A Primer on Pseudospectral Methods for Solving Optimal Control Problems

Anil V. Rao

Gainesville, FL 32607

12 – 15 March 2012

Preliminaries

This Primer consists of material that has been taken exclusively from the following documents available in the public domain:

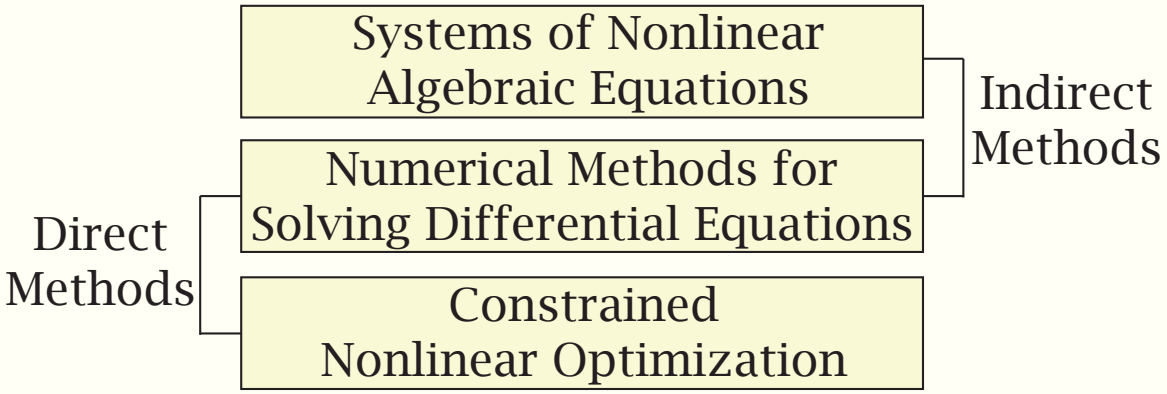
- The Following Textbooks on Optimal Control:
  - Kirk, D. E., *Optimal Control Theory: An Introduction*, Dover Publications, Mineola, New York, 2004.
  - Athans, M. A. and Falb, P. E., *Optimal Control: An Introduction to the Theory and Its Applications*, Dover Publications, Mineola, New York, 2006.
  - Bryson, A. E. and Ho, Y-C, *Applied Optimal Control*, Hemisphere Publishing, New York, 1975.
  - Dr. Anil V. Rao's Lecture Notes from University of Florida Course EML 6934. URL: http://vdol.mae.ufl.edu.
  - Various Journal and Conference Articles on Pseudospectral Methods Co-Authored by Dr. Anil V. Rao. URL: http://vdol.mae.ufl.edu.
  - Benson, D. A., *A Gauss Pseudospectral Transcription for Optimal Control*, PhD Thesis, Department of Aeronautics and Astronautics, MIT, November 2004.

- Huntington, G. T., *Advancement and Analysis of a Gauss Pseudospectral Transcription for Optimal Control*, PhD Thesis, Department of Aeronautics and Astronautics, MIT, May 2007.
- Darby, C. L., *hp Pseudospectral Methods for Solving Nonlinear Optimal Control Problems*, PhD Thesis, Department of Mechanical and Aerospace Engineering, University of Florida, April 2011.
- Garg, D., *Advancements in Global Pseudospectral Methods for Optimal Control*, PhD Thesis, Department of Mechanical and Aerospace Engineering, University of Florida, June 2011.

Course Schedule

- Day 1
  - Part I: Calculus of Variations (1.5 Hour)
  - Part II: Optimal Control Theory (1.5 Hour)
  - Part III: Nonlinear Optimization (1.5 Hours)
  - Part IV: Numerical Methods for Optimal Control (1.5 Hours)
- Day 2
  - Part V: Numerical Integration and Interplation (1 Hour)
  - Part VI: Foundation of Pseudospectral Methods (1 Hours)
  - Part VII: Implementation of a Pseudospectral Method (2 Hours)
  - Part VIII: Wrap-Up (1 Hour)

Ingredients for Solving an Optimal Control Problem



- Indirect Methods
  - ▷ First-Order Optimality Conditions of Continuous Problem Derived
  - ▷ Need to Solve Systems of Nonlinear Equations
  - ▷ Becomes a Root-Finding Problem
- Direct Methods
  - ▷ Optimal Control Problem Approximated in an Appropriate Manner
  - ▷ Becomes a Nonlinear Programming Problem
  - ▷ Problem Solved Using Techniques from Nonlinear Optimization
- Both Indirect and Direct Methods: Need to Solve Differential Equations

Part I: Calculus of Variations

Functions and Functionals

Definition 1

A *function* $f(\mathbf{q})$ is a rule or correspondence that assigns to each element $\mathbf{q} \in \mathcal{D}$ an element $f(\mathbf{q}) \in \mathcal{R}$. The quantities $\mathcal{D}$ and $\mathcal{R}$ are called the *domain* and *range* of the function, respectively.

Definition 2

A *functional* $J(\mathbf{x})$ is a rule or correspondence that assigns to each function $\mathbf{x} \in \Omega$ an element $J(\mathbf{x}) \in \mathbb{R}$. The quantity Ω is the *class* of functions over which the functional is defined and $\mathbb{R}$ is the set of real numbers.

Linearity of Functions

A function is said to be linear if the following property is satisfied $\forall\ \mathbf{p}, \mathbf{q} \in \mathcal{D}$ and $\forall\ \alpha, \beta \in \mathbb{R}$:

$$f(\alpha\mathbf{p} + \beta\mathbf{q}) = \alpha f(\mathbf{p}) + \beta f(\mathbf{q})$$

Linearity of Functionals

A functional is said to be linear if the following property is satisfied $\forall\ \mathbf{x}, \mathbf{y} \in \Omega$ and $\forall\ \alpha, \beta \in \mathbb{R}$:

$$J(\alpha\mathbf{x} + \beta\mathbf{y}) = \alpha J(\mathbf{x}) + \beta J(\mathbf{y})$$

Norms

The norm in $\mathbb{R}^n$ is a rule or correspondence that assigns to each point $\mathbf{q} \in \mathcal{D}$ a real number. The norm of $\mathbf{q}$, denoted $\|\mathbf{q}\|$, satisfies the following properties

1. $\|\mathbf{q}\| \geq 0, \forall\ \mathbf{q} \in \mathcal{D}\ (\mathbf{q} \neq \mathbf{0})$
2. $\|\mathbf{q}\| = 0 \Leftrightarrow \mathbf{q} = \mathbf{0}$
3. $\|\alpha\mathbf{q}\| = |\alpha|\|\mathbf{q}\|, \forall\ \alpha \in \mathbb{R}$
4. $\|\mathbf{p} + \mathbf{q}\| \leq \|\mathbf{p}\| + \|\mathbf{q}\|, \forall\ \mathbf{p}, \mathbf{q} \in \mathcal{D}$

Norm of a Function

The norm of a function is a rule or correspondence that assigns to each function $\mathbf{x} \in \Omega$ (where $\mathbf{x}$ is defined on $t \in [t_0, t_f]$) a real number. The norm of $\mathbf{x}$, denoted $\|\mathbf{x}\|$, satisfies the following properties

1. $\|\mathbf{x}\| \geq 0, \forall\ \mathbf{x} \in \Omega\ (\mathbf{x} \neq \mathbf{0})$
2. $\|\mathbf{x}\| = 0 \Leftrightarrow \mathbf{x} \equiv \mathbf{0}$ on $t \in [t_0, t_f]$
3. $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|, \forall\ \alpha \in \mathbb{R}$
4. $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|, \forall\ \mathbf{x}, \mathbf{y} \in \mathcal{D}$

Increment of a Function

Let f be a function and let $\mathbf{q}, \Delta\mathbf{q} \in \mathcal{D}$. Then the *increment* of the function from $\mathbf{q}$ to $\mathbf{q} + \Delta\mathbf{q}$, denoted Δf, is defined as

$$\Delta f = f(\mathbf{q} + \Delta\mathbf{q}) - f(\mathbf{q})$$

It is noted that Δf depends upon both $\mathbf{q}$ and $\Delta\mathbf{q}$, i.e., $\Delta f = \Delta f(\mathbf{q}, \Delta\mathbf{q})$

Increment of a Functional

Let J be a functional and let $\mathbf{x}, \delta\mathbf{x} \in \Omega$. Then the *increment* of the functional J from $\mathbf{x}$ to $\mathbf{x} + \delta\mathbf{x}$, denoted ΔJ, is defined as

$$\Delta J = J(\mathbf{x} + \delta\mathbf{x}) - J(\mathbf{x})$$

It is noted that δJ depends upon both $\mathbf{x}$ and $\delta\mathbf{x}$, i.e., $\Delta J = \Delta J(\mathbf{x}, \delta\mathbf{x})$

Differential of a Function

The general form for the increment of a function is given as

$$\Delta f(\mathbf{q}, \Delta\mathbf{q}) = df(\mathbf{q}, \Delta\mathbf{q}) + g(\mathbf{q}, \Delta\mathbf{q})\|\Delta\mathbf{q}\|$$

The quantity $df(\mathbf{q}, \Delta\mathbf{q})$ is called the *differential* of the function. The differential of a function is a linear operator on $\Delta\mathbf{q}$, i.e.,

$$df(\mathbf{q}, \alpha\mathbf{p} + \beta\mathbf{r}) = \alpha df(\mathbf{q}, \mathbf{p}) + \beta df(\mathbf{q}, \mathbf{r})$$

Variation of a Functional

The general form for the increment of a functional is given as

$$\Delta J(\mathbf{x}, \delta\mathbf{x}) = \delta J(\mathbf{x}, \delta\mathbf{x}) + g(\mathbf{x}, \delta\mathbf{x})\|\delta\mathbf{x}\|$$

The quantity $\delta J(\mathbf{x}, \delta\mathbf{x})$ is called the *variation* of the functional. The variation of a functional is a linear operator on $\delta\mathbf{x}$, i.e.,

$$\delta J(\mathbf{x}, \alpha\mathbf{y} + \beta\mathbf{z}) = \alpha\delta J(\mathbf{x}, \mathbf{y}) + \beta\delta J(\mathbf{x}, \mathbf{z})$$

Finally, the quantity $\delta\mathbf{x}$ is called the *variation* of the function $\mathbf{x}$.

Minima and Maxima of Functions and Functionals

Definition 3

A function f with domain $\mathcal{D}$ is said to have a *local extremum* at the point $\mathbf{q}^*$ if $\exists\ \epsilon > 0$ such that $\forall\ \mathbf{q} \in \mathcal{D}$ such that $\|\mathbf{q} - \mathbf{q}^*\| < \epsilon$, the increment Δf has the same sign. If

$$\Delta f = f(\mathbf{q}) - f(\mathbf{q}^*) \geq 0$$

then $f(\mathbf{q}^*)$ is said to be a *local minimum* of f. If

$$\Delta f = f(\mathbf{q}) - f(\mathbf{q}^*) \leq 0$$

then $f(\mathbf{q}^*)$ is said to be a *local maximum* of f. If either condition is satisfied for arbitrarily large ϵ, then $f(\mathbf{q}^*)$ is said to be a *global minimum* or *global maximum*.

Definition 4

A functional J with domain Ω is said to have a *local extremum* at the function $\mathbf{x}^*$ if $\exists\ \epsilon > 0$ such that $\forall\ \mathbf{x} \in \Omega$ such that $\|\mathbf{x} - \mathbf{x}^*\| < \epsilon$, the increment ΔJ has the same sign. If

$$\Delta J = J(\mathbf{x}) - J(\mathbf{x}^*) \geq 0$$

then $J(\mathbf{x}^*)$ is said to be a *local minimum* of J. If

$$\Delta J = J(\mathbf{x}) - J(\mathbf{x}^*) \leq 0$$

then $J(\mathbf{x}^*)$ is said to be a *local maximum* of J. If either condition is satisfied for arbitrarily large ϵ, then $J(\mathbf{x}^*)$ is said to be a *global minimum* or *global maximum*.

Basic Problem

- Solve Basic Calculus of Variations Problem Using *Variations*
- Specifically, Will Take the **First Variation** of J, denoted δJ
- Will Set $\delta J = 0$

$$\delta J = \delta \int_{t_0}^{t_f} \mathcal{L}[x(t), \dot{x}(t), t] dt$$

- Question: Which Quantities Vary (and Which Do Not Vary)?
  - ▷ Quantities $x(t)$ and $\dot{x}(t)$ Vary
  - ▷ However, t Does Not Vary Because It is a Dummy Variable of Integration
- Computation of δJ

$$\delta J = \delta \int_{t_0}^{t_f} \mathcal{L}[x(t), \dot{x}(t), t] dt$$

- Because t_0 and t_f are *Fixed*, We Have

$$\delta J = \int_{t_0}^{t_f} \delta\mathcal{L}[x(t), \dot{x}(t), t] dt$$

- Also, Because $x(t)$ and $\dot{x}(t)$ Vary on $t \in [t_0, t_f]$, We Have

$$\delta\mathcal{L} = \frac{\partial\mathcal{L}}{\partial x}\delta x + \frac{\partial\mathcal{L}}{\partial\dot{x}}\delta\dot{x}$$

- We Then Obtain

$$\delta J = \int_{t_0}^{t_f}\left[\frac{\partial\mathcal{L}}{\partial x}\delta x + \frac{\partial\mathcal{L}}{\partial\dot{x}}\delta\dot{x}\right]dt$$

- Applying Integration By Parts on Second Term,

$$\int_{t_0}^{t_f}\frac{\partial\mathcal{L}}{\partial\dot{x}}\delta\dot{x}dt = \left[\frac{\partial\mathcal{L}}{\partial\dot{x}}\delta x\right]_{t_0}^{t_f} - \int_{t_0}^{t_f}\frac{d}{dt}\left(\frac{\partial\mathcal{L}}{\partial\dot{x}}\right)\delta x dt$$

- Now Utilize the Following Information:
  - ▷ Initial and Terminal State is Fixed $\longrightarrow \delta x_0 = \delta x_f = 0$
  - ▷ Initial and Terminal Time is Fixed $\longrightarrow \delta t_0 = \delta t_f = 0$
  - ▷ Furthermore,

$$\begin{aligned}\delta x_0 &= \delta x(t_0) + \dot{x}(t_0)\delta t_0\\ \delta x_f &= \delta x(t_f) + \dot{x}(t_0)\delta t_f\end{aligned}$$

▷ Therefore, $\delta x(t_0) = \delta x(t_f) = 0$ Which Implies That

$$\left[\frac{\partial \mathcal{L}}{\partial \dot{x}}\delta x\right]_{t_0}^{t_f} = \left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right)_{t_f} \delta x(t_f) - \left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right)_{t_0} \delta x(t_0) = 0$$

- Consequently, We Obtain

$$\delta J = \int_{t_0}^{t_f} \left[\frac{\partial \mathcal{L}}{\partial x}\delta x - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right)\delta x\right] dt$$

- Last Equation Can Be Rewritten As

$$\delta J = \int_{t_0}^{t_f} \left[\frac{\partial \mathcal{L}}{\partial x} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right)\right] \delta x dt$$

- Because δJ Must be Zero for *All* Variations,

$$\frac{\partial \mathcal{L}}{\partial x} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right) = 0 \longrightarrow \text{Euler-Lagrange Equation}$$

Calculus of Variations Example

- Find an Extremal for the Functional

$$J[x(t)] = \int_{t_0}^{t_f} \left[\dot{x}^2(t) - x^2(t)\right] dt, \quad x(0) = 0, \quad x(\pi/2) = 1$$

- Solution: Euler-Lagrange Equation Given as

$$\frac{\partial \mathcal{L}}{\partial x} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right) = -2x - 2\ddot{x} = 0 \Longrightarrow \ddot{x} + x = 0$$

- General Solution of Differential Equation:

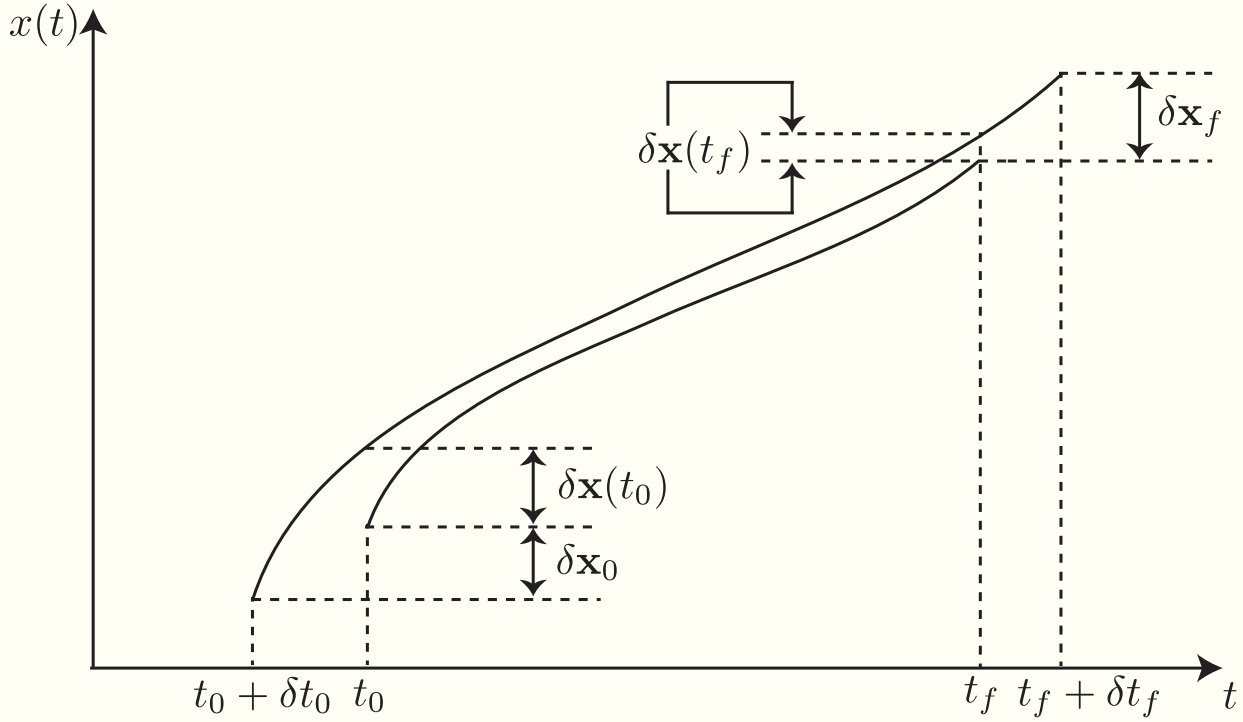
$$x(t) = c_1 \cos t + c_2 \sin t$$

- Applying Boundary Conditions, We Obtain $c_0 = 0$ and $c_1 = 1$. Therefore,

$$x^*(t) = \sin t$$

More General Calculus of Variations Problems

- Starting Point for Functional Optimization
- More General Problems: t_0, t_f, $x(t_0)$, and $x(t_f)$ May Be Free
- $\delta\mathbf{x}_0$ and $\delta\mathbf{x}_f$: Variation in Initial and Final value of $\mathbf{x}$
- $\delta\mathbf{x}(t_0)$ and $\delta\mathbf{x}(t_f)$: $\delta\mathbf{x}$ Evaluated at Initial and Terminal time
- Specifically, $\delta\mathbf{x}_0 = \delta\mathbf{x}(t_0) + \dot{\mathbf{x}}(t_0)\delta t_0$ and $\delta\mathbf{x}_f = \delta\mathbf{x}(t_f) + \dot{\mathbf{x}}(t_f)\delta t_f$



Free Final Time Functional Optimization Problem

- Consider Following Problem. Minimize

$$J[x(t)] \equiv J = \int_{t_0}^{t_f} \mathcal{L}[x(t), \dot{x}(t), t]dt$$

- Boundary Conditions:

$$\begin{aligned} t_0 &= \text{fixed} \quad , \quad t_f &= \text{free} \\ x(t_0) &= \text{fixed} \quad , \quad x(t_f) &= \text{fixed} \end{aligned}$$

Solution of Free Final Time COV Problem

- Taking δJ Gives

$$\delta J = \delta \int_{t_0}^{t_f} \mathcal{L}[x(t), \dot{x}(t), t]dt$$

- Because t_f is Free $\longrightarrow \delta t_f \neq 0$. Therefore,

$$\delta J = \frac{\partial J}{\partial t_f}\delta t_f + \int_{t_0}^{t_f} \delta\mathcal{L}[x(t), \dot{x}(t), t]dt$$

- Applying the Fundamental Theorem of Calculus, We Have

$$\frac{\partial J}{\partial t_f} = \mathcal{L}[x(t_f), \dot{x}(t_f), t_f] \Longrightarrow \delta J = \mathcal{L}[x(t_f), \dot{x}(t_f), t_f]\delta t_f + \int_{t_0}^{t_f} \delta\mathcal{L}[x(t), \dot{x}(t), t]dt$$

- Already Have Term Inside Integral from Earlier. Therefore,

$$\delta J = \mathcal{L}[x(t_f), \dot{x}(t_f), t_f]\delta t_f + \left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right)_{t_f} \delta x(t_f) - \left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right)_{t_0} \delta x(t_0)$$
$$+ \int_{t_0}^{t_f} \left[\frac{\partial \mathcal{L}}{\partial x} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right)\right] \delta x dt$$

- Now t_0 is Fixed $\longrightarrow \delta t_0 = 0$. Therefore,

$$\delta J = \mathcal{L}[x(t_f), \dot{x}(t_f), t_f]\delta t_f + \left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right)_{t_f} \delta x(t_f)$$
$$+ \int_{t_0}^{t_f} \left[\frac{\partial \mathcal{L}}{\partial x} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right)\right] \delta x dt$$

- Now the Final Value of x is Fixed. Therefore, $\delta x_f = 0$ Which Implies That

$$\delta x_f = \delta x(t_f) + \dot{x}(t_f)\delta t_f = 0 \Longrightarrow \delta x(t_f) = -\dot{x}(t_f)\delta t_f$$

- Therefore,

$$\delta J = \mathcal{L}[x(t_f), \dot{x}(t_f), t_f]\delta t_f - \left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right)_{t_f} \dot{x}(t_f)\delta t_f$$
$$+ \int_{t_0}^{t_f} \left[\frac{\partial \mathcal{L}}{\partial x} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right)\right] \delta x dt$$

- Combining Terms, We Obtain

$$\delta J = \left\{ \mathcal{L}[x(t_f), \dot{x}(t_f), t_f] - \left(\frac{\partial \mathcal{L}}{\partial \dot{x}} \right)_{t_f} \dot{x}(t_f) \right\} \delta t_f + \int_{t_0}^{t_f} \left[\frac{\partial \mathcal{L}}{\partial x} - \frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{x}} \right) \right] \delta x dt$$

- Now, Because δx and δt_f are Independent, We have

$$\begin{aligned} \mathcal{L}[x(t_f), \dot{x}(t_f), t_f] - \left(\frac{\partial \mathcal{L}}{\partial \dot{x}} \right)_{t_f} \dot{x}(t_f) &= 0 \\ \frac{\partial \mathcal{L}}{\partial x} - \frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{x}} \right) &= 0 \end{aligned}$$

Example with Free Final Time

- Minimize

$$J[x(t)] = \int_1^{t_f} \left[2x(t) + \frac{1}{2}\dot{x}^2(t)\right] dt, \quad x(1) = 4,\ x(t_f) = 4.\ t_f > 1$$

- Solution:
  - ▷ Euler-Lagrange Equation: $\ddot{x} = 2 \Longrightarrow x(t) = t^2 + c_1 t + c_2$
  - ▷ Boundary Condition $2x(t_f) - \frac{1}{2}\dot{x}^2(t_f) = 0$
  - ▷ Applying Boundary Condition Gives

$$\begin{aligned} x(1) &= 4 = 1 + c_1 + c_2 \Longrightarrow c_1 + c_2 = 3 \\ 2x(t_f) - \tfrac{1}{2}\dot{x}^2(t_f) &= 2(t_f^2 + c_1 t_f + c_2) + 2t_f + c_1 = 0 \end{aligned}$$

  - ▷ Final Solution

$$\begin{aligned} x^*(t) &= t^2 - 6t + 9 \\ t_f &= 5 \end{aligned}$$

Free Final Endpoint Functional Optimization Problem

- Consider Following Problem. Minimize

$$J[x(t)] \equiv J = \int_{t_0}^{t_f} \mathcal{L}[x(t), \dot{x}(t), t]dt$$

- Boundary Conditions

$$\begin{aligned} t_0 &= \text{fixed} \quad , \quad t_f &= \text{fixed} \\ x(t_0) &= \text{fixed} \quad , \quad x(t_f) &= \text{free} \end{aligned}$$

Minimization of Functionals of Vector Functions

- In General, Functionals May be Functions of *Vector* Functions
- Functional of a Vector Function: Let $\mathbf{x}(t) \in \mathbb{R}^n$,

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix}$$

- Functional of $\mathbf{x}(t)$

$$J[\mathbf{x}(t)] = \int_{t_0}^{t_f} \mathcal{L}[\mathbf{x}(t), \dot{\mathbf{x}}(t), t]dt$$

- Extremization of Functional of Vector Function

$$\delta J = \delta \int_{t_0}^{t_f} \mathcal{L}[\mathbf{x}(t), \dot{\mathbf{x}}(t), t]dt = 0$$

- Solved in a Manner Analogous to Function Optimization

- Assume That t_0, t_f, $x(t_0)$, and $x(t_f)$ Are *Fixed*

$$\Longrightarrow \delta J = \int_{t_0}^{t_f} \delta\mathcal{L}[\mathbf{x}(t), \dot{\mathbf{x}}(t), t]dt = 0$$

- Consequently,

$$\delta J = \int_{t_0}^{t_f} \left[\left(\frac{\partial \mathcal{L}}{\partial \mathbf{x}} \right)^{\mathsf{T}} \delta\mathbf{x} + \left(\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{x}}} \right)^{\mathsf{T}} \delta\dot{\mathbf{x}} \right] dt$$

- Integrating Second Term of Integrand By Parts Gives

$$\delta J = \left[\left(\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{x}}} \right)^{\mathsf{T}} \delta\mathbf{x} \right]_{t_0}^{t_f} + \int_{t_0}^{t_f} \left[\left(\frac{\partial \mathcal{L}}{\partial \mathbf{x}} \right)^{\mathsf{T}} - \frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{x}}} \right)^{\mathsf{T}} \right] \delta\mathbf{x} dt$$

- Applying Boundary Conditions $\Rightarrow \delta\mathbf{x}(t_0) = \delta\mathbf{x}(t_f) = \mathbf{0}$. Therefore,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} - \frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{x}}} \right) = \mathbf{0} \Longrightarrow \textbf{Vector Euler-Lagrange Equation}$$

- In Scalar Form

$$\frac{\partial \mathcal{L}}{\partial x_i} - \frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial x_i} \right) = 0, \qquad (i = 1, \ldots, n)$$

- Extremization of More General Functionals Proceeds in Analogous Manner

Summary

- Calculus of Variations
  - Foundation for *Functional* Optimization
  - Deals with Continuous-Time (Infinite-Dimensional) Problem
  - Candidate Optimal Solutions are Called *Extremals*
- Determination of an Extremal Solution
  - Compute First Variation of Cost Functional
  - Set All Independent Terms in Variation to Zero
- Importance of Calculus of Variations
  - Foundation for Optimal Control Theory
  - Optimal Control Problem: *Application* of Variational Calculus

Part II: Optimal Control Theory

General Optimal Control Problem

- Determine $\mathbf{x}(t)$, $\mathbf{u}(t)$, and $\mathbf{p}$ that Minimize the Cost Functional

$$J = \Phi(\mathbf{x}(t_f), t_f, \mathbf{p}) + \int_{t_0}^{t_f} \mathcal{L}[\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t]dt$$

- Subject to the Dynamic Constraint

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

- The Boundary Conditions

$$\boldsymbol{\phi}(\mathbf{x}(t_0), t_0, \mathbf{x}(t_f), t_f, \mathbf{p}) = \mathbf{0}$$

- And the Path Constraints

$$\mathbf{C}_{\min} \leq \mathbf{C}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}) \leq \mathbf{C}_{\max}$$

Key Features of Optimal Control Problem

- **Functional** Minimization Problem
  - ▷ $\mathbf{x}(t)$ and $\mathbf{u}(t)$ are *Functions* of Time
  - ▷ J is a Function of Functions $\mathbf{x}(t)$ & $\mathbf{u}(t)$ $\longrightarrow$ J is a *Functional*
- Optimal Control Problems are **Constrained**
- Constraints: Differential Equations, Boundary Conditions, & Path
- Need to Solve a *Constrained* Functional Minimization Problem
- Broad Classes of Optimal Control Problems
  - ▷ Free and Fixed Terminal Times
  - ▷ Free Terminal State (Initial-Value Problem)
  - ▷ Constrained Terminal State (Boundary-Value Problem)
  - ▷ Path Constrained and Path Unconstrained

Conventions Used in Derivation

- Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}$. The *Gradient* and *Hessian* of f are Defined As

$$\frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix}$$

$$\frac{\partial^2 f}{\partial \mathbf{x}^2} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_2 x_1} & \cdots & \frac{\partial^2 f}{\partial x_n x_1} \\ \frac{\partial^2 f}{\partial x_1 x_2} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_n x_1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 x_n} & \frac{\partial^2 f}{\partial x_2 x_n} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

- Using Conventions,We Have

$$\frac{\partial^2 f}{\partial \mathbf{x}^2} = \frac{\partial}{\partial \mathbf{x}} \left[\frac{\partial f}{\partial \mathbf{x}} \right]^{\mathsf{T}}$$

- Let $\mathbf{f} : \mathbb{R}^n \longrightarrow \mathbb{R}^m$. Then the *Jacobian* of $\mathbf{f}(\mathbf{x})$ is an $m \times n$ *matrix*, i.e.,

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \frac{\partial \mathbf{f}}{\partial x_2} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

Optimal Control Problem without Path Constraints

- Construct Augmented Cost Functional

$$J_a = \Phi - \boldsymbol{\nu}^{\mathsf{T}}\boldsymbol{\phi} + \int_{t_0}^{t_f} \left[\mathcal{L} + \boldsymbol{\lambda}^{\mathsf{T}}(\mathbf{f} - \dot{\mathbf{x}})\right] dt$$

  where

$$\begin{aligned} \boldsymbol{\lambda}(t) &= \text{Costate} \\ \boldsymbol{\nu} &= \text{Lagrange Multiplier of Boundary Condition} \end{aligned}$$

- Make Following Assumptions (For Now)
  - ▷ Initial and Terminal Times are Free
  - ▷ Control is Unconstrained
  - ▷ No Path Constraints

First-Order Optimality Conditions

- Augmented Cost Functional

$$J_a = \Phi - \boldsymbol{\nu}^{\mathsf{T}}\boldsymbol{\phi} + \int_{t_0}^{t_f} \left[\mathcal{L} + \boldsymbol{\lambda}^{\mathsf{T}}(\mathbf{f} - \dot{\mathbf{x}})\right] dt$$

- Terminology
  - ▷ $\boldsymbol{\nu}$: Lagrange Multiplier of Boundary Condition
  - ▷ $\boldsymbol{\lambda}$: Costate or Adjoint of Differential Equation
- Define

$$\mathcal{H} = \mathcal{L} + \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{f} = \mathcal{H}(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{u})$$

- The Quantity $\mathcal{H}$ is called the *Hamiltonian*
- In Terms of the Hamiltonian We Have

$$J_a = \Phi - \boldsymbol{\nu}^{\mathsf{T}}\boldsymbol{\phi} + \int_{t_0}^{t_f} \left[\mathcal{H} - \boldsymbol{\lambda}^{\mathsf{T}}\dot{\mathbf{x}}\right] dt$$

- First Variation of Augmented Cost Functional

$$\delta J_a = \delta\Phi - \delta\left\{\boldsymbol{\nu}^{\mathsf{T}}\boldsymbol{\phi} + \int_{t_0}^{t_f} \left[\mathcal{H} - \boldsymbol{\lambda}^{\mathsf{T}}\dot{\mathbf{x}}\right] dt\right\}$$

which gives

$$\delta J_a = \delta\Phi - \delta\boldsymbol{\nu}^{\mathsf{T}}\boldsymbol{\phi} - \boldsymbol{\nu}^{\mathsf{T}}\delta\boldsymbol{\phi} + \delta\int_{t_0}^{t_f}\left[\mathcal{H} - \boldsymbol{\lambda}^{\mathsf{T}}\dot{\mathbf{x}}\right]dt$$

- Now We Have

$$\begin{aligned}\delta\Phi = &\frac{\partial\Phi}{\partial\mathbf{x}(t_0)}\delta\mathbf{x}(t_0) + \left[\frac{\partial\Phi}{\partial\mathbf{x}(t_0)}\dot{\mathbf{x}}(t_0) + \frac{\partial\Phi}{\partial t_0}\right]\delta t_0\\ &+ \frac{\partial\Phi}{\partial\mathbf{x}(t_f)}\delta\mathbf{x}(t_f) + \left[\frac{\partial\Phi}{\partial\mathbf{x}(t_f)}\dot{\mathbf{x}}(t_f) + \frac{\partial\Phi}{\partial t_f}\right]\delta t_f\end{aligned}$$

- Combining Terms, We Have

$$\begin{aligned}\delta\Phi = &\frac{\partial\Phi}{\partial\mathbf{x}(t_0)}\left[\delta\mathbf{x}(t_0) + \dot{\mathbf{x}}(t_0)\delta t_0\right] + \frac{\partial\Phi}{\partial t_0}\delta t_0\\ &+ \frac{\partial\Phi}{\partial\mathbf{x}(t_f)}\left[\delta\mathbf{x}(t_f) + \dot{\mathbf{x}}(t_f)\delta t_f\right] + \frac{\partial\Phi}{\partial t_f}\delta t_f\end{aligned}$$

- Terminology

$$\begin{aligned} \delta\mathbf{x}_0, \delta\mathbf{x}_f &= \text{Variations in Initial and Terminal State} \\ \delta\mathbf{x}(t_0), \delta\mathbf{x}(t_f) &= \text{Variations in State at Initial} \\ &\quad \text{and Terminal Times} \end{aligned}$$

- Note That $\delta\mathbf{x}(t_0) \neq \delta\mathbf{x}_0$ and $\delta\mathbf{x}(t_f) \neq \delta\mathbf{x}_f$
- Relationships Between $\delta\mathbf{x}_0, \delta\mathbf{x}_f$ and $\delta\mathbf{x}(t_0), \delta\mathbf{x}(t_f)$

$$\begin{aligned} \delta\mathbf{x}_0 &= \delta\mathbf{x}(t_0) + \dot{\mathbf{x}}(t_0)\delta t_0 \\ \delta\mathbf{x}_f &= \delta\mathbf{x}(t_f) + \dot{\mathbf{x}}(t_f)\delta t_f \end{aligned}$$

- In Terms of $\delta\mathbf{x}_0$ and $\delta\mathbf{x}_f$, We Obtain

$$\boxed{\begin{aligned} \delta\Phi = &\frac{\partial\Phi}{\partial\mathbf{x}(t_0)}\delta\mathbf{x}_0 + \frac{\partial\Phi}{\partial t_0}\delta t_0 \\ &+ \frac{\partial\Phi}{\partial\mathbf{x}(t_f)}\delta\mathbf{x}_f + \frac{\partial\Phi}{\partial t_f}\delta t_f \end{aligned}}$$

- In a Similar Manner, We Have for $\boldsymbol{\phi}$ That

$$\boxed{\begin{aligned}\delta\boldsymbol{\phi} = &\frac{\partial\boldsymbol{\phi}}{\partial\mathbf{x}(t_0)}\delta\mathbf{x}_0 + \frac{\partial\boldsymbol{\phi}}{\partial t_0}\delta t_0 \\ &+ \frac{\partial\boldsymbol{\phi}}{\partial\mathbf{x}(t_f)}\delta\mathbf{x}_f + \frac{\partial\boldsymbol{\phi}}{\partial t_f}\delta t_f\end{aligned}}$$

- Next, We Have

$$\begin{aligned}\delta\int_{t_0}^{t_f}\left[\mathcal{H} - \boldsymbol{\lambda}^{\mathsf{T}}\dot{\mathbf{x}}\right]dt = &\int_{t_0}^{t_f}\left[\delta\mathcal{H} - \delta\boldsymbol{\lambda}^{\mathsf{T}}\dot{\mathbf{x}} - \boldsymbol{\lambda}^{\mathsf{T}}\delta\dot{\mathbf{x}}\right]dt \\ &+ \left[\mathcal{H} - \boldsymbol{\lambda}^{\mathsf{T}}\dot{\mathbf{x}}\right]_{t_f}\delta t_f \\ &- \left[\mathcal{H} - \boldsymbol{\lambda}^{\mathsf{T}}\dot{\mathbf{x}}\right]_{t_0}\delta t_0\end{aligned}$$

- Where

$$\delta\mathcal{H} = \left[\frac{\partial\mathcal{H}}{\partial\mathbf{x}}\right]\delta\mathbf{x} + \left[\frac{\partial\mathcal{H}}{\partial\boldsymbol{\lambda}}\right]\delta\boldsymbol{\lambda} + \left[\frac{\partial\mathcal{H}}{\partial\mathbf{u}}\right]\delta\mathbf{u}$$

And, via Integration by Parts, We Have

$$\int_{t_0}^{t_f} \boldsymbol{\lambda}^{\mathsf{T}} \delta\dot{\mathbf{x}} dt = \left[\boldsymbol{\lambda}^{\mathsf{T}} \delta\mathbf{x}\right]_{t_0}^{t_f} - \int_{t_0}^{t_f} \dot{\boldsymbol{\lambda}}^{\mathsf{T}} \delta\mathbf{x} dt$$

- **Important Note:**
  - ▷ Terms Outside Integral are *evaluated* at t_0 and t_f
  - ▷ Consequently, We Have $\delta\mathbf{x}(t_0)$ and $\delta\mathbf{x}(t_f)$
- Therefore,

$$\begin{aligned}
\delta \int_{t_0}^{t_f} \left[\mathcal{H} - \boldsymbol{\lambda}^{\mathsf{T}}\dot{\mathbf{x}}\right] dt &= \boldsymbol{\lambda}^{\mathsf{T}}(t_0)\delta\mathbf{x}(t_0) - \boldsymbol{\lambda}^{\mathsf{T}}(t_f)\delta\mathbf{x}(t_f) \\
&+ \left[\mathcal{H}(t_f) - \boldsymbol{\lambda}^{\mathsf{T}}(t_f)\dot{\mathbf{x}}(t_f)\right]\delta t_f - \left[\mathcal{H}(t_0) - \boldsymbol{\lambda}^{\mathsf{T}}(t_0)\dot{\mathbf{x}}(t_0)\right]\delta t_0 \\
&+ \int_{t_0}^{t_f} \left\{\left[\frac{\partial\mathcal{H}}{\partial\mathbf{x}}\right]\delta\mathbf{x} + \left[\frac{\partial\mathcal{H}}{\partial\boldsymbol{\lambda}}\right]\delta\boldsymbol{\lambda} + \left[\frac{\partial\mathcal{H}}{\partial\mathbf{u}}\right]\delta\mathbf{u}\right\} dt \\
&- \int_{t_0}^{t_f} \left\{\delta\boldsymbol{\lambda}^{\mathsf{T}}\dot{\mathbf{x}} + \dot{\boldsymbol{\lambda}}^{\mathsf{T}}\delta\mathbf{x}\right\} dt
\end{aligned}$$

- Using the Earlier Results for $\delta\mathbf{x}(t_0)$ and $\delta\mathbf{x}(t_f)$,

$$
\begin{aligned}
&\delta\int_{t_0}^{t_f}\left[\mathcal{H}-\boldsymbol{\lambda}^{\mathsf{T}}\dot{\mathbf{x}}\right]dt \\
&=\boldsymbol{\lambda}^{\mathsf{T}}(t_0)\left[\delta\mathbf{x}_0-\dot{\mathbf{x}}(t_0)\delta t_0\right]-\boldsymbol{\lambda}^{\mathsf{T}}(t_f)\left[\delta\mathbf{x}_f-\dot{\mathbf{x}}(t_f)\delta t_f\right] \\
&+\left[\mathcal{H}(t_f)-\boldsymbol{\lambda}^{\mathsf{T}}(t_f)\dot{\mathbf{x}}(t_f)\right]\delta t_f-\left[\mathcal{H}(t_0)-\boldsymbol{\lambda}^{\mathsf{T}}(t_0)\dot{\mathbf{x}}(t_0)\right]\delta t_0 \\
&+\int_{t_0}^{t_f}\left\{\left[\frac{\partial\mathcal{H}}{\partial\mathbf{x}}\right]\delta\mathbf{x}+\left[\frac{\partial\mathcal{H}}{\partial\boldsymbol{\lambda}}\right]\delta\boldsymbol{\lambda}+\left[\frac{\partial\mathcal{H}}{\partial\mathbf{u}}\right]\delta\mathbf{u}\right\}dt \\
&-\int_{t_0}^{t_f}\left\{\delta\boldsymbol{\lambda}^{\mathsf{T}}\dot{\mathbf{x}}+\dot{\boldsymbol{\lambda}}^{\mathsf{T}}\delta\mathbf{x}\right\}dt
\end{aligned}
$$

- Rearranging Gives

$$
\boxed{
\begin{aligned}
&\delta\int_{t_0}^{t_f}\left[\mathcal{H}-\boldsymbol{\lambda}^{\mathsf{T}}\dot{\mathbf{x}}\right]dt=\boldsymbol{\lambda}^{\mathsf{T}}(t_0)\delta\mathbf{x}_0-\boldsymbol{\lambda}^{\mathsf{T}}(t_f)\delta\mathbf{x}_f \\
&\quad+\mathcal{H}(t_f)\delta t_f-\mathcal{H}(t_0)\delta t_0 \\
&\quad+\int_{t_0}^{t_f}\left[\left(\frac{\partial\mathcal{H}}{\partial\mathbf{x}}+\dot{\boldsymbol{\lambda}}^{\mathsf{T}}\right)\delta\mathbf{x}+\left(\frac{\partial\mathcal{H}}{\partial\boldsymbol{\lambda}}-\dot{\mathbf{x}}^{\mathsf{T}}\right)\delta\boldsymbol{\lambda}\right. \\
&\qquad\qquad\left.+\frac{\partial\mathcal{H}}{\partial\mathbf{u}}\delta\mathbf{u}\right]dt
\end{aligned}
}
$$

- Terms Outside Integral
  - ▷ Terms involving $\delta\boldsymbol{\nu}$:

$$\boxed{\delta\boldsymbol{\nu} = \mathbf{0}} \text{ or } \boxed{\boldsymbol{\phi}(\mathbf{x}(t_0), t_0, \mathbf{x}(t_f), t_f) = \mathbf{0}}$$

  - ▷ Terms involving $\delta\mathbf{x}_0$ and $\delta\mathbf{x}_f$:

$$\boxed{\delta\mathbf{x}_0 = \mathbf{0}} \text{ or } \boxed{\boldsymbol{\lambda}(t_0) = -\left[\frac{\partial\Phi}{\partial\mathbf{x}(t_0)}\right]^{\mathsf{T}} + \left[\frac{\partial\boldsymbol{\phi}}{\partial\mathbf{x}(t_0)}\right]^{\mathsf{T}}\boldsymbol{\nu}}$$

$$\boxed{\delta\mathbf{x}_f = \mathbf{0}} \text{ or } \boxed{\boldsymbol{\lambda}(t_f) = \left[\frac{\partial\Phi}{\partial\mathbf{x}(t_f)}\right]^{\mathsf{T}} - \left[\frac{\partial\boldsymbol{\phi}}{\partial\mathbf{x}(t_f)}\right]^{\mathsf{T}}\boldsymbol{\nu}}$$

  - ▷ Terms involving δt_0 and δt_f:

$$\boxed{\delta t_0 = 0} \text{ or } \boxed{\mathcal{H}(t_0) = \frac{\partial\Phi}{\partial t_0} - \boldsymbol{\nu}^{\mathsf{T}}\frac{\partial\boldsymbol{\phi}}{\partial t_0}}$$

$$\boxed{\delta t_f = 0} \text{ or } \boxed{\mathcal{H}(t_f) = -\frac{\partial\Phi}{\partial t_f} + \boldsymbol{\nu}^{\mathsf{T}}\frac{\partial\boldsymbol{\phi}}{\partial t_f}}$$

- Terms Inside Integral (i.e., $\delta\mathbf{x}$, $\delta\mathbf{u}$, and $\delta\boldsymbol{\lambda}$):

$$\begin{aligned}\frac{\partial \mathcal{H}}{\partial \mathbf{x}} + \dot{\boldsymbol{\lambda}}^{\mathsf{T}} &= \mathbf{0} \\ \frac{\partial \mathcal{H}}{\partial \boldsymbol{\lambda}} - \dot{\mathbf{x}}^{\mathsf{T}} &= \mathbf{0} \\ \frac{\partial \mathcal{H}}{\partial \mathbf{u}} &= \mathbf{0}\end{aligned}$$

- Rearranging These Last Three Equations, We Obtain

$$\boxed{\dot{\mathbf{x}} = \left[\frac{\partial \mathcal{H}}{\partial \boldsymbol{\lambda}}\right]^{\mathsf{T}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t)} \iff \textbf{State Dynamics}$$

$$\boxed{\dot{\boldsymbol{\lambda}} = -\left[\frac{\partial \mathcal{H}}{\partial \mathbf{x}}\right]^{\mathsf{T}}} \iff \textbf{Costate Dynamics}$$

$$\boxed{\left[\frac{\partial \mathcal{H}}{\partial \mathbf{u}}\right]^{\mathsf{T}} = \mathbf{0}} \iff \textbf{Condition for Optimal Control}$$

Necessary Conditions and Extremals

- Double-Boxed Terms Are *Necessary Conditions*
- Let $(\mathbf{x}^*(t), \boldsymbol{\lambda}^*(t), \mathbf{u}^*(t))$ Satisfy Necessary Conditions
  - $(\mathbf{x}^*(t), \boldsymbol{\lambda}^*(t))$ is an *extremal trajectory*
  - $\mathbf{u}^*(t)$ is an *extremal control*
- Important Notes About Extremal Solutions
  - Only *candidate* optimal solutions
  - Each Extremal May Be Either
    - Minimum
    - Maximum
    - Saddle

Understanding Structure of Optimality Conditions

- Dynamics of Augmented System

$$\begin{aligned}\dot{\mathbf{x}} &= \mathcal{H}_{\boldsymbol{\lambda}}(\mathbf{x},\boldsymbol{\lambda}) \\ \dot{\boldsymbol{\lambda}} &= -\mathcal{H}_{\mathbf{x}}(\mathbf{x},\boldsymbol{\lambda})\end{aligned}$$

- Vector Field $(\mathcal{H}_{\boldsymbol{\lambda}}, -\mathcal{H}_{\mathbf{x}}) \in \mathbb{R}^{2n}$ is a *Hamiltonian System*
- Time Derivative of Hamiltonian

$$\begin{aligned}\frac{d\mathcal{H}}{dt} &= \frac{\partial \mathcal{H}}{\partial t} + \left[\frac{\partial \mathcal{H}}{\partial \mathbf{x}}\right]^{\mathsf{T}} \frac{d\mathbf{x}}{dt} + \left[\frac{\partial \mathcal{H}}{\partial \boldsymbol{\lambda}}\right]^{\mathsf{T}} \frac{d\boldsymbol{\lambda}}{dt} \\ &= \frac{\partial \mathcal{H}}{\partial t} + \left[\frac{\partial \mathcal{H}}{\partial \mathbf{x}}\right]^{\mathsf{T}} \left[\frac{\partial \mathcal{H}}{\partial \boldsymbol{\lambda}}\right] - \left[\frac{\partial \mathcal{H}}{\partial \boldsymbol{\lambda}}\right]^{\mathsf{T}} \left[\frac{\partial \mathcal{H}}{\partial \mathbf{x}}\right] \\ &= \frac{\partial \mathcal{H}}{\partial t} + \mathcal{H}_{\mathbf{x}}^{\mathsf{T}} \mathcal{H}_{\boldsymbol{\lambda}} - \mathcal{H}_{\boldsymbol{\lambda}}^{\mathsf{T}} \mathcal{H}_{\mathbf{x}}\end{aligned}$$

  But

$$\mathcal{H}_{\mathbf{x}}^{\mathsf{T}} \mathcal{H}_{\boldsymbol{\lambda}} - \mathcal{H}_{\boldsymbol{\lambda}}^{\mathsf{T}} \mathcal{H}_{\mathbf{x}} = 0$$

  Therefore,

$$\frac{d\mathcal{H}}{dt} = \frac{\partial \mathcal{H}}{\partial t}$$

- Important Properties of Hamiltonian
  - ▷ Total Time Derivative = Partial Derivative With Respect to Time
  - ▷ If Hamiltonian not an *explicit* function of time

$$\Longrightarrow \frac{\partial \mathcal{H}}{\partial t} = 0 \Rightarrow \frac{d\mathcal{H}}{dt} = 0 \Rightarrow \mathcal{H} = \textbf{constant}$$

  - ▷ If Hamiltonian not an *explicit* function of time **and** t_f is *free*

$$\mathcal{H}(t_f) = -\frac{\partial \Phi}{\partial t_f} + \boldsymbol{\nu}^{\mathsf{T}} \frac{\partial \boldsymbol{\phi}}{\partial t_f} = 0 \Rightarrow \mathcal{H} = 0$$

Example

- Cost Functional

$$J = \frac{1}{2}\int_0^{t_f} (qx^2 + ru^2)dt$$

where $x(t) \in \mathbb{R}$, $u \in \mathbb{R}$ and $q > 0$ and $r > 0$ are constants

- Dynamic Constraints

$$\dot{x} = ax + bu$$

where a and b are constants

- Boundary Conditions

$$\begin{aligned} x(0) &= x_0 \\ x(t_f) &= x_f \end{aligned}$$

where x_0 and x_f are *given* values

Solution to Example

- Form the Hamiltonian

$$\mathcal{H} = \mathcal{L} + \lambda f = \frac{1}{2}qx^2 + \frac{1}{2}ru^2 + \lambda(ax + bu)$$

- State and Costate Dynamics

$$\begin{aligned} \dot{x} &= \mathcal{H}_\lambda = ax + bu \\ \dot{\lambda} &= -\mathcal{H}_x = -qx - a\lambda \end{aligned}$$

- Optimal Control

$$\mathcal{H}_u = ru + \lambda b = 0 \Rightarrow u^* = -\frac{b}{r}\lambda$$

- Substitution of Optimal Control into State Dynamics

$$\dot{x} = ax - \frac{b^2}{r}\lambda$$

- State-Costate Dynamics Including Condition for Optimal Control

$$\begin{bmatrix} \dot{x} \\ \dot{\lambda} \end{bmatrix} = \begin{bmatrix} a & -\frac{b^2}{r} \\ -q & -a \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix}$$

- Transversality Conditions

$$\boldsymbol{\phi} = \begin{bmatrix} x(0) - x_0 \\ x(t_f) - x_f \end{bmatrix}$$

$$\nu = \begin{bmatrix} \nu_1 \\ \nu_2 \end{bmatrix}$$

Therefore,

$$\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{x}(0)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{x}(t_f)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

