The Direct Transcription Method For Optimal Control

Part 2: Optimal Control

John T. Betts *

*Partner, Applied Mathematical Analysis, LLC
Fundamental Principle of Transcription Methods

- *Transcription Method*
  1. Transcribe a dynamic system into a problem with a finite set of variables
  2. Solve the finite dimensional problem using a parameter optimization method (i.e. the nonlinear programming subproblem)
  3. Assess accuracy of finite dimensional problem, and if necessary repeat transcription and optimization steps

- Purpose: Identify the NLP
  - variables
  - constraints
  - objective function

for common applications.
Dynamic Systems

- Dynamics described for \( t_0 \leq t \leq t_f \) by a system of \( n \) ordinary differential equations:

\[
\dot{z} = \begin{pmatrix}
\dot{z}_1 \\
\dot{z}_2 \\
\vdots \\
\dot{z}_n
\end{pmatrix} = \begin{pmatrix}
f_1[z_1(t), \ldots, z_n(t), t] \\
f_2[z_1(t), \ldots, z_n(t), t] \\
\vdots \\
f_n[z_1(t), \ldots, z_n(t), t]
\end{pmatrix} = f(z, t)
\]

- \textit{Initial Value Problem}: Given the initial value for the dependent variables \( z(t_0) \) determine the values at some other point \( t_f \).

- \textit{Boundary Value Problem}: Determine the dependent variables such that they have specified values at two or more points, say \( t_0 \) and \( t_f \).
Example–One Variable

Dynamics: \( \frac{dz}{dt} = z \)

Analytic solution: \( z = z(t_0)e^t \)

Problem: Find \( z(0) \equiv z_0 \) such that \( z(t_f) \equiv z_f = b \).

Transcription formulation:

- One Variable; \( x \equiv z_0 \)
- One Constraint; \( c(x) = z_f = z_0e^{tf} = xe^{tf} = b \)
Shooting Method

- Iterative method
  1. Guess initial conditions \( \mathbf{x} = \mathbf{z}(t_0) \);
  2. Propagate differential equations from \( t_0 \) to \( t_f \), i.e. “shoot”
  3. Evaluate error in boundary conditions \( \mathbf{c}(\mathbf{x}) = \mathbf{z}(t_f) - \mathbf{b} \).
  4. Use NLP to adjust variables \( \mathbf{x} \) to satisfy constraints \( \mathbf{c}(\mathbf{x}) = \mathbf{0} \), i.e. repeat steps 1-3.

- Advantage — small number of variables

- Disadvantage — small change in initial condition can produce very large change in final conditions (“the tail wagging the dog”).
Example–One Variable

- Reduce sensitivity by breaking problem into two shorter steps, i.e. don’t shoot as far
  1. First step: \( t_0 \leq t \leq t_1 \)
  2. Second step: \( t_1 \leq t \leq t_f \)

  where \( t_1 = \frac{1}{2}(t_f + t_0) \)

- Guess value of \( z \) at midpoint to start second step

- Add a constraint to force continuity between steps, i.e.

  \[
  z(t_1^-) \equiv z_1^- = z_1^+ \equiv z(t_1^+)
  \]

- Transcription formulation:
  - Two Variables; \( x \equiv [z_0, z_1^+] \)
  - Two Constraints:

    \[
    \begin{bmatrix}
      c_1(x) \\
      c_2(x)
    \end{bmatrix} = \begin{bmatrix}
      z_1^- - z_1^+ \\
      z_f - b
    \end{bmatrix} = \begin{bmatrix}
      x_1e^{(t_1-t_0)} - x_2 \\
      x_2e^{(t_f-t_1)} - b
    \end{bmatrix}.
    \]
Example—(Cont.)
Multiple Shooting Method

• Break interval into $N$ pieces (segments)

$$t_0 \leq t_1 \leq \ldots \leq t_N = t_f$$

and denote $z(t_k) \equiv z_k$

• Iterative method

1. Guess initial values for $z$ at all segments; $x = [z_0, z_1, \ldots, z_N]$;

2. Propagate differential equations from $t_k$ to $t_{k+1}$, for all $N$ segments

3. Evaluate error across all segment boundaries and at final point

$$c(x) = [\zeta_1, \zeta_2, \ldots, \zeta_{N-1}, \Psi_f]^\top$$

where

$$\zeta_k = z_k^- - z_k^+$$

and $\Psi_f \equiv z_f - b$.

4. Use NLP to adjust variables $x$ to satisfy constraints $c(x) = 0$, i.e. repeat steps 1-3.

• Advantage — nonlinearity is not propagated, sensitivities localized

• Disadvantage — large number of variables and constraints
Propagation–One Step Methods

- Shooting and multiple shooting require *propagation*—given the value $z(t)$, find the value $z(t + h)$.

- Define $\bar{t} = t + h$ where *integration stepsize* $h$; Notation:

  \[
  z \equiv z(t) \quad \bar{z} \equiv z(t + h) = z(\bar{t})
  \]

  \[
  f \equiv f[z(t), t] \quad \bar{f} \equiv f[z(t + h), t + h] = f[z, \bar{t}]
  \]

- Propagation over single step:

  \[
  \bar{z} = z + \int_t^{\bar{t}} \dot{z} dt = z + \int_t^{\bar{t}} f(z, t) dt
  \]

- *One step methods* approximate integral using quadrature formula which require evaluation of integrand inside the interval.
k-stage Runge-Kutta Scheme

- Subdivide the integration step into $k$ subintervals

  \[ t_j = t + h \rho_j \]

  with

  \[ 0 \leq \rho_1 \leq \rho_2 \leq \ldots \leq \rho_k \leq 1 \]

  for $1 \leq j \leq k$.

