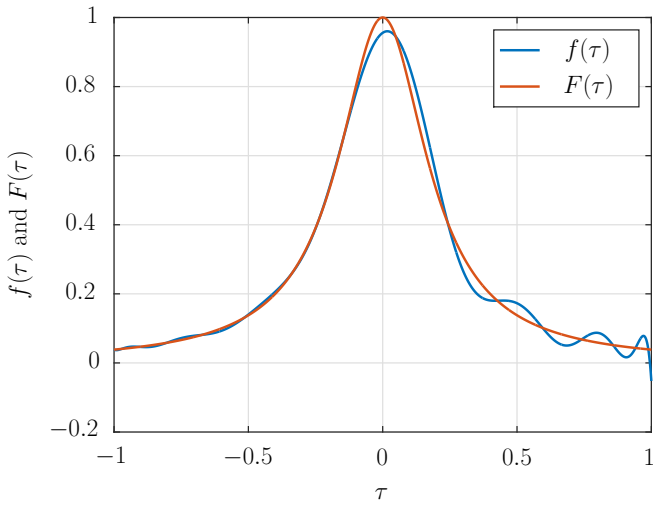


Lagrange Interpolation of $f(\tau) = 1/(1 + 25\tau^2)$

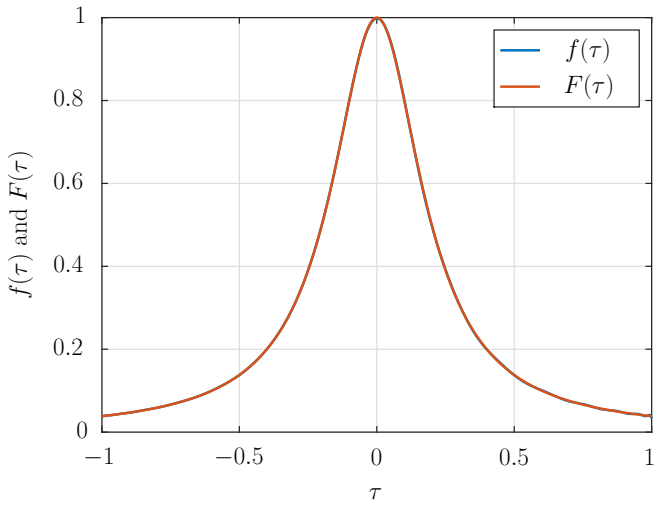
8 LGR Points



16 LGR Points



32 LGR Points



Choice of Gauss Quadrature in Collocation

- LGL[*,†]: not an integration method $\leftrightarrow$ non-convergent
- LG & LGR:[*,‡,§] converge exponentially; LGR easier to implement

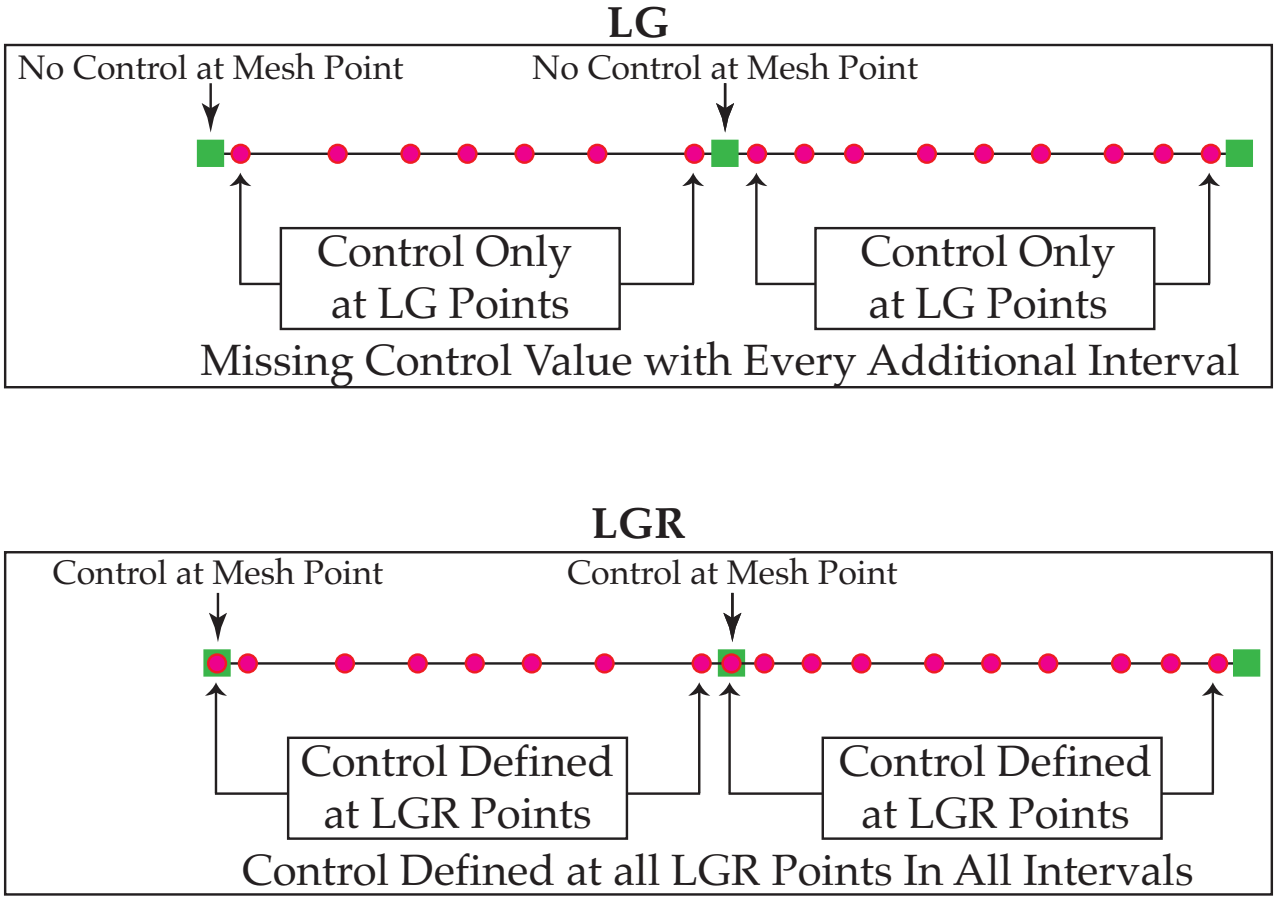
[*] Benson, D. A., Huntington, G. T., Thorvaldsen, T. P., and **Rao, A. V.**, "Direct Trajectory Optimization and Costate Estimation via an Orthogonal Collocation Method, *Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 6, November–December, 2006, pp. 1435–1440.

[†] Garg, D., Patterson, M. A., Hager, W. W., **Rao, A. V.**, Benson, D. A., and Huntington, G. T., "A Unified Framework for the Numerical Solution of Optimal Control Problems Using Pseudospectral Methods," *Automatica*, Vol. 46, No. 11, November 2010, pp. 1843-1851.

[‡] Hager, W. W., Hou, H., and **Rao, A. V.**, "Convergence Rate for a Gauss Collocation Method Applied to Unconstrained Optimal Control," *Journal of Optimization Theory and Applications*, Vol, 169, No. 3, June 2016, pp. 801 - 824.

[§] Hager, W. W., Hou, H., and **Rao, A. V.**, "Convergence Rate for a Gauss Collocation Method Applied to Unconstrained Optimal Control," *SIAM Journal on Control and Optimization*, Vol. 56, No. 2, March–April 2018, pp. 1386–1411.

Choice of Gauss Quadrature in Collocation



*Benson, D. A., Huntington, G. T., Thorvaldsen, T. P., and **Rao, A. V.**, "Direct Trajectory Optimization and Costate Estimation via an Orthogonal Collocation Method, *Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 6, November–December, 2006, pp. 1435–1440.

†Garg, D., Patterson, M. A., Hager, W. W., **Rao, A. V.**, Benson, D. A., and Huntington, G. T., "A Unified Framework for the Numerical Solution of Optimal Control Problems Using Pseudospectral Methods," *Automatica*, Vol. 46, No. 11, November 2010, pp. 1843-1851.

‡Hager, W. W., Hou, H., and **Rao, A. V.**, "Convergence Rate for a Gauss Collocation Method Applied to Unconstrained Optimal Control," *Journal of Optimization Theory and Applications*, Vol, 169, No. 3, June 2016, pp. 801 - 824.

§Hager, W. W., Hou, H., and **Rao, A. V.**, "Convergence Rate for a Gauss Collocation Method Applied to Unconstrained Optimal Control," *SIAM Journal on Control and Optimization*, Vol. 56, No. 2, March–April 2018, pp. 1386–1411.

hp-Adaptive Methods Using LGR Points

- State approximation in mesh interval $k \in [1, \ldots, K]$:

$$\mathbf{y}^{(k)}(s) \approx \mathbf{Y}^{(k)}(s) = \sum_{j=1}^{N_k+1} \mathbf{Y}_j^{(k)} \ell_j^{(k)}(s), \quad \frac{d\mathbf{Y}^{(k)}(s)}{ds} = \sum_{j=1}^{N_k+1} \mathbf{Y}_j^{(k)} \frac{d\ell_j^{(k)}(s)}{ds}$$

$$\ell_j^{(k)}(s) = \prod_{\substack{l=1 \\ l \neq j}}^{N_k+1} \frac{s - s_l^{(k)}}{s_j^{(k)} - s_l^{(k)}}, \qquad \left.\begin{array}{rcl} (s_1^{(k)}, \ldots, s_{N_k}^{(k)}) & = & \text{LGR points,} \\ N_k & = & \text{polynomial degree} \end{array}\right\} k \in [1, \ldots, K]$$

- Discrete approximation of cost functional ($t = \frac{1}{2}\left[(t_f - t_0)s + t_f + t_0\right]$)

$$\mathcal{J} \approx \mathcal{M}(\mathbf{Y}_1^{(1)}, t_0, \mathbf{Y}_{N_K+1}^{(K)}, t_K) + \frac{t_f - t_0}{2} \sum_{k=1}^{K} \sum_{j=1}^{N_k} w_j^{(k)} \mathcal{L}(\mathbf{Y}_j^{(k)}, \mathbf{U}_j^{(k)}, t(s_j^{(k)}, t_0, t_f)),$$

$$\begin{array}{rcl} w_j^{(k)} & = & \text{LGR quadrature weights, } (j = 1, \ldots, N_k), \\ \mathbf{U}_j^{(k)} & = & \text{control approximation, } (j = 1, \ldots, N_k), \\ \mathbf{Y}_j^{(k)} & = & \text{state approximation, } (j = 1, \ldots, N_k + 1) \end{array}$$

hp-Adaptive Methods Using LGR Points

- Collocation constraints (discretization of dynamics)

$$\sum_{j=1}^{N_k+1} D_{ij}^{(k)} \mathbf{Y}_j^{(k)} - \frac{t_f - t_0}{2} \mathbf{a}(\mathbf{Y}_i^{(k)}, \mathbf{U}_i^{(k)}, t(s_i^{(k)}, t_0, t_f)) = \mathbf{0}, \quad (i = 1, \ldots, N_k),$$

$$D_{ij}^{(k)} = \left[\frac{d\ell_j^{(k)}(s)}{ds} \right]_{s_i^{(k)}} = \begin{array}{l} N_k \times (N_k + 1) \text{ LGR} \\ \text{Differentiation Matrix} \end{array}, \begin{cases} i &= 1, \ldots, N_k, \\ j &= 1, \ldots, N_k + 1, \\ k &= 1, \ldots, K \end{cases}$$

- Path constraints

$$\mathbf{c}_{\min} \leq \mathbf{c}(\mathbf{Y}_i^{(k)}, \mathbf{U}_i^{(k)}, t_i^{(k)}) \leq \mathbf{c}_{\max}, \quad (i = 1, \ldots, N_k).$$

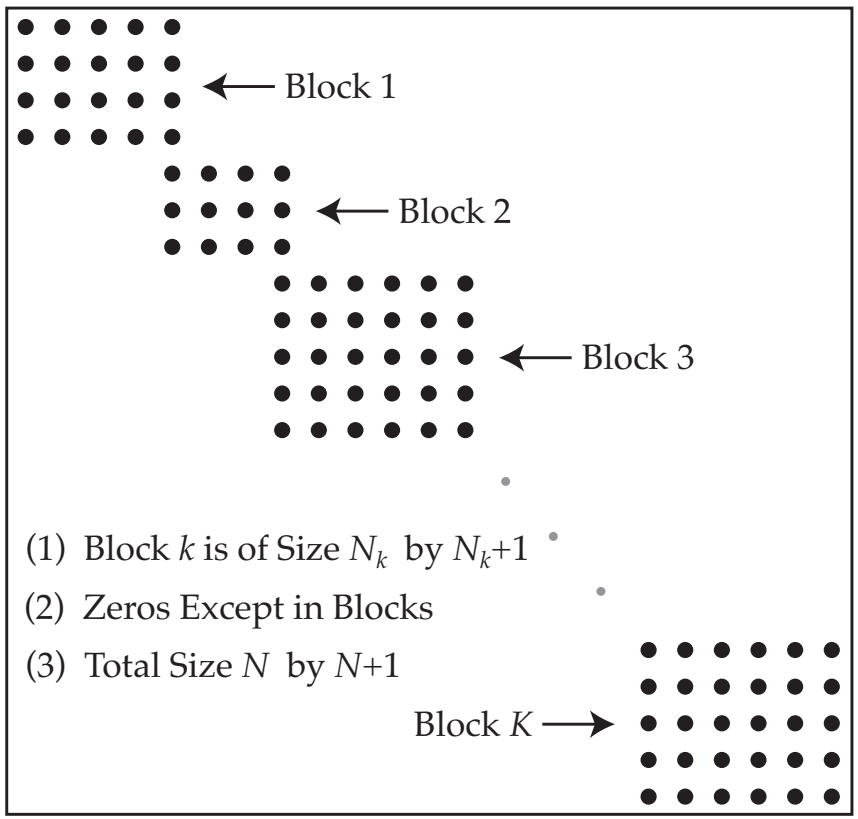
- Boundary conditions

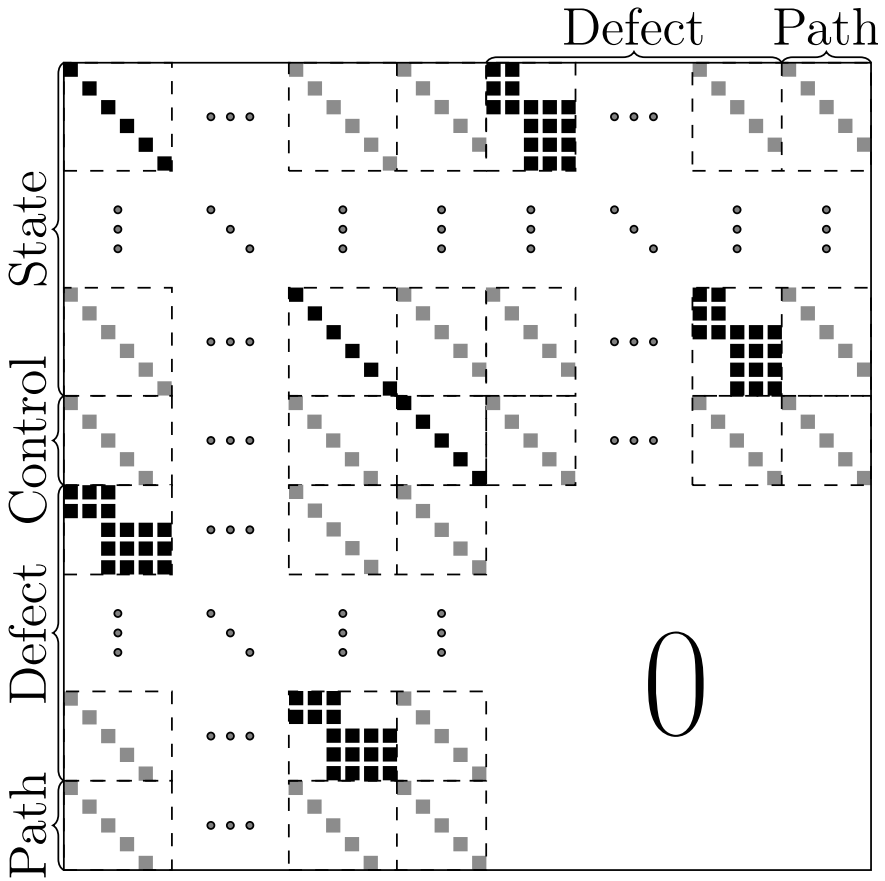
$$\mathbf{b}(\mathbf{Y}_1^{(1)}, t_0, \mathbf{Y}_{N_K+1}^{(K)}, t_f) = \mathbf{0}.$$

