SUPPLEMENT 2 ADDS FUNDS AND EXTENDS THE PERIOD OF PERFORMANCE. ALSO, CHANGE IN CONTRACTS POC. BUDGETED INTO SUBPROJECT E-16-M90 DUE TO CAPPED O/H RATE.
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Project #</td>
<td>E-16-M37</td>
</tr>
<tr>
<td>Sponsor</td>
<td>105 001</td>
</tr>
<tr>
<td>NASA/LANGLLEY RESEARCH CTR</td>
<td></td>
</tr>
<tr>
<td>Project director</td>
<td>Y</td>
</tr>
<tr>
<td>Administrative network representative</td>
<td>Y</td>
</tr>
<tr>
<td>TRI Accounting/Grants and contracts</td>
<td>Y</td>
</tr>
<tr>
<td>Procurement/Supply services</td>
<td>Y</td>
</tr>
<tr>
<td>Research property management</td>
<td>Y</td>
</tr>
<tr>
<td>Research security services</td>
<td>N</td>
</tr>
<tr>
<td>Project File</td>
<td>N</td>
</tr>
<tr>
<td>OCA/CSD</td>
<td>N</td>
</tr>
<tr>
<td>Other</td>
<td>N</td>
</tr>
<tr>
<td>Trig Notice Date</td>
<td>950321</td>
</tr>
<tr>
<td>Archive Box #</td>
<td>950321</td>
</tr>
<tr>
<td>Notice Date</td>
<td>950321</td>
</tr>
<tr>
<td>Closeout Complete Date</td>
<td>950721</td>
</tr>
<tr>
<td>Burn Date</td>
<td>950721</td>
</tr>
</tbody>
</table>

F7 Previous F12 Return
Development of an hp-Version
Finite Element Method for
Computational Optimal Control

Semi-Annual Progress Report
NASA Grant NAG-1-1435
Feb. 22, 1993 – Aug. 21 1993

Prof. Dewey H. Hodges, Principal Investigator
Michael S. Warner, Graduate Research Assistant
School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332-0150

Research Supported by:
Spacecraft Control Branch
Technical Monitor: Dr. Daniel D. Moerder
Mail Stop 161
NASA Langley Research Center
Hampton, Virginia 23681
Introduction

The purpose of this research effort is to develop a means to use, and to ultimately implement, \( hp \)-version finite elements in the numerical solution of optimal control problems. Under NAG-939, the hybrid MACSYMA/FORTRAN code GENCODE was developed which utilized \( h \)-version finite elements to successfully approximate solutions to a wide class of optimal control problems. In that code the means for improvement of the solution was the refinement of the time-discretization mesh. With the extension to \( hp \)-version finite elements, the degrees of freedom include both nodal values and extra interior values associated with the unknown states, co-states, and controls, the number of which depends on the order of the shape functions in each element. For details, see [1].

Progress During This Period

Optimal Control Problems

A FORTRAN code has been developed using higher-order finite-elements to approximate solutions to a particular subset of optimal control problems. The cost function to be minimized for these problems can contain both an scalar penalty, \( \phi \), on the states, \( x \), at the initial time or final time, \( t_f \), plus an integral penalty, \( L \), on the states and controls, \( u \), in this form:

\[
J = \phi[x(t_0), x(t_f), t_f] + \int_{t_0}^{t_f} L(x, u) \, dt
\]  

The state rates are governed by differential equations

\[
\dot{x} = f(x, u) \quad x \in \mathbb{R}^{n_x}, u \in \mathbb{R}^{n_u}
\]

where \( f \) is an autonomous function of the state vector \( x \). The boundary conditions can be specified at the initial time, the final time, or some combination of both, in the form

\[
\Psi[x(0), x(t_f), t_f] = 0 \quad \Psi \in \mathbb{R}^{n_{bc}}
\]

This formulation also allows for control inequality constraints of the form:

\[
g(x, u) \leq 0 \quad g \in \mathbb{R}^{n_p} \quad n_p < n_u
\]

which are enforced through use of slack variables, \( k \).

Adjoining the differential equations, boundary conditions, and control constraints to the original cost function by means of Lagrange multipliers \( \lambda, \nu, \) and \( \mu \) respectively yields a new cost function \( J' \):

\[
J' = \phi[x(t_0), x(t_f), t_f] + \nu^T \Psi[x(0), x(t_f), t_f] + \alpha^T (\dot{x} - x)|_{t_0}^{t_f} + \int_{t_0}^{t_f} \{L(x, u, t) + \lambda^T [f(x, u) - \dot{x}] + \mu^T [g(x, u) + k] \} \, dt
\]
where \( \dot{x} \) are the values of the states at the final time, and \( \alpha \) is a set of Lagrange multipliers used to ensure that the states are continuous at the initial and final times.

Next, to simplify notation, we define a Hamiltonian as

\[
H = L + \lambda^T f + \mu^T g
\]

and define \( x_f \equiv x(t_f) \) and \( x_0 \equiv x(t_0) \). To satisfy the first-order necessary conditions for a local minimum, we follow the development in [1] and take the first variation of \( J \), allowing variations in the states, state rates, controls, Lagrange multipliers, slack variables, and final time, yielding

\[
\delta J' = \left[ \frac{\partial \phi}{\partial t} + \nu^T \frac{\partial \Psi}{\partial t} + L + \lambda^T (f - \dot{x}) + \mu^T (g + k^2) \right] dt_f + \delta \nu^T \Psi \\
+ \left( \frac{\partial \phi}{\partial x_f} + \nu^T \frac{\partial \Psi}{\partial x_f} \right) dx(t_f) + \left( \frac{\partial \phi}{\partial x_0} + \nu^T \frac{\partial \Psi}{\partial x_0} \right) dx(t_0) + \delta \alpha(\dot{x} - x)
\]

\[
+ \alpha(d\dot{x} - dx)\bigg|_{t_0}^{t_f} + \int_{t_0}^{t_f} \left[ \frac{\partial H}{\partial x} \delta x + \frac{\partial H}{\partial u} \delta u - \lambda^T \delta \dot{x} + \delta \lambda^T (f - \dot{x}) \right] dt
\]

\[
+ \delta \mu^T (g + k^2) + 2\delta \mu^T k \delta k \bigg] dt
\]

Given that the variations are continuous, \( d\dot{x} = dx \). We then choose \( \delta \alpha = d\lambda \), integrate \( \lambda^T \delta \dot{x} \) by parts and expand the total differentials at the end points.

\[
dx(t_0) = \delta x(t_0) \\
dx(t_f) = \delta x(t_f) + \dot{x}(t_f)dt_f \\
d\lambda(t_f) = \delta \lambda(t_f) + \dot{\lambda}(t_f)dt_f
\]

Eq. (7) then becomes

\[
\delta J' = \left( \frac{\partial \phi}{\partial x} + \nu^T \frac{\partial \Psi}{\partial x} \right)^T dx(t_f) + \left( \frac{\partial \phi}{\partial x_0} + \nu^T \frac{\partial \Psi}{\partial x_0} \right) dx(t_0)
\]

\[
+ dt_f \left[ \dot{x}^T \left( \frac{\partial \phi}{\partial x_f} + \nu^T \frac{\partial \Psi}{\partial x_f} \right) + \dot{x}^T (x - \dot{x}) - \lambda^T \dot{x} + \mu^T k^2 \right]_{t_f}
\]

\[
+ dt_f \left( \frac{\partial \phi}{\partial t} + \nu^T \frac{\partial \Psi}{\partial t} + H \right)_{t_f} + \delta \nu^T \Psi + \delta \lambda^T (x - \dot{x})|_{t_0}^{t_f} - \lambda^T \delta x|_{t_0}^{t_f}
\]

\[
+ \int_{t_0}^{t_f} \left[ \frac{\partial H}{\partial x} \delta x + \frac{\partial H}{\partial u} \delta u + \dot{\lambda}^T \delta x + \delta \lambda^T (f - \dot{x}) \right] dt + \delta \mu^T (g + k^2) + 2\delta \mu^T k \delta k
\]
Defining subscripts on $H$ to denote partial derivatives and rearranging terms gives:

$$
\delta J' = dt_f \left( \frac{\partial \phi}{\partial t} + \nu^T \frac{\partial \Psi}{\partial t} + H \right)_{t_f} + dt_f \dot{x}^T \left( \frac{\partial \phi}{\partial x_f} + \nu^T \frac{\partial \Psi}{\partial x_f} - \lambda \right)_{t_f} + \left( \frac{\partial \phi}{\partial x_f} + \nu^T \frac{\partial \Psi}{\partial x_f} - \lambda^T \right)_{t_f} \delta x(t_f) + \left( \frac{\partial \phi}{\partial x_0} + \nu^T \frac{\partial \Psi}{\partial x_0} + \lambda^T \right) \delta x(t_0)
$$

$$
+ dt_f \lambda^T(x - \hat{x}) + dt_f \mu^T k^2|_{t_f} + \delta \nu^T \Psi + \delta \lambda^T(x - \hat{x})|_{t_f} + \int_{t_0}^{t_f} [H_u \delta u]
$$

$$
+ H_x \delta x + \dot{\lambda}^T \delta x + \delta \lambda^T(f - \hat{x}) + \delta \mu^T(g + k^2) + 2 \mu^T k \delta k \right] dt
$$

Now, defining $\dot{\lambda}$ as

$$
\dot{\lambda}^T \equiv \begin{cases} 
- \frac{\partial \phi}{\partial x_0} - \nu^T \frac{\partial \Psi}{\partial x_0} & t = t_0 \\
\frac{\partial \phi}{\partial x_f} + \nu^T \frac{\partial \Psi}{\partial x_f} & t = t_f
\end{cases}
$$

and integrating the terms $\dot{\lambda}^T \delta x$ and $\delta \lambda^T \dot{x}$ by parts, Eq. (10) becomes:

$$
\delta J' = dt_f \left( \frac{\partial \phi}{\partial t} + \nu^T \frac{\partial \Psi}{\partial t} + H \right)_{t_f} + \lambda^T \delta x|_{t_0}^{t_f} - \delta \lambda^T x|_{t_0}^{t_f}
$$

$$
+ \left[ (\dot{\lambda}^T - \lambda^T \right] \delta x|_{t_f}^{t_0} + \delta \nu^T \Psi + dt_f \left[ \dot{x}^T (\dot{\lambda} - \lambda) \right]_{t_f}
$$

$$
- \left[ (\dot{x} - x)^T \delta \lambda \right]_{t_f}^{t_0} + dt_f \mu^T k^2|_{t_f} - dt_f \left[ \lambda^T (\dot{x} - x) \right]_{t_f}
$$

$$
+ \int_{t_0}^{t_f} \left[ H_u \delta u + H_x \delta x - \lambda^T \delta \dot{x} + \delta \lambda^T f + \delta \dot{\lambda}^T x
$$

$$
+ \delta \mu^T(g + k^2) + 2 \mu^T k \delta k \right] dt
$$

We will now enforce that $\dot{x} = x$ and $\dot{\lambda} = \lambda$ at $t_0$ and $t_f$ and define

$$
\dot{H} \equiv \frac{\partial \phi}{\partial t} + \nu^T \frac{\partial \Psi}{\partial t}
$$

then Eq. (12) becomes

$$
\delta J' = dt_f \left( \dot{H} + H \right)_{t_f} + dt_f \mu^T k^2|_{t_f} + \delta \nu^T \Psi + \lambda^T \delta x|_{t_0}^{t_f} - \dot{x}^T \delta \lambda|_{t_0}^{t_f}
$$

$$
+ \int_{t_0}^{t_f} \left[ H_u \delta u + H_x \delta x - \delta \dot{x}^T \lambda + \delta \lambda^T f + \delta \dot{\lambda}^T x
$$

$$
+ \delta \mu^T(g + k^2) + 2 \mu^T k \delta k \right] dt
$$
The time interval is broken up into \( N \) not necessarily equal length time elements, \( \Delta t_i \), such that the time at each element boundary \( \hat{t}_i \) is calculated as:

\[
\hat{t}_1 = t_0 \\
\hat{t}_i = \hat{t}_{i-1} + \Delta t_i \quad i = 2, \ldots, N + 1
\]

(15)

and we define the states and controls at these nodes to be:

\[
\hat{x}_i = x(\hat{t}_i) \quad i = 1, \ldots, N + 1 \\
\hat{u}_i = u(\hat{t}_i) \quad i = 1, \ldots, N + 1
\]

(16)  
(17)

The time within the \( i^{th} \) element, \( t_i \) is expressed as:

\[
t_i = \hat{t}_{i-1} + \tau \Delta t_i \quad 0 \leq \tau \leq 1
\]

(18)

so that \( dt = \Delta t_i d\tau \). Substituting these relationships into Eq. (14) gives:

\[
\delta J' = \left[ dt_f \left( \hat{H} + H \right)_{t_f} + dt_f \mu^T k^2 |_{t_f} + \delta u^T \Psi + \lambda^T \delta x |_{t_0} - \lambda^T \delta x |_{t_f} \right] + \sum_{i=1}^{N} \Delta t_i \int_{0}^{1} \left[ H_{ui} \delta u_i + H_{xi} \delta x_i + \frac{\delta \lambda^T_i}{\Delta t_i} x_i - \frac{\delta x^T_i}{\Delta t_i} \lambda_i + \delta \mu_i^T f_i + \delta \mu_i^T (g_i + k_2^2) + 2 \mu_i^T k_1 \delta k_i \right] d\tau
\]

(19)

where subscript \( i \) refers either to quantities within the \( i^{th} \) element or to functions evaluated using quantities within the \( i^{th} \) element.