- Quadrature formula for $\int_t^\bar{t} f \, dt$ gives:

  \[ \bar{z} = z + h \sum_{j=1}^{k} \beta_j f_j \]

  where $f_j \equiv f(t_j, z_j)$ and a quadrature formula for $\int_t^{t_j} f \, dt$ defines the intermediate points

  \[ z_j = z + h \sum_{\ell=1}^{k} \alpha_{j\ell} f_\ell \]

  for $1 \leq j \leq k$. 
k-stage Runge-Kutta Scheme (cont)

- Family of integration schemes defined by coefficients, i.e. Butcher Array

\[
\begin{array}{c|ccc}
\rho_1 & \alpha_{11} & \ldots & \alpha_{1k} \\
\vdots & \vdots & \ddots & \vdots \\
\rho_k & \alpha_{k1} & \ldots & \alpha_{kk} \\
\hline
\beta_1 & \ldots & \beta_k
\end{array}
\]

- **Explicit Methods:**
  - \( \rho_1 = 0 \) and \( \alpha_{jl} = 0 \) for \( j \leq l \) (\( \alpha \)-block is lower triangular)
  - \( \bar{z} \) can be computed explicitly from values at \( t \) and intermediate points.

- **Implicit Methods:**
  - \( \bar{z} \) cannot be computed explicitly from values at \( t \) and intermediate points.
  - \( \bar{z} \) and \( z \) appear implicitly in nonlinear quadrature formula
  - Computation of \( \bar{z} \) requires
    * an initial guess (the *predictor*)
    * an iterative solution (the *corrector*)
    * to drive the discretization “defect” to zero.

- **Boundary Value problem is inherently implicit**
Euler's Method

- Explicit, $k = 1$

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

- Common representation:

\[
\bar{z} = z + hf
\]
Classical Runge-Kutta Method

- Explicit, \( k = 4 \)

\[
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 0 \\
1/2 & 1/2 & 0 & 0 & 0 \\
1/2 & 0 & 1/2 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 \\
\hline
1/6 & 1/3 & 1/3 & 1/6
\end{array}
\]

- Common representation:

\[
\begin{align*}
k_1 &= hf \\
k_2 &= hf \left( z + \frac{1}{2} k_1, t + \frac{h}{2} \right) \\
k_3 &= hf \left( z + \frac{1}{2} k_2, t + \frac{h}{2} \right) \\
k_4 &= hf(z + k_3, \bar{t}) \\
z &= z + \frac{h}{6} (k_1 + 2k_2 + 2k_3 + k_4)
\end{align*}
\]
Trapezoidal Method

- Implicit, $k = 2$

\[
\begin{array}{c|ccc}
0 & 0 & 0 \\
1 & 1/2 & 1/2 \\
1/2 & 1/2 & 1/2 \\
\end{array}
\]

- Common representation:

\[
\bar{z} = z + \frac{h}{2} (f + \bar{f})
\]
Hermite-Simpson Method

- Implicit, $k = 3$

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
1/2 & 5/24 & 1/3 & -1/24 \\
1 & 1/6 & 2/3 & 1/6 \\
\hline
& 1/6 & 2/3 & 1/6 \\
\end{array}
\]

- Common representation:

\[
\begin{align*}
\tilde{z} &= \frac{1}{2}(z + \bar{z}) + \frac{h}{8}(f - \bar{f}) \\
\bar{f} &= f\left(\tilde{z}, t + \frac{h}{2}\right) \\
\bar{z} &= z + \frac{h}{6}\left(f + 4\bar{f} + \bar{f}\right)
\end{align*}
\]
Boundary Value Example Problem

• Golf Ball Dynamics*: 

\[ \ddot{z} = -gk + gn_3n - \mu_3gn_3 \frac{\dot{z}}{||z||} \]

where geometry of the green is 

\[ z_3 = S(z_1, z_2) = \frac{(z_1 - 10)^2}{125} + \frac{(z_2 - 5)^2}{125} - 1 \]

with 

\[ n = \frac{N}{||N||} \quad N = \left( -\frac{\partial S}{\partial z_1}, -\frac{\partial S}{\partial z_2}, 1 \right) \]

• The Putting Problem:
  – Choose (4) Variables \( \dot{z}(0) \) and \( t_f \)
  – Subject to (4) Constraints

\[ z(t_f) = z_H \]
\[ ||\dot{z}|| = 0 \]

given \( z(0) = (0, 0, 0) \) and \( z_H = (20, 0, 0) \).

Boundary Value Example Problem (cont)

- *Ill-posed Problem!* At desired boundary $\|\dot{z}\| = 0$ the friction force term $\mu n_3 \dot{z} / \|\dot{z}\|$ is not defined!

- Author suggests “realistic” problem if $\|\dot{z}\| \leq s_T$
  - What is $s_T$?
  - Solution is not unique! How does iteration terminate?

