

Algorithmic Choices
When Solving
an Optimal Control Problem

John T. Betts *

*Partner, Applied Mathematical Analysis, LLC

Introduction

- Designing An Algorithm to Solve a Complicated Problem Requires Choices
- The Problem: Optimal Control
- The Choices:
 - Indirect vs Direct Formulation
 - “Optimize then Discretize” vs “Discretize the Optimize”
 - Boundary Value Problem: Shooting vs Collocation
 - Polynomial Approximation: High vs Low Order
 - Hessian Approximation: Quasi-Newton vs Newton
 - Nonlinear Programming (NLP): Interior Point vs SQP

Optimal Control Problem

- *State equations*

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), t] \quad t_I \leq t \leq t_F$$

state vector \mathbf{y} , control vector \mathbf{u} .

- *Path constraints*

$$\mathbf{g}_L \leq \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t] \leq \mathbf{g}_U,$$

- *Bounds*

$$\begin{aligned} \mathbf{y}_L &\leq \mathbf{y}(t) \leq \mathbf{y}_U \\ \mathbf{u}_L &\leq \mathbf{u}(t) \leq \mathbf{u}_U. \end{aligned}$$

- *Boundary conditions*

$$\Psi_L \leq \Psi[\mathbf{y}(t_I), \mathbf{u}(t_I), t_I, \mathbf{y}(t_F), \mathbf{u}(t_F), t_F] \leq \Psi_U,$$

- *Objective Function (minimize)*

$$J = \phi[\mathbf{y}(t_I), t_I, \mathbf{y}(t_F), t_F].$$

Optimality Conditions

Calculus of variations leads to differential-algebraic equations (DAE's)

$$\begin{aligned}\dot{\mathbf{y}} &= \mathbf{f}[\mathbf{y}, \mathbf{u}, t] \\ \dot{\boldsymbol{\lambda}} &= -\mathbf{H}_y^\top \\ \mathbf{0} &= \mathbf{H}_u^\top\end{aligned}$$

With no path constraints the *Hamiltonian*

$$H = \boldsymbol{\lambda}^\top(t) \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t)]$$

Path constraints can be included using

$$H = \boldsymbol{\lambda}^\top(t) \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t)] + \mu g^{(m)}$$

where $u(t)$ appears explicitly in $g^{(m)}(t) = d^m g / dt^m$.

- Nonlinear boundary value problem with DAE's
- Active path constraint arcs must be explicitly identified
- Path constraint *index* must be known
- Explicit derivation of optimality conditions required for all arcs.

So What's the Rub?

Iterative Scheme on a Digital Computer

Works with a Finite Set of Variables \mathbf{x} and Functions $F(\mathbf{x}), \mathbf{c}(\mathbf{x})$

⋮

But Optimal Control is an Infinite Dimensional Problem;
i.e. the functions $\mathbf{u}(t)$ and $\mathbf{y}(t)$,
and possibly $\lambda(t)$ and $\mu(t)$

How do we formulate the problem?

Shooting Methods

“Eliminate” Infinite Dimensional Problem by solving

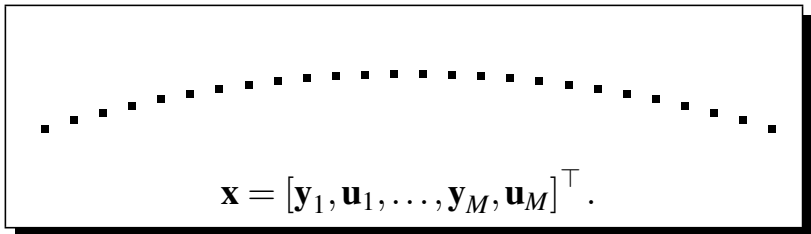
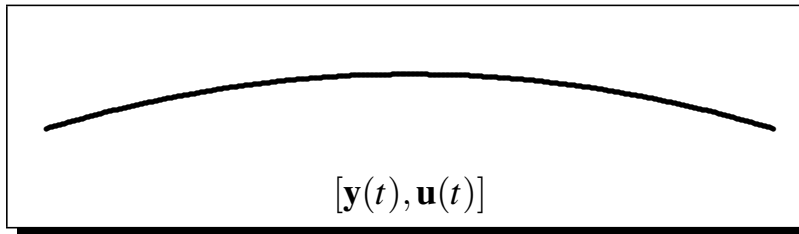
$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), t] \quad \text{and/or} \quad \begin{array}{l} \dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), t] \\ \mathbf{0} = \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t] \end{array}$$