We Then Obtain

$$\begin{aligned} \lambda(0) &= \nu_1 = \text{Unknown} \\ \lambda(t_f) &= \nu_2 = \text{Unknown} \end{aligned}$$

- Note: Neither $(\lambda(0), \lambda(t_f))$ *nor* (ν_1, ν_2) is Known
  $\Longrightarrow$ Determining $\lambda \Longleftrightarrow$ Determining ν

Hamiltonian Boundary-Value Problem for Example

- State-Costate Dynamics and Boundary Conditions

$$\begin{bmatrix} \dot{x} \\ \dot{\lambda} \end{bmatrix} = \begin{bmatrix} a & -\frac{b^2}{r} \\ -q & -a \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} \quad , \quad \begin{bmatrix} x(0) = x_0 \\ x(t_f) = x_f \end{bmatrix}$$

- $\lambda(0)$ and $\lambda(t_f)$ are *unknown*
- General Form of Dynamics: Linear Time-Invariant (LTI) System

$$\dot{\mathbf{p}} = \mathbf{A}\mathbf{p}$$

- General Solution of First-Order LTI System

$$\mathbf{p}(t) = c_1 e^{\mu_1 t}\mathbf{w}_1 + c_2 e^{\mu_2 t}\mathbf{w}_2$$

where

$$\begin{aligned} \mathbf{p} &= \begin{bmatrix} x \\ \lambda \end{bmatrix} \\ \mu_1, \mu_2 &= \text{Eigenvalues of } \mathbf{A} \\ \mathbf{w}_1, \mathbf{w}_2 &= \text{Eigenvectors of } \mathbf{A} \end{aligned}$$

Determination of Solution to HBVP

- Eigenvalues of System Matrix $\mathbf{A}$

$$\det\left[\mu\mathbf{I}-\mathbf{A}\right]=\det\begin{bmatrix}\mu-a & \frac{b^2}{r}\\ q & \mu+a\end{bmatrix}=\mu^2-a^2-\frac{qb^2}{r}=0$$

- Assuming That $q>0$ and $r>0$, We Obtain

$$\mu=\pm\sqrt{a^2+\frac{qb^2}{r}}$$

- Note: Eigenvalues Have Same Magnitude But Opposite Sign
- Let $\omega=\sqrt{a^2+\frac{qb^2}{r}}\Longrightarrow\mu=\pm\omega$
- Eigenvectors of $\mathbf{A}$
  - ▷ Eigenvector for $\mu=\omega$

$$\begin{bmatrix}\omega-a & \frac{b^2}{r}\\ q & a\omega+a\end{bmatrix}\begin{bmatrix}w_{11}\\ w_{12}\end{bmatrix}=\begin{bmatrix}0\\ 0\end{bmatrix}$$

From first equation we have

$$(\omega - a)w_{11} + \frac{b^2}{r}w_{12} = 0$$

Let $w_{11} = -1/(\omega - a)$. We then have

$$w_{12} = \frac{r}{b^2}$$

This gives

$$\mathbf{w}_1 = \begin{bmatrix} -1/(\omega - a) \\ r/b^2 \end{bmatrix}$$

- Eigenvector for $\mu = -\omega$

$$\begin{bmatrix} -\omega - a & \frac{b^2}{r} \\ q & -\omega + a \end{bmatrix} \begin{bmatrix} w_{21} \\ w_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

From the first equation we have

$$-(\omega + a)w_{21} + \frac{b^2}{r}w_{22} = 0$$

Let $w_{21} = 1/(\omega + a)$. We then obtain

$$w_{22} = \frac{r}{b^2}$$

Which implies that

$$\mathbf{w}_2 = \begin{bmatrix} 1/(\omega + a) \\ r/b^2 \end{bmatrix}$$

- Solving for Constants c_1 and c_2
  - ▷ At $t = 0$ we have

$$x(0) = x_0 = c_1 w_{11} + c_2 w_{21} = -\frac{c_1}{\omega - a} + \frac{c_2}{\omega + a}$$

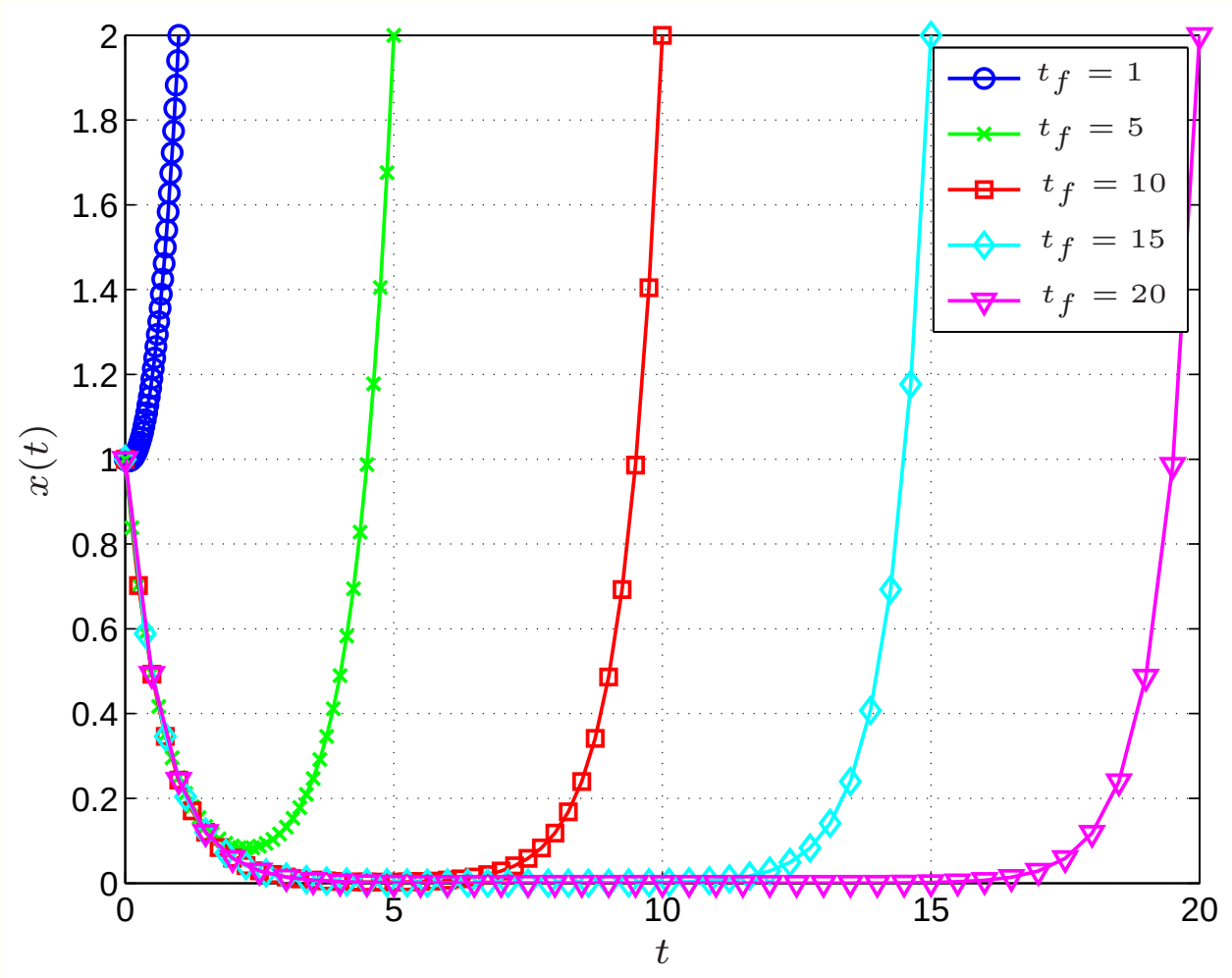
  - ▷ At $t = t_f$ we have

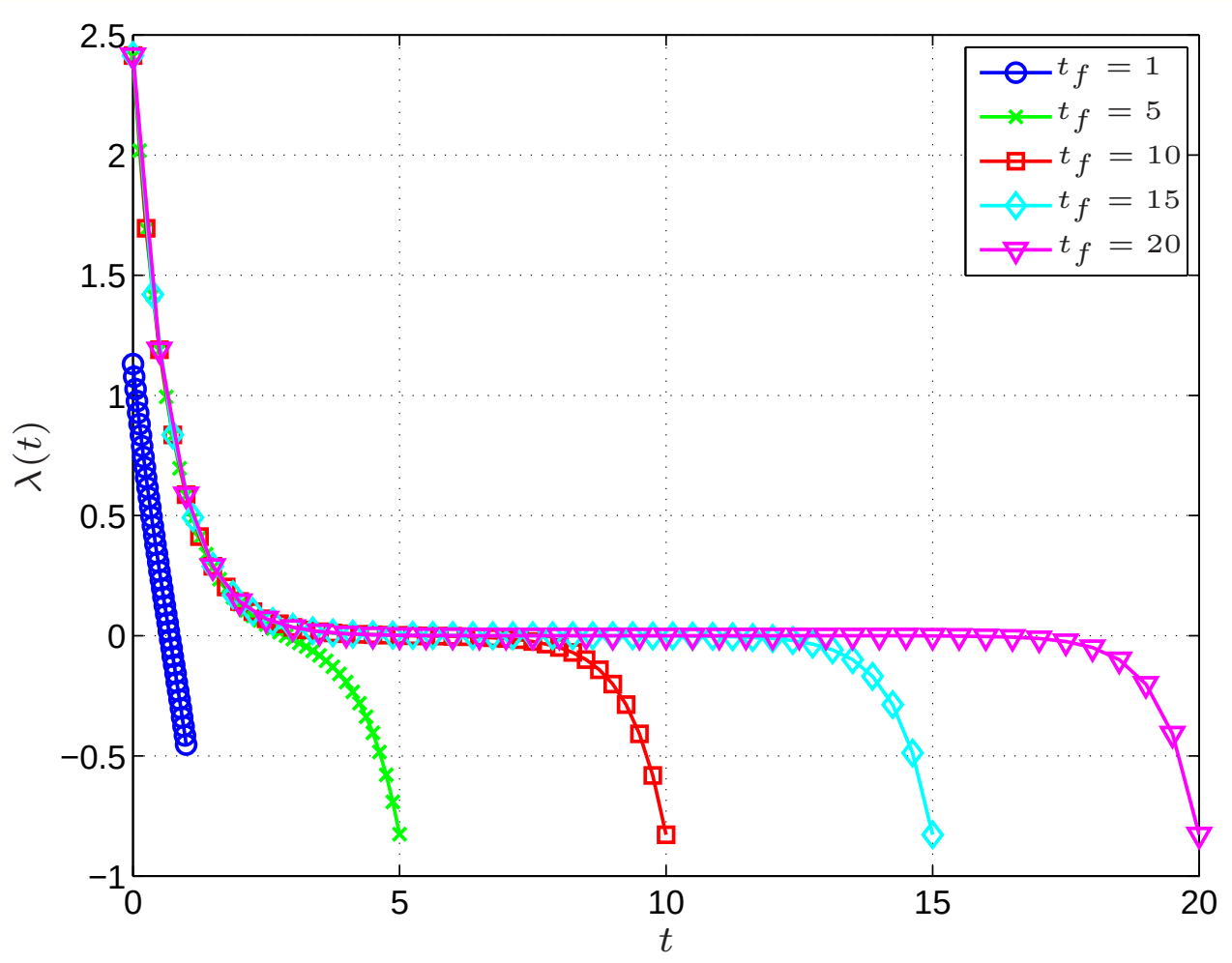
$$x(t_f) = x_f = c_1 e^{\omega t_f} w_{11} + c_2 e^{-\omega t_f} w_{21} = -\frac{c_1 e^{\omega t_f}}{\omega - a} + \frac{c_2 e^{-\omega t_f}}{\omega + a}$$

  - ▷ Solving last two equations for c_1 and c_2 gives

$$\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \frac{1}{e^{\omega t_f} - e^{-\omega t_f}} \begin{bmatrix} e^{-\omega t_f}(\omega - a) & -(\omega - a) \\ e^{\omega t_f}(\omega + a) & -(\omega + a) \end{bmatrix} \begin{bmatrix} x_0 \\ x_f \end{bmatrix}$$

- Solution for $a = b = q = r = 1$, $t_0 = 0$, $x(t_0) = 1$, $t_f = (1, 5, 10, 15, 20)$, and $x(t_f) = 2$

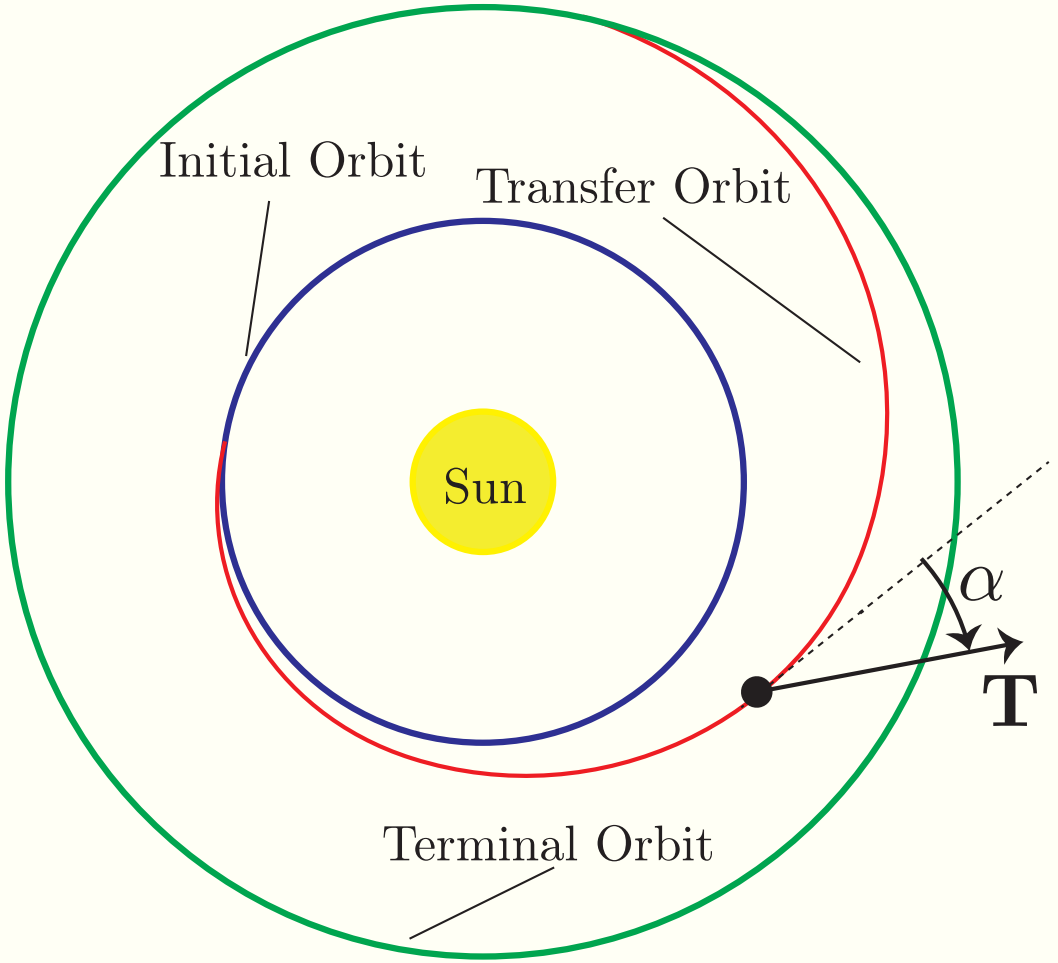




- Observations of Solution Structure
  - ▷ For small t_f, solution
    - ○ Exhibits no decay
    - ○ Moves essentially directly from initial to terminal condition

- For large t_f, solution
  - Appears to have a three-segment structure
  - First segment: decay from initial condition to zero
  - Second segment: remains very close to zero
  - Third segment: rapid growth to meet terminal condition
- Key points
  - **Both** state and costate have decaying and growing behavior
  - Growth and decay show up as t_f increases

Example: Orbit-Raising Problem



Problem: determine the thrust direction as a function of time that transfers a spacecraft from Earth orbit to Mars orbit in minimum time (Reference: Bryson and Ho, *Applied Optimal Control*, Hemisphere Publishing, New York, 1975).

- Dynamics in First-Order Form

$$\begin{aligned}
\dot{r} &= u \\
\dot{u} &= \frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T}{m}\sin\alpha \\
\dot{v} &= -\frac{uv}{r} + \frac{T}{m}\cos\alpha \\
\dot{m} &= -\frac{T}{g_0 I_{sp}}
\end{aligned}$$

- Astronomical System of Units (AU) Chosen Such That $\mu = 1$ and Distance of Sun to Earth is Unity
- Boundary Conditions
  - ▷ Initial conditions: circular orbit of size 1 AU (i.e., Earth orbit)

$$\begin{aligned}
r(0) &= r_0 = 1 \text{ AU} \\
u(0) &= 0 \\
v(0) &= \sqrt{\mu/r_0} = 1 \text{ AU/TU} \\
m(0) &= 1
\end{aligned}$$

- ▷ Terminal conditions: circular orbit of size 1.5 AU (i.e., Mars orbit)

$$\begin{aligned} r(t_f) &= r_f = 1.5 \text{ AU} \\ u(t_f) &= 0 \\ v(t_f) &= \sqrt{\mu/r_f} \approx 0.816496581 \text{ AU/TU} \end{aligned}$$

- ▷ Objective Functional: Minimize Time

$$J = \overset{\text{Mayer Form}}{\overset{\Downarrow}{t_f}} = \overset{\text{Lagrange Form}}{\overset{\Downarrow}{\int_0^{t_f}}} dt$$

- ▷ We will use Mayer form of cost functional

- Hamiltonian (Observe that $\mathcal{L} \equiv 0$ and $\Phi = t_f$)

$$\begin{aligned} \mathcal{H} &= \mathcal{L} + \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{f} = \lambda_r \dot{r} + \lambda_u \dot{u} + \lambda_v \dot{v} + \lambda_m \dot{m} \\ &= \lambda_r u + \lambda_u \left[\frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T}{m} \sin\alpha \right] + \lambda_v \left[-\frac{uv}{r} + \frac{T}{m} \cos\alpha \right] \\ &+ \lambda_m \left[-\frac{T}{g_0 I_{sp}} \right] \end{aligned}$$

- Costate Equations

$$\begin{aligned}
\dot{\lambda}_r &= -\frac{\partial \mathcal{H}}{\partial r} = \lambda_u \left[\frac{v^2}{r^2} - \frac{2\mu}{r^3}\right] - \lambda_v \frac{uv}{r^2} \\
\dot{\lambda}_u &= -\frac{\partial \mathcal{H}}{\partial u} = -\lambda_r + \lambda_v \frac{v}{r} \\
\dot{\lambda}_v &= -\frac{\partial \mathcal{H}}{\partial v} = -\lambda_u \frac{2v}{r} + \lambda_v \frac{u}{r} \\
\dot{\lambda}_m &= -\frac{\partial \mathcal{H}}{\partial m} = \lambda_u \left[\frac{T}{m^2} \sin\alpha\right] + \lambda_v \left[\frac{T}{m^2} \cos\alpha\right]
\end{aligned}$$

- Condition for Optimal Control

$$\frac{\partial \mathcal{H}}{\partial \alpha} = 0 \Longrightarrow \lambda_u \frac{T}{m} \cos\alpha - \lambda_v \frac{T}{m} \sin\alpha = 0$$

  Solving for α Gives

$$\alpha^* = \tan^{-1}(\lambda_u, \lambda_v)$$

  Note: A *four-quadrant* inverse tangent is used because $\alpha \in [-\pi, \pi]$

- Transversality Conditions

▷ The boundary conditions can be written in the general form

$$\boldsymbol{\phi}(\mathbf{x}(t_0), t_0, \mathbf{x}(t_f), t_f) = \begin{bmatrix} r(t_0) - r_0 \\ u(t_0) \\ v(t_0) - \sqrt{\mu/r_0} \\ m(t_0) - m_0 \\ r(t_f) - r_f \\ u(t_f) \\ v(t_f) - \sqrt{\mu/r_f} \end{bmatrix} = \mathbf{0}$$

▷ Let $\boldsymbol{\nu}^{\mathsf{T}} = (\nu_{r0}, \nu_{u0}, \nu_{v0}, \nu_{m0}, \nu_{rf}, \nu_{uf}, \nu_{vf})$

$$\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{x}(t_0)} = \left[\begin{array}{cccc} \frac{\partial \boldsymbol{\phi}}{\partial r(t_0)} & \frac{\partial \boldsymbol{\phi}}{\partial u(t_0)} & \frac{\partial \boldsymbol{\phi}}{\partial v(t_0)} & \frac{\partial \boldsymbol{\phi}}{\partial m(t_0)} \end{array} \right]$$

$$= \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

$$\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{x}(t_f)} = \left[\begin{array}{cccc} \frac{\partial \boldsymbol{\phi}}{\partial r(t_f)} & \frac{\partial \boldsymbol{\phi}}{\partial u(t_f)} & \frac{\partial \boldsymbol{\phi}}{\partial v(t_f)} & \frac{\partial \boldsymbol{\phi}}{\partial m(t_f)} \end{array} \right]$$

$$= \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array} \right]$$

- Initial transversality conditions

$$\boldsymbol{\lambda}(0) = -\frac{\partial \Phi}{\partial \mathbf{x}(t_0)} + \left[\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{x}(t_0)} \right]^{\mathsf{T}} \boldsymbol{\nu} \Rightarrow \left[\begin{array}{c} \lambda_r(0) \\ \lambda_u(0) \\ \lambda_v(0) \\ \lambda_m(0) \end{array} \right] = \left[\begin{array}{c} \nu_{r0} \\ \nu_{u0} \\ \nu_{v0} \\ \nu_{m0} \end{array} \right]$$

- ▷ Terminal transversality conditions

$$\boldsymbol{\lambda}(t_f) = \frac{\partial \Phi}{\partial \mathbf{x}(t_0)} - \left[\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{x}(t_f)}\right]^{\mathsf{T}} \boldsymbol{\nu} \Rightarrow \begin{bmatrix} \lambda_r(t_f) \\ \lambda_u(t_f) \\ \lambda_v(t_f) \\ \lambda_m(t_f) \end{bmatrix} = \begin{bmatrix} -\nu_{rf} \\ -\nu_{uf} \\ -\nu_{vf} \\ 0 \end{bmatrix}$$

- Notes
  - ▷ Only *one* of the boundary costates is known
  - ▷ Seven unknown boundary conditions
- One Last Important Point: $\partial \mathcal{H}/\partial \alpha = 0$ is Only a *Necessary Condition*
  - ▷ Sufficient condition: $\partial^2 \mathcal{H}/\partial \alpha^2 > 0$

$$\frac{\partial^2 \mathcal{H}}{\partial \alpha^2} = -\lambda_u \frac{T}{m} \sin\alpha - \lambda_v \frac{T}{m} \cos\alpha > 0$$

$$\Longrightarrow -\lambda_u \sin\alpha - \lambda_v \cos\alpha > 0 \Longrightarrow \tan\alpha > -\frac{\lambda_v}{\lambda_u}$$

  - ▷ Therefore,

$$\frac{\lambda_u}{\lambda_v} > -\frac{\lambda_v}{\lambda_u} \Longrightarrow \lambda_u^2 + \lambda_v^2 > 0$$

Hamiltonian Boundary-Value Problem for Orbit-Transfer Problem

- Dynamics ($\alpha^* = \tan^{-1}(\lambda_u, \lambda_v)$):

$$\begin{aligned}
\dot{r} &= u & , \quad \dot{u} &= \frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T}{m}\sin\alpha^* \\
\dot{v} &= -\frac{uv}{r} + \frac{T}{m}\cos\alpha^* & , \quad \dot{m} &= -\frac{T}{g_0 I_{sp}} \\
\dot{\lambda}_r &= \lambda_u\left[\frac{v^2}{r^2} - \frac{2\mu}{r^3}\right] - \lambda_v\frac{uv}{r^2} & , \quad \dot{\lambda}_u &= -\lambda_r + \lambda_v\frac{v}{r} \\
\dot{\lambda}_v &= -\lambda_u\frac{2v}{r} + \lambda_v\frac{u}{r} & , \quad \dot{\lambda}_m &= \lambda_u\left[\frac{T}{m^2}\sin\alpha^*\right] + \lambda_v\left[\frac{T}{m^2}\cos\alpha^*\right]
\end{aligned}$$

- Boundary Conditions:

$$\begin{aligned}
r(t_0) &= r_0 \quad , \quad u(t_0) = 0 \quad , \quad v(t_0) = \sqrt{\mu/r_0} \quad , \quad m(t_0) = m_0, \\
r(t_f) &= r_f \quad , \quad u(t_f) = 0 \quad , \quad v(t_f) = \sqrt{\mu/r_f} \quad , \quad \lambda_m(t_f) = 0
\end{aligned}$$

- Notes:
  - ▷ Initial and Terminal Conditions are *Incomplete*
  - ▷ Half of the initial conditions and terminal conditions are specified

The Minimum Principle of Pontryagin

- In General, the Optimal Control Satisfies the Condition

$$J(\mathbf{u}) - J(\mathbf{u}^*) \geq 0$$

  for all $\mathbf{u}$ sufficiently close to $\mathbf{u}^*$

- Therefore $J(\mathbf{u}) = J(\mathbf{u}^*) + \delta J(\mathbf{u}^*, \delta\mathbf{u})$
- Then, from Variation of Augmented Cost Functional

$$\delta J(\mathbf{u}^*, \delta\mathbf{u}) = \int_{t_0}^{t_f} \left[\frac{\partial \mathcal{H}}{\partial \mathbf{u}}\right]_{\mathbf{u}^*} \delta\mathbf{u} dt$$

- Term in Integrand Given As

$$\left[\frac{\partial \mathcal{H}}{\partial \mathbf{u}}\right]_{\mathbf{u}^*} \delta\mathbf{u} = \mathcal{H}(\mathbf{x}^*, \mathbf{u}^* + \delta\mathbf{u}, \boldsymbol{\lambda}^*) - \mathcal{H}(\mathbf{x}^*, \mathbf{u}^*, \boldsymbol{\lambda}^*) + \text{higher order terms}$$

- Consequently, to First Order

$$\delta J(\mathbf{u}^*, \delta\mathbf{u}) = \int_{t_0}^{t_f} \left[\mathcal{H}(\mathbf{x}^*, \mathbf{u}^* + \delta\mathbf{u}, \boldsymbol{\lambda}^*) - \mathcal{H}(\mathbf{x}^*, \mathbf{u}^*, \boldsymbol{\lambda}^*)\right] dt \geq 0$$

- Because $\delta J(\mathbf{u}^*, \delta\mathbf{u}) >$ for *All* Admissible Variations in $\mathbf{u}$,

$$\mathcal{H}(\mathbf{x}^*, \mathbf{u}^*, \boldsymbol{\lambda}^*) \leq \mathcal{H}(\mathbf{x}^*, \mathbf{u}^* + \delta\mathbf{u}, \boldsymbol{\lambda}^*)$$

- Leads to Following Condition for Optimal Control

$$\mathbf{u}^* = \arg\min_{\mathbf{u}\in\mathcal{U}} \mathcal{H}(\mathbf{x}^*, \mathbf{u}, \boldsymbol{\lambda}^*), \quad \mathcal{U} = \text{Admissible Control Set}$$

- Possibilities for Determining Optimal Control
  - ▷ Control Lies *Interior* to $\mathcal{U}$

$$\frac{\partial\mathcal{H}}{\partial\mathbf{u}} = \mathbf{0} \Longrightarrow \text{Strong Form of Minimum Principle}$$

  - ▷ Control Lies on *Boundary of* $\mathcal{U}$
    - ○ *One-sided* differential $\rightarrow$ must use weak form of minimum principle

$$\mathbf{u}^* = \arg\min_{\mathbf{u}\in\mathcal{U}} \mathcal{H}(\mathbf{x}^*, \mathbf{u}, \boldsymbol{\lambda}^*) \Longleftrightarrow \text{Weak Form of Minimum Principle}$$

  - ▷ Example: Hamiltonian *linear in control*

$$\mathcal{H} = \mathcal{L}(\mathbf{x}) + \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{f}(\mathbf{x}) + \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{u}$$

    - ○ $\partial\mathcal{H}/\partial\mathbf{u}$ provides no information about $\mathbf{u}$
    - ○ Must use *weak* form of minimum principle

Example: Minimum-Time Double Integrator

- Consider the Following Optimal Control Problem. Minimize

$$J = t_f = \int_0^{t_f} dt$$

Subject to

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= u \end{aligned}$$

with

$$\begin{aligned} x_1(0) &= x_{10} \quad , \quad x_1(t_f) = 0 \\ x_2(0) &= x_{20} \quad , \quad x_2(t_f) = 0 \end{aligned}$$

and

$$|u| \leq 1 \Longleftrightarrow -1 \leq u \leq 1$$

- Problem in Words

Determine the optimal trajectory and control that steers the system from an initial nonzero state to the origin in minimum time.

Solution of Double Integrator Minimum Time Example

- Hamiltonian (Using Lagrange Form of Problem)

$$\mathcal{H} = 1 + \lambda_1 x_2 + \lambda_2 u$$

- Optimal Control
  - ▷ $\partial\mathcal{H}/\partial u$ gives no information
  - ▷ Need to apply general form of minimum principle

$$u^* = \arg \min_{u \in [-1,1]} \mathcal{H} = -\text{sgn } \lambda_2 = \pm 1$$

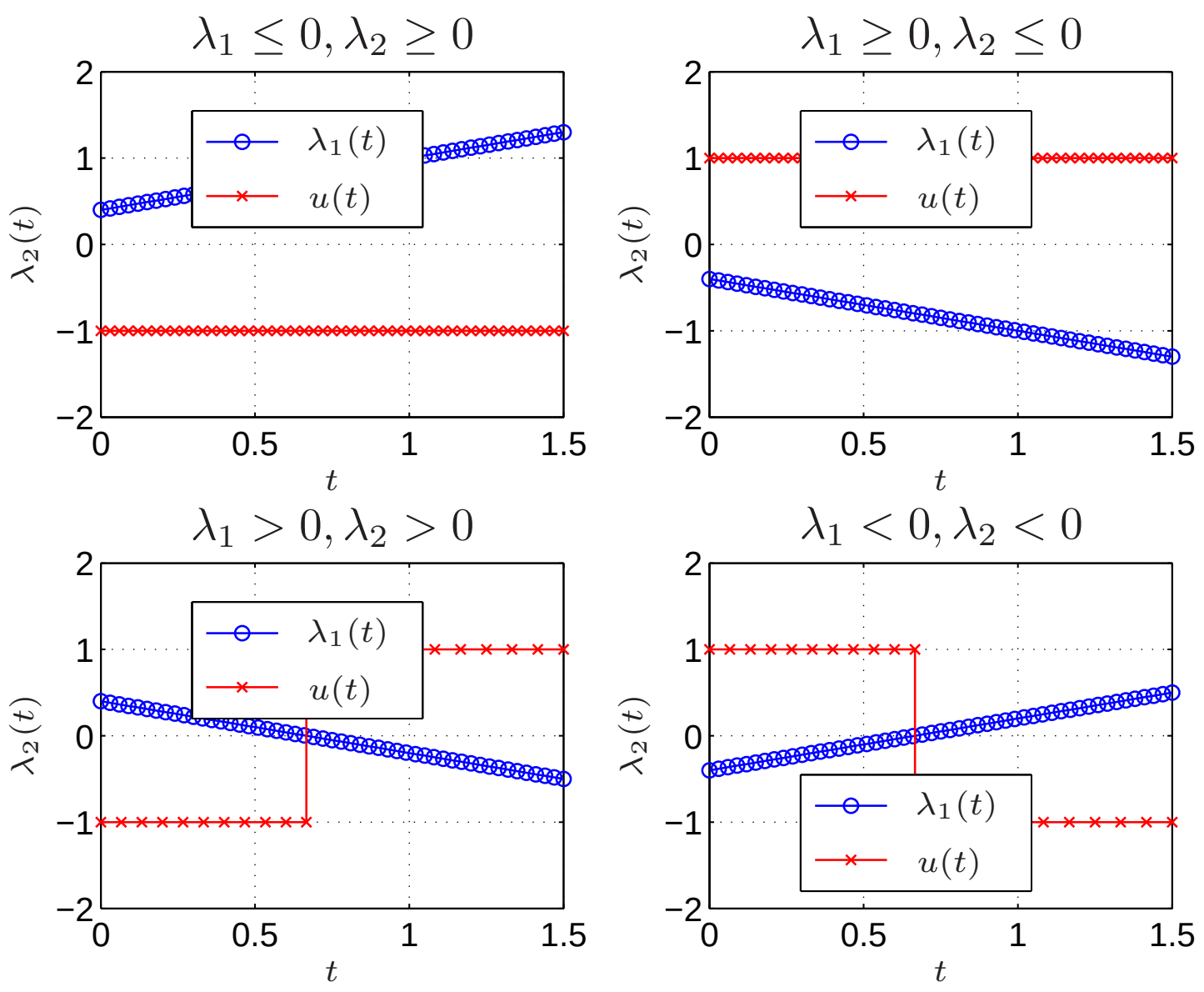
  where $\text{sgn}(\cdot)$ is the *sign* or *signum* function

- Costate Equations

$$\begin{aligned} \dot{\lambda}_1 &= -\partial\mathcal{H}/\partial x_1 = 0 \quad &\Longrightarrow \lambda_1(t) = \text{constant} = \lambda_{10} \\ \dot{\lambda}_2 &= -\partial\mathcal{H}/\partial x_2 = -\lambda_1 \quad &\Longrightarrow \lambda_2(t) = -\lambda_{10} t + \lambda_{20} \end{aligned}$$

- Observation: $\lambda_2(t)$ is a *Straight Line*
- Four Possibilities for $\lambda_2(t)$ (Depending on Sign of λ_{10})

- $\lambda_2(t)$ is always positive
- $\lambda_2(t)$ is always negative
- $\lambda_2(t)$ switches from positive to negative
- $\lambda_2(t)$ switches from negative to positive



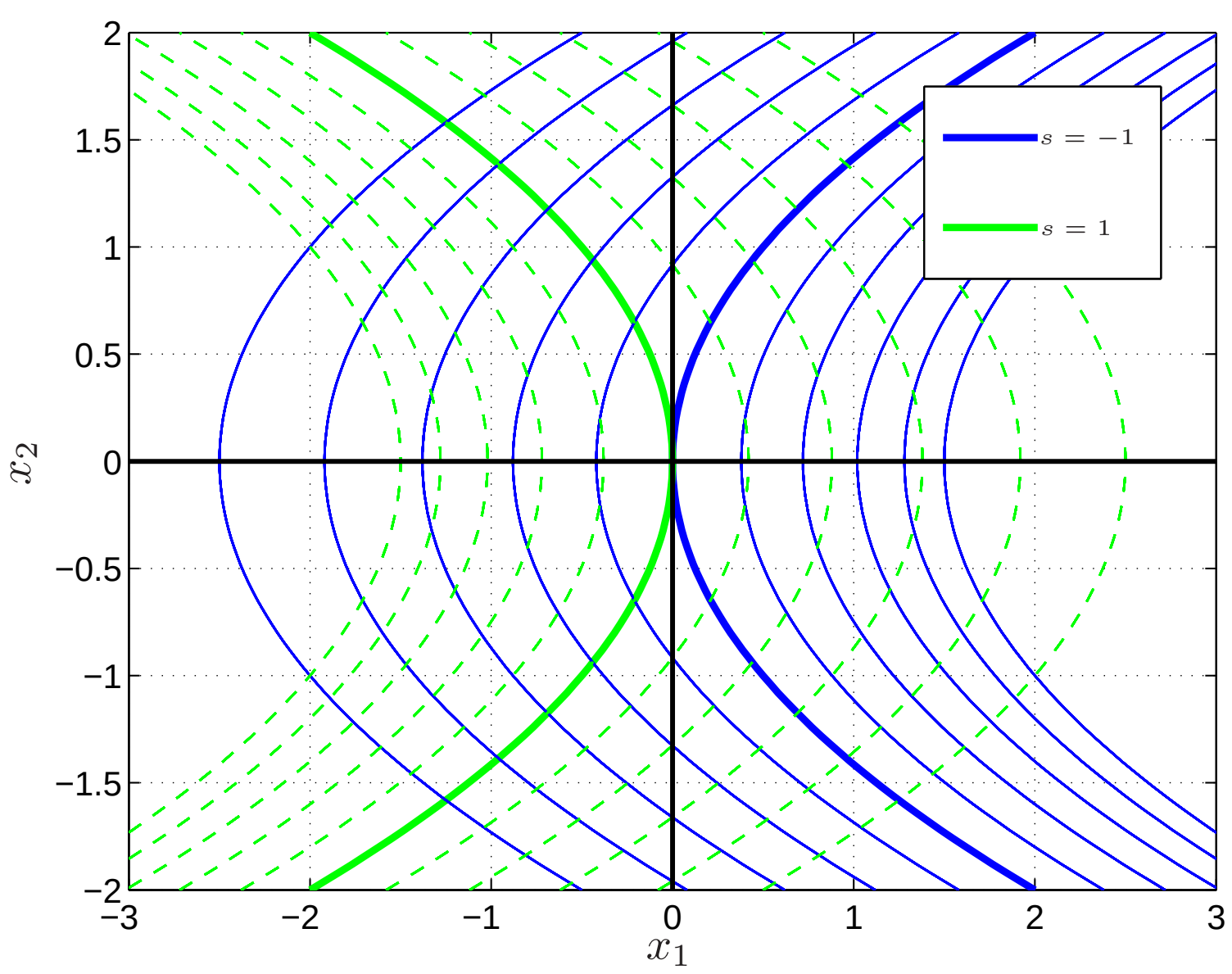
- Four Possible Optimal Controls: $\{-1\}, \{+1\}, \{-1, +1\}, \{+1, -11\}$

- Suppose we let $s = \pm 1$. Then we have

$$\begin{aligned} x_2(t) &= st + x_{20} \\ x_1(t) &= \tfrac{1}{2}st^2 + x_{20}t + x_{10} \end{aligned}$$

- We can then eliminate t from above equations to obtain

$$\begin{aligned} x_1 &= x_{10} + \tfrac{1}{2}sx_2^2 - \tfrac{1}{2}sx_{20}^2 \\ t &= s(x_2 - x_{20}) \end{aligned}$$



- Important Notes
  - ▷ Control *must* steer initial state to *origin*

- ▷ As a result, the extremal solution *must include* one of the parabolas that pass through the origin

- Structure of Optimal Solution
  - ▷ Case 1: Follow thick green curve into origin: $u = -1$
  - ▷ Case 2: Follow thick blue curve into origin: $u = +1$
  - ▷ Case 3: start on a green curve and *switch* to thick blue curve: $u = -1$ followed by $u = +1$
  - ▷ Case 4: start on a blue curve and *switch* to thick green curve: $u = 1$ followed by $u = -1$
- Form of optimal control; *bang-bang*
  - ▷ u^* is *never* on the interior of admissible control set
  - ▷ u^* is *always* at its minimum or maximum value
- Physical Example: Liquid Rocket Thruster: $T \in [0, T_{\max}]$

Singular Arcs

- Recall the Minimum Principle of Pontryagin

$$\mathcal{H}(\mathbf{x}^*, \mathbf{u}^*, \boldsymbol{\lambda}^*) \leq \mathcal{H}(\mathbf{x}^*, \mathbf{u}, \boldsymbol{\lambda}^*)$$

- Suppose Now that There Exists an Interval $[t_1, t_2]$ Such That
  - ▷ $\mathbf{u}^*(t)$ *cannot be determined* from Minimum Principle
  - ▷ Then $[t_1, t_2]$ is called a *singular interval* or *singular arc*
- In General, Singular Intervals are Difficult to Detect
- Certain Classes of Problems: Singular Problems May Be Solvable
  - ▷ Hamiltonian not an explicit function of time $\longrightarrow$ H^* is constant
  - ▷ Can use conditions on H^* to assist with singular arc detection

Determination of Singular Arc Conditions

- Consider a Problem where H^* is constant
- Let $[t_1, t_2]$ be an Interval Such That
  - ▷ Optimal control cannot be determined from necessary conditions
  - ▷ Auxiliary conditions must be used to determine $u^*(t)$ on $[t_1, t_2]$
- Approach for Determining Singular Arc Conditions
  - ▷ Differentiate H with respect to *time* q times until control appears
  - ▷ Because H^* is constant, we know that

$$\frac{d^{(q)}H}{dt^{(q)}} = 0$$

  - ▷ Furthermore, $dH^{(q)}/dt^{(q)}$ is an explicit function of control
  - ▷ Thus, $dH^{(q)}/dt^{(q)}$ can be used to solve for control on $[t_1, t_2]$

Singular Arc Example: Goddard Rocket Problem

- Maximize $h(t_f)$ Subject to
  - ▷ Dynamics

$$\begin{aligned} \dot{h}(t) &= v \\ \dot{v}(t) &= \frac{T(t)-D(v,h)}{m} - g \\ \dot{m}(t) &= -\frac{T(t)}{c} \end{aligned}$$

  - ▷ Boundary Conditions

$$\begin{aligned} h(0) &= 0 \\ v(0) &= 0 \\ m(0) &= m_0 = \text{given} \\ h(t_f) &= \text{Free} \\ v(t_f) &= \text{Free} \\ m(t_f) &= m_f = \text{Given} \end{aligned}$$

  - ▷ Control constraint: $0 \leq T(t) \leq T_{\max}$

Goddard Rocket Problem (Continued)

- Hamiltonian

$$H = \lambda_h v + \lambda_v \left(\frac{T - D}{m} - g \right) - \lambda_m \frac{T}{c}$$
$$= \lambda_h v + \left(\frac{\lambda_v}{m} - \frac{\lambda_m}{c} \right) T - \lambda_v \left(\frac{D}{m} + g \right)$$

- Key Issue
  - ▷ Optimal control may not be bang-bang
  - ▷ Reason: depends upon sign of $\lambda_v/m - \lambda_m/c$

- Possibilities

$$T^*(t) = \begin{cases} 0 & , \quad \lambda_v/m - \lambda_m/c > 0 \\ 0 \leq T^*(t) \leq T_{\max} & , \quad \lambda_v/m - \lambda_m/c = 0 \\ T_{\max} & , \quad \lambda_v/m - \lambda_m/c < 0 \end{cases}$$

- Suppose that $\lambda_v/m - \lambda_m/c = 0 \Longrightarrow$ Singular Arc

Goddard Rocket Problem (Continued)

- H Does Not Depend Explicitly on $t \Longrightarrow H^*$ is constant
- All Time Derivatives of H^* are *Zero*
- In This Case, on Singular Arc We Have

$$\frac{\lambda_v}{m} - \frac{\lambda_m}{c} = 0 \Longrightarrow c\lambda_v - m\lambda_m = 0$$

- Can Let $I = \lambda_v/m - \lambda_m/c$
- Differentiate I with Respect to t Until Control Appears
- This Problem: Control Appears in $\ddot{I}$ and We Know $\ddot{I}^* = 0$
- The Condition $\ddot{I}^* = 0$
  - ▷ Enables determining the control on the singular arc
  - ▷ Gives a "feedback law" for optimal control on singular arc

Summary

- Optimal Control Theory Founded on Calculus of Variations
- Using This Approach, Must Formulate
  - ▷ First-Order Optimality Conditions of Continuous Problem
  - ▷ Leads to a Hamiltonian Boundary-Value Problem
- Determination of Optimal Control
  - ▷ If Control is Interior to Feasible Control Set: $\mathcal{H}_{\mathbf{u}} = \mathbf{0}$
  - ▷ If Control Lies on Boundary of Control Set

$$\mathbf{u}^* = \arg\min_{\mathbf{u}\in\mathcal{U}} \mathcal{H}(\mathbf{x}^*, \mathbf{u}^*, \boldsymbol{\lambda}^*, t^*)$$

- Even in Numerical Methods, Optimal Control Theory Important
  - ▷ Provides Systematic Means of Formulating Problem
  - ▷ Enables Analysis of Solution
- Later: Will Discuss Solution Methods for HBVP

Part III: Nonlinear Optimization

Preliminaries

- A *scalar* is any real number, i.e., x is a scalar if $x \in \mathbb{R}$ where $\mathbb{R}$ is the set of real numbers
- A *column vector* $\mathbf{x}$ lies in n-dimensional real space, i.e., $\mathbf{x}$ is a column vector if $\mathbf{x} \in \mathbb{R}^n$ and is defined as

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

- A *row vector* lies in n-dimensional real space, i.e., $\mathbf{y}$ is a row vector if $\mathbf{y} \in \mathbb{R}^n$ and is defined as

$$\mathbf{y} = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix} \equiv (y_1, y_2, \ldots, y_n)$$

- A matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is defined as

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

- The *Transpose* of a Matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, Denoted $\mathbf{A}^T$, is Defined as

$$\mathbf{A}^{\mathsf{T}} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{m2} & \cdots & a_{nm} \end{bmatrix}$$

- Column Vectors and Row Vectors related by a transpose, i.e.,

$$\mathbf{x} \text{ is a column vector} \iff \mathbf{x}^{\mathsf{T}} \text{ is a row vector}$$

- Conventions Used for Vector Functions
  - ▷ The notation $\mathbb{R}^m \to \mathbb{R}^n$ means *mapping from* $\mathbb{R}^m$ to $\mathbb{R}^n$.

- Let $F : \mathbb{R}^n \to \mathbb{R}$. Then the *gradient* of F is a *column vector* and is defined as

$$\mathbf{G}(\mathbf{x}) \equiv \frac{\partial F}{\partial \mathbf{x}} = \left[\begin{array}{cccc} \frac{\partial F}{\partial x_1} & \frac{\partial F}{\partial x_2} & \cdots & \frac{\partial F}{\partial x_n} \end{array} \right]$$

- Let $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ be a function that maps vectors from $\mathbb{R}^n$ to $\mathbb{R}^m$. Then $\mathbf{h}(\mathbf{x})$ is a *column vector* defined as

$$\mathbf{f}(\mathbf{x}) = \left[\begin{array}{c} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{array} \right]$$

  where $f_i(\mathbf{x}) \equiv f_i(x_1, \ldots, x_n),\ i = 1, \ldots, m.$

- The *Jacobian matrix* or *Jacobian* of $\mathbf{f}(\mathbf{x})$, defined as $\mathbf{J}(\mathbf{x}) \in \mathbb{R}^{m \times n}$,

$$\mathbf{J}(\mathbf{x}) \equiv \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \left[\begin{array}{cccc} \frac{\partial \mathbf{f}}{\partial x_1} & \frac{\partial \mathbf{f}}{\partial x_2} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{array} \right]$$

where $\partial \mathbf{f}/\partial x_j,\ (j = 1, \ldots, n)$ are m-dimensional *column vectors*, i.e.,

$$\frac{\partial \mathbf{f}}{\partial x_j} = \begin{bmatrix} \frac{\partial f_1}{\partial x_j} \\ \frac{\partial f_2}{\partial x_j} \\ \vdots \\ \frac{\partial f_n}{\partial x_j} \end{bmatrix}$$

- In terms of $f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})$ and $x_1, \ldots, x_n$ the Jacobian can be written as

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \ldots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \ldots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \ldots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

- Using These Conventions, for the special case of a function $F : \mathbb{R}^n \to \mathbb{R}$, the gradient and Jacobian of F are defined, respectively, as

$$\begin{aligned} \mathbf{G}(\mathbf{x}) &= \frac{\partial F}{\partial \mathbf{x}} \equiv \nabla_{\mathbf{x}} F \\ \mathbf{J}(\mathbf{x}) &= \frac{\partial \mathbf{G}}{\partial \mathbf{x}} \equiv \frac{\partial^2 F}{\partial \mathbf{x}^2} \equiv \mathbf{H}(\mathbf{x}) = \nabla^2_{\mathbf{x}} F \end{aligned}$$

where the function $\mathbf{H}(\mathbf{x})$ has a special name called the *Hessian* of F. The Hessian of F is written explicitly in terms of the gradient as

$$\mathbf{H}(\mathbf{x}) = \left[\begin{array}{cccc} \frac{\partial \mathbf{G}}{\partial x_1} & \frac{\partial \mathbf{G}}{\partial x_2} & \cdots & \frac{\partial \mathbf{G}}{\partial x_n} \end{array} \right]$$

or, alternatively, in terms of F as

$$\mathbf{H}(\mathbf{x}) = \left[\begin{array}{cccc} \frac{\partial^2 F}{\partial x_1^2} & \frac{\partial^2 F}{\partial x_2 x_1} & \cdots & \frac{\partial^2 F}{\partial x_n x_1} \\ \frac{\partial^2 F}{\partial x_1 x_2} & \frac{\partial^2 F}{\partial x_2^2} & \cdots & \frac{\partial^2 F}{\partial x_n x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 F}{\partial x_1 x_n} & \frac{\partial^2 F}{\partial x_2 x_n} & \cdots & \frac{\partial^2 F}{\partial x_n^2} \end{array} \right] \in \mathbb{R}^{n \times n}$$

It is noted that the Hessian is *symmetric*.

- **Important Notes:** Using the above conventions, the Hessian is derivative of the *transpose* of the gradient, i.e.,

$$\mathbf{H}(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}} \left[\mathbf{G}^{\mathsf{T}}(\mathbf{x}) \right]$$

- The Hessian matrix $\mathbf{H}(\mathbf{x})$ is *symmetric*, i.e., $\mathbf{H} = \mathbf{H}^{\mathsf{T}}$
- Abbreviated Notation for Gradient, Jacobian, and Hessian

- Gradient of a Scalar Function $F(\mathbf{x})$:

$$\mathbf{G}(\mathbf{x}) \equiv \frac{\partial F}{\partial \mathbf{x}} \equiv F_{\mathbf{x}}$$

- Hessian of a Scalar Function $F(\mathbf{x})$:

$$\mathbf{H}(\mathbf{x}) \equiv \frac{\partial^2 F}{\partial \mathbf{x}^2} \equiv F_{\mathbf{xx}}$$

- Jacobian of a Vector Function $\mathbf{f}(\mathbf{x})$:

$$\mathbf{J}(\mathbf{x}) \equiv \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \equiv \mathbf{f}_{\mathbf{x}}$$

Unconstrained Optimization

- Let $F : \mathbb{R}^n \to \mathbb{R}$. Then $\mathbf{x}^*$ is called a **strong minimum** of F if there exists $\delta \in \mathbb{R}$ such that

$$F(\mathbf{x}^*) < F(\mathbf{x}^* + \Delta\mathbf{x})$$

for all $0 < \|\Delta x\| < \delta$. Similarly, $\mathbf{x}^*$ is called a **weak minimum** of F if $\mathbf{x}^*$ is not a strong minimum and there exists $\delta \in \mathbb{R}$ such that

$$F(\mathbf{x}^*) \leq F(\mathbf{x}^* + \Delta\mathbf{x})$$

$0 < \|\Delta x\| < \delta$.

- Global vs. Local Minimum
  - ▷ $\mathbf{x}^*$ is said to be a **global minimum** of F if $\delta = \infty$.
  - ▷ $\mathbf{x}^*$ is said to be a **local minimum** of F if $\delta < \infty$.

