RIOTS_95 – a MATLAB Toolbox for Solving General Optimal Control Problems And Its Applications to Chemical Processes

ADAM L. SCHWARTZ
LGC Wireless Inc.
2540 Junction Avenue, San Jose, CA 95134-1902, USA
E-mail: `adam@lgcwireless.com`
URL: `http://www.accesscom.com/~adam`

YANGQUAN CHEN
Center for Self-Organizing and Intelligent Systems (CSOIS),
Dept. of Electrical and Computer Engineering, 4160 Old Main Hill,
Utah State University, Logan, UT 84322-4160, USA.
E-mail: `yqchen@ieee.org`
URL: `http://www.crosswinds.net/~yqchen`

ABSTRACT

`RIOTS_95` is a group of programs and utilities, written mostly in C, Fortran and M-file scripts and designed as a toolbox for Matlab [1] , that provides an interactive environment for solving a very broad class of optimal control problems. `RIOTS_95` comes pre-compiled for use with the `Windows 95/98/2000` or `Windows NT` operating systems. This Chapter describes the use and operation of `RIOTS_95` together with a demonstrative example in solving optimal control problems appeared in chemical enigneering.

The numerical methods used by `RIOTS_95` are supported by the theory in the PhD. Dissertations of Dr. Adam L. Schwartz [1], which uses the approach of consistent approximations as defined by Polak [2]. In this approach, a solution is obtained as an accumulation point of the solutions to a sequence of discrete-time optimal control problems that are, in a

[1]Matlab is a registered trademark of Mathworks, Inc. (`http://www.mathworks.com/`)

specific sense, consistent approximations to the original continuous-time, optimal control problem. The discrete-time optimal control problems are constructed by discretizing the system dynamics with one of four fixed step-size Runge-Kutta integration methods and by representing the controls as finite-dimensional B-splines. Note that RIOTS_95 also includes a variable step-size integration routine and a discrete-time solver. The integration proceeds on a (possibly non-uniform) mesh that specifies the spline breakpoints. The solution obtained for one such discretized problem can be used to select a new integration mesh upon which the optimal control problem can be re-discretized to produce a new discrete-time problem that more accurately approximates the original problem. In practice, only a few such re-discretizations need to be performed to achieve an acceptable solution.

RIOTS_95 provides three different programs that perform the discretization and solve the finite-dimensional discrete-time problem. The appropriate choice of optimization program depends on the type of problem being solved as well as the number of points in the integration mesh. In addition to these optimization programs, RIOTS_95 also includes other utility programs that are used to refine the discretization mesh, to compute estimates of integration errors, to compute estimates for the error between the numerically obtained solution and the optimal control and to deal with oscillations that arise in the numerical solution of singular optimal control problems.

Key words: Optimal control problem solver, MATLAB Toolbox, consistent approximation, B-splines, Runge-Kutta integration, fed-batch chemical process.

1. INTRODUCTION

Optimization is a routine work in engineering practice. In general, optimizatior tasks can be classified into two categories static optimization tasks and dynamic ones Dynamic optimization has not been developed as "matured" as static optimization More often, dynamic optimization is referrec to as "optimal control problems (OCPs)" Numerical methods for solving optimal control problems have not reached the stage that, say, methods for solving differentia equations have reached. Solving an optima control problem can, depending on the difficulty of the problem, require significant use involvement in the solution process. This sometimes requires the user to understand the theory of optimal control, optimizatior and/or numerical approximation methods Among the existing software packages for numerically solving dynamic optimization o optimal control problems, such as SOCS [3] RIOTS_95 [4], DIRCOL [5], or MISER3 [6], nc single program can solve all sort of problems For example, a typical challenge is the optimal drug scheduling for cancer chemotherapy problem [7]. For a recent survey on solving OCPs, refer to [8].

Nevertherless, theoretically speaking, it is well known that dynamic programming (DP) of Bellman can solve all types of OCPs [9, 10, 11]. It is a powerful method for solving optimization problems by breaking up a complex optimization problem into a number of simpler problems. The solution o the simpler problems leads to the solutior of the original problem. This is an attractive feature with guaranteed global optimum Although the computers are now more anc more powerful, use of dynamic programming idea to solving OCPs is still not popular today due to the inherent drawbacks of DP method: curse of dimensionality, problems in the expanding grids and problems in the interpolations etc., that limit its use to only solving problems of very low dimension. To overcome these limitations, the author Rein Luus suggested using DP in an iterative fashion, first introduced in [12] in terms of "Iterative Dynamic Programming (IDP)". After 10 years development, IDP has evolved as a standard macro numerical procedure to cost-effectively solve various hard OCPs as evidenced in the book [13] with a published book review [14].

In view of the above discussions, while solving OCPs is still an "*art*", it is important to have a good software environment for the user to play with. Ideally, no compiling/linking is requred. To this regard, `RIOTS_95` is the most favorable due to the MATLAB platform. `RIOTS_95` is designed as a MATLAB toolbox written mostly in C, Fortran and M-file scripts. It provides an interactive environment for solving a very broad class of optimal control problems. `RIOTS_95` comes pre-compiled for use with the `Windows 95/98/2000` or `Windows NT` operating systems. The user-OCPs can be prepared purely in M-files and no compiler is needed to solve the OCPs. To speed up the OCP solving process, there are two ways go: by using the MATLAB Compiler or by providing the user-OCP in C which is to be compiled by a C-compiler and then linked with some pre-built linking libraries (currently, only WATCOM C compiler is supported). This Chapter describes the use and operation of `RIOTS_95` together with a demonstrative example in solving optimal control problems appeared in chemical enigneering. We will demonstrate both M-file interface and the CMEX/DLL interface with detailed user-supplied codes listed in the Appendices A and B.

The major objective of this Chapter is to demonstrate that `RIOTS_95` is an efficient and easy-to-use platform for solving OCPs through two examples. After introducing the major features of `RIOTS_95` in Sec. 2, the optimal control problems which can be handled by `RIOTS_95` are described in Sec. 3. Section 4 presents the first example for the textbook bang-bang OCP (M-files listed in Appendix A). Section 4 presents the second example for a relatively tough OCP of a bed-batch fermentor (C-files listed in Appendix B). Finally, concluding remarks and future work are discussed in Sec. 6.

2. MAJOR FEATURES OF `RIOTS_95`

The name RIOTS stands for "**R**ecursive [2] **I**ntegration **O**ptimal **T**rajectory **S**olver." This name highlights the fact that the function values and gradients needed to find the optimal solutions are computed by forward and backward integration of certain differential equations.

This chapter describes the implementation of a Matlab Toolbox called `RIOTS_95` for solving optimal control problems.

`RIOTS_95` [3] is a collection of programs that are callable from the mathematical simulation program Matlab for Windows. Most of these programs are written in either C,

[2] *Iterative* is more accurate but would not lead to a nice acronym.

[3] It is runnable in Matlab version 4.0, 4.2c, 5.x, or 6.0. Spline Toolbox is required. Evaluation versions are downloadable from `http://www.accesscom.com/~adam/RIOTS/`

Fortran (and linked into Matlab using Matlab's MEX/DLL facility) or Matlab's M-script language. All of Matlab's functionality, including command line execution and data entry and data plotting, are available to the user. The following is a list of some of the main features of `RIOTS_95`.

- Solves a very large class of finite-time optimal controls problems that includes: trajectory and endpoint constraints, control bounds, variable initial conditions (free final time problems), and problems with integral and/or endpoint cost functions.

- System functions can be supplied by the user as either object code or M-files.

- System dynamics can be integrated with fixed step-size Runge-Kutta integration, a discrete-time solver or a variable step-size method. The software automatically computes gradients for all functions with respect to the controls and any free initial conditions. These gradients are computed exactly for the fixed step-size routines.

- The controls are represented as splines. This allows for a high degree of function approximation accuracy without requiring a large number of control parameters.