- Continuity at mesh points

$$\mathbf{Y}_{N_k+1}^{(k)} = \mathbf{Y}_1^{(k+1)}, \quad (k = 1, \ldots, K-1),$$

Sparse Structure of hp LGR NLP





Novel hp-Adaptive Mesh Refinement Methods

- Mesh that meets specified accuracy must be determined iteratively
- Our research: novel hp-adaptive mesh refinement methods
- Earlier methods
  - Method 1:[*] remains an h method for as long as possible
  - Method 2:[†] remains a p method as long as possible
  - Method 3:[‡] estimates smoothness of solution in mesh interval
    - Smooth $\rightarrow$ polynomial degree increased (to a limit)
    - Nonsmooth or limit exceeded $\rightarrow$ increase number of mesh intervals
- **Latest Method:**[§] uses our proven LGR convergence rate
  - Solution smooth $\rightarrow$ fix width and solve for N_k
  - Solution nonsmooth $\rightarrow$ fix N_k and determine number of intervals
  - First method capable of *reducing* size of mesh
- Next: our latest hp method compared against h methods

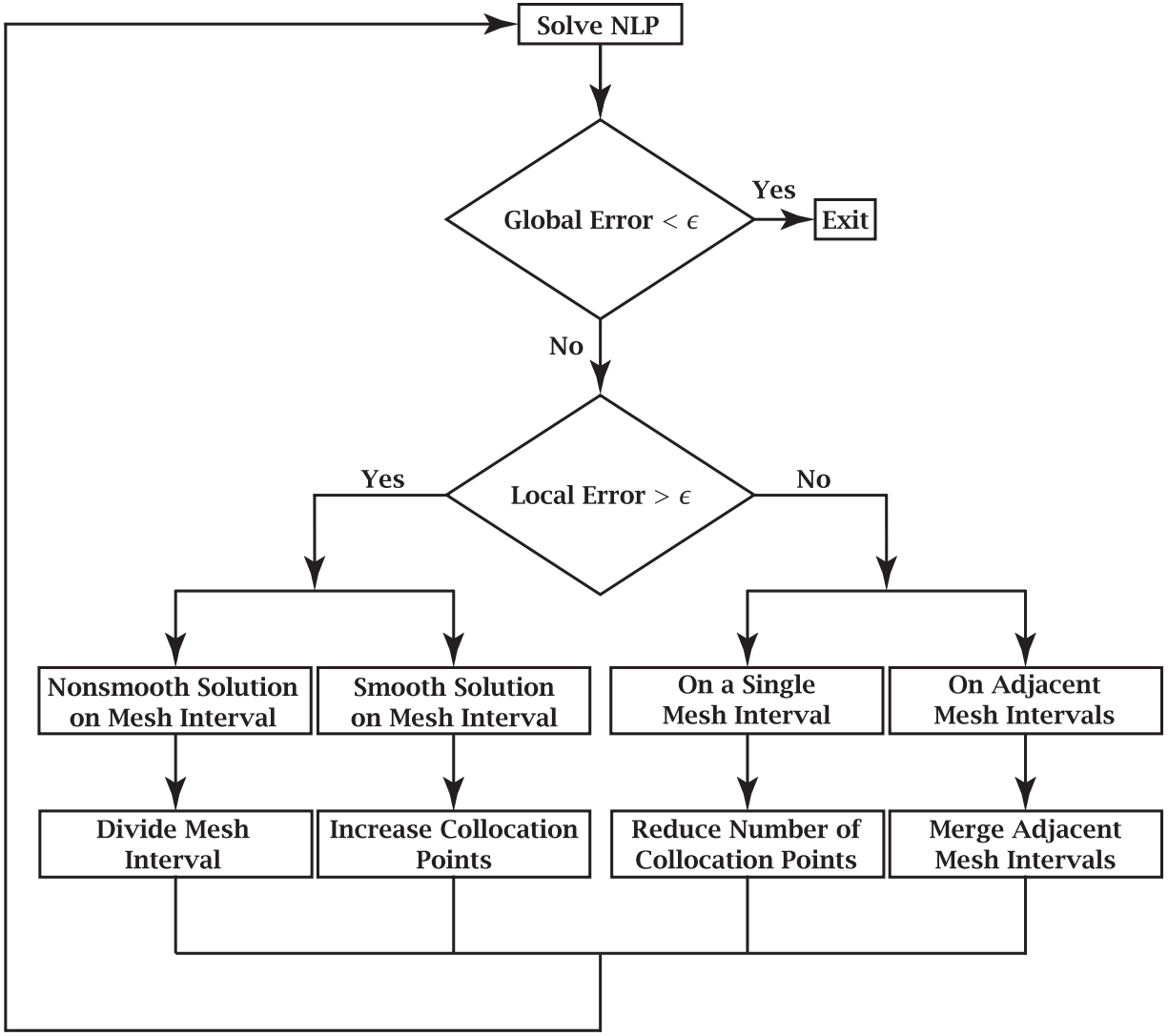
[*] Darby, C. L., Hager, W. W., and **Rao, A. V.**, "Direct Trajectory Optimization Using a Variable Low-Order Adaptive Pseudospectral Method," Journal of Spacecraft and Rockets, Vol. 48, No. 3, May-June 2011, pp. 433-445.

[†] Darby, C. L., Hager, W. W., and **Rao, A. V.**, "An hp-Adaptive Pseudospectral Method for Solving Optimal Control Problems," *Optimal Control Applications and Methods*, Vol. 32, No. 4, July-August 2011, pp. 476-502.

[‡] Patterson, M. A., Hager, W. W., and **Rao, A. V.**, "A ph Mesh Refinement Method for Optimal Control," *Optimal Control Applications and Methods*, Vol. 36, No. 4, July–August 2015, pp. 398–421.

[§] Liu, F. Hager, W. W., and **Rao, A. V.**, "Adaptive Mesh Refinement for Optimal Control Using Nonsmoothness Detection and Mesh Size Reduction," *Journal of the Franklin Institute*, Vol. 352, No. 10, October 2015, pp. 4081–4106.

hp-Adaptive Method with Mesh Size Reduction *



*Liu, F. Hager, W. W., and **Rao, A. V.**, "Adaptive Mesh Refinement for Optimal Control Using Nonsmoothness Detection and Mesh Size Reduction," *Journal of the Franklin Institute*, Vol. 352, No. 10, October 2015, pp. 4081–4106.

Benchmark Example: Minimum-Time Supersonic Aircraft Climb[*,†]

- Famous problem in performance optimization of supersonic aircraft[†]
- Table interpolation: 1-D (aerodynamic data); 2-D (Thrust)
- Solution non-intuitive
- Large NLP with increasing mesh size: 1000s by 1000s

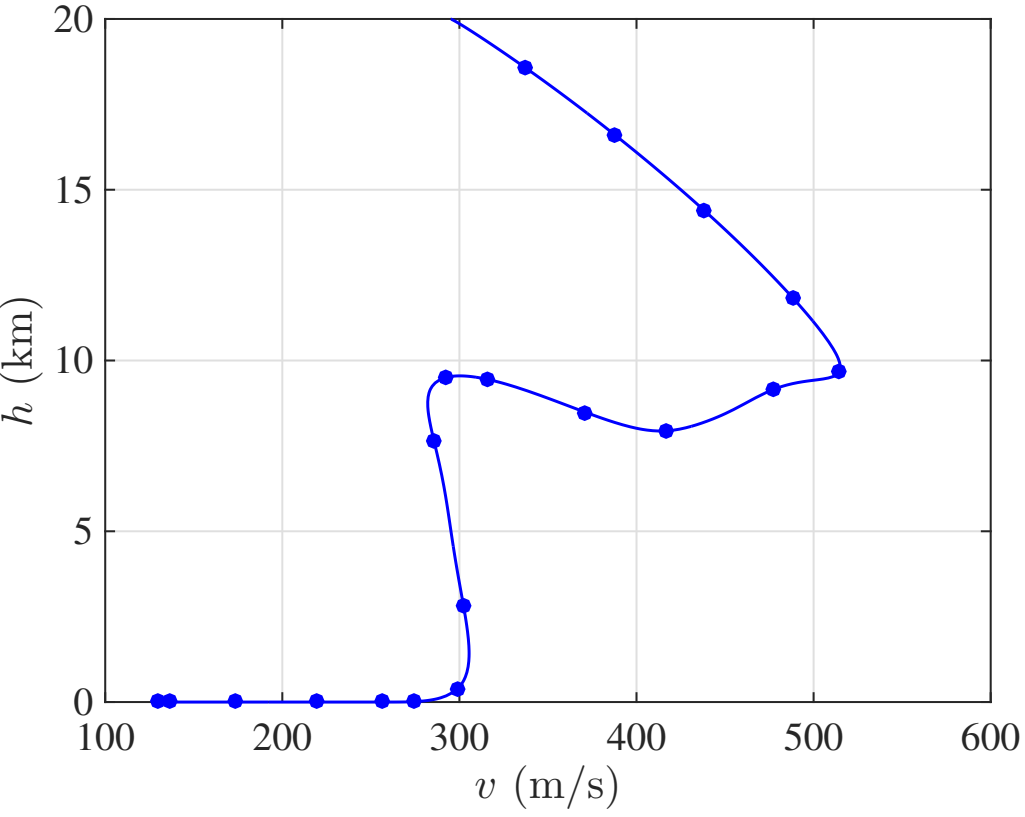
Minimize t_f subject to

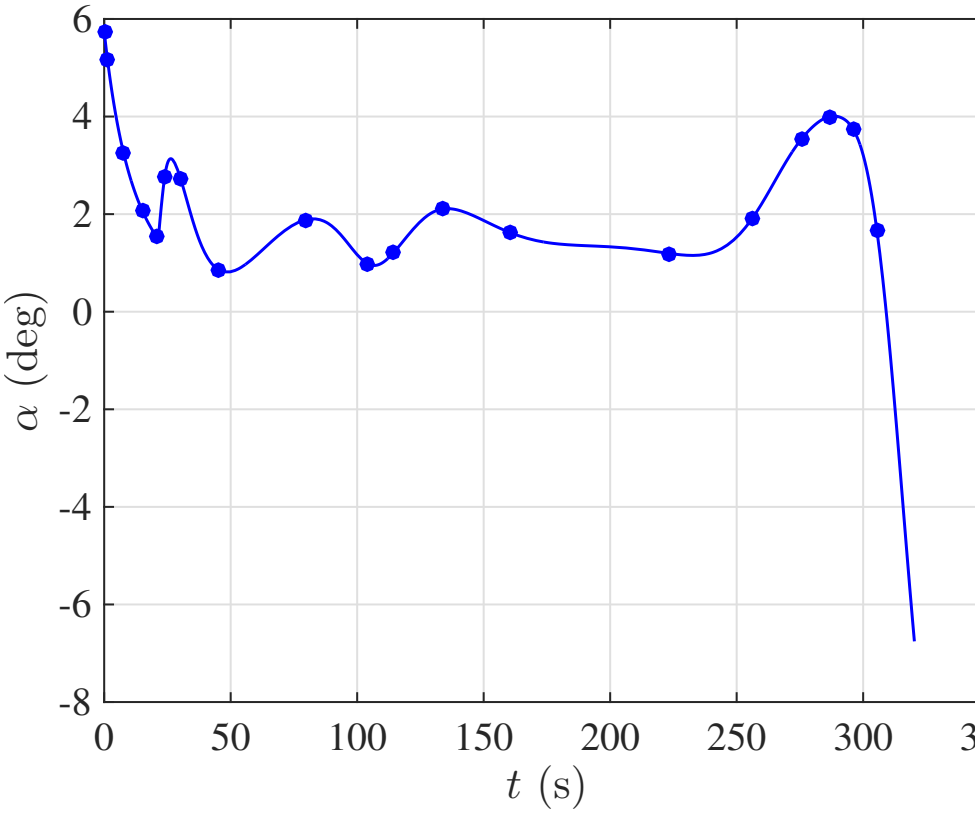
$$
\begin{aligned}
\dot{h} &= v\sin\gamma && , && (h(0), h(t_f)) &= (0, 20)\ \text{km} \\
\dot{v} &= \frac{T\cos\alpha - D}{m} - \frac{\mu}{r^2} sin\gamma, && , && (v(0), v(t_f)) &= (129, 295)\ \text{m}\cdot\text{s}^{-1}, \\
\dot{\gamma} &= \frac{T\sin\alpha + L}{m} + \left(\frac{v}{r} - \frac{\mu}{r^2}\right)\cos\gamma && , && (\gamma(0), \gamma(t_f)) &= (0, 0)\ \text{rad}, \\
\dot{m} &= -\frac{T}{V_e} && , && (m(0), m(t_f)) &= (19050, \text{Free})\ \text{kg}, \\
h &\geq 0. &&
\end{aligned}
$$

[*] Betts, J. T., *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, SIAM Press, Philadelphia, 2010

[†] Bryson, A. E., Desai, M. N., and Hoffman, W. C., "Energy-State Approximation in Performance Optimization of Supersonic Aircraft,", *Journal of Aircraft*, Vol. 6, No. 6, November-December 1969, pp. 481–488.

Benchmark Example: Minimum-Time Supersonic Aircraft Climb*,†





*Betts, J. T., *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, SIAM Press, Philadelphia, 2010

†Bryson, A. E., Desai, M. N., and Hoffman, W. C., "Energy-State Approximation in Performance Optimization of Supersonic Aircraft,", *Journal of Aircraft*, Vol. 6, No. 6, November-December 1969, pp. 481–488.