  In general, a scalar function $F(\mathbf{x})$ will have many local minima of which one of them *may* be a global minimum.

Necessary & Sufficient Conditions for a Local Minimum

- Assume $F(\mathbf{x})$ is Twice Differentiable in a Neighborhood of $\mathbf{x}^*$, We Have

$$F(\mathbf{x}^*) \approx F(\mathbf{x}^* + \Delta\mathbf{x}) + \left[\frac{\partial F}{\partial \mathbf{x}}\right]_{\mathbf{x}^*} \Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^{\mathsf{T}} \left[\frac{\partial^2 F}{\partial \mathbf{x}^2}\right]_{\mathbf{x}^*} \Delta\mathbf{x}$$
$$= F(\mathbf{x}^* + \Delta\mathbf{x}) + \mathbf{G}^{\mathsf{T}}(\mathbf{x}^*)\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^{\mathsf{T}}\mathbf{H}(\mathbf{x})\Delta\mathbf{x}$$

- $\mathbf{x}^*$ is said to be a *stationary point* of $F(\mathbf{x})$ if

$$\mathbf{G}(\mathbf{x}^*) = \mathbf{0}$$

- Necessary Conditions
  - $\mathbf{G}(\mathbf{x}^*) = \mathbf{0}$, i.e., $\mathbf{x}^*$ must be a stationary point of F
  - $\mathbf{H}(\mathbf{x}^*) \geq 0$, (i.e., $\mathbf{H}(\mathbf{x}^*)$ is positive semi-definite)
- Sufficient Conditions
  - $\mathbf{G}(\mathbf{x}^*) = \mathbf{0}$, i.e., $\mathbf{x}^*$ must be a stationary point of F
  - $\mathbf{H}(\mathbf{x}^*) > 0$, (i.e., $\mathbf{H}(\mathbf{x}^*) > 0$, is positive definite)

Solution of Unconstrained Optimization Problems

- What Needs to Be Done
  - Determine stationary points from $\mathbf{G}(\mathbf{x})$: give candidate local minima
  - Check definiteness of $\mathbf{H}(\mathbf{x})$ at all stationary points
  - Accept all stationary points where $\mathbf{H}$ is either PD or PSD
  - Reject all stationary points where $\mathbf{H}$ is neither PD nor PSD
- Above Mathematical Approach Nice in Theory
- General Problem
  - No analytic solution
  - Need to solve numerically
  - Numerical methods are generally iterative
- Categories of numerical methods for unconstrained optimization
  - Univariate (functions of a single variable) problems
  - Multivariate (functions of many variables) problems

Univariate Optimization

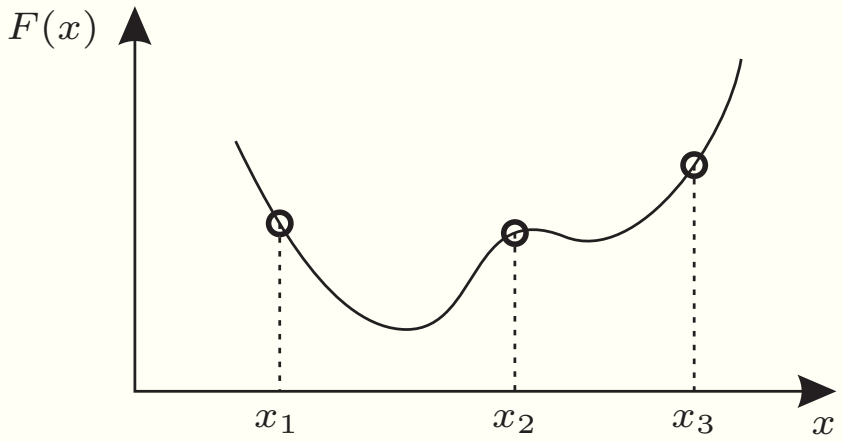
- Consider the following problem where $x \in \mathbb{R}$:

$$\min_{x \in \mathbb{R}} F(x)$$

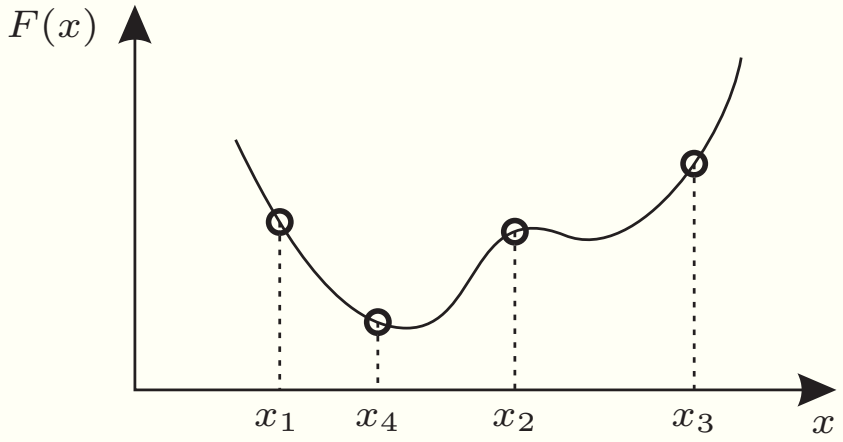
- Necessary & sufficient conditions for a minimum are

$$F'(x) = \frac{dF}{dx} = 0 \quad , \quad F''(x) = \frac{d^2F}{dx^2} > 0$$

- Complicated F: Need to Employ an Iterative Method
- First Step: Bracket the Minimum
  - ▷ Know values of F at x_1, x_2, and x_3
  - ▷ Condition: $F(x_2) < F(x_1)$ and $F(x_2) < F(x_3)$



▹ Take an Additional Point x_4



- Then

$$F(x_4) < F(x_2) \Rightarrow \text{New Interval: } (x_1, x_4, x_2)$$

$$F(x_4) > F(x_2) \Rightarrow \text{New Interval: } (x_4, x_2, x_3)$$

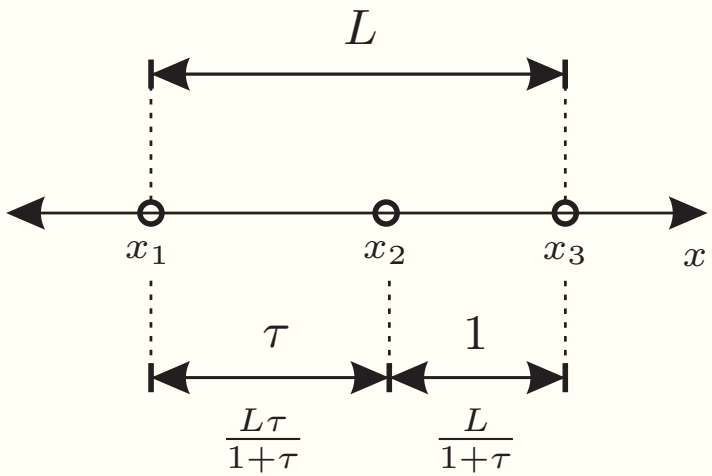
- Question: How Do We Choose x_4?

- Commonly Used Methods
  - Golden section
  - Polynomial approximation

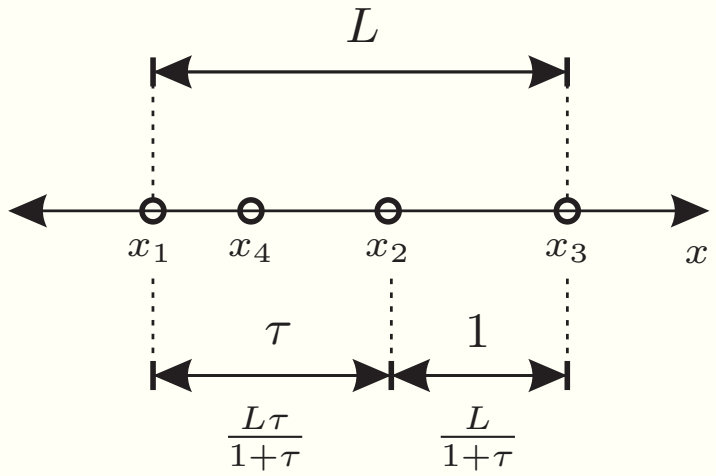
Golden Section Search

- Let $L = x_3 - x_1$ and $\tau = \frac{x_2 - x_1}{x_3 - x_2}$, i.e.,

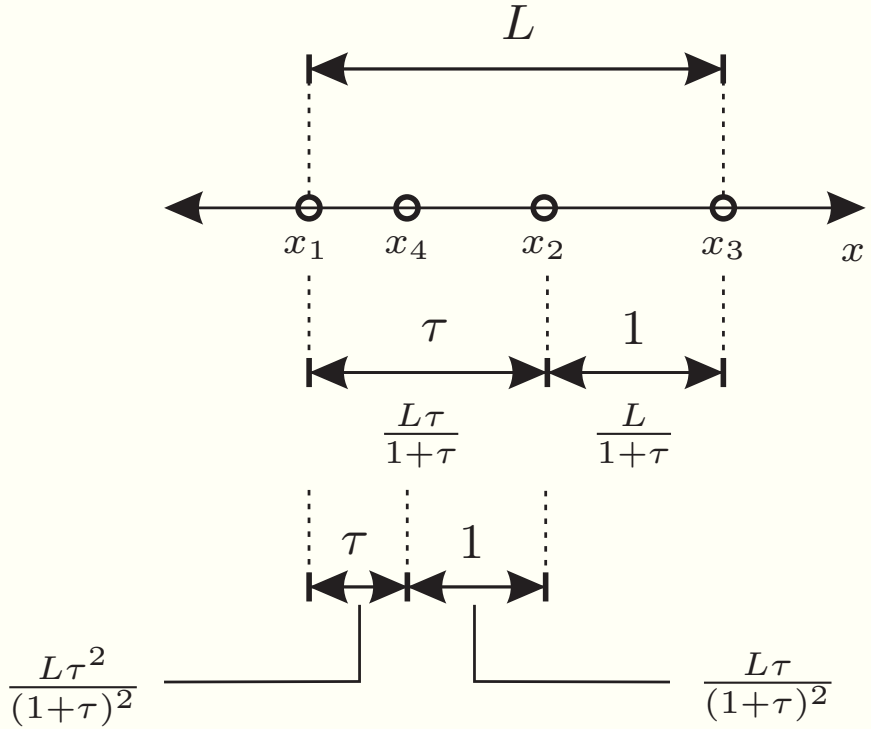
$$\tau = \frac{x_2 - x_1}{x_3 - x_2} \Longrightarrow \begin{cases} x_2 - x_1 &= \frac{L\tau}{1+\tau} \\ x_3 - x_2 &= \frac{L}{1+\tau} \end{cases}$$



- Goal: Maximize Reduction in Bracket at Each Step
- Choose x_4 To Be in Larger Interval



- Want New Interval to Have Same Ratio as Original Interval



- Therefore,

$$x_2 - x_4 = \frac{1}{1+\tau}\frac{L\tau}{1+\tau} = \frac{L\tau}{(1+\tau)^2}$$

- Want $x_2 - x_1$ to equal $x_3 - x_4$ Which Implies That

$$x_2 - x_1 \rightleftharpoons x_3 - x_4 = \frac{L\tau}{1+\tau}$$

- Using the constraint that $x_2 - x_1 = x_3 - x_4$, we obtain

$$L = x_3 - x_1 = x_4 - x_1 + x_3 - x_4 = \frac{L\tau^2}{(1+\tau)^2} + \frac{L\tau}{1+\tau}$$

- Rearranging, we obtain

$$(1+\tau)^2 = \tau(1+\tau) + \tau^2 \Longrightarrow \tau^2 - \tau - 1 = 0$$

- Roots of Polynomial:

$$\tau_{1,2} = \frac{1 \pm \sqrt{5}}{2}$$

- Because τ Must Be Greater Than Unity (and Positive),

$$\tau = \frac{1+\sqrt{5}}{2} = \textbf{Golden Ratio}$$

Algorithm 1 (Golden Section)

1. Given Interval (x_1, x_2, x_3) Where $F(x_2) < F(x_1)$ and $F(x_2) < F(x_3)$
2. Choose x_4 To Be in Large Part of Interval Such That Ratio of Large Part of Interval is τ
3. Find $F(x_4)$
4. If $F(x_4) > F(x_2)$, New Interval is (x_1, x_2, x_4), Otherwise New Interval is (x_2, x_4, x_3)
5. Assign New Interval $\Longrightarrow (x_1, x_2, x_3)$
6. Tolerance met? Yes $\Rightarrow$ Stop; Otherwise $\Rightarrow$ Step 2

Polynomial Line Search

- Key Attributes of Golden Section
  - ▷ No assumptions were made about function
  - ▷ Very robust, but has slow (linear) convergence
  - ▷ Would like a method with faster convergence
- Faster Convergence: Quadratic Approximation to F
- Approach: Assume $F(x)$ is Well-Behaved
  - ▷ Start with three points x_1, x_2, and x_3
  - ▷ Fit quadratic through these points, i.e.,

$$p(x) = a(x-b)^2 + c$$

  - ▷ x_4 obtained where $p(x)$ is minimized (i.e., $x = b$)
  - ▷ After some algebra, it turns out that

$$b = \frac{1}{2}\frac{F(x_1)(x_3^2 - x_2^2) + F(x_2)(x_1^2 - x_3^2) + F(x_3)(x_2^2 - x_1^2)}{F(x_1)(x_3 - x_2) + F(x_2)(x_3 - x_1) + F(x_3)(x_2 - x_1)}$$

- Benefits of Quadratic Line Search

- ▷ Much faster convergence than golden section
- ▷ Near minimum: *superlinear* convergence
- ▷ Caution: no such thing as a free lunch
  - ◦ Quadratic line search can fail
  - ◦ May need to use a less aggressive approach

- Polynomial Line Search Can Be of Any Order
  - ▷ For N^{th}-degree approximation, need $N + 1$ points
  - ▷ Generally only need to go as high as a cubic
- Algorithm for Quadratic Line Search

Algorithm 2 (Quadratic Line Search)

1. Given Interval (x_1, x_2, x_3) Where $F(x_2) < F(x_1)$ and $F(x_2) < F(x_3)$
2. Approximate F by a Quadratic $p(x)$ Using x_1, x_2, and x_3
3. Determine x_4 from Minimum of $p(x)$
4. If $F(x_4) > F(x_2)$, New Interval is (x_1, x_2, x_4), Otherwise New Interval is (x_2, x_4, x_3)

5. Assign New Interval $\Longrightarrow (x_1, x_2, x_3)$

6. Tolerance met? Yes $\Rightarrow$ Stop; Otherwise $\Rightarrow$ Step 2

Multivariate Unconstrained Optimization

- Supply an initial guess of $\mathbf{x}^*$, i.e., $\mathbf{x}_0$
- Perform an update at each step k as follows:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{D}_k \mathbf{p}_k$$

where $\mathbf{p}_k$ is the **search direction**, α_k is the **step length** and is determined via a **line search**, and $\mathbf{D}_k \in \mathbb{R}^{n \times n}$ is a positive definite matrix.

Algorithm 3 (Unconstrained Minimization)

1. Set $k = 0$ and choose initial guess, $\mathbf{x}_k = \tilde{\mathbf{x}}$
2. Test for convergence: $\mathbf{G}(\mathbf{x}_k) = \mathbf{0}$ and $\mathbf{H}(\mathbf{x}_k) > 0$?
3. If answer to step (2) is "yes", then quit
4. Otherwise, compute
   (a) **search direction**, $\mathbf{p}_k$
   (b) **step length**, α_k
   (c) **Positive definite matrix**, $\mathbf{D}_k$

5. Update $\mathbf{x}_k$ as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{D}_k \mathbf{p}_k$$

6. Return to step (2)

Methods for Determination of $\mathbf{p}_k$, α_k, and $\mathbf{D}_k$

- Determination of $\mathbf{p}_k$ and $\mathbf{D}_k$
  - ▷ First-order methods
    - ○ Use gradient information
    - ○ Common method: **steepest descent**
  - ▷ Second-order methods
    - ○ Use both gradient *and* Hessian information
    - ○ Common method: **Newton method**
  - ▷ Quasi-Newton methods
    - ○ Use an approximation to Hessian
    - ○ Common methods: DFP and BFGS
- Methods for Determination of Step Length, α_k
  - ▷ Step Length Determined from Univariate Optimization Methods
    - ○ Exact Line Search
    - ○ Inexact Line Search

Method of Steepest Descent

- Want a Search Direction that Decreases Value of F at Each Step
- Perform a Taylor Series Expansion About $\mathbf{x}_k$

$$F(\mathbf{x}_{k+1}) \approx F(\mathbf{x}_k) + \mathbf{G}^{\mathsf{T}}(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k)$$
$$= F(\mathbf{x}_k) + \mathbf{G}_k^{\mathsf{T}}(\alpha_k \mathbf{D}_k \mathbf{p}_k)$$

  This last result implies that

$$\mathbf{x}_{k+1} - \mathbf{x}_k = \alpha_k \mathbf{D}_k \mathbf{p}_k$$

- Now assume that $\alpha_k > 0$ and is chosen such that the function F decreases in moving from $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$, i.e., $F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k)$. Then it must be the case that

$$\mathbf{G}_k^{\mathsf{T}} \mathbf{D}_k \mathbf{p}_k < 0$$

- **Suppose We Choose** $\mathbf{p}_k = -\mathbf{G}_k$ and $\mathbf{D}_K = \mathbf{I}$. Then

$$\mathbf{x}_{k+1} - \mathbf{x}_k = -\alpha_k \mathbf{G}_k \Longrightarrow \mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{G}_k$$

- The Direction $\mathbf{p}_k = -\mathbf{G}_k$ is Called the Direction of **Steepest Descent**

Newton Method

- Suppose at Each Step we Choose $\mathbf{x}_{k+1}$ Such That

$$\mathbf{G}(\mathbf{x}_{k+1}) = \mathbf{0}$$

- Then

$$\mathbf{G}(\mathbf{x}_{k+1}) \approx \mathbf{G}(\mathbf{x}_k) + \left[\frac{\partial \mathbf{G}}{\partial \mathbf{x}}\right]_{\mathbf{x}_k} (\mathbf{x}_{k+1} - \mathbf{x}_k) = \mathbf{0}$$

- But We Know That

$$\left[\frac{\partial \mathbf{G}}{\partial \mathbf{x}}\right]_{\mathbf{x}_k} = \mathbf{H}(\mathbf{x}_k)$$

- Solving for $\mathbf{x}_{k+1}$, we obtain the **Newton Method**

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}^{-1}(\mathbf{x}_k)\mathbf{G}(\mathbf{x}_k)$$

- Therefore, for a Newton Method,

$$\mathbf{D}_k = \mathbf{H}^{-1}(\mathbf{x}_k) \quad ; \quad \mathbf{p}_k = \mathbf{G}(\mathbf{x}_k) \quad ; \quad \alpha_k = 1$$

Comparison Between Steepest Descent & Newton Iteration

- Steepest Descent
  - ▷ Simple Iterative Approach: Uses Only *First-Derivatives*
  - ▷ Slow (Linear) Convergence, But Very Reliable
- Newton Method
  - ▷ More Sophisticated: Uses *Second-Derivatives*
  - ▷ Much Faster Convergence (Quadratic)
  - ▷ Not As Reliable
    - ○ **H** May Not Remain Positive Definite
    - ○ Known to Diverge Depending Upon Initial Guess
- Want an Iterative Procedure with Fast Convergence and Reliability
- Common Balanced Approach: Quasi-Newton Methods

Quasi-Newton Methods

- Definition of a Quasi-Newton Method

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{D}_k \mathbf{G}_k$$

  where $\mathbf{D}_k$ is an *approximation* to the *inverse* of the Hessian at each step

- Key Idea: Approximate Convexity (i.e., Second Derivative)
  - ▷ Does *Not* Require Computation the Hessian of F
  - ▷ Requires Only Information Functions and Gradients
- Construct Quasi-Newton Approximation to $\mathbf{H}$
  - ▷ Consider two successive iterates $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$
  - ▷ We know that

$$\begin{aligned} \mathbf{G}(\mathbf{x}_{k+1}) &\approx \mathbf{G}(\mathbf{x}_k) + \mathbf{H}_k(\mathbf{x}_{k+1} - \mathbf{x}_k) \\ \Longrightarrow \mathbf{G}(\mathbf{x}_{k+1}) - \mathbf{G}(\mathbf{x}_k) &\approx \mathbf{H}_k(\mathbf{x}_{k+1} - \mathbf{x}_k) \\ \Longleftrightarrow \mathbf{G}_{k+1} - \mathbf{G}_k &\approx \mathbf{H}_k(\mathbf{x}_{k+1} - \mathbf{x}_k) \end{aligned}$$

- ▷ Suppose now that we let

$$\mathbf{q}_k = \mathbf{G}_{k+1} - \mathbf{G}_k$$

- ▷ Then, using the fact that $\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$, we have

$$\mathbf{p}_k^{\mathsf{T}} \mathbf{H}_k \mathbf{p}_k \approx (\mathbf{G}_{k+1} - \mathbf{G}_k)^{\mathsf{T}} \mathbf{p}_k = \mathbf{q}_k^{\mathsf{T}} \mathbf{p}_k$$

- ▷ It is noted that $\mathbf{p}_k^{\mathsf{T}} \mathbf{H}_k \mathbf{p}_k$ is an approximation to the *curvature* of F along $\mathbf{p}_k$.
- ▷ Approximation to Hessian Constructed Using Information from Previous Iterations

- Suppose Now That We Have $\mathbf{B}_k$, where $\mathbf{B}_k$ is an Approximation to $\mathbf{H}_k$
  - ▷ At Step $k+1$, $\mathbf{B}_k$ Updated Using *Update Matrix* $\mathbf{U}_k$ As

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \mathbf{U}_k$$

  - ▷ $\mathbf{B}_{k+1}$ must satisfy the *Quasi-Newton condition*

$$\mathbf{B}_{k+1} \mathbf{p}_k = \mathbf{q}_k$$

- ▷ Assume now that $\mathbf{U}_k = \mathbf{v}\mathbf{w}^\mathsf{T}$. Then

$$\begin{aligned} \mathbf{B}_{k+1}\mathbf{p}_k &= (\mathbf{B}_k + \mathbf{v}\mathbf{w}^\mathsf{T})\mathbf{p}_k = \mathbf{q}_k \\ \Longrightarrow \mathbf{v}(\mathbf{w}^\mathsf{T}\mathbf{p}_k) &= \mathbf{q}_k - \mathbf{B}_k\mathbf{p}_k \end{aligned}$$

- ▷ Can Be Shown That

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{1}{\mathbf{w}^\mathsf{T}\mathbf{p}_k}\left(\mathbf{q}_k - \mathbf{B}_k\mathbf{p}_k\right)\mathbf{w}^\mathsf{T}$$

- Well-Known Quasi-Newton Update Algorithms
  - ▷ **Davidon-Fletcher-Powell (DFP)**

$$\begin{aligned} \mathbf{B}_{k+1} = \mathbf{B}_k - \frac{\mathbf{B}_k\mathbf{p}_k\mathbf{p}_k^\mathsf{T}\mathbf{B}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{B}_k\mathbf{p}_k} + \frac{\mathbf{q}_k\mathbf{q}_k^\mathsf{T}}{\mathbf{q}_k^\mathsf{T}\mathbf{p}_k} \\ + \left(\mathbf{p}_k^\mathsf{T}\mathbf{B}_k\mathbf{p}_k^\mathsf{T}\right)\mathbf{y}_k\mathbf{y}_k^\mathsf{T} \end{aligned}$$

where

$$\mathbf{y}_k = \frac{\mathbf{q}_k}{\mathbf{q}_k^\mathsf{T}\mathbf{p}_k} - \frac{\mathbf{B}_k\mathbf{p}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{B}_k\mathbf{q}_k}$$

  - ▷ **Broyden-Fletcher-Goldfarb-Shanno (BFGS)**

$$\mathbf{B}_{k+1} = \mathbf{B}_k - \frac{\mathbf{B}_k\mathbf{p}_k\mathbf{p}_k^\mathsf{T}\mathbf{B}_K}{\mathbf{p}_k^\mathsf{T}\mathbf{B}_k\mathbf{p}_k} + \frac{\mathbf{q}_k\mathbf{q}_k^\mathsf{T}}{\mathbf{q}_k^\mathsf{T}\mathbf{p}_k}$$

- Using the Quasi-Newton Approximation to the Inverse of the Hessian ($\mathbf{B}_k$),

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{B}_k \mathbf{p}_k$$

- Important Points About Quasi-Newton Updates
  - ▷ $\mathbf{B}_0$ Positive Definite $\longrightarrow$ $\mathbf{B}_k$ Remains Positive Definite
  - ▷ $\mathbf{B}_0$ Generally Taken As Identity Matrix

Line Search

- What is the Best Value of α_k?
- Obtain α_k from Solution of Minimization Problem

$$\min_{\alpha_k \geq 0} F(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$$

- Solution of Line Search Minimization Problem
  - ▷ Optimal value of α_k: **exact line search**
  - ▷ Suboptimal value of α_k: **inexact line search**
  - ▷ Reasons to employ inexact line search
    - ○ Exact line searches are inefficient and add little to increase rate of convergence
    - ○ Inexact line searches are fast and efficient
- **Commonly Used Inexact Line Search Methods**
  - ▷ Golden Section
  - ▷ Bisection
  - ▷ Polynomial Approximations

- Important Points About Line Search
  - A line search is a *univariate* optimization problem
  - Line searches satisfy **Goldstein-Armijo** condition:

$$0 \leq -\kappa_1 \alpha_k F'(0) \leq F(0) - F(\alpha_k) \leq -\kappa_2 \alpha_k F'(0)$$

Trust Region Methods

- Pose the Following Constrained Optimization Problem:

$$\min_{\mathbf{p}} \mathbf{G}_k^{\mathsf{T}}\mathbf{p} + \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}_k\mathbf{p} \text{ subject to } \mathbf{p}^{\mathsf{T}}\mathbf{p} < \Delta$$

- Idea: Δ is a "Trust-Radius"
- Property: for $\lambda \in \mathbb{R}$ The Solution $\mathbf{p}$ of

$$(\mathbf{H}_K + \lambda I)\mathbf{p} = -\mathbf{G}_k$$

Solves the problem if *either*

$$\lambda = 0 \quad \text{and} \quad \mathbf{p}^{\mathsf{T}}\mathbf{p} \leq \Delta$$

or

$$\lambda \geq 0 \quad \text{and} \quad \mathbf{p}^{\mathsf{T}}\mathbf{p} = \Delta$$

- Properties of Trust Region Methods
  - ▷ Δ Sufficiently large $\longrightarrow$ Newton method
  - ▷ $\Delta \rightarrow 0$ and $\|\mathbf{p}\| \rightarrow 0 \longrightarrow \mathbf{p}$ Approaches Steepest Descent Direction

Example of Unconstrained Optimization

- Quartic Unconstrained Minimization Problem:

$$\min_{x_1,x_2} F(x_1, x_2)$$

where

$$F(x_1, x_2) = 100(x_1 - x_2^2)^2 + (1 - x_1)^2$$

- Necessary Conditions for Optimality

$$\begin{aligned} \frac{\partial F}{\partial x_1} &= 200(x_1 - x_2^2) - 2(1 - x_1) = 0 \\ \frac{\partial F}{\partial x_2} &= -400x_2(x_1 - x_2^2) = 0 \end{aligned}$$

- Stationary Points

$$\begin{aligned} (x_1, x_2) &= (101, 0) \\ (x_1, x_2) &= (1, 1) \end{aligned}$$

- Hessian of F

$$\mathbf{H} = \begin{bmatrix} 200 & -400x_2 \\ -400x_2 & 1200x_2^2 \end{bmatrix}$$

- Check Definiteness of $\mathbf{H}$ at Each Stationary Point

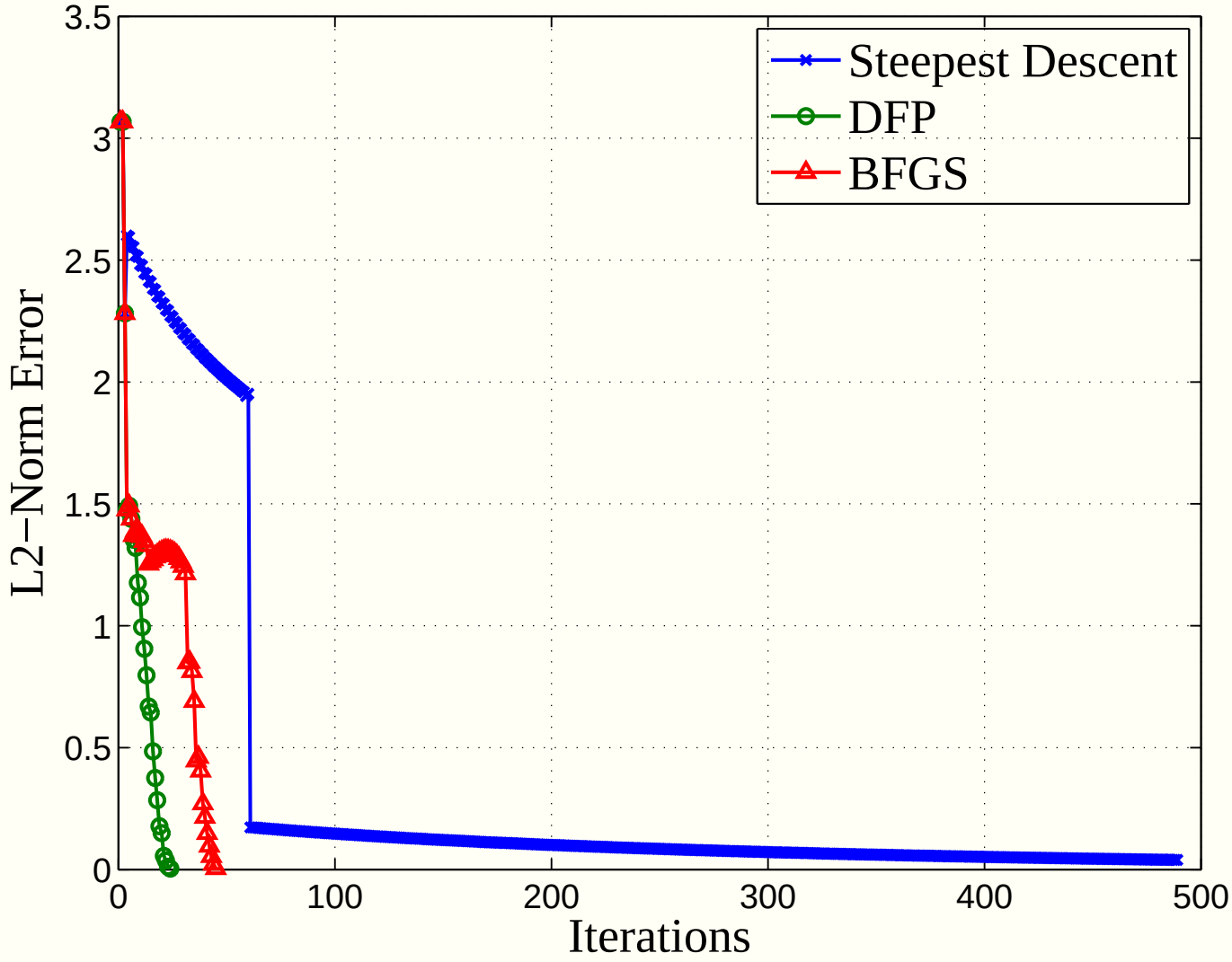
$$\mathbf{H}(x_1 = 101, x_2 = 0) = \begin{bmatrix} 200 & 0 \\ 0 & 0 \end{bmatrix} \geq 0$$

$$\mathbf{H}(x_1 = 1, x_2 = 1) = \begin{bmatrix} 200 & -400 \\ -400 & 1200 \end{bmatrix} > 0$$

$\Longrightarrow$ Optimal solution is $(x_1, x_2) = (1, 1)$

Solution of Example Via Iteration

- Use Various Iterative Approaches to Solve Example
- MATLAB: can use **fminunc** (Code on Next Slides)
  - ▷ Three different Hessian updates: steepest descent, DFP, and BFGS
  - ▷ Need to set tolerances on $\mathbf{x}$ and F: Choose 10^{-5}
- Results



FMINUNC Code for Quartic Example Problem

```
global xpath iters;
xpath = zeros(10000,2);
iters = 0;
xlow = -Inf*ones(2,1);
xupp =  Inf*ones(2,1);
opt = optimset('fminunc');
opt = optimset(opt,'Hessupdate','steepdesc','gradobj','on','Display', ...
             'iter','LargeScale','off','InitialHessType','identity', ...
             'tolfun',1e-5,'tolx',1e-5,'MaxFunEvals',100000,'MaxIter', ...
             10000,'OutputFcn',@quartic_outfun);
x0 = [-1.9 2];
xout_gradsearch = fminunc(@quartic_usrfun,x0,opt);
xpath_gradsearch = xpath(1:iters,:);

xpath = zeros(10000,2);
iters = 0;
opt = optimset('fminunc');
opt = optimset(opt,'Hessupdate','bfgs','gradobj','on','Display', ...
             'iter','LargeScale','off','InitialHessType','identity', ...
             'tolfun',1e-5,'tolx',1e-5,'MaxFunEvals',100000,'OutputFcn',@ ...
             quartic_outfun);
xout_bfgssearch = fminunc(@quartic_usrfun,x0,opt);
xpath_bfgssearch = xpath(1:iters,:);

xpath = zeros(10000,2);
iters = 0;
opt = optimset('fminunc');
opt = optimset(opt,'Hessupdate','dfp','gradobj','on','Display', ...
             'iter','LargeScale','off','InitialHessType','identity', ...
             'tolfun',1e-5,'tolx',1e-5,'MaxFunEvals',100000,'OutputFcn',@ ...
             quartic_outfun);
xout_dfpsearch = fminunc(@quartic_usrfun,x0,opt);
```

```
xpath_dfpsearch = xpath(1:iters,:);

x_ex = [1 1];
x_exact = [ones(length(xpath_gradsearch),1) ones(length(xpath_gradsearch),1)];
err_grad = xpath_gradsearch-x_exact;
err_grad = sqrt(dot(err_grad,err_grad,2));
x_exact = [ones(length(xpath_bfgssearch),1) ones(length(xpath_bfgssearch),1)];
err_bfgs = xpath_bfgssearch-x_exact;
err_bfgs = sqrt(dot(err_bfgs,err_bfgs,2));
x_exact = [ones(length(xpath_dfpsearch),1) ones(length(xpath_dfpsearch),1)];
err_dfp = xpath_dfpsearch-x_exact;
err_dfp = sqrt(dot(err_dfp,err_dfp,2));

figure(1);
hh1=plot(xpath_gradsearch(:,1),xpath_gradsearch(:,2),'o');
hold on;
hh2=plot(x_ex(1),x_ex(2),'d');
set(hh2,'MarkerSize',12,'MarkerFaceColor','Black');
print -depsc2 quartic_iterations.eps;

figure(2);
hh1=plot(xpath_bfgssearch(:,1),xpath_bfgssearch(:,2),'x');
hold on;
hh2=plot(x_ex(1),x_ex(2),'d');
set(hh2,'MarkerSize',12,'MarkerFaceColor','Black');

figure(3);
hh1=plot(xpath_dfpsearch(:,1),xpath_dfpsearch(:,2),'x');
hold on;
hh2=plot(x_ex(1),x_ex(2),'d');
set(hh2,'MarkerSize',12,'MarkerFaceColor','Black');

figure(4);
iters_grad =length(err_grad);
iters_bfgs =length(err_bfgs);
iters_dfp =length(err_dfp);
n_grad = 1:iters_grad;
```

```
n_bfgs = 1:iters_bfgs;
n_dfp  = 1;iters_dfp;
hh=plot(n_grad,err_grad,'-x',n_bfgs,err_bfgs,'-o',n_dfp,err_dfp,'-^');
set(hh,'LineWidth',1.5);
grid on;
xl=xlabel('Iterations');
yl=ylabel('L_2-Norm Error');
ll=legend('Steepest Descent','DFP','BFGS');
set(xl,'FontName','Times','FontSize',16);
set(yl,'FontName','Times','FontSize',16);
set(ll,'FontName','Times','FontSize',16);

% Quartic Objective Function
function [F,G] = quartic_usrfun(x);

if size(x,1)==2,
  x=x';
end;
F =  100*(x(:,2)-x(:,1).^2).^2+(1-x(:,1)).^2;
G = [100*(4*x(1)^3-4*x(1)*x(2))+2*x(1)-2; 100*(2*x(2)-2*x(1)^2)];
return;

% Quartic Output Function
function stop = quartic_outfun(x,optimValues,state,varargin)
global xpath iters;
stop=[];
persistent history
persistent searchdir
hold on
switch state
case 'init'
history = []; searchdir = [];
case 'iter'
% Concatenate current point and objective function value
% with history. x must be a row vector.
history = [history;[x optimValues.fval]];
% Concatenate current search direction with searchdir.
```



```
 searchdir = [searchdir; optimValues.searchdirection'];
 plot(x(1),x(2),'o');
 % Label points with iteration number.
 text(x(1)+.15,x(2),num2str(optimValues.iteration));
 iters = iters+1;
 xpath(iters,:) = x;
 case 'done'
 assignin('base','hist', history);
 assignin('base','search', searchdir);
otherwise
 end
 hold off
 return;
```

Constrained Optimization

- Most Optimization Problems are *Constrained*
- Different Types of Constraints
  - ▷ Equality: constraints are *active* at optimal solution
  - ▷ Inequality: constraints may be either active *or* inactive at optimal solution
- General form of constrained optimization problem:

$$\min_{\mathbf{x}} F(\mathbf{x})$$

s.t.

$$\mathbf{g}(\mathbf{x}) = \mathbf{0}$$

$$\mathbf{h}(\mathbf{x}) \leq \mathbf{0}$$

  where $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^m$ and $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^p$

- Above Optimization Problem is Called a *Nonlinear Programming Problem* (NLP)

- Following Discussion
  - ▷ Nonlinear equality-constrained problem (NEP)
  - ▷ Nonlinear inequality-constrained problem (NIP)

Equality Constraints

- Nonlinear Equality-Constrained Problem (NEP)

$$\min_{\mathbf{x}} F(\mathbf{x})$$

s.t.

$$\mathbf{g}(\mathbf{x}) = \mathbf{0}$$

- Lagrangian Function

$$L(\mathbf{x}, \boldsymbol{\lambda}) = F(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i \mathbf{g}_i(\mathbf{x})$$

- Define the following quantities:
  - ▷ **Jacobian** of the constraints as

$$\mathbf{A}(\mathbf{x}) = \frac{\partial \mathbf{g}}{\partial \mathbf{x}} \in \mathbb{R}^{m \times n}$$

  - ▷ **Hessian** of the constraints, defined as

$$\hat{\mathbf{H}}_i(\mathbf{x}) = \frac{\partial^2 g_i}{\partial \mathbf{x}^2} \in \mathbb{R}^{n \times n}, \quad (i = 1, \ldots, m)$$

where $\mathbf{A}_i(\mathbf{x})$ is the i^{th} column of $\mathbf{A}(\mathbf{x})$

- A matrix $\mathbf{Z}(\mathbf{x})$ such that

$$\mathbf{Z}(\mathbf{x}^*) \in \mathbb{R}^{n \times m} \Longrightarrow \mathbf{Z}^{\mathsf{T}} \in \mathbb{R}^{m \times n}$$

where the columns of $\mathbf{Z}(\mathbf{x}^*)$ form a basis for the set of vectors orthogonal to the rows of $\mathbf{A}(\mathbf{x}^*)$

- The **Hessian of Lagrangian**: $\mathbf{W}(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{H}(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i \hat{\mathbf{H}}_i \in \mathbb{R}^{n \times n}$

Theorem 1 (Necessary Conditions for NEP)

1. $\mathbf{g}(\mathbf{x}^*) = \mathbf{0}$
2. $\mathbf{Z}^{\mathsf{T}}(\mathbf{x}^*)\mathbf{G}(\mathbf{x}^*) = \mathbf{0} \Longleftrightarrow \mathbf{G}(\mathbf{x}^*) = \mathbf{A}^{\mathsf{T}}(\mathbf{x}^*)\boldsymbol{\lambda}^*$
3. $\mathbf{Z}^{\mathsf{T}}(\mathbf{x}^*)\mathbf{W}(\mathbf{x}^*, \boldsymbol{\lambda}^*)\mathbf{Z}(\mathbf{x}^*) \geq 0$ (i.e., PSD)

Theorem 2 (Sufficient Conditions for NEP)

1. $\mathbf{g}(\mathbf{x}^*) = \mathbf{0}$
2. $\mathbf{Z}^{\mathsf{T}}(\mathbf{x}^*)\mathbf{G}(\mathbf{x}^*) = \mathbf{0} \Longleftrightarrow \mathbf{G}(\mathbf{x}^*) = \mathbf{A}^{\mathsf{T}}(\mathbf{x}^*)\boldsymbol{\lambda}^*$
3. $\mathbf{Z}^{\mathsf{T}}(\mathbf{x}^*)\mathbf{W}(\mathbf{x}^*, \boldsymbol{\lambda}^*)\mathbf{Z}(\mathbf{x}^*) > 0$

Inequality Constraints

- Nonlinear Inequality-Constrained Problem (NIP)

$$\min_{\mathbf{x}} F(\mathbf{x})$$

s.t.

$$\mathbf{g}(\mathbf{x}) = \mathbf{0}$$

$$\mathbf{h}(\mathbf{x}) \leq \mathbf{0}$$

- Lagrangian Function

$$L(\mathbf{x}, \boldsymbol{\lambda}) = F(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i \mathbf{g}_i(\mathbf{x}) + \sum_{i=1}^{p} \mu_i \mathbf{h}_i(\mathbf{x})$$

- Define the following quantities at optimal solution:
  - q = number of active constraints
  - $\mathbf{a}(\mathbf{x}^*)$ = vector of all active constraints
  - $\boldsymbol{\nu}$ = vector of Lagrange multipliers corresponding to $\mathbf{a}(\mathbf{x}^*)$
- Furthermore, define the following quantities:

- ▷ **Jacobian** of the *active* constraints as

$$\mathbf{A}(\mathbf{x}) = \frac{\partial \mathbf{a}}{\partial \mathbf{x}} \in \mathbb{R}^{q \times n}$$

- ▷ **Hessian** of the *active* constraints,

$$\hat{\mathbf{H}}_i(\mathbf{x}) = \frac{\partial^2 a_i}{\partial \mathbf{x}^2} \in \mathbb{R}^{n \times n}, \quad (i = 1, \ldots, q)$$

  where $\mathbf{A}_i(\mathbf{x})$ is the i^{th} column of $\mathbf{A}(\mathbf{x})$

- ▷ A matrix $\mathbf{Z}(\mathbf{x})$ such that

$$\mathbf{Z}(\mathbf{x}^*) \in \mathbb{R}^{n \times q} \Longrightarrow \mathbf{Z}^{\mathsf{T}} \in \mathbb{R}^{q \times n}$$

  where the columns of $\mathbf{Z}(\mathbf{x}^*)$ form a basis for the set of vectors orthogonal to the rows of $\mathbf{A}(\mathbf{x}^*)$

- ▷ The **Hessian of the Lagrangian Function**

$$\mathbf{W}(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{H}(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i \hat{\mathbf{H}}_i \in \mathbb{R}^{n \times n}$$

- Associate the Following Two Sets with the Inequality Constraints
  - ▷ $\mathcal{A}(\mathbf{x}) = \{j \mid h_j(\mathbf{x}) = 0\}$
  - ▷ $\mathcal{A}'(\mathbf{x}) = \{j \mid h_j(\mathbf{x}) < 0\}$

Theorem 3 (Necessary Conditions for NIP)

1. $\mathbf{g}(\mathbf{x}^*) = \mathbf{0}$ and $\mathbf{h}(\mathbf{x}^*) \leq \mathbf{0}$
2. $\mathbf{Z}^\mathsf{T}(\mathbf{x}^*)\mathbf{G}(\mathbf{x}^*) = \mathbf{0} \Leftrightarrow \mathbf{G}(\mathbf{x}^*) = \mathbf{A}^\mathsf{T}(\mathbf{x}^*)\boldsymbol{\lambda}^*$
3. $\mu_i^* \geq 0$
4. $\mathbf{Z}^\mathsf{T}(\mathbf{x}^*\mathbf{W}(\mathbf{x}^*, \boldsymbol{\lambda}^*)\mathbf{Z}(\mathbf{x}^*) \geq 0$

Theorem 4 (Sufficient Conditions for NIP)

1. $\mathbf{g}(\mathbf{x}^*) = \mathbf{0}$ and $\mathbf{h}(\mathbf{x}^*) \leq \mathbf{0}$
2. $\mathbf{Z}^\mathsf{T}(\mathbf{x}^*)\mathbf{G}(\mathbf{x}^*) = \mathbf{0} \Leftrightarrow \mathbf{G}(\mathbf{x}^*) = \mathbf{A}^\mathsf{T}(\mathbf{x}^*)\boldsymbol{\lambda}^*$
3. $\mu_i^* \geq 0$
4. $\mathbf{Z}^\mathsf{T}(\mathbf{x}^*\mathbf{W}(\mathbf{x}^*, \boldsymbol{\lambda}^*)\mathbf{Z}(\mathbf{x}^*) > 0$

Numerical Solution of General NLP

- General NLP Must Be Solved Numerically
- Commonly Used Approaches for Solving NLPs
  - ▷ Sequential Quadratic Programming (SQP)
    - ○ Approximate Lagrangian as a quadratic
    - ○ Linearize Constraints
    - ○ Make progress to optimal solution based on a **merit function**
  - ▷ Interior Point (Barrier)
    - ○ Equality Constraints Enforced Using Quadratic Penalty Term
    - ○ Inequality Constraints Managed via a **Barrier Function**
    - ○ Sequence of Problems Solved Using Homotopy Parameter
- Notation
  - ▷ $\mathbf{x}$: Optimization Variables
  - ▷ $\boldsymbol{\lambda}$: Lagrange Multipliers of *equality* Constraints
  - ▷ $\boldsymbol{\mu}$: Lagrange Multipliers of *inequality* Constraints

Sequential Quadratic Programming (SQP)

- Choose an initial guess $(\mathbf{x}.\boldsymbol{\lambda}, \boldsymbol{\mu})$
- Approximate L by a Quadratic, i.e.,

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \approx \mathbf{g}^{\mathsf{T}}\mathbf{p} + \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p}$$

- Linearize Constraints

$$\begin{aligned}\left[\frac{\partial \mathbf{g}}{\partial \mathbf{x}}\right]_{\mathbf{x}_k} \mathbf{p}_k &= \mathbf{g}_k \\ \left[\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right]_{\mathbf{x}_k} \mathbf{p}_k &\leq \mathbf{h}_k\end{aligned}$$

- Equations Form a **Quadratic Program** (QP)
- QP Solved to Determine
  - ▷ Search direction, $\mathbf{p}$
  - ▷ Estimates of active constraints and Lagrange multipliers
- Globalization Strategy: Use a **Merit Function**
  - ▷ Used to maintain progress toward optimal solution

- At optimal solution, merit function = cost function
- Types of merit functions
  - Penalty functions
  - Barrier functions

Penalty Functions

- Let

$$\mathbf{c}(\mathbf{x}) = \begin{bmatrix} \mathbf{g}(\mathbf{x}) \\ \mathbf{h}(\mathbf{x}) \end{bmatrix}$$

- Penalty Function:

$$P(\mathbf{x}, \rho) = F(\mathbf{x}) + \frac{\rho}{2}\mathbf{c}^{\mathsf{T}}(\mathbf{x})\mathbf{c}(\mathbf{x})$$

where ρ is a penalty parameter.