**Higher-Order Shape Functions**

From [2], we define \( C^0 \) shape functions for the variations of state equations' Lagrange multipliers (or costates) and states in each element in terms of nodal values (superscript \(^n\)) and internal values (superscript \(^i\)).

\[
\delta \lambda_i = \delta \lambda_i (1 - \tau) + \delta \lambda_{i+1} \tau + \sum_{j=1}^{n_b-1} (1 - \tau) \tau \beta_j (\tau) \delta \lambda_{ij}
\]

(20)

\[
\delta x_i = \delta \dot{x}_i (1 - \tau) + \delta \dot{x}_{i+1} \tau + \sum_{j=1}^{n_b-1} (1 - \tau) \tau \beta_j (\tau) \delta \dot{x}_{ij}
\]

Here \( n_b \) is the order of the shape function polynomial being used, with \( n_b = 1 \) representing \( h \)-version shape functions, and the summation would be ignored. The functions \( \beta_j (\tau) \) are polynomials of order \((j - 1)\) as defined in [2]. That functional form is necessitated by the
time derivatives in Eq. (19) and required end conditions. The time derivative expression looks like
\[
\frac{d\delta x_i}{d\tau} \equiv \delta x_i' = \Delta t_i \delta \dot{x}_i = -\delta \dot{x}_i + \delta \dot{x}_{i+1} + \sum_{j=1}^{n_b-1} \gamma_j(\tau) \delta \ddot{x}_{ij}
\]
(21)
where
\[
\gamma_j(\tau) \equiv [(1 - \tau)\tau \beta_j(\tau)]'
\]
Similar expressions hold for the costates.

No time derivatives of the states and costates themselves are needed in Eq. (19), so simpler shape functions are used
\[
x_i = \begin{cases} 
\dot{x}_i & \tau = 0 \\
\sum_{j=1}^{n_b} \alpha_j(\tau) & 0 < \tau < 1 \\
\dot{x}_{i+1} & \tau = 1
\end{cases}
\]
(22)
\[
\dot{\lambda}_i = \begin{cases} 
\sum_{j=1}^{n_b} & 0 < \tau < 1 \\
\dot{\lambda}_{i+1} & \tau = 1
\end{cases}
\]
where the functions \(\alpha_j(\tau)\) are again polynomials of order \(j - 1\) as defined in [2].

For the controls, control constraint Lagrange multipliers, slack variables, and their variations, again no time derivatives exist in Eq. (19), so the same easier shape functions are used:
\[
\delta u_i = \sum_{j=1}^{n_b} \alpha_j(\tau) \delta \bar{u}_{ij}
\]
(23)
\[
\delta \mu_i = \sum_{j=1}^{n_b} \alpha_j(\tau) \delta \bar{\mu}_{ij} \quad \delta k_i = \sum_{j=1}^{n_b} \alpha_j(\tau) \delta \bar{k}_{ij}
\]
(24)
\[
u_i = \sum_{j=1}^{n_b} \alpha_j(\tau) \bar{u}_{ij} \quad 0 < \tau < 1
\]
(25)
\[
\mu_i = \sum_{j=1}^{n_b} \alpha_j(\tau) \bar{\mu}_{ij} \quad 0 < \tau < 1 \quad k_i = \sum_{j=1}^{n_b} \alpha_j(\tau) \bar{k}_{ij} \quad 0 < \tau < 1
\]
(26)
Substituting these relationships into Eq. (19) gives

\[ \delta J' = \int_t^T \left[ f_H + f_H(x_i) + \sum_i \int_0^1 H_{ui} \left( \sum_{j=1}^{n_i} \alpha_j(\tau) \delta \mu_{ij} \right) \right] dt \]

\[ + \sum_i \int_0^1 \left[ \delta \tilde{x}_i(1-\tau) + \delta \tilde{x}_{i+1}(1-\tau) + \sum_{j=1}^{n_i} \delta \lambda_{ij} \right] \int_0^1 \left[ \sum_{j=1}^{n_i} \alpha_j(\tau) \delta \tilde{\lambda}_{ij} \right] dt \]

\[ + \delta \bar{v}_t + \delta v^T + \sum_i \left( \sum_{j=1}^{n_i} \alpha_j(\tau) \delta \tilde{\lambda}_{ij} \right) \int_0^1 \left[ \sum_{j=1}^{n_i} \alpha_j(\tau) \delta \tilde{\lambda}_{ij} \right] dt \]

\[ + \int_t^T \left[ \delta \tilde{x}_i(1-\tau) + \delta \tilde{x}_{i+1}(1-\tau) + \sum_{j=1}^{n_i} \delta \lambda_{ij} \right] \int_0^1 \left[ \sum_{j=1}^{n_i} \alpha_j(\tau) \delta \tilde{\lambda}_{ij} \right] dt \]

\[ + \left( \sum_{j=1}^{n_i} \alpha_j(\tau) \delta \tilde{\lambda}_{ij} \right) \int_0^1 \left[ \sum_{j=1}^{n_i} \alpha_j(\tau) \delta \tilde{\lambda}_{ij} \right] dt \]

\[ + \sum_i \int_0^1 \left[ \delta \tilde{x}_i(1-\tau) + \delta \tilde{x}_{i+1}(1-\tau) + \sum_{j=1}^{n_i} \delta \lambda_{ij} \right] \int_0^1 \left[ \sum_{j=1}^{n_i} \alpha_j(\tau) \delta \tilde{\lambda}_{ij} \right] dt \]

where

\[ \epsilon_j(\tau) \equiv (1-\tau)\beta_j(\tau) \]

Due to the orthogonality of the chosen polynomials \( \alpha(\tau) \) and \( \gamma(\tau) \), this equation reduces to
\[ \delta J' = dt_f \mu_{N+1}^T k_{N+1}^2 + \lambda_{N+1}^T \delta x_{N+1} - \lambda_1^T \delta x_1 - \dot{x}_{N+1}^T \delta \lambda_{N+1} + \dot{x}_1^T \delta \lambda_1 \\
+ \delta v^T \Psi + dt_f (\dot{H} + H)_{t_f} + \sum_i^N \Delta t_i \int_0^1 \left\{ H_{w_i} \left( \sum_{j=1}^{n_b} \alpha_j(\tau) \delta \tilde{u}_{ij} \right) \right\} \\
+ \delta x_i^T (1 - \tau) H_{x_i} + \delta x_{i+1}^T (1 - \tau) H_{x_i} + \delta \lambda_i^T (1 - \tau) f_i + \delta \lambda_{i+1}^T f_i \\
+ \sum_{j=1}^{n_b-1} \left[ \epsilon_j(\tau) \delta x_{ij}^T H_{x_i} + \epsilon_j(\tau) \delta \lambda_{ij}^T f_i \right] \\
+ \left( \sum_{j=1}^{n_b} \alpha_j(\tau) \delta \bar{u}_{ij} \right)^T \left[ g_i + \sum_{j=1}^{n_b} \alpha_j(\tau) \bar{k}_{ij}^2 \right] \\
+ 2 \left( \sum_{j=1}^{n_b} \alpha_j(\tau) \bar{u}_{ij} \right)^T \left( \sum_{j=1}^{n_b} \alpha_j(\tau) \bar{k}_{ij} \right) \sum_{j=1}^{n_b} \alpha_j(\tau) \delta \bar{k}_{ij} \right] \right\} d\tau \\
+ \sum_i^N \left( -\delta \lambda_i^T \ddot{x}_{i,1} + \delta \lambda_{i+1}^T \ddot{x}_{i,1} - \sum_{j=2}^{n_b} \delta \lambda_{ij}^T \ddot{x}_{ij} \right) \\
+ \sum_i^N \left( \delta \dot{x}_i^T \tilde{\lambda}_{i,1} - \delta \dot{x}_{i+1}^T \tilde{\lambda}_{i,1} + \sum_{j=2}^{n_b} \delta \dot{x}_{i,j-1} \tilde{\lambda}_{ij} \right) \]
Rearranging terms gives

\[
\delta J' = \delta \tilde{\lambda}^T_1 \left[ -\bar{x}_1 + \bar{x}_{1,1} - \Delta t_1 \int_0^1 (1 - \tau) f_1 \, d\tau \right]
+ \sum_{i=2}^{N} \delta \tilde{\lambda}^T_i \left[ -\bar{x}_{i-1,1} - \Delta t_{i-1} \int_0^1 \tau f_{i-1} \, d\tau + \bar{x}_{i,1} - \Delta t_i \int_0^1 (1 - \tau) f_i \, d\tau \right]
+ \delta \tilde{\lambda}^T_{N+1} \left[ -\bar{x}_{N,1} - \Delta t_N \int_0^1 \tau f_N \, d\tau + \bar{x}_{N+1} \right]
+ \sum_{i=1}^{N-1} \sum_{j=1}^{n_b-1} \delta \tilde{\lambda}^T_{ij} \left[ -\bar{x}_{i,j+1} + \Delta t_i \int_0^1 \epsilon_j(\tau) f_i \, d\tau \right]
+ \delta \bar{x}_1^T \left\{ \bar{\lambda}_1 - \tilde{\lambda}_1,1 - \Delta t_i \int_0^1 (1 - \tau) \left[ L_{x_1} + \sum_{j=1}^{n_b} \alpha_j(\tau) \tilde{\lambda}_{ij}^T f_{x_1} \right] \, d\tau \right\}
+ \sum_{i=2}^{N} \delta \bar{x}_i^T \left\{ -\tilde{\lambda}_{i,1} - \Delta t_i \int_0^1 (1 - \tau) \left[ L_{x_i} + \sum_{j=1}^{n_b} \alpha_j(\tau) \tilde{\lambda}_{ij}^T f_{x_i} \right] \, d\tau \right\}
+ \bar{\lambda}_{i-1,1} - \Delta t_{i-1} \int_0^1 \tau \left[ L_{x_{i-1}} + \sum_{j=1}^{n_b} \alpha_j(\tau) \tilde{\lambda}_{i-1,1,j} f_{x_{i-1}} \right] \, d\tau \right\}
+ \sum_{i=1}^{N-1} \sum_{j=1}^{n_b-1} \delta \bar{x}_{ij}^T \left\{ \tilde{\lambda}_{i,j+1} + \Delta t_i \int_0^1 \epsilon_j(\tau) \left[ L_{x_i} + \sum_{j=1}^{n_b} \alpha_j(\tau) \tilde{\lambda}_{ij}^T f_{x_i} \right] \, d\tau \right\}
+ \delta \bar{x}_{N+1}^T \left\{ -\tilde{\lambda}_{N+1} + \bar{\lambda}_{N,1} - \Delta t_N \int_0^1 \tau \left[ L_{x_N} + \sum_{j=1}^{n_b} \alpha_j(\tau) \tilde{\lambda}_{N,j}^T f_{x_N} \right] \, d\tau \right\}
+ dt_1 \bar{\mu}_{N+1}^T \kappa_{N+1}^2 + \delta \nu^T \Psi
+ dt_1 \left( \tilde{H} + H \right)_{tf} + \sum_{i=1}^{N} \sum_{j=1}^{n_b} \delta \bar{u}_{ij} \Delta t_i \int_0^1 H_{ui}(\tau) \, d\tau
+ \sum_{i=1}^{N} \sum_{j=1}^{n_b} \delta \bar{v}_{ij} \Delta t_i \int_0^1 \alpha_j(\tau) \left[ g_i + \sum_{j=1}^{n_b} \alpha_j(\tau) \bar{k}_{ij}^2 \right] \, d\tau
+ \sum_{i=1}^{N} \sum_{j=1}^{n_b} \delta \bar{k}_{ij} \Delta t_i \int_0^1 2 \alpha_j(\tau) \left[ \sum_{j=1}^{n_b} \alpha_j(\tau) \bar{u}_{ij} \right]^T \left[ \sum_{j=1}^{n_b} \alpha_j(\tau) \bar{k}_{ij} \right] \, d\tau
\]

This grouping of terms shows the equations to be solved as coefficients of the various variational quantities. When the coefficients are set equal to zero, the variation of \( J' \) is
zero, approximating the first-order necessary conditions for optimal control. In addition to
the above, the costate boundary conditions need to be enforced, as provided for previously

\[
\hat{\lambda}_1^T + \frac{\partial \phi}{\partial x_0} + \nu^T \frac{\partial \Psi}{\partial x_0} = 0 \\
\hat{\lambda}_{N+1}^T - \frac{\partial \phi}{\partial x_f} - \nu^T \frac{\partial \Psi}{\partial x_f} = 0
\]

(30)

The equations when there are multiple phases are similar and can also be handled
by the code. Extra boundary conditions have to be specified, and the appropriate jump
conditions for the costates and Hamiltonian are handled automatically.

Implementation

The above equations are solved using a restricted-step Newton-Raphson method, as
implemented in a FORTRAN code. Sparse linear systems solvers from the Harwell subrou-
tine library [3] are used. The user needs to specify an initial guess for all of the variables
for the Newton-Raphson iteration.

The symbolic-manipulation package Macsyma developed by Symbolics [4] is used to
generate analytic partial derivatives. Macsyma is also used to generate the \( \alpha(\tau) \) and \( \epsilon(\tau) \)
polynomials, which come from a recursion formula involving derivatives and integrals of
polynomials, as developed in [2]. The user specifies the order of the polynomials, and
Macsyma generates the necessary ones. At this time all variables have the same order
shape function, but eventually the order for each variable will be able to be independently
specified. The order of the shape functions can be changed between runs, but only if the
desired number is less than that specified when the Macsyma-generated code was made. Otherwise the necessary polynomials will not be available.

The integrals in Eq. 29 are approximated using Gaussian integration, with the user
selecting the number of Gauss points, which at this time is constant for all of the inte-
grations. That number, as well as the number of elements can be adjusted between runs
through use of a namelist file and appropriate changes to the initial guess file.

Results

The code has been tested on a linear, single-state, single-control system and on a
multiple-state, single-control system with nonlinear system dynamics.