Benchmark Example: Minimum-Time Supersonic Aircraft Climb[*,†]

- Indirect methods
  - Deriving calculus of essentially impossible due to tabular data
  - Need to make drastic simplifications to model
  - Even after simplifications, tremendous hand-holding required
- Direct methods
  - Shooting/multiple-shooting: low-accuracy & inefficient or or failure
  - Collocation: accuracy/efficiency dependent upon discretization
- This study: demonstrate effectiveness of hp-adaptive direct collocation

[*] Betts, J. T., *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, SIAM Press, Philadelphia, 2010

[†] Bryson, A. E., Desai, M. N., and Hoffman, W. C., "Energy-State Approximation in Performance Optimization of Supersonic Aircraft,", *Journal of Aircraft*, Vol. 6, No. 6, November-December 1969, pp. 481–488.

Analysis of p, h, and hp Methods on Aircraft Climb Example

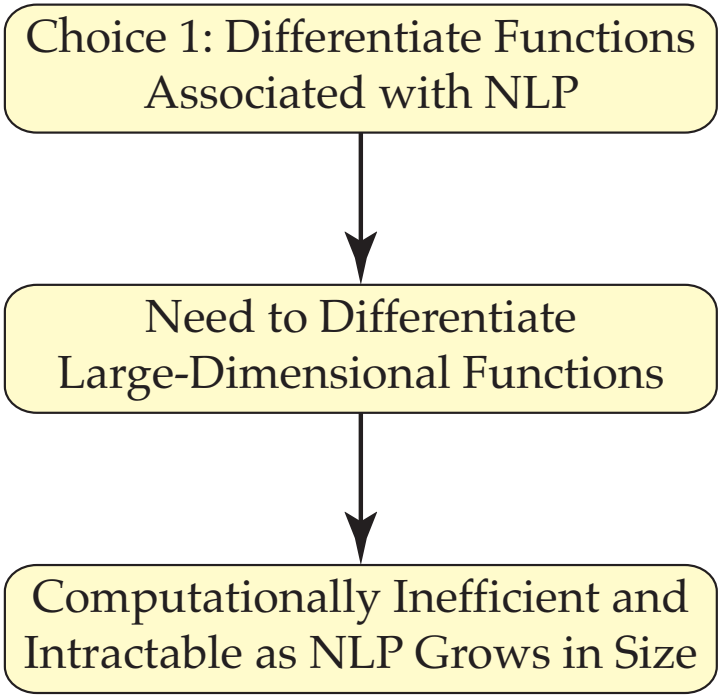
- Employ p, h, & hp methods with full Newton NLP solver IPOPT
- Important: derivatives computed by exploiting sparse structure*

*Patterson, M. A. and **Rao, A. V.**, "Exploiting Sparsity in Direct Collocation Pseudospectral Methods for Solving Optimal Control Problems," *Journal of Spacecraft and Rockets*, Vol. 49, No. 2, March–April 2012, pp. 364–377.

†Weinstein, M. J. and **Rao, A. V.**, "A Source Transformation via Operator Overloading Method for the Automatic Differentiation of Mathematical Functions in MATLAB," *ACM Transactions on Mathematical Software*, Vol. 42, No. 2, Article 11, June 2016, pp. 11:1 - 11:44.

Analysis of p, h, and hp Methods on Aircraft Climb Example

- Employ p, h, & hp methods with full Newton NLP solver IPOPT
- Important: derivatives computed by exploiting sparse structure*

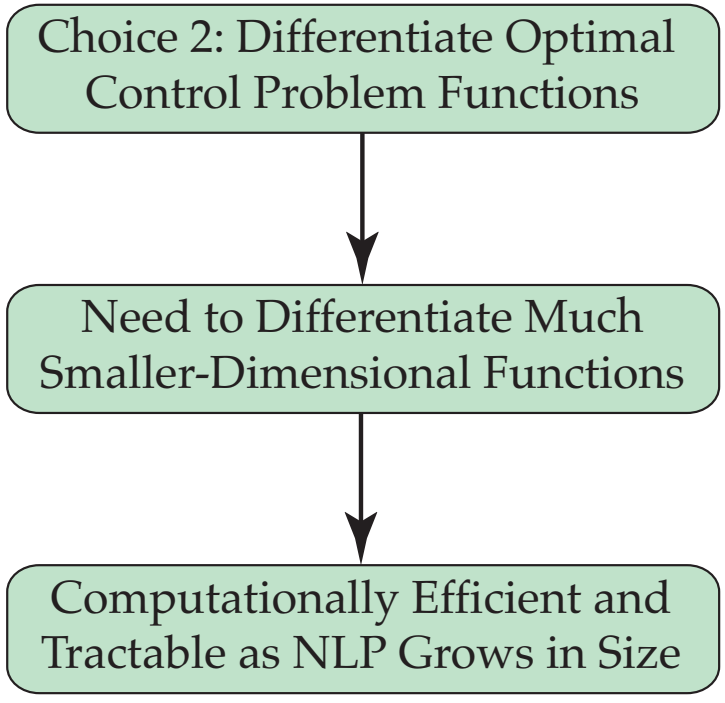


*Patterson, M. A. and **Rao, A. V.**, "Exploiting Sparsity in Direct Collocation Pseudospectral Methods for Solving Optimal Control Problems," *Journal of Spacecraft and Rockets*, Vol. 49, No. 2, March–April 2012, pp. 364–377.

†Weinstein, M. J. and **Rao, A. V.**, "A Source Transformation via Operator Overloading Method for the Automatic Differentiation of Mathematical Functions in MATLAB," *ACM Transactions on Mathematical Software*, Vol. 42, No. 2, Article 11, June 2016, pp. 11:1 - 11:44.

Analysis of p, h, and hp Methods on Aircraft Climb Example

- Employ p, h, & hp methods with full Newton NLP solver IPOPT
- Important: derivatives computed by exploiting sparse structure*

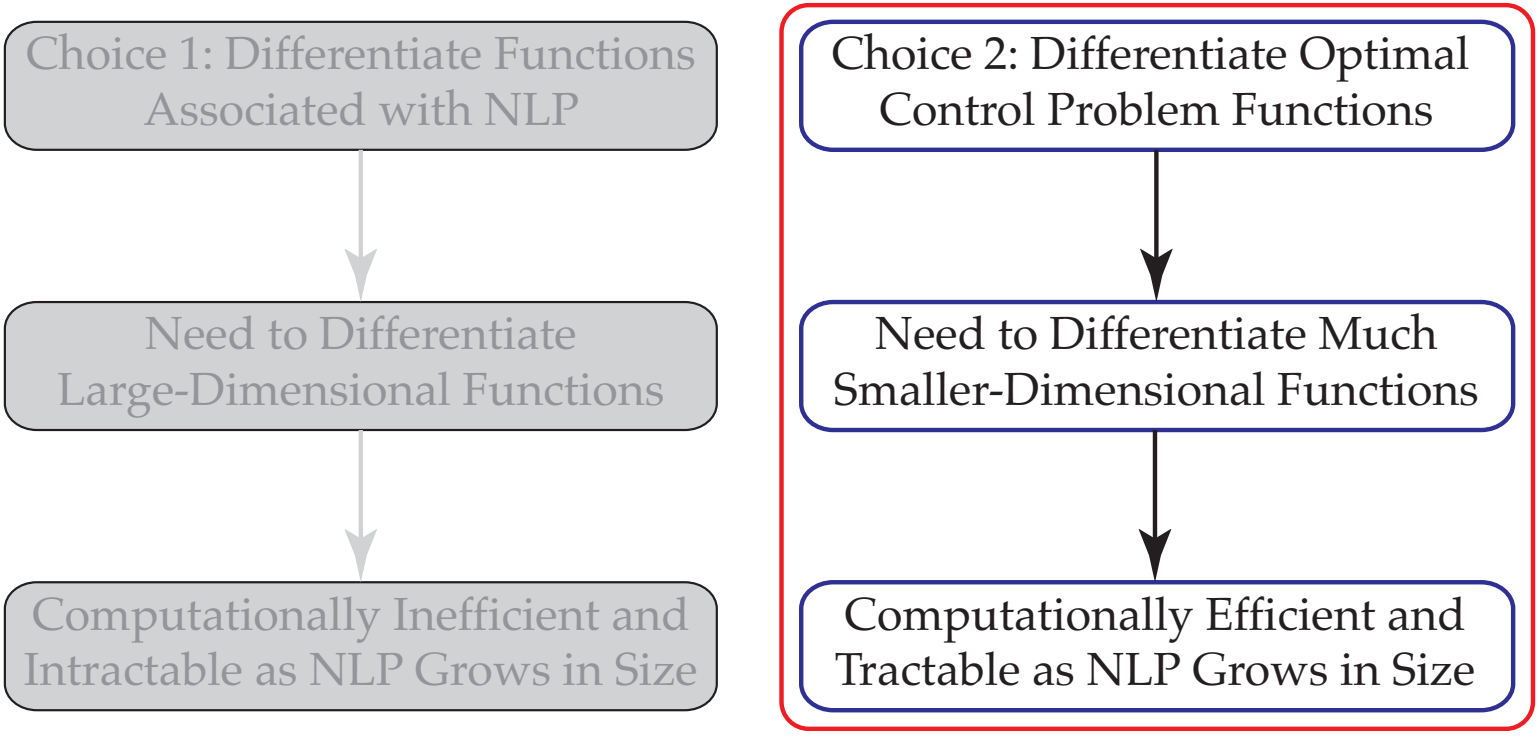


*Patterson, M. A. and **Rao, A. V.**, "Exploiting Sparsity in Direct Collocation Pseudospectral Methods for Solving Optimal Control Problems," *Journal of Spacecraft and Rockets*, Vol. 49, No. 2, March–April 2012, pp. 364–377.

†Weinstein, M. J. and **Rao, A. V.**, "A Source Transformation via Operator Overloading Method for the Automatic Differentiation of Mathematical Functions in MATLAB," *ACM Transactions on Mathematical Software*, Vol. 42, No. 2, Article 11, June 2016, pp. 11:1 - 11:44.

Analysis of p, h, and hp Methods on Aircraft Climb Example

- Employ p, h, & hp methods with full Newton NLP solver IPOPT
- Important: derivatives computed by exploiting sparse structure*

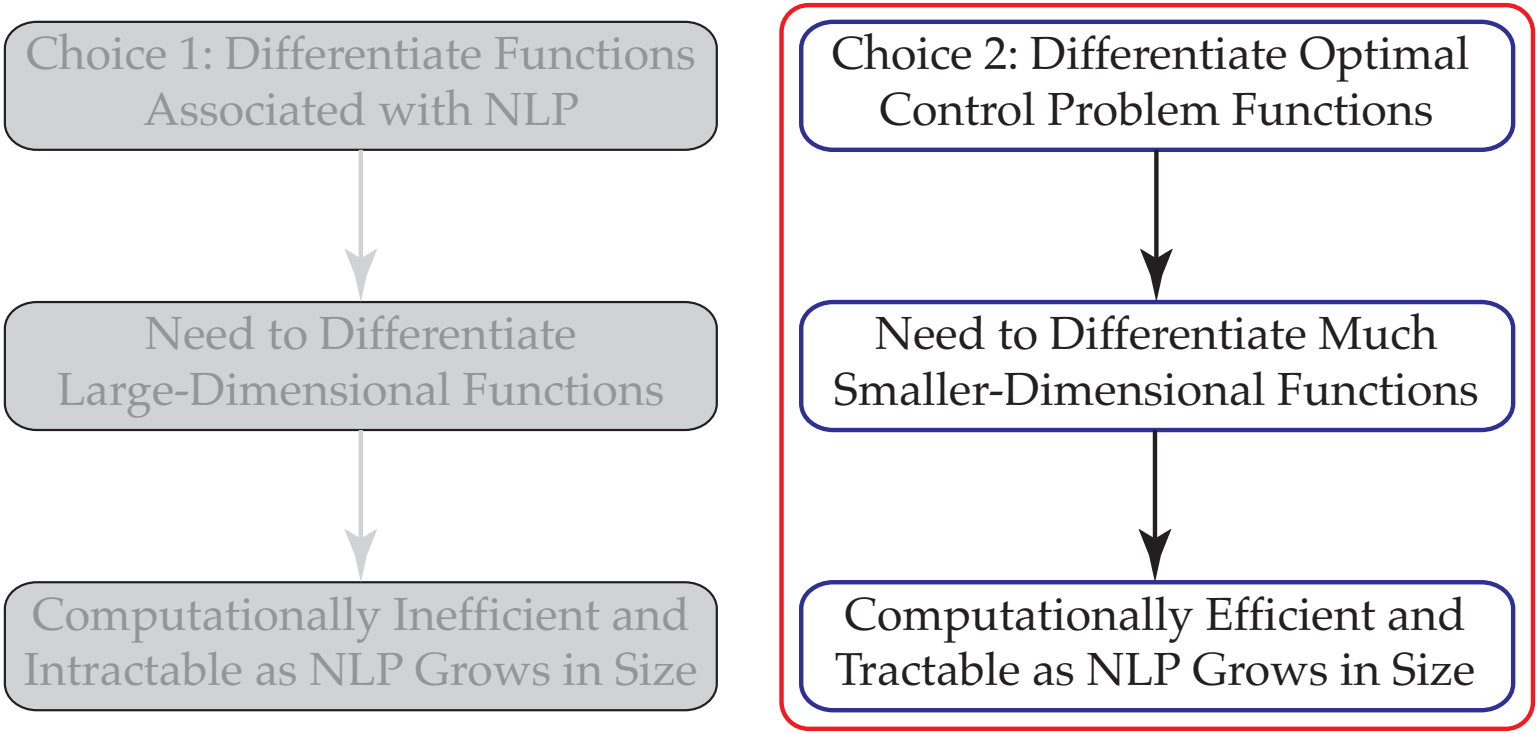


*Patterson, M. A. and **Rao, A. V.**, "Exploiting Sparsity in Direct Collocation Pseudospectral Methods for Solving Optimal Control Problems," *Journal of Spacecraft and Rockets*, Vol. 49, No. 2, March–April 2012, pp. 364–377.

†Weinstein, M. J. and **Rao, A. V.**, "A Source Transformation via Operator Overloading Method for the Automatic Differentiation of Mathematical Functions in MATLAB," *ACM Transactions on Mathematical Software*, Vol. 42, No. 2, Article 11, June 2016, pp. 11:1 - 11:44.

Analysis of p, h, and hp Methods on Aircraft Climb Example

- Employ p, h, & hp methods with full Newton NLP solver IPOPT
- Important: derivatives computed by exploiting sparse structure*



- Will employ sparse finite-differences and algorithmic differentiation†
- Goal: assess performance as demand for accuracy increases

*Patterson, M. A. and **Rao, A. V.**, "Exploiting Sparsity in Direct Collocation Pseudospectral Methods for Solving Optimal Control Problems," *Journal of Spacecraft and Rockets*, Vol. 49, No. 2, March–April 2012, pp. 364–377.

†Weinstein, M. J. and **Rao, A. V.**, "A Source Transformation via Operator Overloading Method for the Automatic Differentiation of Mathematical Functions in MATLAB," *ACM Transactions on Mathematical Software*, Vol. 42, No. 2, Article 11, June 2016, pp. 11:1 - 11:44.