Algorithm 4 (Penalty Function Method)

1. If $\mathbf{x}_k$ satisfies the optimality conditions, then terminate.
2. Using the current iterate $\mathbf{x}_k$, solve the unconstrained subproblem

$$\min_{\mathbf{x}\in\mathbb{R}^n} P(\mathbf{x}, \rho_k)$$

3. Set $\rho_{k+1} > \rho_k$, increment counter $k \to k+1$, and return to step 1

- Can be shown that

$$\lim_{\rho\to\infty} \mathbf{x}^*(\rho) = \mathbf{x}^*$$

Penalty Function Method (Continued)

- Issues with Standard Penalty Function
  - ▷ As $\rho \to \infty$ Unconstrained Problem Becomes Difficult to Solve
  - ▷ Need to Overcome Ill-Conditioning
- Augmented Lagrangian Penalty Function

$$\begin{aligned} P(\mathbf{x}) &= L(\mathbf{x}, \lambda, \mu) + \frac{\rho}{2}\mathbf{c}^{\mathsf{T}}(\mathbf{x})\mathbf{c}(\mathbf{x}) \\ &= F(\mathbf{x}) - \lambda^{\mathsf{T}}\mathbf{c}(\mathbf{x}) + \frac{\rho}{2}\mathbf{c}^{\mathsf{T}}(\mathbf{x})\mathbf{c}(\mathbf{x}) \end{aligned}$$

- Improvements of Augmented Lagrangian Penalty Function
  - ▷ Avoids Ill-Conditioning of Penalty Function
  - ▷ Converges for Finite Value of ρ
  - ▷ However, Choosing "reasonable" Value of ρ is Difficult

Barrier Function Method

- Quadratic Penalty Functions
  - ▷ Generate Infeasible Iterates
  - ▷ Inappropriate When Necessary to Maintain Feasibility
  - ▷ Solution: Use a *Feasible-Point* Method
  - ▷ Barrier Method is Such an Approach
- Idea of a Barrier Function
  - ▷ Choose a Function that Prevents Infeasibilities
  - ▷ Common to Choose a *Logarithmic* Barrier Function, i.e.,

$$B(\mathbf{x}, r) = F(\mathbf{x}) - r \sum_{i=1}^{m} \ln c_i(\mathbf{x})$$

  where r is the *Barrier parameter*

- Can Be Shown That

$$\lim_{r \to 0} \mathbf{x}^*(r) = \mathbf{x}^*$$

Well-Known NLP Solvers

- NPSOL
  - Dense NLP Solver
  - Good for Problem Up To A Few Hundred Variables
- SNOPT
  - Sparse NLP Solver: Uses SQP Quasi-Newton Method
  - Capable of Solving Problems with Thousands of Variables & Constraints
- IPOPT
  - Also a Sparse NLP Solver: Uses a Barrier Method
  - Capable of Using Either Quasi-Newton or Newton Method
  - Capable of Solving Problems with Thousands of Variables & Constraints
- Will Show Examples Using SNOPT and IPOPT

Example Using SNOPT

- Consider the Following Optimization Problem

$$\text{minimize } F(\mathbf{x}) = (x_1 - x_2)^2 + (x_2 - x_3)^3 + (x_3 - x_4)^4 + (x_4 - x_5)^4$$

subject to

$$\begin{aligned} x_3 + x_4 + x_5 &\geq 3 \\ x_1 + x_2^2 + x_3^3 &= 3 \\ x_1 + x_2 &\geq 1 \\ x_2 - x_3^2 + x_4 &= 1 \\ x_1 x_5 &= 1 \end{aligned}$$

- Constraint Jacobian and Objective Function Sparsity Pattern

	1	2	3	4	5
1	0	0	1	1	1
2	1	1	1	0	0
3	1	1	0	0	0
4	0	1	1	1	0
5	1	0	0	0	1
6	1	1	1	1	1

- Observations
  - 12 Out of 30 Derivatives Are Zero
  - Percentage of Nonzeros = 18/30 = 60%
- Can Take Advantage of Sparsity in SNOPT
- Problem Solved More Efficiently Using Sparse Linear Algebra

Code for SNOPT Example

- Main Function Code

```
function [x,F,xmul,Fmul,INFO] = hsmain()

snprint('hsmain.out');

hsmain.spc = which('hsmain.spc');
snspec (hsmain.spc);

snseti('Major Iteration limit', 250);

[x,xlow,xupp,xmul,xstate,Flow,Fupp,Fmul,Fstate,ObjAdd,ObjRow,    ...
 A,iAfun,jAvar,iGfun,jGvar] = snoptExampleData;

[x,F,xmul,Fmul,INFO]= snsolve( x, xlow, xupp, xmul, xstate,    ...
       Flow, Fupp, Fmul, Fstate,       ...
       ObjAdd, ObjRow, A, iAfun, jAvar,...
       iGfun, jGvar, 'snoptExampleFun');

snprint off; % Closes the file and empties the print buffer


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [x,xlow,xupp,xmul,xstate,Flow,Fupp,Fmul,Fstate,ObjAdd,ObjRow, ...
          A,iAfun,jAvar,iGfun,jGvar] = snoptexampledata()

neF    = 6; % Number of Functions (Including Objective Function)
n      = 5; % Number of Variables
Obj    = 6; % Objective Function Row
ObjRow = 6; % Objective Function Row

% Assign the Sparsity Pattern for the Non-Constant Elements
%              Column
```

```
%            | 1   2    3    4    5
%            +----------------------
%         1  |
%         2  |     6    7
%         3  |
%         4  |          8
%         5  | 9                   10
%     row 6  | 1   2    3    4    5    Objective row
%
G = [ Obj,    1;
      Obj,    2;
      Obj,    3;
      Obj,    4;
      Obj,    5;
        2,    2;
        2,    3;
        4,    3;
        5,    1;
        5,    5 ];

iGfun = G(:,1); jGvar = G(:,2);

% Assign the Sparsity Pattern for the Constant Derivatives
%              Column
%            | 1   2    3    4    5
%            +----------------------
%         1  |          4    5    6
%         2  | 1
%         3  | 7   8
%         4  |     2         3
%         5  |
%     row 6  |                        Objective row
%
A = [ 2,  1,  1;
      4,  2,  1;
      4,  4,  1;
      1,  3,  1;
```

```
      1,  4,  1;
      1,  5,  1;
      3,  1,  1;
      3,  2,  1 ];

iAfun = A(:,1); jAvar = A(:,2); A = A(:,3);

ObjAdd = 0;

% Initial Guess of Solution

x = [  2;
       sqrt(2) - 1 ;
       sqrt(2) - 1 ;
       2;
       0.5   ];

% Set Lower and Upper Bounda on Variables and Constraints
xlow   = -Inf*ones(n,1);
xupp   =  Inf*ones(n,1);
xstate =  zeros(n,1);
xmul   =  zeros(n,1);

Flow   = -Inf*ones(neF,1);
Fupp   =  Inf*ones(neF,1);

Flow(1) = 3;
Flow(2) = 3;
Fupp(2) = 3;
Flow(3) = 1;
Flow(4) = 1;
Fupp(4) = 1;
Flow(5) = 1;
Fupp(5) = 1;

Fmul    = zeros(neF,1);
Fstate  = zeros(neF,1);
```


- # Code to Compute Functions

```
 function [F,G] = snoptExampleFun(x)
% Computes the Functions for the SNOPT Example

% Part 1:  Define the Nonlinear Part of Functions and Derivatives for SNOPT Example
F = [    0;
       x(2)*x(2) + x(3)*x(3)*x(3);
         0;
      -x(3)*x(3);
       x(1)*x(5);
      (x(1)-x(2))^2 + (x(2)-x(3))^3 + (x(3)-x(4))^4 + (x(4)-x(5))^4 ];

% Part 2:  Compute Derivatives of Functions
Obj = 6;  % Objective Function is the 6th Row
G = [ Obj,    1,   2*(x(1)-x(2));
      Obj,    2,   3*(x(2)-x(3))^2 - 2*(x(1)-x(2));
      Obj,    3,   4*(x(3)-x(4))^3 - 3*(x(2)-x(3))^2;
      Obj,    4,   4*(x(4)-x(5))^3 - 4*(x(3)-x(4))^3;
      Obj,    5,  -4*(x(4)-x(5))^3;
        2,    2,   2* x(2);
        2,    3,   3* x(3)^2;
        4,    3,  -2* x(3);
        5,    1,   x(5);
        5,    5,   x(1)  ];
G = G(:,3);
```

Example Using IPOPT

- Consider the Following Optimization Problem

$$\text{minimize } x_3 + x_1 x_4 \sum_{i=1}^{3} x_i$$

subject to the constraints

$$\begin{aligned} x_i & \geq 1, \quad (i = 1, \ldots, 4) \\ x_i & \leq 5, \quad (i = 1, \ldots, 4) \\ x_1 x_2 x_3 x_4 & \geq 25 \\ \textstyle\sum_{i=1}^{4} x_i^2 & = 40 \end{aligned}$$

- To Use IPOPT in Newton Mode, Must Supply
  - ▷ Objective Function, Gradient, Constraints, and Jacobian
  - ▷ Hessian of NLP Lagrangian
  - ▷ Sparsity Pattern for Jacobian and Hessian
- This Example: Will Show How to Supply These Functions in IPOPT

Code for IPOPT Example

```
function [x, info] = exampleipopt
% Test IPOPT Problem Using Matlab Interface
% Reference:  Hock and Schittowski (1981), Test Examples for
% Nonlinear Programming Codes.  Lecture Nodes in Economics and
% Mathematical Systems, Vol. 187, Springer-Verlag.
x0  = [1 5 5 1]; % Initial Guess of Solution
xlb = [1 1 1 1]; % Lower Bound on Decision Variables
xub = [5 5 5 5]; % Upper Bound on Decision Variables
clb = [25 40];   % Lower Bound on Constraints
cub = [inf 40];  % Upper Bound on Constraints
options.lb = xlb;
options.ub = xub;
options.cl = clb;
options.cu = cub;

% Initialize the dual point.
options.zl     = [1 1 1 1];
options.zu     = [1 1 1 1];
options.lambda = [1 1];

% Set the IPOPT options.
options.ipopt.mu_strategy = 'adaptive';
options.ipopt.tol         = 1e-7;

% The callback functions.
funcs.objective         = @objective;
funcs.constraints       = @constraints;
funcs.gradient          = @gradient;
funcs.jacobian          = @jacobian;
funcs.jacobianstructure = @jacstructure;
funcs.hessian           = @hessian;
funcs.hessianstructure  = @hessstructure;
```

```
% Run IPOPT.
[x info] = ipopt(x0,funcs,options);

% ----------------------------------------------------------------------
function f = objective(x)

f = x(1)*x(4)*sum(x(1:3)) + x(3);

% ----------------------------------------------------------------------
function g = gradient (x)
g = [ x(1)*x(4) + x(4)*sum(x(1:3))
      x(1)*x(4)
      x(1)*x(4) + 1
      x(1)*sum(x(1:3)) ];

% ----------------------------------------------------------------------
function c = constraints(x)

c = [prod(x);sum(x.^2)];

% ----------------------------------------------------------------------
function J = jacobian(x)

J =  sparse([prod(x)./x; 2*x]);

% ----------------------------------------------------------------------
function S = jacstructure

S = sparse(ones(2,4));

% ----------------------------------------------------------------------
function H = hessian (x, sigma, lambda)

H = sigma*[ 2*x(4)             0      0   0;
            x(4)               0      0   0;
            x(4)               0      0   0;
            2*x(1)+x(2)+x(3)  x(1)  x(1)  0 ];
```



```
H = H + lambda(1)*[    0          0         0         0;
                    x(3)*x(4)     0         0         0;
                    x(2)*x(4) x(1)*x(4)     0         0;
                    x(2)*x(3) x(1)*x(3) x(1)*x(2)     0  ];
H = H + lambda(2)*diag([2 2 2 2]);
H = sparse(H);

function HS = hessstructure

HS = sparse(tril(ones(4)));
```

Summary

- Nonlinear Optimization
  - Foundation for *Function* Optimization
  - Deals with Discrete (Finite-Dimensional) Problem
  - Formulated as a Nonlinear Programming Problem (NLP)
  - Candidate Optimal Solutions are Called *Extrema*
- Determination of an Extreme Solution
  - Compute First Derivative of Augmented Cost Function
  - Set All Independent Terms in Variation to Zero
- In Practice, NLPs Solved Numerically via Iteration
  - SQP Methods
  - Barrier Methods
  - Trust-Region Methods
- Previously Developed Sophisticated NLP Software Has Been Described

Part IV: Numerical Methods for Optimal Control

Overview

- Parts I and II of Course
  - Optimal Control Theory
  - Nonlinear Optimization
- This Part of Course
  - We Develop Numerical Methods Using Parts I and II
  - Numerical Methods Consist of Three Parts
    - Solution of Differential Equations
    - Application of Methods for Root-Finding
    - Application of Nonlinear Optimization
- Categories of Numerical Methods for Optimal Control
  - Indirect Methods: Combine Differential Equations with Root-Finding
  - Direct Methods: Combine Differential Equations with Nonlinear Optimization

Overview of Indirect Methods

- Indirect Methods
  - ▷ Use Optimal Control Theory
  - ▷ Develop First-Order Optimality Conditions from Calculus of Variations (Part I of Course)
  - ▷ Attempt to Solve Hamiltonian Boundary-Value Problem (HBVP)
  - ▷ Recall State-Costate Dynamics

$$(\dot{\mathbf{x}}, \dot{\boldsymbol{\lambda}}) = (\mathcal{H}_{\boldsymbol{\lambda}}, -\mathcal{H}_{\mathbf{x}})$$

  - ▷ Boundary Conditions on State and Costate

$$\begin{aligned}
\boldsymbol{\phi}(\mathbf{x}(t_0, t_0, \mathbf{x}(t_f), t_f) &= \mathbf{0} \\
\boldsymbol{\lambda}(t_0) &= -\frac{\partial \Phi}{\partial \mathbf{x}(t_0)} + \left[\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{x}(t_0)}\right]^{\mathsf{T}} \boldsymbol{\nu} \\
\boldsymbol{\lambda}(t_f) &= \frac{\partial \Phi}{\partial \mathbf{x}(t_f)} - \left[\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{x}(t_f)}\right]^{\mathsf{T}} \boldsymbol{\nu}
\end{aligned}$$

Overview of Direct Methods

- Calculus of Variation is *Not* Used $\Longrightarrow$ Optimality conditions are *not derived*
- Instead
  - State and/or Control Parametrized Using Function Approximation
  - Dynamics are Integrated Either Explicitly or Implicitly (Will Explain)
  - Cost functional Replaced with Integral Approximation (Quadrature)
  - Infinite-Dimensional Optimal Control Problem $\Longrightarrow$ Nonlinear Programming Problem (NLP)
  - NLP solved by well-known methods

Commonly Used Indirect Methods

- Shooting Method
  - ▷ Guess Made of Unknown Initial and/or Terminal Conditions
  - ▷ State-Costate Dynamics Integrated from t_0 to t_f
  - ▷ Error in Initial and Terminal Boundary Conditions Assessed
  - ▷ Iterated Until All Boundary Conditions are Satisfied
- Multiple-Shooting Method
  - ▷ Time Interval Divided Into Segments
  - ▷ Guess Made of Unknown Initial Conditions at Start of *Each Segment*
  - ▷ State-Costate Dynamics Integrated on Each Segment
  - ▷ Errors at Each Segment Interface and Boundary Conditions Assessed
  - ▷ Iterated Until All Boundary Conditions are Satisfied

Indirect Shooting Method

- Consider the Following HBVP

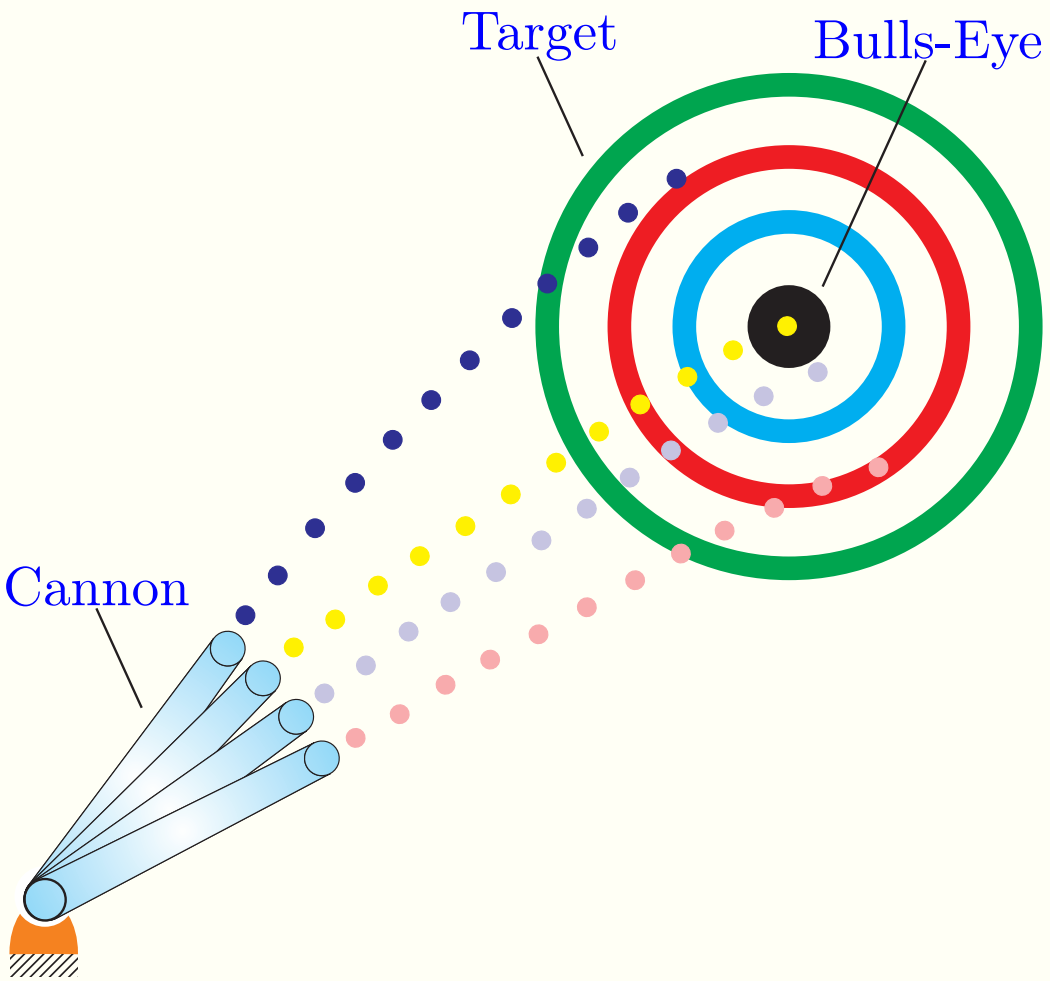
$$\dot{\mathbf{p}} = \mathbf{G}(\mathbf{p}), \quad \mathbf{p} = (\mathbf{x}, \boldsymbol{\lambda}), \quad \mathbf{G}(\mathbf{p}) = (\mathcal{H}_{\boldsymbol{\lambda}}, -\mathcal{H}_{\mathbf{x}})$$

  with boundary conditions

$$\boldsymbol{\phi}(\mathbf{p}(t_0), t_0, \mathbf{p}(t_f), t_f) = \mathbf{0}$$

- Observation: Do Not Have complete Set of Initial and/or Terminal Conditions
- Iterative Procedure (Indirect Shooting)
  - ▷ Guess Made of of $\mathbf{p}(t_0) = \tilde{\mathbf{p}}_0$ and, possibly, t_0 and t_f
  - ▷ Integrate $\dot{\mathbf{p}} = \mathbf{G}(\mathbf{p})$ From t_0 to t_f
  - ▷ At t_f, have $\tilde{\mathbf{p}}_f$ from Integration
  - ▷ Compute $\boldsymbol{\phi}(\tilde{\mathbf{p}}(t_0), t_0, \tilde{\mathbf{p}}(t_f), t_f)$ and Assess Proximity to Zero
  - ▷ Iterate on Unknown Components of $\tilde{\mathbf{p}}(t_0)$, t_0, & t_f Until $\|\boldsymbol{\phi}\| < \epsilon$
- Observe That Error in $\boldsymbol{\phi}$ Depends Upon $\tilde{\mathbf{p}}_0$; Can Solve Using Newton method

Schematic of Indirect Shooting Method



Appeal of Indirect Shooting Method

- Shooting is a Simple *Root Finding* Problem
- Number of Variables In Root-Finding Problem is *Small*
  - ▷ Only Number of Unknown Initial Conditions
  - ▷ Low-Dimensional Problem $\longrightarrow$ Requires Little Memory on a Computer
- Why Not Always Use Indirect Shooting?
  - ▷ Hamiltonian Systems are *Unstable* in Either Direction of Time
  - ▷ Error in Unknown Initial Conditions Grows When Integrating t_0 to t_f
  - ▷ Even for Simple Problem, Indirect Shooting Can Break Down
- Next Slides: Example of Difficulties Using Indirect Shooting

Hamiltonian Boundary-Value Problem for Simple LQ Problem

- State-Costate Dynamics and Boundary Conditions

$$\begin{bmatrix} \dot{x} \\ \dot{\lambda} \end{bmatrix} = \begin{bmatrix} a & -\frac{b^2}{r} \\ -q & -a \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} \quad , \quad \begin{bmatrix} x(0) = x_0 \\ x(t_f) = x_f \end{bmatrix}$$

- General Solution of First-Order LTI System

$$\mathbf{p}(t) = c_1 e^{\mu_1 t} \mathbf{w}_1 + c_2 e^{\mu_2 t} \mathbf{w}_2$$

where

$$\begin{aligned} \mathbf{p} &= \begin{bmatrix} x \\ \lambda \end{bmatrix} \\ \mu_1, \mu_2 &= \text{Eigenvalues of } \mathbf{A} \\ \mathbf{w}_1, \mathbf{w}_2 &= \text{Eigenvectors of } \mathbf{A} \end{aligned}$$

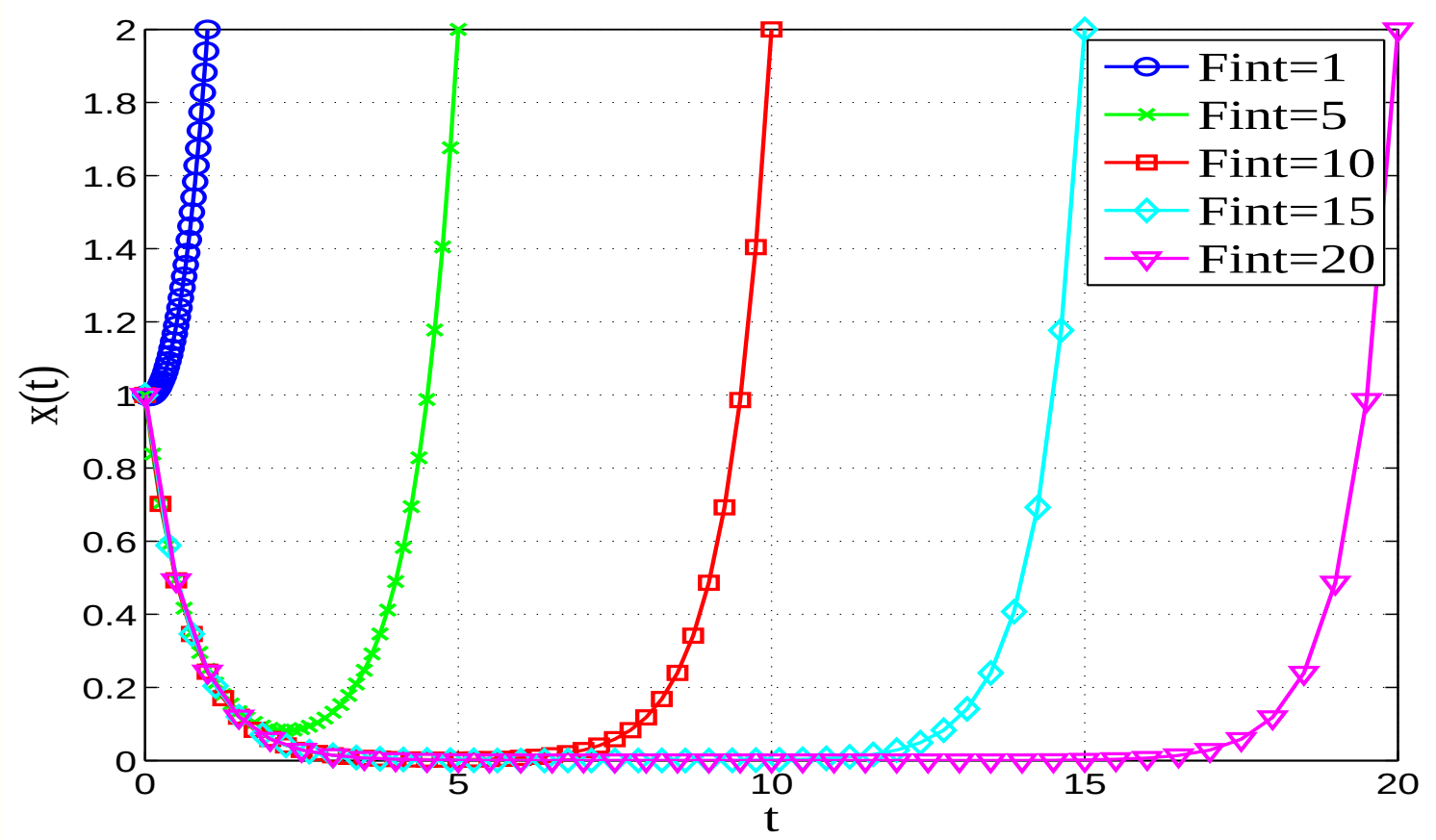
- Eigenvalues of System Matrix $\mathbf{A}$

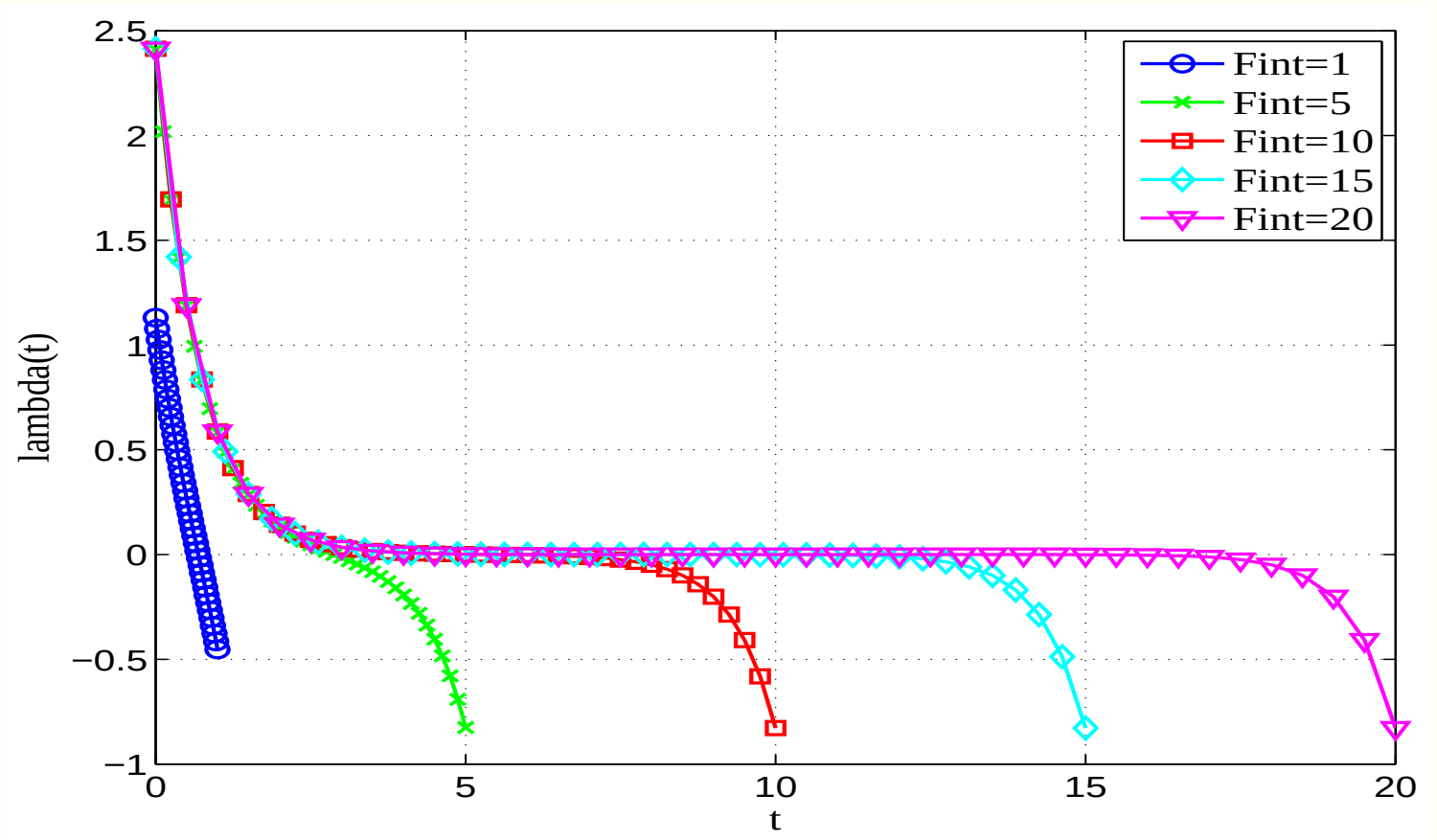
$$\det\left[\mu\mathbf{I} - \mathbf{A}\right] = \det \begin{bmatrix} \mu - a & \frac{b^2}{r} \\ q & \mu + a \end{bmatrix} = \mu^2 - a^2 - \frac{qb^2}{r} = 0$$

- Assuming That $q > 0$ and $r > 0$, We Obtain

$$\mu = \pm\sqrt{a^2 + \frac{qb^2}{r}} \longrightarrow \text{Real Eigenvalues with Opposite Sign}$$

- Consider Solution for $a = b = q = r = 1$, $t_0 = 0$, $x(t_0) = 1$, $t_f = (1, 5, 10, 15, 20)$, and $x(t_f) = 2$





- Key points
  - **Both** state and costate have decaying and growing behavior
  - Growth and decay show up as t_f increases

Solution of LQ Problem Using Indirect Shooting

- Two missing boundary conditions: $\lambda(0)$ and $\lambda(t_f)$
- Can develop iteration as follows:
  - ▷ Integrate state-costate dynamics either
    - ○ From t_0 to $t_f \Longrightarrow$ need to iterate on $\lambda(0)$
    - ○ From t_f to $t_0 \Longrightarrow$ need to iterate on $\lambda(t_f)$
  - ▷ Issue
    - ○ **A** has both a positive and negative eigenvalue
    - ○ State-costate dynamics are unstable in **either direction of time**
- Numerical Problems Arise When t_f is Large
  - ▷ Growing Component of Dynamics Due to Eigenvalue at $\mu = +\omega$
  - ▷ Let $\Delta\lambda_0$ be Error in Guess of $\lambda(0)$
  - ▷ Then Error in Initial Condition Given as

$$\Delta\mathbf{p}(0) = \begin{bmatrix} 0 \\ \Delta\lambda_0 \end{bmatrix}$$

- ▷ Resulting Error at t_f

$$\Delta\mathbf{p}(t_f) = e^{\mathbf{A}t_f}\Delta\mathbf{p}(0)$$

- ▷ Now $e^{\mathbf{A}t_f}$ has a Mode That Grows Like $e^{\omega t}$
- ▷ Sufficiently large $t_f \longrightarrow e^{\omega t_f}$ will Result in Overflow on a Computer
- ▷ Note: Overflow will Eventually occur t_f Sufficiently Large, Even for *extremely small* $\Delta\lambda(0)$
- ▷ **Numerical Solution of a HBVP Can be Extremely Difficult**

- Overcoming Difficulties of Indirect Shooting: **Indirect Multiple-Shooting Method**

Example: Orbit-Transfer Problem

- Hamiltonian Boundary-Value Problem

$$
\begin{aligned}
\dot{r} &= u \\
\dot{u} &= \frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T}{m}\sin\alpha^* \\
\dot{v} &= -\frac{uv}{r} + \frac{T}{m}\cos\alpha^* \\
\dot{m} &= -\frac{T}{g_0 I_{sp}} \\
\dot{\lambda}_r &= \lambda_u\left[\frac{v^2}{r^2} - \frac{2\mu}{r^3}\right] - \lambda_v\frac{uv}{r^2} \\
\dot{\lambda}_u &= -\lambda_r + \lambda_v\frac{v}{r} \\
\dot{\lambda}_v &= -\lambda_u\frac{2v}{r} + \lambda_v\frac{u}{r} \\
\dot{\lambda}_m &= \lambda_u\left[\frac{T}{m^2}\sin\alpha^*\right] + \lambda_v\left[\frac{T}{m^2}\cos\alpha^*\right]
\end{aligned}
$$

$$
\begin{aligned}
r(t_0) &= r_0 \quad , \quad u(t_0) = 0 \\
v(t_0) &= \sqrt{\mu/r_0} \quad , \quad m(t_0) = m_0 \\
r(t_f) &= r_f \quad , \quad u(t_f) = 0 \\
v(t_f) &= \sqrt{\mu/r_f} \quad , \quad \lambda_m(t_f) = 0
\end{aligned}
$$

- Notes:
  - ▷ Initial and Terminal Conditions are *Incomplete*
  - ▷ Half of the initial conditions and terminal conditions are specified
- Solved Numerically in MATLAB Using Indirect Shooting Method

Code for Orbit Transfer (Uses SNOPT)

```
% Minimum-Time Orbit Transfer Example
% Solves Hamiltonian Boundary-Value Problem
% Using SNOPT with Feasible Point Option
clear snoptcmex
clear all
close all
clc
warning off
mu = 1;
r0 = 1;
u0 = 0;
v0 = 1;
m0 = 1;
rf = 1.5;
uf = 0;
vf = sqrt(mu/rf);

lam0 = [-1; -1; 1; 1];
tf0 = 2;
guess = [lam0; tf0];
global xpath iters;
iters = 0;
xlow = -50*ones(5,1);
xupp =  50*ones(5,1);
Flow = [-Inf; zeros(4,1)];
Fupp = [+Inf; zeros(4,1)];
snset('Feasible Point');
userfun = 'orbitObjandCons';
snscreen on
[xout,F,inform,xmul,Fmul] = snopt(guess,xlow,xupp,Flow,Fupp,userfun);
p0 = [r0; u0; v0; m0; xout(1:4)];
pf = [rf; uf; vf; xout(1:4); 0];
tspan = linspace(0,xout(5),40);
```

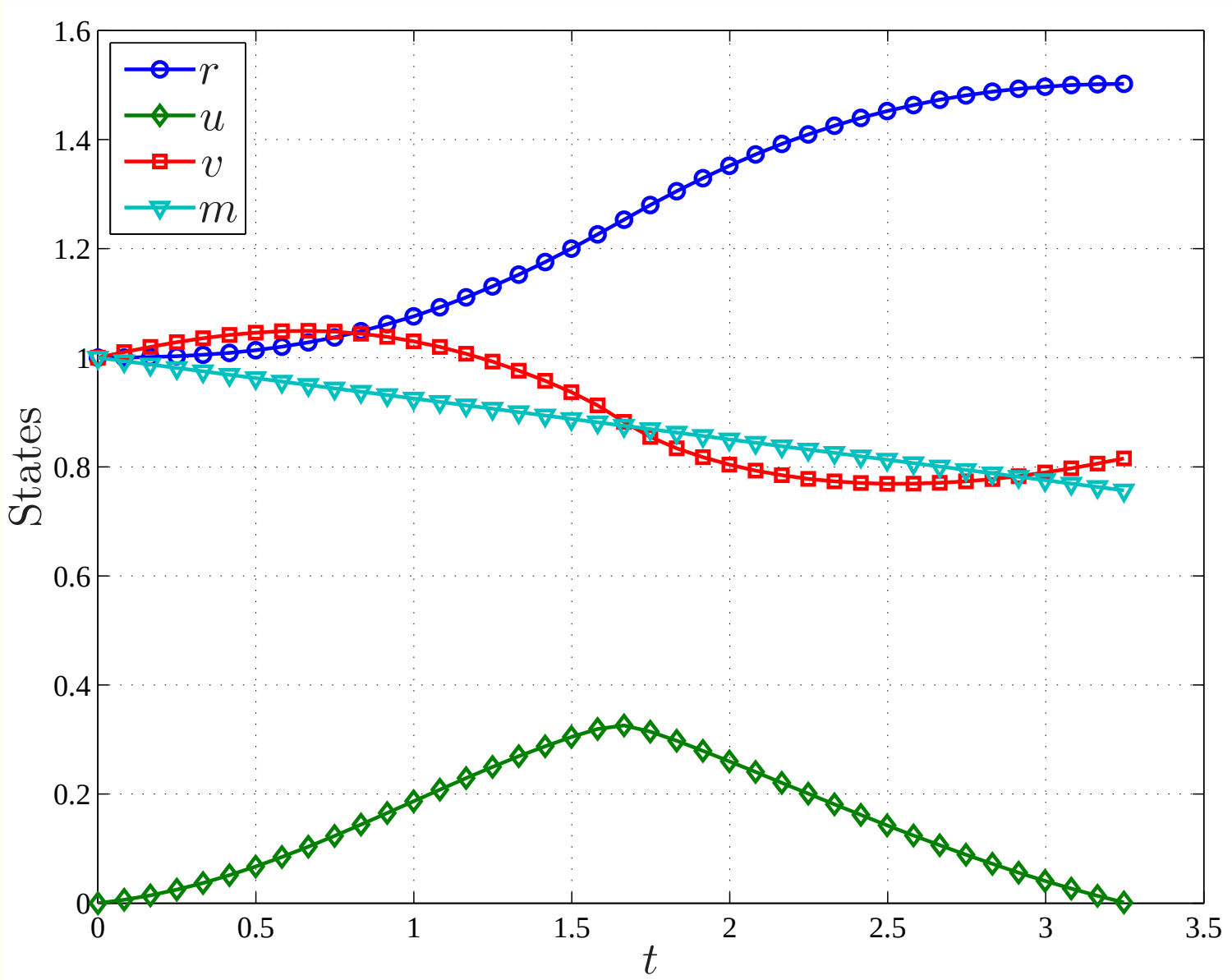
```
odeopts = odeset('RelTol',1e-12,'AbsTol',1e-12);
[tout,pout]=ode15s(@orbitode_bvp,tspan,p0,odeopts);
alpha = unwrap(atan2(pout(:,6),pout(:,7)))*180/pi;
orbitIndirectTraj = [tout pout alpha];
save orbitIndirectTraj.mat orbitIndirectTraj;

figure(1);
pp = plot(tout,pout(:,1),'-o',tout,pout(:,2),'-d',tout,pout(:,3),'-s',tout,pout(:,4),'-v');
set(pp,'LineWidth',1.5,'MarkerSize',6);
xl = xlabel('Time');
yl = ylabel('States');
ll = legend('r','u','v','m','Location','NorthWest');
set(xl,'FontSize',12);
set(yl,'FontSize',12);
set(ll,'FontSize',16);
set(gca,'FontName','Times','FontSize',12);
grid on;
print -depsc2 orbitRaisingIndirectShootingStates.eps

figure(2);
pp = plot(tout,pout(:,5),'-o',tout,pout(:,6),'-d',tout,pout(:,7),'-s',tout,pout(:,8),'-v');
set(pp,'LineWidth',1.5,'MarkerSize',6);
xl = xlabel('Time');
yl = ylabel('Costates');
ll = legend('lamr','lamu','lamv','lamm','Location','NorthWest');
set(xl,'FontSize',12);
set(yl,'FontSize',12);
set(ll,'FontSize',16);
set(gca,'FontName','Times','FontSize',12);
grid on;
print -depsc2 orbitRaisingIndirectShootingCostates.eps

figure(3);
pp = plot(tout,alpha,'-o');
set(pp,'LineWidth',1.5,'MarkerSize',6);
xl = xlabel('Time');
yl = ylabel('Control');
```

```
set(xl,'FontSize',12);
set(yl,'FontSize',12);
set(gca,'FontName','Times','FontSize',12);
grid on;
print -depsc2 orbitRaisingIndirectShootingControl.eps

function C = orbitObjandCons(x);
mu = 1;
r0 = 1;
u0 = 0;
v0 = sqrt(mu/r0);
m0 = 1;
rf = 1.5;
uf = 0;
vf = sqrt(mu/rf);
pfinal = [rf; uf; vf];
pinitial =[r0; u0; v0; m0];
p = [r0; u0; v0; m0; x(1:4)];
target = [pfinal; 0];
tf = x(5);
tspan = linspace(0,tf,100);
opts = odeset('RelTol',1e-3);
[tout,pout]=ode15s(@orbitode_bvp,tspan,p,opts);
error = target-[pout(end,1:3) pout(end,8)].';
C = [0; error];

function pdot = orbitode_bvp(t,p);

mu = 1;
T = 0.1405;
mass_flow_rate = 0.0749;

r = p(1);
u = p(2);
v = p(3);
m = p(4);
lamr = p(5);
lamu = p(6);
```

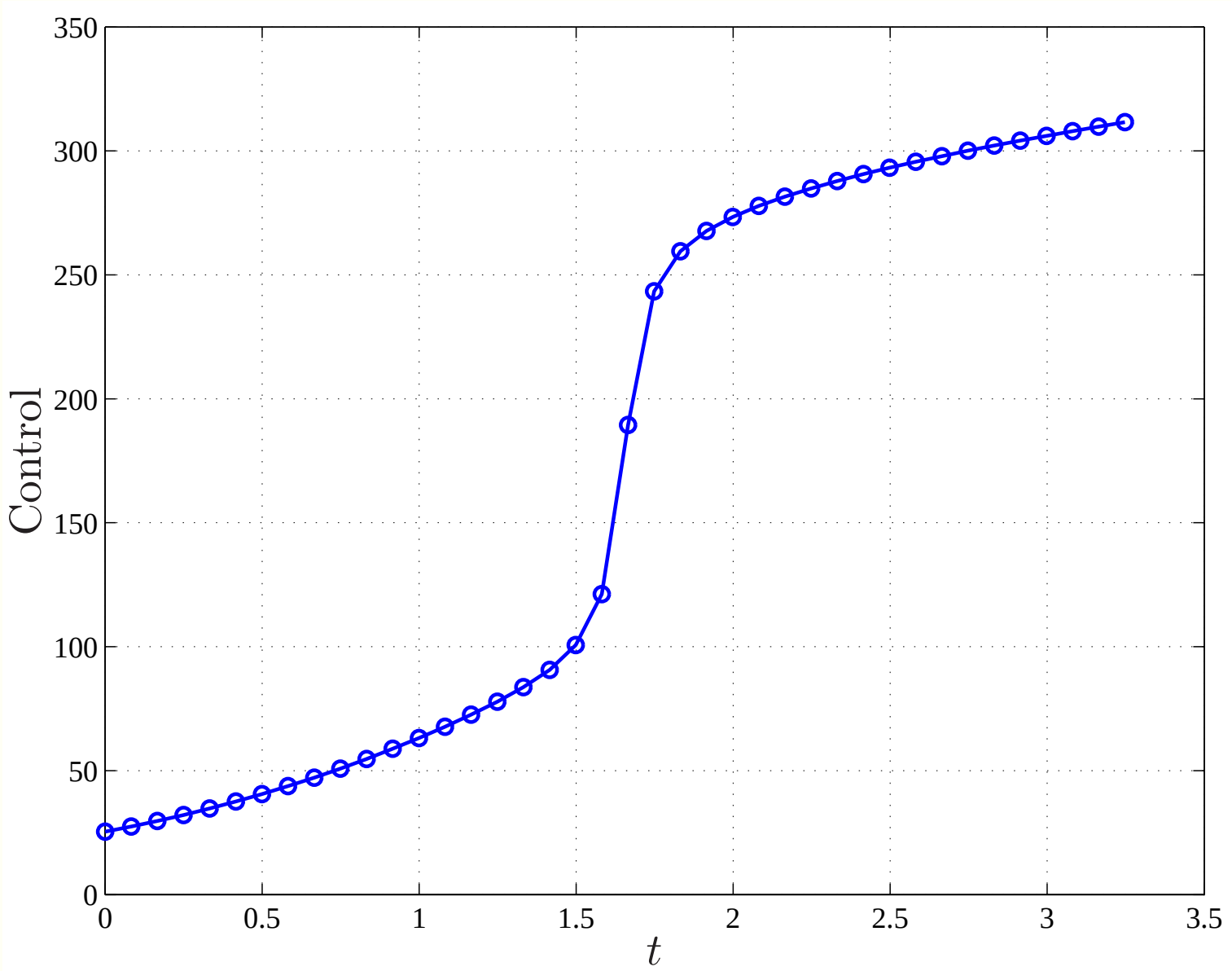


```
lamv = p(7);
lamm = p(8);

%sinalpha = lamu./sqrt(lamu.^2+lamv.^2);
%cosalpha = lamv./sqrt(lamu.^2+lamv.^2);
alpha     = atan2(lamu,lamv);
sinalpha  = sin(alpha);
cosalpha  = cos(alpha);
pdot      = zeros(8,1);
rdot      = u;
udot      = v.^2./r-mu./r.^2+(T./m).*sinalpha;
vdot      = -(u.*v)./r+(T./m).*cosalpha;
mdot      = -mass_flow_rate;
lamrdot   = lamu.*(v.^2./r.^2-2*mu./r.^3)-lamv.*(u.*v)./r.^2;
lamudot   = -lamr+lamv.*v./r;
lamvdot   = -lamu.*(2.*v./r)+lamv.*(u./r);
lammdot   = lamu.*(T.*sinalpha./m.^2)+lamv.*(T.*cosalpha./m.^2);
pdot(1:8) = [rdot; udot; vdot; mdot; lamrdot; lamudot; lamvdot; lammdot];
```

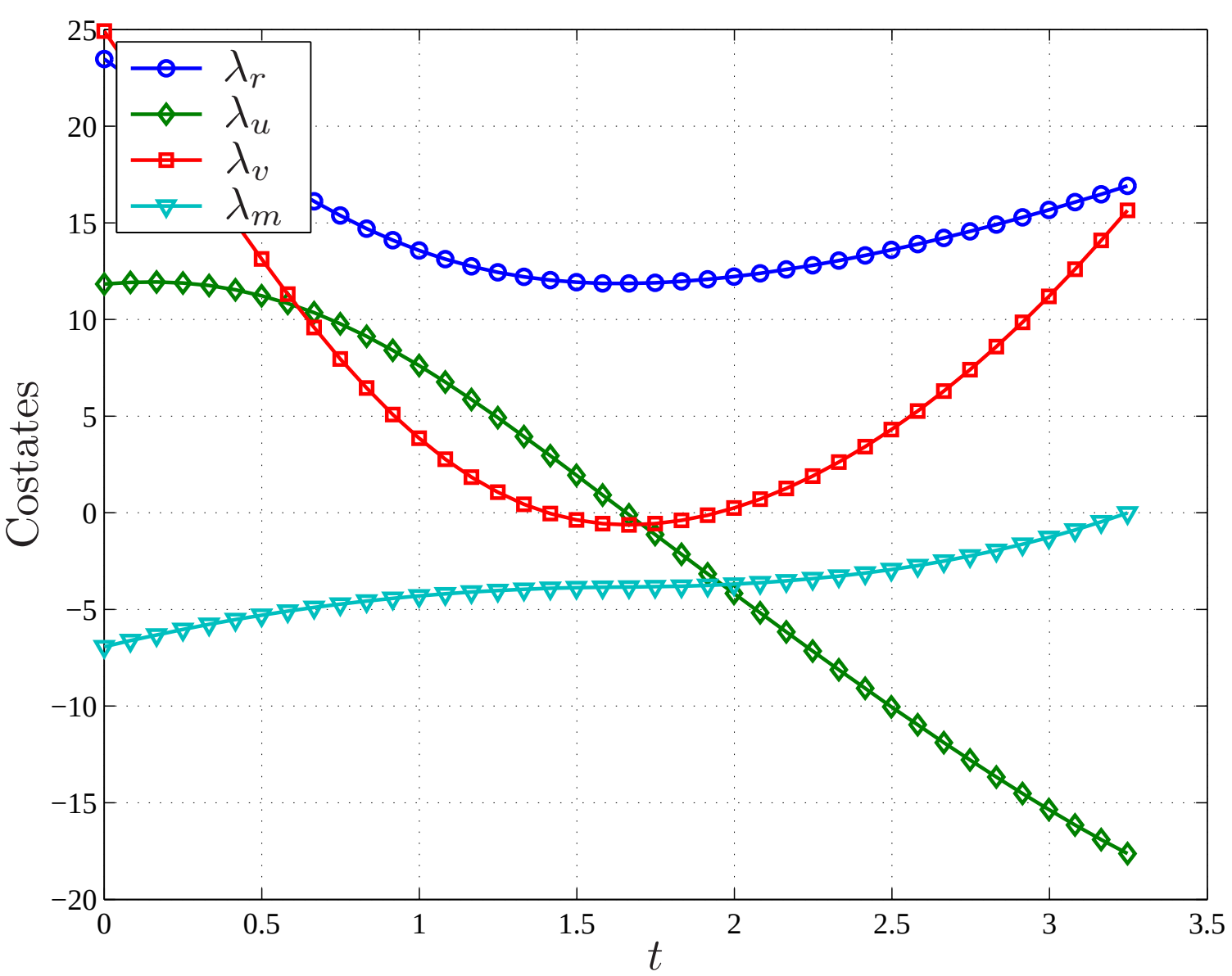
State Solution of Orbit Transfer Problem



Control Solution of Orbit Transfer Problem



Costate Solution of Orbit Transfer Problem



The Indirect Multiple-Shooting Method

- Basics of Multiple-Shooting Method
  - ▷ Break time interval into subintervals
  - ▷ Shoot over each subinterval
- Implementation of a Multiple-Shooting Method
  - ▷ Let $I_s = [t_s, t_{s+1}]$, $(s = 1, \ldots, S)$
  - ▷ Ensure following conditions

$$\begin{aligned} \bigcap_{s=1}^{S} I_s &= \emptyset \\ \bigcap_{s=1}^{S} I_s &= [t_0, t_f] \end{aligned}$$

  - ▷ Integrate over each subinterval I_s
  - ▷ Need to enforce continuity constraints at interior interfaces

$$\begin{aligned} \mathbf{x}(t_s^-) &= \mathbf{x}(t_s^+) \\ \boldsymbol{\lambda}(t_s^-) &= \boldsymbol{\lambda}(t_s^+) \end{aligned}$$

Pros and Cons of Indirect Multiple-Shooting

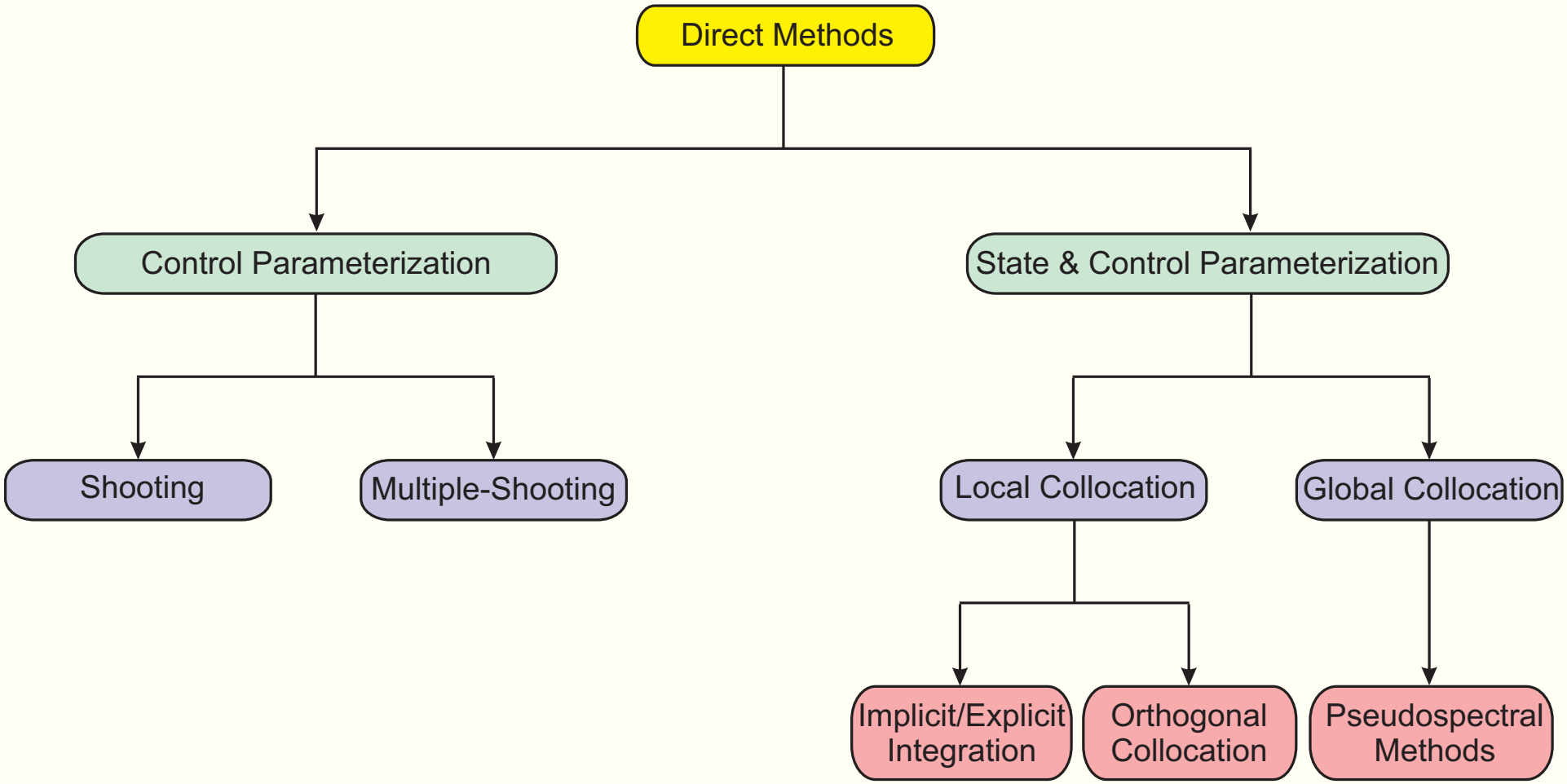
- Pros
  - ▷ Integration Performed Over Shorter Time Intervals
  - ▷ Does Not Allow Hamiltonian Dynamics to "Blow Up"
  - ▷ Can Overcome Difficulties Associated with Indirect Shooting
  - ▷ Increases Range of Problems
- Cons
  - ▷ Need to guess values at beginning of each subinterval
    - ○ $\mathbf{p}(t_s^+),\ (s = 2, \ldots, S-1)$ Become Additional Variables in Problem
    - ○ Total Set of Variables Increased With Each Added Subinterval
  - ▷ Many more variables and constraints in problem
  - ▷ Guessing Values of $\mathbf{p}(t_s^+),\ (s = 2, \ldots, S-1)$ Usually Not Intuitive
  - ▷ Root-Finding Problem of Much Higher-Dimension Than Indirect Shooting