The iteration involves the Finite Set of Boundary Values

BVP is *very* nonlinear
ODE or DAE can be very unstable
ODE error control at suboptimal points \implies inefficient
Iteration Gradients \implies variational equations and/or finite differences
Path inequalities cumbersome (impractical?)
Shooting for Control \iff GRG for NLP

Discretization Methods

Variables



Constraints

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), t]$$

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \frac{h_k}{2} (\mathbf{f}_k + \mathbf{f}_{k+1})$$

Optimal Control Algorithm

Direct Transcription Transcribe the optimal control problem into a nonlinear programming (NLP) problem by discretization;

Sparse Nonlinear Program Solve the sparse NLP

Mesh Refinement Assess the accuracy of the approximation (i.e. the finite dimensional problem), and if necessary refine the discretization, and then repeat the optimization steps.

Sequential Nonlinear Programming (SNLP)

Nonlinear Programming

- Find the n -vector $\mathbf{x} = (x_1, \dots, x_n)$ to *minimize* the scalar objective function

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

subject to the m constraints

$$\mathbf{c}_L \leq \mathbf{c}(\mathbf{x}) \leq \mathbf{c}_U,$$

and the simple bounds

$$\mathbf{x}_L \leq \mathbf{x} \leq \mathbf{x}_U,$$

- Equality constraints imposed by setting $\mathbf{c}_L = \mathbf{c}_U$
- Karush-Kuhn-Tucker (KKT) necessary conditions for optimal point \mathbf{x}^*

- \mathbf{x}^* is feasible
- Lagrange multipliers λ and ν such that

$$\mathbf{g} = \mathbf{G}^T \lambda + \nu$$

- Inequality multipliers \Rightarrow

$$\left\{ \begin{array}{ll} \text{non-positive} & \text{for active upper bounds} \\ \text{zero} & \text{for strictly satisfied constraints} \\ \text{non-negative} & \text{for active lower bounds} \end{array} \right.$$

- *Constraint Qualification Test:* Active constraint Jacobian $\tilde{\mathbf{G}}$ has full row rank

Transcription Formulation

- Discretization: $M = n_s + 1$ grid points; stepsize $h_k \equiv t_{k+1} - t_k$

$$\begin{aligned} \mathbf{y}_k &\equiv \mathbf{y}(t_k) & \bar{\mathbf{u}}_k &\equiv \mathbf{u}[(t_k + t_{k-1})/2] \\ \mathbf{u}_k &\equiv \mathbf{u}(t_k) & \mathbf{f}_k &\equiv \mathbf{f}[\mathbf{y}(t_k), \mathbf{u}(t_k), t_k] \end{aligned}$$

- Trapezoidal

- NLP variables

$$\mathbf{x} = [\mathbf{y}_1, \mathbf{u}_1, \mathbf{y}_2, \mathbf{u}_2, \dots, \mathbf{y}_M, \mathbf{u}_M, t_I, t_F]^\top.$$

- Defect constraints for $k = 1, \dots, n_s$

$$\zeta_k = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{2} [\mathbf{f}_{k+1} + \mathbf{f}_k] = 0$$