- The optimization routines use a coordinate transformation that creates an orthonormal basis for the spline subspace of controls. The use of an orthogonal basis can results in a significant reduction in the number of iterations required to solve a problem and an increase in the solution accuracy. It also makes the termination tests independent of the discretization level.

- There are three main optimization routines, each suited for different levels of generality of the optimal control problem. The most general is based on sequential quadratic programming methods. The most restrictive, but most efficient for large discretization levels, is based on the projected descent method. A third algorithm uses the projected descent method in conjunction with an augmented Lagrangian formulation.

- There are programs that provide estimates of the integration error for the fixed step-size Runge-Kutta methods and estimates of the error of the numerically obtained optimal control.

- The main optimization routine includes a special feature for dealing with singular optimal control problems.

- The algorithms are all founded on rigorous convergence theory.

In addition to being able to accurately and efficiently solve a broad class of optimal control problems, `RIOTS_95` is designed in a modular, toolbox fashion that allows the user to experiment with the optimal control algorithms and construct new algorithms. The programs `outer` and `aug_lagrng`, described in detail in [4], are examples of this toolbox approach to constructing algorithms.

`RIOTS_95` is a collection of several different programs (including a program which is, itself, called `riots`) that fall into roughly three categories: integration/simulation routines, optimization routines, and utility programs. Of these programs, the ones available to the user are listed in the following table, Several of the programs in `RIOTS_95` require functions that are available in the

Table 1: Three categories of programs in RIOTS_95

Simulation Routines	Optimization Routines	Utility Programs
`simulate`	`riots`	`control_error`
`check_deriv`	`pdmin`	`distribute`
`check_grad`	`aug_lagrng`	`est_error`
`eval_fnc`	`outer`	`make_spline`
		`transform`

Matlab Spline toolbox. In addition to these programs, the user must also supply a set of routines that describe the optimal control problem which must be solved. Several example optimal control problems come supplied with RIOTS_95. Finally, there is a Matlab script called RIOTSdem which provides a demonstration of some of the main features of RIOTS_95.

Limitations: This is the first version of RIOTS_95. As it stands, there are a few significant limitations on the type of problems which can be solved by RIOTS_95:

- Problems with inequality state constraints that require a very high level of discretization cannot be solved by RIOTS_95. Also, the computation of gradients for trajectory constraints is not handled as efficiently as it could be.

- Problems that have highly unstable, nonlinear dynamics may require a very good initial guess for the solution in order to be solved by RIOTS_95.

- General constraints on the controls that do not involve state variables are not handled efficiently: adjoints are computed but not used.

- RIOTS_95 does not allow delays in the systems dynamics (although Padé approximations can be used).

- Numerical methods for solving optimal control problems have not reached the stage that, say, methods for solving differential equations have reached. Solving an optimal control problem can, depending on the difficulty of the problem, require significant user involvement in the solution process. This sometimes requires the user to understand the theory of optimal control, optimization and/or numerical approximation methods.

3. OPTIMAL CONTROL PROBLEMS IN RIOTS_95

RIOTS_95 is designed to solve optimal control problem of the form [4]

$$\mathbf{OCP}: \min_{(u,\xi)\in L_\infty^m[a,b]\times R^n} \Big\{ f(u,\xi) \doteq g_o(\xi, x(b)) + \int_a^b l_o(t,x,u)\mathrm{d}t \Big\}$$

subject to:

$$\dot{x} = h(t,x,u), \quad x(a) = \xi, \quad t \in [a,b],$$
$$u^j_{min}(t) \le u^j(t) \le u^j_{max}(t), j = 1,\cdots,m,$$
$$\xi^j_{min} \le \xi^j \le \xi^j_{max}, j = 1,\cdots,n,$$
$$l^\nu_{ti}(t, x(t), u(t)) \le 0, \quad \nu \in \mathbf{q}_{ti}, \quad t \in [a,b],$$
$$g^\nu_{ei}(\xi, x(b)) \le 0, \quad \nu \in \mathbf{q}_{ei},$$
$$g^\nu_{ee}(\xi, x(b)) = 0, \quad \nu \in \mathbf{q}_{ee},$$

where $x(t) \in R^n$, $u(t) \in R^m$, $g : R^n \times R^n \to R$, $l : R \times R^n \times R^m \to R$, $h : R \times R^n \times R^m \to R^n$ and we have used the notation $\mathbf{q} \doteq 1,\cdots,q$ and $L^m_\infty[a,b]$ is the space of Lebesgue measurable, essentially bounded functions $[a,b] \to R^m$. The functions in **OCP** can also depend upon parameters which are passed from Matlab at execution time using `get_flags`. Refer to [4](Sec. 4) for details.

[4]Not all of the optimization routines in RIOTS_95 can handle the full generality of problem **OCP**.

The subscripts o, ti, ei, and ee on the functions $g(\cdot,\cdot)$ and $l(\cdot,\cdot,\cdot)$ stand for, respectively, "objective function", "trajectory constraint", "endpoint inequality constraint" and "endpoint equality constraint". The subscripts for $g(\cdot,\cdot)$ and $l(\cdot,\cdot,\cdot)$ are omitted when all functions are being considered without regard to the subscript. The functions in the description of problem **OCP**, and the derivatives of these functions [5], must be supplied by the user as either object code or as M-files. The bounds on the components of xi and u are specified on the Matlab command line at run-time.

The optimal control problem **OCP** allows optimization over both the control u and one or more of the initial states ξ. To be concise, we will define the variable

$$\eta = (u, \xi) \in H_2 \doteq L_\infty^m[a,b] \times R^n.$$

With this notation, we can write, for example, $f(\eta)$ instead of $f(\xi, u)$. We define the inner product on H_2 as

$$< \eta_1, \eta_2 >_{H_2} \doteq < u_1, u_2 >_{L_2} + < \xi_1, \xi_2 > .$$

The norm corresponding to this inner product is given by $\| \eta \|_{H_2} = < \eta, \eta >_{H_2}^{1/2}$. Note that H_2 is a pre-Hilbert space.

Transcription for Free Final Time Problems. Problem **OCP** is a fixed final time optimal control problem. However, free final time problems are easily incorporated into the form of **OCP** by augmenting the system dynamics with two additional states (one additional state for autonomous problems). The idea is to specify a nominal time interval, $[a,b]$, for the problem and to use a scale factor, adjustable by the optimization procedure, to scale the system dynamics and hence, in effect, scale the duration of the time interval. This scale factor, and the scaled time, are represented by the extra states. Then `RIOTS_95` can minimize over the initial value of the extra states to adjust the scaling. For example, the free final time optimal control problem

$$\min_{u,T} \tilde{g}(T, y(T)) + \int_a^{a+T} \tilde{l}(t, y, u) \mathrm{d}t$$

$$\text{subjectto } \dot{y} = \tilde{h}(t, y, u), y(a) = \zeta, t \in [a, a+T$$

can, with an augmented state vector $x \doteq (y, x^{n-1}, x^n)$, be converted into the equivalent fixed final time optimal control problem

$$\min_{u,\xi^n} g(\xi, x(b)) \ + \ \int_a^b l(t, x, u) \mathrm{d}t$$

$$\text{subjectto } \dot{x} = h(t, x, u) \doteq \begin{pmatrix} x^n \tilde{h}(x^{n-1}, y, u) \\ x^n \\ 0 \end{pmatrix}$$

$$x(a) = \xi = \begin{pmatrix} \zeta \\ a \\ \xi^n \end{pmatrix} \ , \ t \in [a, b] \ ,$$

where y is the first $n-2$ components of x, $g(\xi, x(b)) \doteq \tilde{g}(a + T\xi^n, y(b))$, $l(t, x, u) \doteq \tilde{l}(x^{n-1}, y, u)$ and $b = a + T$. Endpoint and trajectory constraints can be handled in the same way. The quantity $T = b - a$ is the nominal trajectory duration. In this transcription, x^{n-1} plays the role of time and ξ^n is the *duration scale factor* so named because $T\xi^n$ is the effective duration of the trajectories for the scaled dynamics. Thus, for any $t \in [a, b]$, $x^n(t) = \xi^n$, $x^{n-1}(t) = a + (t-a)\xi^n$ and the solution, t_f, for the final time is $t_f = x^{n-1}(b) = a + (b-a)\xi^n$. Thus, the optimal duration is $T^* = t_f - a = (b-a)\xi^n = T\xi^n$. If $a = 0$ and $b = 1$, then $t_f = T^* = \xi^n$. The main disadvantage to this transcription

[5]If the user does not supply derivatives, the problem can still be solved using `riots` with finite-difference computation of the gradients.

is that it converts linear systems into nonlinear systems.