Other General Problems with Indirect Methods

- Need to Derive First-Order Optimality Conditions
  - Costate Equations
  - Transversality Conditions
- Why is Deriving Such Conditions Problematic?
  - Functions May Be Very Complex
  - Some Functions May Not Even Have Analytic Forms (Table Interpolation)
  - Becomes Very Difficult for Practical Problem
- Multiple-Phase Problems
  - Dynamics and Path Constraints May Be Different in Each Phase
  - Extremely Cumbersome to Derive Conditions for Complex Problem
- Bottom Line: Don't Want to Have to Derive Optimality Conditions

Direct Methods

- Optimality conditions are *not derived*. Instead
  - State and/or Control Parametrized Using Function Approximation
  - Cost Approximated Using Quadrature
  - Optimal Control Problem Converted to an NLP
  - Infinite-dimensional problem $\Longleftrightarrow$ finite-dimensional problem
- Direct Methods Divided Into Two General Categories
  - Control Parametrization: Shooting and Multiple-Shooting
  - State & Control Parametrization: Collocation



- Control Parametrization: Simple, But Not As Capable
- State & Control Parametrization: More Capable, But More Involved
- Will Discuss Both Types of Methods

Direct Shooting Method

- Step 1: Choose a Function Approximation for Control

$$\mathbf{u}(t) = \sum_{k=0}^{N} \mathbf{c}_k \psi_k(t)$$

- Basis Functions $\psi_0(t), \ldots, \psi_N(t)$ Arbitrary
- Usually Choose Polynomials
- Step 2: Guess Unknown Coefficients $(\mathbf{c}_0, \ldots, \mathbf{c}_N)$
- Step 3: Integrate Dynamics from t_0 to t_f
- Iterate on Coefficients Until Cost Minimized and Constraints Satisfied

Example: Linear-Tangent Steering Problem

- Cost Function: $J = t_f$
- Dynamics

$$\begin{aligned} \dot{x}_1 &= x_3 \\ \dot{x}_2 &= x_4 \\ \dot{x}_3 &= a\cos(u) \\ \dot{x}_4 &= a\sin(u) \end{aligned}$$

- Initial Conditions: $(x_1(0), x_2(0), x_3(0), x_4(0)) = (0, 0, 0, 0)$
- Terminal Conditions: $(x_2(t_f), x_3(t_f), x_4(t_f)) = (5, 45, 0)$

Code for Linear Tangent Problem

```
global CONSTANTS

CONSTANTS.polynomialDegree = 3;
CONSTANTS.a = 100;
x10 = 0;
x20 = 0;
x30 = 0;
x40 = 0;
x2f = 5;
x3f = 45;
x4f = 0;
tfMin = 0.01;
tfMax = 100;
CONSTANTS.initialState  = [x10; x20; x30; x40];
CONSTANTS.terminalState = [x2f; x3f; x4f];
tfGuess = 10;
z0 = [ones(CONSTANTS.polynomialDegree+1,1); tfGuess];
zlow = [-10*ones(CONSTANTS.polynomialDegree+1,1); tfMin];
zupp = [+10*ones(CONSTANTS.polynomialDegree+1,1); tfMax];
Flow = [tfMin; zeros(size(CONSTANTS.terminalState))];
Fupp = [tfMax; zeros(size(CONSTANTS.terminalState))];
snscreen on
snprint('snoptmain.out');
snseti('Derivative Option',0);
snseti('Print Level',10);
[z,F,inform,zmul,Fmul] = snopt(z0,zlow,zupp,Flow,Fupp,'linearTangentObjandCons');
p0 = CONSTANTS.initialState;
tspan = [0 z(end)];
opts = odeset('RelTol',1e-3);
[tout,pout]=ode45(@linearTangentOde,tspan,p0,opts,z(1:end-1));
uout = polyval(z(1:end-1),tout);

figure(1);
```

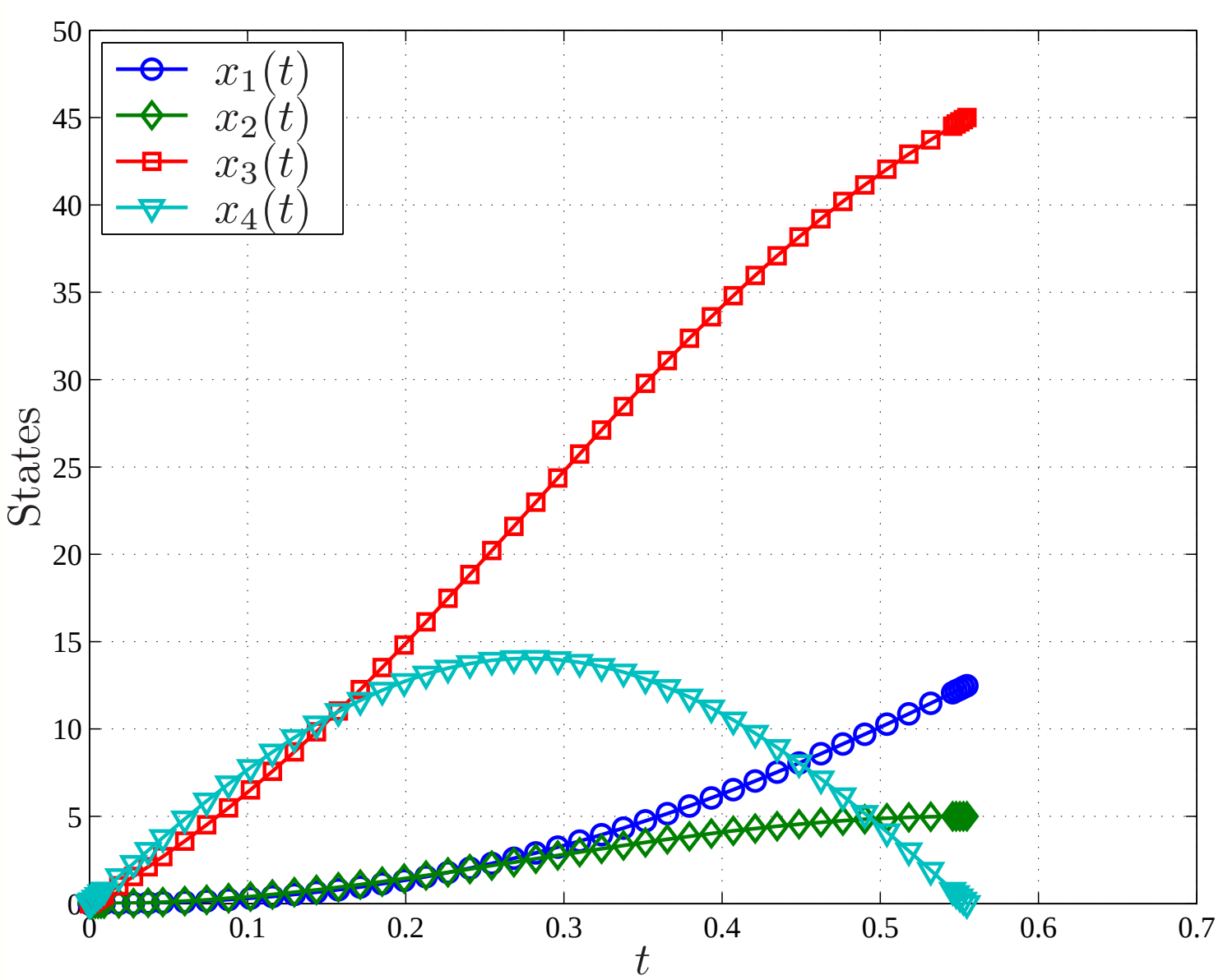
```
pp = plot(tout,pout(:,1),'-o',tout,pout(:,2),'-d',tout,pout(:,3),'-s',tout,pout(:,4),'-v');
set(pp,'LineWidth',1.5,'MarkerSize',8);
xl = xlabel('Time');
yl = ylabel('States');
ll = legend('State1(t)','State2(t)','State3(t)','State4(t)','Location','NorthWest');
set(xl,'FontSize',12);
set(yl,'FontSize',12);
set(ll,'FontSize',16);
set(gca,'FontName','Times','FontSize',12);
grid on;
print -depsc2 linearTangentDirectShootingState.eps

figure(2);
pp = plot(tout,uout,'-o');
set(pp,'LineWidth',1.5,'MarkerSize',8);
xl = xlabel('Time');
yl = ylabel('Control');
set(xl,'FontSize',12);
set(yl,'FontSize',12);
set(gca,'FontName','Times','FontSize',12);
grid on;
print -depsc2 linearTangentDirectShootingControl.eps
```

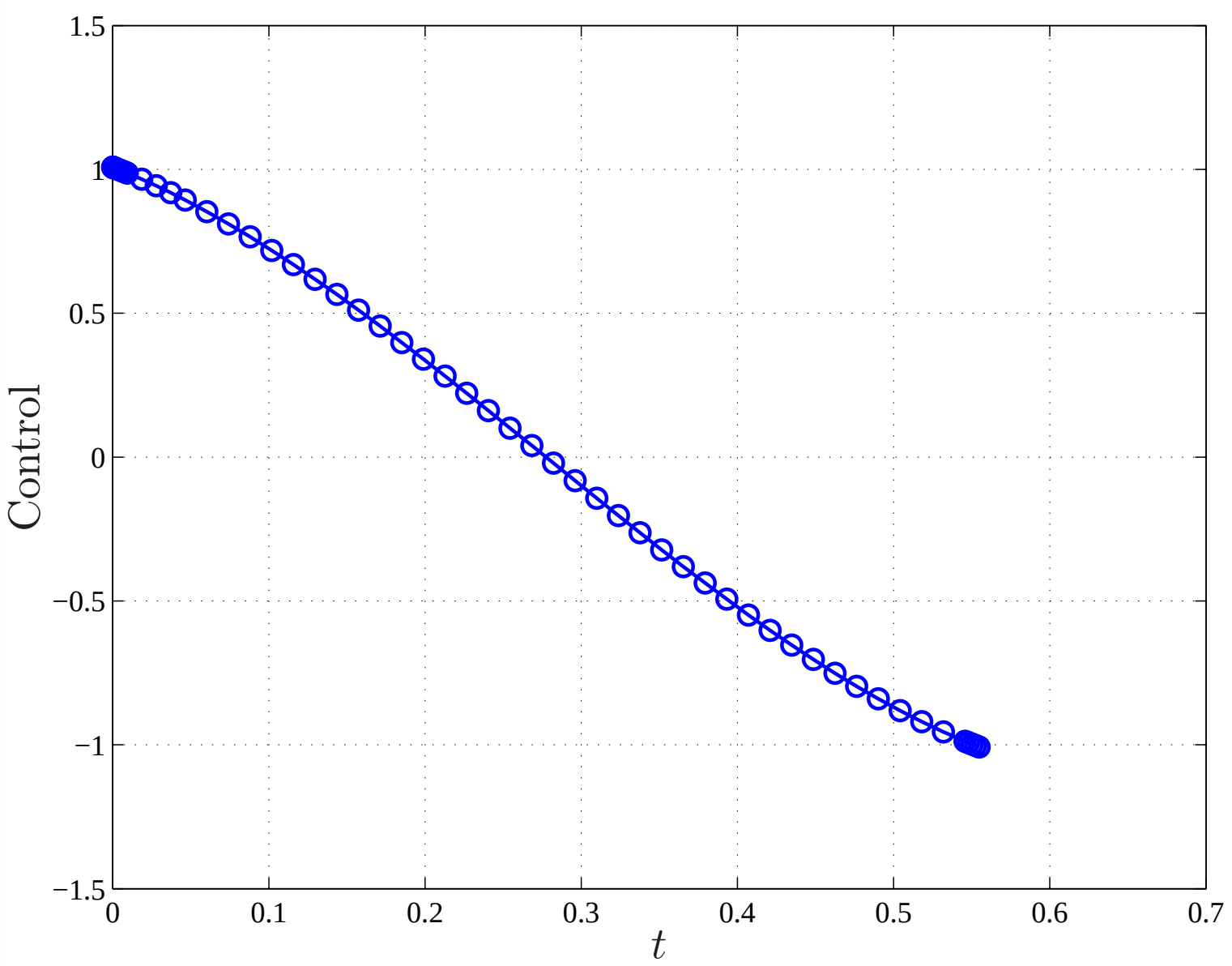
```
function C = linearTangentObjandCons(z);

global CONSTANTS

controlCoefs = z(1:end-1);
tf           = z(end);

p0 = CONSTANTS.initialState;
tspan = [0 tf];
opts = odeset('RelTol',1e-3);
[tout,pout]=ode45(@linearTangentOde,tspan,p0,opts,controlCoefs);
C = [tf; pout(end,2:4).'-CONSTANTS.terminalState];


function pdot=linearTangentOde(t,p,c);
```



```
global CONSTANTS

u = 0;
u = polyval(c,t);

p1dot = p(3);
p2dot = p(4);
p3dot = CONSTANTS.a*cos(u);
p4dot = CONSTANTS.a*sin(u);
pdot = [p1dot; p2dot; p3dot; p4dot];
```

Direct Shooting Solution of Linear-Tangent Problem





Example: Direct Shooting Solution of Orbit-Raising Problem

- Will Formulate Problem Using Two Control Parametrization
- Parametrization 1: Global Polynomial

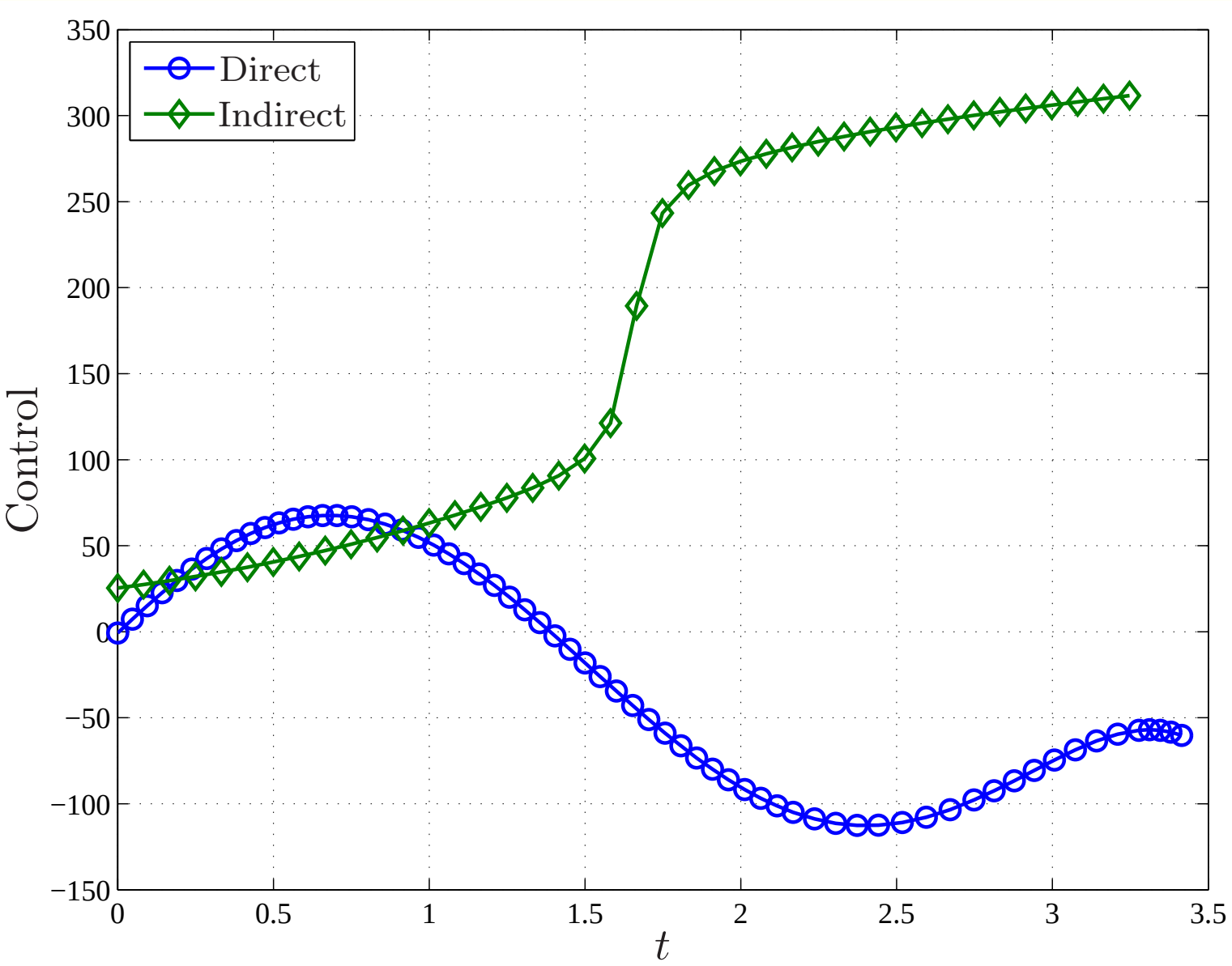
$$\beta(t) \approx \sum_{k=0}^{5} c_k t^k \longrightarrow \text{Degree 5 Polynomial}$$

- Parametrization 2: Arc-tangent of Ratio of Two Polynomials

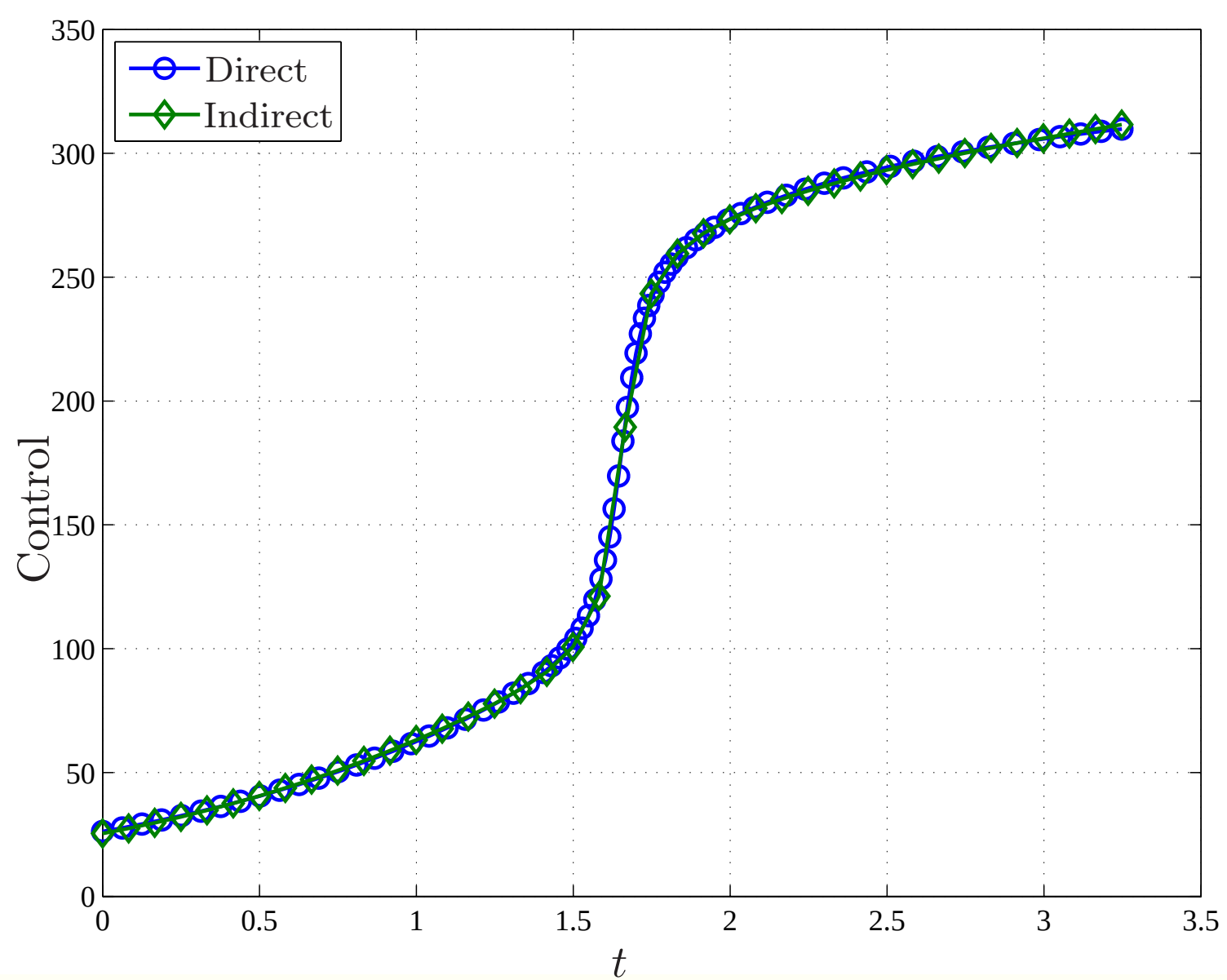
$$\begin{aligned} \beta(t) &\approx \tan^{-1}(n(t), d(t)) \\ n(t) &= \textstyle\sum_{k=0}^{5} n_k t^k \qquad \longrightarrow \text{Ratio of Two Degree 5 Polynomials} \\ d(t) &= \textstyle\sum_{k=0}^{5} d_k t^k \end{aligned}$$

- Objective of Comparison
  - ▷ Show How Quality of Direct Shooting Solution Depends on Control Parametrization
  - ▷ Need Some Insight to Use Appropriate Control Parametrization

Control Solution to Orbit-Raising Problem Using Polynomial Control



Control Solution to Orbit-Raising Problem Using $\tan^{-1}$ Control



Code for Direct Shooting Solution of Orbit-Raising Problem

```
clear snoptcmex
clear all
close all
global CONSTANTS numseg breaks nstates nterminal numcons degree control_type

control_type = 'polynomial';
% control_type = 'arctangent';
degree = 5;
numseg = 1;
nstates = 4;
nterminal = 3;
numcons = (numseg-1)*nstates+nterminal;
numvars = (numseg-1)*nstates+numseg*(degree+1)+1;
width = 1/numseg;
breaks = width:width:1;
CONSTANTS.mu = 1;
CONSTANTS.T  = 0.1405;
CONSTANTS.Ve = 1.8758;
r0 = 1; rf = 1.5;
u0 = 0; uf = 0;
v0 = 1; vf = sqrt(CONSTANTS.mu/rf);
m0 = 1;
rmin = 1;
rmax = 2;
umin = -0.1;
umax = 0.5;
vmin = 0.5;
vmax = 1.1;
mmax = 1.05;
mmin = 0.5;
stateLow = [rmin; umin; vmin; mmin];
stateUpp = [rmax; umax; vmax; mmax];
tfMin = 2;
```

```
tfMax = 3.6;
CONSTANTS.initialState  = [r0; u0; v0; m0];
CONSTANTS.terminalState = [rf; uf; vf];
tfGuess = 3;
coefsGuess = [zeros(degree,1); 0.2];
z0 = coefsGuess;
coefMax = 3;
zlow = -coefMax*ones(degree+1,1);
zupp =  coefMax*ones(degree+1,1);
Flow = [];
Fupp = [];
rr = [0.5; 0.5; 0.5; 0.5];
for i=2:numseg;
  z0 = [z0; rr; coefsGuess];
  zlow = [zlow; stateLow; -coefMax*ones(degree+1,1)];
  zupp = [zupp; stateUpp;  coefMax*ones(degree+1,1)];
  Flow = [Flow; zeros(nstates,1)];
  Fupp = [Fupp; zeros(nstates,1)];
end;
z0 = [z0; tfGuess];
zlow = [zlow; tfMin];
zupp = [zupp; tfMax];
Flow = [Flow; zeros(nterminal,1)];
Fupp = [Fupp; zeros(nterminal,1)];
Flow = [-inf; Flow];
Fupp = [inf; Fupp];
snscreen on
if isequal(control_type,'polynomial'),
    snprint('snoptmainShootingPolynomial.out');
else
    snprint('snoptmainShootingArctangent.out');
end;
snsetr('Optimality Tolerance',1e-5);
snsetr('Feasibility Tolerance',1e-5);
snseti('Derivative Option',0);
snseti('Print Level',10);
ObjAdd = 0;
```



```
ObjRow = 1;
A = [];
iAfun = []; jAvar = [];
JJ = ones(numcons+1,numvars);
[iGfun,jGvar]=find(JJ);
userfun = 'orbitRaisingObjandCons';
[z,F,inform,zmul,Fmul] = snopt(z0,zlow,zupp,Flow,Fupp,userfun,0,1,A,iAfun,jAvar,iGfun,jGvar);
p0 = CONSTANTS.initialState;
tfOptimal = z(end);
tfuse = tfOptimal*breaks;
opts = odeset('RelTol',1e-10);
istart = 1;
ifinish = degree+1;
controlCoefs = z(istart:ifinish);
tspan = [0:tfuse(1):tfuse(1)];
[tout,pout]=ode45(@orbitRaisingOde,tspan,p0,opts,controlCoefs);
ttot = tout;
ptot = pout;
if isequal(control_type,'polynomial'),
    utot = polyval(controlCoefs,ttot);
else
    lc = length(controlCoefs);
    numerator = polyval(controlCoefs(1:lc/2),tout);
    denominator = polyval(controlCoefs(lc/2+1:end),tout);
    utot = atan2(numerator,denominator);
end;
if 0,
for jj=2:numseg;
  istart = ifinish+1;
  ifinish = istart+nstates+degree;
  zuse = z(istart:ifinish);
  p0 = zuse(1:nstates);
  controlCoefs = zuse(nstates+1:nstates+degree+1);
  tspan = [tfuse(jj-1):(tfuse(jj)-tfuse(jj-1))/10:tfuse(jj)];
  [tout,pout]=ode45(@orbitRaisingOde,tspan,p0,opts,controlCoefs);
  % uout = polyval(controlCoefs,tout);
  numerator = polyval(controCoefs(1:lc/2,tout));
```



```
  denominator = polyval(controCoefs(lc/2+1:end,tout));
  uout = atan2(numerator,denominator);
  % uout = polyval(controlCoefs,tout);
  ttot = [ttot; tout];
  ptot = [ptot; pout];
  utot = [utot; uout];
end;
end;
utot = unwrap(utot)*180/pi;

figure(1);
pp = plot(ttot,ptot(:,1),'-o',ttot,ptot(:,2),'-d',ttot,ptot(:,3),'-s',ttot,ptot(:,4),'-v');
set(pp,'LineWidth',1.5,'MarkerSize',8);
xl = xlabel('Time');
yl = ylabel('States');
ll = legend('r','u','v','m','Location','Northwest');
set(xl,'FontSize',12);
set(yl,'FontSize',12);
set(ll,'FontSize',16);
set(gca,'FontName','Times','FontSize',12);
grid on;
if isequal(control_type,'polynomial'),
  print -depsc2 orbitRaisingDirectShootingStatePolynomial.eps
else
  print -depsc2 orbitRaisingDirectShootingStateArctangent.eps
end;

currdir = pwd;
cd('../../../Indirect/orbitRaising');
load orbitIndirectTraj.mat;
cd(currdir);
tindirect = orbitIndirectTraj(:,1);
uindirect = orbitIndirectTraj(:,end);
figure(2);
pp = plot(ttot,utot,'-o',tindirect,uindirect,'-d');
set(pp,'LineWidth',1.5,'MarkerSize',8);
xl = xlabel('Time');
```

```
yl = ylabel('Control');
ll = legend('Direct','Indirect','Location','Northwest');
set(xl,'FontSize',12);
set(yl,'FontSize',12);
set(ll,'FontSize',16);
set(gca,'FontName','Times','FontSize',12);
grid on;
if isequal(control_type,'polynomial'),
  print -depsc2 orbitRaisingDirectShootingControlPolynomial.eps
else
  print -depsc2 orbitRaisingDirectShootingControlArctangent.eps
end;

function C = orbitRaisingObjandCons(z);

global CONSTANTS numseg breaks nstates nterminal numcons degree

tf    = z(end);
tfuse = tf*breaks;
C = zeros(numcons+1,1);
opts = odeset('RelTol',1e-10);
p0 = CONSTANTS.initialState;
istart = 1;
ifinish = degree+1;
controlCoefs = z(istart:ifinish);
tspan = [0 tfuse(1)];
[tout,pout]=ode45(@orbitRaisingOde,tspan,p0,opts,controlCoefs);
lastStateSave = pout(end,:).';
lastDerivativeSave = orbitRaisingOde(tspan(end),pout(end,:).',controlCoefs);
err = [];
derr = [];
for jj=2:numseg;
  istart = ifinish+1;
  ifinish = istart+nstates+degree;
  zuse = z(istart:ifinish);
  p0 = zuse(1:nstates);
  err(:,jj-1) = p0-lastStateSave;
```



```
  controlCoefs = zuse(nstates+1:nstates+degree+1);
  tspan = [tfuse(jj-1) tfuse(jj)];
  dp0 = orbitRaisingOde(tspan(1),p0,controlCoefs);
  % derr(:,jj-1) = dp0-lastDerivativeSave;
  [tout,pout]=ode45(@orbitRaisingOde,tspan,p0,opts,controlCoefs);
  lastStateSave = pout(end,:).';
  lastDerivativeSave = orbitRaisingOde(tspan(end),pout(end,:).',controlCoefs);
end;
errtot = [err; derr];
Errorf = pout(end,1:3).'-CONSTANTS.terminalState;
C = [tf; errtot(:); Errorf];

function pdot=orbitRaisingOde(t,p,c);

global CONSTANTS control_type

r = p(1); u = p(2); v = p(3); m = p(4);

if isequal(control_type,'polynomial'),
    alpha = polyval(c,t);
else
    lc = length(c);
    numerator = polyval(c(1:lc/2),t);
    denominator = polyval(c(lc/2+1:end),t);
    alpha = atan2(numerator,denominator);
end;
rdot = u;
udot = v^2/r-CONSTANTS.mu/r^2+CONSTANTS.T*sin(alpha)/m;
vdot = -u*v/r +CONSTANTS.T*cos(alpha)/m;
mdot = -CONSTANTS.T/CONSTANTS.Ve;
pdot = [rdot; udot; vdot; mdot];
```

Lessons Learned from Direct Shooting

- Pros
  - No Need to Derive Complicated Optimality Conditions
  - Integrate Only State Dynamics, Not Costate Dynamics
  - Coding is Very Straightforward
- Cons
  - Solution Very Sensitive to Choice of Control
  - Without Insight Into Problem, Low-Quality Solution Produced
- Modification of Direct Shooting: Direct *Multiple* Shooting

Direct Multiple-Shooting

- Let t_0=Start Time of Phase and t_f=Terminus Time of Phase
- Divide Trajectory Into *Segments*, $t_0 < t_1 < t_2 < \cdots < t_{N-1} < t_N$ where $t_N = t_f$
- Points $t_0, t_1, t_2, \ldots, t_N$ Called *Grid Points* or *Mesh Points*
- Parametrize the Control in Each Segment As

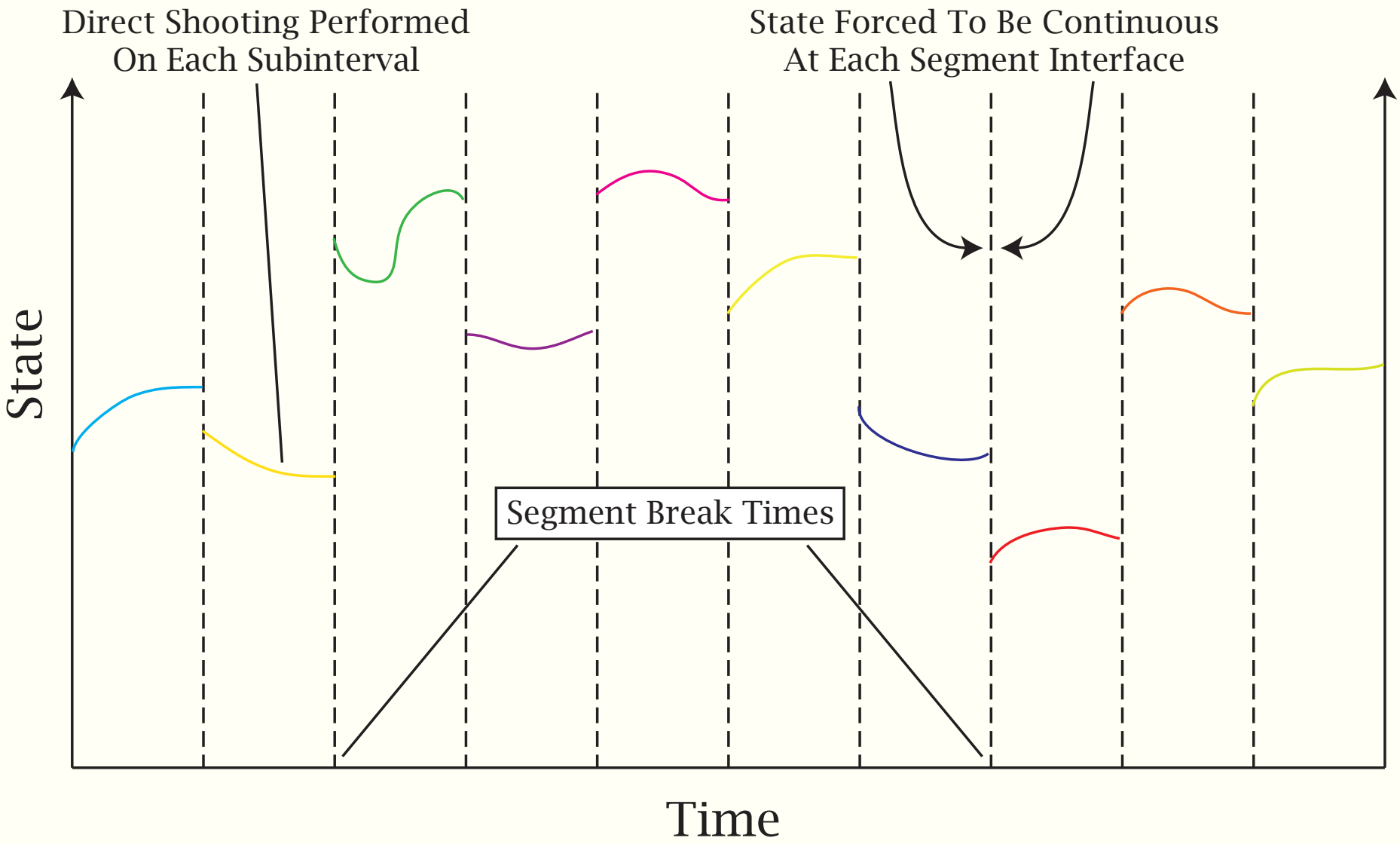
$$\mathbf{u}^{(s)}(t) = \sum_{k=0}^{N} \mathbf{c}_k^{(s)} \psi_k^{(s)}(t)$$

- Connect Segments with Continuity Constraint on State

$$\mathbf{x}(t_s^-) = \mathbf{x}(t_s^+), \quad s = 1, \ldots, S-1$$

- Total Cost = Sum of Cost in Each Segment
- Enforce Boundary Conditions in First and Last Segment

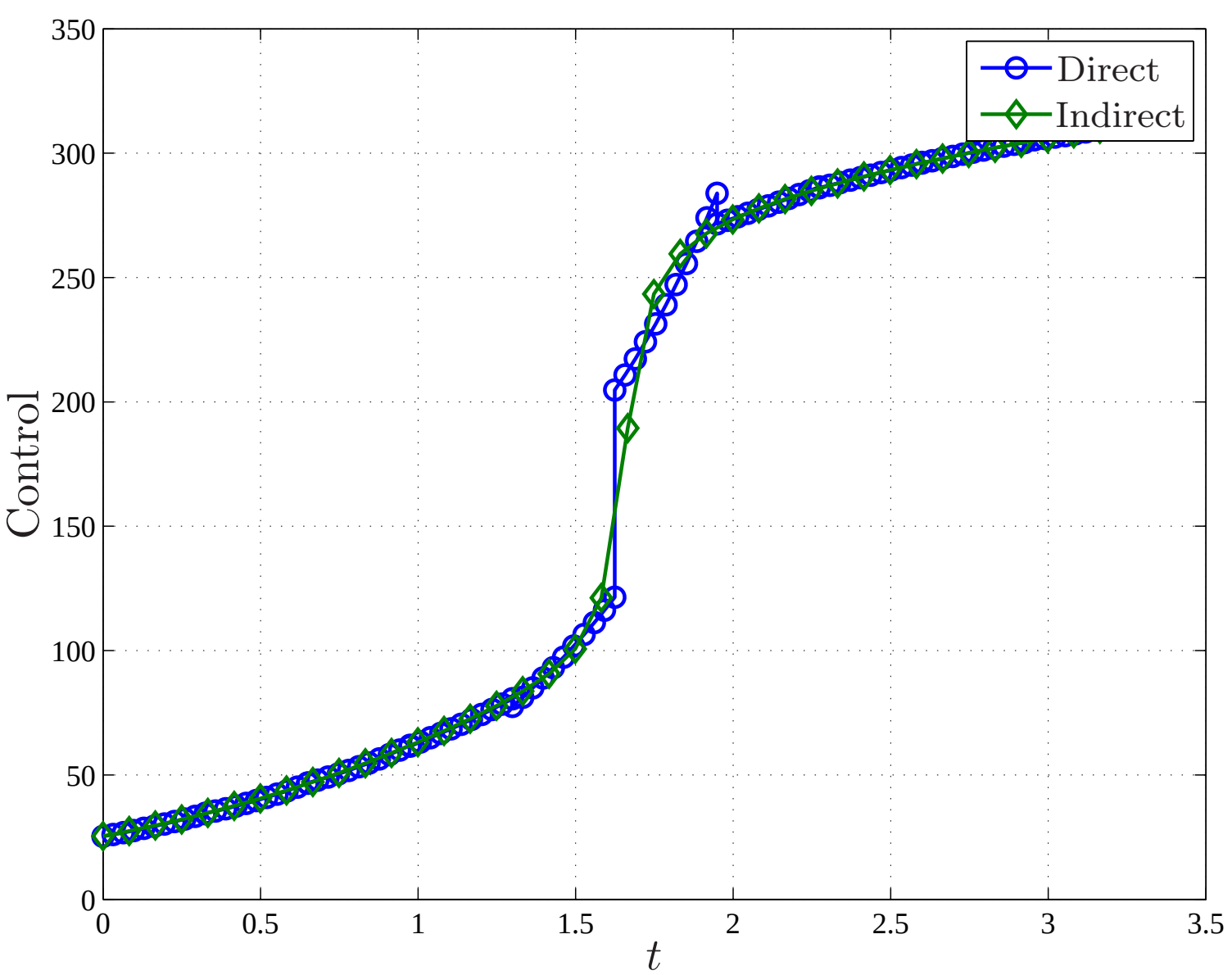
Schematic of Multiple-Shooting



Example: Orbit-Raising Problem

- Same Problem as Solved Earlier Using Indirect and Direct Shooting
- In This Example We Choose
  - Number of Segments = 10
  - Degree of Polynomial Approximation in Each Segment = 3
- Problem Divided Into Equal Width Segments

Direct Multiple-Shooting Solution to Orbit-Raising Problem



Comments on Direct Multiple-Shooting

- Improves Solution Over Naively Implemented Direct Shooting
- More Complex Than Direct Shooting
  - Need to Add Continuity Constraints on State at Segment Interfaces
  - Increases the Size of the Problem (More Variables, More Constraints)
  - May Obtain Discontinuous Control (Even if True Optimal Control is Continuous)
- Benefits of Direct Multiple Shooting
  - Improves the Quality of the Control over Direct Shooting
  - Allows for Possible Control Discontinuities

Limitations with Shooting Methods

- Generally Produces Low-Accuracy Solutions
- Computationally Inefficient
- May Not Converge to Solution for Inappropriate Control Approximations
- Serious Limiting Factors: Inclusion of Constraints
- Control Constraints: Difficult to Place Bounds on Control
- State Path Constraints
  - Need to Formulate Problem Using *Differential-Algebraic Equations* (DAEs)
  - DAEs difficult to Solve via Explicit Integration
  - Path Constraints Difficult to Include in Problem

Added Complexity: Phases

- Real Optimal Control Problems are Modeled Using *Phases*
- What is a Phase? $\Rightarrow$ a Distinct *Chunk* of the Trajectory
- How are Phases Implemented?
  - ▷ Divide Problem into *Phases*
  - ▷ Each Phase has
    - ○ Its Own Dynamics and Path Constraints
    - ○ Dynamics May Be *Different* from Phase to Phase
  - ▷ Phases are *Linked* Using Continuity Conditions on State

$$\boldsymbol{\psi}_{\min} \leq \boldsymbol{\psi}(\mathbf{p}_f^s, t_f^s, \mathbf{p}_0^{s+1}, t_0^{s+1}) \leq \boldsymbol{\psi}_{\max}$$

    where $s \in [1, \ldots, S]$ are the *phases* in the problem

- Why Phases Are Important
  - ▷ Consider the Problem of a Two-Stage Launch Vehicle
  - ▷ Stage 1: First Engine Burns; Dry Mass Jettisoned at End of Stage
  - ▷ Stage 2: Second Engine Burns; Dry Mass Jettisoned at End of Stage

  - ▷ Connection Between Stages: Position and Velocity Continuous; Mass Discontinuous
  - ▷ Problem Cannot Be Solved Using a Single Phase
- Shooting Methods Become *Even More Complex* With Multiple-Phase Problems
- Need Approach That Scales to Complex Problems
- **More Sophisticated Approach: Direct Collocation**

Direct Collocation Methods for Optimal Control

- Generally, Structure of Optimal Control Not Known *a Priori*
- Would Like a Computationally Efficient Method That Is Flexible
  - ▷ Does Not Require an a Priori Knowledge of Control
  - ▷ Provides Greater Accuracy As Compared with Shooting
- More Sophisticated Class of Direct Methods: Collocation
  - ▷ Collocation: *State and Control Parameterization* Methods
  - ▷ Allows for Very Flexible Formulation of Optimal Control Problem
    - ○ Multiple Phases
    - ○ Path Constraints (Equality and Inequality)
- In Collocation, Dynamics are *Not* Propagated
  - ▷ Instead, Differential Equations Converted to Algebraic Constraints
  - ▷ Leads to NLP: min $f(\mathbf{z})$ subject to $\mathbf{g}(\mathbf{z}) = 0$ and $\mathbf{h}_{\min} \leq \mathbf{h}(\mathbf{z}) \leq \mathbf{h}_{\max}$
  - ▷ Entire System Then Optimized *Simultaneously* for State and Control

Example of Collocation: Euler Forward

- Euler Forward Integration: Given (t_k, u_k, x_k), Determine x_{k+1}

$$x_{k+1} = x_k + hf(x_k, u_k, t_k)$$

- However, We Could Write Euler Forward As Follows

$$\begin{aligned} x_1 &= x_0 + hf(x_0, u_0, t_0) \\ x_2 &= x_1 + hf(x_1, u_1, t_1) \\ &\vdots \\ x_M &= x_{M-1} + hf(x_{M-1}, u_{M-1}, t_{M-1}) \end{aligned}$$

- Rearranging, We Have a System of Simultaneous Equations

$$\begin{aligned} x_1 - x_0 - hf(x_0, u_0, t_0) &= 0 \\ x_2 - x_1 - hf(x_1, u_1, t_1) &= 0 \\ &\vdots \\ x_M - x_{M-1} - hf(x_{M-1}, u_{M-1}, t_{M-1}) &= 0 \end{aligned}$$

- Approximation of Cost

$$J \approx \Phi(x_0, t_0, x_M, t_M) + h \sum_{k=1}^{M-1} \mathcal{L}[x_i, u_i, t_i]$$

- Boundary Conditions

$$\phi(x_0, t_0, x_M, t_M) = 0$$

- Result: We End Up with an NLP of the Form

$$\text{minimize } f(\mathbf{z})$$

  Subject to

$$\mathbf{g}(\mathbf{z}) = \mathbf{0}$$

- NLP Variables: $\mathbf{z} = (x_0, \ldots, x_M, u_0, \ldots, u_{M-1})$
- NLP Constraints
  - ▷ System of Equations Resulting from Euler Discretization
  - ▷ Boundary Conditions

Euler Forward Provides Key Features of Collocation

- M=Number of Intervals $\approx$ Number of Time Steps
- As M Gets Large
  - ▷ Problem Divided Into Increasingly More Subintervals
  - ▷ Number of NLP Variables $\approx 2M$
- More General Result
  - ▷ Suppose We Have n States and m Controls
  - ▷ Number of NLP Variables $\approx M(n+m)$
  - ▷ Suppose $M = 1000$, $n = 6$, and $m = 3 \longrightarrow M(n+m) \approx 9000$
- Complex Problem
  - ▷ Thousands to Tens of Thousands Variables and Constraints
  - ▷ NLP Using Direct Collocation is *Huge*
- Question: Is Direct Collocation Computationally Viable?

Why Use a Collocation Method?

- At First Glance
  - ▷ Seems As If We've Made Problem Harder
  - ▷ Many Equations in Many Unknowns
- As It Happens, Collocation Methods Are
  - ▷ Computationally Efficient
  - ▷ Robust to Initial Guesses
- Why?
  - ▷ Advanced NLP Solvers Designed to Solve *Large Sparse* Problems
  - ▷ Using Collocation Leads to a Large Sparse NLP
- What is a Sparse NLP?
  - ▷ Problem May Have Many Variables and Constraints
  - ▷ But Derivatives of Functions Have Large Number of Zeros
  - ▷ Sparsity Makes It Possible to Develop Efficient Iteration Methods

Collocation Formulation

- Divide Phase into *Segments*

$$t_0 < t_1 < t_2 < \cdots < t_{N-1} < t_N$$

  where $t_N = t_f$

- Points $t_0, t_1, t_2, \ldots, t_N$ Called *Grid Points* or *Mesh Points*
- Now Consider a Segment $I_s = [t_k, t_{k+1}]$
  - ▷ Treat state and control as variables in discretization
  - ▷ Differential equations replaced by *algebraic* approximations
  - ▷ All constraints will have the general form

$$\mathbf{c}_L \leq \mathbf{c}(\mathbf{y}) \leq \mathbf{c}_U$$

    where

$$\mathbf{c}(\mathbf{y}) = \left[\boldsymbol{\zeta}_1, \boldsymbol{\zeta}_2, \ldots, \boldsymbol{\zeta}_{M-1}, \boldsymbol{\phi}, \mathbf{g}_1, \ldots, \mathbf{g}_M\right]$$

- Note that $\mathbf{y}$ = Vector of Variables in *Discretized* Problem

Common Discretization Schemes

- Euler Method
  - Variables: $\mathbf{y} = (\mathbf{x}_1, \mathbf{u}_1, \ldots, \mathbf{x}_M, \mathbf{u}_M)$
  - Defects:

$$\boldsymbol{\zeta}_k = \mathbf{x}_{k+1} - \mathbf{x}_k - h_k \mathbf{f}_k$$

- Classical Runge-Kutta Method
  - Variables: $(\mathbf{x}_1, \mathbf{u}_1, \mathbf{u}_2, \ldots, \bar{\mathbf{u}}_M, \mathbf{x}_M, \mathbf{u}_M)$
  - Defects

$$\boldsymbol{\zeta}_k = \mathbf{x}_{k+1} - \mathbf{x}_k - \frac{1}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_2 + \mathbf{k}_3\right)$$

    where

$$\begin{aligned}
\mathbf{k}_1 &= h_k \mathbf{f}_k \\
\mathbf{k}_2 &= h_k \mathbf{f}\left(\mathbf{x}_k + \tfrac{1}{2}\mathbf{k}_1, \bar{\mathbf{u}}_{k+1}, t_k + \tfrac{1}{2}h_k\right) \\
\mathbf{k}_3 &= h_k \mathbf{f}\left(\mathbf{x}_k + \tfrac{1}{2}\mathbf{k}_2, \bar{\mathbf{u}}_{k+1}, t_k + \tfrac{1}{2}h_k\right) \\
\mathbf{k}_4 &= h_k \mathbf{f}\left(\mathbf{x}_k + \mathbf{k}_3, \mathbf{u}_{k+1}, t_k + h_k\right) \\
\bar{\mathbf{u}}_{k+1} &= \tfrac{1}{2}\left(\mathbf{u}_k + \mathbf{u}_{k+1}\right)
\end{aligned}$$

- Hermite-Simpson Method

- Variables: $\mathbf{y} = (\mathbf{x}_1, \mathbf{u}_1, \bar{\mathbf{u}}_2, \ldots, \bar{\mathbf{u}}_M, \mathbf{x}_M, \mathbf{u}_M)$
- Defects:

$$\boldsymbol{\zeta}_k = \mathbf{x}_{k+1} - \mathbf{x}_k - \frac{1}{6} h_k \left(\mathbf{f}_k + 4\bar{\mathbf{f}}_{k+1} + \mathbf{f}_{k+1} \right)$$

where

$$\begin{aligned} \bar{\mathbf{x}}_{k+1} &= \tfrac{1}{2} \left(\mathbf{x}_k + \mathbf{x}_{k+1} \right) + \tfrac{1}{8} h_k \left(\mathbf{f}_k - \mathbf{f}_{k+1} \right) \\ \bar{\mathbf{f}}_{k+1} &= \mathbf{f} \left(\bar{\mathbf{x}}_{k+1}, \bar{\mathbf{u}}_{k+1}, t_k + \tfrac{1}{2} h_k \right) \end{aligned}$$

Variable-Length Phase Durations

- Phase Duration May Be a Variable, i.e., t_0 and t_f May Not Be Fixed
- For Variable-Length Phase,
  - Need to include t_0 and/or t_f as variables in discretization
  - Discretization step must be altered such that

$$h_k = \tau_k (t_f - t_0) = \tau_k \Delta t$$

    where $\Delta t = t_f - t_0$ and $0 < \tau_k < 1$
  - Essentially, t_0 and t_f change like an "accordion"

Highlights of Direct Approach

- Continuous-Time Dynamics $\Longrightarrow$ Algebraic Equations
- Algebraic Equations Have Defects
- Objective: Make Defects Zero
- In Other Words, Defects are *Equality Constraints*
- Problem is Transcribed to an NLP
- Basic Structure of NLP
  - ▷ Minimize

$$f(\mathbf{z})$$

subject to

$$\left.\begin{array}{c} \mathbf{g}(\mathbf{z}) = \mathbf{0} \\ \mathbf{h}_{\min} \leq \mathbf{h}(\mathbf{z}) \leq \mathbf{h}_{\max} \end{array}\right\} \Longrightarrow \begin{array}{c} \text{Contains Disretized Dynamics} \\ \text{Path Constraints, and Boundary Conditions} \end{array}$$

Other Positive Attributes of Collocation Methods

- Inclusion of Path Constraints Much Easier
- To See This, Consider Euler Collocation Method Again
- Recall NLP Variables in Euler Scheme: $\mathbf{z} = (x_0, \ldots, x_M, u_0, \ldots, u_{M-1})$
- Suppose We Have Path Constraint $\mathbf{C}(\mathbf{x}, \mathbf{u}) \leq \mathbf{0}$
- Then We Can Enforce Path Constraint at Collocation Points As

$$\mathbf{C}(x_i, u_i) \leq \mathbf{0}, \quad (i = i, \ldots, M-1)$$

- Observations
  - ▷ Path Constraints No More Complicated Than Dynamic Constraints
  - ▷ Can Be Inequalty or Equality
  - ▷ Does Not Require Ad Hoc Procedure to Include

Trade-Off Between Indirect and Direct Methods

- Indirect Methods
  - Beautiful mathematics (calculus of variations)
  - If solvable, HBVP produces state, control, and costate
  - Can gain great insight into structure of solution (e.g., LQR problem)
  - We know how close to optimal is the solution (because we have costate)
  - However,
    - Small radii of convergence
    - Difficult to solve HBVP
    - Often, cannot even derive optimality conditions
    - Indirect methods are generally intractable
- Direct Methods
  - Mathematics not so elegant (finite-difference methods)
  - No costate produced from solution to NLP
  - Difficult to gain insight into structure of solution
  - Do not know how close we are to "true" optimal solution

- However,
  - Large radii of convergence
  - Much easier to solve NLP than to solve HBVP
  - No need to derive optimality conditions
  - Direct methods are highly tractable

- So What is "Ideal"
  - Want Direct Method
  - Want Direct Method to Have Features of Indirect Method
    - In particular, want an accurate costate
    - Then we effectively have same information as an indirect method
  - Question: Do Such Methods Exist?
  - Answer: Yes
  - Question: What Are These Methods?
  - Answer: *Pseudospectral Methods*

Part V: Numerical Integration & Interpolation

Numerical Quadrature and Interpolation

- Numerical Quadrature
  - ▷ Need to Accurately Approximate Integral of a Function
  - ▷ Quadrature Approximations
    - ◦ Sample Function at Points in the Interval (Support Points)
    - ◦ Use a Weighted Sum of These Points (Quadrature Weights)
  - ▷ Generally Quadrature Approximations Done on $\tau \in [-1, +1]$.

