

A short study on Free End Time OCPs

Maria Vilches Mejias

Supervised by Jose Javier Muñoz Romero

BIMR, July 2026

1 Theoretical background

Free end time optimal control problems are a type of optimal control problems (or OCPs for short) in which we are interested in, not only, optimizing a control function (denoted here by $u(t)$), but also the end time T , which we consider to be free. Therefore, given a state, $x(t)$, we can formulate this type of problem in the following way:

Definition 1. Let $f(t, x(t), u(t))$ and $g(t, x(t), u(t))$ be two continuously differentiable functions and $\phi(t, x(t))$ be a differentiable function. The free end time OCP of state $x(t)$ and control $u(t)$ is

$$\begin{aligned} \max_{u, x, T} \quad & \left(\int_{t_0}^T f(t, x(t), u(t)) dt + \phi(T, x(T)) \right) \\ \text{subject to} \quad & \dot{x}(t) = g(t, x(t), u(t)), \\ & x(t_0) = x_0 \end{aligned} \tag{1}$$

where $\phi(T, x(T))$ is called the payoff term. The analogous problem for a minimizer can also be defined. The solution functions and variables to this problem are denoted as $u^*(t)$, $x^*(t)$ and T^* . Using the terms introduced here we are able to define the cost function, Hamiltonian and Lagrangian.

Definition 2. We refer to the argument of the maximization (or minimization) condition of the free end time OCP as the cost function, and we denote it as

$$J(x(t), u(t), T) := \int_{t_0}^T f(t, x(t), u(t)) dt + \phi(T, x(T)) \tag{2}$$

Definition 3. Let $\lambda(t)$ be a piecewise differentiable function called the adjoint function. Then, for these OCPs, we define the Hamiltonian as

$$H(t, x(t), u(t), \lambda(t)) := f(t, x(t), u(t)) + \lambda(t)g(t, x(t), u(t)) \tag{3}$$

Definition 4. Because we are dealing with an optimization problem under a restriction, it is useful to introduce the following Lagrangian, where the adjoint function takes the role of a Lagrange multiplier:

$$L(x(t), u(t), \lambda(t), T) := \int_{t_0}^T [f(t, x(t), u(t)) + \lambda(t)(g(t, x(t), u(t)) - \dot{x}(t))] dt + \phi(T, x(T)) \tag{4}$$

Therefore, the problem's solution should satisfy $\partial_x L|_* = 0$, $\partial_u L|_* = 0$ and $\partial_T L|_* = 0$. Using these

equations and applying the Hamiltonian's definition, we get the following necessary conditions:

$$\partial_u H|_* = 0 = f_u(t, x^*(t), u^*(t)) + \lambda(t)g_u(t, x^*(t), u^*(t)) \text{ optimality condition} \quad (5)$$

$$\dot{\lambda}(t) = -\partial_x H|_* = -f_x(t, x^*(t), u^*(t)) - \lambda(t)g_x(t, x^*(t), u^*(t)) \text{ adjoint equation} \quad (6)$$

$$\lambda(T^*) = \phi_x(T^*, x^*(T^*)) \text{ transversality equation} \quad (7)$$

$$H(T^*, x^*(T^*), u^*(T^*), \lambda(T^*)) + \phi_t(T^*, x^*(T^*)) = 0 \quad (8)$$

Where $|_*$ denotes the evaluation of the partial derivative in $(t, x^*(t), u^*(t), \lambda(t))$. Furthermore, given that $\partial_\lambda L|_* = 0$, the state equation can be rewritten as

$$\dot{x}^*(t) = \partial_\lambda H|_* = g(t, x^*(t), u^*(t)) \quad (9)$$

Therefore, by defining the new variable $z(t) := (x(t), \lambda(t))$ and $\tilde{J} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$, we can rewrite the state and adjoint equations as

$$\dot{z}^*(t) = \begin{pmatrix} \dot{x}^*(t) \\ \dot{\lambda}(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} \partial_x H|_* \\ \partial_\lambda H|_* \end{pmatrix} = \tilde{J} \nabla H|_* \quad (10)$$

Definition 5. We say the solution of our free end time OCP, $z^*(t) = (x^*(t), \lambda(t))$, is symplectic if it satisfies $(\partial_{z_0} z^*)^\top \tilde{J} (\partial_{z_0} z^*) = \tilde{J}$, where $z_0 = (x_0, \lambda_0) = (x(t_0), \lambda(t_0))$.

Proposition 1. The total derivative of the Hamiltonian with respect to time is equal to its partial derivative with respect to time, evaluated in the free end time OCP solution.

Proof. Applying the chain rule, knowing that $H(t, x(t), u(t), \lambda(t))$:

$$\frac{d}{dt} H(t, x(t), u(t), \lambda(t))|_* = \partial_t H|_* + \partial_x H|_* \dot{x}^*(t) + \partial_u H|_* \dot{u}^*(t) + \partial_\lambda H|_* \dot{\lambda}(t)$$

Using the optimality condition and the state and adjoint equations, the expression simplifies to

$$\frac{d}{dt} H(t, x(t), u(t), \lambda(t))|_* = \partial_t H|_* + (-\dot{\lambda}(t))\dot{x}^*(t) + (\dot{x}^*(t))\dot{\lambda}(t) = \partial_t H(t, x(t), u(t), \lambda(t))|_*$$