**Linear Problem** The linear problem we considered is a minimum energy problem for getting
from one position to another. The system dynamics are

\[
\dot{x} = x + u
\]

(31)
where \( x \) is the state and \( u \) is the control. The boundary conditions are

\[
\begin{align*}
\Psi_1 &= x_0 - (e - 1) \\
\Psi_2 &= x_f - (1 - e) = 0
\end{align*}
\]  

(32)

and the cost function is

\[
J = \int_0^1 \frac{1}{2} u^2 \, dt
\]

(33)

The analytic solution to this problem is

\[
\begin{align*}
x(t) &= e^{(1-t)} - e^t \\
\lambda(t) &= 2e^{(1-t)} \\
u(t) &= -2e^{(1-t)}
\end{align*}
\]

(34)

This problem was run for a variety of shape function orders, numbers of Gauss points, and numbers of elements. The code assumes a free time problem, introducing a nonlinear equation \( H(t_f) - \dot{H} = 0 \) even into linear problems. Knowing this, in all cases all of the linear equations were solved in a single iteration, while that last nonlinear equation converged better than quadratically. This problem is easy enough to work by hand reasonably, and the Jacobian matrix generated by the code matched the one we derived explicitly for the second-order shape function case.

Also, since all of the errors in all of the equations in (29) were solved to within \( 1.0e-10 \), the values for the costate were always just the negative of those for the control and the state boundary conditions were met. Therefore in comparing results, we will look at the initial and final values of only the control.

Table 1: Errors in initial and final values of control for various values of higher-order finite-element parameters

<table>
<thead>
<tr>
<th>( u(t_0) )</th>
<th>( u(t_f) )</th>
<th>Gauss Points</th>
<th>Shape Fn. Order</th>
<th>Number of Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>Error</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.18e-2</td>
<td>1.41e-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8.58e-4</td>
<td>2.33e-3</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6.00e-6</td>
<td>1.63e-5</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2.36e-8</td>
<td>6.41e-8</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5.92e-11</td>
<td>1.61e-10</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>1.23e-2</td>
<td>3.34e-2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5.12e-5</td>
<td>1.39e-4</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9.12e-8</td>
<td>2.48e-7</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>9.01e-11</td>
<td>2.45e-10</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5.35e-14</td>
<td>1.50e-13</td>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>
As one can see, the errors reduce dramatically as the order of the shape functions increases. Not shown is a similar decrease in error as the number of Gauss points increases, to a minimum when the number of Gauss points equals the order of the element. This is similar performance to the time-marching algorithm, as presented in our previous report. With further analysis we will do trade studies comparing the computational effort necessary to solve the problem with more elements vs. using higher-order shape functions.

**Nonlinear Problem** Next we computed some preliminary solutions to a problem with nonlinear system dynamics that involves maximum velocity transfer of a particle of mass $m$ to a rectilinear path in a fixed time (see [5], pg. 59). The mass is acted on by a force of constant magnitude $ma$ and variable heading $\beta(t)$. The states for this problem are position of the particle $x$ (horizontal) and $y$ (vertical) and the corresponding velocity components $u$ and $v$. The differential equations for this system are then:

\[
\begin{align*}
\dot{x} &= u \\
\dot{y} &= v \\
\dot{u} &= a \cos \beta \\
\dot{v} &= a \sin \beta
\end{align*}
\]

with initial conditions corresponding to a zero velocity at the origin and with terminal constraints that the particle is in horizontal flight at a given height (assumed here to be 1)

\[
\begin{align*}
\Psi_1 &= u(0) = 0 \\
\Psi_2 &= v(0) = 0 \\
\Psi_3 &= x(0) = 0 \\
\Psi_4 &= y(0) = 0 \\
\Psi_5 &= y(t_f) - 1 = 0 \\
\Psi_6 &= v(t_f) = 0
\end{align*}
\]

for an unspecified final horizontal position $x(t_f)$ and for a final horizontal velocity $u(t_f)$ to be maximized. The cost function is then

\[
J = u(t_f)
\]
Preliminary results are shown in Table 2. Missing values were for cases that did not converge, which is not surprising given the small number of elements being used. For this nonlinear problem the improvement with the higher-order element is not as dramatic. For 4 and 5 elements all errors decrease with the higher-order elements. For 2 elements some errors do not decrease. This may be due to the need for a larger number of Gauss points when the order is increased (6 points were used). For the simplest (h-version) element, 1 Gauss point is optimum in the sense that it is the smallest number of points which gives acceptable error. However, when one adds the higher-order shape functions, a larger number of Gauss points will always be required for nonlinear problems than for linear ones.

The optimum number of Gauss points seems to be dependent on both the maximum element order and the types of nonlinearities in the problem. It is possible that there is a simple calculation, using some terms from the equations under consideration, which can be done for any given problem and which will approximately determine the optimum number of Gauss points for that particular problem as a function of the orders of element from 2 up to the maximum needed. Whether this is in fact true should become clearer as more problems are tried and more cases of these problems are run. Recent developments by Hinnant (of the Computational Mechanics Branch at Langley) may also lead to savings in the numerical quadrature costs.

One possible drawback of higher-order finite element schemes is the increased computational effort within each element required in implementing hp-version finite elements. We will ultimately determine whether this computational effort is sufficiently offset by the reduction in the number of time elements used and improvements in the Newton-Raphson convergence so as to be useful in solving optimal control problems in real time. Also, because certain of the element interior unknowns can be eliminated at the element level by solving a small set of nonlinear algebraic equations in which the nodal values are taken as given, the scheme may turn out to be especially powerful in a parallel computing environment. A different processor could be assigned to each element. The number of processors,
strictly speaking, would not be required to be any larger than the number of sub-regions which are free of discontinuities.

References


Introduction

The purpose of this research effort was to begin the study of the application of \( hp \)-version finite elements to the numerical solution of optimal control problems. Under NAG-939, the hybrid MACSYMA/FORTRAN code GENCODE was developed which utilized \( h \)-version finite elements to successfully approximate solutions to a wide class of optimal control problems. In that code the means for improvement of the solution was the refinement of the time-discretization mesh. With the extension to \( hp \)-version finite elements, the degrees of freedom include both nodal values and extra interior values associated with the unknown states, co-states, and controls, the number of which depends on the order of the shape functions in each element. For details, see [1].

One possible drawback is the increased computational effort within each element required in implementing \( hp \)-version finite elements. We are trying to determine whether this computational effort is sufficiently offset by the reduction in the number of time elements used and improved Newton-Raphson convergence so as to be useful in solving optimal control problems in real time. Because certain of the element interior unknowns can be eliminated at the element level by solving a small set of nonlinear algebraic equations in which the nodal values are taken as given, the scheme may turn out to be especially powerful in a parallel computing environment. A different processor could be assigned to each element. The number of processors, strictly speaking, is not required to be any larger than the number of sub-regions which are free of discontinuities of any kind.

Summary of Completed Work

In order to acquaint Mr. Warner better with the workings of GENCODE and with finite element methods in general, much of the first part of the year was spent in the study and modification of GENCODE. The code as described in [1] handled only two stages, defined here as a time intervals with distinct differential equations. Mesh refinement was also prescribed beforehand such that the code always began with two elements per phase, doubling upon successful convergence until the final desired number discretization was reached.

The modifications were conducted in parallel with a similar effort at NASA Langley Research Center to implement the \( h \)-version finite elements solution procedure into a MATLAB optimal control problem solver, VTOTS. Since the code was being expanded to include an arbitrary number of stages, the governing equations were rearranged to better take advantage of the sparse matrix solvers included in the Harwell subroutine library. Instead of having separate initial conditions, terminal conditions, and jump conditions at stage breaks, the code was modified such that all boundary conditions, include continuity between stages, were included in a single set of boundary conditions.

The other major advancement of GENCODE was the freedom to specify arbitrarily the desired time discretization in each phase. Once the solution to that discretization converges successfully, that solution can be interpolated in the code to provide initial guesses for any subsequently desired discretization. This flexibility proved most helpful in solving difficult problems.
Once completed, the new GENCODE was tested against VTOTS, with refinements and error correction occurring in both codes as errors were discovered. Since then, VTOTS has been updated to include state constraints with a more automated switching structure, an improvement that we plan to incorporate in the new GENCODE.

After the GENCODE testing was completed, we formulated the structure of using p-version finite elements to solve two-point boundary value problems. A summary of that development is included below. At this stage, a time-marching algorithm has been developed and tested which utilizes p-version elements. Results were obtained for a number of problems, looking at accuracy and computational effort over various length time elements, numbers of Gauss quadrature points, and orders of shape functions. The improvements in accuracy over h-version elements are dramatic, as expected, and the overall results are encouraging enough that we plan to move forward into the development of a two-point boundary problem solver.

Higher-Order Elements

In moving toward eventually applying p-version finite elements to the solution of optimal control problems, we first developed how to use them to solve nonlinear two-point boundary value problems. In order to test their performance we started solving problems involving given initial states and only one time element (i.e., initial-value or time-marching problems).

Formulation for Optimal Control Problems

The differential equations of interest are assumed to be of the form

\[ \dot{x} = f(x) \quad x \in \mathbb{R}^n \]  

where \( f \) is an autonomous function of the state vector \( x \). The boundary conditions can be specified at the initial time, \( t_0 \), the final time \( t_f \), or some combination of both. For this we denote

\[ x(t_0) = x_0 \]
\[ x(t_f) = x_f \]  

For solving optimal control problems, this problem formulation corresponds to the Euler-Lagrange equations when the control can be solved explicitly in terms of the states and costates from the optimality condition. Then the vector \( x \) would include both the states and the costates, and the boundary conditions would include the derived costate boundary conditions.

The time interval is broken up into \( N \) not necessarily equal length time elements, \( \Delta t_i \), such that the time at each element boundary \( t_i \) is calculated as

\[ t_1 = t_0 \]
\[ t_i = t_{i-1} + \Delta t_i \quad i = 2, \ldots, N + 1 \]
and we define the states at these nodes to be

$$\hat{x}_i = x(t_i) \quad i = 1, \ldots, N + 1$$

(4)

Ultimately, only the $t_0$ and $t_f$ nodal values of the states are of interest as the values at the internal nodes will be known functions of the values of the states within each time interval. The time within the $i$th element, $t_i$ is expressed as $t_i = t_{i-1} + \tau \Delta t_i$ where $0 \leq \tau \leq 1$ and

$$\tau = \frac{t_i - t_{i-1}}{\Delta t_i}$$

(5)

so that $dt = \Delta t_i d\tau$.

The state equations are enforced through use of Lagrange multipliers $\delta \lambda$ inside the time interval and the boundary conditions are enforced as natural boundary conditions at the endpoints:

$$\int_{t_0}^{t_f} \delta \lambda^T [\dot{x} - f(x)] dt + [\delta \lambda^T (\dot{x} - x)]_{t_0}^{t_f} = 0$$

(6)

Substituting for the normalized time $\tau$ yields

$$\sum_{i=1}^{N} \Delta t_i \int_0^1 \left[ \delta \lambda_i^T \dot{x}_i - \delta \lambda_i^T f(x_i) \right] d\tau + [\delta \lambda_i^T (\dot{x} - x)]_{t_0}^{t_f} = 0$$

(7)

where the subscript denotes the element number or node for each variable. Integrating by parts results in

$$\sum_{i=1}^{N} \Delta t_i \int_0^1 \left[ \delta \lambda_i^T x_i + \delta \lambda_i^T f(x_i) \right] d\tau + \delta \lambda_i^T \dot{x}_1 - \delta \lambda_{N+1}^T \dot{x}_{N+1} = 0$$

(8)

Now from Ref. 2, we define $C^0$ shape functions for the Lagrange multipliers in each element in terms of nodal values and internal values

$$\delta \lambda_i = \delta \lambda_i (1 - \tau) + \delta \lambda_{i+1} \tau + \delta \lambda_{i,1} (1 - \tau) \sqrt{6} + \cdots + \delta \lambda_{i,m} \alpha_m(\tau)$$

$$\frac{d\delta \lambda_i}{d\tau} = \delta \lambda_i' = \Delta t_i \delta \lambda_i = -\delta \lambda_i + \delta \lambda_{i+1} + \delta \lambda_{i,1} (1 - 2 \tau) \sqrt{6} + \cdots + \delta \lambda_{i,m} \alpha_m'(\tau)$$

(9)

Here $m$ is the order of the shape function, with $m = 0$ corresponding to $h$-version finite elements. The polynomial function $\alpha_m(\tau)$ is determined through a set of relationships developed in Ref. 2, and $\alpha'_m(\tau)$ is the derivative of $\alpha_m(\tau)$ with respect to $\tau$. Similarly, the shape functions for the values of the states internal to each element are

$$x_i = \bar{x}_{i,1} + \bar{x}_{i,2} \sqrt{6} \left( \tau - \frac{1}{2} \right) + \cdots + \bar{x}_{i,m+1} \beta_{m+1}(\tau)$$

(10)
Here again the polynomial function $\beta_m(\tau)$ is a recursive relationship developed in [2]. One fewer $\delta \lambda$ term is used than $\bar{x}$ terms to ultimately ensure an equal number of equations and unknowns.