For autonomous systems, the extra variable x^{n-1} is not needed. Note that, it is possible, even for non-autonomous systems, to transcribe minimum time problems into the form of **OCP** using only one extra state variable. However, this would require functions like $h(t,x,u) = \tilde{h}(tx^n, y, u)$. Since `RIOTS_95` does not expect the user to supply derivatives with respect to the t argument it can not properly compute derivatives for such functions. Hence, in the current implementation of `RIOTS_95`, the extra variable x^{n-1} is needed when transcribing non-autonomous, free final time problems.

Trajectory constraints. The definition of problem **OCP** allows trajectory constraints of the form $l_{ti}(t,x,u) \leq 0$ to be handled directly. However, constraints of this form are quite burdensome computationally. This is mainly due to the fact that a separate gradient calculation must be performed for each point at which the trajectory constraint is evaluated.

At the expense of increased constraint violation, reduced solution accuracy and an increase in the number of iterations required to obtain solutions, trajectory constraints can be converted into endpoint constraints which are computationally much easier to handle. This is accomplished as follows. The system is augmented with an extra state variable x^{n+1} with

$$\dot{x}^{n+1}(t) = \mu \max\{0, l_{ti}(t,x(t),u(t))\}^2 \ ,$$

$$x^{n+1}(a) = 0 \ ,$$

where $\mu > 0$ is a positive scalar. The right-hand side is squared so that it is differentiable with respect to x and u. Then it is clear that either of the endpoint constraints

$$g_{ei}(\xi, x(b)) \doteq x^{n+1}(b) \leq 0 \qquad \text{or}$$

$$g_{ee}(\xi, x(b)) \doteq x^{n+1}(b) = 0$$

is satisfied if and only if the original trajectory constraint is satisfied. In practice, the accuracy to which **OCP** can be solved with these endpoint constraints is quite limited because these endpoint constraints do not satisfy the standard constraint qualification (see detailed description in Sec. 4 of [4]). This difficulty can be circumvented by eliminating the constraints altogether and, instead, adding to the objective function the penalty term $g_o(\xi, x(b)) \doteq x^{n+1}(b)$ where now μ serves as a penalty parameter.

However, in this approach, μ must now be a large positive number and this will adversely affect the conditioning of the problem.

Continuum Objective Functions and Minimax Problems. Objective functions of the form

$$\min_{u} \max_{t\in[a,b]} l(t,x(t),u(t))$$

can be converted into the form used in problem **OCP** by augmenting the state vector with an additional state, w, such that

$$\dot{w} = 0 \ ; \quad w(0) = \xi^{n+1}$$

and forming the equivalent trajectory constrained problem

$$\min_{(u,\xi^{n+1})} \xi^{n+1}$$

subject to

$$l(t,x(t),u(t)) - \xi^{n+1} \leq 0 \ , \quad t \in [a,b] \ .$$

A similar transcription works for standard min-max objective functions of the form

$$\min_{u} \max_{\nu\in\mathbf{q}_o} g^{\nu}(u,\xi) + \int_a^b l^{\nu}(t,x(t),u(t)\mathrm{d}t \ .$$

In this case, an equivalent endpoint constrained problem with a single objective function,

$$\min_{u,\xi^{n+1}} \xi^{n+1}$$

subject to

$$\tilde{g}^{\nu}(u,\xi) - \xi^{n+1} \leq 0 \ , \quad \nu \in \mathbf{q}_o$$

is formed by using the augmented state vector (x, w, z) with

$$\dot{w} = 0 \ , \quad w(0) = \xi^{n+1}$$

$$\dot{z}^{\nu} = l^{\nu}(t, x(t), u(t)) \ , \quad z^{\nu}(0) = 0 \ , \quad \nu \in \mathbf{q}_o \ ,$$

and defining

$$\tilde{g}^{\nu}(u,\xi) \doteq g^{\nu}(u,\xi) + z^{\nu}(b) \ .$$

4. EXAMPLE 1: BANG-BANG OCP

This is a textbook OCP problem [15, p. 112] with control bounds and free final time. It is used to demonstrate the transcription, explained in Sec. 3, of a free final time problem into a fixed final time problem. The transcribed problem has bounds on the control and free initial states. Also, **distribute.m** (see [4]) is used to improve integration mesh after an initial solution is found. A more accurate solution will then be computed by re-solving the problem on the new mesh. For self-containing, this OCP is given as follows:

$$\texttt{Problem Bang}: \quad \min_{u,T} J(u,T) = T$$

subject to

$$\dot{x}_1 = x_2; \quad x_1(0) = 0, x_1(T) = 300$$

$$\dot{x}_2 = u; \quad x_2(0) = 0, x_2(T) = 0$$

and

$$-2 \leq u(t) \leq 1, \quad \forall t \in [0,T].$$

This problem has an analytical solution which is given by $T^* = 30$. When $t \in [0,20)$, $u^*(t) = 1$, $x_1^*(t) = t^2/2$, $x_2^*(t) = t$ while when $t \in [20,30]$, $u^*(t) = -2$, $x_1^*(t) = -t^2 + 60t - 600$, $x_2^*(t) = 60 - 2t$.

The above OCP is a minimum-time problem with two states and one input. This problem is converted into a fixed final time problem using the transcription described in Sec. 3. Only one extra state variable was needed since the problem has time-independent (autonomous) dynamics. The augmented problem is implemented in the M-file form listed in Appendix A. First we will define the integration mesh and then the initial conditions.

```
>> N = 20;      % Discretization level
>> T = 10;      % Nominal final time
>> t=[0:T/N:T]; % Nominal time interval
```

The nominal time interval is of duration T. Next, we specify a value for xi^3, the duration scale factor, which is the initial condition for the augmented state. The quantity $T\xi^3$ represents our guess for the optimal duration of the maneuver.

```
>> x0=[0 0 1]';     % Init. cond. for aug. sys.
>> fixed=[1 1 0]';  % with init. cond.  fixed
>> x0_lower=[0 0 .1]';%free init.cond. low bnd
>> x0_upper=[0 0 10]';%free init.cond.  up bnd
>> X0=[x0,fixed,x0_lower,x0_upper]
X0 =
         0    1.0000         0         0
         0    1.0000         0         0
    1.0000         0    0.1000   10.0000
```

The first column of `X0` is the initial conditions for the problem; there are three states including the augmented state. The initial conditions for the original problem were

$x(0) = (0,0)^T$. The initial condition for the augmented state is set to $x0(3) = \xi^3 = 1$ to indicate that our initial guess for the optimal final time is one times the nominal final time of $T = 10$, i.e. $\xi^3 T$. The second column of `X0` indicates which initial conditions are to be considered fixed and which are to be treated as free variables for the optimization program to adjust. A one indicates fixed and a zero indicates free. The third and fourth col umns provide lower an upper bound for the free initial conditions.

```
>> u0=zeros(1,N+1);% f=x(3,1)=x0(3)
>> [u,x,f]=riots(X0,u0,t,-2,1,[],100,2);
>> f*T    % Show the final time.
ans =
   29.9813
```