Performance of p and h Methods: Sparse Finite-Differences

- Performance of p and h methods
  - ϵ =Relative error accuracy tolerance
  - N =mesh size
  - M =mesh iterations

p			
ϵ	CPU (s)	N	M
10^{-4}	48.78	72	40
10^{-5}	198.33	137	95
10^{-6}	Fail	Fail	Fail
10^{-7}	Fail	Fail	Fail
10^{-8}	Fail	Fail	Fail
10^{-9}	Fail	Fail	Fail
10^{-10}	Fail	Fail	Fail

h			
ϵ	CPU (s)	N	M
10^{-4}	5.66	96	3
10^{-5}	5.86	156	4
10^{-6}	7.11	264	5
10^{-7}	10.77	498	7
10^{-8}	17.33	921	10
10^{-9}	72.80	1614	34
10^{-10}	196.97	2964	61

Performance of p and h Methods: Sparse Finite-Differences

- Performance of p and h methods
  - ϵ =Relative error accuracy tolerance
  - N =mesh size
  - M =mesh iterations

p			
ϵ	CPU (s)	N	M
10^{-4}	48.78	72	40
10^{-5}	198.33	137	95
10^{-6}	Fail	Fail	Fail
10^{-7}	Fail	Fail	Fail
10^{-8}	Fail	Fail	Fail
10^{-9}	Fail	Fail	Fail
10^{-10}	Fail	Fail	Fail

h			
ϵ	CPU (s)	N	M
10^{-4}	5.66	96	3
10^{-5}	5.86	156	4
10^{-6}	7.11	264	5
10^{-7}	10.77	498	7
10^{-8}	17.33	921	10
10^{-9}	72.80	1614	34
10^{-10}	196.97	2964	61

- Observations
  - p method fails outright for anything but low accuracy
  - h method CPU time very large when high accuracy is required
  - h method mesh size & number of mesh iterations grow even faster

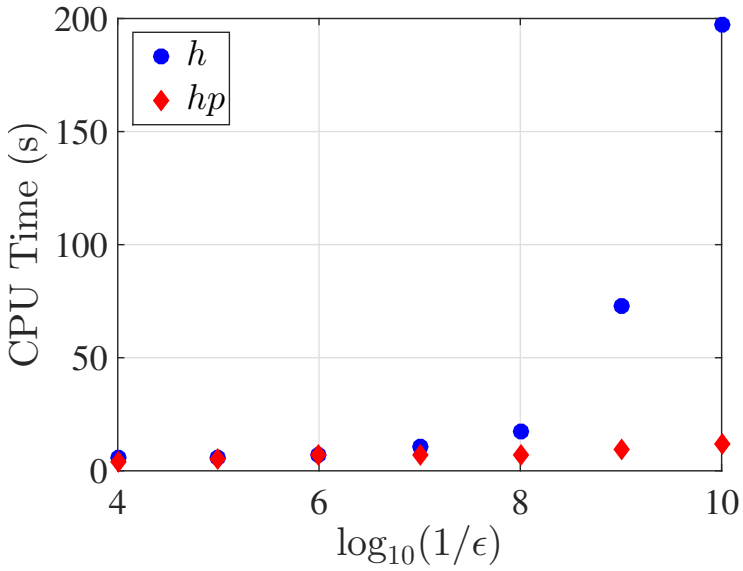
Performance of h and hp Methods: Sparse Finite Differences

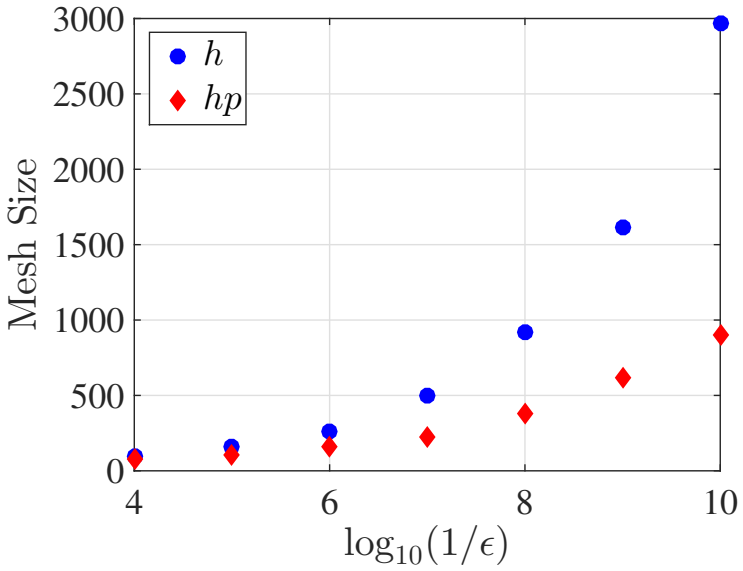
- Comparison of h and hp methods

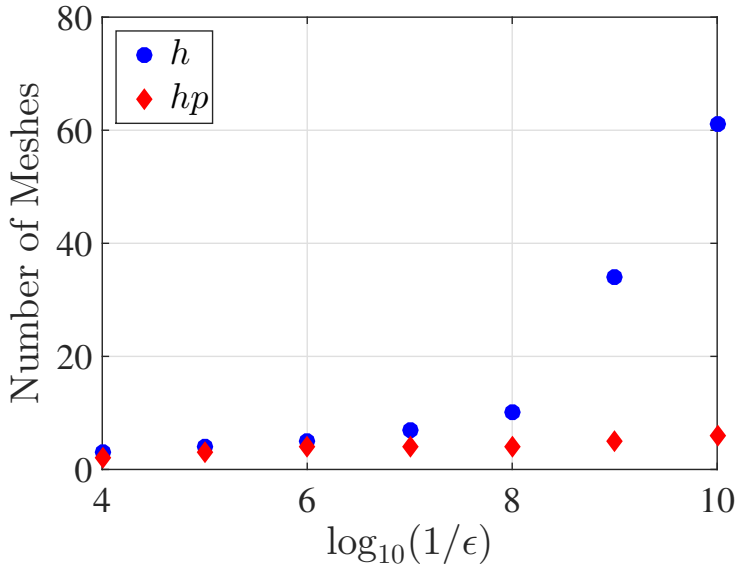
h				hp			
ϵ	CPU (s)	N	M	ϵ	CPU (s)	N	M
10^{-4}	5.66	96	3	10^{-4}	3.92	74	2
10^{-5}	5.86	156	4	10^{-5}	5.28	102	3
10^{-6}	7.11	264	5	10^{-6}	6.71	157	4
10^{-7}	10.77	498	7	10^{-7}	7.12	227	4
10^{-8}	17.33	921	10	10^{-8}	6.87	380	4
10^{-9}	72.80	1614	34	10^{-9}	9.49	620	5
10^{-10}	196.97	2964	61	10^{-10}	11.87	902	6

- Observation: hp method clearly outperforms h method
- As accuracy demand increases, gap in performance widens

Performance of h and hp Methods: Sparse Finite Differences







Increasing Efficiency: Algorithmic Differentiation

- Results so far generated using finite-difference approximations
- Direct collocation methods create *large* NLPs
  - Quasi-Newton NLP solvers: first derivative required
  - Newton NLP solvers: first and second derivatives
- *Large* portion of NLP computation time spent calculating derivatives
- Because of the importance of efficient and accurate derivatives
  - Developed new methods for algorithmic differentiation
  - Our application was the efficient solution of direct collocation NLPs
  - Note: our methods have broader impact to other applications

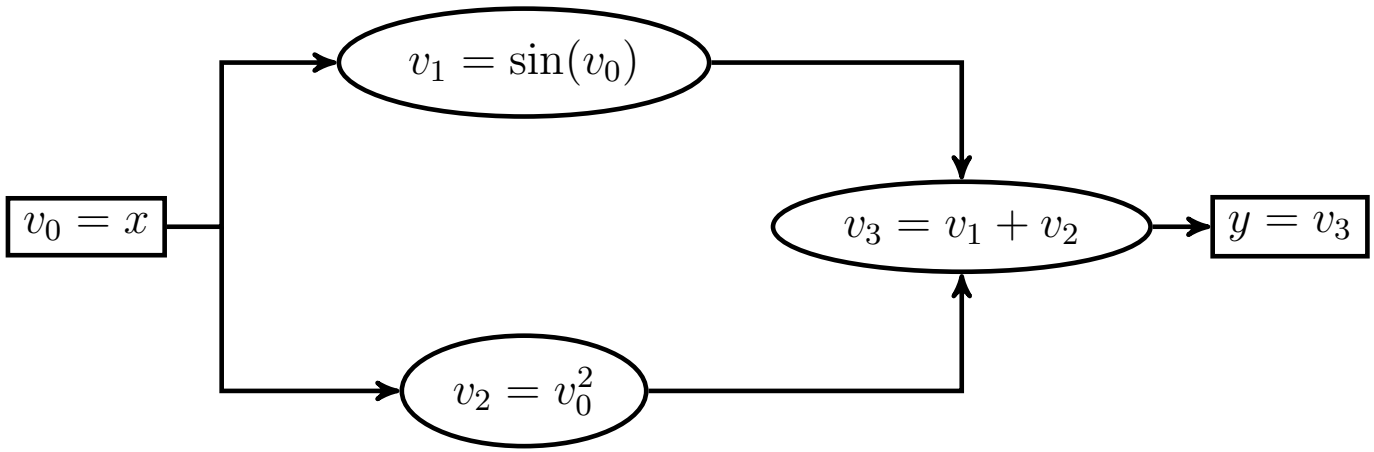
Methods for Computing Derivatives

- Hand Derivations - both "costly" and inherently error prone.
- Symbolic Differentiation (e.g. *Maple*, *Mathematica*, *MATLAB Symbolic Toolbox*) - suffer from expression explosion, cannot differentiate functions containing branches
- Finite Differences - accuracy dependent upon step-size, even if optimally chosen accuracy is "poor", particularly at higher order derivatives

Algorithmic Differentiation (AD)

- *Definition: AD is the process of determining accurate derivatives of a function defined by computer programs using the rules of differential calculus**

Ex: $y = \sin x + x^2$



- A program $y = P(x)$ may be broken into a sequence of operations $p_1(\cdot), p_2(\cdot), \ldots, p_k(\cdot) = y$. Given the rules of differentiation of each operation $p_i(\cdot)$, derivative of y with respect to x may be obtained by systematic application of the chain rule.

*Griewank, A. and Walther, A, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, Second Edition, SIAM Press, Philadelphia, 2008

Forward and Reverse Mode AD

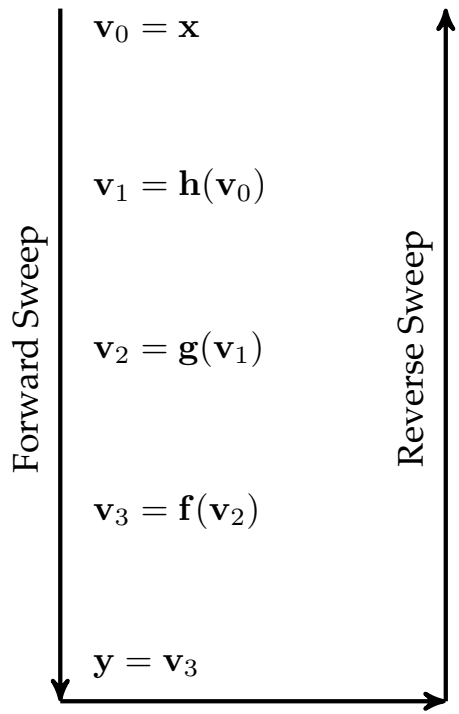
Ex: Given a program $\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{h}(\mathbf{x})))$

$$\nabla_{\mathbf{x}}\mathbf{y} = \mathbf{I}_m \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \mathbf{I}_n$$

Forward Sweep

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} \\
\nabla_{\mathbf{x}}\mathbf{v}_0 &= \mathbf{I} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) \\
\nabla_{\mathbf{x}}\mathbf{v}_1 &= \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_0\right] \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) \\
\nabla_{\mathbf{x}}\mathbf{v}_2 &= \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_1\right] \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) \\
\nabla_{\mathbf{x}}\mathbf{v}_3 &= \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_2\right] \\
\mathbf{y} &= \mathbf{v}_3 \\
\nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{x}}\mathbf{v}_3
\end{aligned}$$

Forward Mode



Reverse Mode

Forward and Reverse Mode AD

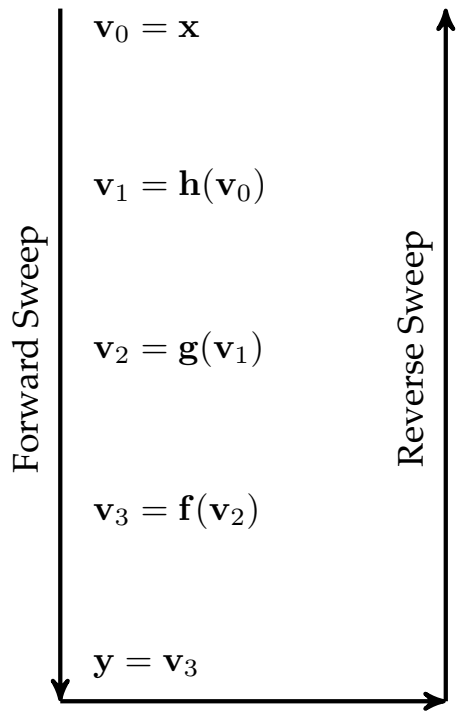
Ex: Given a program $\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{h}(\mathbf{x})))$

$$\nabla_{\mathbf{x}}\mathbf{y} = \mathbf{I}_m \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \mathbf{I}_n$$

Forward Sweep

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} \\
\nabla_{\mathbf{x}}\mathbf{v}_0 &= \mathbf{I} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) \\
\nabla_{\mathbf{x}}\mathbf{v}_1 &= \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_0\right] \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) \\
\nabla_{\mathbf{x}}\mathbf{v}_2 &= \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_1\right] \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) \\
\nabla_{\mathbf{x}}\mathbf{v}_3 &= \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_2\right] \\
\mathbf{y} &= \mathbf{v}_3 \\
\nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{x}}\mathbf{v}_3
\end{aligned}$$

Forward Mode



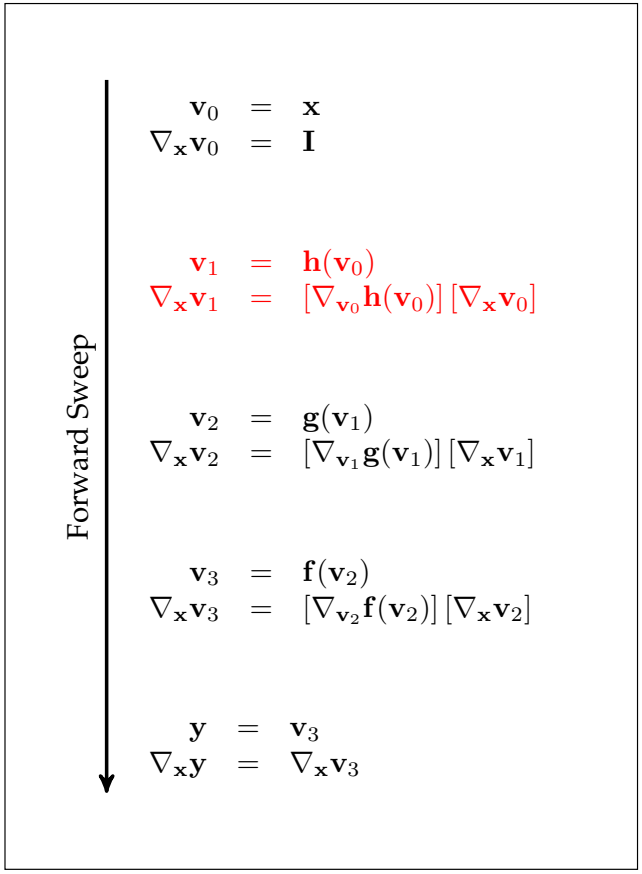
$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} & \nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{v}_0}\mathbf{y} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) & \nabla_{\mathbf{v}_0}\mathbf{y} &= \left[\nabla_{\mathbf{v}_1}\mathbf{y}\right] \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) & \nabla_{\mathbf{v}_1}\mathbf{y} &= \left[\nabla_{\mathbf{v}_2}\mathbf{y}\right] \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) & \nabla_{\mathbf{v}_2}\mathbf{y} &= \left[\nabla_{\mathbf{v}_3}\mathbf{y}\right] \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \\
\mathbf{y} &= \mathbf{v}_3 & \nabla_{\mathbf{v}_3}\mathbf{y} &= \mathbf{I}
\end{aligned}$$