- **The Real Putting Problem:** Two regions
  - On the green: $\|z(t) - z_H\| \geq R_H$ (dynamics as above)
  - In the hole: $\|z(t) - z_H\| \leq R_H$ where dynamics have
    \[
    gn_3 n - \mu_k gn_3 \frac{\dot{z}}{\|\dot{z}\|} = 0
    \]
  - Choose (5) Variables $\dot{z}(0)$, $t_f$ and $t_1$
  - Subject to (2) Constraints
    \[
    \|z(t_1) - z_H\| = R_H
    \]
    \[
    z(t_f) - z_H \leq R_H
    \]
  - Minimize final horizontal velocity $\ddot{z}_1^2 + \ddot{z}_2^2$
Putting Problem
Dynamic Modeling Concepts

- A trajectory is made up of a collection of *phases*

- The set of differential equations cannot change during a phase; conversely different sets of differential equations can be used in different phases.

- Phases are *linked* together by *linkage conditions* to construct complete problem description

- Phases terminate at *events*.

- Multiple shooting segments may (or may not) be treated as phases.

- Spectrum of possibilities:
  - (One multiple shooting segment) and (Multiple number of integration steps) \(\Rightarrow\) Shooting method
  - (Limited number of multiple shooting segments, e.g. 5) and (Multiple number of integration steps per segment)
  - (Number of multiple shooting segments) = (Number of integration steps)
Function Generator

Phases

Multiple Shooting Segments

Integration Steps

x → F(x), c(x)
NLP Considerations

- **Goal:** Construct a function generator, i.e. select a transcription method, which is **noise free**.

- Consistency vs. Accuracy
  - A *consistent* function generator executes the same sequence of arithmetic operations for all values of \( x \).
  - An *accurate* function generator computes accurate approximations to the dynamics \( \dot{z} = f(z,t) \).

- Adaptive quadrature (e.g. variable order and/or stepsize) is “noise” to the NLP—it is not consistent!

- A fixed number of steps with an explicit integration method is consistent, but not accurate.

- **Achieving the Goal:**
  - Select transcription with one integration step per multiple shooting segment \( \implies \text{consistent} \)
  - Treat accuracy **outside** the NLP

Discretize Then Optimize
Optimal Control Problem

• Find the \( n_u \)-dimensional control vector \( u(t) \) to minimize the performance index

\[
\phi[y(t_f), t_f]
\]

evaluated at the final time \( t_f \) and satisfy the state equations

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

where the \( n_y \) dimension state vector \( y \) may have some specified initial and terminal values.

• Autonomous Form when \( f[y(t), u(t), t] = f[y(t), u(t)] \)

• Transcribe to finite dimensional NLP
  - NLP optimization variables

\[
x = (u_1, y_2, u_2, \ldots, y_M, u_M)
\]

  - NLP constraints (Euler’s Method)

\[
c_k(x) = y_{k+1} - y_k - h f(y_k, u_k)
\]

for \( k = 1, \ldots, M - 1 \), with \( y_0 \) and \( u_0 \) given, and \( h = t_f / M \)

  - NLP objective function – minimize

\[
\phi(x) = \phi(y_M)
\]
Necessary Conditions for Discrete Problem

- Define the Lagrangian

\[
L(x, \lambda) = \phi(x) - \lambda^T c(x)
\]

\[
= \phi(y_M) - \sum_{k=1}^{M-1} \lambda_k^T [y_{k+1} - y_k - hf(y_k, u_k)]
\]

- Necessary Conditions

\[
\frac{\partial L}{\partial \lambda_k} = y_{k+1} - y_k - hf(y_k, u_k) = 0
\]

\[
\frac{\partial L}{\partial u_k} = h\lambda_k^T \frac{\partial f}{\partial u_k} = 0
\]

\[
\frac{\partial L}{\partial y_k} = (\lambda_k - \lambda_{k-1}) + h\lambda_k^T \frac{\partial f}{\partial y_k} = 0
\]

\[
\frac{\partial L}{\partial y_M} = -\lambda_M + \frac{\partial \phi}{\partial y_M} = 0
\]
Necessary Conditions for Limiting Form of Discrete Problem

- Let $M \to \infty$, $h \to 0$

- **State Equations**
  \[ \dot{y} = f(y, u) \]

- **Pontryagin Maximum Principle**
  \[ \frac{\partial H}{\partial u} = \lambda^\top \frac{\partial f}{\partial u} = 0 \]

- **Adjoint Equations**
  \[ \dot{\lambda} = -\lambda^\top \frac{\partial f}{\partial y} \]

- **Transversality Condition**
  \[ \lambda(t_f) - \frac{\partial \phi}{\partial y} \bigg|_{t=t_f} = 0 \]
Direct vs Indirect Methods

**Indirect**  Construct and solve adjoint equations, Maximum principle, and boundary conditions.

- Maximum Principle is
  - an algebraic side condition, (possibly implicit)
  - (state + adjoint + maximum principle) $\Rightarrow$ differential-algebraic system (DAE)
  - an NLP at each time step $\Rightarrow$ choose controls $u(t_k)$ to maximize the Hamiltonian $H = \lambda^T f$

- Solution via shooting, multiple shooting, or discretization

- **Pros**: Accurate solution for “special cases”, e.g. singular arcs
- **Cons**: Not general (requires derivation and implementation of adjoint equations); Not robust

**Direct**  Directly optimize the objective without formation of the necessary conditions (adjoints, transversality, etc.)