In this call to `riots`, we have also specified a lower bound of -2 and an upper bound of 1 for all of the control spline coefficients. Since we are using second order splines, this is equivalent to specifying bounds on the value of the control at the spline breakpoints, i.e. bounds on $u(t_k)$. We also specify that the second order Runge-Kutta integration routine should be used. The objective value $f = \xi^3$ is the duration scale factor. The final time is given by $a + (b - a)\xi^3 = T\xi^3 = 10f$. Here we see that the final time is 29.9813. A plot of the control solution indicates a fairly broad transition region whereas we expect a bang-bang solution. We can try to improve the solution by redistributing the integration mesh. We can then re-solve the problem using the new mesh and starting from the previous solution interpolated onto the new mesh. This new mesh is stored in `new_t`, and `new_u` contains the control solution interpolated onto this new mesh.

```
>> [new_t,new_u]=distribut(t,u,x,2,[],1,1);
redistribute_factor = 7.0711
Redistributing mesh.

>> X0(:,1) = x(:,1);
>> [u,x,f]=riots(X0,new_u,new_t,-2,1,[],100,2);
>> f*10
ans =
   30.0000
```

Notice that before calling `riots` the second time, we set the initial conditions (the first column of `X0`) to `x(:,1)`, the first column of the trajectory solution returned from the preceding call to `riots`. Because ξ^3 is a free variable in the optimization, `x(3,1)` is different than what was initially specified for `x0(3)`. Since `x(3,1)` is likely to be closer to the optimal value for ξ^3 than our original guess we set the current guess for `X0(3,1)` to `x(3,1)`. We can see the improvement in the control solution and the solution for the final time. The reported final time solution is 30 and this happens to be the **exact answer**. The plot of the control solution before and after the mesh redistribution is shown below. The circles indicate where the mesh points are located. The improved solution does appear to be a bang-bang solution.

5. EXAMPLE 2: FED-BATCH REACTOR OPTIMIZATION

The fed-batch fermentor involving biosynthesis of penicillin was studied by Lim et al. [16] and revised by Cuthrell and Biegler [17], where the system is described by the four differential equations

$$\begin{aligned}
\frac{dx_1}{dt} &= h_1 x_1 - \frac{x_1}{500x_4}u \\
\frac{dx_2}{dt} &= h_2 x_1 - 0.01x_2 - \frac{x_2}{500x_4}u \\
\frac{dx_3}{dt} &= -\frac{h_1 x_1}{0.47} - \frac{h_2 x_1}{1.2} - \frac{0.029x_3 x_1}{0.0001 + x_3}
\end{aligned}$$

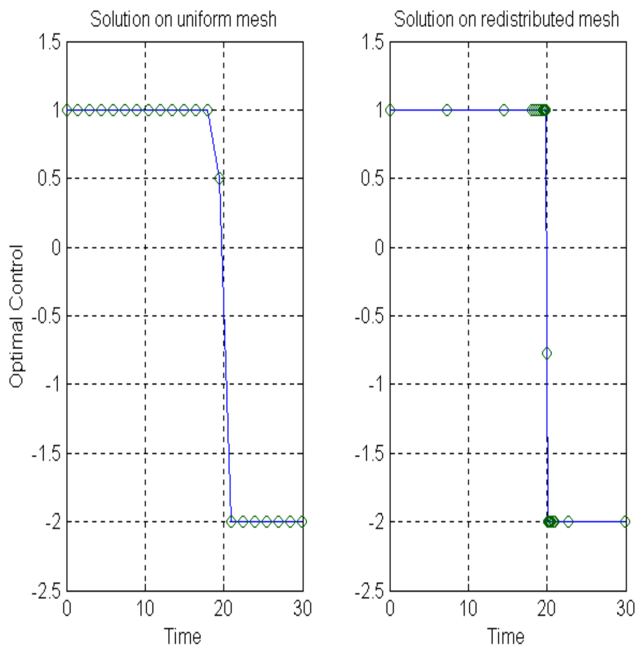

Figure 1: Bang-bang OCP solutions with and without mesh re-distribution

$$+\frac{500 - x_3}{500x_4}u$$

$$\frac{dx_4}{dt} = \frac{u}{500} \tag{1}$$

with

$$h_1 = \frac{0.11x_3}{0.006x_1 + x_3}$$

$$h_2 = \frac{0.0055x_3}{0.0001 + x_3(1 + 10x_3)} \tag{2}$$

where x_1 is the biomass concentration [g/l], x_2 the penicillin concentration [g/l] (product concentration [g/l]), x_3 the glucose concentration [g/l] (nutrition concentration [g/l]), x_4 the working volume of fermentor [l] and u the glucose feeding rate [l/h]. The initial state is

$$\mathbf{x}(0) = [1.5 \quad 0 \quad 0 \quad 7 \quad]^T. \tag{3}$$

The constraints on the feed rate are

$$0 \le u \le 50, \tag{4}$$

and on the state variables are

$$0 \le x_1 \le 40$$
$$0 \le x_3 \le 25$$
$$x_4(t_f) - 10 = 0 \tag{5}$$

where t_f is the batch time. The performance index to be maximized is the total amount of penicillin produced at time t_f, given by

$$J[u] = -x_2(t_f)x_4(t_f). \tag{6}$$

The optimal control problem is to find the batch time t_f and the control policy $u(t)$ in the time interval $0 \le t < t_f$ so that the performance index given in (6) is minimized $J[u]$ (or $x_2(t_f)x_4(t_f)$ maximized) and all the constraints are satisfied.

Determining the optimal control policy of highly nonlinear systems in the presence of state constraints is a very difficult problem. To handle the state equality constraint (for x_4) and state inequality constraints (for x_1 and x_3), we introduce the auxiliary variables x_5, x_6 and x_7 through the differential equations

$$\frac{\mathrm{d}x_5}{\mathrm{d}t} = \begin{cases} w_1(x_1 - 40)^2 & \text{if } x_1 > 40 \\ 0 & \text{if } 0 \le x_1 \le 40 \\ w_1x_1^2 & \text{if } x_1 < 0 \end{cases} \tag{7}$$

$$\frac{\mathrm{d}x_6}{\mathrm{d}t} = \begin{cases} w_3(x_3 - 25)^2 & \text{if } x_3 > 25 \\ 0 & \text{if } 0 \le x_3 \le 25 \\ w_3x_3^2 & \text{if } x_3 < 0. \end{cases} \tag{8}$$

$$\frac{\mathrm{d}x_7}{\mathrm{d}t} = \begin{cases} w_4(x_4 - 10)^2 & \text{if } x_4 > 10 \\ 0 & \text{if } 0 \le x_4 \le 10 \\ w_4x_4^2 & \text{if } x_4 < 0. \end{cases} \tag{9}$$

where w_1, w_3 and w_4 are positive constants specified by the user. The above treatment for state constraints on x_1 and x_3 is similar to what was done by Mekarapiruk and Luus [18], and by Luus and Hennessy [19] to handle the state inequality constraints. The initial conditions for x_5, x_6 and x_7 are chosen zero. If constraints are violated then these auxiliary variables will become positive. We now are ready to form the augmented performance index to be minimized

$$J_1 = J + x_4(t_f) + x_5(t_f) + x_6(t_f). \quad (10)$$

The C source codes (OCP1.C) for the above transcribed OCP are listed in Appendix B. Within MATLAB, the following script will generate the simulate.dll for running the OCP

```
>> mex -V4 simulate.obj utility.obj ocp1.c ...
       drivers.lib libf77.lib libi77.lib;
```

Here libf77.lib and libi77.lib are the official libraries for f2c [6]. simulate.obj and utility.obj are precompiled objective files for the RIOTS_95 users. drivers.lib contains all the supported integration methods including the LSODA [7] module converted from its Fortran form.