Substituting Eqs. (9) and (10) into Eq. (8) and taking $m = 1$ for simplicity results in

$$
\sum_i^N \int_0^1 \left\{ \left[ -\delta \lambda_i + \delta \lambda_{i+1} + \delta \lambda_{i,1}(1 - 2\tau)\sqrt{6} \right]^T \left[ \bar{x}_{i,1} + \bar{x}_{i,2}\sqrt{6} (\tau - \frac{1}{2}) \right] \\
+ \Delta t_i \left[ \delta \lambda_i (1 - \tau) + \delta \lambda_{i+1}\tau + \delta \lambda_{i,1}(1 - \tau)\tau\sqrt{6} \right]^T f \left[ \bar{x}_{i,1} + \bar{x}_{i,2}\sqrt{6} (\tau - \frac{1}{2}) \right] \right\} d\tau \\
+ \delta \bar{x}_1^T - \delta \lambda_{N+1}^T \bar{x}_{N+1} = 0
$$

Integrating all the terms that do not depend on $f(\tau)$ and simplifying results in

$$
\sum_i^N \Delta t_i \int_0^1 \left[ \delta \lambda_i (1 - \tau) + \delta \lambda_{i+1}\tau + \delta \lambda_{i,1}(1 - \tau)\tau\sqrt{6} \right]^T f \left[ \bar{x}_{i,1} + \bar{x}_{i,2}\sqrt{6} (\tau - \frac{1}{2}) \right] d\tau \\
+ \sum_i^N \left( -\delta \lambda_i^T \bar{x}_{i,1} + \delta \lambda_{i+1}^T \bar{x}_{i,1} - \delta \lambda_{i,1}^T \bar{x}_{i,2} \right) + \delta \bar{x}_1^T - \delta \lambda_{N+1}^T \bar{x}_{N+1} = 0
$$

Finally, grouping terms multiplying each of the LaGrange multipliers yields

$$
\delta \bar{x}_1^T \left[ -\bar{x}_{1,1} + \bar{x}_1 + \Delta t_1 \int_0^1 (1 - \tau)f(x_1) \ d\tau \right] \\
+ \delta \lambda_{N+1}^T \left[ \bar{x}_{N,1} - \bar{x}_{N+1} + \Delta t_N \int_0^1 \tau f(x_N) \ d\tau \right] \\
+ \sum_{i=2}^N \delta \lambda_i^T \left[ -\bar{x}_{i,1} + \bar{x}_{i-1,1} + \Delta t_i \int_0^1 (1 - \tau)f(x_i) \ d\tau + \Delta t_{i-1} \int_0^1 \tau f(x_{i-1}) \ d\tau \right] \\
+ \sum_{i=1}^N \delta \lambda_{i,1}^T \left[ -\bar{x}_{i,2} + \Delta t_i \int_0^1 \sqrt{6}(1 - \tau)\tau f(x_i) \ d\tau \right] = 0
$$

If $n$ is the dimension of the state vector, then Eq. (13) results in $(2N + 1)n$ equations for $(2N + 2)n$ unknowns: $n$ unknowns at each end point and $2n$ unknowns within each element. If the order of the shape function $m$ is greater than one, then the last block of Eq. (13) becomes $nm$ equations, and $x_i$ would then depend on $\bar{x}_{i,j}$ for $j = 1, \ldots, m + 1$. This brings the total to $(N + 1 + Nm)n$ equations and $(N(m + 1) + 2)n$ unknowns. Thus for a unique solution to the problem, a combination of $n$ initial and final conditions on the states must be specified. Note that when solving these equations by a Newton-Raphson approach that the Jacobian matrix is block diagonal.
Note that if the order of the shape function \( m \) is zero, then \( x_1 = \bar{x}_{i,1} \) the \( h \)-version equations result

\[
\begin{align*}
\delta \lambda^T \left[ -\bar{x}_1 + \bar{x}_1 + \Delta t_1 f(x_i)/2 \right] \\
+ \delta \lambda^T_{i+1} \left[ \bar{x}_N - \bar{x}_{N+1} + \Delta t_N f(x_N)/2 \right] \\
+ \sum_{i=2}^{N} \delta \lambda^T_i \left[ -\bar{x}_i + \bar{x}_{i-1} + \Delta t_i f(x_i)/2 + \Delta t_{i-1} f(x_{i-1})/2 \right] = 0
\end{align*}
\]

(14)

**Time-Marching Algorithm**

To help determine the feasibility of using \( p \)-version finite elements to solve optimal control problems, Eqs. (13) were first incorporated into a time-marching algorithm. In this case, the required number of specified boundary conditions are given as initial conditions, \( \bar{x}_1 = \pi(t_0) \). Eqs. (13) then reduce to the case of only one element \( N = i = 1 \), and we can drop the element subscripts. Thus, the remaining subscripts refer only to the element order. Rewriting Eqs. (13) in this way one obtains

\[
\begin{align*}
\bar{x}_1 &= \bar{x}_1 - \Delta t \int_0^1 (1 - \tau) f \left[ \bar{x}_1 + \bar{x}_2 \sqrt{6} \left( \tau - \frac{1}{2} \right) \right] d\tau \\
0 &= -\bar{x}_2 + \Delta t \sqrt{6} \int_0^1 (1 - \tau) f \left[ \bar{x}_1 + \bar{x}_2 \sqrt{6} \left( \tau - \frac{1}{2} \right) \right] d\tau \\
\bar{x}_2 &= \bar{x}_1 + \Delta t \int_0^1 f \left[ \bar{x}_1 + \bar{x}_2 \sqrt{6} \left( \tau - \frac{1}{2} \right) \right] d\tau 
\end{align*}
\]

(15)

again assuming first-order shape functions. For each additional order, there is one more equation and one more unknown.

For shape functions of arbitrary order, the first two of Eqs. (15) become \( m + 1 \) equations, and they are solved for the internal values, \( \bar{x}_i \) for \( i = 1, 2, \ldots, m + 1 \). These values are then used to calculate the nodal values, \( \bar{x}_2 \) from the third of Eqs. (15). These final values become the initial values for the next element, and the process repeats over all the elements. Notice that in the solution of this problem with a Newton-Raphson algorithm, the Jacobian matrix is, in general, fully populated.

The process of solving these equations has been implemented in a FORTRAN subroutine using a standard full-step Newton-Raphson algorithm. The routine utilizes the Harwell family of subroutines for solving fully-populated linear systems. A user interface was developed using the symbolic manipulation software MACSYMA which generates the appropriate expressions for error evaluation and the Jacobian matrix. Unlike for \( h \)-version finite elements, the quadratures in Eq. (15) in general cannot be done be done by inspection and similarly are too complicated for software such as MACSYMA. To perform the integrations, Gaussian quadrature with a variable number of Gauss points was implemented here.
Example Problem Formulations

The code was run on a variety of problems for varying orders of shape functions, numbers of elements, and numbers of Gauss points. The goal was to determine whether such a code could run in a reasonable length of time for complicated problems, thereby determining the feasibility of applying p-version elements to two-point boundary value problems. That being established, the next step was to determine the relationship between the number of Gauss points and the accuracy of the solution, in hopes of finding a formula such as in the case of h-version elements. The results of this experimenting will be described for a two-state linear system, two one-state systems with different types of nonlinear dynamics, and an 8-state missile system.

Linear System

The first problem that was examined was an idealized spring-mass system, with a spring constant, $k$ and mass, $m$.

$$m\ddot{x} = -kx$$

with

$$x(0) = 1.0$$
$$\dot{x}(0) = 1.0$$

(16)

The analytical solution to this with $k = m = 1$ is

$$x(t) = \cos(t) \quad x(1) = 0.543087528$$
$$\dot{x}(t) = -\sin(t) \quad \dot{x}(1) = -0.839676091$$

(17)

In Fig. 1, the log of the percentage error in the time-marching algorithm at the final time is plotted versus the number of Gauss points for various orders of shape functions. The number of elements in all cases was 5. The percentage error, $E$, was calculated in terms of the exact answer from Eq. (17) as

$$E(1) = \frac{100}{n} \sum_{i=1}^{n} \frac{y_i(1) - \hat{x}_N}{y_i(1)}$$

(18)

where $N$ is the number of elements, and $n$ is the dimension of the state vector $y = \{x, \dot{x}\}$. In subsequent examples, the error is calculated similarly with $y$ again being the appropriate state vector. This error criterion gives a good idea of how errors are being propagating through the algorithm.

Notice that in this case the error is minimized when the number of gauss points is one more than the order of the shape function in each element. This will hold true for all linear systems because the polynomial in $r$ to be integrated is of no higher order than $2m + 1$, where $m$ is the order of the shape function. As shown in [3], using $n$ Gauss quadrature points and weights results in exact integration of polynomials up to order $2n - 1$. Thus
only \( m + 1 \) Gauss points are needed to exactly evaluate the integrals when the order of the shape functions is \( m \). Unfortunately this result does not hold for nonlinear systems.

Changing the number of elements adjusted the error appropriately in each case, but did not change the relative characteristics between the orders of the shape functions. Also, the number of Newton steps required was only one for all elements sizes, orders of shape function, and numbers of Gauss points. Thus for a linear system, the tradeoffs between accuracy and computational effort are very clear.

**Nonlinear System with Quadratic Nonlinearities**

The next problem complicates matters some by replacing the linear term with a quadratic in a first order differential equation

\[
\dot{x} = ax^2
\]

\[
x(0) = 1.0
\]

(19)

where \( a \) is a constant. The analytic solution to this problem with \( a = 0.5 \) is

\[
x(t) = \frac{-1}{(0.5t - 1)} \rightarrow x(1) = 2.0
\]

(20)

In Fig. 2, the log of the percentage error at the final time from Eq. (18) is again plotted versus the number of gauss points for various orders of shape functions. The results plotted were obtained with 5 elements, but the trends are the same for any number. In this case the error is again minimized for the number of Gauss points equals the order of the shape function plus one. What is different here is that the accuracy actually goes down as more Gauss points are added, enforcing the point described in [3] that more Gauss points does not always mean higher accuracy.

The number of Newton steps per element decreased on average as the number of elements increased, but only from 6 to 5. And there was no significant decrease in the number of Newton steps required as the order of the element increased. Thus the high accuracy solutions still require considerably more computational effort.

**Nonlinear System with Infinite Order Nonlinearities**

The next problem examined replaced the quadratic nonlinearity with an exponential function, an infinite-order nonlinearity.

\[
\dot{x} = e^x
\]

\[
x(0) = -1
\]

(21)

The analytic solution to this problem is

\[
x(t) = -\ln(-t + e) \rightarrow x(1) = -\ln(e - 1)
\]

(22)
In Fig. 3, the log of the percentage error at the final time from Eq. (18) is again plotted versus the number of Gauss points for various orders of shape functions. The figure correspond to 5 elements used, but the trends are the same for any number. Here the line begins to blur about the optimal number of Gauss points. In all cases, the error is very close to the minimum with one more Gauss point than the order of the shape function, but depending on the order and the number of elements, one additional Gauss point can still improve accuracy significantly.

It is encouraging, however, that the number of Newton steps here did decrease with increased order, though only from 6 to 5 on average for each element. In this problem, extra iterations are rather insignificant, as the problem has only one state. Also note that in none of the above examples were there any cases in which the algorithm failed to converge.

Nonlinear Missile Model

To give the time-marching algorithm a harder test, we next tested it on an 8-state missile system. While the equations themselves are unimportant, it should be noted that all the nonlinear system dynamics were retained, including table-lookups on highly nonlinear lift and drag coefficients. The missile model was integrated from launch for 5 s, when the engines would be throttled back. The two controls are angle of attack and roll angle, both of which were set nominally to 1° throughout the time interval.

To provide a comparison, the model was integrated using the Runge-Kutta method and 1000 time steps. Fig. 4 plots the error in the finite-element algorithm as calculated in Eq. (18) using the shooting results as the exact answer. With any fewer than 5 time elements, the algorithm had difficulty converging. A restricted-step Newton method [4] was then implemented, which alleviated some of the convergence difficulties while not appreciably affecting accuracy.

Unfortunately, none of the trends we encountered before with regard to optimal number of Gauss points held true in this case. It is clear that for "real life" problems, the optimal number of Gauss points may have to be determined on a case-by-case basis. Therefore, any future algorithms will include capability for adjusting this number. Also, increasing the order once again brought down the number of Newton steps required for convergence.

Future work

The improvement in accuracy achievable by increasing the order of the shape functions in all four problems presented here is exceptional. The question still remains, however, as to how the extra computational effort will trade off with this increased accuracy when the methodology is extended to two-point boundary value problems. Another open question is how to find a simple means for obtaining the optimal number of Gauss points for more complicated problems.

Given the encouraging results we have seen using h-version finite elements so far, we plan to extend GENCODE to include the hp-version finite elements and begin testing it on
optimal control problems, including those with state constraints. There are also a number of theoretical aspects of these finite elements yet to be explored including the effect on element stability and convergence properties.

References


Fig. 1: Time-Marching Error at Final Time for Spring Mass System

Fig. 2: Time-Marching Error at Final Time for $\dot{x} = x^2$
Fig. 3: Time-Marching Error at Final Time for $\dot{x} = e^x$

Fig. 4: Time-Marching Error at Final Time for Missile system
Development of an Adaptive $hp$-Version Finite Element Method for Computational Optimal Control

Final Report
NASA Grant NAG-1-1435
Apr. 22, 1992 – Nov. 30 1994

Prof. Dewey H. Hodges, Principal Investigator
Michael S. Warner, Graduate Research Assistant
School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332-0150

Research Supported by
Spacecraft Control Branch
Technical Monitor: Dr. Daniel D. Moerder
Mail Stop 161
NASA Langley Research Center
Hampton, Virginia 23681-0001
1 Introduction

In this research effort, the usefulness of $hp$-version finite elements and adaptive solution-refinement techniques in generating numerical solutions to optimal control problems has been investigated. Under NAG-939, a general FORTRAN code was developed which approximated solutions to optimal control problems with control constraints and state constraints [1, 2]. Within that methodology, to get high-order accuracy in solutions, the finite element mesh would have to be refined repeatedly through bisection of the entire mesh in a given phase. In the current research effort, the order of the shape functions in each element has been made a variable, giving more flexibility in error reduction and smoothing. Similarly, individual elements can each be subdivided into many pieces, depending on the local error indicator, while other parts of the mesh remain coarsely discretized. The problem remains to reduce and smooth the error while still keeping computational effort reasonable enough to calculate time histories in a short enough time for on-board applications.

As an aid in evaluation of the error for use in solution refinement, in optimal control problems, the integral of the boundary value problem (the Hamiltonian) can be made to equal zero. Using this information alone, a reasonable error indicator can be developed, but as is shown in later sections, it is not consistently accurate and only tends to model parts of the error well. However, the Hamiltonian can be used as a check of the more sophisticated error estimators.

The error estimator currently being developed is based on work done by Estep, et al., [3], for initial value problems. This will involve the solution of a dual problem, a linear differential equation of the same order as the main problem, the behavior of which indicates the overall level of error in the main problem. $hp$-refinements will then be build around this error indicator to attempt to equidistribute the error through the time interval and attempt to provide a minimum of error for a given computational effort. It remains to be investigated if this refinement process can be accomplished in a short enough time.