- Control Parameterization
- State and Control Parameterization

- **Pros**: Very robust and general
- **Cons**: Special treatment for “special cases”
### Relationship Between Continuous and Discrete Problem

<table>
<thead>
<tr>
<th>Discrete</th>
<th>Continuous</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(y_1, y_2, \ldots, y_M)$</td>
<td>NLP Variables</td>
</tr>
<tr>
<td>$(u_1, u_2, \ldots, u_M)$</td>
<td>NLP Variables</td>
</tr>
<tr>
<td>$\zeta_k = 0$</td>
<td>Defect Constraints</td>
</tr>
<tr>
<td>$(\lambda_1, \lambda_2, \ldots, \lambda_M)$</td>
<td>Lagrange Multipliers</td>
</tr>
</tbody>
</table>
General Formulation

- Formulate problem as collection of $N$ phases, where

$$ t_0^{(k)} \leq t \leq t_f^{(k)} . $$

- Dynamic variables

$$ z^{(k)} = \begin{bmatrix} y^{(k)}(t) \\ u^{(k)}(t) \end{bmatrix} $$

made of state variables $y^{(k)}(t)$ and control variables $u^{(k)}(t)$.

- Parameters $p^{(k)}$ independent of $t$.

- State equations

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

- Boundary conditions

$$ \psi_{0l} \leq \psi[y(t_0), u(t_0), p, t_0] \leq \psi_{0u}, $$

$$ \psi_{fl} \leq \psi[y(t_f), u(t_f), p, t_f] \leq \psi_{fu}, $$
General Formulation (Cont.)

- **Algebraic path constraints**
  \[ g_{\ell} \leq g[y(t), u(t), p, t] \leq g_{u}, \]

- **Variable bounds**
  \[ y_{\ell} \leq y(t) \leq y_{u}, \]
  \[ u_{\ell} \leq u(t) \leq u_{u}, \]
  \[ p_{\ell} \leq p \leq p_{u}. \]

- **Optimal Control Problem**: Find dynamic variables \( z \) and parameters \( p \) subject to constraints and bounds, which minimize
  \[ J = \phi \left[ y(t_0^{(1)}, t_0^{(1)}, y(t_f^{(1)}), p^{(1)}, t_f^{(1)}), \ldots, y(t_0^{(N)}, t_0^{(N)}, y(t_f^{(N)}), p^{(N)}, t_f^{(N)} \right]. \]
Direct Transcription Formulation

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

\[
\begin{align*}
y_k &\equiv y(t_k) & \bar{u}_{k+1} &\equiv u[(t_{k+1} + t_k)/2] \\
u_k &\equiv u(t_k) & f_k &\equiv f[y(t_k), u(t_k), p, t_k]
\end{align*}
\]

• Trapezoidal
  – NLP variables
    \[x = [y_1, u_1, y_2, u_2, \ldots, y_M, u_M, p, t_1, t_M]^\top\]
  – Defect constraints for $k = 1, \ldots, M - 1$
    \[
    \zeta_k = y_{k+1} - y_k - \frac{h_k}{2} [f_{k+1} + f_k] = 0
    \]

• Hermite-Simpson (Compressed)
  – NLP variables
    \[x = [y_1, u_1, \bar{u}_2, y_2, u_2, \bar{u}_3, \ldots, y_M, u_M, p, t_1, t_M]^\top\]
  – Defect constraints for $k = 1, \ldots, M - 1$
    \[
    \zeta_k = y_{k+1} - y_k - \frac{h_k}{6} [f_{k+1} + 4\bar{f}_{k+1} + f_k] = 0
    \]
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 matrix of partial derivatives, i.e. the Jacobian matrix is sparse.

• The Jacobian matrix is defined as

\[
G_{ij} = \frac{\text{Change in Defect Constraint on Segment } i}{\text{Change in Optimization Variable at Grid Point } j}
\]

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 Differences* used to compute \( G \) and \( H_L \)
  
  – Number of index sets \( \gamma \approx \) number of dynamic variables \( z \); not number of grid points \( M \)
  
  – \( G \) and \( H_L \) computed with \( \gamma(\gamma + 3)/2 \) perturbations of function generator.
Standard Approach

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

\[
\begin{align*}
\dot{y}_1 &= y_3 \\
\dot{y}_2 &= y_4 \\
\dot{y}_3 &= a \cos u \\
\dot{y}_4 &= a \sin u
\end{align*}
\]

- NLP variables: $x = (t_f, y_1, u_1, \ldots, y_M, u_M)$

- Approximate ODE

with NLP constraints:

\[
0 = y_{k+1} - y_k - \frac{h_k}{2} [f_{k+1} + f_k] = c(x)
\]

for $k = 1, \ldots, M - 1$ (Trapezoidal Method).