Now we are ready to solve this OCP via RIOTS_95. This is shown in the following scripts.

```
>> N = 126*2; T = 126.5; t = [0:T/N:T];
>> u00=16.5; % initial control is a constant
>> U_max=50; % maximum control (upper limit)
>> u0 = u00+zeros(1,N); %use Spline of order 1
>> x0 = [1.5 0 0 7 0 0 0 ]'; % initial states
>> simulate(0,[1,1,1]);%initialization of simu.
>> [f,x]=simulate(1,x0,u0,t,5,2);
>> disp('Before Optimization, J value =')
>> x(2,N+1)*x(4,N+1)
>> tic
>> [u,x,f]=riots(x0,u0,t,0,U_max,[1,1,1],...
          [100,.01],5);
>> toc
>> disp('Optimal value ='); x(2,N+1)*x(4,N+1)
```

We first run the simulate without performing the optimization using just the initial states. Of course, the state trajectories may violate the contraints although the J value looks bigger.

```
Fed-batch penicillin biosynthesis
This is a nonlinear system with 7 states,
1 inputs and 3 parameters,
...
LSODA detected stiffness.
Before Optimization, J value = 112.1933
```

Now by calling riots, we proceed tc sovle the OCP via the embedded NPSOL module [8]. The following lists a part of the running results.

```
Treating linear constraints as nonlinear.
LSODA detected stiffness.

Calling NPSOL.
...
LSODA detected stiffness.

Exit  NP phase.  Inform =  6     Majits =    9
                   nfun =   33   ngrad  =    9

Exit NPSOL-Current point cannot be improvd upon

Final nonlinear objective value =   -84.53666
Completed 9 riots iterations.Line search failed

elapsed_time =   13.5600
Optimal value =   84.5367
```

We can see that the solving process is quite fast. Now, based on the above solution, a call to distribute will automatically re-distribute the mesh points and then based on the new mesh, we can call one more time riots to refine the solution. This is illustrated as follows:

```
>> [new_t,new_u]=distribut(t,u,x,5,[],0,1);
>> [u,x,f]=riots(x0,new_u,new_t,0,U_max, ...
               [1,1,1],50,5);
>> disp('Optimal value after distribut()=')
>> x(2,N+1)*x(4,N+1)
```

From the results listed below, we can see that the optimal performance has been improved from 84.5367 to 86.58981 which is about 2.43% more productive.

[6]For more information on f2c, visit http://www.netlib.org/f2c/.

[7]See http://www.netlib.no/netlib/odepack/ for details.

[8]See http://www.sbsi-sol-optimize.com/NPSOL.htm for details.

```
Changing spline order.
...
Calling NPSOL.
...

Exit  NP phase.  Inform =  6  Majits =    3
                   nfun = 19   ngrad =    3

Exit NPSOL-Current point cannot be improved upon

Final nonlinear objective value =   -86.58981

Completed 3 riots iterations. Line search failed
elapsed_time = 4.9500

Optimal value after distribut()= 86.5897
```

The detailed optimal control function and the states are shown in Fig. 2 and Figs. 3-4. We can conclude that by using RIOTS_95, it is relatively easy to obtain an optimal solution with reasonable accurary and computational cost. Notice that all optimal results shown in this chapter are for the fixed $t_f = 126.5$ hours.

It should be mentioned that, so far, the maximum yield of $J = 88.076$ is obtained with $\theta_4 = 0$ and with the corresponding batch time t_f 139.48 h as reported in [13]. It was noted in [13] that the performance index is not affected much if the batch time t_f is reduced from 139.48 yielding $J = 88.076$ to $t_f = 132$ giving $J = 88.009$. A 5% reduction in batch time reduces the yield by only 0.1%.

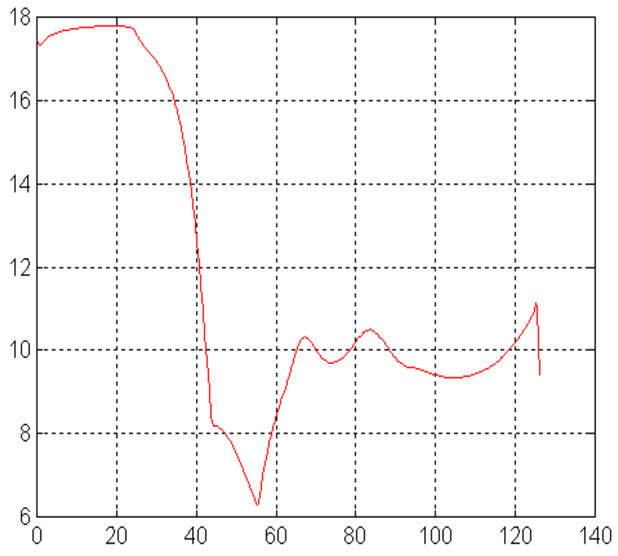

Figure 2: The optimial control function u

6. Future Work

This version of RIOTS_95 was developed over a period of more than two years. Many desirable features that could have been included were omitted because of time constraints. Moreover, there are many extensions and improvements that we have envisioned for future versions. We provide here a synopsis of some of the improvements currently being planned for hopefully, upcoming versions of RIOTS_95.

1. *Automatic Differentiation of user-supplied functions.* This would provide automatic generation of the derivative functions Dh, Dl and Dg using techniques of automatic differentiation [20, 21].

2. *Extension to Large-Scale Problems.* The size of the mathematical programming problem created by discretizing an optimal control problem (the way it is done in RIOTS_95 depends primarily on the discretization level N. The work done by the projected descent algorithm, pdmin, grows only linearly with N and hence pdmin (aug_lagrng) can solve

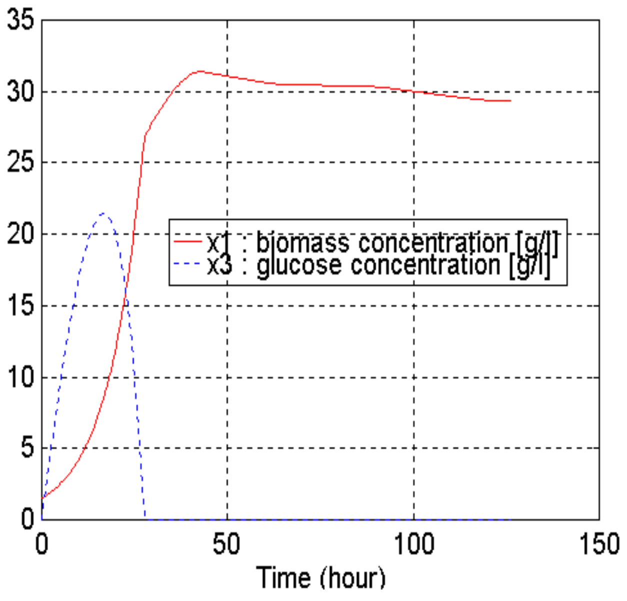

Figure 3: Optimal trajectories for x_1 and x_3

very large problems. However, these programs cannot handle trajectory constraints or endpoint equality constraints [9]. The main program in, `riots`, is based on dense sequential quadratic programming (SQP). Hence, `riots` is not well-suited for high discretization levels. There are many alternate strategies for extending SQP algorithms to large-scale problems as discussed in [1] (Chapter 6) The best approach is not known at this time and a great deal of work, such as the work in [22, 23, 24, 25] as well as our on investigations, is being done in this area.

3. *Trajectory constraints.*

   Our current method of computing functions gradients with respect to the control is based on adjoint equations. There is one adjoint equation for each function. This is quite inefficient when there are

[9]Endpoint inequality constraints can be handled effectively with `aug_lagrng` by incorporating a suitable active constraint set strategy.