- Hermite-Simpson

- NLP variables

$$\mathbf{x} = [\mathbf{y}_1, \mathbf{u}_1, \bar{\mathbf{u}}_2, \mathbf{y}_2, \mathbf{u}_2, \bar{\mathbf{u}}_3, \dots, \mathbf{y}_M, \mathbf{u}_M, t_I, t_F]^\top.$$

- Defect constraints for $k = 1, \dots, n_s$

$$\zeta_k = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{6} [\mathbf{f}_{k+1} + 4\bar{\mathbf{f}}_{k+1} + \mathbf{f}_k] = 0$$

Properties of the Approximation

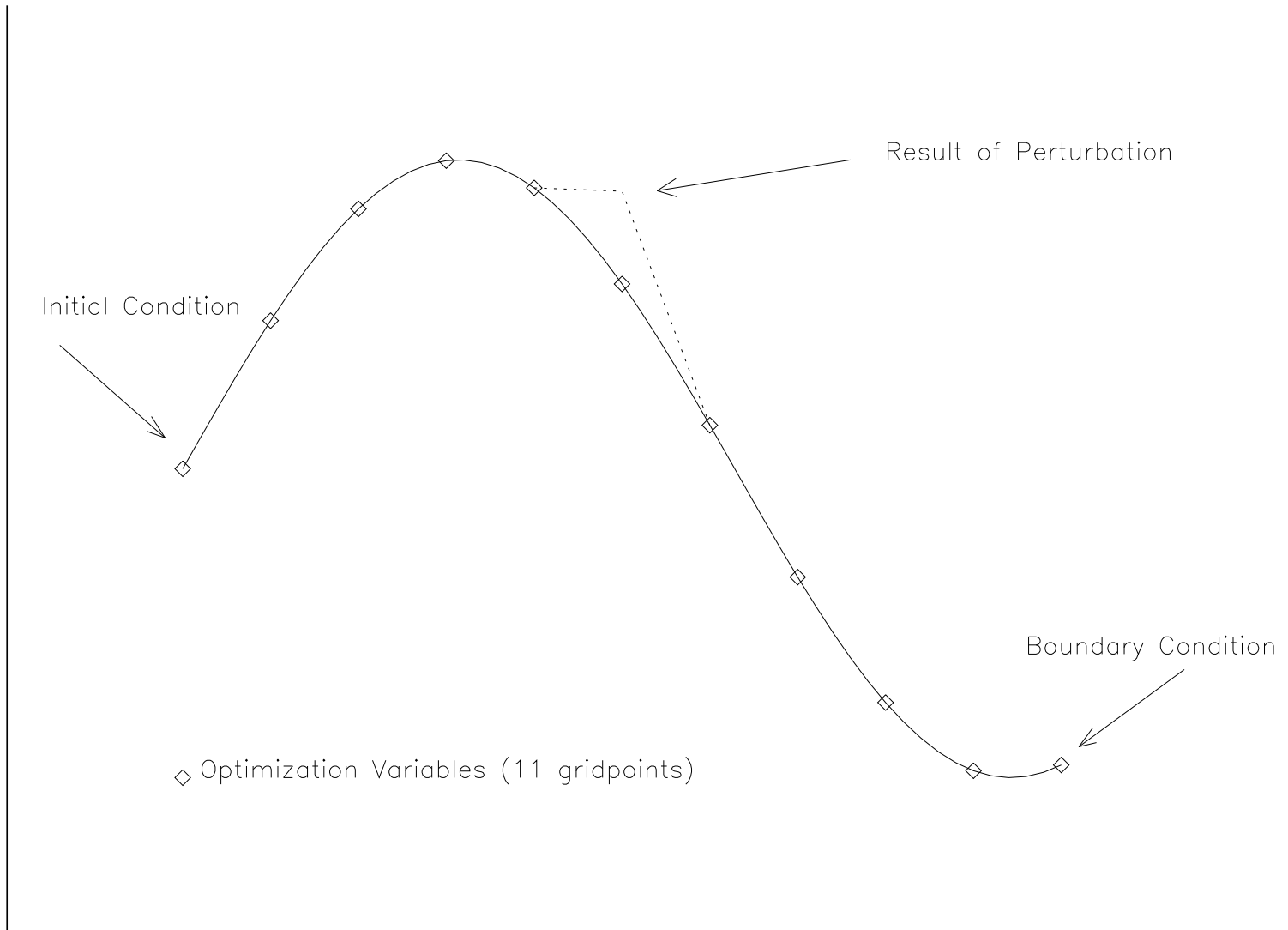
- Implicit Runge-Kutta (IRK) Lobatto IIIA methods.
 - IRK methods are collocation methods
 - Lobatto methods collocate at the mesh points
 - Unique Cubic B-spline polynomial interpolation *between* mesh points
- Why Use a Low Order Piecewise Approximation?
 - Good Approximation Properties:

This allows us to illustrate the essential limitation of polynomial approximation: If the function to be approximated is badly behaved *anywhere* in the interval of approximation, then the approximation is poor *everywhere*. This global dependence on local properties can be avoided when using *piecewise* polynomial approximants*.
 - Good Matrix Sparsity:

This structural difference also strongly influences the construction of approximations, requiring the solution of a full system in the polynomial case and usually only a banded system in the piecewise polynomial case*.

*Carl de Boor

TRANSCRIPTION METHOD



NLP Considerations — Sparsity

- Changing a variable at gridpoint only alters nearby constraints \implies the derivatives of many of the constraints with respect to many of the variables are zero.
- The Jacobian matrix of partial derivatives is sparse.
- The Jacobian matrix is defined as

$$\mathbf{G}_{ij} = \frac{\text{Change in Defect Constraint on Segment } i}{\text{Change in Optimization Variable at Grid Point } j}$$

with m rows (the total number of defect constraints) and n columns (the total number of optimization variables)

- Reduced Sensitivity in Boundary Value Problem \iff Matrix Sparsity

Sparse Difference Derivatives

- First derivatives of ν functions $q_i(\mathbf{x})$ with respect to n variables \mathbf{x} , defines the $\nu \times n$ matrix

$$\mathbf{D} \equiv \begin{bmatrix} (\nabla q_1)^\top \\ (\nabla q_2)^\top \\ \vdots \\ (\nabla q_\nu)^\top \end{bmatrix} = \frac{\partial \mathbf{q}}{\partial \mathbf{x}}$$

- Second derivatives of the function

$$\Upsilon(\mathbf{x}) = \sum_{i=1}^{\nu} \omega_i q_i(\mathbf{x})$$

with respect to n variables \mathbf{x} , defines the $n \times n$ matrix

$$\mathbf{E} \equiv \nabla^2 \Upsilon(\mathbf{x}) \equiv \sum_{i=1}^{\nu} \omega_i \nabla^2 q_i(\mathbf{x})$$

Sparse Differences (Cont.)

- Partition columns of \mathbf{D} into subsets (*index sets*) Γ^k such that each subset has at most one nonzero element per *row*.

- Define perturbation direction vector by

$$\Delta^k = \sum_{j \in \Gamma^k} \delta_j \mathbf{e}_j$$

where δ_j is perturbation size for variable j , and \mathbf{e}_j is unit vector in direction j .

- First derivative estimates for $i = 1, \dots, \nu$ and $j \in \Gamma^k$ are

$$\mathbf{D}_{ij} \approx \frac{1}{2\delta_j} [q_i(\mathbf{x} + \Delta^k) - q_i(\mathbf{x} - \Delta^k)].$$

- Second derivative estimates for $i \in \Gamma^k$ and $j \in \Gamma^\ell$

$$\mathbf{E}_{ij} \approx \frac{1}{\delta_i \delta_j} [\Upsilon(\mathbf{x} + \Delta^k + \Delta^\ell) + \Upsilon(\mathbf{x}) - \Upsilon(\mathbf{x} + \Delta^k) - \Upsilon(\mathbf{x} + \Delta^\ell)].$$

$$\mathbf{E}_{ii} \approx \frac{1}{\delta_i^2} [\Upsilon(\mathbf{x} + \Delta^k) + \Upsilon(\mathbf{x} - \Delta^k) - 2\Upsilon(\mathbf{x})].$$

Sparse Differences (Cont.)