The rest of this paper describes the work that has already been done in developing an adaptive scheme for implementing the $hp$-version of the finite element method to solve optimal control problems. The family of optimal control problems that can currently be solved will be defined in
section 2. Next in section 3, the variables of the problem will be modelled using shape functions, thereby establishing the algebraic equations to be solved. In section 4, some textbook-style problems will be solved using with this method and the results will be analyzed. In section 5, the Hamiltonian will be shown to be a reasonable, but inconsistent, error measure, along with some preliminary results in adaptively refining solutions to textbook problems. Following that will be the preliminary development of the residual error indicator discussed above. Finally in section 6, an outline of future research work will be given.

2 Optimal Control Problems

A FORTRAN code has been developed which uses $hp$-version finite elements to approximate solutions to a particular subset of optimal control problems. In this section this subset of problems will be defined, and the equations to be solved using finite elements will be established.

2.1 Problem formulation

Systems being studied are governed by general, nonlinear differential equations

$$\dot{x} = f(x, u, t) \quad x \in R^n, u \in R^m, t \in [t_0, t_f]$$

(1)

where the state vector $x$ describes the state of the system, $u$ is a vector of control variables, and $t$ is the time. Although the methodology and the code are not so restricted, for simplicity's sake, this discussion is confined to problems with only a single set of differential equations in a single time interval. Furthermore, the functions $f$ are assumed to be differentiable with respect to their arguments, and the states are assumed to be continuous, from initial time $t_0$ (presumed to be zero) to final time $t_f$, where $t_f$ can be fixed or free.

General boundary conditions on the states can be specified at the initial time, the final time, or some combination of both, in the form

$$\Psi[x(t_0), x(t_f), t_f] = 0 \quad \Psi \in R^{ne}$$

(2)

Let $J$ be a cost functional to be minimized that can contain both a scalar penalty on the states at the endpoints $t_0$ or $t_f$ plus an integral penalty on
the states, controls, and time:

\[ J = \phi[x(t_0), x(t_f), t_f] + \int_{t_0}^{t_f} L(x, u, t) \, dt \]  

(3)

The optimal control problem is then to find the control vector time history \( u(t) \) which causes the system governed by Eq. (1) to meet the boundary conditions (2) such that the given cost functional (3) is minimized. Admissible control histories are assumed to be bounded and continuous.

This formulation also allows for inequality constraints of the form:

\[ g(x, u) \leq 0 \quad g \in R^{n_x} \]  

(4)

no more than \( n_u \) of which can be active at any one time. The function \( g \) need not be a function of the states, but it must be a function of the control vector. Otherwise it falls under the category of state constraints, which require special handling that has not been developed yet under this methodology.

The constraints in Eq. (4) are enforced through use of slack variables, \( k \), such that (4) is replaced by the equality constraints,

\[ g_i(x, u) + k_i^2 = 0 \quad i = 1 \ldots n_g \]  

(5)

To simplify notation, a vector of these squared slack variables is defined such that

\[ K_i = k_i^2 \quad i = 1 \ldots n_g \]  

(6)

2.2 Calculus of variations

In using calculus of variations to minimize a cost functional subject to constraints [4, 5], the residuals of the differential equations, control constraints, and boundary conditions are adjoined to the original cost function by means of Lagrange multipliers \( \lambda(t) \), \( \mu(t) \), and \( \nu \) respectively. This yields a new cost function \( J' \) such that

\[ J' = \phi[x(t_0), x(t_f), t_f] + \nu^T \Psi[x(t_0), x(t_f), t_f] \]

\[ + \int_{t_0}^{t_f} \left\{ L(x, u, t) + \lambda^T [f(x, u, t) - \dot{x}] + \mu^T [g(x, u) + K] \right\} dt \]  

(7)
which equals $J$ if all the constraints are satisfied. The functions $\lambda(t)$ are known as the costates.

Next, to simplify notation, define quantities

$$H \equiv L + \lambda^T f + \mu^T (g + K)$$

$$\Phi \equiv \phi + \nu^T \Psi$$

where $H$ is known as the Hamiltonian of the system. Also define $x_f \equiv x(t_f)$ and $x_0 \equiv x(t_0)$, thus reducing $J'$ to

$$J' = \Phi(x_0, x_f, t_f) + \int_{t_0}^{t_f} \left[ H(x, u, t) - \lambda^T \dot{x} \right] dt$$

To satisfy the first-order necessary conditions for a local minimum, the authors of Ref. [5] find an extremal of $J'$ by varying the control vector, with the variation in the state vector (and other variables) being a result of the variation in the control vector. Here instead, the development in [1] is followed in which the first variation of $J'$ is taken, allowing independent variations in the states, state rates, controls, Lagrange multipliers, slack variables, and final time, yielding

$$\delta J' = \delta \nu^T \Psi + \frac{\partial \Phi}{\partial x_0} dx(t_0) + \frac{\partial \Phi}{\partial x_f} dx(t_f) + \left[ \frac{\partial \Phi}{\partial t} + H - \lambda^T \dot{x} \right]_{t_f} dt_f$$

$$+ \int_{t_0}^{t_f} \left[ \frac{\partial H}{\partial x} \delta x + \frac{\partial H}{\partial u} \delta u - \lambda^T \delta \dot{x} + \delta \lambda^T (f - \dot{x}) \right] dt$$

$$+ \delta \mu^T (g + K) + \mu^T \delta K \right] dt$$

where

$$\delta K_i = 2k_i \delta k_i \quad i = 1 \ldots n_g$$

This results in a weaker form of the equations, though as will be shown, the equations developed in [5] are all still satisfied.

The variations denoted by $\delta$ are variations holding time fixed (appropriate for variations of quantities under an integral), while those with a $d$ are total variations or differentials, which allow time to vary (appropriate for varying quantities at an end point). These variations are related at the end points by

$$dx(t_0) = \delta x(t_0)$$

$$dx(t_f) = \delta x(t_f) + \dot{x}(t_f) dt_f$$
since \( t_f \) is allowed to vary while \( t_0 \) is not.

In an attempt to group terms by their variational coefficient, one can remove the time derivatives of variational parameters by integrating \( \lambda^T \delta \dot{x} \) by parts and expand the total variations at the end points. Eq. (11) then becomes

\[
\delta J' = \delta v^T \Psi + \left( \frac{\partial \Phi}{\partial t} + H \right)_{t_f} dt_f + \left( \frac{\partial \Phi}{\partial x_0} - \lambda^T \right)_{t_0} \delta x(t_0)
+ \left[ \frac{\partial \Phi}{\partial x_f} - \lambda^T \right]_{t_f} \dot{x}(t_f) dt_f + \left( \frac{\partial \Phi}{\partial \dot{x}} \right)_{t_f} \delta \dot{x}(t_f)
+ \int_{t_0}^{t_f} \left[ \frac{\partial H}{\partial x} \delta x + \frac{\partial H}{\partial u} \delta u + \lambda^T \delta x + \delta \lambda^T (f - \dot{x}) \right]
+ \delta \mu^T (g + K) + \mu^T \delta K \] dt

Defining subscripts on \( H \) to denote partial derivatives and rearranging terms gives:

\[
\delta J' = \delta v^T \Psi + \left( \frac{\partial \Phi}{\partial t} + H \right)_{t_f} dt_f + \left( \frac{\partial \Phi}{\partial x_0} - \lambda^T \right)_{t_0} \delta x(t_0)
+ \left( \frac{\partial \Phi}{\partial x_f} - \lambda^T \right)_{t_f} \dot{x}(t_f) dt_f + \left( \frac{\partial \Phi}{\partial \dot{x}} \right)_{t_f} \delta \dot{x}(t_f)
+ \int_{t_0}^{t_f} \left[ H_u \delta u + H_z \delta x + \dot{\lambda}^T \delta x + \delta \lambda^T (f - \dot{x}) \right]
+ \delta \mu^T (g + K) + \mu^T \delta K \] dt

Now, defining \( \hat{H} \) and \( \hat{\lambda} \) as

\[
\hat{\lambda}^T \equiv \begin{cases} -\frac{\partial \Phi}{\partial x_0} & t = t_0 \\ \frac{\partial \Phi}{\partial x_f} & t = t_f \end{cases}
\]

\[
\hat{H} \equiv \frac{\partial \Phi}{\partial t}
\]
and combining terms yields:

\[ \delta J' = \delta v^T \Psi + \left( \dot{H} + H \right) t_f + \left[ (\dot{\lambda} - \lambda)^T dx \right]_{t_0}^{t_f} + \int_{t_0}^{t_f} \left[ H_u \delta u + H_x \delta x + \dot{\lambda}^T \delta x + \delta \lambda^T f + \delta \lambda^T \dot{x} + \delta \mu^T (g + K) + \mu^T \delta K \right] dt \] (19)

In an analogous way to finding the extremum of a scalar function of many variables, the first order necessary condition for an extremal solution to the functional \( J' \) is that \( \delta J' = 0 \) for arbitrary independent variations in the states, costates, time, slack variables and other Lagrange multipliers. Since this includes the case where all the variations except any one are zero, this condition implies that for this cost functional to have a minimum, all the quantities multiplied by each of the variations must be zero. At the boundaries, this results in the following equations:

\[ \lambda(t_0) + \dot{\lambda}(t_0) = 0 \] (20)
\[ \lambda(t_f) + \dot{\lambda}(t_f) = 0 \] (21)
\[ \Psi = 0. \] (22)

Also, if the final time is indeed free to vary, then this relation must also hold:
\[ H + \dot{H} = 0 \] (23)
otherwise, \( dt_f = 0 \) and Eq. (23) can be ignored. In appendix A the difference between these two cases will be discussed.

The integral in Eq. (19) is made equal to zero in continuous time by enforcing the following equations:

\[ \dot{\lambda}(t) + H_x(x, \lambda, u, t) = 0 \] (24)
\[ \dot{x}(t) - f(x, u, t) = 0 \] (25)
\[ H_u = 0 \] (26)
(herby refered to as the costate equations, state equations, and the optimality condition respectively), and the control constraint equations:
\[ g + K = 0 \] (27)
\[ 2\mu_ik_i = 0 \quad i = 1 \ldots n_g \] (28)

These equations are equivalent to the Euler-Lagrange equations developed in [5] and define a two-point boundary value problem in \( x \) and \( \lambda \).

To approximately solve this problem using \( hp \)-version finite elements, a form of these equations will be developed such that all the boundary conditions are enforced weakly through the use of Lagrange multipliers. This will allow shape functions to be chosen that do not need to meet the boundary conditions, thus allowing the same set of shape functions to be used to solve a wide class of optimal control problems. This way the number of equations to be solved is not affected by the boundary conditions, except for fixed vs. free \( t_f \).

To look at the equations on the interior of the time interval, first assume the boundary conditions are satisfied. Eq. (19) then becomes:

\[ \delta J' = \int_{t_0}^{t_f} \left[ H_u \delta u + H_x \delta x - \delta \dot{x}^T \lambda + \delta \dot{\lambda}^T f + \delta \dot{\lambda}^T x + \delta \mu^T (g + K) + \mu^T \delta K \right] dt \] (29)

At this stage, the analogy can be drawn between variational methods and Galerkin methods. Interpreting the variations in the variables as test functions, the equation \( \delta J' = 0 \) becomes the representation of the residuals in meeting Eqs. (24 – 28) being orthogonal to the test functions (\( \delta x, \delta u, \ldots \)). This observation will become important when we later look at \emph{a posteriori} error estimators.

### 2.3 Discretization of the problem

Again looking at Eq. (29) as a variational problem, to solve \( \delta J' = 0 \), the infinite dimensional problem is simplified by projecting the true unknown solution onto a finite set of piecewise polynomials. The variations are then approximated using piecewise polynomials of appropriate order to generate a set of nonlinear equations.

Obviously, different choices for these polynomials yield different sets of equations. Previous work [2] used discontinuous, piecewise-constant polynomials for the main variables. This choice is prudent for problems which have discontinuities in the states or costates, which many optimal control problems do, so in this effort, discontinuous shape functions will be used.
Another advantage is that in using discontinuous polynomials, the simplest version of those (piecewise constants) allows the integrals in Eq. (29) to be done by inspection rather than by numerical quadrature. Even the simplest continuous polynomial approximations will require numerical quadrature over an element.

As Eq. (29) stands though, with the \( \dot{x} \) and \( \dot{\lambda} \) terms under the integral, using discontinuous shape functions results in jump-terms at each node point when the time interval is discretized, adding to the complexity (specifically, the dimension) of the problem. To avoid this, continuous approximations for \( \delta x \) and \( \delta \lambda \) are used, and the terms \( \dot{\lambda}^T \delta x \) and \( \delta \lambda^T \dot{x} \) are integrated by parts. Eq. (29) then becomes:

\[
\delta J' = \lambda^T \delta x|_{t_0}^{t_f} - \delta \lambda^T x|_{t_0}^{t_f} + \int_{t_0}^{t_f} \left[ H_u \delta u + H_x \delta x - \lambda^T \dot{x} + \delta \lambda^T f \right. \\
+ \left. \delta \lambda^T x + \delta \mu^T (g + K) + \mu^T \delta K \right] \, dt
\]

Next, the time interval is broken up into \( N \) not necessarily equal length time elements, \( \Delta t_i \), such that the time at each element boundary (or node) \( \hat{t}_i \) is calculated as:

\[
\hat{t}_i = \hat{t}_{i-1} + \Delta t_i \quad i = 2, \ldots, N + 1
\]

and the states and controls at these nodes are defined to be

\[
\dot{x}_i = x(\hat{t}_i) \quad i = 1, \ldots, N + 1
\]

\[
\dot{u}_i = u(\hat{t}_i) \quad i = 1, \ldots, N + 1
\]

The time within the \( i^{th} \) element, \( t_i \) is expressed as \( t_i = \hat{t}_{i-1} + \tau \Delta t_i \) where \( 0 \leq \tau \leq 1 \) and

\[
\tau = \frac{t_i - \hat{t}_{i-1}}{\Delta t_i}
\]

so that

\[
\frac{d(\cdot)}{dt} = \frac{1}{\Delta t_i} \frac{d(\cdot)}{d\tau} \equiv \frac{1}{\Delta t_i} (\cdot)'
\]
Substituting these relationships into Eq. (30) gives:

$$
\delta J' = \lambda^T \delta x_0' - \dot{x}^T \delta \lambda_0'
+ \sum_i^N \Delta t_i \int_0^1 \left[ (H_u)_i \delta u_i + (H_x)_i \delta x_i + \frac{\delta \lambda_i^T}{\Delta t_i} x_i - \frac{\delta x_i^T}{\Delta t_i} \lambda_i \right.
+ \frac{\delta \lambda_i^T}{\Delta t_i} f_i + \delta \mu_i^T (g_i + K_i) + \mu_i^T \delta K_i \bigg] d\tau
$$

Equation (37)

In this equation, a subscript \( i \) on a variable refers to the value of that variable within the \( i^{th} \) element, while the subscript on a function indicates the value of that function evaluated using variables within the \( i^{th} \) element.