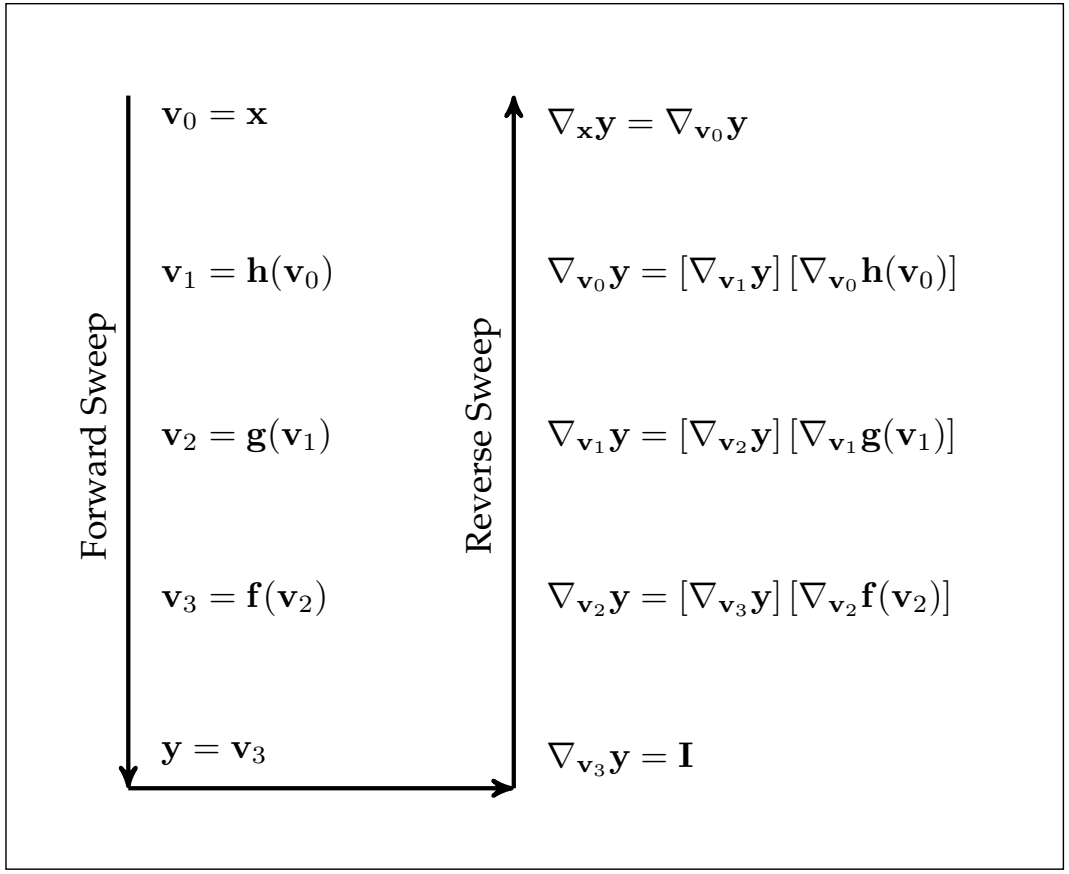
Reverse Mode

Forward and Reverse Mode AD

Ex: Given a program $\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{h}(\mathbf{x})))$

$$\nabla_{\mathbf{x}}\mathbf{y} = \mathbf{I}_m \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \mathbf{I}_n$$





Forward Mode

Reverse Mode

Forward and Reverse Mode AD

Ex: Given a program $\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{h}(\mathbf{x})))$

$$\nabla_{\mathbf{x}}\mathbf{y} = \mathbf{I}_m \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \mathbf{I}_n$$

Forward Sweep

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} \\
\nabla_{\mathbf{x}}\mathbf{v}_0 &= \mathbf{I} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) \\
\nabla_{\mathbf{x}}\mathbf{v}_1 &= [\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)]\,[\nabla_{\mathbf{x}}\mathbf{v}_0] \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) \\
\nabla_{\mathbf{x}}\mathbf{v}_2 &= [\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)]\,[\nabla_{\mathbf{x}}\mathbf{v}_1] \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) \\
\nabla_{\mathbf{x}}\mathbf{v}_3 &= [\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)]\,[\nabla_{\mathbf{x}}\mathbf{v}_2] \\
\mathbf{y} &= \mathbf{v}_3 \\
\nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{x}}\mathbf{v}_3
\end{aligned}$$

Forward Mode

Forward Sweep

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) \\
\mathbf{y} &= \mathbf{v}_3
\end{aligned}$$

Reverse Sweep

$$\begin{aligned}
\nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{v}_0}\mathbf{y} \\
\nabla_{\mathbf{v}_0}\mathbf{y} &= [\nabla_{\mathbf{v}_1}\mathbf{y}]\,[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)] \\
\nabla_{\mathbf{v}_1}\mathbf{y} &= [\nabla_{\mathbf{v}_2}\mathbf{y}]\,[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)] \\
\nabla_{\mathbf{v}_2}\mathbf{y} &= [\nabla_{\mathbf{v}_3}\mathbf{y}]\,[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)] \\
\nabla_{\mathbf{v}_3}\mathbf{y} &= \mathbf{I}
\end{aligned}$$

Reverse Mode

Forward and Reverse Mode AD

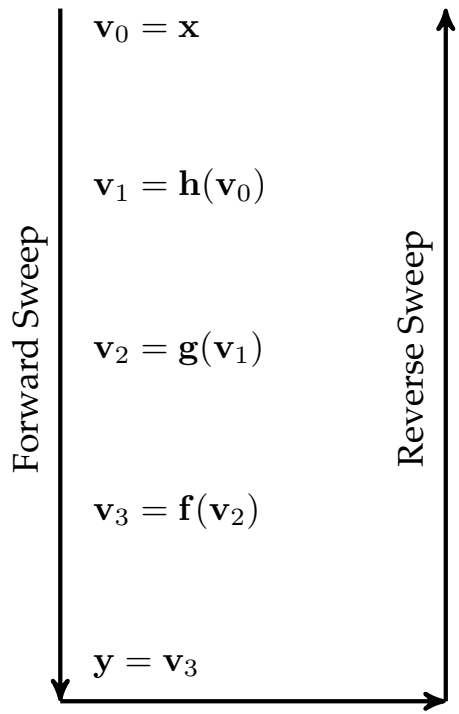
Ex: Given a program $\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{h}(\mathbf{x})))$

$$\nabla_{\mathbf{x}}\mathbf{y} = \mathbf{I}_m \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \mathbf{I}_n$$

Forward Sweep

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} \\
\nabla_{\mathbf{x}}\mathbf{v}_0 &= \mathbf{I} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) \\
\nabla_{\mathbf{x}}\mathbf{v}_1 &= [\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)]\,[\nabla_{\mathbf{x}}\mathbf{v}_0] \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) \\
\nabla_{\mathbf{x}}\mathbf{v}_2 &= [\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)]\,[\nabla_{\mathbf{x}}\mathbf{v}_1] \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) \\
\nabla_{\mathbf{x}}\mathbf{v}_3 &= [\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)]\,[\nabla_{\mathbf{x}}\mathbf{v}_2] \\
\mathbf{y} &= \mathbf{v}_3 \\
\nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{x}}\mathbf{v}_3
\end{aligned}$$

Forward Mode



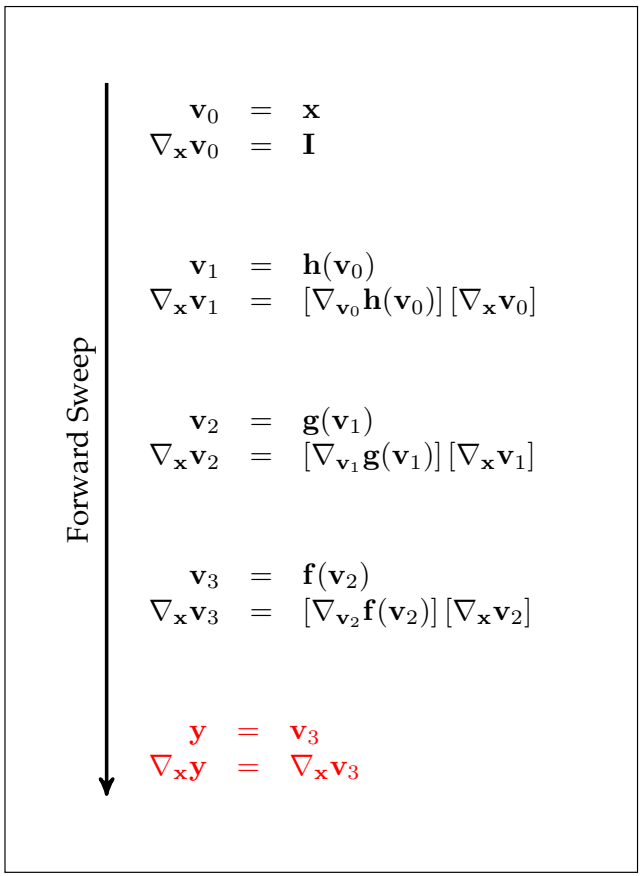
$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} & \nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{v}_0}\mathbf{y} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) & \nabla_{\mathbf{v}_0}\mathbf{y} &= [\nabla_{\mathbf{v}_1}\mathbf{y}]\,[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)] \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) & \nabla_{\mathbf{v}_1}\mathbf{y} &= [\nabla_{\mathbf{v}_2}\mathbf{y}]\,[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)] \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) & \nabla_{\mathbf{v}_2}\mathbf{y} &= [\nabla_{\mathbf{v}_3}\mathbf{y}]\,[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)] \\
\mathbf{y} &= \mathbf{v}_3 & \nabla_{\mathbf{v}_3}\mathbf{y} &= \mathbf{I}
\end{aligned}$$

Reverse Mode

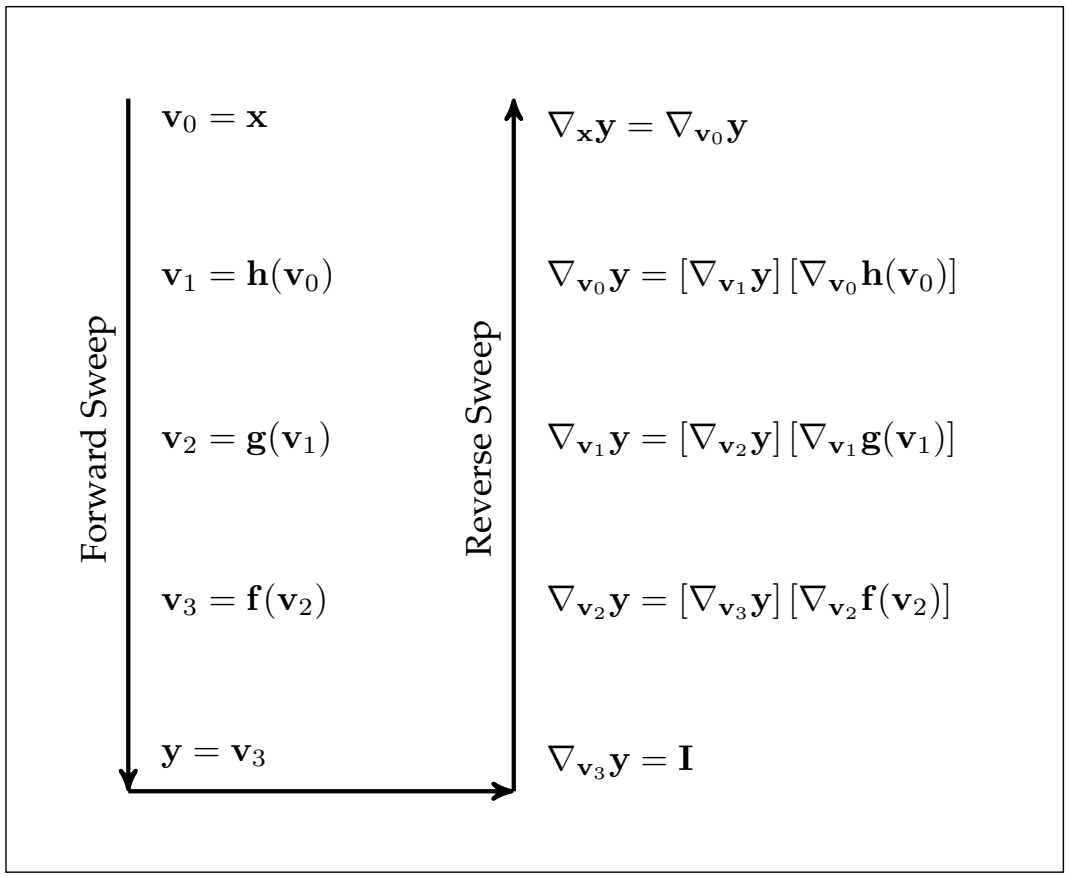
Forward and Reverse Mode AD

Ex: Given a program $\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{h}(\mathbf{x})))$

$$\nabla_{\mathbf{x}}\mathbf{y} = \mathbf{I}_m \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right]\left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right]\left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right]\mathbf{I}_n$$



Forward Mode



Reverse Mode

Forward and Reverse Mode AD

Ex: Given a program $\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{h}(\mathbf{x})))$

$$\nabla_{\mathbf{x}}\mathbf{y} = \mathbf{I}_m \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \mathbf{I}_n$$

Forward Sweep

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} \\
\nabla_{\mathbf{x}}\mathbf{v}_0 &= \mathbf{I} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) \\
\nabla_{\mathbf{x}}\mathbf{v}_1 &= \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_0\right] \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) \\
\nabla_{\mathbf{x}}\mathbf{v}_2 &= \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_1\right] \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) \\
\nabla_{\mathbf{x}}\mathbf{v}_3 &= \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_2\right] \\
\mathbf{y} &= \mathbf{v}_3 \\
\nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{x}}\mathbf{v}_3
\end{aligned}$$

Forward Mode

Forward Sweep

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) \\
\mathbf{y} &= \mathbf{v}_3
\end{aligned}$$