- Denote total number of index sets Γ^k needed to span columns of \mathbf{D} by γ
- Using same index sets for first and second derivatives:
 - First Derivative (central difference) requires 2γ perturbations
 - First and Second Derivatives require $\gamma(\gamma+3)/2$ perturbations
- $\gamma \geq$ Maximum Number of Nonzeros in any row of \mathbf{D}
- $\gamma \ll n$
- **GOAL** : keep γ as small as possible!

The General Approach

- For path constrained problems in semi-explicit form

$$\begin{aligned} \dot{\mathbf{y}} &= \mathbf{f}[\mathbf{y}, \mathbf{u}, t] \\ \mathbf{g}_\ell &\leq \mathbf{g}[\mathbf{y}, \mathbf{u}, t] \leq \mathbf{g}_u, \end{aligned}$$

To exploit separability write transcribed NLP functions as

$$\begin{bmatrix} \mathbf{c}(\mathbf{x}) \\ F(\mathbf{x}) \end{bmatrix} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{q}(\mathbf{x})$$

where \mathbf{A} and \mathbf{B} are constant matrices and \mathbf{q} involves right hand sides at grid points.

- Construct sparsity template for DAE right hand side

$$\begin{bmatrix} \frac{\partial \mathbf{f}}{\partial \mathbf{y}} & \left| \right. & \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \\ \frac{\partial \mathbf{g}}{\partial \mathbf{y}} & \left| \right. & \frac{\partial \mathbf{g}}{\partial \mathbf{u}} \end{bmatrix}$$

- Construct sparsity for \mathbf{D} and compute index sets.

Sparse Differences (Cont.)

- NLP gradient and Jacobian

$$\begin{bmatrix} \mathbf{G} \\ \mathbf{g} \end{bmatrix} = \mathbf{A} + \mathbf{B}\mathbf{D}$$

- Lagrangian:

$$\begin{aligned} L(\mathbf{x}, \lambda) &= \begin{bmatrix} -\lambda^\top, 1 \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ F \end{bmatrix} \\ &= \begin{bmatrix} -\lambda^\top, 1 \end{bmatrix} \mathbf{A}\mathbf{x} + \begin{bmatrix} -\lambda^\top, 1 \end{bmatrix} \mathbf{B}\mathbf{q} \\ &= \boldsymbol{\sigma}^\top \mathbf{x} + \boldsymbol{\omega}^\top \mathbf{q} \end{aligned}$$