- NLP Jacobian:

\[
G = \frac{\partial c}{\partial x}
\]

- Standard Approach: $c = q$ and $G = D$
  - Index sets: $\gamma = 2 \times (n_y + n_u) + 1 = 11$
  - Variables: $n = M(n_y + n_u) + 1 = 51$
  - Constraints: $m = (M - 1)n_y = 36$
Discretization Separability

- Group terms by grid point

\[ 0 = y_{k+1} - y_k - \frac{h_k}{2} [f_{k+1} + f_k] = \left[ y_{k+1} - \frac{h_k}{2} f_{k+1} \right] + \left[ -y_k - \frac{h_k}{2} f_k \right] \]

- NLP constraints: \( c(x) = Bq(x) \) where

\[
B = \begin{bmatrix}
I & I & I & \ldots & I & I \\
I & I & I & \ldots & I & I \\
\vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\
I & I & I & \ldots & I & I \\
\end{bmatrix}
\]

\[
q(x) = \begin{bmatrix}
-y_1 - \frac{h_1}{2} f_1 \\
y_2 - \frac{h_1}{2} f_2 \\
\vdots \\
-y_{M-1} - \frac{h_{M-1}}{2} f_{M-1} \\
y_M - \frac{h_{M-1}}{2} f_M \\
\end{bmatrix}
\]

- Construct sparse difference estimates for the matrix

\[
D \doteq \frac{\partial q}{\partial x}
\]

- NLP Jacobian:

\[
G \doteq \frac{\partial c}{\partial x} = BD
\]
Discretization Separability

\[ G = \begin{bmatrix} \gamma = 11 \end{bmatrix} \]

\[ D = \begin{bmatrix} \gamma = 6 \end{bmatrix} \]
Right Hand Side Sparsity

- Group terms by grid point and isolate linear terms

\[ 0 = y_{k+1} - y_k - \frac{h_k}{2} [f_{k+1} + f_k] \]

\[ = [y_{k+1} - y_k] - \frac{1}{2} \tau_k [t_f f_{k+1}] - \frac{1}{2} \tau_k [t_f f_k] \]

where \( h_k = \tau_k (t_f - t_0) = \tau_k t_f \) with \( 0 < \tau_k < 1 \)

- NLP constraints: \( c(x) = Ax + Bq(x) \) where

\[
A = \begin{bmatrix}
0 & -I & 0 & I \\
-1 & 0 & I & 0 \\
\vdots & & & & \ddots \\
\end{bmatrix} \quad B = -\frac{1}{2} \begin{bmatrix}
\tau_1 I & \tau_1 I \\
\tau_2 I & \tau_2 I \\
\vdots & & & & \ddots \\
\tau_{M-1} I & \tau_{M-1} I \\
\end{bmatrix}
\]

and

\[
q = \begin{bmatrix}
t_f f_1 \\
t_f f_2 \\
\vdots \\
t_f f_M
\end{bmatrix}
\]
Right Hand Side Sparsity (cont.)

- Right hand side sparsity template:

\[
\begin{bmatrix}
\frac{\partial f}{\partial y} & \frac{\partial f}{\partial u}
\end{bmatrix} =
\begin{bmatrix}
\cdot & \cdot & x & \cdot \\
\cdot & \cdot & \cdot & x \\
\cdot & \cdot & \cdot & x \\
\cdot & \cdot & \cdot & x
\end{bmatrix}
\]

- Construct right hand side template using \textit{random} perturbations about \textit{random} nominal points.

- Construct sparse difference estimates for the matrix

\[
D \triangleq \frac{\partial q}{\partial x}
\]

using sparse right hand sides

- NLP Jacobian:

\[
G = A + BD
\]
Right Hand Side Sparsity

\[ D = \begin{bmatrix} \gamma = 6 \end{bmatrix} \]

\[ D = \begin{bmatrix} \gamma = 2 \end{bmatrix} \]
Hermite-Simpson (Compressed)

- Hermite-Simpson defect constraints \((h_k = \tau_k t_f)\)

\[
0 = y_{k+1} - y_k - \frac{h_k}{6} [f_{k+1} + 4\bar{f}_{k+1} + f_k]
\]

where

\[
\bar{f}_{k+1} = f\left[\bar{y}_{k+1}, \bar{u}_{k+1}, t_k + \frac{h_k}{2}\right]
\]

\[
\bar{y}_{k+1} = \frac{1}{2}(y_{k+1} + y_k) + \frac{h_k}{8} (f_k - f_{k+1})
\]

- NLP variables: \(x = (t_f, y_1, u_1, u_2, \ldots, u_M, y_M, u_M)\)

- NLP constraints:

\[
c(x) = Ax + Bq(x)
\]

where

\[
q \doteq \begin{bmatrix}
    t_f (f_2 + 4\bar{f}_2 + f_1) \\
    t_f (f_3 + 4\bar{f}_3 + f_2) \\
    \vdots \\
    t_f (f_M + 4\bar{f}_M + f_{M-1})
\end{bmatrix}
\]
Hermite-Simpson (Compressed) Sparsity

Sparsity template:

\[
\begin{bmatrix}
\frac{\partial v}{\partial y} & \frac{\partial v}{\partial u}
\end{bmatrix}
= \begin{bmatrix}
\vdots & x & \vdots & x \\
\vdots & x & \vdots & x \\
\vdots & \vdots & \vdots & x
\end{bmatrix}
\]

where

\[
v = t_f (f_{k+1} + 4\bar{f}_{k+1} + f_k)
\]
Hermite-Simpson (Separated)