### 3 Shape Functions

In this section, the specific shape functions alluded to in the previous section will be defined and used to solve Eq. (37). What will result is a set of nonlinear algebraic equations to be solved numerically.

Following the work of [8], to avoid jump-terms while also keeping the shape functions as simple as possible, \( C^0 \) shape functions for the variational quantities are implemented in each element. These are in terms of nodal values (\( \cdot \)) and polynomial coefficients on element interiors (\( \cdot \)).

$$
\delta \lambda_i = \delta \dot{\lambda}_i (1 - \tau) + \delta \dot{\lambda}_{i+1} \tau + \sum_{j=1}^{n_b-1} (1 - \tau) \tau \beta_j(\tau) \delta \tilde{\lambda}_{ij} \quad (38)
$$

$$
\delta x_i = \delta \dot{x}_i (1 - \tau) + \delta \dot{x}_{i+1} \tau + \sum_{j=1}^{n_b-1} (1 - \tau) \tau \beta_j(\tau) \delta \tilde{x}_{ij} \quad (39)
$$

Here \( n_b \) is the order of the shape function polynomial being used, with \( n_b = 1 \) the summation would be ignored, and this represents what is used in the \( h \)-version development. The functions \( \beta_j(\tau) \) are Jacobi polynomials of order \( (j - 1) \) as detailed in [9].

The time derivative of these expressions is of the form

$$
\frac{d\delta x_i}{d\tau} \equiv \delta x'_i = \Delta t_i \delta \dot{x}_i = -\delta \dot{x}_i + \delta \dot{x}_{i+1} + \sum_{j=1}^{n_b-1} \gamma_j(\tau) \delta \tilde{x}_{ij} \quad (40)
$$
where
\[ \gamma_j(\tau) \equiv [(1 - \tau)\beta_j(\tau)]' \]  
(41)

Similar expressions hold for the costates.

For the states and costates shape functions are chosen that are only continuous within the element:

\[
x_i = \begin{cases} 
\dot{x}_i & \tau = 0 \\
\sum_{j=1}^{n_x} \alpha_j(\tau) \bar{x}_{ij} & 0 < \tau < 1 \\
\dot{x}_{i+1} & \tau = 1
\end{cases}
\]
\[ \lambda_i = \begin{cases} 
\dot{\lambda}_i & \tau = 0 \\
\sum_{j=1}^{n_x} \alpha_j(\tau) \bar{\lambda}_{ij} & 0 < \tau < 1 \\
\dot{\lambda}_{i+1} & \tau = 1
\end{cases} \]  
(42)

where the functions \( \alpha_j(\tau) \) are polynomials of order \( j - 1 \) as derived in [8] to simplify the algebraic equations later on. Note that \( \dot{x}_i \) and \( \dot{\lambda}_i \) are discrete values, distinct from the shape functions within the element. The boundary conditions on the states and costates will be enforced weakly through these nodal values.

For the controls, control constraint Lagrange multipliers, slack variables, and their variations, again no time derivatives exist in Eq. (37), so the same shape functions are used:

\[ \delta u_i = \sum_{j=1}^{n_k} \alpha_j(\tau) \delta \bar{u}_{ij} \]  
(43)

\[ \delta \mu_i = \sum_{j=1}^{n_k} \alpha_j(\tau) \delta \bar{\mu}_{ij} \quad \delta k_i = \sum_{j=1}^{n_k} \alpha_j(\tau) \delta \bar{k}_{ij} \]  
(44)

\[ u_i = \sum_{j=1}^{n_k} \alpha_j(\tau) \bar{u}_{ij} \quad 0 < \tau < 1 \]  
(45)

\[ \mu_i = \sum_{j=1}^{n_k} \alpha_j(\tau) \bar{\mu}_{ij} \quad 0 < \tau < 1 \quad k_i = \sum_{j=1}^{n_k} \alpha_j(\tau) \bar{k}_{ij} \quad 0 < \tau < 1 \]  
(46)

No nodal values of the controls, slack variables, or control constraint Lagrange multipliers are needed in this problem formulation. They can be calculated as needed after the fact through the optimality condition and the control constraint equations of the previous section, Eqs. (26) – (28).
Substituting the shape functions from Eqs. (42) – (46) into the integral which remains in the cost function (37) gives

\[
\delta J' = \lambda_{N+1}^T \delta x_{N+1} - \lambda_i^T \delta x_i - \tilde{\lambda}_{N+1}^T \delta \lambda_{N+1} + \tilde{\lambda}_i^T \delta \lambda_i \\
+ \sum_i \Delta t_i \int_0^1 \left\{ (H_x)_i \left( \sum_{j=1}^{n_h} \alpha_j(\tau) \delta \bar{u}_{ij} \right) \\
+ (H_x)_i \left[ \delta \hat{x}_i(1-\tau) + \delta \hat{x}_{i+1} \tau + \sum_{j=1}^{n_i-1} \epsilon_j(\tau) \delta \bar{x}_{ij} \right] \\
+ f_i^T \left[ \delta \hat{\lambda}_i(1-\tau) + \delta \hat{\lambda}_{i+1} \tau + \sum_{j=1}^{n_i-1} \epsilon_j(\tau) \delta \bar{\lambda}_{ij} \right] \\
+ \left[ \sum_{j=1}^{n_h} \alpha_j(\tau) \delta \bar{\mu}_{ij} \right]^T \left[ g_i + \left( \sum_{j=1}^{n_h} \alpha_j(\tau) \bar{k}_{ij} \right)^2 \right] \\
+ 2 \left( \sum_{j=1}^{n_h} \alpha_j(\tau) \bar{\mu}_{ij} \right)^T \left( \sum_{j=1}^{n_h} \alpha_j(\tau) \bar{k}_{ij} \right) \sum_{j=1}^{n_h} \alpha_j(\tau) \delta \bar{k}_{ij} \right\} d\tau \\
+ \sum_i \int_0^1 \left[ -\delta \hat{\lambda}_i + \delta \hat{\lambda}_{i+1} + \sum_{j=1}^{n_i-1} \gamma_j(\tau) \delta \bar{\lambda}_{ij} \right]^T \sum_{j=1}^{n_i} \alpha_j(\tau) \bar{x}_{ij} d\tau \\
- \sum_i \int_0^1 \left[ -\delta \hat{x}_i + \delta \hat{x}_{i+1} + \sum_{j=1}^{n_i-1} \gamma_j(\tau) \delta \bar{x}_{ij} \right]^T \sum_{j=1}^{n_i} \alpha_j(\tau) \bar{\lambda}_{ij} d\tau
\]

where

\[
\epsilon_j(\tau) \equiv (1-\tau) \tau \beta_j(\tau)
\]

Note that the only nodal values of the states and costates that appear here are at the endpoints of the time interval. The other internal nodal values can be generated from the state equations once the other nodal and interior values have been solved for.
Because the polynomials $\alpha(\tau)$ and $\gamma(\tau)$ were chosen to be orthogonal, Eq. (47) reduces to

$$
\delta J' = \hat{\lambda}_{N+1}^T \delta x_{N+1} - \hat{\lambda}_1^T \delta x_1 - \hat{x}_{N+1}^T \delta \lambda_{N+1} + \hat{x}_1^T \delta \lambda_1 \\
+ \sum_{i} \Delta t_i \int_0^1 \left\{ (H_u)_i \left( \sum_{j=1}^{n_b} \alpha_j(\tau) \delta \bar{u}_{ij} \right) \\
+ \delta \hat{x}_i^T (H_x)_i (1 - \tau) + \delta \hat{x}_{i+1}^T (H_x)_i \tau + \delta \hat{\lambda}_i^T f_i (1 - \tau) + \delta \hat{\lambda}_{i+1}^T f_i \tau \\
+ \sum_{j=1}^{n_b-1} \left[ \epsilon_j(\tau) \delta \hat{x}_i^T (H_x)_i + \epsilon_j(\tau) \delta \hat{\lambda}_i^T f_i \right] \\
+ \left( \sum_{j=1}^{n_b} \alpha_j(\tau) \delta \bar{u}_{ij} \right)^T \left[ g_i + \left( \sum_{j=1}^{n_b} \alpha_j(\tau) \bar{k}_{ij} \right)^2 \right] \right\} \left( \sum_{j=1}^{n_b} \alpha_j(\tau) \bar{k}_{ij} \right) d\tau \\
+ \sum_{i} \left( -\delta \hat{\lambda}_i^T \bar{x}_{i,1} + \delta \hat{\lambda}_{i+1}^T \bar{x}_{i,1} - \sum_{j=2}^{n_b} \delta \hat{\lambda}_i^T \bar{x}_{ij} \right) \\
+ \sum_{i} \left( \delta \hat{x}_i^T \bar{\lambda}_{i,1} - \delta \hat{x}_{i+1}^T \bar{\lambda}_{i,1} + \sum_{j=2}^{n_b} \delta \hat{x}_{i,j-1} \bar{\lambda}_{ij} \right)
$$

(49)
Arranging terms by their variational coefficient gives

\[
\delta J' = \delta \lambda_1^T \left[ -\dot{x}_1 + \ddot{x}_{1,1} - \Delta t_1 \int_0^1 (1 - \tau) f_1 \, d\tau \right] \\
+ \sum_{i=2}^{N} \delta \lambda_i^T \left[ -\dot{x}_{i-1,1} - \Delta t_{i-1} \int_0^1 \tau f_{i-1} \, d\tau \right. \\
\left. + \ddot{x}_{i,1} - \Delta t_i \int_0^1 (1 - \tau) f_i \, d\tau \right] \\
+ \sum_{i=1}^{N} \sum_{j=1}^{n_i-1} \delta \lambda_{ij}^T \left[ -\dot{x}_{i,j+1} + \Delta t_i \int_0^1 \epsilon_j(\tau) f_i \, d\tau \right] \\
+ \delta \lambda_{N+1}^T \left[ -\dot{x}_{N,1} - \Delta t_N \int_0^1 \tau f_N \, d\tau + \dot{x}_{N+1} \right] \\
+ \delta \dot{\lambda}_1^T \left[ \dot{\lambda}_1 - \ddot{\lambda}_{1,1} - \Delta t_1 \int_0^1 (1 - \tau)(H_x)_1 \, d\tau \right] \\
+ \sum_{i=2}^{N} \delta \dot{\lambda}_i^T \left[ -\ddot{\lambda}_{i,1} - \Delta t_i \int_0^1 (1 - \tau)(H_x)_i \, d\tau \right. \\
\left. + \ddot{\lambda}_{i-1,1} - \Delta t_{i-1} \int_0^1 \tau(H_x)_{i-1} \, d\tau \right] \\
+ \sum_{i=1}^{N} \sum_{j=1}^{n_i-1} \delta \dot{\lambda}_{ij} \left[ \dot{\lambda}_{i,j+1} + \Delta t_i \int_0^1 \epsilon_j(\tau)(H_x)_i \, d\tau \right] \\
+ \delta \dot{\lambda}_{N+1} \left[ -\ddot{\lambda}_{N+1} + \ddot{\lambda}_{N,1} - \Delta t_N \int_0^1 \tau(H_x)_N \, d\tau \right] \\
+ \sum_{i=1}^{N} \sum_{j=1}^{n_i-1} \delta \ddot{\lambda}_{ij} \int_0^1 (H_u)_i \alpha_j(\tau) \, d\tau \\
+ \sum_{i=1}^{N} \sum_{j=1}^{n_i-1} \delta \ddot{\lambda}_{ij} \int_0^1 \alpha_j(\tau) \left[ g_i + \left( \sum_{j=1}^{n_i} \alpha_j(\tau) \tilde{k}_{ij} \right)^2 \right] \, d\tau \\
+ \sum_{i=1}^{N} \sum_{j=1}^{n_i-1} \delta \ddot{\lambda}_{ij} \int_0^1 2\alpha_j(\tau) \left[ \sum_{j=1}^{n_i} \alpha_j(\tau) \tilde{\mu}_{ij} \right]^T \left[ \sum_{j=1}^{n_i} \alpha_j(\tau) \tilde{k}_{ij} \right] \, d\tau
\]

(50)

The variational coefficients are independent and arbitrary, due to the weak form of the equations. Therefore, for the variation of \( J' \) to be zero, the expressions multiplied by the coefficients must be identically zero. In this way, the first-order necessary conditions for optimal control are approx-
imated. To supplement the equations derived from setting Eq. (50) equal to zero, the boundary conditions need to be enforced, as provided in the previous section:

\[
\lambda_0^T + \frac{\partial \phi}{\partial x_0} + \nu^T \frac{\partial \Psi}{\partial x_0} = 0 \tag{51}
\]

\[
\lambda_{N+1}^T - \frac{\partial \phi}{\partial x_f} - \nu^T \frac{\partial \Psi}{\partial x_0} = 0 \tag{52}
\]

\[
H(t_f) + \frac{\partial \phi}{\partial t_f} + \nu^T \frac{\partial \Psi}{\partial t_f} = 0 \tag{53}
\]

\[
\Psi(\dot{x}_0, \dot{x}_{N+1}, t_f) = 0 \tag{54}
\]

Thus, the two-point boundary value problem (20) – (28) is approximated by the set of nonlinear algebraic equations (50) – (54). When a problem has multiple phases (whether by discontinuity of the states or the differential equations, or both), the equations are similar and can also be handled by the code. The differences are that the user has to specify extra boundary conditions, including whatever will trigger the discontinuity, and then the code handles the corresponding jump conditions for the costates and Hamiltonian automatically.