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

  - ▷ Interpolation
    - ◦ Function Values Given at Discrete Points $(\tau_1, \ldots, \tau_N)$
    - ◦ Need to Construct Continuous Approximation

$$f(\tau) \approx F(\tau) = \sum_{i=1}^{N} c_i \psi_i(\tau)$$

    - ◦ $\psi_i(\tau)$ are Trial or Basis Functions

Methods for Numerical Integration

- Shooting and Multiple-Shooting Require Numerical Integration
- Methods for Numerical Integration: Initial-Value Problems
- Consider Integration from t_i to t_{i+1}

$$\mathbf{p}_{i+1} = \mathbf{p}_i + \int_{t_i}^{t_{i+1}} \mathbf{G}(\mathbf{p}, t)dt$$

- Consider a *Multi-Stage* Method of The Form

$$\tau_j = t_i + h_i \rho_j$$

where $0 \leq \rho_1 \leq \rho_2 \leq \cdots \leq \rho_k \leq 1$ and $1 \leq j \leq k$

- Now Apply a *Quadrature Formula* of The Form

$$\int_{t_i}^{t_{i+1}} \mathbf{G}(\mathbf{p})dt \approx h_i \sum_{j=1}^{k} \beta_j \mathbf{G}_j$$

where $\mathbf{G}_j = \mathbf{G}(\mathbf{p}_j, \tau_j)$

- Next Apply a Quadrature *within* a Quadrature of the Form

$$\int_{t_i}^{\tau_j} \mathbf{G}(\mathbf{p})dt \approx h_i \sum_{l=1}^{k} \alpha_{jl}\mathbf{G}_l$$

for $1 \leq j \leq k$

- Popular Multi-Stage Methods: Runge-Kutta

$$\mathbf{p}_{i+1} = \mathbf{p}_i + h_i \sum_{j=1}^{k} \beta_j \mathbf{G}_{ij}$$

where

$$\mathbf{G}_{ij} = \mathbf{G}\left[\left(\mathbf{p}_i + h_i \sum_{l=1}^{k} \alpha_{jl}\mathbf{G}_{il}\right)\right]$$

- Runge-Kutta Schemes Visualizable Using a *Butcher Array*

$$\begin{array}{c|cccc} \rho_1 & \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1k} \\ \rho_2 & \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2k} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \rho_k & \alpha_{k1} & \alpha_{k2} & \cdots & \alpha_{kk} \\ \hline & \beta_1 & \beta_2 & \cdots & \beta_k \end{array}$$

- Runge-Kutta Schemes Are
  - ▷ Explicit: $\alpha_{jl} = 0$ for $l \geq j$
  - ▷ Implicit: otherwise
- Common Runge-Kutta Schemes
  - ▷ Euler Method (explicit, $k = 1$)

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

$$\mathbf{p}_{i+1} = \mathbf{p}_i + h_i \mathbf{G}_i$$

▷ Trapezoidal Method (implicit, $k = 2$)

0	0	0
1	1/2	1/2
	1/2	1/2

$$\mathbf{p}_{i+1} = \mathbf{p}_i + \frac{1}{2} h_i \left(\mathbf{G}_i + \mathbf{G}_{i+1} \right)$$

▷ Classical Runge-Kutta Method (explicit, $k = 4$)

0	0	0	0	0
1/2	1/2	0	0	0
1/2	0	1/2	0	0
1	0	0	1	0
	1/6	1/3	1/3	1/6

$$\begin{aligned}
\mathbf{k}_1 &= h_i \mathbf{G}_i \\
\mathbf{k}_2 &= h_i \mathbf{G} \left(\mathbf{p}_i + \tfrac{1}{2}\mathbf{k}_1, t_i + \tfrac{1}{2} h_i \right) \\
\mathbf{k}_3 &= h_i \mathbf{G} \left(\mathbf{p}_i + \tfrac{1}{2}\mathbf{k}_2, t_i + \tfrac{1}{2} h_i \right) \\
\mathbf{k}_4 &= h_i \mathbf{G} \left(\mathbf{p}_i + \mathbf{k}_3, t_{i+1} \right)
\end{aligned}$$

$$\mathbf{p}_{i+1} = \mathbf{p}_i + \frac{1}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right)$$

▷ Hermite-Simpson (implicit, $k = 3$)

0	0	0	0
1/2	5/24	1/3	−1/24
1	1/6	2/3	1/6
	1/6	2/3	1/6

$$\begin{aligned}\bar{\mathbf{p}} &= \tfrac{1}{2}\left(\mathbf{p}_i + \mathbf{p}_{i+1}\right) + \tfrac{1}{8}h_i\left(\mathbf{G}_i - \mathbf{G}_{i+1}\right)\\ \bar{\mathbf{G}} &= \mathbf{G}\left(\bar{\mathbf{p}}, t_i + \tfrac{1}{2}h_i\right)\end{aligned}$$

$$\mathbf{p}_{i+1} = \mathbf{p}_i + \frac{1}{6}h_i\left(\mathbf{G}_i + 4\bar{\mathbf{G}} + \mathbf{G}_{i+1}\right)$$

- Alternate Motivation for Runge-Kutta Methods
  - ▷ Suppose $\mathbf{p}(t)$ is approximated on $[t_i, t_{i+1}]$ by the function $\tilde{\mathbf{p}}(t)$ as

$$\mathbf{p}(t) \approx \tilde{\mathbf{p}}(t) = \sum_{k=1}^{m} a_k (t - t_i)^k$$

where $(a_0, \ldots, a_k)$ are chosen such that $\tilde{\mathbf{p}}(t_i) = \mathbf{p}_i$

- ▷ Choose a finite set of points $\tau_1, \ldots, \tau_k$, i.e.,

$$\dot{\tilde{\mathbf{p}}}(\tau_j) = \mathbf{G}\left[\mathbf{p}(\tau_j)\right]$$

- ▷ Last equation is called a *collocation* condition

- Thus, Runge-Kutta Methods are *Collocation Methods*
- Three Common Forms of Collocation
  - ▷ Lobatto methods: both endpoints included in collocation
  - ▷ Radau methods: one endpoint included in collocation
  - ▷ Gauss methods: neither endpoint included in collocation
- Examples of Different Forms of Collocation
  - ▷ Midpoint rule $\Longleftrightarrow$ Gauss method

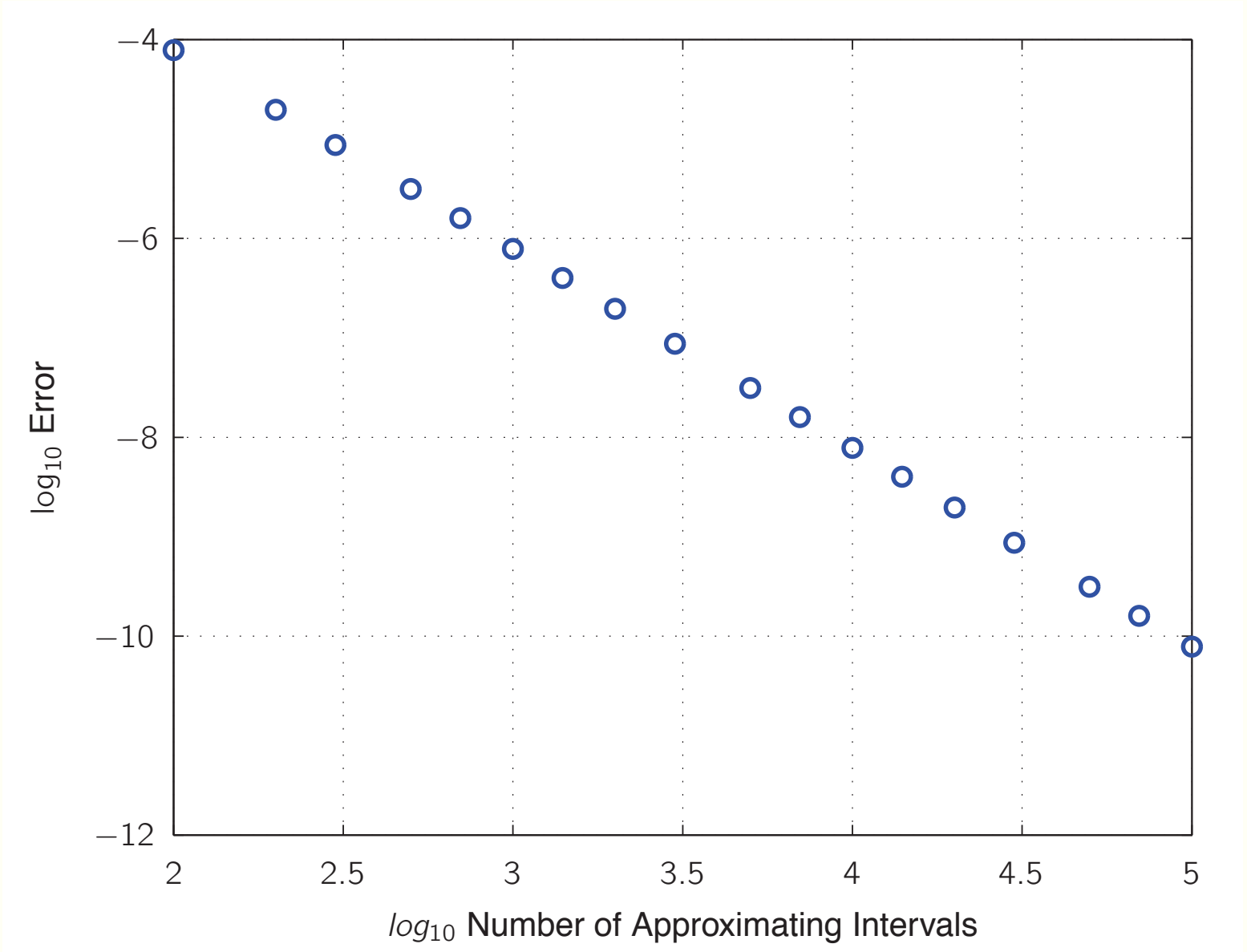
$$\mathbf{p}_{i+1} = \mathbf{p}_i + h_i \mathbf{G}\left[\frac{1}{2}\left(\mathbf{p}_i + \mathbf{p}_{i+1}\right), t_i + \frac{1}{2}h_1\right]$$

  - ▷ Euler backward rule $\Longleftrightarrow$ Radau method

$$\mathbf{p}_{i+1} = \mathbf{p}_i + h_i \mathbf{G}_{i+1}$$

  - ▷ Hermite-Simpson (see above) $\Longleftrightarrow$ Lobatto method

Convergence Rate of Low-Order Integrators



Gaussian Quadrature

- Using Evenly Spaced Points $\Longrightarrow$ Poor Convergence of Integral
- Alternative: Choose Points to Minimize Error in Quadrature Approximation
- Suppose We Want to Approximate $\int_{-1}^{+1} g(\tau)d\tau$ Using N Points. Then

$$\int_{-1}^{+1} p(\tau)d\tau \approx \sum_{k=1}^{N} w_k p(\tau_k)$$

- Error in Quadrature Approximation, $E(g)$

$$E(g) = \int_{-1}^{+1} g(\tau)d\tau - \sum_{k=1}^{N} w_k g(\tau_k)$$

- Gaussian Quadature Points Chosen to Minimize $E(p)$
- Question: Why Use Gaussian Quadrature Points for Collocation?
- Answer: Exponential Convergence for Smooth Functions

Family of Gaussian Quadrature Points

- Let $P_N(\tau)$ Be the N^{th}-Degree Legendre Polynomial
- Gaussian Quadrature Collocation Points
  - ▷ Legendre-Gauss: Roots of $P_N(\tau)$
  - ▷ Legendre-Gauss-Radau: Roots of $P_N(\tau) + P_{N-1}(\tau)$
  - ▷ Legendre-Gauss-Lobatto: Roots of $\dot{P}_{N-1}(\tau)$ Together with $\tau = -1$ and $\tau = +1$
- Exactness of Gaussian Quadrature Approximations
  - ▷ Legendre-Gauss (LG): polynomial of degree $2N - 1$
  - ▷ Legendre-Gauss-Radau (LGR): polynomial of degree $2N - 2$
  - ▷ Legendre-Gauss-Lobatto (LGL): polynomial of degree $2N - 3$

Exactness of Gaussian Quadrature

- Let
  - $(\tau_1, \ldots, \tau_N)$ be the roots of $P_N(\tau)$
  - $g(\tau)$ be a polynomial of degree *at most* $2N-1$
- Will Show That

$$\int_{-1}^{+1} g(\tau) d\tau$$

  is Exact *if* $g(\tau)$ is a Polynomial of Degree at Most $2N-1$
- First, Because $g(\tau)$ is a Polynomial of Degree at Most $2N-1$, $g(\tau)$ Can Be Written as

$$g(\tau) = P_N(\tau) f(\tau) + h(\tau)$$

  Where $f(\tau)$ and $h(\tau)$ are Polynomials of Degree at most $N-1$
- Then,

$$\int_{-1}^{+1} g(\tau) d\tau = \int_{-1}^{+1} \left(P_N(\tau) f(\tau) + h(\tau) \right) d\tau = \int_{-1}^{+1} P_N(\tau) f(\tau) d\tau + \int_{-1}^{+1} h(\tau) d\tau$$

- Because $f(\tau)$ is a Polynomial of Degree at Most $N-1$,

$$f(\tau) = \sum_{i=0}^{N-1} c_i P_i(\tau), \quad P_i(\tau) = i^{th} - \text{Degree Legendre Polynomial}$$

- Next, $P_N(\tau)$ is Orthogonal to All Other Legendre Polynomials. Therefore

$$\int_{-1}^{+1} P_N(\tau) f(\tau) d\tau = \int_{-1}^{+1} P_N(\tau) \sum_{i=0}^{N-1} c_i P_i(\tau) d\tau$$
$$= \sum_{i=0}^{N-1} c_i \int_{-1}^{+1} P_N(\tau) P_i(\tau) d\tau = 0$$

- Consequently,

$$\int_{-1}^{+1} g(\tau) d\tau = \int_{-1}^{+1} h(\tau) d\tau$$

- Recall Now that $h(\tau)$ is
  - ▷ A Polynomial of Degree at Most $N-1$
  - ▷ Thus, $h(\tau)$ can be decomposed in term of a basis of N Lagrange

polynomials that are each of degree $N-1$,

$$h(\tau) = \sum_{i=1}^{N} h(\tau_i) L_i(\tau), \quad L_i(\tau) = \prod_{\substack{j=1 \\ i \neq j}}^{N} \frac{\tau - \tau_j}{\tau_i - \tau_j}$$

- Therefore,

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

- Because $g(\tau) = P_N(\tau) f(\tau) + h(\tau) \Rightarrow g(\tau_i) = P_N(\tau_i) f(\tau_i) + h(\tau_i)$
- But $P_N(\tau_i) f(\tau_i) = 0$ Because $(\tau_1, \ldots, \tau_N)$ are the Roots of $P_N(\tau)$
- Consequently, $g(\tau_i) = h(\tau_i)$

- We Then Obtain

$$\int_{-1}^{+1} g(\tau)d\tau = \int_{-1}^{+1} h(\tau)d\tau = \sum_{i=1}^{N} h(\tau_i) \int_{-1}^{+1} L_i(\tau)d\tau$$

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

$$= \sum_{i=1}^{N} w_i g(\tau_i)$$

  Where

$$w_i = \int_{-1}^{+1} L_i(\tau)d\tau = \text{Quadrature Weight}$$

- Thus,

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

  is Exact if $g(\tau)$ is a Polynomial of Degree at Most $2N-1$

Accuracy of LGR and LGL Quadrature

- Legendre-Gauss-Radau Quadrature
  - ▷ LGR Points: $N-1$ roots on $(-1,+1]$ of $P_N(\tau)+P_{N-1}(\tau)$, Together with $\tau_1=-1$
  - ▷ Because $\tau_1=-1$ is specified $\Longrightarrow$ have $N-1$ free points
  - ▷ Can use similar argument for LGR as was used for LG with following changes
    - ◦ Given $N-1$ free points on $(-1,+1]$,

$$\int_{-1}^{+1} g(\tau)d\tau = \sum_{i=1}^{N} w_i g(\tau_i), \quad \tau_1=-1$$

is exact if $g(\tau)$ is a polynomial of degree at most $2N-2$

- Legendre-Gauss-Lobatto Quadrature
  - ▷ LGL Points: $N-2$ roots on $(-1,+1)$ of $\dot{P}_N(\tau)$, Together with $\tau_1=-1$ and $\tau_N=+1$
  - ▷ Because $\tau_1=-1$ and $\tau_N=+1$ are specified $\Longrightarrow$ have $N-2$ free points

- ▷ Can use similar argument for LGL as was used for LG with following changes
  - ○ Given $N-2$ free points on $(-1,+1)$,

$$\int_{-1}^{+1} g(\tau)d\tau = \sum_{i=1}^{N} w_i g(\tau_i), \quad \tau_1 = -1$$

is exact if $g(\tau)$ is a polynomial of degree at most $2N-3$

- Key Result: Given Desire to Approximate $\int_{-1}^{+1} g(\tau)d\tau$ via N Point Quadrature,
  - ▷ LG quadrature is exact if $g(\tau)$ is a polynomial of degree $2N-1$ or less
  - ▷ LGR quadrature is exact if $g(\tau)$ is a polynomial of degree $2N-2$ or less
  - ▷ LGL quadrature is exact if $g(\tau)$ is a polynomial of degree $2N-3$ or less

Formulae for LG, LGR, and LGL Quadrature Weights

- LG

$$w_i = \frac{2}{1-\tau_i^2 \dot{P}_N^2(\tau_i)}, \quad i = 1, \ldots, N$$

- LGR

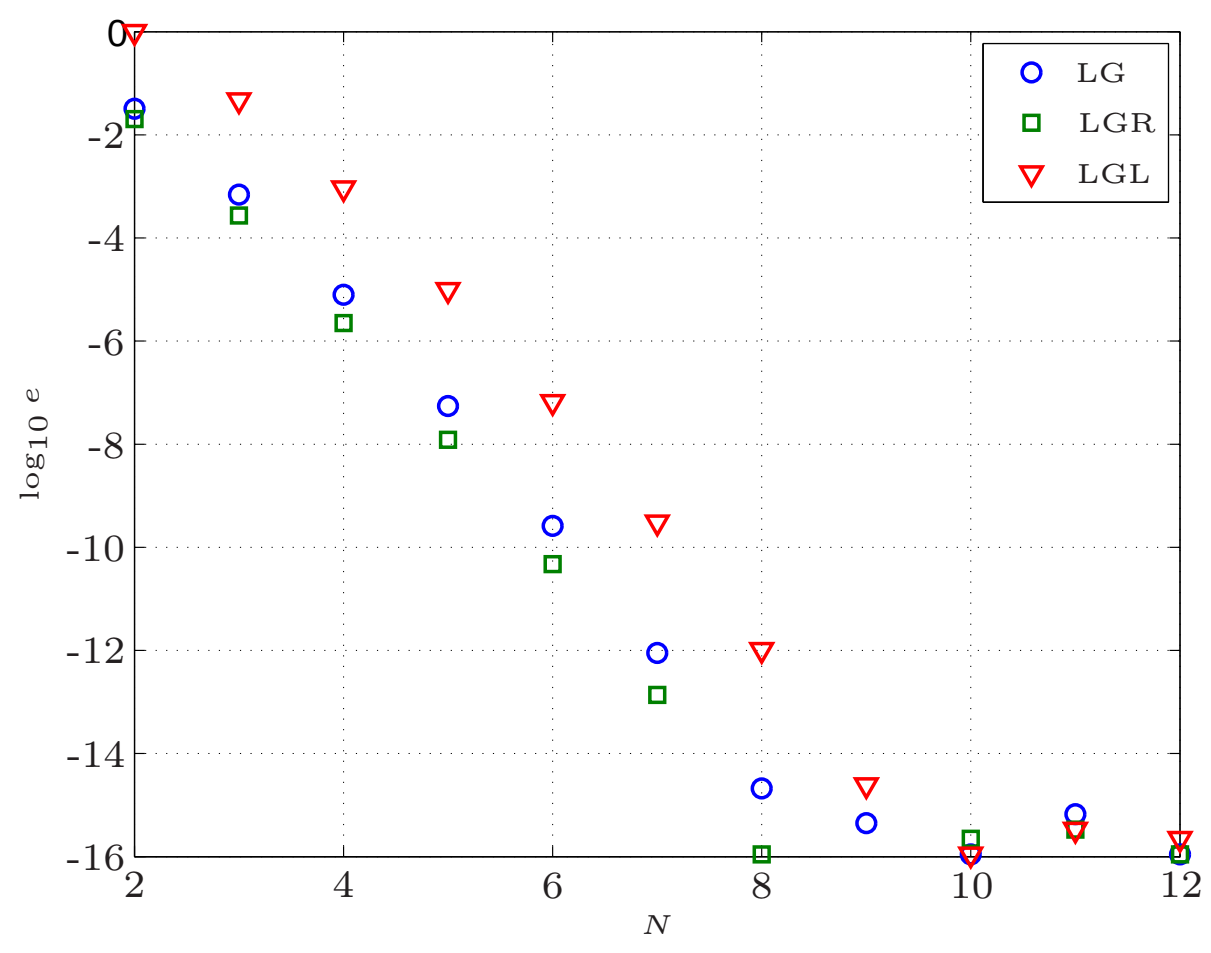
$$\begin{aligned} w_1 &= \frac{2}{N^2} \\ w_i &= \frac{1}{(1-\tau_i)\dot{P}_{N_1}^2(\tau_i)} \end{aligned}$$

- LGL

$$\begin{aligned} w_1 = w_N &= \frac{2}{N(N-1)} \\ w_i &= \frac{2}{N(N-1)P_{N_1}^2(\tau_i)} \end{aligned}$$

Example of Exponential Convergence of Gaussian Quadrature

- Approximate $(\pi/4)\int_{-1}^{+1}\cos(\pi\tau/2)d\tau$ (Exact Integral =1)



- Note: All Gaussian Quadrature Rules Convergence Exponentially

Function Approximation and Interpolation

- Solving Continuous-Time Problem Numerically
  - ▷ Continuous problem replaced with discrete approximation
  - ▷ Discrete approximation require that functions be approximated
  - ▷ Approximations generally computed using polynomials
- In Principle, Any Polynomial Can Be Used
- Practically, Easier to Utilize *Lagrange Interpolating Polynomials*

Lagrange Interpolating Polynomials

- Given Samples of a Function $f(t)$ at Support Points $(t_1, \ldots, t_N)$
- Using $(t_1, \ldots, t_N)$, Construct N *Lagrange Polynomials* of Degree $N-1$

$$L_i(t) = \prod_{\substack{k=1 \\ k \neq i}}^{N} \frac{t - t_k}{t_i - t_k}, \quad (i = 1, \ldots, N),$$

- Isolation Property of Lagrange Interpolating Polynomials

$$L_i(t_k) = \begin{cases} 1 & , \quad i = k \\ 0 & , \quad i \neq k \end{cases}$$

- Now Construct Interpolant for $f(t)$ Using $L_i(t)$, $(i = 1, \ldots, N)$

$$f(t) \approx \tilde{f}(t) = \sum_{i=1}^{N} c_i L_i(\tau) \Longrightarrow c_i = f(t_i)$$

- Consequently, Interpolated Value at Support Points = Function Value!

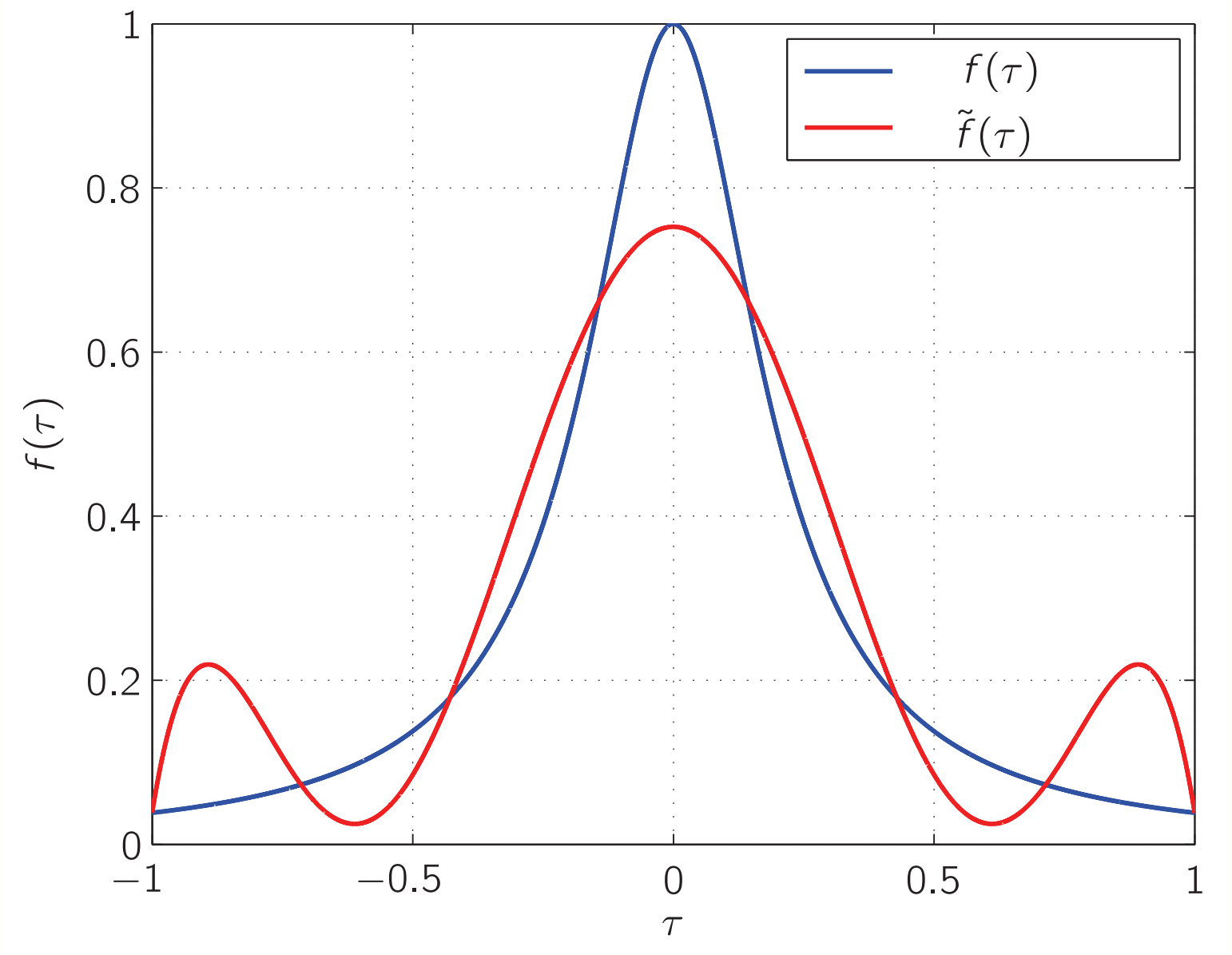
Choice of Support Points for Lagrange Polynomials

- Theoretically, Can Choose Any Values for $(t_1, \ldots, t_N)$
- Reality: Problems Arise When Support Points Not Chosen Well
- Simplest Choice: Uniformly-Spaced Points
- Consider the Following Function

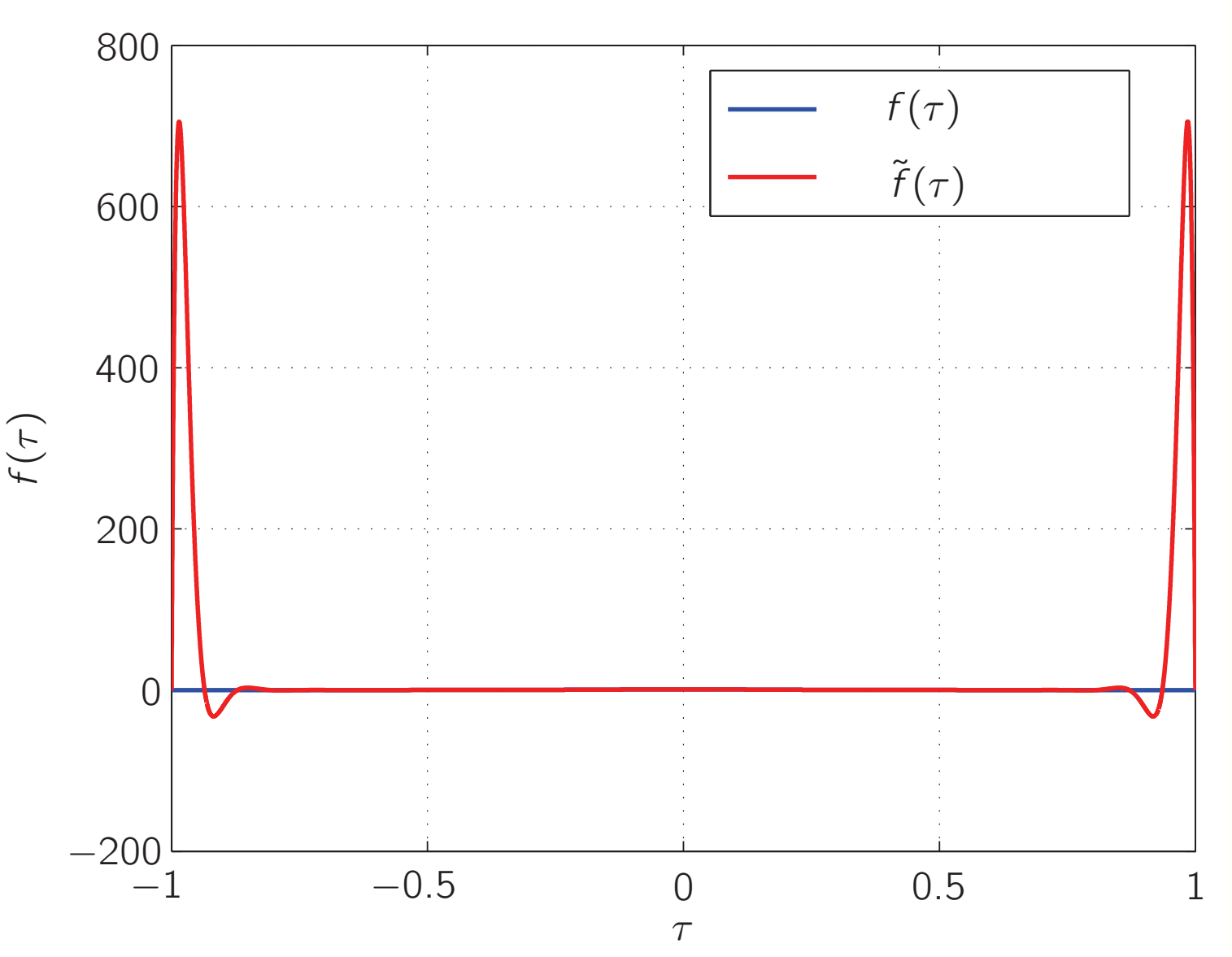
$$f(\tau) = \frac{1}{1 + 25\tau^2}, \quad \tau \in [-1, +1]$$

- Next Four Slides
  - ▷ 8 and 32 uniformly-spaced points
  - ▷ 8 and 32 Legendre-Gauss points
  - ▷ Uniformly-spaced $\Longrightarrow$ *Runge phenomenon*
  - ▷ Legendre-Gauss: Runge phenomenon disappears

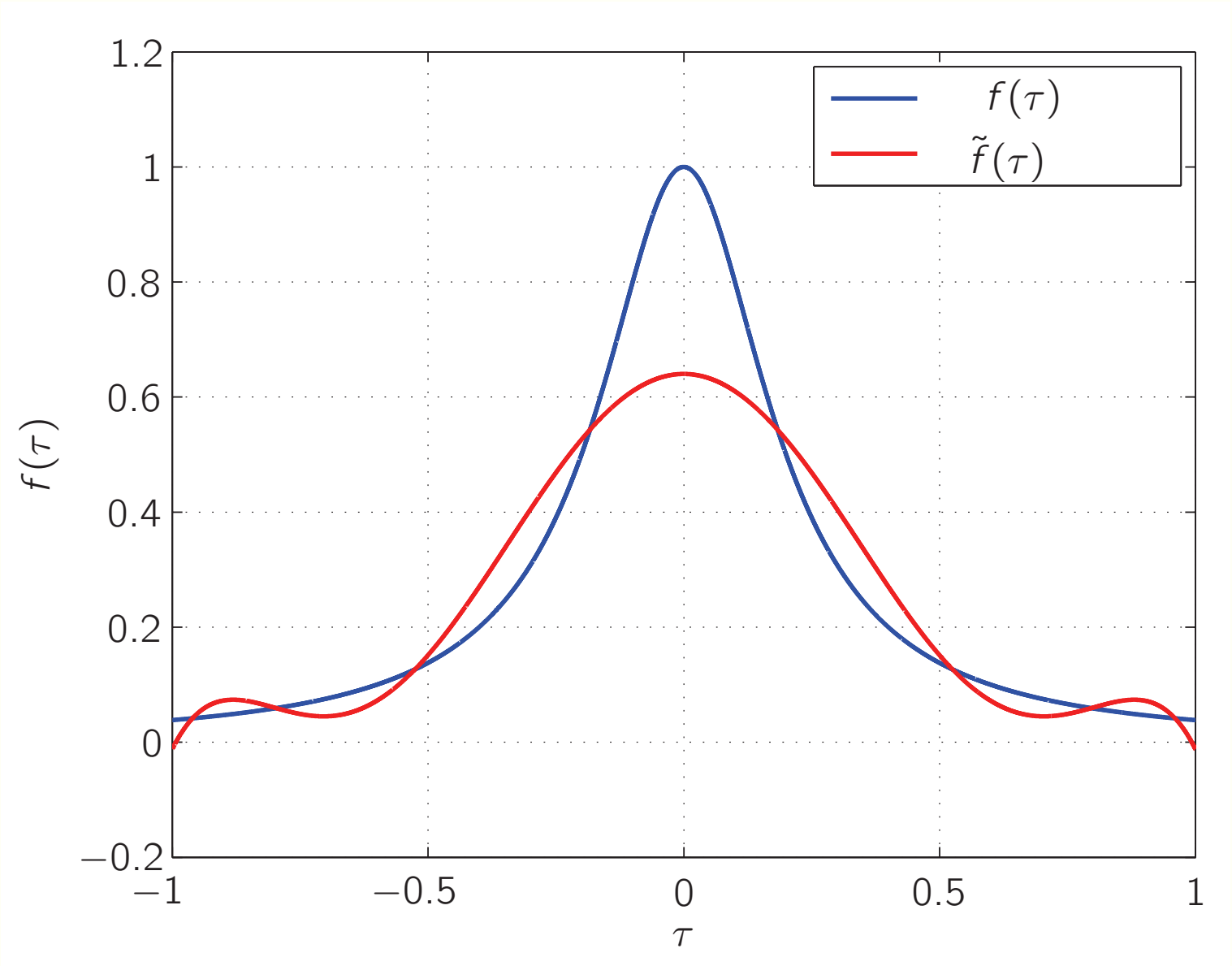
Behavior Using 8 Uniformly-Spaced Points



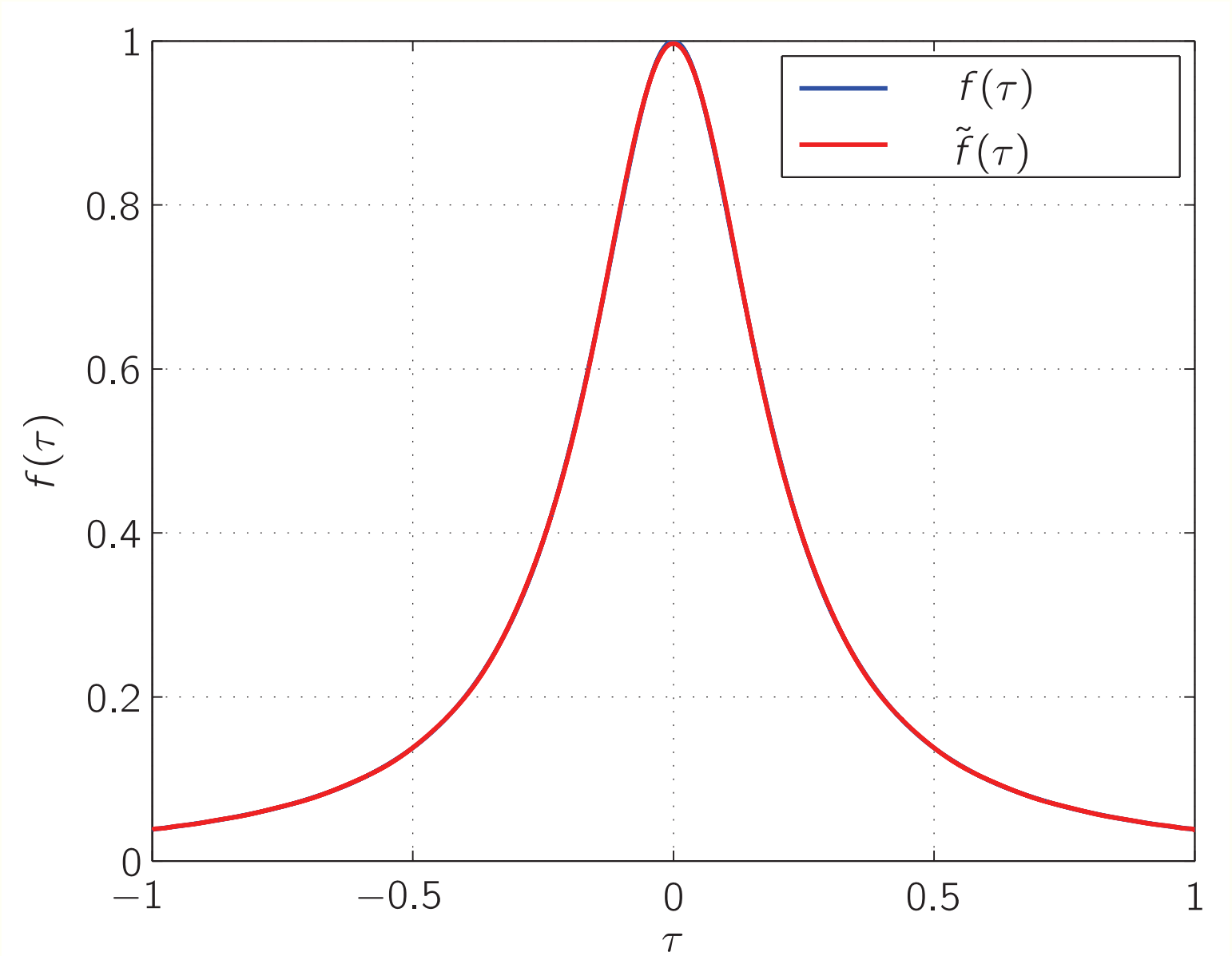
Behavior Using 32 Uniformly-Spaced Points



Behavior Using 8 LG Points



Behavior Using 32 LG Points



Motivation for a Pseudospectral Method

- Gaussian Quadrature
  - ▷ High-accuracy approximation to integral
  - ▷ Small number of support points needed
- Lagrange Interpolating Polynomials
  - ▷ Polynomial coefficients = function values at support points
  - ▷ Poorly chosen support point $\Longrightarrow$ Runge phenomenon
  - ▷ Choose Gaussian quadrature points $\Longrightarrow$ no Runge phenomenon
- Pseudospectral Method
  - ▷ Uses Gaussian quadrature points to approximate integral of a function
  - ▷ Employs Lagrange polynomials for function approximation
- Next Part of Course: Foundations of Pseudospectral Methods

Part VI: Foundations of Pseudospectral Methods

Pseudospectral Methods

- Collocation Points are Nodes of a Gaussian Quadrature

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

- Basis Functions are *Lagrange Polynomials*

$$L_i(\tau) = \prod_{\substack{j=M \\ j\neq i}}^{N} \frac{\tau - \tau_j}{\tau_i - \tau_j}, \quad L_i(\tau_j) = \begin{cases} 0 & , \quad i \neq j \\ 1 & , \quad i = j \end{cases} \Longrightarrow \text{Isolation Property}$$

- Three Sets of Gaussian Quadrature Collocation Points
  - ▷ LG: strictly on open interval $\tau \in (-1, +1)$
  - ▷ LGR: half-open interval $\tau \in [-1, +1)$ or $\tau \in (-1, +1]$
  - ▷ LGL: closed interval $\tau \in [-1, +1]$
- High Accuracy+Low-Dimensionality

$$\mathbf{x}(\tau) \approx \mathbf{X}(\tau) = \sum_{k=M}^{N} \mathbf{X}_i L_i(\tau), \quad (M = 0 \text{ or } M = 1)$$

$$\mathbf{X}(\tau_i) = \mathbf{X}_i \Longrightarrow \text{Due to Isolation Property}$$

- Key Question: What is the Difference Between LG, LGR, and LGL Points?
  - First Glance
    - All three Gaussian quadrature rules extremely accurate
    - Only differ by one polynomial degree LG $\rightarrow$ LGR $\rightarrow$ LGL
- Obtain Exponential Convergence for Integral of a Smooth Function
- Will Begin with Global Pseudospectral Methods
- Simplified Problem Used to Develop Methods
  - Minimize the Cost Functional

$$J = \Phi(\mathbf{x}(+1))$$

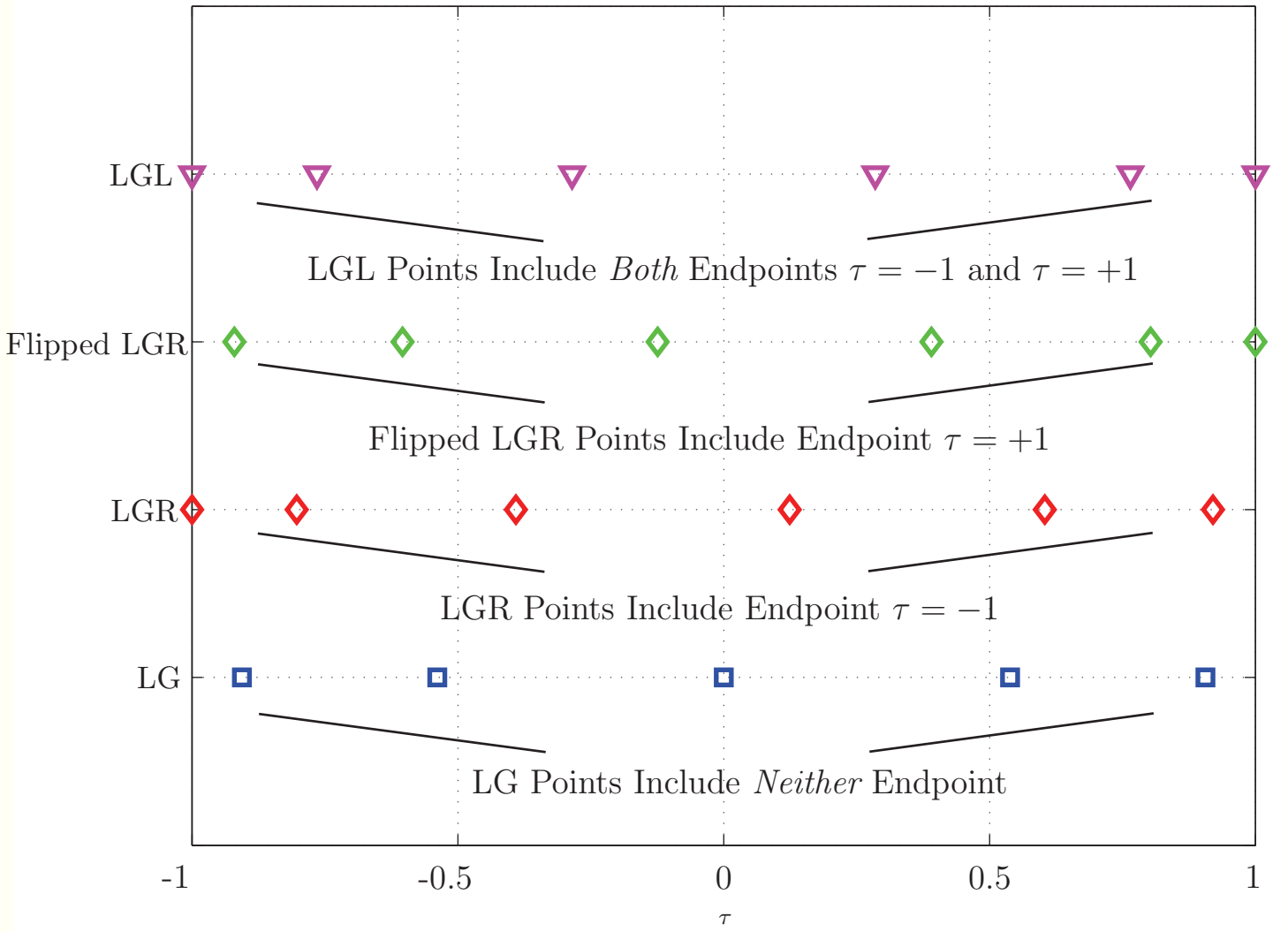
  - Dynamic Constraints

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

  - Initial Condition

$$\mathbf{x}(-1) = \mathbf{x}_0$$

Schematic of Gaussian Quadrature Points



Notation and Conventions

- This Presentation: Treat State and Control as *Row* Vectors
  - ▷ State, $\mathbf{x}(\tau) \in \mathbb{R}^n \longrightarrow \mathbf{x}(\tau) = \left[\begin{array}{ccc} x_1(\tau) & \cdots & x_n(\tau) \end{array} \right]$
  - ▷ Control, $\mathbf{u}(\tau) \in \mathbb{R}^m \longrightarrow \mathbf{u}(\tau) = \left[\begin{array}{ccc} u_1(\tau) & \cdots & u_m(\tau) \end{array} \right]$
- Vectorization of State and Control Approximations

$$\mathbf{X}_{i:j} = \left[\begin{array}{c} \mathbf{X}_i \\ \vdots \\ \mathbf{X}_j \end{array} \right] \quad , \quad \mathbf{U}_{i:j} = \left[\begin{array}{c} \mathbf{U}_i \\ \vdots \\ \mathbf{U}_j \end{array} \right]$$

- Differentiation Matrix: $\mathbf{D}_{i:j} = \textit{Columns } i \text{ Through } j \text{ of } \mathbf{D}$

$$\mathbf{D}_{i:j} = \left[\begin{array}{ccc} \mathbf{D}_i & \cdots & \mathbf{D}_j \end{array} \right]$$

- Optimal Control Functions Vectorized in Similar Manner

$$\mathbf{F}_{i:j} = \begin{bmatrix} \mathbf{f}(\mathbf{X}_i, \mathbf{U}_i) \\ \vdots \\ \mathbf{f}(\mathbf{X}_j, \mathbf{U}_j) \end{bmatrix}$$

- Notation is Similar To That Used for MATLAB Operations
- For Example: $\mathbf{DX} \longrightarrow$ Matrix-Vector Multiplication
- Collocation Equations Can Then Be Written As

$$\mathbf{DX} = \mathbf{F}$$

Gauss Pseudospectral Method (GPM)

- State Approximation:
  $\mathbf{x}(\tau) \approx \mathbf{X}(\tau) = \sum_{i=0}^{N} \mathbf{X}_i L_i(\tau) \Longrightarrow N^{th}$-Degree Polynomial

$$\dot{\mathbf{x}}(\tau_k) \approx \dot{\mathbf{X}}(\tau_k) = \sum_{i=0}^{N} \mathbf{X}_i \dot{L}_i(\tau_k) = \sum_{i=0}^{N} D_{ki} \mathbf{X}_i \Longrightarrow (N-1)^{th}\text{-Degree Polynomial}$$

$$\text{NLP:} \qquad \begin{aligned} &\text{minimize} \quad \Phi(\mathbf{X}_{N+1}) \\ &\text{subject to} \quad \mathbf{DX} = \mathbf{F}(\mathbf{X}^{\mathsf{LG}}, \mathbf{U}^{\mathsf{LG}}) \ , \ \mathbf{X}_{N+1} = \mathbf{X}_0 + \mathbf{w}^{\mathsf{T}} \mathbf{F}(\mathbf{X}^{\mathsf{LG}}, \mathbf{U}^{\mathsf{LG}}), \\ &\qquad\qquad\qquad\quad \mathbf{X}_0 = \mathbf{x}_0 \end{aligned}$$

- Can Divide $\mathbf{D}$ Into: $\mathbf{D} = [\mathbf{D}_0 \ \ \mathbf{D}_{1:N}]$ where $\mathbf{D}_{1:N}$ is Square
- Have Proven Following Theorems About $\mathbf{D}$ Matrix for GPM
  - $\mathbf{D}_{1:N}$ is Nonsingular
  - $\mathbf{D}_{1:N}^{-1}\mathbf{D}_0 = -\mathbf{1}$
  - $\mathbf{D}_{1:N}^{-1} = \mathbf{A}$ where $\mathbf{A}$ is the Gauss *Integration Matrix*

- Key Result: GPM is an *Implicit Integration Method*

$$\mathbf{X}^{\mathsf{LG}} = \mathbf{1}\mathbf{X}_0 + \mathbf{A}\mathbf{F}^{\mathsf{LG}} \Longrightarrow \text{IMPLICIT INTEGRAL FORM OF GPM}$$

- In Other Words
  - ▷ GPM Can Be Written in *Either* Differential or Integral Form
  - ▷ Differential Form $\Longleftrightarrow$ Integral Form
  - ▷ GPM *Is* an Integral Method

Radau Pseudospectral Method (RPM)

- State Approximation:
  $\mathbf{x}(\tau) \approx \mathbf{X}(\tau) = \sum_{i=1}^{N+1} \mathbf{X}_i L_i(\tau) \Longrightarrow N^{th}$-Degree Polynomial

  $$\dot{\mathbf{x}}(\tau_k) \approx \dot{\mathbf{X}}(\tau_k) = \sum_{i=0}^{N} \mathbf{X}_i \dot{L}_i(\tau_k) = \sum_{i=1}^{N+1} D_{ki} \mathbf{X}_i \Longrightarrow (N-1)^{th}\text{-Degree Polynomial}$$

  $$\text{NLP:} \qquad \begin{aligned} &\text{minimize} \quad \Phi(\mathbf{X}_{N+1}) \\ &\text{subject to} \quad \mathbf{DX} = \mathbf{F}(\mathbf{X}^{\mathsf{LGR}}, \mathbf{U}^{\mathsf{LGR}}) \\ &\qquad\qquad\quad \mathbf{X}_1 = \mathbf{x}_0 \end{aligned}$$

- Can Divide $\mathbf{D}$ Into: $\mathbf{D} = [\mathbf{D}_1 \;\; \mathbf{D}_{2:N+1}]$ where $\mathbf{D}_{2:N+1}$ is Square
- Have Proven Following Theorems About $\mathbf{D}$ Matrix for RPM
  - $\mathbf{D}_{2:N+1}$ is Nonsingular
  - $\mathbf{D}_{2:N+1}^{-1}\mathbf{D}_1 = -\mathbf{1}$
  - $\mathbf{D}_{2:N+1}^{-1} = \mathbf{A}$ where $\mathbf{A}$ is the Radau *Integration Matrix*

- Key Result: RPM is an *Implicit Integration Method*

$$\mathbf{X}_{2:N+1} = \mathbf{1}\mathbf{X}_0 + \mathbf{A}\mathbf{F}^{\mathsf{LGR}} \Longrightarrow \text{IMPLICIT INTEGRAL FORM OF RPM}$$

- In Other Words
  - ▷ RPM Can Be Written in *Either* Differential or Integral Form
  - ▷ Differential Form $\Longleftrightarrow$ Integral Form
  - ▷ RPM *Is* an Integral Method

Structure Common to Both GPM and RPM

- Right-Hand and Left-Hand Sides of Collocation Equations
  - ▷ Right-Hand Side Includes Only Collocation Points
  - ▷ Left-Hand Side: Includes Collocation Points Plus One Endpoint
  - ▷ Differentiation Matrices Have One More Column Than Row (Size= $N \times (N+1)$)
- Key Property of GPM and RPM Differentiation Matrices
  - ▷ Eliminating First Column, Remaining Square Matrix is Invertible
  - ▷ Remaining Square Matrix is the *Integration Matrix*
- GPM/RPM are *Integration* Methods

Why Choose Differential Form Over Integral Form?