- Discretization constraints without local compression

\[
0 = \bar{y}_{k+1} - \frac{1}{2}(y_{k+1} + y_k) - \frac{\tau_{k+1}f}{8}(f_k - f_{k+1}) \quad \text{Hermite}
\]

\[
0 = y_{k+1} - y_k - \frac{\tau_{k+1}f}{6}[f_{k+1} + 4\bar{f}_{k+1} + f_k] \quad \text{Simpson}
\]

- NLP variables: \(x = (t_f, y_1, u_1, \bar{y}_2, \bar{u}_2, y_2, u_2, \ldots, \bar{y}_M, \bar{u}_M, y_M, u_M)\)

- NLP constraints: \(c(x) = Ax + Bq(x)\) where \(A\) and \(B\) are constant matrices and

\[
q = \begin{bmatrix}
t_f f_1 \\
t_f \bar{f}_1 \\
t_f f_2 \\
t_f \bar{f}_2 \\
\vdots \\
t_f f_{M-1} \\
t_f \bar{f}_{M-1} \\
t_f f_M \\
t_f \bar{f}_M \\
t_f f_M
\end{bmatrix}
\]
Hermite-Simpson (Separated) Sparsity

\[ D = \begin{bmatrix}
\gamma = 2
\end{bmatrix} \]
The General Approach

- For path constrained problems in semi-explicit form

\[
\begin{align*}
\dot{y} &= f[y, u, t] \\
g_{\ell} &\leq g[y, u, t] \leq g_u,
\end{align*}
\]

write transcribed NLP functions as

\[
\begin{bmatrix}
c(x) \\
F(x)
\end{bmatrix} = Ax + Bq(x)
\]

where \(A\) and \(B\) are constant matrices and \(q\) involves right hand sides at grid points.

- Construct sparsity template for DAE right hand side

\[
\begin{bmatrix}
\frac{\partial f}{\partial y} & \frac{\partial f}{\partial u} \\
\frac{\partial g}{\partial y} & \frac{\partial g}{\partial u}
\end{bmatrix}
\]

- Construct sparsity for \(D\) and compute index sets.

- Define NLP Jacobian sparsity from \(G = A + BD\)

- Define NLP Hessian sparsity from \((BD)^T(BD)\).
Specific Performance Highlights

- Spatial discretization of Nonlinear Parabolic PDE (Heinkenschloss)
- DAE system (50 states, 3 controls, 2 algebraic equations)
- Five mesh refinements; Final NLP—9181 variables, 9025 active constraints, and 156 degrees of freedom

<table>
<thead>
<tr>
<th></th>
<th>Dense</th>
<th>Sparse</th>
<th>Reduction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>γ_s</td>
<td>53</td>
<td>4</td>
<td>-92.5%</td>
</tr>
<tr>
<td>γ_c</td>
<td>109</td>
<td>15</td>
<td>-86.2%</td>
</tr>
<tr>
<td>Func. Eval.</td>
<td>42552</td>
<td>983</td>
<td>-97.7%</td>
</tr>
<tr>
<td>CPU Time</td>
<td>1851.43</td>
<td>377.86</td>
<td>-79.6%</td>
</tr>
</tbody>
</table>
Summary

- Write transcribed NLP functions as
  \[
  \begin{bmatrix}
  c(x) \\
  F(x)
  \end{bmatrix}
  = Ax + Bq(x)
  \]
  to exploit sparsity in \( D = \frac{\partial q}{\partial x} \).

- Sparsity considerations lead to desirable discretizations:
  - Separated form of k-stage Runge-Kutta Schemes

- Construction and analysis of RHS sparsity
  - Numerical with limited analysis of rank
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 using an SQP or Barrier Method

**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.
Mesh Refinement

- **The Problem:** Approximate continuous state and control functions using values at discrete points
  - Many discrete values give “good” approximation
  - Many discrete values make numerical problem “hard”

- **Mesh Refinement:** Choose
  - discretization technique and
  - number and location of points
to achieve
  - an “accurate” solution and
  - an “efficient” solution method.
The Challenge for Mesh Refinement

- Computation of solution accuracy depends on **order** of the discretization method

- **Order** depends on **stiffness** of ODE

- **Order** depends on **index** of DAE

- **Index** depends on which path constraints are **active**, e.g.

\[
\begin{align*}
\text{Index 1} & \\
\dot{y} &= f[y, u, t] \\
0 &= g_1[y, u, t]
\end{align*}
\]

\[
\begin{align*}
\text{Index 2} & \\
\dot{y} &= f[y, u, t] \\
0 &= g_2[y, u, t]
\end{align*}
\]

- Order variation makes acceleration techniques (e.g. extrapolation, deferred correction) cumbersome
Discretization Selection Strategy

**Trapezoidal**  Robust, $O(h^2)$, good for coarse grid

**Separated Hermite-Simpson**  $O(h^4)$, fewer index sets than HSC, but larger NLP

**Compressed Hermite-Simpson**  $O(h^4)$, more index sets than HSS, but smaller NLP
Representing the Solution

- **Goal:** Construct approximate continuous solution \( [\tilde{y}(t), \tilde{u}(t)] \) from the discretization.