In the following sections, how these equations are implemented will be explained, and example problems will be shown which demonstrate the use and performance of \(hp\)-version finite elements. Following that will be a discussion of how to improve the solution adaptively by gauging the error in each interval, followed by some preliminary results using this refinement technique.

4 Implementation and Results

A numerical solution which makes the expressions from Eq. (50) equal to zero is found using a restricted-step Newton-Raphson method, implemented in a FORTRAN code. A restricted-step Newton-Raphson method differs from a standard Newton-Raphson method in that if the full Newton step yields a higher value of the objective function than the starting point, the step size is halved repeatedly until an improving step is achieved.

The determination of the Newton step requires the solution of a linear system of equations. The Jacobian of these equations can be very large as
shown in Figure 1, which is for the problem defined in section 4.1. The
equations for the most part only depend on neighboring elements, so the
Jacobian ends up being quite sparse, as seen in Figure 2 for the same problem.
That feature is exploited by the use of sparse linear systems solvers from the
Harwell subroutine library [11]. These routines use a special sparse version
of Gaussian elimination with partial pivoting, where information is retained
between calls to the routine, so subsequent linear systems to be solved using
the same Jacobian structure (as is common in determining different Newton
steps) or simply different right-hand sides (as is common in repeatedly step
halving) take considerably less time.

The symbolic-manipulation package MACSYMA developed by Symbolics
[12] is used to generate analytical partial derivatives of the system equations,
boundary conditions, and control constraints for use in the Jacobian. MAC-
SYMA is also used to generate the $\alpha(\tau)$ and $\epsilon(\tau)$ polynomials from Eq. (50),
which come from a recursion formula involving derivatives and integrals of
polynomials, as developed in [8]. The user specifies the order of the poly-
nomials, and MACSYMA generates the necessary expressions, but the user
does have to supply initial guesses at the polynomial coefficients.

The integrals in the equations are approximated using Gauss-Legendre
integration [10], with the user selecting the number of Gauss points, which
at this time is constant for all of the integrations. No fewer Gauss quadrature
points should be chosen than the order of the shape functions being used plus
one. This is because the zeros of the shape functions correspond to zeros of
polynomials used in Gauss-Legendre integration. More Gauss points may be
used, and that has shown to be more accurate in some problems, but not
all, nor has the value of this extra computational effort been established. In
Refs. [6] and [7], it is established that the minimum error (or most accurate
stresses) in finite element approximations occur at the Gauss points.

4.1 Unconstrained problem

In perhaps the simplest representation of an aerospace problem with non-
linear system dynamics, this first problem involves the maximum velocity
transfer to a particle of mass $m$ to a specified horizontal flight path in a fixed
time (see [5], pg. 59). The mass is acted on by a force of constant magnitude
$ma$ and variable heading $\beta(t)$. The states for this problem are position of
the particle $x$ (horizontal) and $y$ (vertical) and the corresponding velocity
components \( u \) and \( v \). The differential equations for this system are then:

\[
\begin{align*}
\dot{x} &= u \\
\dot{y} &= v \\
\dot{u} &= a \cos \beta \\
\dot{v} &= a \sin \beta
\end{align*}
\]  

(55)

with initial conditions corresponding to a zero velocity at the origin and with terminal constraints that the particle is in horizontal flight at a given height (assumed here to be 1), yielding a boundary condition vector,

\[
\begin{align*}
\Psi_1 &= u(0) \\
\Psi_2 &= v(0) \\
\Psi_3 &= x(0) \\
\Psi_4 &= y(0) \\
\Psi_5 &= y(t_f) - 1 \\
\Psi_6 &= v(t_f)
\end{align*}
\]  

(56)

The final horizontal position \( x(t_f) \) is unspecified, and the final horizontal velocity \( u(t_f) \) is to be maximized. The cost function is then

\[
J = u(t_f)
\]  

(57)

with the boundary conditions enforced by setting \( \Psi = 0 \).

Reference [5] gives the analytic solution in terms of the initial force heading angle, the final time, and the final altitude in unspecified units. These values were chosen to be 75°, 1, and 1 respectively.

The code was run for this problem for a variety of combinations of finite-element parameters. Figures 3 and 4 show the square root of the integral of the squares (i.e. the two-norm) of the relative error time histories for the states and controls as a function of the CPU time involved in calculating solutions. The data points correspond to the 2-, 4-, 8-, 16-, and 32-element solutions. The initial guess for each case was extrapolated from the converged solution for the case of half as many elements (or for one fewer shape function coefficient if only 2 elements). All of the specified boundary conditions were met to within \( 10^{-12} \) for the case when all the elements are evenly
spaced. Figures 5 and 6 show the same error data vs. the number of free parameters in the problem, which equals the dimension of the Jacobian.

Once an initial solution was obtained for a low-order shape function and a small number of elements, the code converged easily as the parameters were changed. Not surprisingly, the overall errors reduced as the order of shape functions increased and as the number of elements increased.

What is surprising is the particularly good performance of the first-order shape function through the higher error regions. It isn't until errors get below $10^{-6}$ in the states that the higher shape function orders get significantly better, with each of the other shape function orders being best for each subsequent 2 orders of magnitude error. That pattern gets pushed even further down and wider with the control variables. First-order is best all the way down to $10^{-8}$ with each subsequent higher-order shape function being best seemingly for another 3 or 4 orders of magnitude, until the code runs into problems with round-off error.

Later these results will be compared with those when the mesh is refined using an adaptive scheme. For now these results show the increase in accuracy that is possible using higher-order shape functions in solving optimal control problems.

### 4.2 Example with control constraint

Next a problem with a control constraint was studied, again from Ref. [5]. The problem is to minimize the cost function

$$J = \frac{1}{2} x(t_f)^2 + \frac{1}{2} \int_0^{t_f} u^2 \, dt$$

(58)

where $x$ and $u$ are scalars. The system dynamics are governed by

$$\dot{x} = h(t)u$$

(59)

for some function $h(t)$, subject to two control inequality constraints,

$$g_1 = u - 1 \leq 0,$$

and

$$g_2 = -(u + 1) \leq 0$$

(60)

An exact solution is available [2] if the final time is chosen to be 10, the initial condition is a given constant, and

$$h(t) = 1 + t - \frac{3}{17} t^2$$

(61)
The exact value for the state at the final time is \( x(t_f) = -17/39 \) and the optimal control is

\[
u(t) = \begin{cases} 
-x(t_f)h(t) & 0 \leq t \leq 2 \\
1 & 2 \leq t \leq 11/3 \\
-x(t_f)h(t) & 11/3 \leq t \leq 8 \\
-1 & 8 \leq t \leq 10 
\end{cases}
\] (62)

This problem is not as well-behaved numerically as the previous problem, and so solutions are not readily available for the sequences of distributions of elements. Errors do reduce significantly as the mesh is refined and the shape function orders are increased, but showing those results will be deferred to Section 5 when they will be compared to results using adaptive error-reduction techniques.

The penalty of increased CPU time associated with using higher-order shape functions could be serious enough to thwart using this methodology to solve problems in real time. However, many of the element interior unknowns can be eliminated at the element level by solving a small set of nonlinear algebraic equations in which the nodal values are taken as given, splitting the problem (and hence the Jacobian) into outer and inner loops for the nodal and interior values to an element respectively. The scheme may then turn out to be especially powerful in a parallel computing environment since a different processor could be assigned to each element. The number of processors, strictly speaking, would not be required to be any larger than the number of sub-regions which are free of discontinuities.

Another way to bring down the computational effort while still reducing and smoothing errors is by more prudent selection of the element mesh and shape function orders. Starting with a coarse discretization, the optimal distribution of elements and higher-order shape functions could be found that minimizes the error for a given number of parameters. In the next section some indicators available to gauge error for use in such a process will be examined.
5 Implementation and Results with Adaptivity

To develop a means of adaptively adjusting mesh and shape function parameters, subroutines were developed to split a given element or raise the shape function order if a certain error criterion was met. Initial guesses are then determined for any new parameters, and then the new (larger) set of equations is solved again using the Newton-Raphson iteration.

This section describes two possible error indicators, one involving the calculation of a single function and one more elaborate method involving the solution to an adjoint problem. The simpler one has been implemented, and the corresponding results will be described first. The more elaborate error indicator is still in development, so the development of it will be more sketchy.

5.1 Hamiltonian as error indicator

The Hamiltonian is a logical choice for an error criterion since it is a first integral of the two point boundary value problem derived from the first-order necessary conditions for optimal control. In the problems under consideration, it contains all of the controls and costates, at least some of the states, and all of the state rates. Thus looking at how the Hamiltonian varies from its optimal value can be a valuable gauge for the magnitude of the error in the variables in the problem.

The development in appendix A shows how the optimal control problem can be cast such that the Hamiltonian at the optimal solution is always zero. With the Hamiltonian having been transformed in this way, the difference between \( H(t) \) and the optimal (namely zero) can now easily be measured. What remains to be seen is how good of an indicator of error (either pointwise or integrated) in the solution that the Hamiltonian is.

The two criteria for parameter adjustment that have been looked at most closely are the deviation from zero of the Hamiltonian at any given node and the jump that the Hamiltonian makes between two adjacent nodes.

The two problems studied were the same as in Section 4. The first set of results is for the unconstrained problem in Section 4.1. The problem was first solved for 2 elements. At this point, the Hamiltonian was a non-zero constant
across both elements, so both elements were split. The resulting Hamiltonian and relative error distributions are shown in Figure 7. The "Total Error" is the two-norm of the relative errors in all the states, costates, and control; the "x-u error" includes only errors in the states and controls; and the "u error" includes only the error in the control. Relative error was unavailable at the endpoints for the states since they were constrained to be zero. Similarly, the optimal value of the control at the midpoint of the time interval is zero, so that data point will also be missing in each of these plots.

As can be seen, the Hamiltonian jumps the most in the two central elements. Using a criterion that elements should be split if the Hamiltonian jump across them was 10% of the maximum jump along the trajectory, the code split these two central elements, resulting in the smoother Hamiltonian distribution in Figure 8.

At this point, all the Hamiltonian jumps were about the same, so all the elements were split, resulting in the 12-element solution shown in Figure 9, where again some peaks developed, which were smoothed out in the 16-element solution of Figure 10. This process can continue until point the Hamiltonian and the overall error in all variables is below $10^{-10}$.

In each case, the Hamiltonian peaks near the middle elements picked up the peak error in the control in that region. Meanwhile the peak in the overall state error (including the two states which did not appear in the Hamiltonian) seemed to correspond to a secondary peak in the Hamiltonian. The total error was dominated by the error in one of the costates which was a constant. Even so, the total error was in the same order of magnitude as the error in the Hamiltonian.

In the second example with the control constraints from Section 4.2, jumps in the Hamiltonian seem to better highlight regions where the mesh should be refined. Starting with 7 elements, so as not to have a node fall upon an optimal entrance or exit point for a control constraint, the Hamiltonian is constant within each phase of the problem, whether on or off the control constraint, as shown in Figure 11. The code senses the jumps in the Hamiltonian near each of the switching points and put more nodes there, resulting in the 16-element solution in Figure 12. To compare, Figure 13 shows how the error and nodes would be distributed with 17 uniformly spaced elements. The errors are an order of magnitude higher. Figure 14 shows the results of a couple of optimizing runs done on the results from Figure 13.

In all the plots for this problem, the control error is necessarily zero.
along constrained arcs. Similarly, the Hamiltonian error was always zero in the last constrained arc since the Hamiltonian is actually enforced to be zero at the final time. With a constant control in that region, the equations can be integrated exactly. The plots show that to bring the error down in the early regions, the switching points have to be nailed down precisely, which is exactly what happens when optimization is based on Hamiltonian jumps. Unfortunately, the code can only make one set of element splits before resolving the problem. This bisection technique would take a long time to reach the switching points exactly. When state constraint capability is added to the code, the endpoints of a control constraint arc can then become variables.

Thus, element splitting based on Hamiltonian jumps seems to do a reasonable job of smoothing and reducing error, and the Hamiltonian itself seems to be a reasonable measure of errors within the elements, but the results are not consistent enough. The Hamiltonian, while easy to calculate from given information and a reasonable gauge for error, lacks the information about how well the differential equations are being approximated. Also lacking is a sense of how finely to discretize a given element, as demonstrated in the control constraint problem. The next a posteriori error estimator to be investigated is being designed to gauge the residual errors in meeting the equations within each element.

5.2 Residual error technique

As mentioned in section 2.2, the variational equations can be interpreted as residuals multiplied by test functions, the definition a Galerkin method. Since Galerkin methods minimize the two-norms of the residual errors, formulating the problem in this framework will help greatly in determining an error estimator. In this section, the preliminary work done in analyzing this system to find adequate residual error estimators will be presented.

As a starting point, the analysis will not treat complete optimal control problems, but rather two point boundary value problems (TPBVP) without controls, using the work of Estep et al. [3] in initial value problems as a basis. The extra constraints of the optimality condition and any control constraints will be added later. Likewise, only time-varying linear systems have been examined thus far, and each state must have exactly one given boundary condition. Dr. Estep’s continued collaboration in developing this error estimator is acknowledged and appreciated.
The states are first split depending on at which endpoint the boundary condition is specified. For this case, the system being analyzed has the form

\[
\dot{y} = A(t)y \quad 0 \leq t < t_f \\
y_L(0) = y_0 \\
y_R(t_f) = y_f
\]

where \( y \in \mathbb{R}^n \) is partitioned as

\[
y = \begin{pmatrix} y_L \\ y_R \end{pmatrix}
\]

Using the continuous Galerkin finite element method, the object is to find the polynomial \( Y \in C^q \) such that:

\[
\int_0^{t_f} [\dot{Y} - A(t)Y, v] \, dt = 0 \quad \forall v \in D^{q-1} \\
Y_L(0) = y_0 \\
Y_R(t_f) = y_f
\]

where \( C^q \) is the set of all piecewise polynomials on \([0, t_f]\) of order \( q \) which are continuous across element boundaries. \( D^q \) is the set of all piecewise polynomials on \([0, t_f]\) of order \( q \) which are not continuous across element boundaries.