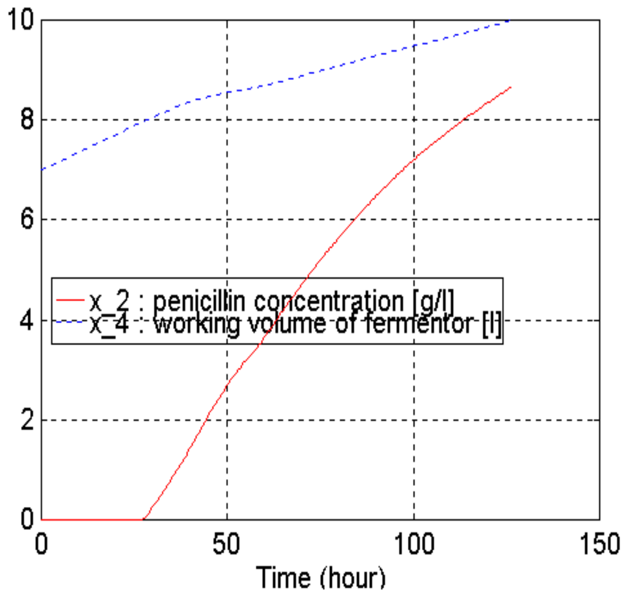

Figure 4: Optimal trajectories for x_2 and x_4

trajectory constraints because for each trajectory constraint there is, in effect, one constraint function per mesh point. Thus, for an integration mesh with $N+1$ breakpoints, roughly N adjoint equations have to be solved to compute the gradients at each point of a trajectory constraint. An alternate strategy based on the state-transition (sensitivity) matrix may prove to be much more efficient. Also, it is really only necessary to compute gradients at points, t_k, where the trajectory constraints are active or near-active. The other mesh points should be ignored. Algorithms for selecting the active or almost active constraint are present in [26, 27] along with convergence proofs.

4. *Stabilization of Iterates.* One of the main limitations of the current implementation of RIOTS is that it is not well-equipped to deal with problems whose dynamics are highly unstable. For such problems, the iterates produced by the

optimization routines in RIOTS can easily move into regions where the system dynamics "blow-up" if the initial control guess is not close to a solution. For instance, a very difficult optimal control problem is the Apollo re-entry problem [28]. This problem involves finding the optimum re-entry trajectory for the Apollo space capsule as it enters the Earth's atmosphere. Because of the physics of this problem, slight deviations of the capsules trajectory can cause the capsule to skip off the Earth's atmosphere or to burn up in the atmosphere. Either way, once an iterate is a control that drives the system into such a region of the state-space, there is no way for the optimization routine to recover. Moreover, in this situation, there is no way to avoid these regions of the state-space using control constraints.

This problem could be avoided using constraints on the system trajectories. However, this is a very expensive approach for our method (not for collocation-based methods), especially at high discretization levels. Also, for optimization methods that are not feasible point algorithms, this approach still might not work. An intermediate solution is possible because it is really only necessary to check the trajectory constraints at a few points, called nodes, in the integration mesh. This can be accomplished as follows. Let t_k be one such node. Then define the decision variable $\tilde{x}_{k,0}$ which will be taken as the initial condition for integrating the differential equations starting at time t_k. This $\tilde{x}_{k,0}$ is allowed to be different than the value $\bar{x}_k$ of the state integrated up to time t_k. However, to ensure that these values do, in fact, coincide at a solution, a constraint of the form $g_k(u) \doteq \tilde{x}_{k,0} - \bar{x}_k = 0$ must be added at each node. Note that, for nonlinear systems, $g_k(u)$ is a nonlinear constraint. The addition of these node variables allows bounds on that states to be applied at each node point. This procedure is closely related to the multiple shooting method for solving boundary value problems and is an intermediate approach between using a pure control variable parameterization and a control/state parameterization (as in collocation methods). See [29] for a discussion of node placement for multiple shooting methods.

5. *Other Issues and Extensions.* Some other useful features for RIOTS would include:

   - *A graphical user interface.* This would allow much easier access to the optimization programs and selection of options. Also, important information about the progress of the optimization such as error messages and warnings, condition estimates, stepsizes, constraint violations and optimality conditions could be displayed in a much more accessible manner.
   - *Dynamic linking.* Currently, the user of RIOTS must re-link `simulate` for each new optimal control problem. It would be very convenient to be able to dynamically link in the object code for the optimal control problem directly from Matlab (without having to re-link `simulate`). There are dynamic linkers available but they do not work with Matlab's MEX/DLL facility.

- For problems with dynamics that are difficult to integrate, the main source of error in the solution to the approximating problems is due to the integration error. In this case, it would be useful to use an integration mesh that is finer than the control mesh. Thus, several integration steps would be taken between control breakpoints. By doing this, the error from the integration is reduced without increasing the size (the number of decision variables) of the approximating problem.

- The variable transformation needed to allow the use of a standard inner product on the coefficient space for the approximating problems adds extra computation to each function and gradient evaluation. Also, if the transformation is not diagonal, simple bound constraints on the controls are converted into general linear constraints. Both of these deficits can be removed for optimization methods that use Hessian information to obtain search directions. If the Hessian is computed analytically, then the transformation is not needed at all. If the Hessian is estimated using a quasi-Newton update, it may be sufficient to use the transformation matrix $\mathbf{M}_N$ or $\mathbf{M}_\alpha$ as the initial Hessian estimate (rather than the identity matrix) and dispense with the variable transformation. We have not performed this experiment; it may not work because the the updates will be constructed from gradients computed in non-transformed coordinates [10].

- It may be useful to allow the user tc specify bounds on the control derivatives. This would be a simple matte for piecewise linear control representations.

- Currently the only way to specify general constraints on the controls is using mixed state-control trajectory constraints. This is quite inefficient since adjoint variables are computed but not needed for pure control constraints.

- Currently there is no mechanism in RIOTS for to directly handle systems with time-delays or, more generally, integro-differential equations [32]. This would be a non-trivial extension.

- Add support for other nonlinear programming routines in `riots`.

- There have been very few attempts to make quantitative comparisons between different algorithms for solving optimal control problems. The few reports comparing algorithms [33, 34], involve a small number of example problems, are inconclusive and are out of date. Therefore, it would be of great use to have an extensive comparison of some of the current implementations of algorithms for solving optimal control problems.

- Make it easy for the user to smoothly interpolate from data tables.

[10] With appropriate choice of H_0, quasi-Newton methods are invariant with respect to objective function scalings [30, 31] but not coordinate transformations (which is variable scaling)

References

[1] A. Schwartz, *Theory and Implementation of Numerical Methods Based on Runge-Kutta Integration for Solving Optimal Control Problems*, Ph.D. thesis, U.C. Berkeley, 1996.

[2] E. Polak, "On the use of consistent approximations in the solution of semi-infinite optimization and optimal control problems," *Math. Prog.*, vol. 62, pp. 385–415, 1993.

[3] John T. Betts, "SOCS:the sparse optimal control software family," Tech. Rep., `http://www.boeing.com/assocproducts/socs/index.html`, 1996.

[4] A. Schwartz, E. Polak, and Y. Q. Chen, *RIOTS: A Matlab Toolbox for Solving Optimal Control Problems Version 1.0 for Windows*, `http://www.accesscom.com/~adam/RIOTS`, May 1997.

[5] Oskar von Stryk, "DIRCOL:a direct collocation method for the numerical solution of optimal control problems," Tech. Rep., `http://www-m2.ma.tum.de/~stryk/dircol.html`, 1997.

[6] L. S. Jennings, M. E. Fisher, K. L. Teo, and C. J. Goh, "MISER3: Software for solving optimal control problems," Tech. Rep., `http://www.cado.uwa.edu.au/miser/`, 1997.

[7] R. B. Martin, "Optimal control drug scheduling of cancer chemotherapy," *Automatica*, vol. 28, no. 6, pp. 1113, 1992.

[8] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[9] R. E. Bellman, *Dynamic Programming*, Princeton University Press, 1957.

