# Problem 1

Minimize the performance index

Subject to the constraints:

## Necessary Conditions for Minimum

Set up the Hamiltonian:

Apply the stationarity condition:

The differential equations governing the states and costates can be derived from the Hamiltonian, and the control variables eliminated:

The final values of and are all fixed. The final value of is free. Using the given performance index, the boundary condition on is:

Noting that and , it can be seen that . If the initial values of are known, then the differential equations can be used to solve the problem directly.

## fsolve() method

### Implementation

Using fsolve(), the unknowns to be computed are the values of , subject to the boundary conditions

fsolve() works more efficiently an analytical Jacobian is provided for the constraint functions with respect to the unknowns; i.e., if the following derivatives are specified:

To compute these values, the time histories of must be computed. The fsolve() routine is this:

1. Using an initial (zero) guess for the costate initial condition, compute the state and costate time histories.
2. Use the state and costate time histories to compute the time history of .
3. Take the gradient values at the final time and form the 4 x 4 matrix Jacobian for this problem.
4. fsolve() updates the initial guess for the unknowns based on the boundary conditions and the computed Jacobian

To compute the Jacobian, form the 8 x 8 matrix :

Recall that the time rates of the states and costates can be expressed in terms of the Hamiltonian:

The time rate of change of the elements of can be computed as follows:

Similarly, the other elements of are:

The derivative of can be expressed:

Due to their complexity, analytical expressions for the second partials of are not shown here. These expressions are essential to the [successive sweep approach](#_Implementation) and are shown in that section. These expressions were computed manually and verified using computer algebra (Mathematica). The value is the only one of interest. The sensitivities correspond to columns 5 through 8, rows 1, 3, 4, and 6 of . The analytical sensitivies agree with MATLAB’s finite difference estimation to within floating-point precision.

### Results (fsolve)

The solver uses initial guesses of zero for the initial costate values. fsolve() converges after 4 iterations with 9 function evaluations. Using fsolve(), the initial costates computed are:

The final state values are:

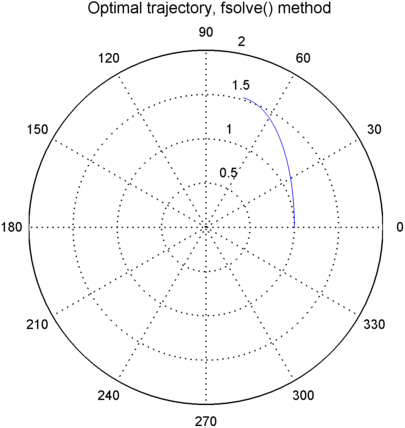


Figure 1: Optimal trajectory using fsolve().

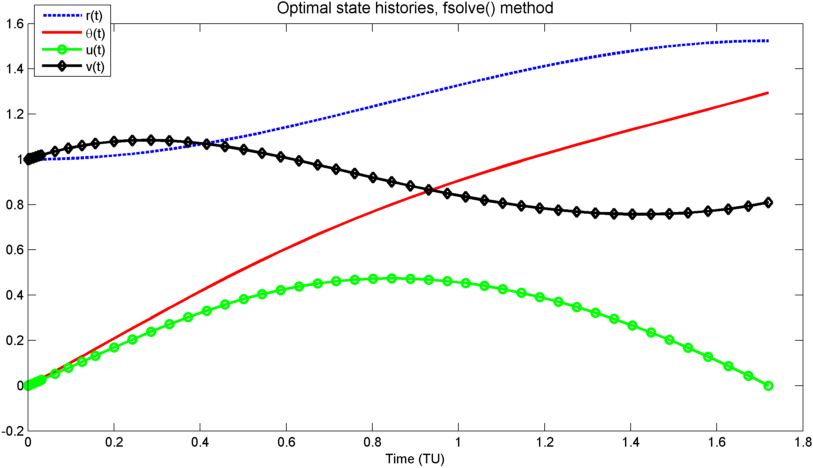


Figure 2: Optimal state history, fsolve()