Reverse Sweep

$$\begin{aligned}
\nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{v}_0}\mathbf{y} \\
\nabla_{\mathbf{v}_0}\mathbf{y} &= \left[\nabla_{\mathbf{v}_1}\mathbf{y}\right] \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \\
\nabla_{\mathbf{v}_1}\mathbf{y} &= \left[\nabla_{\mathbf{v}_2}\mathbf{y}\right] \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \\
\nabla_{\mathbf{v}_2}\mathbf{y} &= \left[\nabla_{\mathbf{v}_3}\mathbf{y}\right] \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \\
\nabla_{\mathbf{v}_3}\mathbf{y} &= \mathbf{I}
\end{aligned}$$

Reverse Mode

Forward and Reverse Mode AD

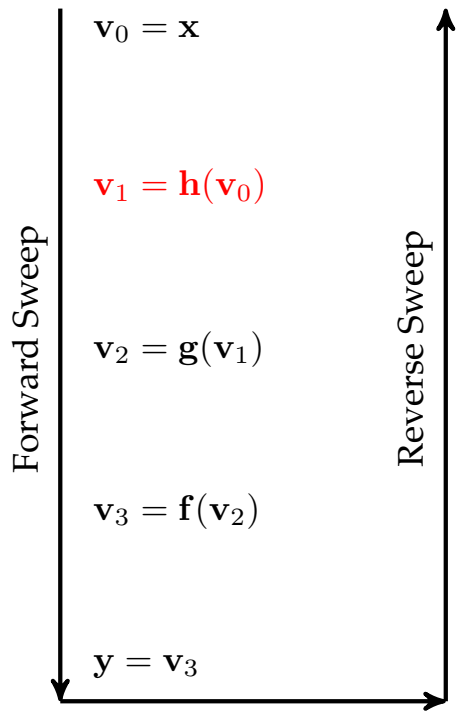
Ex: Given a program $\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{h}(\mathbf{x})))$

$$\nabla_{\mathbf{x}}\mathbf{y} = \mathbf{I}_m \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \mathbf{I}_n$$

Forward Sweep

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} \\
\nabla_{\mathbf{x}}\mathbf{v}_0 &= \mathbf{I} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) \\
\nabla_{\mathbf{x}}\mathbf{v}_1 &= \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_0\right] \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) \\
\nabla_{\mathbf{x}}\mathbf{v}_2 &= \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_1\right] \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) \\
\nabla_{\mathbf{x}}\mathbf{v}_3 &= \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_2\right] \\
\mathbf{y} &= \mathbf{v}_3 \\
\nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{x}}\mathbf{v}_3
\end{aligned}$$

Forward Mode

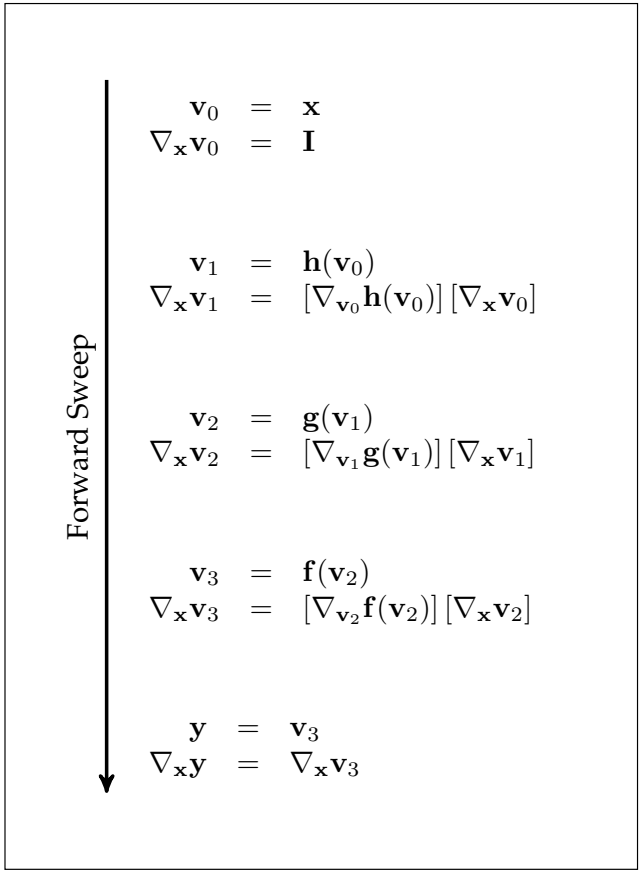


Reverse Mode

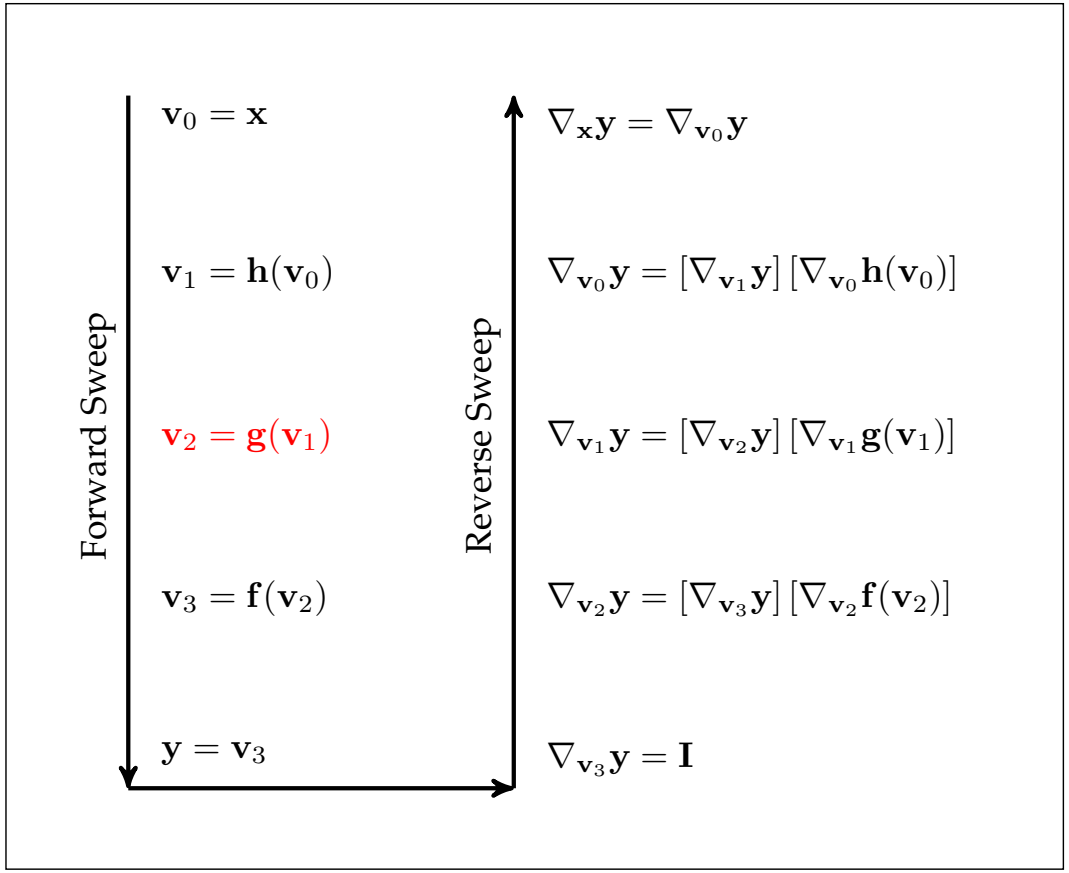
Forward and Reverse Mode AD

Ex: Given a program $\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{h}(\mathbf{x})))$

$$\nabla_{\mathbf{x}}\mathbf{y} = \mathbf{I}_m \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \mathbf{I}_n$$



Forward Mode



Reverse Mode

Forward and Reverse Mode AD

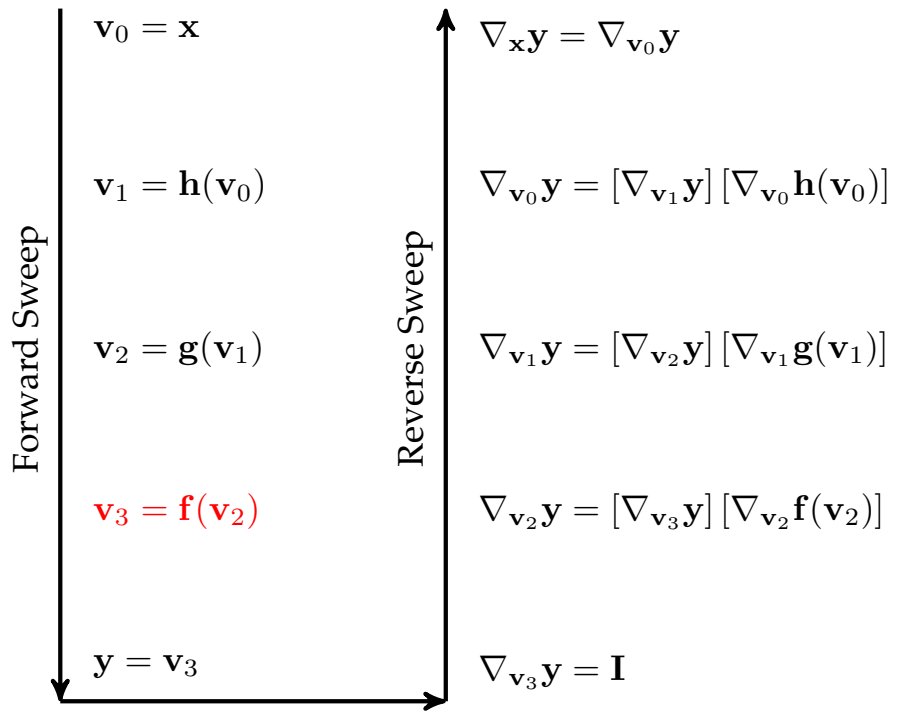
Ex: Given a program $\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{h}(\mathbf{x})))$

$$\nabla_{\mathbf{x}}\mathbf{y} = \mathbf{I}_m \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \mathbf{I}_n$$

Forward Sweep

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} \\
\nabla_{\mathbf{x}}\mathbf{v}_0 &= \mathbf{I} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) \\
\nabla_{\mathbf{x}}\mathbf{v}_1 &= \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_0\right] \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) \\
\nabla_{\mathbf{x}}\mathbf{v}_2 &= \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_1\right] \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) \\
\nabla_{\mathbf{x}}\mathbf{v}_3 &= \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_2\right] \\
\mathbf{y} &= \mathbf{v}_3 \\
\nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{x}}\mathbf{v}_3
\end{aligned}$$

Forward Mode

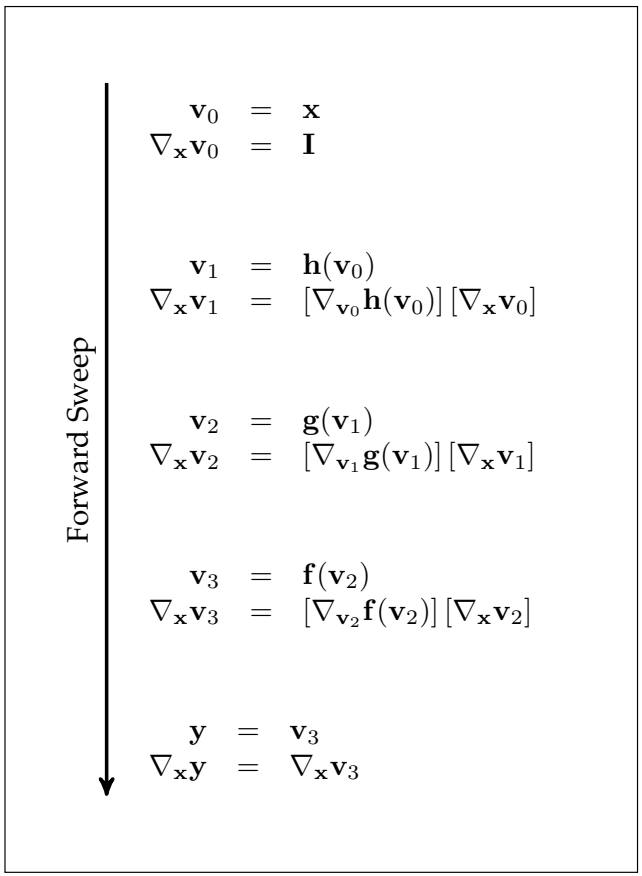


Reverse Mode

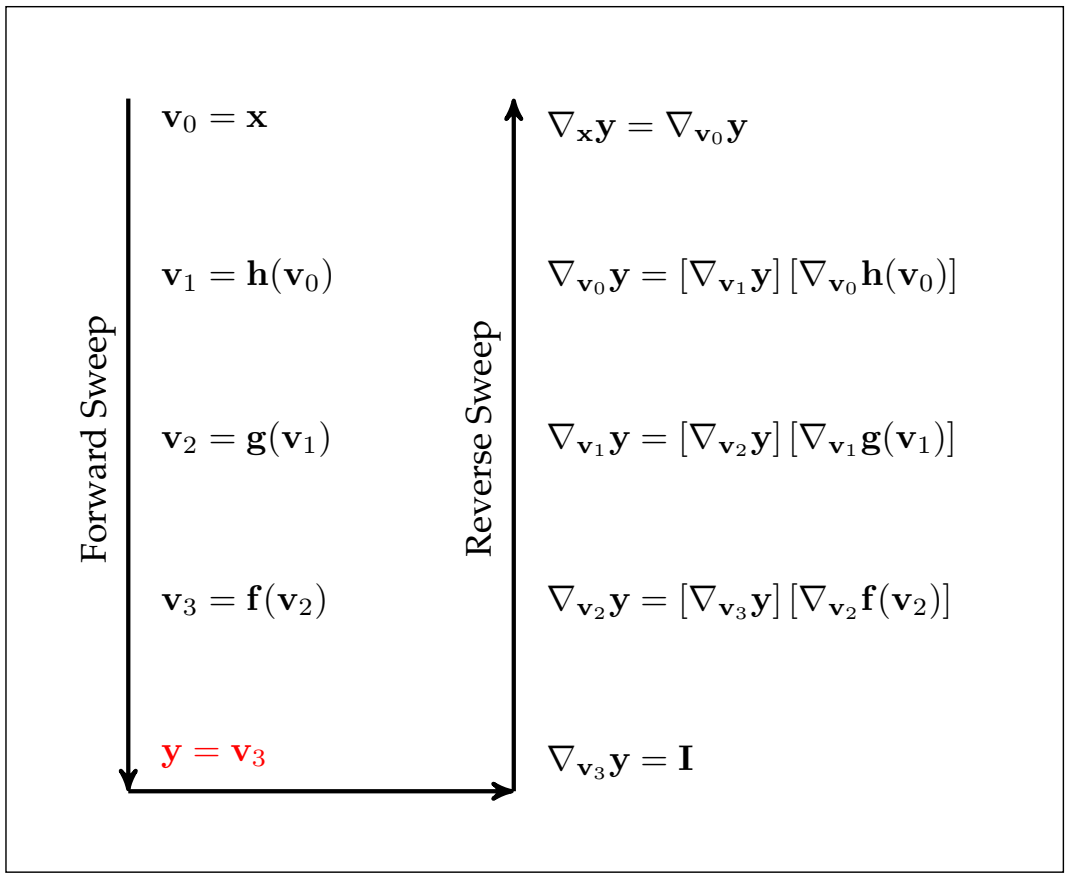
Forward and Reverse Mode AD

Ex: Given a program $\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{h}(\mathbf{x})))$

$$\nabla_{\mathbf{x}}\mathbf{y} = \mathbf{I}_m \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \mathbf{I}_n$$