[10] R.E. Bellman and S Dreyfus, *Applied Dynamic Programming*, Princeton University Press, 1962.

[11] Dimitri P. Bertsekas, *Dynamic Programming and Optimal Control*, vol. 1 of *Optimization and Computation Series*, Athena Scientific, Nov. 2000.

[12] R. Luus, "Optimal control by dynamic programming using systematic reduction in grid size," *Int. J. of Control*, vol. 19, pp. 144–151, 1990.

[13] Rein Luus, *Iterative Dynamic Programming*, vol. 110 of *Monographs and Surveys in Pure and Applied Mathematics Series*, Chapman and Hall/CRC, 2000.

[14] YangQuan Chen, "Book review: "Iterative Dynamic Programming" by Rein Luus," *Int. of Robust and Nonlinear Control*, to appear, 2001.

[15] A. E. Bryson and Y. C. Ho, *Applied Optimal Control*, Hemisphere Publishing Corp., 1975.

[16] H. C. Lim, Y. J. Tayeb, J. M. Modak, and P. Bonte, "Computational algorithms for optimal feed rates for a class of fed-batch fermentation: numerical results for penicillin and cell mass production," *Biotechnol. Bioeng.*, vol. 28, pp. 1480–1420, 1986.

[17] J. E. Cuthrell and L. T. Biegler, "Computational algorithms for optimal feed rates for a class of fed-batch fermentation: numerical results for penicillin and cell mass production," *Comput. Chem. Eng.*, vol. 13, pp. 49–62, 1989.

[18] W. Mekarapiruk and R. Luus, "Optimal control of inequality state constrained systems," *Ind. Eng. Chem. Res.*, vol. 36, pp. 1686–1694, 1997.

[19] R. Luus and D. Hennessy, "Optimization of fed-batch reactors by the Luus-Jaakola optimization procedure," *Ind. Eng. Chem. Res.*, vol. 38, pp. 1948–1955, 1999.

[20] A. Griewank, "On automatic differentiation," Preprint MCS-P10-1088 `ftp://info.mcs.anl.gov/pub/tech_reports/reports`, Argonne National Laboratory, 1988.

[21] A. Griewank, D. Juedes, and J. Utke, "ADOL-C: A package for the automatic differentiation of algorithms written in C/C++," Preprint `ftp://info.mcs.anl.gov/pub/ADOLC`, Argonne National Laboratory, 1988.

[22] J. T. Betts and P. D. Frank, "A sparse nonlinear optimization algorithm," *J. Optim. Theory and Appl.*, vol. 82, no. 3, pp. 519–541, 1994.

[23] J. T. Betts and W.P.Huffman, "Path-constrained trajectory optimization using sparse sequential quadratic programming," *J. Guidance,Control, and Dynamics*, vol. 16, no. 1, pp. 59–68, 1993.

[24] Henrik Jonson, *Newton Method for Solving Non-linear Optimal Control Problems with Genereal constraints*, Ph.D. thesis, Linkoping Studies in Science and Technology, 1983.

[25] J. C. Dunn and D. P.Bertsekas, "Efficient dynamic programming implementations of newton s method for unconstrained optimal control problems," *J. Optim. Theory and Appl.*, vol. 63, no. 1, pp. 23–38, 1989.

[26] J. E. Higgins and E. Polak, "An reactive barrier-function method for solving minimax problems," *Appl. Math. Optim.*, vol. 23, pp. 275–297, 1991.

[27] J. L. Zhou and A. L. Tits, "An SQP algorithm for finely discretized continuous minimax problems and other minimax problems with many objective functions," *SIAM J. of Optimization and Control*, 1996.

[28] O. Stryk and R. Bulirsch, "Direct and indirect methods for trajectory optimization," *Annals of Operational Research*, vol. 37, pp. 357–373, 1992.

[29] U. Ascher, R. Mattheij, and R. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[30] D. F. Shanno and K. H. Phua, "Matrix conditioning and nonlinear optimization," *Math. Prog.*, vol. 14, pp. 149–160, 1978.

[31] S. S. Oren, "Perspectives on self-scaling variable metric algorithms," *J. of Optim. Theory and Appl.*, vol. 37, no. 2, pp. 137–147, 1982.

[32] F. H. Mathis and G. W. Reddien, “Difference approximations to control problems with functional arguments,” *SIAM J. of Control and Optim.*, vol. 16, no. 3, pp. 436–449, 1978.

[33] D. I. Jones and J. W. Finch, “Comparison of optimization algortihms,” *Int. J. of Control*, vol. 40, pp. 747–761, 1984.

[34] S. Strand and J. G. Balchen, “A comparison of constrained optimal control algorithms,” in *IFAC 11th Triennial World Congress*, Estonia, USSR, 1990, pp. 439–447.

APPENDIX 1: M-FILES FOR THE OCP EXAMPLE 1.

To solve any user specified OCP via M-file interface, the user only needs to provide a set of MATLAB M-files. No C-compiler is needed. The following list is the OCP `bang`.

```
function message = sys_activate
message = 'bang';

function neq = sys_init(params)
% Here is a list of the different system
% information paramters.
% neq = 1  : number of state variables.
% neq = 2  : number of inputs.
% neq = 3  : number of parameters.
% neq = 4  : reserved.
% neq = 5  : reserved.
% neq = 6  : number of objective functions.
% neq = 7  : number of nonlinear
%  trajectory constraints.
% neq = 8  : number of linear trajectory
%        constraints.
% neq = 9  : number of nonlinear endpoint
%  inequality constraints.
% neq = 10 : number of linear endpoint
%  inequality constraints.
% neq = 11 : number of nonlinear endpoint
%  equality constraints.
% neq = 12 : number of linear endpoint
%  equality constraints.
% neq = 13 : 0 => nonlinear, 1 => linear,
% 2 => LTI, 3 => LQR, 4 => LQR and LTI.
% The default value is 0 for all except
% neq = 6 which defaults to 1.
% if params == [] then setup neq.  Otherwise
% the system parameters are getting passed.
if isempty(params)
   % Each row of neq specifies a setting for
   %   one of the pieces of system
   % information.  For example, to set the
   %  number of parameters to 5
   % one row in neq should be [3 5].
   neq = [1 3 ; 2 1 ; 12 3];  % nstates = 3;
%  ninputs = 1; 3 endpoint constr.
else
   global sys_params
   sys_params = params;
end

function xdot = sys_h(neq,t,x,u)
global sys_params
% xdot must be a column vector with n rows.
tau = x(3);
xdot = [tau*x(2) ; tau*u(1) ; 0];

function J = sys_g(neq,t,x0,xf)
global sys_params
% J is a scalar.
F_NUM = neq(5);
if F_NUM == 1
   J = x0(3);
elseif F_NUM == 2
   J = xf(1)/300.0 - 1;
elseif F_NUM == 3
   J = xf(2);
end

function z = l(neq,t,x,u)
global sys_params
% z is a scalar.
z = 0;

function [h_x,h_u] = sys_Dh(neq,t,x,u)
global sys_params
% h_x must be an n by n matrix.
% h_u must be an n by m matrix.
tau = x(3);
h_x = zeros(3,3);
h_u = zeros(3,1);
h_x(1,2) = tau;
h_x(1,3) = x(2);
h_x(2,3) = u(1);
h_u(2,1) = tau;

function [J_x0,J_xf,J_t] = sys_Dg(neq,t,x0,xf)
global sys_params
% J_x0 and J_xf are row vectors of length n.
% J_t is not used.
F_NUM = neq(5);
J_x0 = [0 0 0];
J_xf = [0 0 0];
if F_NUM == 1
   J_x0(3) = 1;
elseif F_NUM == 2
   J_xf(1) = 1/300;
elseif F_NUM == 3
   J_xf(2) = 1;
end

function [l_x,l_u,l_t] = sys_Dl(neq,t,x,u)
global sys_params
% l_x should be a row vector of length n.
% l_u should be a row vector of length m.
% l_t is not used.
l_x = [0 0 0];
l_u = 0;
```

APPENDIX 2: C-FILES FOR THE OCP EXAMPLE 2.