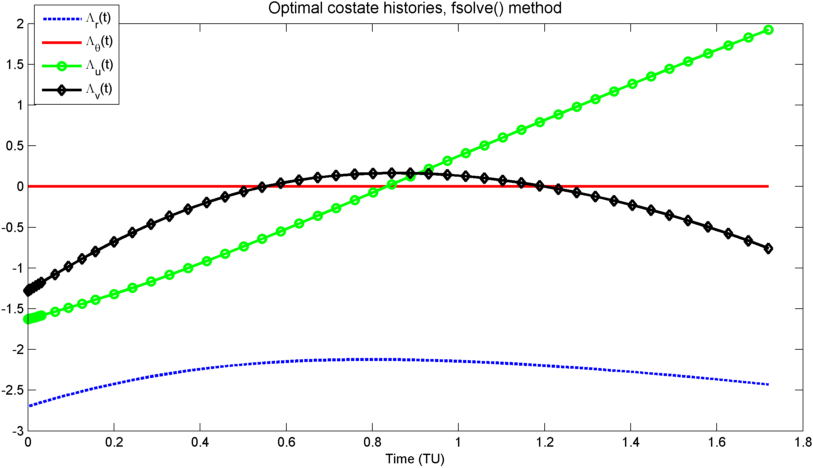


Figure 3: Optimal costate history, fsolve()

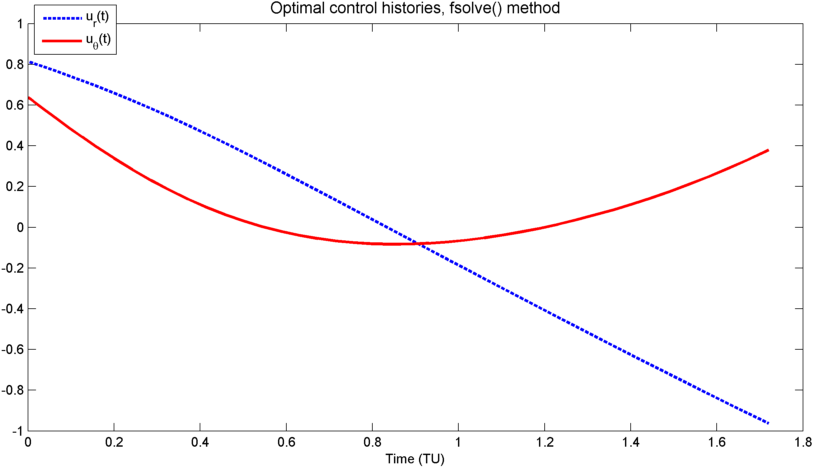


Figure 4: Optimal control history, fsolve()

## fminunc() method

### Implementation

The governing differential equations and boundary conditions are the same as before. For fminunc(), the arguments to the solver are the unknown initial conditions on and the function cost is the sum:

To improve the convergence of this method, the problem is discretized using an Euler first-order method to produce the following difference equations:

### Results (fminunc)

Using a discretization , the initial costate values are:

The final state values are:

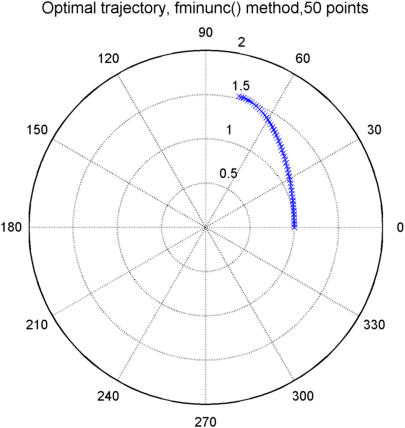


Figure 5: Optimal trajectory for fminunc() method

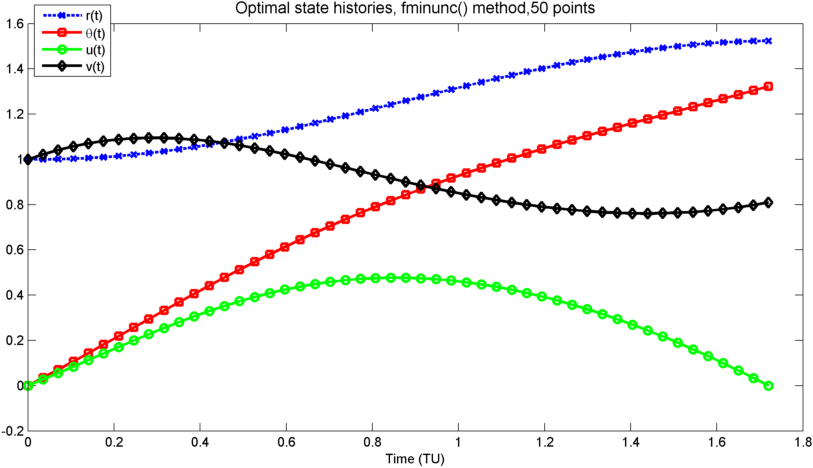


Figure 6: Optimal state histories for fminunc() method

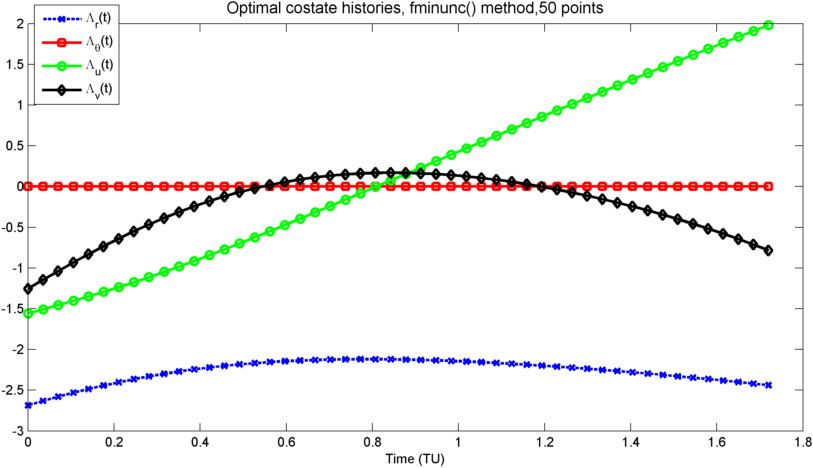


Figure 7: Optimal costate histories for fminunc() method

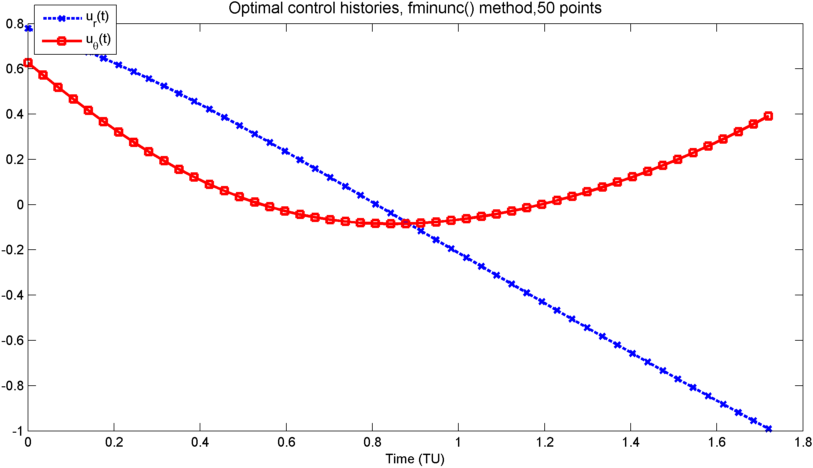


Figure 8: Optimal control history for fminunc() method

The results using fminunc() and fsolve() agree nicely and satisfy the necessary conditions.