Forward Mode



Reverse Mode

Forward and Reverse Mode AD

Ex: Given a program $\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{h}(\mathbf{x})))$

$$\nabla_{\mathbf{x}}\mathbf{y} = \mathbf{I}_m \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right]\left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right]\left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right]\mathbf{I}_n$$

Forward Sweep

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} \\
\nabla_{\mathbf{x}}\mathbf{v}_0 &= \mathbf{I} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) \\
\nabla_{\mathbf{x}}\mathbf{v}_1 &= \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right]\left[\nabla_{\mathbf{x}}\mathbf{v}_0\right] \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) \\
\nabla_{\mathbf{x}}\mathbf{v}_2 &= \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right]\left[\nabla_{\mathbf{x}}\mathbf{v}_1\right] \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) \\
\nabla_{\mathbf{x}}\mathbf{v}_3 &= \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right]\left[\nabla_{\mathbf{x}}\mathbf{v}_2\right] \\
\mathbf{y} &= \mathbf{v}_3 \\
\nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{x}}\mathbf{v}_3
\end{aligned}$$

Forward Mode

Forward Sweep

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) \\
\mathbf{y} &= \mathbf{v}_3
\end{aligned}$$

Reverse Sweep

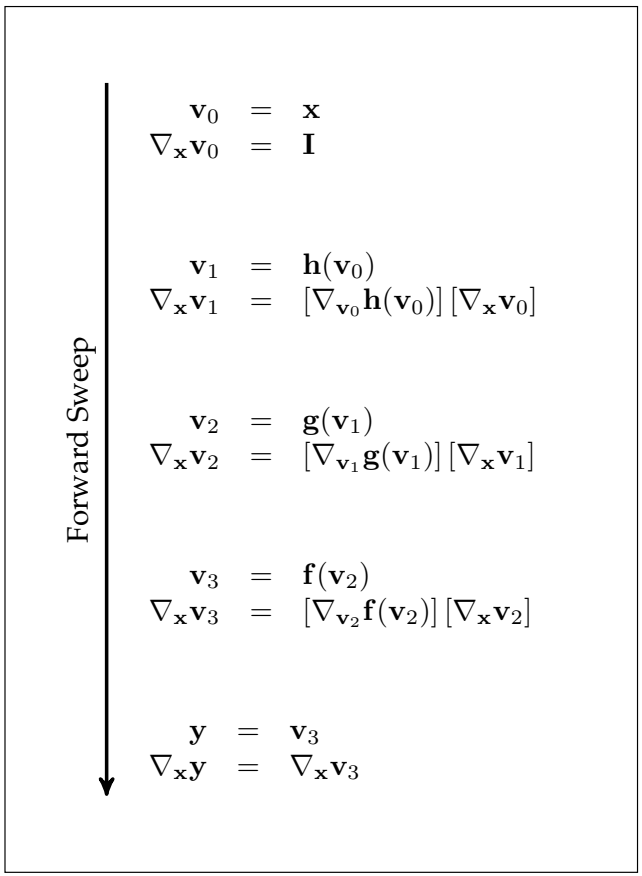
$$\begin{aligned}
\nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{v}_0}\mathbf{y} \\
\nabla_{\mathbf{v}_0}\mathbf{y} &= \left[\nabla_{\mathbf{v}_1}\mathbf{y}\right]\left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \\
\nabla_{\mathbf{v}_1}\mathbf{y} &= \left[\nabla_{\mathbf{v}_2}\mathbf{y}\right]\left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \\
\nabla_{\mathbf{v}_2}\mathbf{y} &= \left[\nabla_{\mathbf{v}_3}\mathbf{y}\right]\left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \\
\nabla_{\mathbf{v}_3}\mathbf{y} &= \mathbf{I}
\end{aligned}$$

Reverse Mode

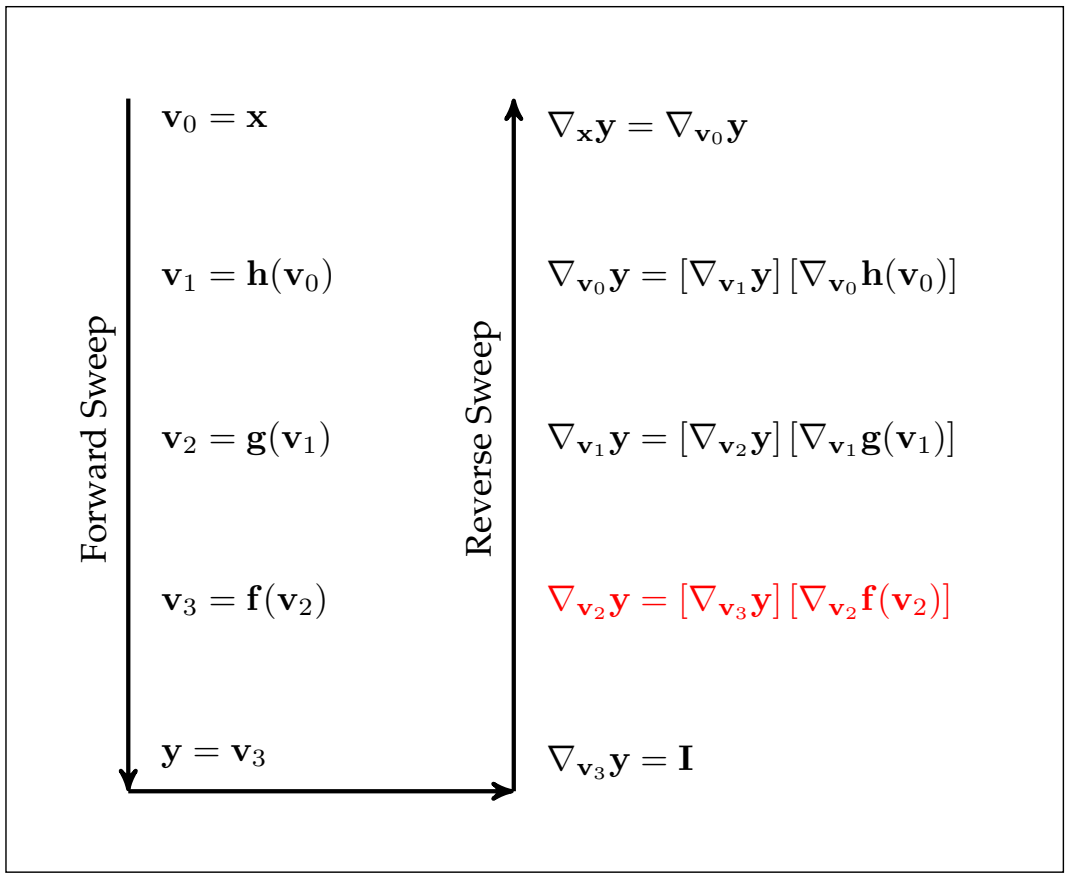
Forward and Reverse Mode AD

Ex: Given a program $\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{h}(\mathbf{x})))$

$$\nabla_{\mathbf{x}}\mathbf{y} = \mathbf{I}_m \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \mathbf{I}_n$$



Forward Mode



Reverse Mode

Forward and Reverse Mode AD

Ex: Given a program $\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{h}(\mathbf{x})))$

$$\nabla_{\mathbf{x}}\mathbf{y} = \mathbf{I}_m \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \mathbf{I}_n$$

Forward Sweep

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} \\
\nabla_{\mathbf{x}}\mathbf{v}_0 &= \mathbf{I} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) \\
\nabla_{\mathbf{x}}\mathbf{v}_1 &= \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right]\left[\nabla_{\mathbf{x}}\mathbf{v}_0\right] \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) \\
\nabla_{\mathbf{x}}\mathbf{v}_2 &= \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right]\left[\nabla_{\mathbf{x}}\mathbf{v}_1\right] \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) \\
\nabla_{\mathbf{x}}\mathbf{v}_3 &= \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right]\left[\nabla_{\mathbf{x}}\mathbf{v}_2\right] \\
\mathbf{y} &= \mathbf{v}_3 \\
\nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{x}}\mathbf{v}_3
\end{aligned}$$

Forward Mode

Forward Sweep

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) \\
\mathbf{y} &= \mathbf{v}_3
\end{aligned}$$

Reverse Sweep

$$\begin{aligned}
\nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{v}_0}\mathbf{y} \\
\nabla_{\mathbf{v}_0}\mathbf{y} &= \left[\nabla_{\mathbf{v}_1}\mathbf{y}\right]\left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \\
\nabla_{\mathbf{v}_1}\mathbf{y} &= \left[\nabla_{\mathbf{v}_2}\mathbf{y}\right]\left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \\
\nabla_{\mathbf{v}_2}\mathbf{y} &= \left[\nabla_{\mathbf{v}_3}\mathbf{y}\right]\left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \\
\nabla_{\mathbf{v}_3}\mathbf{y} &= \mathbf{I}
\end{aligned}$$

Reverse Mode

Forward and Reverse Mode AD

Ex: Given a program $\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{h}(\mathbf{x})))$

$$\nabla_{\mathbf{x}}\mathbf{y} = \mathbf{I}_m \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \mathbf{I}_n$$

Forward Sweep

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} \\
\nabla_{\mathbf{x}}\mathbf{v}_0 &= \mathbf{I} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) \\
\nabla_{\mathbf{x}}\mathbf{v}_1 &= \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_0\right] \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) \\
\nabla_{\mathbf{x}}\mathbf{v}_2 &= \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_1\right] \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) \\
\nabla_{\mathbf{x}}\mathbf{v}_3 &= \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_2\right] \\
\mathbf{y} &= \mathbf{v}_3 \\
\nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{x}}\mathbf{v}_3
\end{aligned}$$

Forward Mode

Forward Sweep

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) \\
\mathbf{y} &= \mathbf{v}_3
\end{aligned}$$

Reverse Sweep

$$\begin{aligned}
\nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{v}_0}\mathbf{y} \\
\nabla_{\mathbf{v}_0}\mathbf{y} &= \left[\nabla_{\mathbf{v}_1}\mathbf{y}\right] \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \\
\nabla_{\mathbf{v}_1}\mathbf{y} &= \left[\nabla_{\mathbf{v}_2}\mathbf{y}\right] \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \\
\nabla_{\mathbf{v}_2}\mathbf{y} &= \left[\nabla_{\mathbf{v}_3}\mathbf{y}\right] \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \\
\nabla_{\mathbf{v}_3}\mathbf{y} &= \mathbf{I}
\end{aligned}$$

Reverse Mode

Forward and Reverse Mode AD

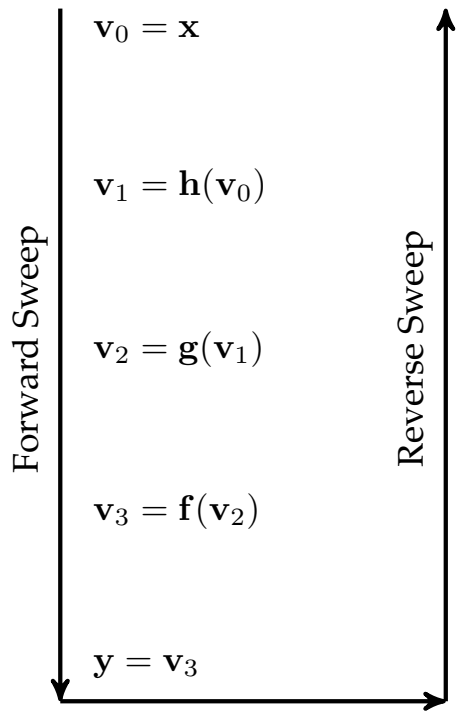
Ex: Given a program $\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{h}(\mathbf{x})))$

$$\nabla_{\mathbf{x}}\mathbf{y} = \mathbf{I}_m \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \mathbf{I}_n$$

Forward Sweep

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{x} \\
\nabla_{\mathbf{x}}\mathbf{v}_0 &= \mathbf{I} \\
\mathbf{v}_1 &= \mathbf{h}(\mathbf{v}_0) \\
\nabla_{\mathbf{x}}\mathbf{v}_1 &= \left[\nabla_{\mathbf{v}_0}\mathbf{h}(\mathbf{v}_0)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_0\right] \\
\mathbf{v}_2 &= \mathbf{g}(\mathbf{v}_1) \\
\nabla_{\mathbf{x}}\mathbf{v}_2 &= \left[\nabla_{\mathbf{v}_1}\mathbf{g}(\mathbf{v}_1)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_1\right] \\
\mathbf{v}_3 &= \mathbf{f}(\mathbf{v}_2) \\
\nabla_{\mathbf{x}}\mathbf{v}_3 &= \left[\nabla_{\mathbf{v}_2}\mathbf{f}(\mathbf{v}_2)\right] \left[\nabla_{\mathbf{x}}\mathbf{v}_2\right] \\
\mathbf{y} &= \mathbf{v}_3 \\
\nabla_{\mathbf{x}}\mathbf{y} &= \nabla_{\mathbf{x}}\mathbf{v}_3
\end{aligned}$$

Forward Mode



Reverse Mode

Previous Approaches to Algorithmic Differentiation

- Operator overloading
  - Create custom class with overloaded versions of all math functions
  - Operate on instances of custom class
  - Derivatives computed at numeric value of class instance
  - Pros and cons
    - ✓ Robust & easy to implement - only concerned with isolated operations
    - × Cannot perform optimizations
    - × Overhead incurred each time a derivative is computed
- Source transformation
  - Create derivative source code from original function code
  - Parse program $\rightarrow$ augment data $\rightarrow$ optimize $\rightarrow$ print derivative code
  - Pros and cons
    - ✓ Code typically more efficient than using operator overloading
    - ✓ Overhead only incurred a single time
    - ✓ In theory, can be applied recursively to obtain higher-order derivatives
    - × Very complex implementation

New Approach: Source Transformation via Operator Overloading

- Create run-time efficient derivative source code
- Employ forward mode AD and exploit derivative sparsity
- Key point: class instances contain *no* numeric information
- Information in a class instance
  - Variable size and symbolic identifier
  - Derivative sparsity pattern
- Key features of new approach to AD
  - Produces stand-alone derivative source code
  - Can be applied recursively to compute higher-order derivatives
  - Exploits sparsity and minimizes run-time overhead
  - Can differentiate large-dimensional functions (thousands of variables)
- Have published extensively on our new approach[*,†,‡]
- Approach has led to the new open-source AD tool ADiGator[‡]

[*] Patterson, M. A., Weinstein, M., and **Rao, A. V.**, "An Efficient Overloaded Method for Computing Derivatives of Mathematical Functions in MATLAB," *ACM Transactions on Mathematical Software*, Vol. 39, No. 3, pp. 17:1 - 17:36.

[†] Weinstein, M. J. and **Rao, A. V.**, "A Source Transformation via Operator Overloading Method for the Automatic Differentiation of Mathematical Functions in MATLAB," *ACM Transactions on Mathematical Software*, Accepted for Publication, Vol. 42, No. 1, October 2015, To Appear.

[‡] Weinstein, M. J. and **Rao, A. V.**, "Algorithm: ADiGator, a Toolbox for the Algorithmic Differentiation of Mathematical Functions in MATLAB Using Source Transformation via Operator Overloading,", *ACM Transactions on Mathematical Software*, Vol. 44, No. 2, Article 21, October 2017, pp. 21:1–21:25.