To solve any user specified OCP via C-MEX/DLL interface to MATLAB, the user needs to provide a set of ANSI C-files. A compiling and linking process is required using a C compiler (WATCOM) together with some pre-compiled libraries distributed with the RIOTS_95. The following is the source list for the OCP `fedbatch`.

```
/*******************************************/
#define NSTATES 7
#define NINPUTS 1
#define NPARAMS 3
/* Number of (fixed) system parameters. */
/* act as 3 weights for the penalty terms.*/
#define NFUNCTIONS 1
/* Number of objective functions. */
#define NLTC  0
/* Nonlinear traj. inequ. constraints. */
#define LTC   0
/* Linear traj. inequ. constraints. */
#define NLEIC 0
/* Nonlinear endpoint inequ. constraints. */
#define LEIC  3
/* Linear endpoint inequ. constraints. */
/* for x1 x3 x4 */
#define NLEEC 0
/* Nonlinear endpoint equ. constraints. */
#define LEEC  0
/* Linear endpoint equ. constraints. */
#define LTI   0
/* 1 => Linear, 2 => LTI, 3 => LQ, 4 => LQTI */
#define F_NUM neq[4]
double Wx1, Wx3, Wx4;
/* user input weights for x1 x4 state const. */
/*******************************************

void activate(message)
     char **message;
{
  *message = "Fed-batch penicillin biosynthesis
}

void init(neq,params)
     int neq[13];
     double *params;
{
  neq[0]  = NSTATES;
  neq[1]  = NINPUTS;
  neq[2]  = NPARAMS;
  neq[5]  = NFUNCTIONS;
  neq[6]  = NLTC;
  neq[7]  = LTC;
  neq[8]  = NLEIC;
  neq[9]  = LEIC;
  neq[10] = NLEEC;
  neq[11] = LEEC;
  neq[12] = LTI;
  Wx1=params[0];
  Wx3=params[2];
  Wx4=params[1];
}

/*  S Y S T E M   D Y N A M I C S            *
void h(neq,t,x,u,xdot)
  int neq[];
  double *t,x[NSTATES],u[NINPUTS],xdot[NSTATES]
{
  double temp,h1x1,h2x1;
  h1x1=.11*x[2]*x[0]/(0.006*x[0]+x[2]);
  h2x1=.0055*x[2]*x[0]/(.0001+x[2]*(1.+10*x[2])
  xdot[0]=h1x1 - x[0]*u[0]/(500*x[3]);
  xdot[1]=h2x1 - .01*x[1] - x[1]*u[0]/(500*x[3]
  xdot[2]=  - h1x1/0.47 - h2x1/1.2
      - 0.029*x[2]*x[0]/(0.0001+x[2])
            + u[0]*(1.0 - x[2]/500)/x[3];
  xdot[3] = u[0]/500;
/* x[0] */
  temp=(x[0]>40.)?Wx1*(x[0]-40.0)*(x[0]-40.0):0
  xdot[4]=temp+(x[0]<0.)?Wx1*(x[0])*(x[0]):0.;
/* x[3] */
  temp=(x[3]>10.)?Wx4*(x[3]-10.)*(x[3]-10.):0.;
  xdot[5]=temp+(x[3]<0.)?Wx4*(x[3])*(x[3]):0.;
/* x[2] */
  temp=(x[2]>25.)?Wx3*(x[2]-25.)*(x[2]-25.):0.;
  xdot[6]=temp+(x[2]<0.)?Wx3*(x[2])*(x[2]):0.;
}

void Dh(neq,t,x,u,h_x,h_u)
int neq[];
double *t,x[NSTATES],u[NINPUTS];
double h_x[NSTATES][NSTATES];
double h_u[NSTATES][NINPUTS];
{
  double num1,den1,num2,den2;
  double temp, h1,h2,h1x1,h2x1;
  double dh1dx1,dh1dx3,dh2dx3;
  num1=0.11*x[2];   den1=(.006*x[0]+x[2]);
  num2=0.0055*x[2]; den2=.0001+x[2]*(1.+10*x[2]);
  h1= num1/den1;   h2= num2/den2;
  dh1dx1 =-x[2]*0.11*0.006/(den1*den1) ;
  dh1dx3 =.11*0.006/(den1*den1) ;
  dh2dx3 =.0055*(.0001-10*x[2]*x[2])/(den2*den2);
  h_x[0][0] = h1 + x[0]*dh1dx1 - u[0]/(500*x[3]);
  h_x[0][2] = x[0]*dh1dx3 ;
  h_x[0][3] = x[0]*u[0]/(500*x[3]*x[3]);
  h_x[1][0] = h2;
  h_x[1][1] = -0.01 - u[0]/(500*x[3]);;
  h_x[1][2] = x[0]*dh2dx3;
  h_x[1][3] = x[1]*u[0]/(500*x[3]*x[3]);
  h_x[2][0] = (-h1-x[0]*dh1dx1)/0.47
               -h2/1.2 - .029*x[2]/(0.0001+x[2]);
  h_x[2][2] = -x[0]*dh1dx3/0.47-x[0]*dh2dx3/1.2
  - 0.029*x[0]*.0001/((.0001+x[2])*(.0001+x[2]))
  - u[0]/(500*x[3]);
  h_x[2][3] = -u[0]*(1.0-x[2]/500)/(x[3]*x[3]);
  temp = (x[0] > 40.0) ? Wx1*2.0*(x[0]-40.0):0.;
  h_x[4][0] = temp+(x[0]<0.)?Wx1*2.0*(x[0]):0.;
  temp = (x[3] > 10.0) ? Wx4*2.0*(x[3]-10.0):0.;
  h_x[5][3] = temp+ (x[3]<0.)?Wx4*2.*(x[3]):0.;
  temp = (x[2] > 25.0) ? Wx3*2.0*(x[2]-25.):0.;
  h_x[6][2] = temp+ (x[2]<0.)?Wx3*2.*(x[2]):0.;
  h_u[0][0] = -x[0]/(500*x[3]);
  h_u[1][0] = -x[1]/(500*x[3]);
  h_u[2][0] = (1.0 - x[2]/500)/x[3];
  h_u[3][0] = 1.0/500.0;
}

/* I N T E G R A L   C O S T   F U N C T I O N */
double l(neq,t,x,u)
     int neq[];
     double *t,x[NSTATES],u[NINPUTS];
{
  switch ( F_NUM ) {
  case 1:
    neq[3] = -1;
    return 0.0;
    break;
  }
}

double Dl(neq,t,x,u,l_x,l_u)
     int neq[];
     double *t,x[NSTATES],u[NINPUTS];
     double l_x[NSTATES],l_u[NINPUTS];
{
  return 0.0;
}

/* F I N A L   C O S T   F U N C T I O N
double g(neq,t,x0,xf)
     int neq[];
     double *t,x0[NSTATES],xf[NSTATES];
{
  switch (F_NUM) {
  case 1:
    return -xf[1]*xf[3];
    break;
  case 2:
    return xf[4];
    break;
  case 3:
    return xf[5];
    break;
  case 4:
    return xf[6];
    break;
  }
}

double Dg(neq,t,x0,xf,J_x0,J_xf)
     int neq[];
     double *t,x0[NSTATES],xf[NSTATES];
     double J_x0[NSTATES],J_xf[NSTATES];
{  switch (F_NUM) {
  case 1:
    J_xf[1] = -xf[3];
    J_xf[3] = -xf[1];
    break;
  case 2:
    J_xf[4] = 1.0;
    break;
  case 3:
    J_xf[5] = 1.0;
    break;
  case 4:
    J_xf[6] = 1.0;
    break;
  }
  return 0.0;
}
```