## Successive Sweep Method

### Implementation

The successive sweep method requires an initial guess for the control history, solves a perturbation control problem about that control for a linear perturbation control , and updates the initial control using . As previous work has shown, neither the state nor the costate has any influence on the optimal control solution. Therefore, these terms are left out of the following derivation. To simplify the derivation of the perturbation problem, redefine the initial problem statement as a function-of-final-state-fixed OCP:

Minimize the performance index

Subject to the constraints:

where

Using an arbitrary control history, the system behavior can be simulated using the equations of motion derived previously. The state and costate histories generated will be referred to as , etc., although in general these histories are sub-optimal.

Once the nonlinear system dynamics have been simulated, the perturbation problem must be formulated and solved. Since the nonlinear control is generally sub-optimal and may not satisfy the constraint function , define the perturbation in the constraint :

Since the state history is suboptimal, set

In the previous equation, is a positive scalar such that . For this implementation was eventually selected. Defining the state equation , the perturbed state differential equation is:

The costate equation can be expressed as , so the perturbed costate equation is

Note that the performance index, , is quadratic and is a function of the control only, so the derivative . Furthermore, the derivative has the expression:

Finally, the stationarity condition is . In this problem, the performance index is , such that . The stationarity condition can be expressed:

For this problem, is constant, so the perturbed stationarity condition is:

Since the initial state conditions are fixed, . The initial and final time are also fixed. The final perturbed state conditions, , are free. The general equation for the free final state boundary condition is

The perturbed form of this equation is

The nonlinear system final state is linear with respect to , so is invariant; therefore, . The boundary condition for is .

To solve, assume a solution of the form subject to the boundary constraints , .

For this problem, . Substituting for , the equations for and are

Substituting the assumed form for produces Riccati equations for and :

These equations can be integrated backwards in time independently of the perturbation states and costates. An Euler approximation is used for integration. To obtain a solution for , however, the unknown must be found. To do this, assume that the following equation can be solved:

The value of is fixed by the solution to the nonlinear equations. Therefore, the derivative of the previous expression is:

Using the equation for , the preceding expression yields Ricatti equations for and :

These equations can be solved using back integration from the known final values of the matrices. Note that . can be expressed as follows:

If the time histories of and are both known, then can be evaluated at any time except as , where . Choosing to evaluate at the initial time, we can write

The perturbation state and control equations can be written:

For completeness, the following derivatives of the Hamiltonian are shown here:

### Initial control vector

To generate an initial solution that was relatively near the optimal control, we assume a nonlinear feedback control of the form:

This produces the effective dynamics

This control does not exactly drive to the desired , and is nonzero. However, this control is close enough for the successive sweep method to converge. It is assumed the successive sweep has converged when the absolute value of each element of the vector is less than a tolerance of .

### Results (successive sweep)

Using the specified initial controller and convergence criteria, the sweep converges after 21 iterations. The initial costates computed are:

The final state values are:

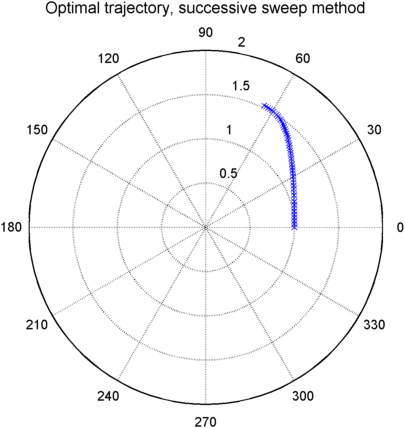


Figure 9: Optimal trajectory, successive sweep method

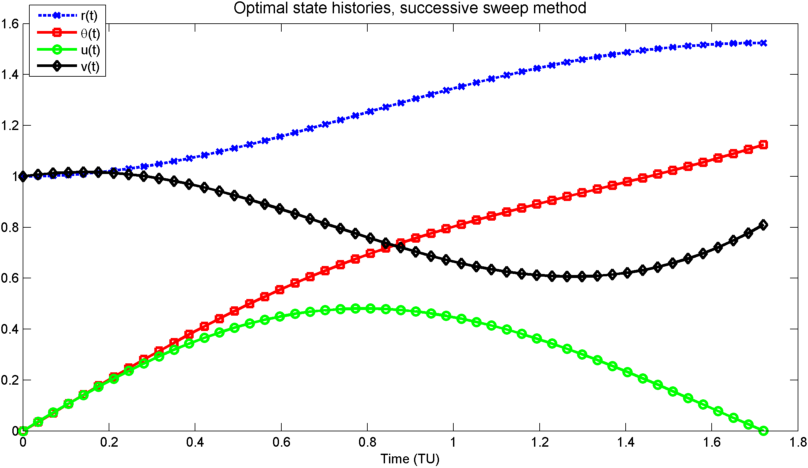


Figure 10: Optimal state histories, successive sweep method

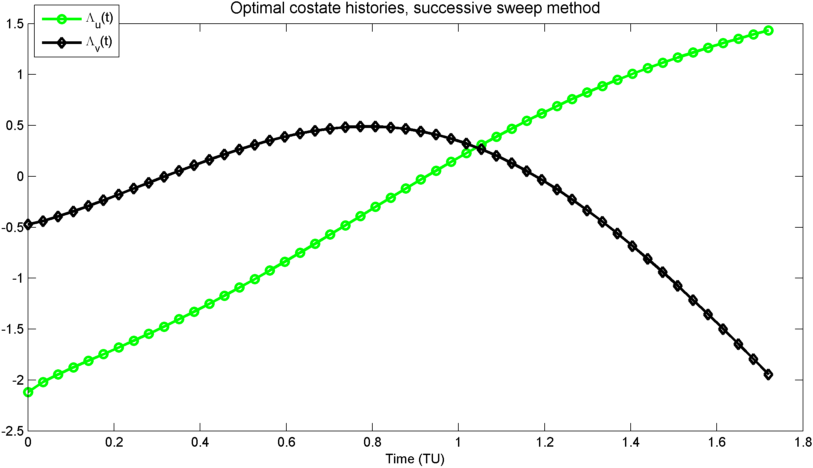


Figure 11: Optimal costate histories, successive sweep method

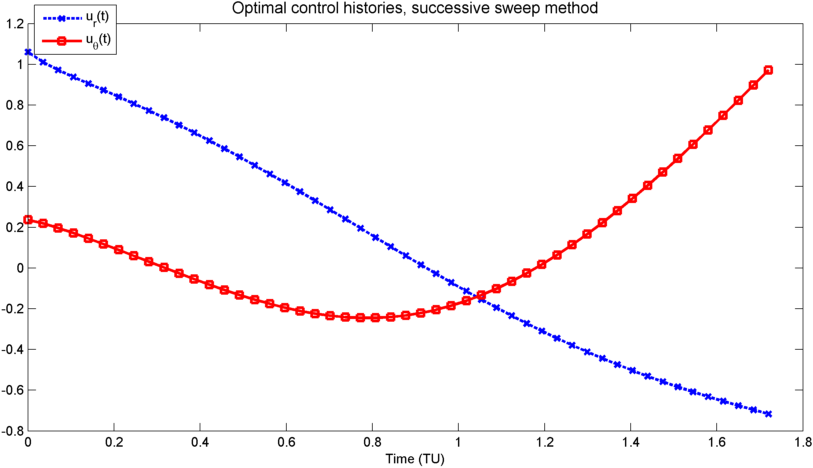


Figure 12: Optimal control history, successive sweep method

It can be seen that the control calculated using the successive sweep differs from the fminunc() and fsolve() control, although the same final state is reached. The cost function for fminunc() and the successive sweep are estimated using trapezoid integration; it is found that the cost for fminunc() is approximately 0.574, while the cost for the successive sweep 0.722.

Checking the 2nd-order necessary conditions for the successive sweep, it is seen that the strengthened Legendre-Clebsh condition is met, and the Jacobi condition, is finite, is satisfied everywhere except at the final time steps, as . It is possible that the successive sweep solution is a local minimum, and the global minimum could be found with a better initial control solution; for instance, if the optimal solution computed with fminunc() is used as the initial control, the successive sweep returns .