- **Spline Representation:**
  \[
  \begin{bmatrix}
  y(t) \\
  u(t)
  \end{bmatrix}
  \approx
  \begin{bmatrix}
  \tilde{y}(t) \\
  \tilde{u}(t)
  \end{bmatrix}
  = \sum_{i=1}^{N} \alpha_i B_i(t)
  \]

  where the coefficients \( \alpha_i \) are defined such that

  - **state variables**
    \[
    \tilde{y}(t_k) = y_k \\
    \frac{d}{dt}\tilde{y}(t_k) = f_k
    \]

  - **control variables**
    \[
    \tilde{u}(t_k) = u_k \\
    \tilde{u}\left(\frac{(t_k + t_{k-1})}{2}\right) = \bar{u}_k \quad \text{(H-S only)}
    \]

- **Properties:**
  - **state variables** \( C^1 \) cubic functions
  - **control variables** \( C^0 \) linear or quadratic functions
Estimating Discretization Error

• Define *absolute local error*

$$\eta_{i,k} = \int_{t_k}^{t_{k+1}} |\varepsilon_i(s)| ds.$$  

where

$$\varepsilon(t) = \dot{\tilde{y}}(t) - f[\tilde{y}(t), \tilde{u}(t), t]$$

• Define *relative local error*

$$\varepsilon_k \approx \max_i \frac{\eta_{i,k}}{(w_i + 1)}$$  

where the scale weight

$$w_i = \max_{k=1}^{M} [\tilde{y}_{i,k}, \dot{\tilde{y}}_{i,k}]$$

defines the maximum value for the $i$-th state variable or its derivative over the $M$ grid points.

• Evaluate $\eta_{i,k}$ accurately using Romberg quadrature
Estimating Order Reduction

- **Local Error** $O(h^{p-r+1})$ has the form

  $\varepsilon_k \approx \|c_k h^{p-r+1}\|

  where order reduction is $r$, and $p = 2$ for trapezoidal, $p = 4$ for Hermite-Simpson.

- Subdivide Interval by adding $I$ points

  \[
  \theta = ch^{p-r+1} \quad \text{Old Grid}
  \]

  \[
  \eta = c \left(\frac{h}{1+I}\right)^{p-r+1} \quad \text{Current Grid}
  \]

- Assume constant $c$ and $r$ solve for **Predicted Reduction**

  \[
  r = \max \left[0, \min \left(\text{nint}(\hat{r}), p\right)\right]
  \]

  where

  \[
  \hat{r} = p + 1 - \frac{\log(\theta/\eta)}{\log(1+I)}.
  \]
Subdividing The Grid

Example: Add One Point to Third Interval
Constructing a New Mesh

- *Subdivide* current grid by adding $I_k$ points to interval $k$.

- Assuming constant coefficients $c_k$ across interval

$$
\varepsilon_k \approx \|c_k\| \left( \frac{h}{1 + I_k} \right)^{p-r_k+1} = \max_i \frac{\eta_{i,k}}{(w_i + 1)} \left( \frac{1}{1 + I_k} \right)^{p-r_k+1}
$$

- *Integer Program:* Choose $I_k$ to minimize

$$
\phi(I_k) = \max_k \varepsilon_k
$$

and satisfy the constraints

$$
\sum_{k=1}^{n_s} I_k \leq M - 1
$$

and

$$
0 \leq I_k \leq M_1
$$

for $k = 1, \ldots, n_s$. 
Refinement Algorithm

Given a discrete solution on the current grid (from NLP), parameters $\delta$, $\kappa \leq 1$, and $M_1$ (typically $\delta = 10^{-7}$, $\kappa = .1$, $M_1 = 5$)

1. Construct Continuous Representation. Compute the cubic spline representation from the discrete solution $x^*$

2. Estimate Discretization Error. Compute an estimate for the discretization error $\varepsilon_k$ in each segment of the current mesh

3. Compute Primary Order of New Mesh.
   - If error is equidistributed for low order method increase order; (If $p = 2$ and $\varepsilon_{\text{max}} \leq 2\bar{\varepsilon}$ then set $p = 4$ and terminate).
   - otherwise if more than two refinements with low order method increase order; (if $p = 2$ and ref. iter. $> 2$ then set $p = 4$ and terminate).

4. Estimate Order Reduction. Compare current and old grid to compute $r_k$
Refinement Algorithm (Cont.)

5. **Construct New Mesh** Solve integer program

1. Compute interval $\alpha$ with maximum error, i.e.
   \[ \varepsilon_{\alpha} = \max_k \varepsilon_k \]

2. Terminate if:
   - $M'$ points have been added ($M' \geq \min [M_1, 1M]$)
   - *Error within tolerance:* $\varepsilon_{\alpha} \leq \delta$ and $I_\alpha = 0$ or;
   - *Predicted error safely within tolerance:* $\varepsilon_{\alpha} \leq \kappa \delta$ and $0 < I_\alpha < M_1$ or;
   - $M - 1$ points have been added or;
   - $M_1$ points have been added to a single interval.