Subtracting Eqs. (63) and (65), and defining the error as \( e = Y - y \), one yields

\[
\int_0^{t_f} [\dot{e} - A(t)e, v] \, dt = 0 \quad \forall v \in D^{q-1} \\
e_L(0) = 0 \\
e_R(t_f) = 0
\]

where \( e \) is partitioned similarly to \( y \) as

\[
e = \begin{pmatrix} e_L \\ e_R \end{pmatrix}
\]

Integrating Eq. 66 by parts yields:

\[
\int_0^{t_f} [\dot{e} - A(t)e, v] \, dt = e(t_f)^T v(t_f) - e(0)^T v(0) + \int_0^{t_f} [e, -\dot{v} - A(t)^T v] \, dt
\]
where
\[ e(t_f)^T v(t_f) = e_L(t_f)^T v_L(t_f) \]
\[ e(0)^T v(0) = e_R(0)^T v_R(0) \]  

(69)

Now let \( v \) be the solution of
\[ -\dot{v} - A(t)^T v = e(t) \quad 0 \leq t < t_f \]
\[ v_L(t_f) = 0 \]
\[ v_R(0) = 0, \]  

(70)

reducing the right hand side of Eq. (68) extensively:
\[ \int_0^{t_f} \|e\|^2 \, dt = \int_0^{t_f} [\dot{e} - A(t)e, v] \, dt \]  

(71)

Adding in Eq. (63) gives:
\[ \int_0^{t_f} \|e\|^2 \, dt = \int_0^{t_f} [Y - A(t)Y, v] \, dt \]  

(72)

Defining \( \pi_q \) as a projection operator into the space of polynomials of order \( q \), Eq. (65) implies that \( \pi_{q-1}v \) is orthogonal to \( Y - A(t)Y \), and hence it can be added to the right hand side of the dot product:
\[ \int_0^{t_f} \|e\|^2 \, dt = \int_0^{t_f} [\dot{Y} - A(t)Y, v - \pi_{q-1}v] \, dt \]  

(73)

Now, \( \dot{Y} \in D^{q-1} \) so it is orthogonal to \( v - \pi_{q-1}v \), reducing the above to:
\[ \int_0^{t_f} \|e\|^2 \, dt = \int_0^{t_f} [-A(t)Y, v - \pi_{q-1}v] \, dt \]  

(74)

By a similar argument, any other function in \( D^{q-1} \) can be added on the left side of the dot product, so in order to make the term on the left small, add in the projection of \( A(t)Y \):
\[ \int_0^{t_f} \|e\|^2 \, dt = \int_0^{t_f} \{ \pi_{q-1} [-A(t)Y] - A(t)Y, -\pi_{q-1}v + v \} \, dt \]  

(75)

Using standard calculus integral inequalities plus the inequality that
\[ \int_0^{t_f} (v - \pi_{q-1}v) \, dt \leq \int_0^{t_f} \|\dot{v}\| \, dt, \]  

(76)
Eq. (75) becomes:

\[
\int_0^{t'} \| e \|^2 \, dt \leq \int_0^{t'} \| \dot{v} \| \, dt \cdot \max_{m \leq n} \left[ k_m \cdot \max_{t \in t_m} \| A(t)Y - \pi_{q-1}A(t)Y \| \right]
\]

\[
\equiv S \cdot \max_{m \leq n} (k_m \cdot R_m)
\]

(77)

where \( k_m \) is the length of the \( m \)th element, \( I_m \) (out of \( n \)), \( S \) is the sensitivity function, and \( R_m \) is the residual on the \( m \)th element. Note that different choices for inequalities to use result in different error estimators, which may happen to suit optimal control problems better. That will be examined as work continues.

With a target integral error of \( TOL \), the original finite element problem (65) is solved once, and the residuals are calculated. At the first iteration, the new time interval lengths \( k_m^{(2)} \) can be computed for each \( m \) as

\[
\frac{k_m^{(2)}}{k_m^{(1)}} = \frac{(TOL)^2}{S \cdot k_m^{(1)} \cdot R_m} \quad \forall m \in [1, n]
\]

(78)

with \( S \) assumed to be one. For subsequent iterations, the dual problem (70) is solved with the forcing function \( e(t) \) approximated as the difference between the approximate solutions from the previous two iterations. Then \( S \) is calculated and used in calculating the next mesh refinement. In this way the solution is refined, equidistributing the error.

This methodology has been implemented into FORTRAN and tested on two time-varying, linear systems, but the results are still preliminary. The next step will be to include nonlinear dynamics, but from previous work of Estep [3], the main difference will be that the Jacobian of the system dynamics will replace the \( A(t) \) matrix in the dual problem, retaining its linear nature, and indeed making it a reasonably trivial calculation compared to solving the huge system of nonlinear equations. Once that is working, the next step is to add scalar functions and constraints to the two-point boundary value problem, which should also be a straightforward extension.

6 Summary and Future Work

An updated version to GENCODE, developed by Hodges and Bless [1], (but without state constraints) has been developed to solve a variety of opti-
mal control problems using $hp$-version finite elements with adaptive mesh-parameter adjustment. Textbook problems with non-linear system dynamics and control constraints have already been solved for a variety of combinations of finite element parameters, including adaptively modifying the distribution of mesh parameters with the Hamiltonian as an error indicator. The Hamiltonian has been shown to be an inadequate error indicator in and of itself, necessitating more sophisticated measures if optimal mesh-parameter design is to be done. The first steps toward a residual error measure have been taken and look promising.

To that end, research will need to continue to be done concerning the techniques for error estimation and adaptive adjustment of mesh parameters. The residual error techniques will have to be further refined and tested, while analysis needs to be done to verify the experimental evidence already obtained regarding the use of the Hamiltonian as an error indicator.

The methodology to handle state constraints, a realistic part of many aerospace applications, will have to be developed and implemented. Once that capability is added, control constraints will be able to be handled with variable endpoints as zeroth order state constraints, if desired, greatly improving the accuracy in solving those kinds of problems. That way any adaptive routine will not have to iterate as much to determine the optimal switching points.

References


A Reformulating the Hamiltonian

In this section, we will develop the properties of the Hamiltonian and reformulate it to become a convenient measure of error in the system. The Hamiltonian will be studied for the general class of optimal control problems studied in section 2, but without control constraints as that simply complicates notation.
Borrowing notation from section 2, define the Hamiltonian in the standard way as
\[ H = L(x, u, t) + \lambda^T f(x, u, t) \] (79)

From the first order necessary conditions for optimality (see section 2), the condition on the Hamiltonian at the final time is determined from
\[ \left[ H(t_f) + \frac{\partial \phi}{\partial t_f} + \nu^T \frac{\partial \Psi}{\partial t_f} \right] dt_f = 0 \] (80)

If \( t_f \) is fixed, then \( dt_f = 0 \), and the Hamiltonian can remain free at the final time. Otherwise \( dt_f \) is a free variation and thus
\[ H(t_f) = -\frac{\partial \phi}{\partial t_f} - \nu^T \frac{\partial \Psi}{\partial t_f} \] (81)
at the optimal solution. In either case, no conditions are imposed on the Hamiltonian at any time other than the final time. Often \( t_f \) is not constrained or penalized. In this case, Eq. (81) reduces to
\[ H(t_f) = 0 \] (82)

In the case when \( t_f \) is a fixed number \( T \), the code is set up to assume that \( t_f \) is still a free variable. This way the same set of equations is solved by the code for any general problem. The user has to add the extra constraint
\[ \Psi_{p+1} = t_f - \tau \] (83)
and Eq. (81) becomes the dummy equation
\[ H(t_f) = \nu_{p+1}, \] (84)
where \( \nu_{p+1} \) is the associated extra dummy unknown. Whatever the code decides the optimal value of \( H \) should be at the final time from the other equations, \( \nu_{p+1} \) will be set to that.

To get an idea of the behavior of the Hamiltonian along the optimal trajectory, take the time derivative:
\[ \dot{H} = \frac{\partial H}{\partial x} \dot{x} + \frac{\partial H}{\partial \lambda} \dot{\lambda} + \frac{\partial H}{\partial u} \dot{u} + \frac{\partial H}{\partial t} \] (85)
For the optimal trajectory, the first two terms cancel, and the third is zero, reducing (85) to
\[ \dot{H} = \frac{\partial H}{\partial t} = H_t \]  
(86)
By definition, if the system is autonomous, \( H_t = 0 \), which implies that \( \dot{H} = 0 \), and thus
\[ H(t) = H(t_f), \]  
(87)
though that's not an enforced constraint.

If \( H(t_f) \) were a known quantity, this could be used as an independent check on the error in an obtained solution. Thus we would like to make \( H(t_f) \) equal to a known constant and make \( H(t) \) equal to \( H(t_f) \) for a general problem formulation.

One solution to both problems is to make time a state in the problem, i.e.,
\[ x_{n+1} = t \]  
(88)
which changes the cost function to:
\[ J = \phi[x(t_f)] + \int_0^{t_f} L(x, u) \, dt, \]  
(89)
where \( x \) here includes both the old states plus time, with constraints
\[ \dot{x}_i = f_i(x, u) \quad i \in 1...n \]  
(90)
\[ \dot{x}_{n+1} = 1 \]  
(91)
\[ \Psi_i[x(t_0), x(t_f)] = 0 \quad i \in 1...p \]  
(92)
\[ \Psi_{p+1} = x_{n+1}(t_0) = 0 \]  
(93)

Now if \( t_f \) is a fixed constant \( \tau \), we add:
\[ \Psi_{p+2} = x_{n+1}(t_f) - \tau = 0 \]  
(94)
In any case, \( \Psi \) is not dependent on \( t_f \) explicitly, so we always enforce:
\[ H(t_f) = 0. \]  
(95)
Similarly, since explicit dependence on time has been removed from the problem,
\[ \frac{\partial H}{\partial t} = 0, \]  
(96)
even for a non-autonomous problem, so \( H(t) = H(t_f) = 0 \) over the whole trajectory.

The new state equation for the time state is enforced with a new costate. This costate is not as meaningless as one might assume at first glance. To find an interpretation for it, we first define a pseudo-Hamiltonian, \( \bar{H} \).

\[
\bar{H} = L(x, u) + \sum_{i=1}^{n} \lambda_i f_i(x, u).
\]  

(97)

Since along the optimal trajectory the states, costates, and control will be the same in either formulation and the new state simply substitutes for the running time, \( \bar{H}(t) \) is the same as the Hamiltonian before the time state was added. The time derivative of the new Hamiltonian is then

\[
\dot{\bar{H}} = \dot{H} + \dot{\lambda}_{n+1} = 0
\]  

(98)

and the enforced boundary condition is:

\[
H(t_f) = \bar{H}(t_f) + \lambda_{n+1}(t_f) = 0
\]  

(99)

These two equations imply that

\[
\dot{\lambda}_{n+1} = -\dot{\bar{H}}
\]  

(100)

and

\[
\lambda_{n+1}(t_f) = -\bar{H}(t_f)
\]  

(101)

which indicates that the extra costate obtained by adding a time state is in fact the old Hamiltonian. This relationship boils down to something very simple if the problem is autonomous.

From the necessary conditions for optimality, the costate equation and boundary conditions for the new costate are

\[
\begin{align*}
\lambda_{n+1}(t_0) &= \nu_{p+1} \\
\lambda_{n+1}(t_f) &= \nu_{p+2} \\
&= 0 & t_f \text{ fixed} \\
&= 0 & t_f \text{ free}
\end{align*}
\]

For an autonomous problem,

\[
\frac{\partial H}{\partial x_{n+1}} = 0.
\]  

(102)
This implies both that

\[ \dot{\lambda}_{n+1} = 0 \Rightarrow \lambda_{n+1}(t) = \lambda_{n+1}(t_f) \]  

(103)

and that \( \lambda_{n+1}(t_f) \) will not appear in the other costate equations or the optimality equation, \( H_u = 0 \) (and of course not the system dynamics.) Thus the only place \( \lambda_{n+1}(t_f) \) enters the problem is by balancing \( \tilde{H}(t_f) \) in Eq. (101). Comparing this to Eqs. (84) and (82) it is clear that for autonomous problems, free- or fixed-time, that the additional costate associated with the time state simply acts as a dummy Lagrange multiplier. So in adding the time state, effectively the Hamiltonian absorbs the right hand side of its terminal boundary condition so that the new Hamiltonian can always be zero. This is essentially what happens in non-autonomous problems except that it is not a constant being absorbed.
Figure 1. Dimension of Jacobian for Bryson and Ho problem, Sect. 2.4

Figure 2. Sparsity of Jacobian for Bryson and Ho problem, Sect. 2.4
Figure 3. 2-norm of relative error in states for Bryson and Ho problem, Sect. 2.4

Figure 4. 2-norm of relative error in control for Bryson and Ho problem, Sect. 2.4
Figure 5. 2-norm of relative error of states for Bryson and Ho problem, Sect. 2.4

Figure 6. 2-norm of relative error of control for Bryson and Ho problem, Sect. 2.4
Figure 7. Error Analysis for Bryson and Ho problem Sect. 2.4, 4 elements

Figure 8. Error Analysis for Bryson and Ho problem, Sect 2.4, 6 elements
Figure 9. Error Analysis for Bryson and Ho problem, Sect 2.4, 12 elements

Figure 10. Error Analysis for Bryson and Ho problem, Sect 2.4, 16 elements
Figure 11. Error Analysis for Bryson and Ho problem, Pg 109, 7 elements

Figure 12. Error Analysis for Bryson and Ho problem Pg. 109, with 16 elements, optimized from 7
Figure 3. Error Analysis for Bryson and Ho problem, pg 109, with 17 elements, not optimized.

Figure 4. Error Analysis for Bryson and Ho problem, pg 109, with 24 elements, optimized from 17.