- Hessian of Lagrangian

$$\mathbf{H}_L = \mathbf{E}$$

with $\boldsymbol{\omega}^\top = [-\lambda^\top, 1]\mathbf{B}$

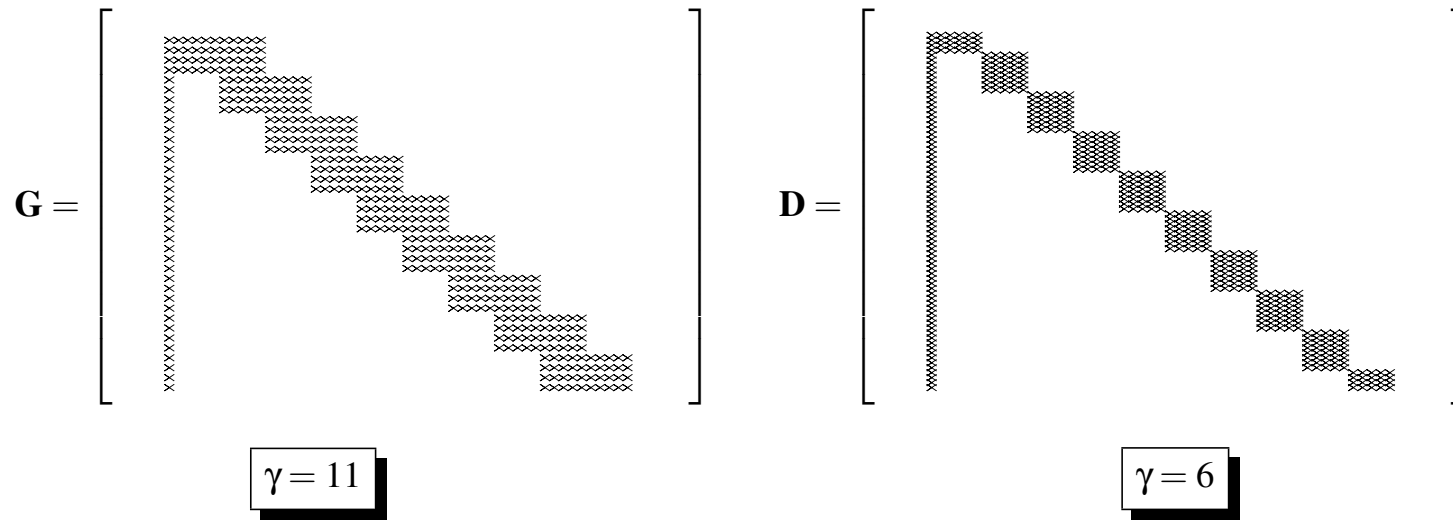
- Number of index sets does not depend on mesh size, M

$$\gamma \sim O[\kappa_0(n_y + n_u)] \ll O(n)$$

where $\kappa_0 \leq 1$ depends on right hand side sparsity

- \mathbf{G} and \mathbf{H}_L computed with $\gamma(\gamma + 3)/2$ perturbations of function generator.