3. Add a point to interval $\alpha$, i.e. $I_\alpha \leftarrow I_\alpha + 1$;

4. Update predicted error for interval $\alpha$, and

5. Repeat steps 1-5
Transcription Accuracy

- Mesh Refinement Algorithm
  - B-spline representation for optimal solution
  - Discretization error based Romberg quadrature
- Interval subdivision by solving integer programming problem
  - Minimizes the maximum relative error
  - Tends to equidistribute error by clustering points where error is large
  - Tends to reduce overall error when error is equidistributed
- Predicted order reduction
  - Computed by comparing successive grids
  - Tends to cluster points where order is low
  - Alters where points are added, but
  - Is not used when computing discretization accuracy!
- Overall strategy uses
  - lower order trapezoidal method for early refinement iterations
  - higher order Hermite-Simpson method for final refinement iterations
  - Separated vs Compressed H-S selected dynamically
Parameter Estimation Using Direct Transcription Methods
The “Forward Problem”

- Given a set of parameters $\mathbf{p}$ find the resulting trajectory.

- “Integrate” the differential equations from $t_I$ to $t_F$

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

state variables $y$, algebraic variables $u$. 
The “Inverse Problem”

- Given the trajectory find the parameters \( p \).

- Trajectory is “given” by

\[
\hat{x}(t_1), \hat{x}(t_2), \ldots, \hat{x}(t_N)
\]
The Parameter Estimation Problem

- **State equations**
  
  \[ \dot{y} = f[y(t), u(t), p, t] \]

  state variables \( y \), algebraic variables \( u \), for \( t_I \leq t \leq t_F \).

- **Path constraints**
  
  \[ g_L \leq g[y(t), u(t), p, t] \leq g_U, \]

- **Bounds**
  
  \[ y_L \leq y(t) \leq y_U, \]
  \[ u_L \leq u(t) \leq u_U, \]
  \[ p_L \leq p \leq p_U. \]

- **Boundary conditions**
  
  \[ \psi_L \leq \psi[y(t_I), u(t_I), t_I, y(t_F), u(t_F), t_F, p] \leq \psi_U, \]

- **Objective Function** (minimize)
  
  \[ F = \frac{1}{2} r^T r = \frac{1}{2} \sum_{k=1}^{\ell} r_k^2. \]
The Least Squares Objective Function

- **Residuals**
  
  \[ r_k = w_{ij} [y_i(\theta_{ij}) - \hat{y}_{ij}] \quad \text{State} \]
  
  \[ r_k = w_{ij} [u_i(\vartheta_{ij}) - \hat{u}_{ij}] \quad \text{Algebraic} \]

  for \( t_I \leq \theta_{ij} \leq t_F \) and \( t_I \leq \vartheta_{ij} \leq t_F \).

- **Maximum Likelihood Objective**

  \[
  F = \frac{1}{2} \sum_{k=1}^{N} [g(y, u, p, \theta_{k}) - \hat{g}_k]^T \Lambda [g(y, u, p, \theta_{k}) - \hat{g}_k]
  \]

  where \( \hat{g}_k \) are the observed values of the function \( g \), and the positive definite inverse covariance matrix \( \Lambda = Q^T Q \).

- **Transformation to Standard Form**

  \[
  v(t) = Qg(y, u, p, t) \quad \text{Algebraic Constraint} \\
  \hat{v}_k = Q\hat{g}_k \quad \text{Normalized Data}
  \]

- **Transformed Objective Function** (minimize)

  \[
  F = \frac{1}{2} \sum_{k=1}^{N} [v_k - \hat{v}_k]^T [v_k - \hat{v}_k]
  \]
Computing The Residuals

Residual $r_k = \left[ y(\theta_k) - \hat{y}_k \right]$  

State Residual: $y(\theta_k) \rightarrow$ Cubic Interpolant using $y_j, y_{j+1}, f_j, f_{j+1}$

Algebraic Residual: $u(\vartheta_k) \rightarrow$
- Quadratic Interpolant using $u_j, \bar{u}_{j+1}, u_{j+1}$
- Linear Interpolant using $u_j, u_{j+1}$
Mesh Refinement and Jacobian Sparsity

Observation: Jacobian and Hessian Sparsity Pattern Change as Mesh is Refined!
“Notorious Test Problem”*

\[ F(p) = \frac{1}{2} \sum_{k=1}^{N} \left[ (y_{1k} - \hat{y}_{1k})^2 + (y_{2k} - \hat{y}_{2k})^2 \right] \]

\[
\begin{align*}
\dot{y}_1 &= y_2 \\
\dot{y}_2 &= \mu^2 y_1 - (\mu^2 + p^2) \sin(pt) \\
y_1(0) &= 0, \ y_2(0) = \pi, \ \mu = 60, \ 0 \leq t \leq 1; \ p^* = \pi
\end{align*}
\]

Observations

- Shooting method (i.e. “forward problem”) is unstable.
- Residual Jacobian sparsity changes with mesh refinement
- Mesh refinement is driven by ODE accuracy, not data

*Bulirsch
"Notorious Test Problem"

**Dashed**
- Refn ......................... 5
- Grid ......................... 92
- Data (equidist) ............. 10

**Solid**
- Refn ......................... 5
- Grid ......................... 91
- Data (random) .............. 2000
Summary

- Mesh refinement is driven by DAE accuracy, not data
- Quadratic Convergence for Nonlinear, Nonzero Residuals
- Sparsity for Least Squares Jacobian and Hessian determined by
  - Discrete data location relative to mesh points
  - Right hand side sparsity
- Jacobian and Hessian Computed Using Sparse Finite Differences