The ADiGator Algorithm*

- Input: user program $P(\mathbf{x})$ together with information to instantiate $\mathcal{X}$
- Transform user program P to intermediate program P'
  - Augment to the original program calls to transformation routines
  - Replace flow control statements with calls to transformation routines
- Evaluate intermediate program on $\mathcal{X}$ *three* times
  1. Empty Parsing Evaluation
     - Collect information on data and control flow
     - Determine what overmapped objects must be built
  2. Overmapping Evaluation
     - Build/store required overmapped objects and collect loop organizational operation data
  3. Printing Evaluation
     - Use data collected from previous two operations to print the derivative file

*Weinstein, M. J. and **Rao, A. V.**, "Algorithm: ADiGator, a Toolbox for the Algorithmic Differentiation of Mathematical Functions in MATLAB Using Source Transformation via Operator Overloading,", *ACM Transactions on Mathematical Software*, Vol. 44, No. 2, Article 21, October 2017, pp. 21:1–21:25.

Solution of Aircraft Climb Problem with ADiGator

hp				hp with ADiGator			
ϵ	CPU (s)	N	M	ϵ	CPU (s)	N	M
10^{-4}	3.92	74	2	10^{-4}	0.46	76	2
10^{-5}	5.28	102	3	10^{-5}	0.72	102	3
10^{-6}	6.71	157	4	10^{-6}	1.06	157	4
10^{-7}	7.12	227	4	10^{-7}	1.27	227	4
10^{-8}	6.87	380	4	10^{-8}	1.97	365	5
10^{-9}	9.49	620	5	10^{-9}	3.16	640	6
10^{-10}	11.87	902	6	10^{-10}	4.11	886	6

Summary: p, h, and hp-Adaptive Methods on Aircraft Climb Problem

p

ϵ	CPU (s)	N	M
10^{-4}	48.78	72	40
10^{-5}	198.33	137	95
10^{-6}	Fail	Fail	Fail
10^{-7}	Fail	Fail	Fail
10^{-8}	Fail	Fail	Fail
10^{-9}	Fail	Fail	Fail
10^{-10}	Fail	Fail	Fail

h

ϵ	CPU (s)	N	M
10^{-4}	5.66	96	3
10^{-5}	5.86	156	4
10^{-6}	7.11	264	5
10^{-7}	10.77	498	7
10^{-8}	17.33	921	10
10^{-9}	72.80	1614	34
10^{-10}	196.97	2964	61

hp

ϵ	CPU (s)	N	M
10^{-4}	3.92	74	2
10^{-5}	5.28	102	3
10^{-6}	6.71	157	4
10^{-7}	7.12	227	4
10^{-8}	6.87	380	4
10^{-9}	9.49	620	5
10^{-10}	11.87	902	6

hp with ADiGator

ϵ	CPU (s)	N	M
10^{-4}	0.46	76	2
10^{-5}	0.72	102	3
10^{-6}	1.06	157	4
10^{-7}	1.27	227	4
10^{-8}	1.97	365	5
10^{-9}	3.16	640	6
10^{-10}	4.11	886	6

Other Novel Applications of Optimal Control Framework

- Formula 1 (F-1) ground vehicle performance optimization[*,†]
  - F-1 community is "going green"
  - New requirements to include car energy recovery systems
  - Goal: minimize lap time subject to new design constraints
- Low-thrust orbital transfer trajectory optimization[‡,§]
  - Space missions moving from chemical to electric propulsion
  - Optimization key in minimizing time or fuel to transfer between orbits
  - Even more restrictive: solar electric propulsion (power only during sunlight)
- Improve walking motion in patients with brain injuries
  - Use optimal control to develop patient-specific therapies
  - Eventually want to test approaches in clinical settings
  - Need to solve complex optimal control problems quickly

[*] Limebeer, D. J. N, Perantoni, G., and **Rao, A. V.**, "Optimal Control of Formula One Car Energy Recovery Systems," *International Journal of Control*, Vol. 87, No. 10, October 2014, pp. 2065 - 2080.

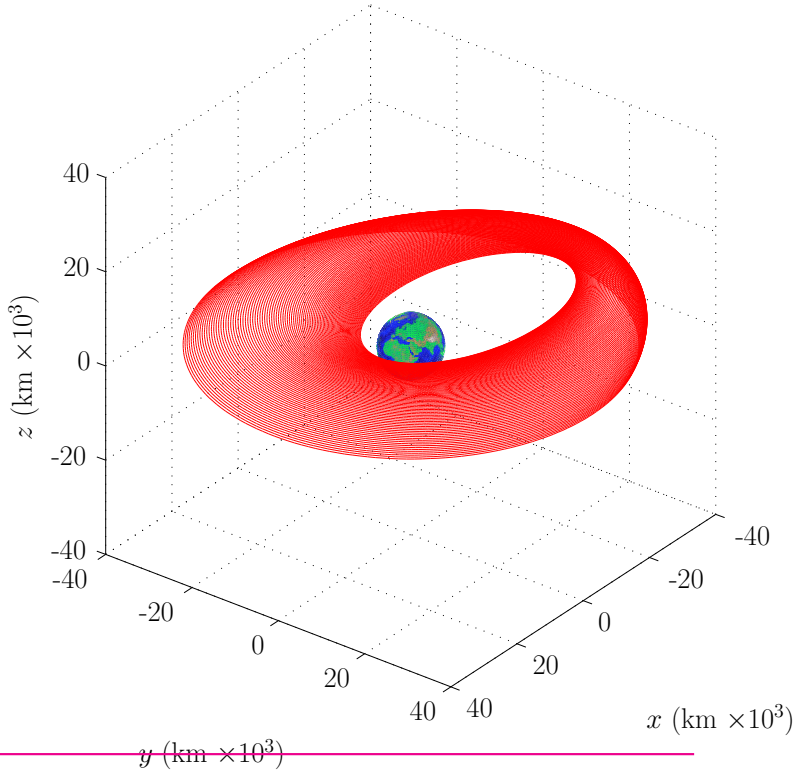
[†] Limebeer, D. J. N. and **Rao, A. V.**, "Faster, Higher, and Greener: Vehicular Optimal Control," *IEEE Control Systems Magazine*, Vol. 35, No. 2, April 2015, pp. 36 - 56.

[‡] Graham, K. F. and Rao, A. V., "Minimum-Time Trajectory Optimization of Many Revolution Low-Thrust Earth-Orbit Transfers," *Journal of Spacecraft and Rockets*, Vol. 52, No. 3, May - June 2015, pp. 711 - 727.

[§] Graham, K. F. and Rao, A. V., "Minimum-Time Trajectory Optimization of Low-Thrust Earth-Orbit Transfers with Eclipsing," *Journal of Spacecraft and Rockets*, Vol. 53, No. 2, March - April 2016, pp. 289 - 303.

Low-Thrust Earth-Orbit Transfer[*,†]

- Space missions moving from chemical to electric propulsion
- Objective: low thrust minimum-time to transfer from GTO to GEO
- Gravity perturbations J_2 through J_4 included
- Extremely challenging space flight optimal control problem
- Solved assuming full-time thrust and non-eclipsing thrust



- Transfer time: 74 days
- Orbital revolutions: 103
- Solution accurate to seven significant figures
- Optimality of solution verified with LGR costate estimation method[a,b]

[a] Garg, D., Patterson, M. A., Darby, C. L., Françolin, C., Huntington, G. T., Hager, W. W., and **Rao, A. V.**, "Direct Trajectory Optimization and Costate Estimation of Finite-Horizon and Infinite-Horizon Optimal Control Problems Using a Radau Pseudospectral Method," *Computational Optimization and Applications*, Vol. 49, No. 2, June 2011, pp. 335-358.

[b] Françolin, C. C., Hager, W. W., and Rao, A. V., "Costate Approximation in Optimal Control Using Integral Gaussian Quadrature Orthogonal Collocation Methods," *Optimal Control Applications and Methods*, Vol. 36, No. 4, July–August 2015, pp. 381–397

[*] Graham, K. F. and **Rao, A. V.**, "Minimum-Time Trajectory Optimization of Many Revolution Low-Thrust Earth-Orbit Transfers," *Journal of Spacecraft and Rockets*, Vol. 52, No. 3, May - June 2015, pp. 711 – 727.

[†] Graham, K. F. and Rao, A. V., "Minimum-Time Trajectory Optimization of Low-Thrust Earth-Orbit Transfers with Eclipsing," *Journal of Spacecraft and Rockets*, Vol. 53, No. 2, March - April 2016, pp. 289 - 303.

Impact Beyond Publications: Software

- *General-Purpose Optimal Control Software*[*] (GPOPS – II))
  - Implemented in MATLAB
  - All aforementioned algorithms included in GPOPS – II
- *Automatic Differentiation by Gators*[†] (ADiGator)
  - Implemented in MATLAB
  - Capable of differentiating highly complex and large functions
  - Special modules designed for use with GPOPS – II
- GPOPS – II & ADiGator used worldwide in academia and industry

[*]Patterson, M. A. and **Rao, A. V.**, "GPOPS-II: A MATLAB Software for Solving Multiple-Phase Optimal Control Problems Using hp-Adaptive Gaussian Quadrature Collocation Methods and Sparse Nonlinear Programming," *ACM Transactions on Mathematical Software*, Vol. 41, No. 1, October 2014, Article 1, pp. 1:1- 1:37.

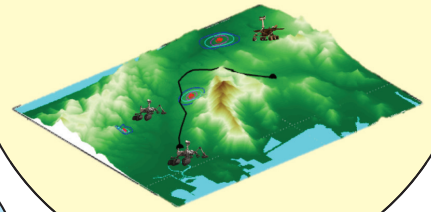
[†]Weinstein, M. J. and **Rao, A. V.**, "Algorithm: ADiGator, a Toolbox for the Algorithmic Differentiation of Mathematical Functions in MATLAB Using Source Transformation via Operator Overloading,", *ACM Transactions on Mathematical Software*, Vol. 44, No. 2, Article 21, October 2017, pp. 21:1–21:25.

Current and Future Research*,†,‡

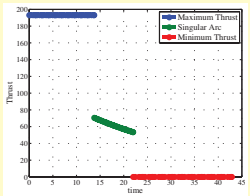
Development of Advanced Software for Use in Wide Variety of Engineering and Non-Engineering Applications

Hypersonic Mission Planning (U.S. Air Force)

Chance-Constrained Optimal Control (NSF)



***hp*-Adaptive Convergence Theory for Optimal Control (NSF and ONR)**



Real-Time Optimal Control of Uninhabited Aerial Vehicles (Navair)

*Cannataro, B., **Rao, A. V.**, and Davis, T. A., "State-Defect Pairing Graph Coarsening Method for Karush-Kuhn-Tucker Matrices Arising in Orthogonal Collocation Methods for Optimal Control," *Computational Optimization and Applications*, Vol. 64, No. 3, July 2016, pp. 793 - 819.

†Hager, W. W., Hou, H., and **Rao, A. V.**, "Convergence Rate for a Gauss Collocation Method Applied to Unconstrained Optimal Control," *Journal of Optimization Theory and Applications*, Vol, 169, No. 3, June 2016, pp. 801 - 824.

‡Hager, W. W., Hou, H., and **Rao, A. V.**, "Convergence Rate for a Gauss Collocation Method Applied to Constrained Optimal Control," *SIAM Journal on Control and Optimization*, Vol. 56, No. 2, March–April 2018, pp. 1386–1411.

Current and Future Research

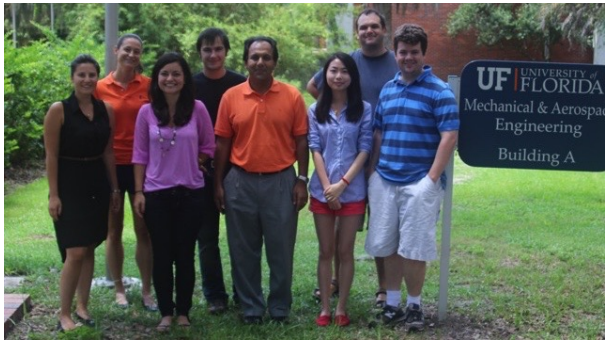
- Sparse structure of NLP arising from hp-adaptive methods
  - Large portion of NLP solver time: Karush-Kuhn-Tucker (KKT) system
  - Developed graph coarsening methods to improve KKT system efficiency*
- Development of specialized hp-adaptive NLP solvers
- Further improvements to hp mesh refinement
  - Still need to pin down locations of discontinuities
  - Need to handle problems with singular arcs
- Convergence theory and analysis †,‡
- Rapid mission planning for hypersonics
  - Current optimal control methods unsuitable for advanced hypersonics
  - Will develop tools in future to assess these technology needs
- Real-time trajectory optimization of aerial vehicles
  - Goal: move framework into real-time applications
  - Working to move technology into commercial sector

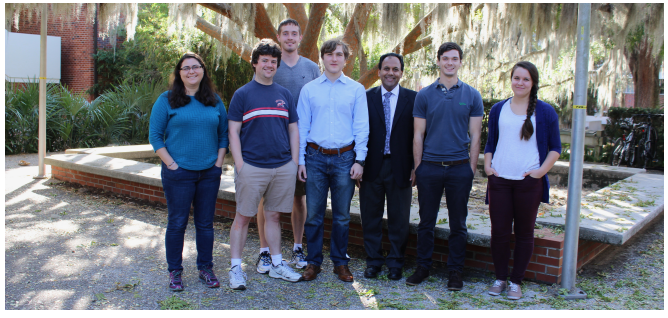
*Cannataro, B., **Rao, A. V.**, and Davis, T. A., "State-Defect Pairing Graph Coarsening Method for Karush-Kuhn-Tucker Matrices Arising in Orthogonal Collocation Methods for Optimal Control," *Computational Optimization and Applications*, Vol. 64, No. 3, July 2016, pp. 793 - 819.

†Hager, W. W., Hou, H., and **Rao, A. V.**, "Convergence Rate for a Gauss Collocation Method Applied to Unconstrained Optimal Control," *Journal of Optimization Theory and Applications*, Submitted for Publication, June 2015.

‡Hager, W. W., Hou, H., and **Rao, A. V.**, "Convergence Rate for a Radau Collocation Method Applied to Unconstrained Optimal Control," *SIAM Journal on Control and Optimization*, Submitted for Publication, August 2015.

Acknowledgments





Graduated Ph.D. Students	Current Ph.D. Students	Colloborators	Undergraduate Honors Theses	Current Undergraduates
Fengjin Liu	Joseph Eide	William Hager	Reagan Fuhr	Alexa Horn
Kathryn Graham	Yunus Agamawi	Mrinal Kumar	Erica Jenson	Anna Montgomery
Matthew Weinstein	Alexander Miller	B.J. Fregly (Rice)	Mallory Daly	Isabel Hess
Camila Françolin	Rachel Keil	David Limebeer (Oxford)	Rachel Hoffmann	Alexa Horn
Michael Patterson	Caleb Bowyer	Timothy Davis (Texas A&M)	Amber Walsh	Ashley Major
Divya Garg	Miriam Dennis	Rafael Haftka		
Christopher Darby		David Benson (Draper)		
Geoffrey Huntington				