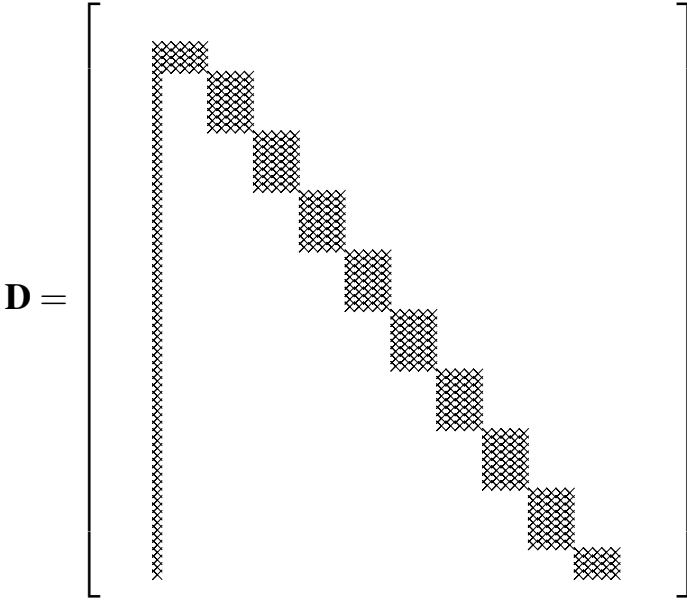
Discretization Separability



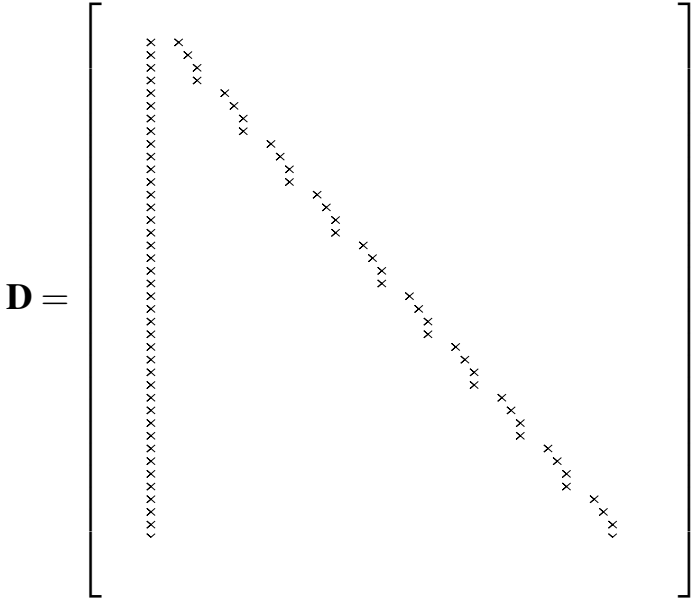
Example Problem
(4 states, 1 control, $M = 10$)

$$\begin{aligned}\dot{y}_1 &= y_3 \\ \dot{y}_2 &= y_4 \\ \dot{y}_3 &= a \cos u \\ \dot{y}_4 &= a \sin u\end{aligned}$$

Right Hand Side Sparsity



$\gamma = 6$



$\gamma = 2$

Quasi-Newton or Newton?

- Cost to Solve NLP, i.e. Number of Iterations
 - Quasi-Newton Hessian:
 - * Quadratic Function: n steps
 - * General Function: $O(\kappa_2 n)$, where κ_2 is a factor related to region of convergence
 - True Hessian:
 - * Quadratic Function: One step
 - * General Function: $O(\kappa_3)$, where κ_3 is a factor related to region of convergence
- There is No Quasi-Newton Update that is
 - Sparse,
 - Symmetric, and
 - Positive Definite
- Projected Hessian $\mathbf{Z}^T \mathbf{H} \mathbf{Z}$ is Dense Even When \mathbf{G} and \mathbf{H} are sparse.

Quasi-Newton Methods are not attractive for Large Sparse NLPs

Sequential Nonlinear Programming

Is an SQP better than a Barrier Method
for
Sequential Nonlinear Programming?

Performance Issues

1. Solving a Single NLP
 - (a) Computational Linear Algebra Cost
 - (b) Treatment of Inequality Constraints
2. Solving Multiple NLP's

Computational Linear Algebra Costs

- NLP Algorithms require repeated solution of

$$\mathbf{Kz} = \mathbf{b}$$

where the symmetric indefinite matrix

$$\mathbf{K} = \begin{bmatrix} \mathbf{H} & \mathbf{G}^T \\ \mathbf{G} & \mathbf{C} \end{bmatrix}$$

- Cost to Solve $\mathbf{Kz} = \mathbf{b}$ Using Direct Matrix Factorization

- Dense \mathbf{K} : $O(N^3)$ *The Elephant in the Room!*
- Sparse \mathbf{K} : $O(\kappa_1 N)$, where κ_1 is a factor related to sparsity

- For optimal control problems

- the number of degrees of freedom is large, and grows with M and
- the projected Hessian $\mathbf{Z}^T \mathbf{H} \mathbf{Z}$ is dense even when G and H are sparse!