- GPM/RPM are Integration Methods
  - ▷ Seems more natural to choose integral form
  - ▷ Why choose differential form?
- Answer: Differential Form is Much More Sparse
  - ▷ Collocation Equations in Differential Form

    $\mathbf{DX} = \mathbf{F} \Longrightarrow$ Each Equation Involves Only One Value of Right-Hand Side

  - ▷ Collocation Equations in Integral Form

    $\mathbf{X} = \mathbf{AF} \Longrightarrow$ Each Equation Involves All Values of Right-Hand Side
- Differential Form Computationally More Efficient

Lobatto Pseudospectral Method (LPM)

- State Approximation:
  $\mathbf{x}(\tau) \approx \mathbf{X}(\tau) = \sum_{i=1}^{N} \mathbf{x}_i L_i(\tau) \Longrightarrow (N-1)^{th}$-Degree Polynomial

$$\dot{\mathbf{x}}(\tau_k) = \sum_{i=1}^{N} \mathbf{X}_i \dot{L}_i(\tau_k) = \sum_{i=1}^{N} D_{ki} \mathbf{X}_i \Longrightarrow (N-2)^{th}\text{-Degree Polynomial}$$

LPM NLP:

$$\begin{aligned} &\text{minimize} \quad \Phi(\mathbf{X}_N) \\ &\text{subject to} \quad \mathbf{DX} = \mathbf{F}(\mathbf{X}, \mathbf{U}) \\ &\mathbf{X}_1 = \mathbf{x}_0 \end{aligned}$$

- LPM Differentiation Matrix is *Square and Singular*
- Integral and Differential Form of LPM *Not* Equivalent $\Longrightarrow$ LPM *Not* an Integration Method

Comparison of GPM/RPM/LPM

- Methods Appear Similar, But Appearances are Deceiving
- Gauss and Radau Pseudospectral Methods
  - ▷ Polynomial Degree of $\mathbf{X}(\tau) = N$ (=Number of Collocation Points)
  - ▷ *Non-Square and Full Rank Differentiation Matrix*
  - ▷ Differential Form = Implicit Integral Form
- Lobatto Pseudospectral Method (LPM)
  - ▷ Polynomial Degree of $\mathbf{X}(\tau) = N - 1$ ($<$ Number of Collocation Points)
  - ▷ **Square and Singular Differentiation Matrix**
  - ▷ Differential Form $\neq$ Implicit Integral Form
- How Do These Differences Play Out in Practice?
  - ▷ GPM/RPM State & Control Converge Much Faster Than LPM
  - ▷ On Even Simple Example, LPM Costate Does Not Converge
- NLPs for GPM/RPM Can Be Smaller Than NLP for LPM

Costate Estimation Using Pseudospectral Methods

- GPM Costate

$$\begin{aligned}
\boldsymbol{\lambda}_{N+1} &= \boldsymbol{\Lambda}_{N+1} && \rightarrow \text{Costate at } \tau = +1 \\
\boldsymbol{\lambda}_i &= \boldsymbol{\Lambda}_i / w_i + \boldsymbol{\Lambda}_{N+1} && \rightarrow \text{Costate at LG Points} \\
\boldsymbol{\lambda}_0 &= \boldsymbol{\Lambda}_{N+1} - \mathbf{D}_0^{\mathsf{T}} \boldsymbol{\Lambda}^{\mathsf{LG}} && \rightarrow \text{Costate at } \tau = -1 \text{ Via Integral Approximation of } \dot{\boldsymbol{\lambda}}
\end{aligned}$$

- RPM Costate

$$\begin{aligned}
\boldsymbol{\lambda}_i &= \boldsymbol{\Lambda}_i / w_i && \rightarrow \text{Costate at LGR Points} \\
\boldsymbol{\lambda}_{N+1} &= \mathbf{D}_{N+1}^{\mathsf{T}} \boldsymbol{\Lambda}^{\mathsf{LGR}} && \rightarrow \text{Costate at } \tau = +1 \text{ Via Integral Approximation of } \dot{\boldsymbol{\lambda}}
\end{aligned}$$

- LPM Costate

$$\boldsymbol{\lambda}_i = \boldsymbol{\Lambda}_i / w_i \quad \rightarrow \text{Costate at LGL Points}$$

- As with State Approximation,
  - ▷ GPM/RPM Endpoint Costate Obtained via Quadrature
  - ▷ **LPM Costate Not Obtained via Numerical Integration!**

Example

Minimize $J = -y(2)$

Subject To The Dynamic Constraint $\dot{y} = \frac{5}{2}(-y + yu - u^2)$

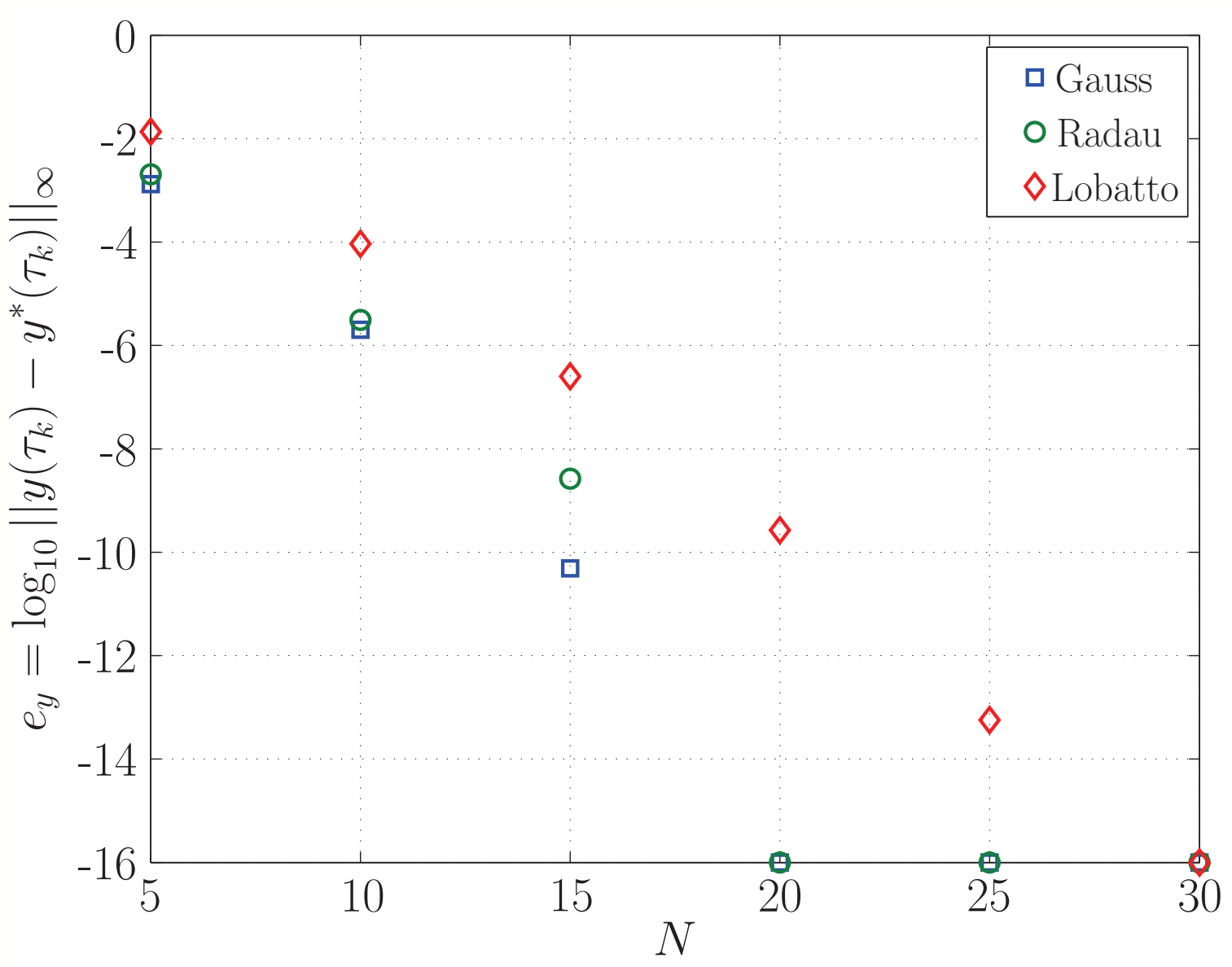
Initial Condition: $y(0) = -1$

Optimal Solution:

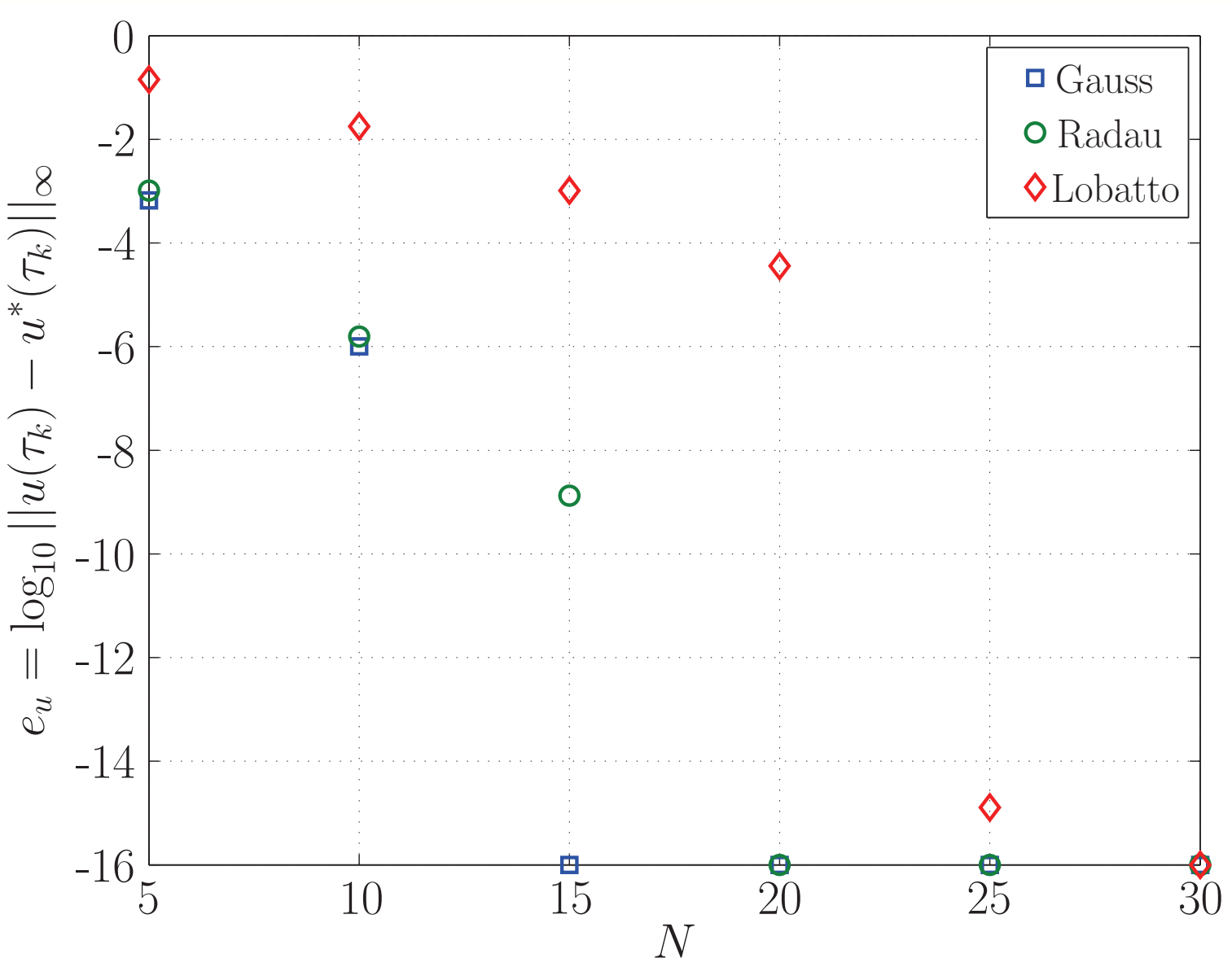
$$
\begin{aligned}
y^*(t) &= 4/a(t), \\
u^*(t) &= y^*(t)/2, \\
\lambda_y^*(t) &= -\exp(2\ln(a(t)) - 5t/2)/b, \\
a(t) &= 1 + 3\exp(5t/2), \\
b &= \exp(-5) + 6 + 9\exp(5).
\end{aligned}
$$

Problem Solved Using GPM, RPM, and LPM for $N = (5, 10, 15, 20, 25, 30)$.

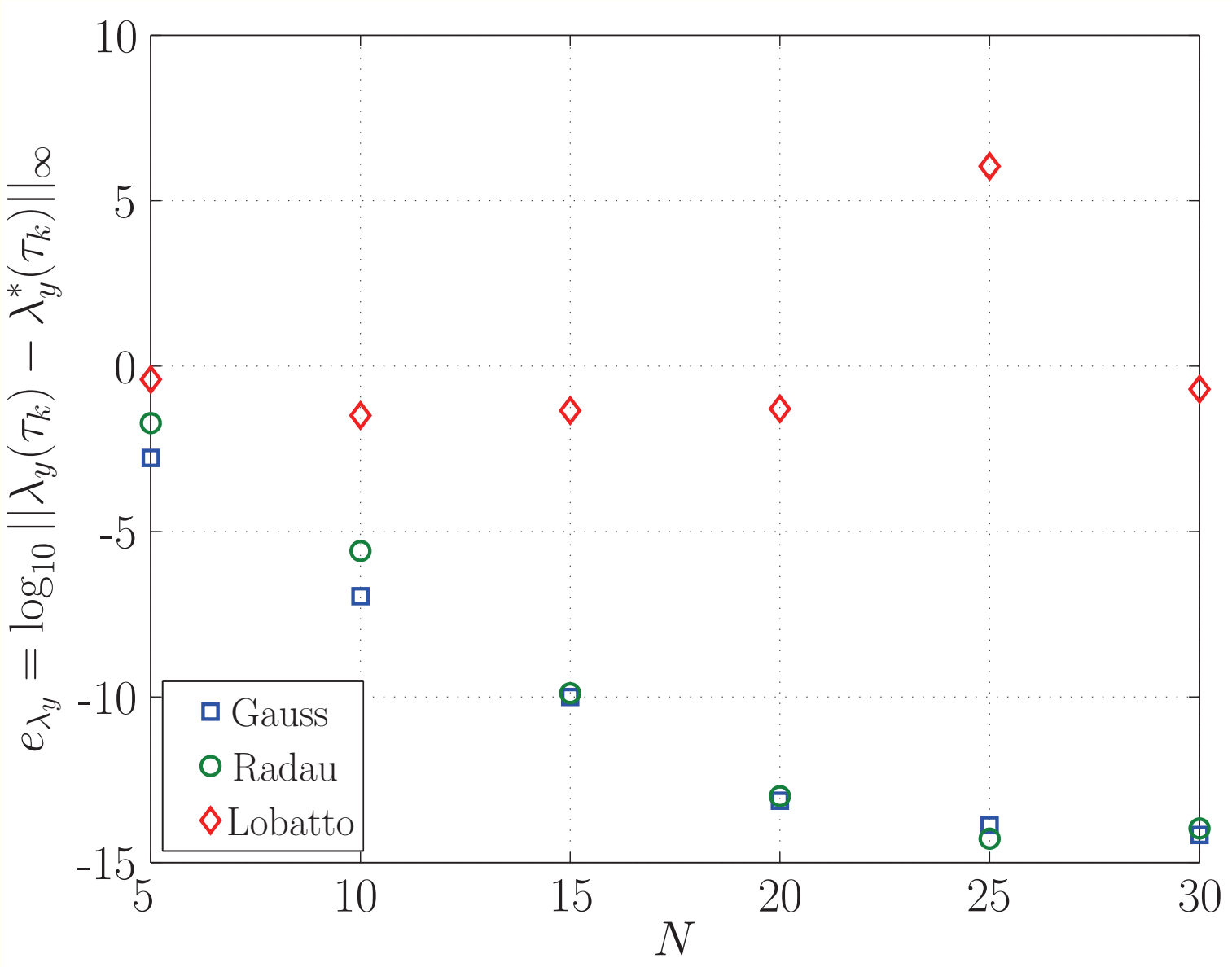
State Errors Using GPM, RPM, and LPM



Control Errors Using GPM, RPM, and LPM



Costate Errors Using GPM, RPM, and LPM



Example

Minimize $J = -r(t_f)$

Subject To The Dynamic Constraints

$$\begin{aligned}
\dot{r} &= v_r, \\
\dot{\theta} &= v_\theta / r, \\
\dot{v}_r &= v_\theta^2/r - \mu/r^2 + a\sin\beta, \\
\dot{v}_\theta &= -v_r v_\theta / r + a\cos\beta,
\end{aligned}$$

Boundary Conditions

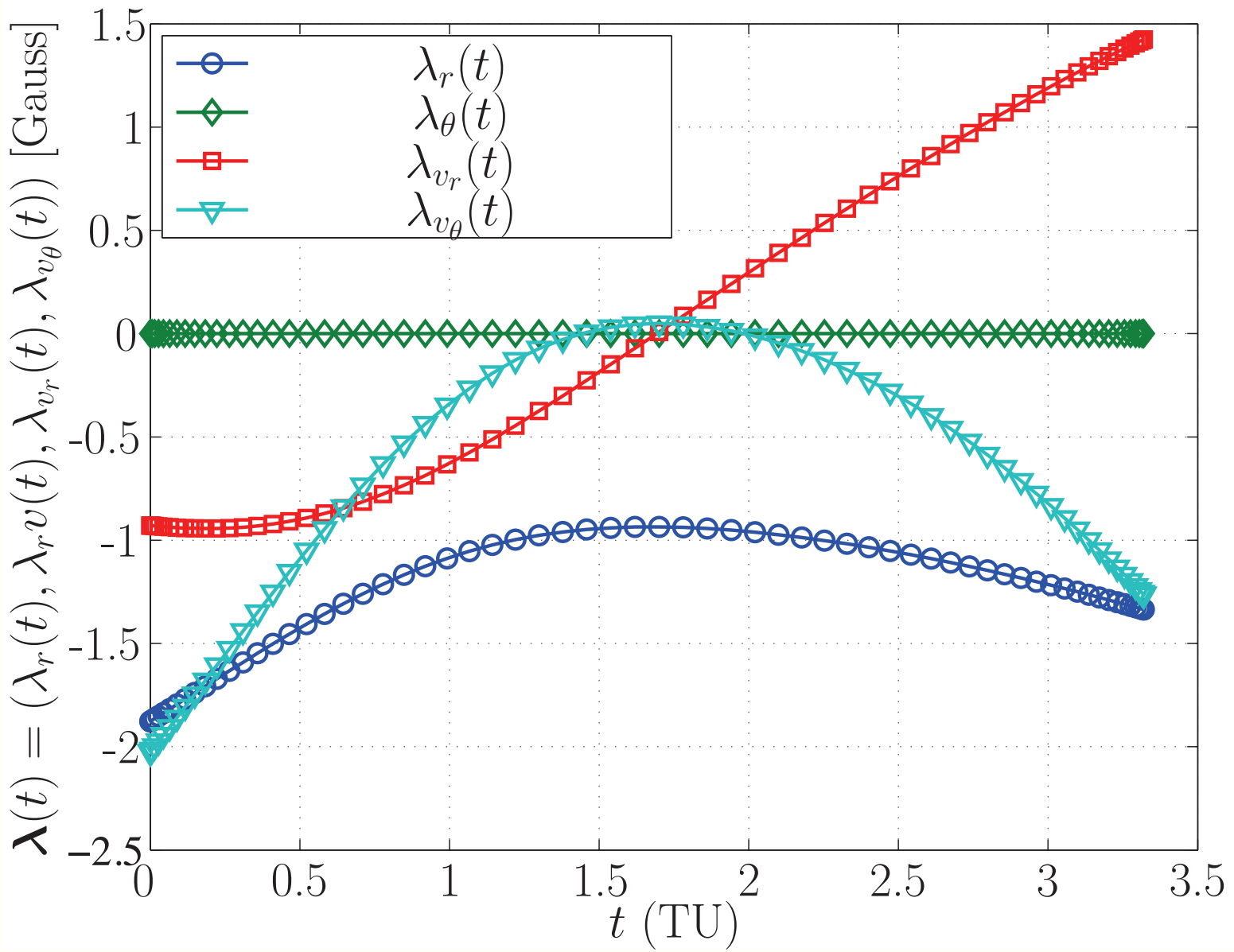
$$\begin{aligned}
(r(0), \theta(0), v_r(0), v_\theta(0)) &= (1, 0, 0, 1), \\
(v_r(t_f), v_\theta(t_f)) &= (0, \sqrt{\mu/r(t_f)}),
\end{aligned}$$

Where

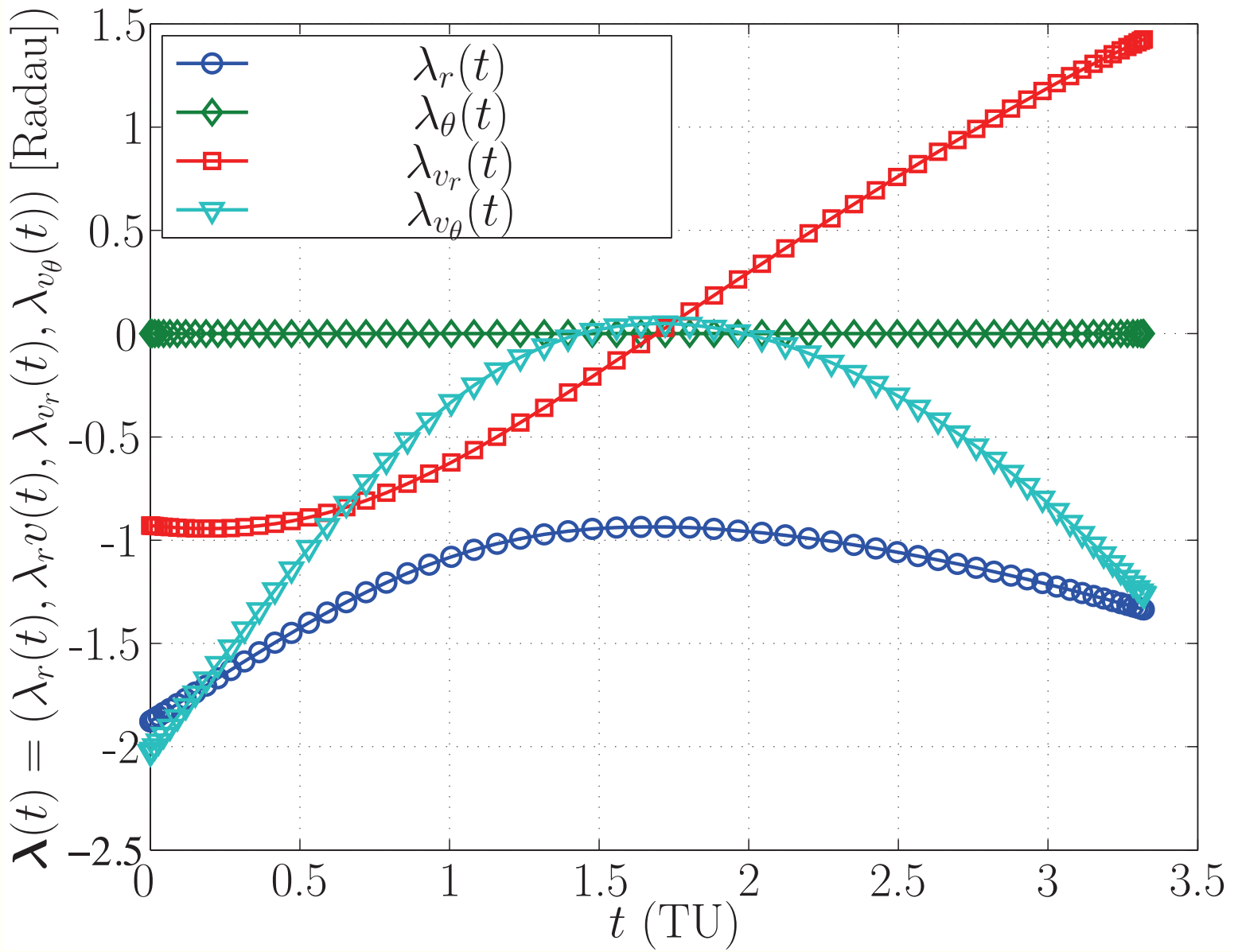
$$a \equiv a(t) = \frac{T}{m_0 - |\dot{m}|t}. \tag{1}$$

Constants: $\mu = 1$, $T = 0.1405$, $m_0 = 1$, $\dot{m} = 0.0749$, and $t_f = 3.32$.

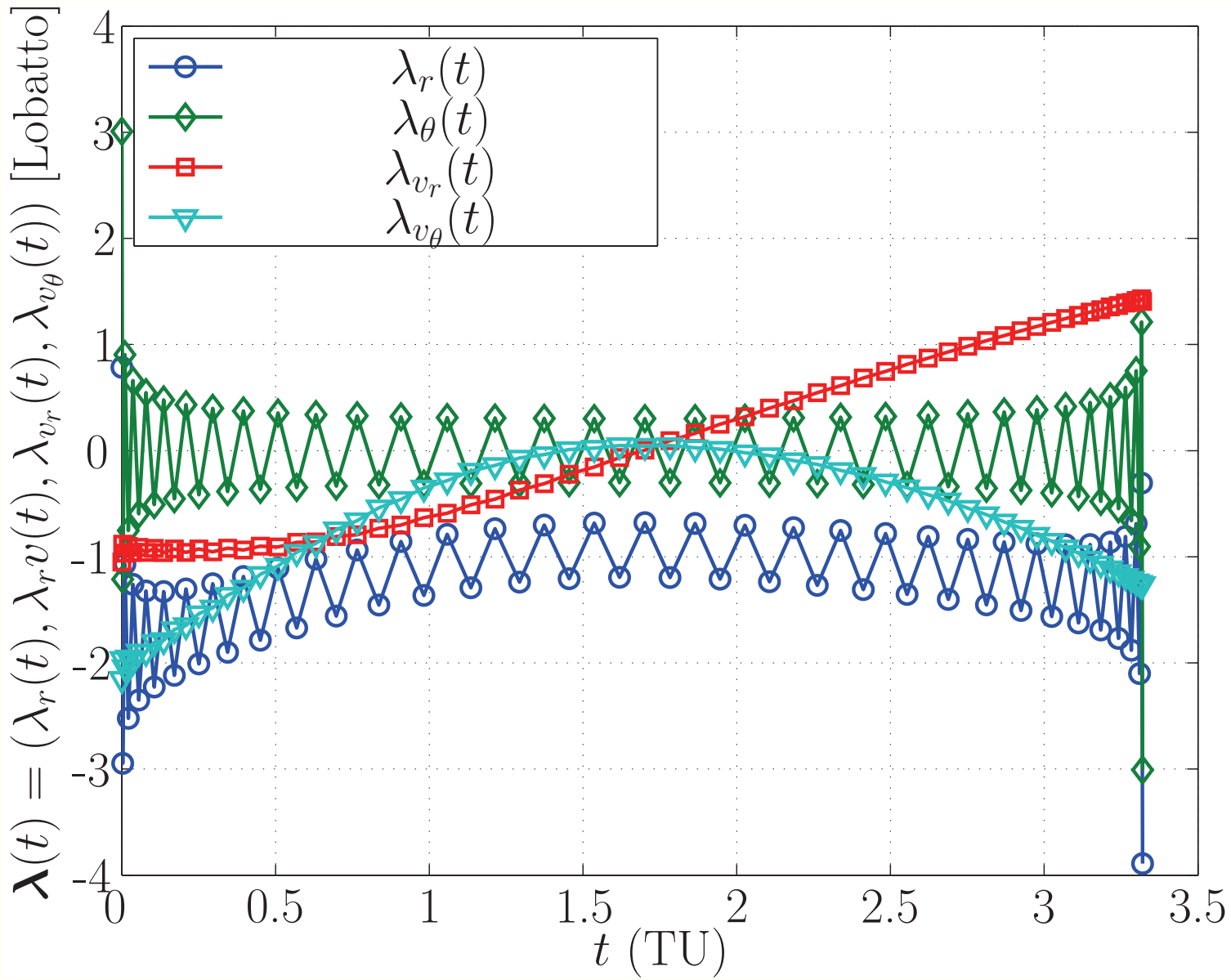
Costate Solution for $N = 64$ Using GPM



Costate Solution for $N = 64$ Using RPM



Costate Solution for $N = 64$ Using LPM



Key Features of Solution Errors

- Gauss and Radau State, Control, and Costate Converge Exponentially
- Lobatto Scheme
  - State and control errors decrease at much slower rate than either Gauss or Radau
  - Costate does not converge
- Upshot of Examples
  - Gauss and Radau
    - Both are implicit *Gaussian quadrature* schemes
    - For a problem with smooth solution, Gauss quadrature converges exponentially
  - Lobatto is *not* an implicit integration method
  - At best, converges much more slowly than either Gauss or Radau
  - Costate potentially nonconvergent

Part VII: Pseudospectral Method Implementation

Use of Multiple-Interval Pseudospectral Methods

- Global Collocation is Adequate for Simple Problems
- Most Problems of Practical Interest
  - ▷ May Have Rapid Changes in Behavior
  - ▷ May Have Points of Nonsmoothness
  - ▷ For Such Problems, Global Collocation is Inadequate
- Multiple-Interval Pseudospectral Methods
  - ▷ Time Interval $t \in [t_0, t_f]$ Divided Into Mesh Intervals
  - ▷ Each Mesh Interval
    - ○ Different Degree State Approximation
    - ○ Different Width
- Leads to *hp*-Adaptive Pseudospectral Methods

Development of Multiple-Interval Formulation

- Transformation of Time: $t = (t_f - t_0)s/2 + (t_f + t_0)/2$ where $s \in [-1, +1]$
- Minimize the Cost Functional

$$\mathcal{J} = \Phi(\mathbf{x}(-1), t_0, \mathbf{x}(+1), t_f) + \frac{t_f - t_0}{2} \int_{-1}^{+1} \mathcal{L}(\mathbf{x}(s), \mathbf{u}(s), s; t_0, t_f)\, ds$$

- Dynamic Constraints

$$\frac{d\mathbf{x}}{ds} = \frac{t_f - t_0}{2} \mathbf{f}(\mathbf{x}(s), \mathbf{u}(s), s; t_0, t_f),$$

- Path Constraints

$$\mathbf{c}_{\min} \leq \mathbf{c}(\mathbf{x}(s), \mathbf{u}(s), s; t_0, t_f) \leq \mathbf{c}_{\max},$$

- Boundary Conditions

$$\mathbf{b}_{\min} \leq \mathbf{b}(\mathbf{x}(-1), t_0, \mathbf{x}(+1), t_f) \leq \mathbf{b}_{\max}.$$

Multiple-Interval Bolza Problem

- Divide $s \in [-1, +1]$ into K *Mesh Intervals* $[s_{k-1}, s_k]$, $k = 1, \ldots, K$
- Mesh Points: $-1 = s_0 < s_1 < s_2 < \cdots < s_K = s_f = +1$.
- Let $\mathbf{x}^{(k)}(s)$ and $\mathbf{u}^{(k)}(s)$ be the State and Control in mesh interval k.
- Minimize the Cost Functional

$$\mathcal{J} = \Phi(\mathbf{x}^{(1)}(-1), t_0, \mathbf{x}^{(K)}(+1), t_f) + \frac{t_f - t_0}{2} \sum_{k=1}^{K} \int_{s_{k-1}}^{s_k} \mathcal{L}(\mathbf{x}^{(k)}(\tau), \mathbf{u}^{(k)}(\tau), \tau; s_{k-1}, s_k)\, d\tau$$

- Dynamic Constraints in Mesh Interval $k = 1, \ldots, K$:

$$\frac{d\mathbf{x}^{(k)}(s)}{ds} = \frac{t_f - t_0}{2} \mathbf{f}(\mathbf{x}^{(k)}(s), \mathbf{u}^{(k)}(s), s; t_0, t_f)$$

- Path Constraints

$$\mathbf{c}_{\min} \leq \mathbf{c}(\mathbf{x}^{(k)}(s), \mathbf{u}^{(k)}(s), s; t_0, t_f) \leq \mathbf{c}_{\max}.$$

- Boundary Conditions

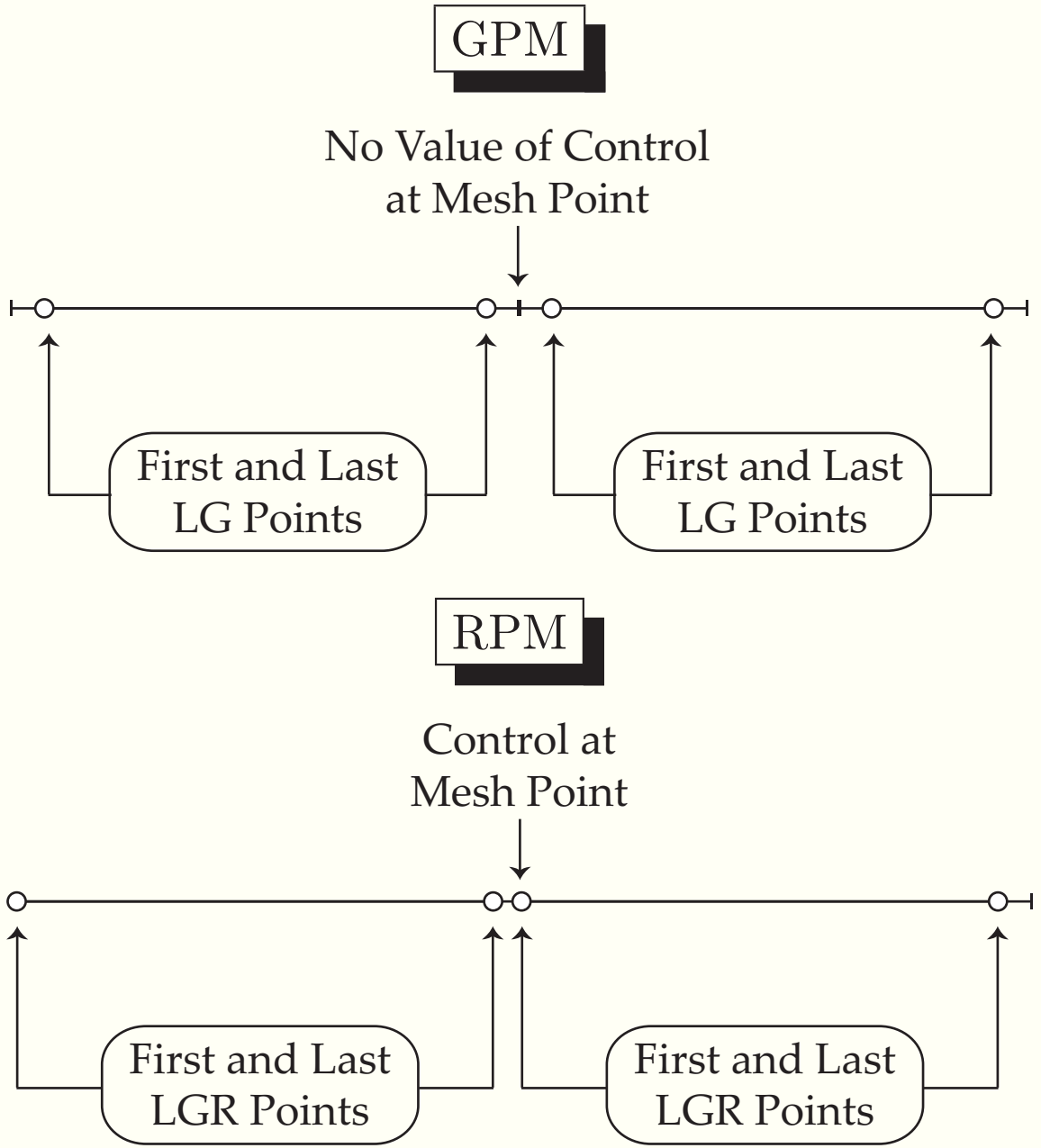
$$\mathbf{b}_{\min} \leq \mathbf{b}(\mathbf{x}^{(1)}(-1), t_0, \mathbf{x}^{(K)}(+1), t_f) \leq \mathbf{b}_{\max}.$$

- Continuity in State at Mesh Points: $\mathbf{x}(s_k^-) = \mathbf{x}(s_k^+)$, be

Choice of Multiple-Interval Pseudospectral Method

- Gauss Pseudospectral Method (GPM)
  - Additional Quadrature Equation Required to Estimate Final State in a Mesh Interval
  - Control at Mesh Point Not an NLP Variable
- Lobatto Pseudospectral Method (LPM)
  - Contains Redundant State and Control at Mesh Point
  - Derivative Approximation at Mesh Point Inconsistent Between Adjacent Mesh Intervals
- **Radau Pseudospectral Method (RPM)**
  - No Redundancy in Control at Mesh Points
  - Overlap is Ideal for Multiple-Interval Formulation
  - Is Most Natural to Implement

Schematic of Two-Interval GPM and RPM



Observations of Structure GPM and RPM

- GPM
  - Has elegant mathematical structure
  - Easy to see equivalence between direct and indirect forms
  - Unfortunately, difficult to use in multiple -interval form
- RPM
  - Not intuitive, but shares same mathematical properties as GPM
  - As a result, actually is just as accurate as GPM
  - Is much more natural for multiple-interval use
- Conclusion: Radau Method is Best Choice

Multiple-Interval Radau Pseudospectral Method

- State Approximation in Mesh Interval $k \in [1, \ldots, K]$:

$$\mathbf{x}^{(k)}(s) \approx \mathbf{X}^{(k)}(s) = \sum_{j=1}^{N_k+1} \mathbf{X}_j^{(k)} \ell_j^{(k)}(s), \quad \ell_j^{(k)}(s) = \prod_{\substack{l=1 \\ l \neq j}}^{N_k+1} \frac{s - s_l^{(k)}}{s_j^{(k)} - s_l^{(k)}},$$

  where $(s_1^{(k)}, \ldots, s_{N_k}^{(k)})$ are the LGR Points in Mesh Interval k.

- Derivative Approximation of State

$$\frac{d\mathbf{X}^{(k)}(s)}{ds} = \sum_{j=1}^{N_k+1} \mathbf{X}_j^{(k)} \frac{d\ell_j^{(k)}(s)}{ds}.$$

- Discrete Approximation of Cost Functional

$$\mathcal{J} \approx \phi(\mathbf{X}_1^{(1)}, t_0, \mathbf{X}_{N_K+1}^{(K)}, t_K) + \sum_{k=1}^{K} \sum_{j=1}^{N_k} \frac{t_f - t_0}{2} w_j^{(k)} g(\mathbf{X}_j^{(k)}, \mathbf{U}_j^{(k)}, s_j^{(k)}; s_{k-1}, s_k),$$

  - $w_j^{(k)}$ = LGR Quadrature Weights $(j = 1, \ldots, N_k)$
  - $\mathbf{U}_j^{(k)}$ = Control Approximations $(j = 1, \ldots, N_k)$
  - $\mathbf{X}_j^{(k)}$ = State Approximations $(j = 1, \ldots, N_k + 1)$

- Collocation Conditions

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

  where $t = (t_f - t_0)/2s + (t_f + t_0)/2$

- Radau Differentiation Matrix ($N_k \times (N_k + 1)$)

$$D_{ij}^{(k)} = \left[\frac{d\ell_j^{(k)}(s)}{ds} \right]_{s_i^{(k)}}, \quad (i = 1, \ldots, N_k, \quad j = 1, \ldots, N_k + 1, \quad k = 1, \ldots, K),$$

- Path Constraints

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

- Boundary Conditions

$$\mathbf{b}_{\min} \leq \mathbf{b}(\mathbf{X}_1^{(1)}, t_0, \mathbf{X}_{N_K+1}^{(K)}, t_f) \leq \mathbf{b}_{\max}.$$

- State Continuity at Mesh Points

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

Vectorized Form of Multiple-Interval Radau Method

- Quantities in Each Mesh Interval

$$\mathbf{s}^{(k)} = \left[s_i^{(k)}\right]_{N_k}^{1}, \; k = 1, \ldots, K-1 \quad , \quad \mathbf{s}^{(K)} = \left[s_i^{(K)}\right]_{N_K+1}^{1},$$

$$\mathbf{t}^{(k)} = \left[t_i^{(k)}\right]_{N_k}^{1}, \; k = 1, \ldots, K-1 \quad , \quad \mathbf{t}^{(K)} = \left[t_i^{(K)}\right]_{N_K+1}^{1},$$

$$\mathbf{X}^{(k)} = \left[\mathbf{X}_i^{(k)}\right]_{N_k}^{1}, \; k = 1, \ldots, K-1 \quad , \quad \mathbf{X}^{(K)} = \left[\mathbf{X}_i^{(K)}\right]_{N_K+1}^{1},$$

$$\mathbf{U}^{(k)} = \left[\mathbf{U}_i^{(k)}\right]_{N_k}^{1}, \; k = 1, \ldots, K \quad , \quad \mathbf{g}^{(k)} = \left[g(\mathbf{X}_i^{(k)}, \mathbf{U}_i^{(k)}, t_i^{(k)})\right]_{N_k}^{1}, \; k = 1, \ldots, K$$

$$\mathbf{A}^{(k)} = \left[\mathbf{a}(\mathbf{X}_i^{(k)}, \mathbf{U}_i^{(k)}, t_i^{(k)})\right]_{N_k}^{1}, \; k = 1, \ldots, K \quad , \quad \mathbf{C}^{(k)} = \left[\mathbf{c}(\mathbf{X}_i^{(k)}, \mathbf{U}_i^{(k)}, t_i^{(k)})\right]_{N_k}^{1}, \; k = 1, \ldots, K$$

$$\mathbf{w}^{(k)} = [w_i]_{N_k}^{1}, \; k = 1, \ldots, K \quad , \quad N = \sum_{k=1}^{K} N_k.$$

- Cumulative Quantities

$$\mathbf{s} = \begin{bmatrix} \mathbf{s}^{(1)} \\ \vdots \\ \mathbf{s}^{(K)} \end{bmatrix}, \quad \mathbf{t} = \begin{bmatrix} \mathbf{t}^{(1)} \\ \vdots \\ \mathbf{t}^{(K)} \end{bmatrix},$$

$$\mathbf{w} = \begin{bmatrix} \mathbf{w}^{(1)} \\ \vdots \\ \mathbf{w}^{(K)} \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} \mathbf{X}^{(1)} \\ \vdots \\ \mathbf{X}^{(K)} \end{bmatrix},$$

$$\mathbf{U} = \begin{bmatrix} \mathbf{U}^{(1)} \\ \vdots \\ \mathbf{U}^{(K)} \end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix} \mathbf{g}^{(1)} \\ \vdots \\ \mathbf{g}^{(K)} \end{bmatrix},$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}^{(1)} \\ \vdots \\ \mathbf{A}^{(K)} \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} \mathbf{C}^{(1)} \\ \vdots \\ \mathbf{C}^{(K)} \end{bmatrix}.$$

- Vectorized Version of Cost

$$\mathcal{J} \approx \phi(\mathbf{X}_1, t_0, \mathbf{X}_{N+1}, t_f) + \frac{t_f - t_0}{2}\mathbf{w}^{\mathsf{T}}\mathbf{g}$$

- Vectorized Version of Dynamics

$$\boldsymbol{\Delta} = \mathbf{DX} - \frac{t_f - t_0}{2}\mathbf{A} = \mathbf{0},$$

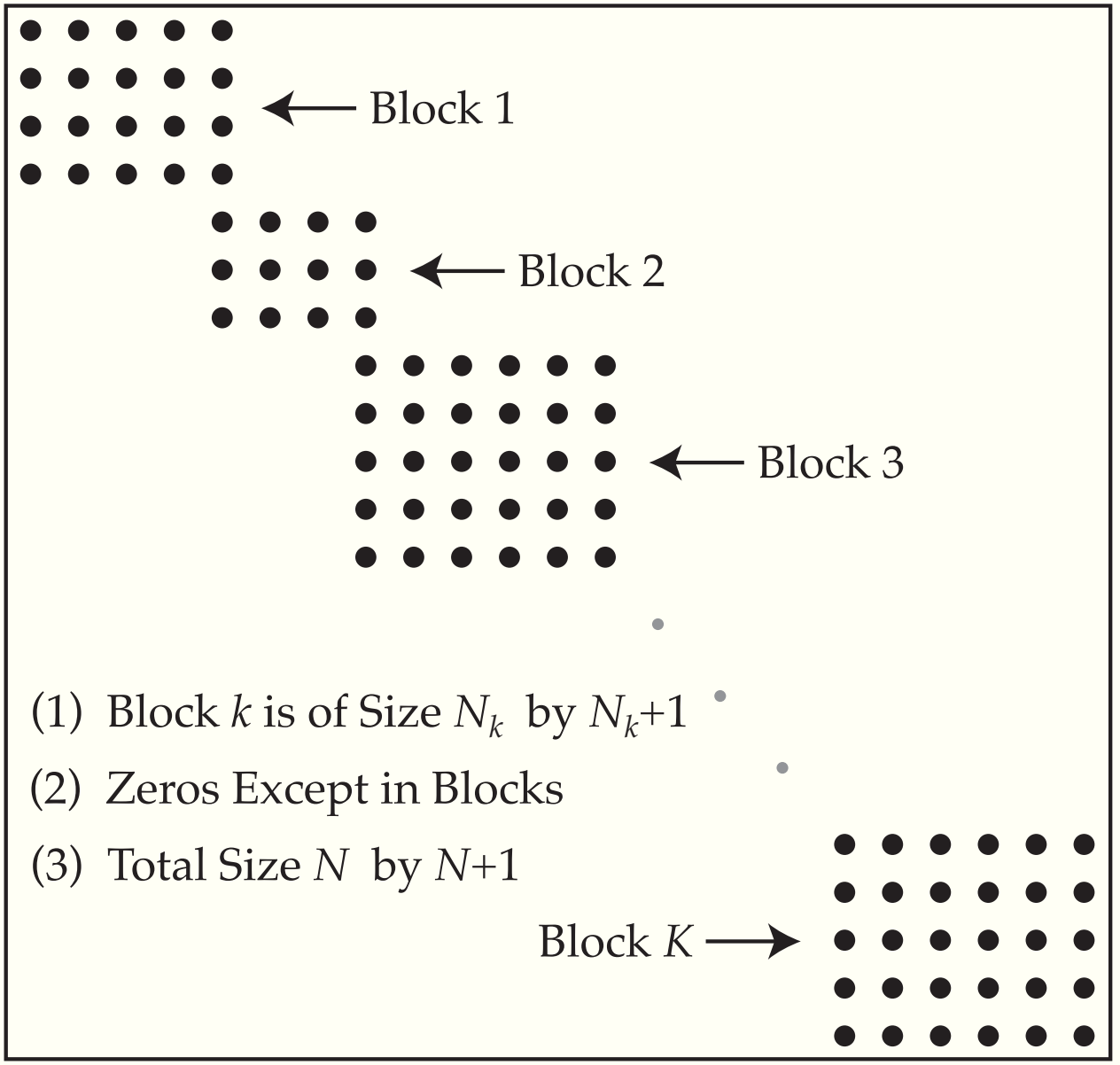
- Vectorized Version of Path Constraints

$$\mathbf{C}_{\min} \leq \mathbf{C} \leq \mathbf{C}_{\max}$$

- Vectorized Version of Boundary Conditions

$$\mathbf{b}_{\min} \leq \mathbf{b}(\mathbf{X}_1, t_0, \mathbf{X}_{N+1}, t_f) \leq \mathbf{b}_{\max}.$$

Structure of Radau Pseudospectral Differentiation Matrix



Key Features of Multiple-Interval Radau Method

- Defect Constraints for Single Interval (Global) Method
  - ▷ Left-Hand Side: LGR Points + Final Point
  - ▷ Right-Hand Side: Only LGR Points
  - ▷ Differentiation Matrix is Size $N \times (N+1)$
- Defect Constraints for Multiple Interval Method
  - ▷ Left-Hand Side: LGR Points in Each Mesh Interval + Final Point
  - ▷ Right-Hand Side: LGR Points in Each Mesh Interval
  - ▷ Differentiation Matrix is Size $N \times (N+1)$ where

$$N = \sum_{k=1}^{K} N_k$$

- Equations Have Identical Form for Both Global and Multiple Interval Method

Implementation of Radau Pseudospectral Method

- Assume a Method with a Fixed Degree in Each Segment
- Choose
  - Number of Segments, S
  - Let K = Number LGR Points on Each Segment
  - Then $N = KS$ and $N + 1 = KS + 1$ =Total Number of Points
- State Approximations at N LGR Points *Plus* Final Point

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N+1} \end{bmatrix}$$

- Control Approximation Includes *Only* LGR Points

$$\mathbf{U} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{bmatrix}$$

- Treat t_0 and t_f as Optimization Variables

LGR Points, Quadrature Weights, & Differentiation Matrix

- Compute LGR Points and Weights in Each Segment

$$\boldsymbol{\tau} \in [-1, +1) = K \text{ LGR Points}$$

$$\mathbf{W} \in [-1, +1) = K \text{ LGR Weights}$$

- Transform the LGR Points on $\tau \in [-1, +1)$ to $[s_{k-1}, s_k]$ via

$$\mathbf{s} = \frac{s_k - s_{k-1}}{2}\boldsymbol{\tau} + \frac{s_k + s_{k-1}}{2}.$$

- Transform the LGR Weights on $\tau \in [-1, +1)$ to $[s_{k-1}, s_k)$ via

$$\mathbf{w} = \frac{s_k - s_{k-1}}{2}\mathbf{W}$$

- Compute Differentiation Matrix, $\mathbf{d}$, on $\tau \in [-1, +1)$
- Transform $\mathbf{d}$ from $\tau \in [-1, +1)$ to $\mathbf{D}$ in $s \in [s_{k-1}, s_k)$ via

$$\mathbf{D} = \frac{2}{s_k - s_{k-1}}\mathbf{d}$$

- $\mathbf{s} \in \mathbb{R}^N$, $\mathbf{w} \in \mathbb{R}^N$, and $\mathbf{D} \in \mathbb{R}^{N\times(N+1)}$ Used to Construct NLP Functions
- Construction of Objective Function

$$J = \Phi(\mathbf{X}_1, t_0, \mathbf{X}_{N+1}, t_f) + \frac{t_f - t_0}{2}\sum_{k=1}^{N} w_i \mathcal{L}(\mathbf{X}_k, \mathbf{U}_k)$$

- Construction of N Defect Constraints

$$\boldsymbol{\Delta}_i = \sum_{j=1}^{N+1} D_{ij}\mathbf{X}_j - \frac{t_f - t_0}{2}\mathbf{f}(\mathbf{X}_i, \mathbf{U}_i) = 0, \quad i = 1, \ldots, N, \; k = 1, \ldots, n$$

- **Note**
  - ▷ Left-Hand Side of Defects: LGR Points *Plus* Final Point

$$\sum_{j=1}^{N+1} D_{ij}\mathbf{X}_j \Longleftarrow \text{Note That Summation Runs From } 1 \rightarrow N+1$$

  - ▷ Right-Hand Side of Defects: Only LGR Points

$$\frac{t_f - t_0}{2}\mathbf{f}(\mathbf{X}_i, \mathbf{U}_i) \leftarrow \text{Note That } i \in (1, \ldots, N)$$

Construction of NLP for SNOPT

- SNOPT Needs Following Information
  - Bounds on NLP Variables and Constraints
  - Objective Function and Constraints File
  - Gradient of Objective and Jacobian of Constraints
  - Sparsity Pattern of Objective/Constraints
- NLP Functions (Rows)
  - First Row for Objective Function Gradient
  - Discretized Continuous Functions: Defect and Path Constraints
  - Discretized Point Functions: Boundary Conditions
- NLP Variables (Columns)
  - Components of State at LGR Points Plus Final Point
  - Components of Control at LGR Points Plus Final Point
  - Initial and Terminal Values of Time

More Details on Sparsity Pattern

- Defect Constraints Consist of Main-Diagonals and Off-Diagonals
- Main Diagonal
  - ▷ Derivative of i^{th} differential equation with respect to i^{th} state
  - ▷ Contains differentiation matrix
- Off Diagonal
  - ▷ Derivative of i^{th} differential equation with respect to j^{th} state $(i \neq j)$
  - ▷ Does Not Contain Differentiation Matrix
- Derivatives of Defects with Respect to Controls: Same as Off-Diagonal Defect
- Derivatives of Path Constraints: Same as Off-Diagonal Defect
- Boundary Conditions: Only Derivatives with Respect to Endpoint Values of State and Time

Bounds on NLP Variables

- Bounds on Variables Corresponding to State (LGR Points Plus Final Point)

$$
\begin{aligned}
\mathbf{x}_{0,\min} &\leq \mathbf{X}_1 \leq \mathbf{x}_{0,\max} && \leftarrow \text{Initial Condition=First LGR Point} \\
\mathbf{x}_{\min} &\leq \mathbf{X}_2 \leq \mathbf{x}_{\max} && \leftarrow \text{First Interior Point} \\
&\vdots \qquad\qquad \vdots && \\
\mathbf{x}_{\min} &\leq \mathbf{X}_N \leq \mathbf{x}_{\max} && \leftarrow \text{Last Interior Point=Last LGR Point} \\
\mathbf{x}_{f,\min} &\leq \mathbf{X}_{N+1} \leq \mathbf{x}_{f,\max} && \leftarrow \text{Terminal Point (Not an LGR Point)}
\end{aligned}
$$

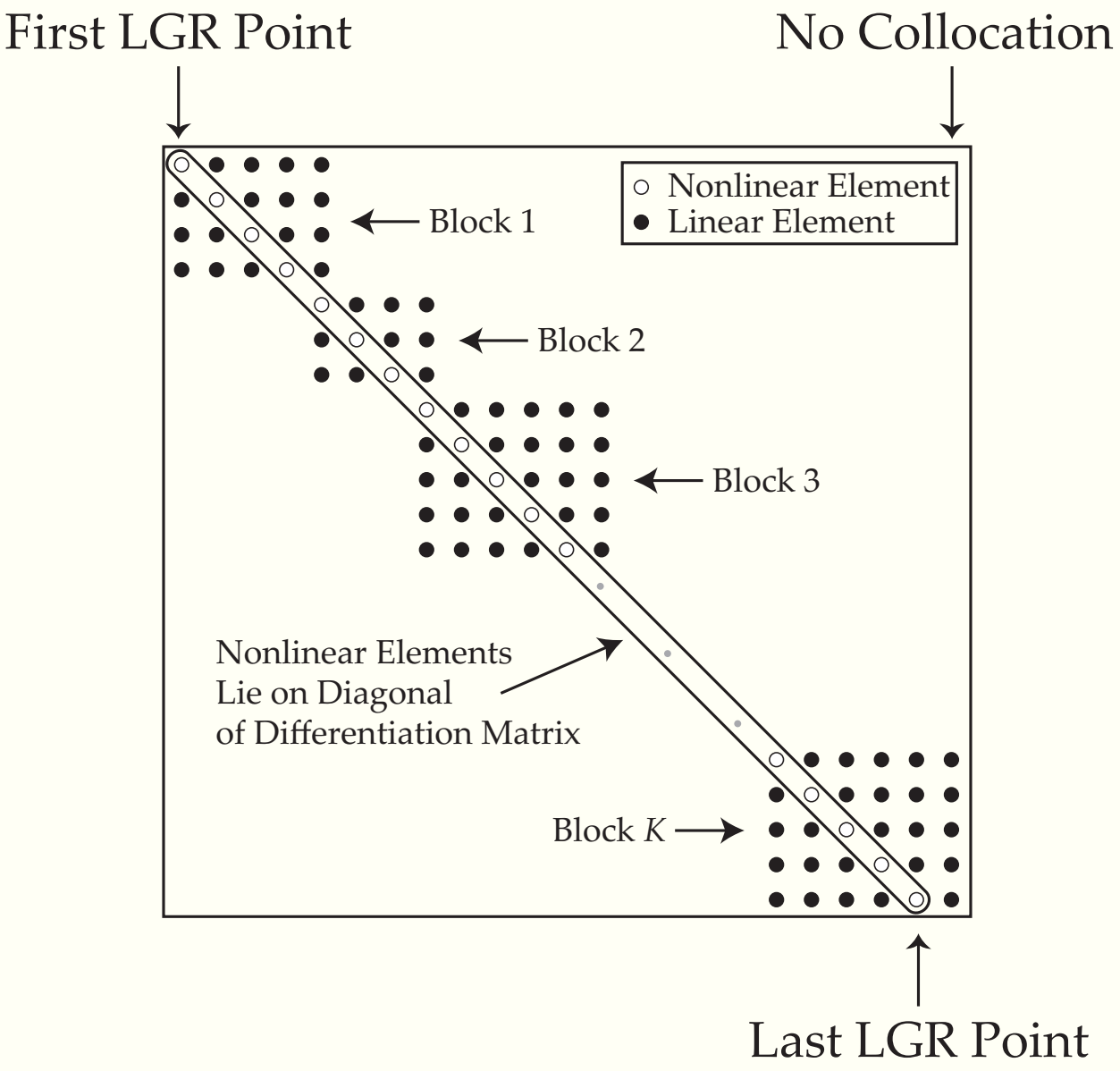
- Bounds on Variables Corresponding to Control

$$
\begin{aligned}
\mathbf{u}_{\min} &\leq \mathbf{U}_1 \leq \mathbf{u}_{\max} && \leftarrow \text{First LGR Point} \\
&\vdots && \\
\mathbf{u}_{\min} &\leq \mathbf{U}_N \leq \mathbf{u}_{\max} && \leftarrow \text{Last LGR Point}
\end{aligned}
$$

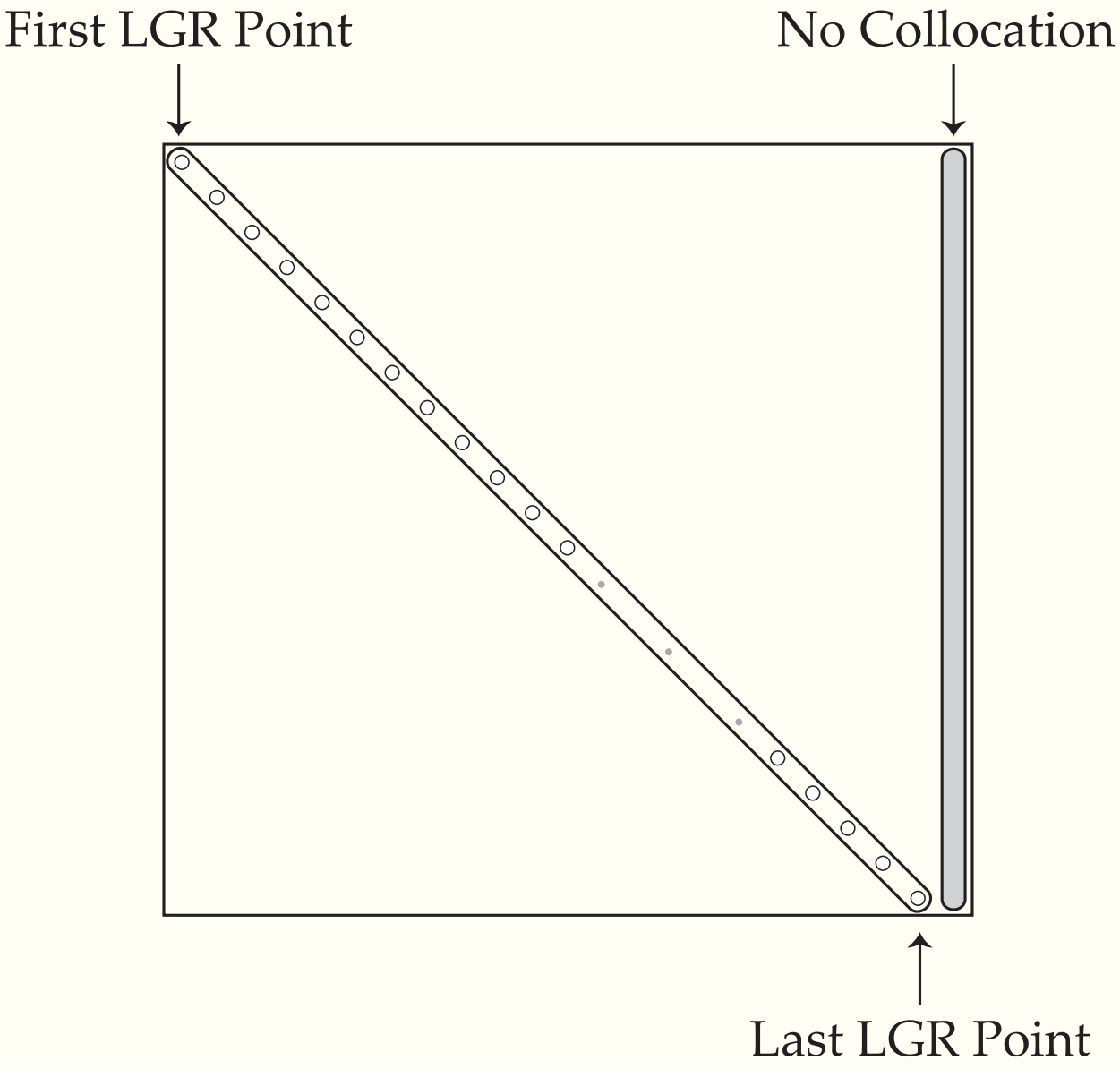
- Bounds on Initial and Terminal Time

$$
\begin{aligned}
t_{0,\min} &\leq t_0 \leq t_{0,\max} && \leftarrow \text{Bounds on Initial Time} \\
t_{f,\min} &\leq t_f \leq t_{f,\max} && \leftarrow \text{Bounds on Terminal Time}
\end{aligned}
$$

Main Diagonal Block of Defect Constraint



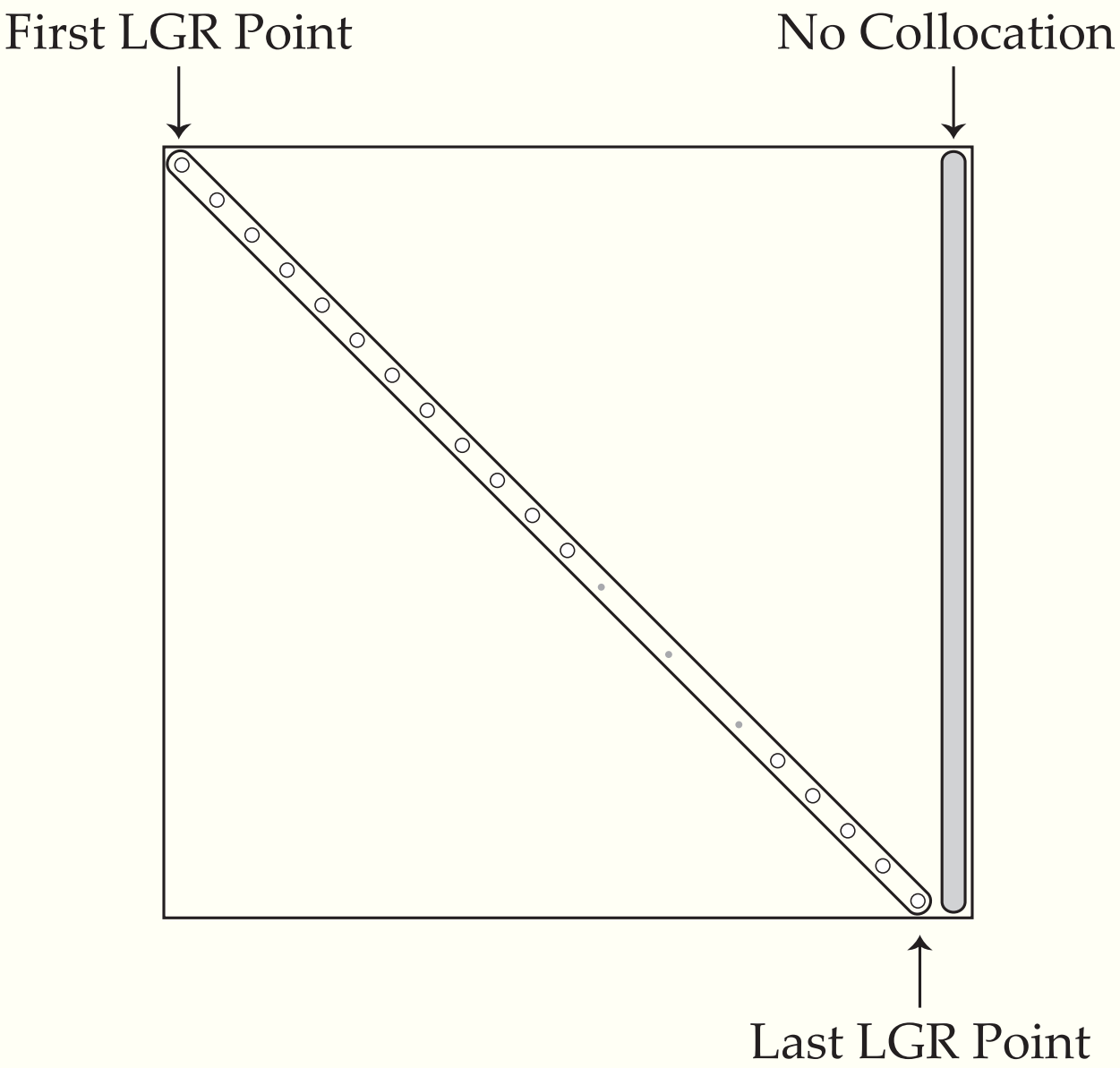
Off-Diagonal Block of Defect Constraint



Path Constraints and Boundary Conditions

- Path Constraints
  - ▷ Included Identically to Off-Diagonals of Defect Constraints
  - ▷ Result: Path Constraint Sparsity Pattern is Purely Nonlinear
- Boundary Conditions Are Functions of $\mathbf{X}_1$, $\mathbf{X}_{N+1}$, t_0 and t_f

Path Constraint Sparsity Pattern Block



Quantities on Diagonal Blocks

- State Diagonal Blocks

$$\left[\begin{array}{cc} \mathbf{D}_{1:N} - \frac{t_f - t_0}{2}\mathrm{diag}\left(\frac{\partial f_i}{\partial x_i}\right) & \mathbf{D}_{N+1} \end{array} \right] \in \mathbb{R}^{N\times(N+1)}$$

- State Off-Diagonal Blocks

$$\left[\begin{array}{cc} -\frac{t_f - t_0}{2}\mathrm{diag}\left(\frac{\partial f_i}{\partial x_j}\right) & \mathbf{0} \end{array} \right] \in \mathbb{R}^{N\times(N+1)}$$

- Control Blocks

$$-\frac{t_f - t_0}{2}\mathrm{diag}\left(\frac{\partial f_i}{\partial u_j}\right) \in \mathbb{R}^{N\times N}$$

- State Path Blocks

$$\left[\begin{array}{cc} \mathrm{diag}\left(\frac{\partial c_i}{\partial x_j}\right) & \mathbf{0} \end{array} \right] \in \mathbb{R}^{N\times(N+1)}$$

- Control Path Blocks

$$\mathrm{diag}\left(\frac{\partial c_i}{\partial u_j}\right) \in \mathbb{R}^{N\times N}$$

- Boundary Conditions With Respect to Initial and Terminal State

$$\begin{bmatrix} \frac{\partial b_1}{\partial x_1} & \mathbf{0}_{1\times(N-1)} & \frac{\partial b_1}{\partial x_2} & \mathbf{0}_{1\times(N-1)} & \frac{\partial b_1}{\partial x_3} & \cdots & \mathbf{0}_{1\times(N-1)} & \frac{\partial b_1}{\partial x_n} \\ \frac{\partial b_2}{\partial x_1} & \mathbf{0}_{1\times(N-1)} & \frac{\partial b_2}{\partial x_2} & \mathbf{0}_{1\times(N-1)} & \frac{\partial b_1}{\partial x_3} & \cdots & \mathbf{0}_{1\times(N-1)} & \frac{\partial b_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \\ \frac{\partial b_{n_b}}{\partial x_1} & \mathbf{0}_{1\times(N-1)} & \frac{\partial b_{n_b}}{\partial x_2} & \mathbf{0}_{1\times(N-1)} & \frac{\partial b_1}{\partial x_3} & \cdots & \mathbf{0}_{1\times(N-1)} & \frac{\partial b_{n_b}}{\partial x_n} \end{bmatrix}$$

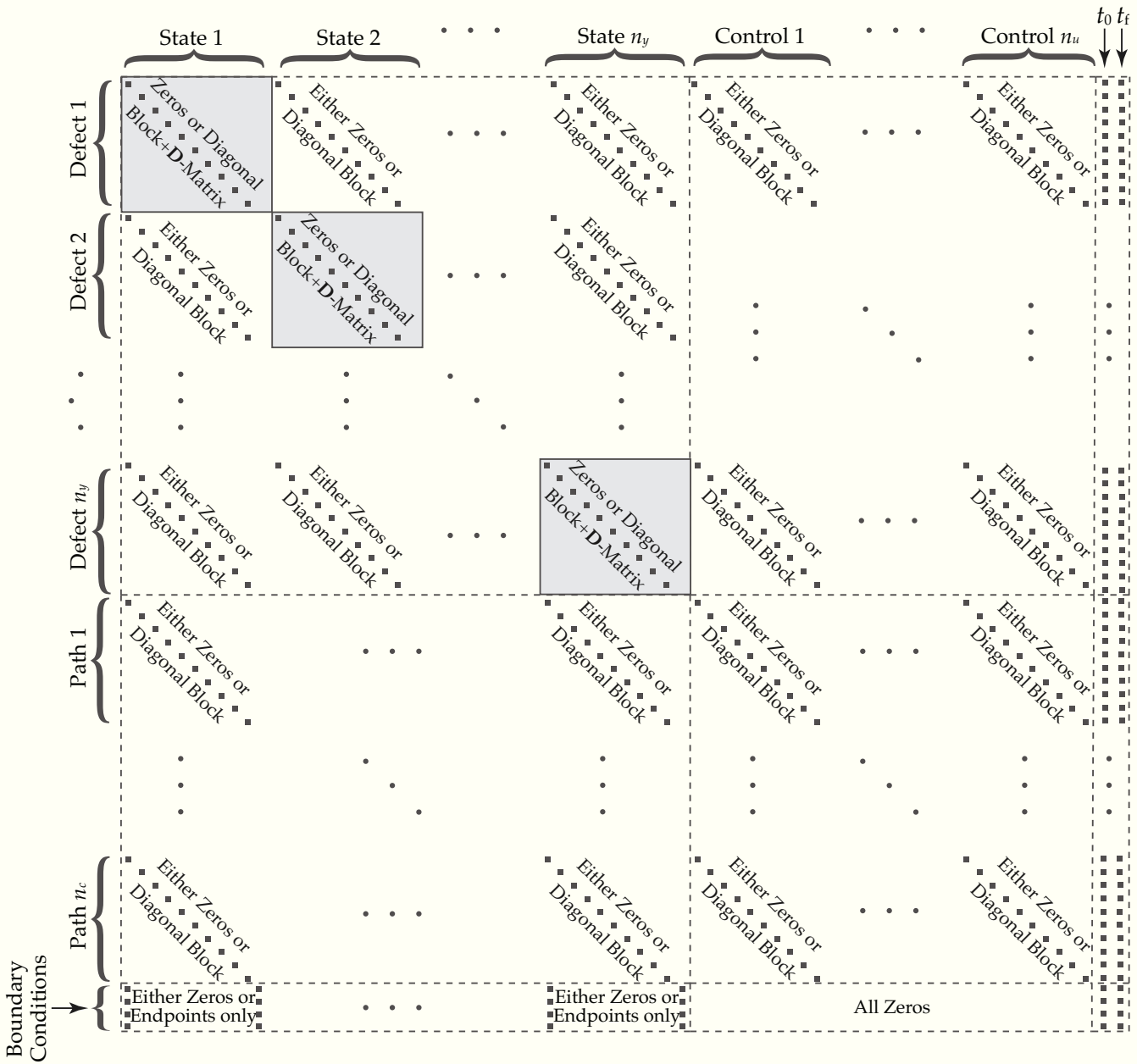
- Final Two Columns for Derivatives With Respect to t_0 and t_f
  - ▷ Derivatives with Respect to t_0

$$\begin{bmatrix} -\frac{1}{2}\left[f_1\right]_N^1 + \frac{t_f - t_0}{2}\frac{\partial f_1}{\partial t_0} \\ \vdots \\ -\frac{1}{2}\left[f_{n_x}\right]_N^1 + \frac{t_f - t_0}{2}\frac{\partial f_n}{\partial t_0} \\ \frac{\partial c_1}{\partial t_0} \\ \vdots \\ \frac{\partial c_{n_c}}{\partial t_0} \\ \frac{\partial b_1}{\partial t_0} \\ \vdots \\ \frac{\partial b_{n_b}}{\partial t_0} \end{bmatrix}$$

▷ Derivatives with Respect to t_f

$$\begin{bmatrix} \frac{1}{2}\left[f_1\right]_N^1 + \frac{t_f - t_0}{2}\left[\frac{\partial f_1}{\partial t_f}\right]_N^1 \\ \vdots \\ \frac{1}{2}\left[f_n\right]_N^1 + \frac{t_f - t_0}{2}\left[\frac{\partial f_n}{\partial t_f}\right]_N^1 \\ \frac{\partial c_1}{\partial t_f} \\ \vdots \\ \frac{\partial c_{n_c}}{\partial t_f} \\ \frac{\partial b_1}{\partial t_f} \\ \vdots \\ \frac{\partial b_{n_b}}{\partial t_f} \end{bmatrix}$$

Complete Sparsity Pattern for RPM Constraint Jacobian



Example 1

- Minimize the Cost Functional

$$J = \frac{1}{2}\int_{t_0}^{t_f}(x^2 + u^2)dt$$

- Dynamic Constraint

$$\dot{x} = f(x,u) = -x^3 + u$$

- Boundary Conditions

$$\begin{aligned} x(0) &= x_0 (\leftrightarrow x_1 = x_0) \\ x(t_f) &= x_f (\leftrightarrow x_{N+1} = x_f) \end{aligned}$$

- This Example: t_0 and t_f are *Fixed*

Bounds on NLP Variables

- Bounds on Variables Corresponding to State (LGR Points Plus Final Point)

$$
\begin{array}{ccccccc}
x_0 & \leq & x_1 & \leq & x_0 & \leftarrow & \text{Initial Condition=First LGR Point} \\
x_{\min} & \leq & x_2 & \leq & x_{\max} & \leftarrow & \text{First Interior Point} \\
 & & \vdots & & & & \vdots \\
x_{\min} & \leq & x_N & \leq & x_{\max} & \leftarrow & \text{Last Interior Point=Last LGR Point} \\
x_f & \leq & x_{N+1} & \leq & x_f & \leftarrow & \text{Terminal Point (Not an LGR Point)}
\end{array}
$$

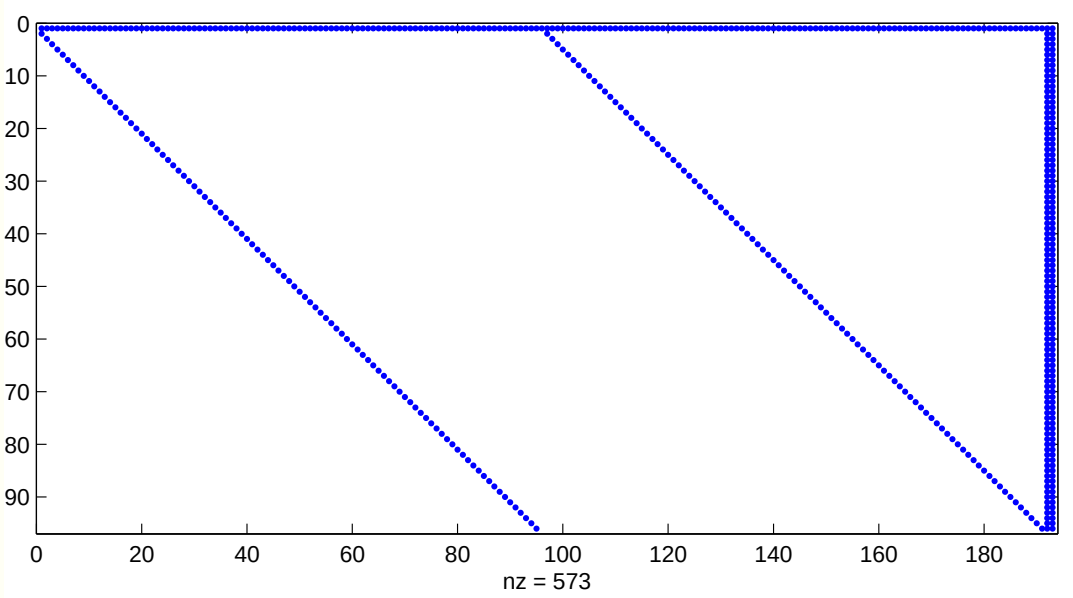
- Bounds on Variables Corresponding to Control

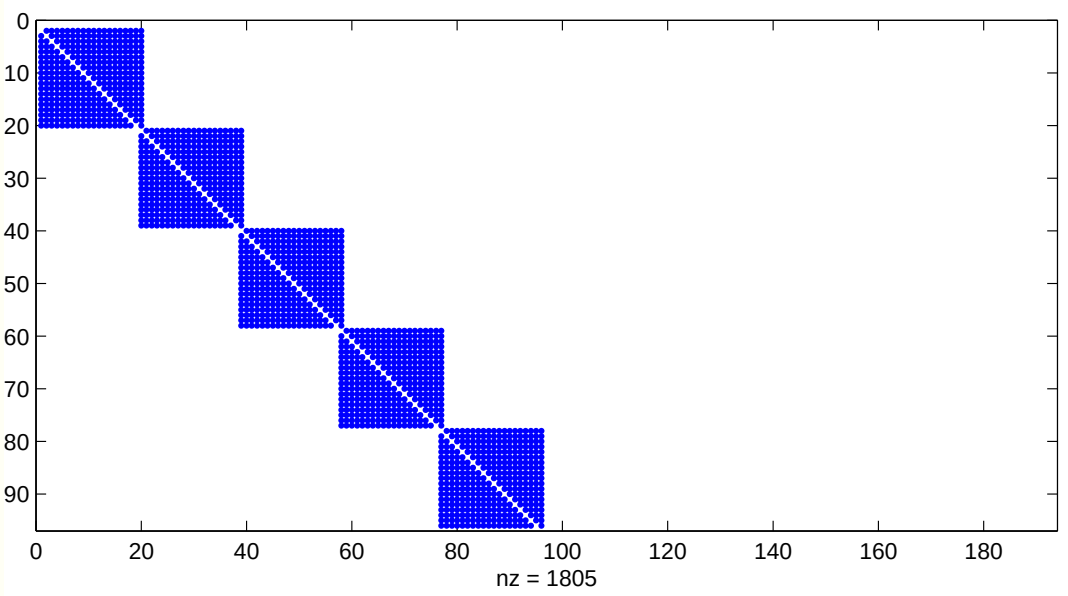
$$
\begin{array}{ccccc}
u_{\min} & \leq u_1 & \leq & u_{\max} & \leftarrow \text{First LGR Point} \\
 & & \vdots & & \\
u_{\min} & \leq u_N & \leq & u_{\max} & \leftarrow \text{Last LGR Point}
\end{array}
$$

- Bounds on Initial and Terminal Time

$$
\begin{array}{ccccc}
t_{0,\min} & \leq t_0 & \leq & t_{0,\max} \\
t_{f,\min} & \leq t_f & \leq & t_{f,\max}
\end{array}
$$

Sparsity Patterns for Hyper-Sensitive Problem





Example 2

- Minimize the Cost Functional

$$J = -r(t_f)$$

- Dynamic Constraints ($\mu = 1$)

$$\begin{aligned} \dot{r} &= u & , \quad \dot{u} &= \frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T}{m} w_1 \\ \dot{v} &= -\frac{uv}{r} + \frac{T}{m} w_2 & , \quad \dot{m} &= -\frac{T}{g_0 I_{sp}} \end{aligned}$$

- Boundary Conditions

$$\begin{aligned} r(0) &= r_0 = 1 & , \quad u(0) &= 0 \\ v(0) &= \sqrt{\mu/r_0} = 1 & , \quad m(0) &= 1 \\ r(t_f) &= r_f = 1 & , \quad u(t_f) &= 0 \\ v(t_f) &= \sqrt{\mu/r_f} & , \quad m(t_f) &= \text{FREE} \end{aligned}$$

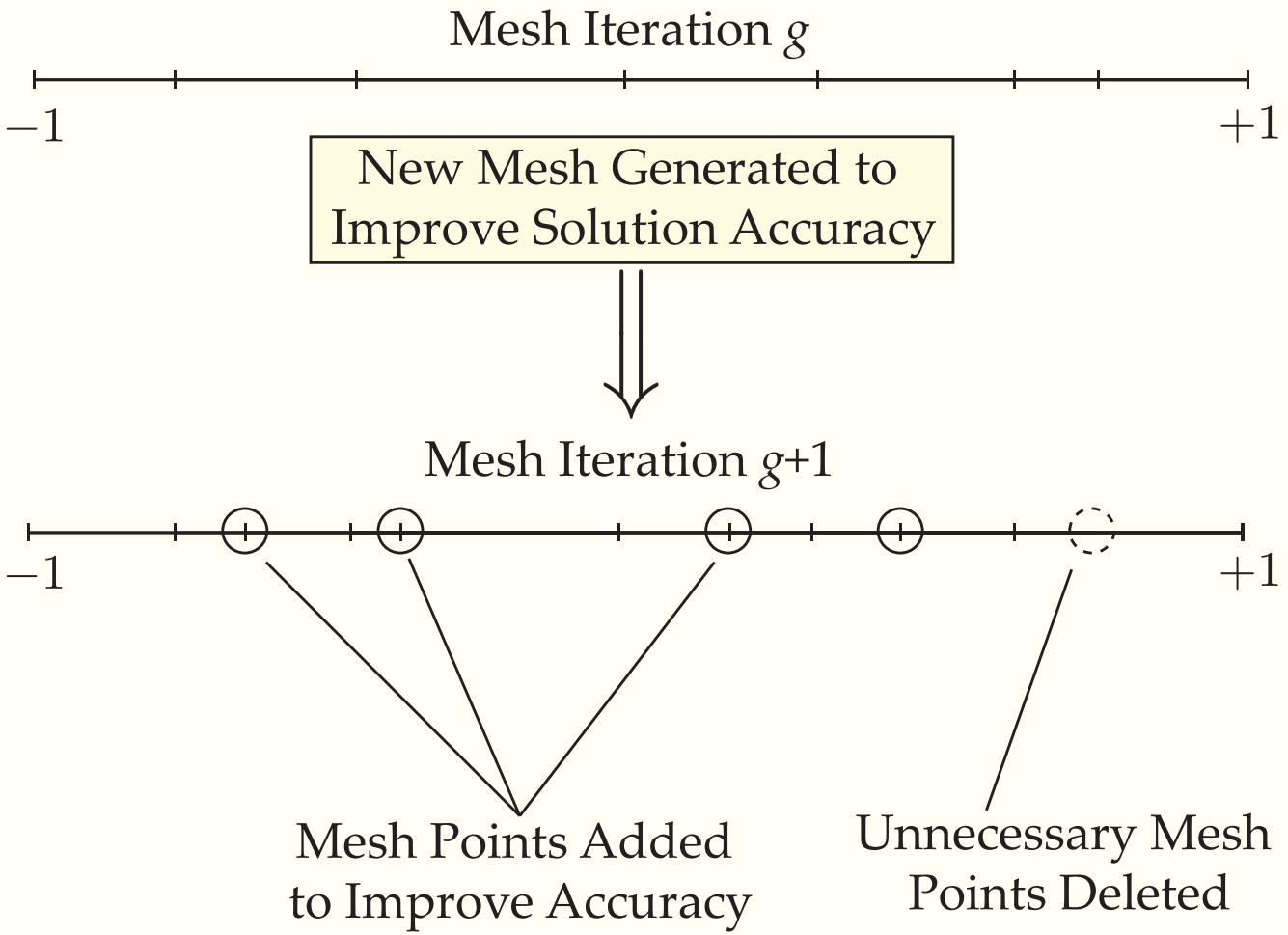
Mesh Refinement

- Usually Desire Solutions of a Specified Accuracy
- Critical to Properly Choose Mesh Points and Number of Collocation Points
- A Priori Knowledge Not Possible in General
- Iterative Method Required to Construct Efficient Mesh
- This Part of Course
  - ▷ Discussion of Mesh Refinement Algorithm
  - ▷ Specifically, will Discuss *hp*–Adaptive Mesh Refinement

Concept of hp–Adaptive Mesh Refinement

- Limitations with Global Pseudospectral Method
  - ▷ Accuracy Increased Simply By Increasing Degree of Global Approximation
    - ◦ Computationally intractable as N increases due to global $\mathbf{D}$ Matrix
    - ◦ Error tolerance not achievable using global polynomial approximation
  - ▷ Need to Divide Time Interval Into Segments
    - ◦ Increases NLP sparsity
    - ◦ Makes high-accuracy solutions possible
- Efficient Implementations of Methods Require Use of
  - ▷ As few mesh intervals as possible
  - ▷ The lowest possible polynomial degree approximation in each mesh interval

Generation of New Mesh



Degree of Approximating Polynomial in Each Mesh Interval May Change from Grid g to Grid g+1

Setting an Initial Mesh

- Solution to General Problem Not Known A Priori
- In Particular, Following are Not Known:
  - Regions where solution changes rapidly
  - Locations of discontinuities in control or state derivative
- Strategy for Initial Mesh
  - Choose Small Number of Evenly Spaced Mesh Intervals
  - Use Low-Order Approximation in Each Interval
- Strategy Enables Fast Computation of Solution on First Grid

How to Change or Add Mesh Intervals

- Recall Terminology:
  - ▷ Mesh intervals $[s_{k-1}, s_k]$, $k = 1, \ldots, K$
  - ▷ N_k =degree of polynomial approximation in mesh interval k
- Mesh Intervals Can Be Changed in One of Two Ways
  - ▷ Degree of polynomial approximation can be increased
  - ▷ Location of s_{k-1} and/or s_k can be changed
- Criteria for Changing Polynomial Degree N_k
  - ▷ Need to assess if solution in mesh interval k is smooth
  - ▷ If so, can use global polynomial approach and increase N_k
- Criteria for Changing s_{k-1} and/or s_k
  - ▷ Need to assess if solution is either nonsmooth/rapidly changing
  - ▷ If so, need to determine locations where nonsmoothness occurs
  - ▷ Divide mesh interval at locations of predicted rapid changes

Determining if a Mesh Interval Needs to Be Modified

- Choose an Integer L
- Let $\left(\bar{s}_1^{(k)}, ..., \bar{s}_L^{(k)}\right)$ Be Arbitrary Points in Mesh Interval k
- Compute a Differentiation Matrix, $\bar{\mathbf{D}}$, Using $\left(\bar{s}_1^{(k)}, ..., \bar{s}_L^{(k)}\right)$
- Compute $\dot{x}_i(s)$ at $\left(\bar{s}_1^{(k)}, ..., \bar{s}_L^{(k)}\right)$ Using $\bar{\mathbf{D}}$
- Compute Error in Dynamic Constraints at $\left(\bar{s}_1^{(k)}, ..., \bar{s}_L^{(k)}\right)$

$$\left|\dot{x}_i^{(k)}(\bar{t}_l^{(k)}) - f_i^{(k)}(\mathbf{X}_l^{(k)}, \mathbf{U}_l^{(k)}, \bar{t}_l^{(k)})\right| \quad = \quad a_{li}^{(k)}, \quad i \in [1, \ldots, L] \; j \in [1, \ldots, n].$$

- Use a_{li} To Determine if Mesh in Interval k Should Be Modified

Determining if a Mesh Interval Needs to Be Modified (Continued)

- Let $x_m^{(k)}(s)$ Be Component of State with Largest Value of $a_{li}^{(k)}$
- Let $r_{\max}$ Be a User-Defined Parameter
- Compute Curvature in Mesh Interval k

$$\kappa^{(k)}(s) = \frac{|\ddot{x}_m^{(k)}(s)|}{\left|\left[1 + \dot{x}_m^{(k)}(s)^2\right]^{3/2}\right|}.$$

- Let $\kappa_{\max}^{(k)}$ =Maximum Curvature
- Let $\bar{\kappa}^{(k)}$ =Average Curvature
- Let $r_k = \kappa_{\max}^{(k)}/\bar{\kappa}^{(k)}$
- If $r_k < r_{\max}$, Then Polynomial Degree N_k Must Be Increased
- If $r_k > r_{\max}$, Then Mesh Interval Must Be Divided

Increasing Degree of Polynomial or Refining Mesh

- Increasing Polynomial Degree
  - ▷ New Polynomial Degree, N_k, Computed As

$$N_k = M + \mathbf{ceil}(\log_{10}(e_{\max}^{(k)}/\epsilon)) + 1$$

  - ▷ $e_{\max}^{(k)}$ =Maximum Value of $a_{li}^{(k)}$
- Refining Mesh
  - ▷ New Number of Mesh Intervals Computed As

$$n_k = \mathbf{ceil}(2\log_{10}(e_{\max}^{(k)}/\epsilon)),$$

  - ▷ Locations of New Mesh Points Determined from κ
    - ◦ Let $\rho(\tau)$ be given as

$$\rho(s) = c\kappa(s)^{1/3},$$

where c is a constant chosen so that

$$\int_{-1}^{+1} \rho(\zeta)d\zeta = 1.$$

- Let $F(\tau)$ be given as

$$F(s) = \int_{-1}^{s} \rho(\zeta) d\zeta.$$

- New Mesh Points, n_k, Chosen So That

$$F(s_i) = \frac{i-1}{n_k}, \quad 1 \leq i \leq n_k + 1.$$

hp–Adaptive Mesh Refinement Algorithm

1. Solve the NLP using the current mesh.

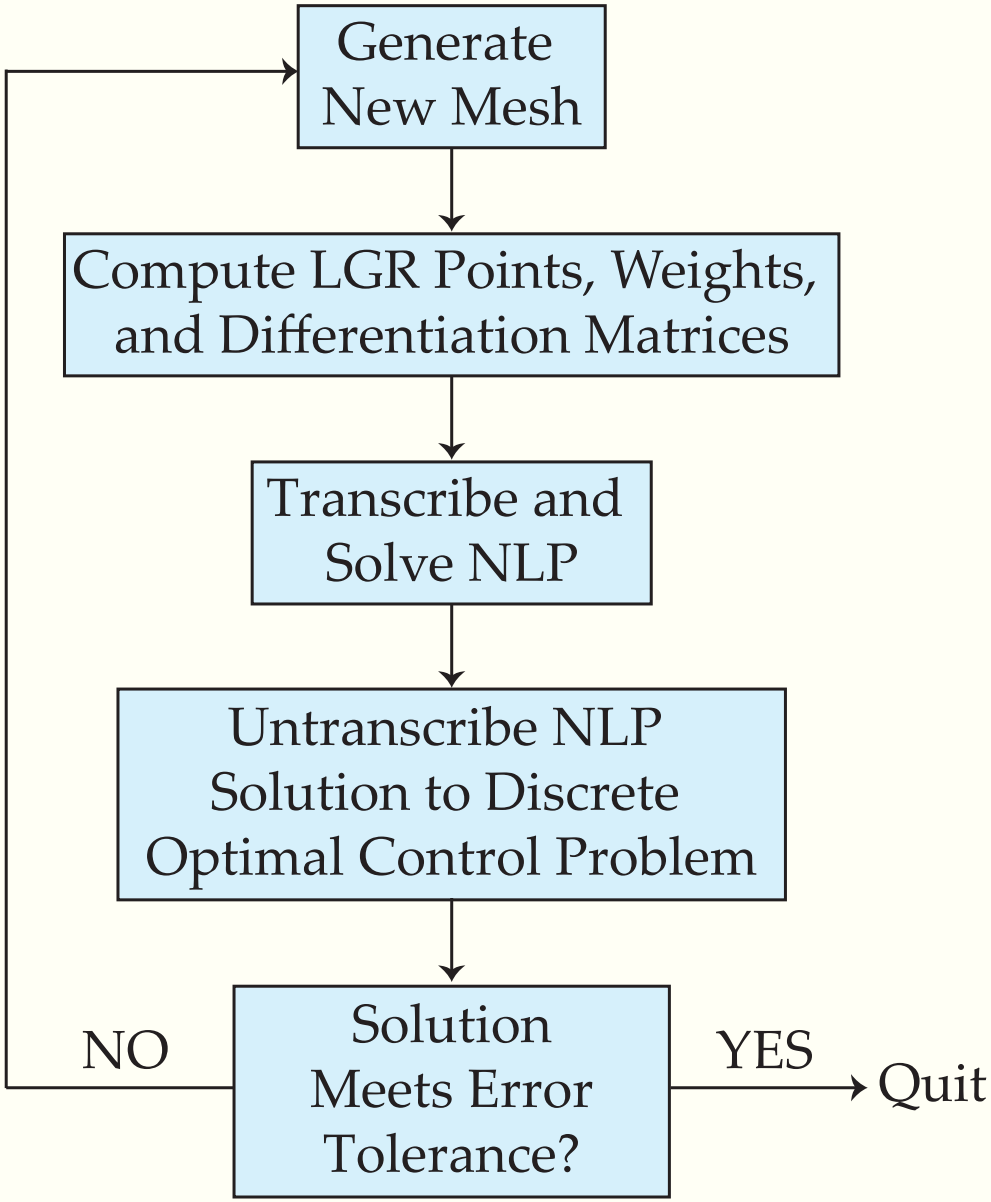
   Begin: For $k = 1, \ldots, K$,

   2. If $e_{\max}^{(k)} \leq \epsilon$, then continue (proceed to next k).
   3. If either $r_k \geq r_{\max}$ or $N_k > M$, then refine the k^{th} mesh interval into n_k subintervals, Set the degree of the polynomials on each of the subintervals to be M and proceed to the next k.
   4. Otherwise, set the degree of the polynomials on the k^{th} subinterval to be N_k.

   End: For $k = 1, \ldots, K$**.**

5. Return to Step 1 Until Error Tolerance is Met.

Schematic of Algorithm to Solve Optimal Control Problem



Generation of Derivatives for NLP Solver

- NLP Solvers Need Derivatives
- Quasi-Newton NLP Solvers (e.g., SNOPT)
  - Objective Function Gradient
  - Constraint Jacobian
- Typical Methods for Generating NLP Derivatives
  - Finite-Differencing
  - Analytic Differentiation
  - Automatic Differentiation

Generating NLP Derivatives for a Pseudospectral Method

- Pseudospectral Methods Have Great Deal of Structure
- Key Point: NLP Derivatives Can Be Computed Very Efficiently
- Suppose NLP Given in Following Form
  - ▷ Minimize $f(\mathbf{z})$
  - ▷ Constraints: $\mathbf{h}_{\min} \leq \mathbf{h}(\mathbf{z}) \leq \mathbf{h}_{\max}$
- Need to Compute $\nabla_{\mathbf{z}} f$ and $\nabla_{\mathbf{z}} \mathbf{h}$
- Using Structure, Only Optimal Control Functions Need to Be Differentiated
- Details Provided in Recent Paper Accepted in *Journal of Spacecraft and Rockets*

Benefit of Computing Derivative of Optimal Control Functions

- Differentiating NLP Functions Inefficient and Memory Intensive
- Optimal Control Functions Much *Smaller* Than NLP Functions
- Each Collocation Condition: Only Involves *One Value of Right-Hand Side*
- Optimal Control Functions Only Differentiated at Collocation Points
- Optimal Control Functions Can Be Differentiated Using Any Technique
  - Finite-Differencing
  - Complex-Step Derivative Approximation
  - Analytic Differentiation
  - Automatic Differentiation
- Complex-Step is Best Choice
  - Produces Extremely Accurate First Derivative
  - Perfect for Quasi-Newton NLP Solver

Complex-Step Derivative Approximation

- Basic Idea: Given a Function of a Complex Variable $z = x + iy$
- If $f(z) = u(x, y) + iv(x, y)$ is Analytic, Then $f(z)$ Satisfies Cauchy-Riemann Equations

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \quad , \quad \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}$$

- We Then Obtain Following Result (Where h is Real):

$$\frac{\partial u}{\partial x} = \lim_{h \to 0} \frac{v(x + i(y + h)) - v(x + iy)}{h}$$

- Setting $y = 0$, $u(x) = f(x)$, and $v(x) = 0$, We Obtain

$$\frac{df}{dx} = \lim_{h \to 0} \frac{\mathrm{Im}[f(x + ih)]}{h}$$

- Therefore

$$\frac{df}{dx} \approx \frac{\mathrm{Im}[f(x + ih)]}{h}$$

- Notice That Approximation Has *No Subtraction*
- Thus, Complex-Step Can Be Extremely Small

- Also, Derivative is Accurate to $\mathcal{O}(h^2)$
- Easy to Implement on a Computer Using Complex Arithmetic
  - ▷ All Analytic Functions Defined for Complex Arguments
  - ▷ Must Avoid Non-Differentiable Functions **abs**, **max**, and **min**

Using Complex-Step in Pseudospectral Method

- At Endpoints of Interval, Compute Derivative Approximations of
  - Mayer Cost
  - Boundary Conditions
- At Collocation Points, Compute Derivative Approximations of
  - Lagrange cost
  - Right-hand side of differential equations
  - Path constraints
- Insert Optimal Control Derivatives Into NLP Derivative Functions

General Approach to Software Design

- Suppose We Are *Given* Mesh Points $(s_1, \ldots, s_K)$ on $s \in [-1, +1]$
- Develop Following Subroutines First *Given* $(s_1, \ldots, s_K)$
  - ▷ LGR Points, Weights, and Differentiation Matrix
  - ▷ NLP Sparsity Pattern Generator
  - ▷ NLP Functions
    - ○ Objective Function: Discrete Version of Bolza Cost
    - ○ Constraint Function: Defects, Paths, Boundary Conditions
  - ▷ SNOPT Wrapper, Where SNOPT Calls Objective and Constaints
- Repeat Above Steps
  - ▷ Using $(s_1, \ldots, s_K)$ computed from mesh refinement
  - ▷ Until user-specified accuracy tolerance on mesh is met

Part VIII: Wrap-Up

Review of Topics Covered

- Calculus of Variations
- Optimal Control Theory
- Nonlinear Optimization
- Numerical Integration and Interpolation
- Foundations of Pseudospectral Methods
- Practical Implementation Using Radau Pseudospectral Method

Summary

- Numerical Methods for Differential Equations
  - Single-Step Methods
  - Multiple-Step Methods
  - Multiple-Stage Methods
- Explicit and Implicit Schemes
  - Explicit
    - Easier to Implement
    - Has Poorer Stability Properties
  - Implicit
    - More Work to Implement
    - Much Better Stability and Accuracy