Avoid formation of $\mathbf{Z}^T \mathbf{H} \mathbf{Z}$ and $O(N^3)$ operations

Treatment of Inequality Constraints

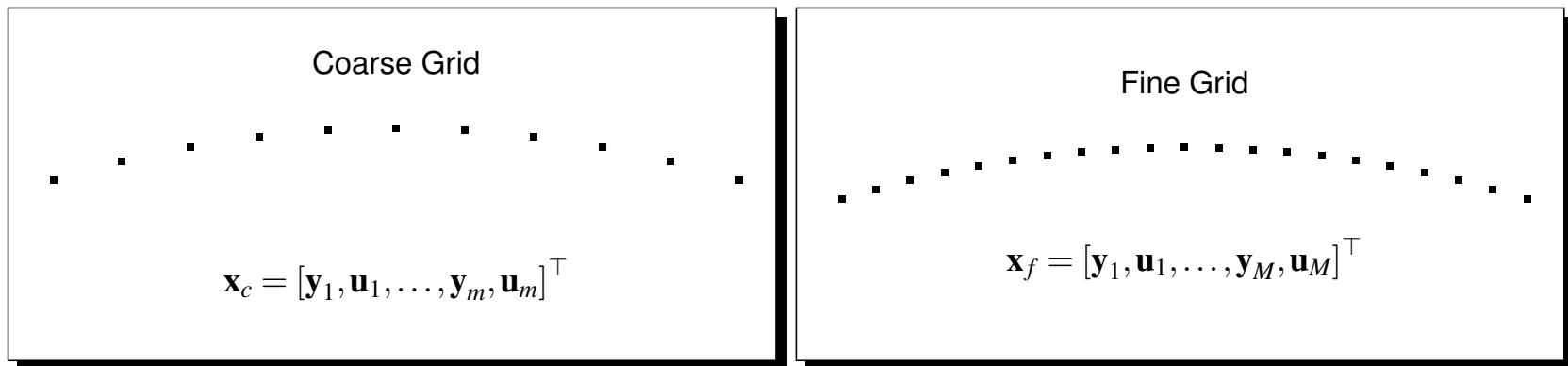
SQP Method

- Active Set Strategy
- Active Set Changes May Require Multiple Linear System Solves
 - Bad Active Set Guess → Costly
 - Good Active Set Guess → Cheap
- Direct Matrix Factorization Only

Barrier Method

- Nonlinear Barrier Transformation Replaces Active Set Strategy
- Nonlinear Transformation May Lead to More Iterations To Solve NLP
- Direct Factorization or Iterative Linear Algebra

Sequential Nonlinear Programming—Solving Multiple NLP's



NLP problem size grows—typically $M > m$

Question: How do we efficiently solve a **sequence** of NLP's?
Answer: Use coarse grid information to “Hot Start” fine grid NLP

Estimating Variables for SNLP

SQP Algorithm

high order interpolation of coarse grid solution

consistent with discretization formula

(e.g. collocation polynomial)

very good guess for active set and variables (primal and dual)

Interior Point Algorithm

must be *feasible* \iff barrier algorithm perturbs guess

not consistent with coarse grid discretization formula

Barrier Algorithm Cannot Exploit a Good Guess!

Computational Experience

Optimal Control and Inverse Problem Test Suite	259
Best Performance from SQP	227
Best Performance from Barrier	32

A Sweeping Generalization

SQP Most Efficient and Robust

Barrier Method Lacks Robustness, Speed

The Checklist

- Indirect vs Direct Formulation
- “Optimize then Discretize” vs “Discretize then Optimize”
- Boundary Value Problem: Shooting vs Collocation
- Polynomial Approximation: High vs Low Order
- Hessian Approximation: Quasi-Newton vs Newton
- Nonlinear Programming (NLP): Interior Point vs SQP

Open Questions — A New Checklist

- Alternate Discretizations Lobatto vs Radau
- Singular Arcs Indirect vs Direct Formulation
- Automatic Differentiation Sparse Differences vs ADIFOR, Complex
- Iterative Linear Algebra SQP vs Barrier

Summary

- Commercial environment requires flexible, robust software.
- Analytic derivatives (i.e. adjoint equations) are not required—sparse finite differences used.
 - Intelligent user not required!
 - Complex “black box” applications handled easily.
 - Flexible approach—new formulations readily handled!
 - Right hand side sparsity treated automatically.
- Path Inequalities do not require *a priori* estimate of constrained arc sequence
 - NLP active set procedure automatically determines arc sequence.
- Algorithmic Efficiency Requires Exploiting
 - Sparsity
 - Quadratic Convergence
 - *Sequential* Nonlinear Programming