□

By the Hamiltonian's definition, if our free end time OCP is autonomous (i.e. if $f(t, x(t), u(t)) = f(x(t), u(t))$ and $g(t, x(t), u(t)) = g(x(t), u(t))$) the Hamiltonian is conserved along the optimal solution:

$$\frac{d}{dt} H(t, x(t), u(t), \lambda(t))|_* = 0$$

In other words, the Hamiltonian is not a function of time, t , and is constant along a fixed end time, T . However, it does change along the end time space, such that we can study the particular case in which the payoff term does not depend explicitly on T . Namely

$$\tilde{H}(T) := H(T, x^*(T), u^*(T), \lambda(T)) \text{ with } \lambda(T) = \phi_x(x^*(T))$$

satisfying $\tilde{H}(T^*) = 0$ from 8. We can also analyze $J(x^*, u^*, T)$ to see how the zeros of $\tilde{H}$ match with the maximums or minimums of the cost function.

2 Simple cancer cell model

We are interested in studying the symplecticity and the $\tilde{H}(T)$ and $J(x^*, u^*, T)$ functions of the free end time OCP problem presented on page 166 of Lenhart and Workman 2007. Namely:

$$\begin{aligned} \min_{u, x, T} \quad & \left(\int_0^T u(t)^2 dt + x(T) \right) \\ \text{subject to} \quad & \dot{x}(t) = \alpha x(t) - u(t), \\ & x(0) = x_0 \end{aligned}$$

In this problem, $x(t)$ represents the number of tumor cells at a certain time, α represents their growth factor ($\alpha > 0$, constant) and $u(t)$, the drug's concentration. The first thing we see is that

$$\begin{aligned} f(t, x(t), u(t)) &= u(t)^2 = f(u(t)) \\ g(t, x(t), u(t)) &= \alpha x(t) - u(t) = g(x(t), u(t)) \end{aligned}$$

Therefore, this specific problem is autonomous and must satisfy $\frac{d}{dt}H|_* = 0$. Also, because

$$\phi(T, x(T)) = x(T) = \phi(x(T))$$

, the transversality condition takes the form: $\lambda(T^*) = \phi_x(x(t))|_* = 1$. Because the Hamiltonian is $H(t, x(t), u(t), \lambda(t)) = u(t)^2 + \lambda(t)[\alpha x(t) - u(t)] = \alpha x(t)\lambda(t) - \frac{1}{4}\lambda(t)^2$, by applying the conditions and making the right substitutions, we arrive at the optimal control and state:

$$u^*(t) = \frac{1}{2}\lambda(t) \quad x^*(t) = \left(x_0 - \frac{\lambda_0}{4\alpha} \right) e^{\alpha t} + \frac{\lambda_0}{4\alpha} e^{-\alpha t}$$

where $\lambda(t) = \lambda_0 e^{-\alpha t}$. This solution allows us to calculate

$$\partial_{z_0} z^*(t) = \begin{pmatrix} \partial_{x_0} x^*(t) & \partial_{\lambda_0} x^*(t) \\ \partial_{x_0} \lambda(t) & \partial_{\lambda_0} \lambda(t) \end{pmatrix} = \begin{pmatrix} e^{\alpha t} & \frac{e^{-\alpha t} - e^{\alpha t}}{4\alpha} \\ 0 & e^{-\alpha t} \end{pmatrix}$$

and finally check that $(\partial_{z_0} z^*(t))^\top \tilde{J}(\partial_{z_0} z^*(t)) = \tilde{J}$. Therefore, we have proven the solution is symplectic analytically. Now let's check it is preserved numerically by applying forward Euler and backward Euler discretization algorithms on the state and adjoint equations, respectively:

$$\begin{cases} \frac{x_{n+1} - x_n}{\Delta t} \approx \dot{x} = g(x_n, u_n(\lambda_n)) = \alpha x_n - \frac{1}{2}\lambda_n \\ \frac{\lambda_{n+1} - \lambda_n}{\Delta t} \approx \dot{\lambda} = -\partial_x H|_{\lambda_{n+1}} = -\alpha \lambda_{n+1} \end{cases}$$

for $0 \leq n \leq N-1$ such that $x^*(0) = x_0$, $x^*(T) = x_N$ and $\lambda(0) = \lambda_0$, $\lambda(T) = \lambda_N$. We need to see whether $(\partial_{z_n} z_{n+1})^\top \tilde{J}(\partial_{z_n} z_{n+1}) = \tilde{J}$. To do so, we will rearrange the last equations into

$$\begin{cases} x_{n+1} \approx x_n(1 + \alpha \Delta t) - \frac{1}{2} \Delta t \lambda_n, \\ \lambda_{n+1} \approx \frac{1}{1 + \alpha \Delta t} \lambda_n. \end{cases} \quad (11)$$

and derive

$$\partial_{z_n} z_{n+1} = \begin{pmatrix} \partial_{x_n} x_{n+1} & \partial_{\lambda_n} x_{n+1} \\ \partial_{x_n} \lambda_{n+1} & \partial_{\lambda_n} \lambda_{n+1} \end{pmatrix} = \begin{pmatrix} 1 + \alpha \Delta t & -\frac{1}{2} \Delta t \\ 0 & \frac{1}{1 + \alpha \Delta t} \end{pmatrix}$$

such that indeed $(\partial_{z_n} z_{n+1})^\top \tilde{J}(\partial_{z_n} z_{n+1}) = \tilde{J}$.

As for the study of $\tilde{H}(T)$ and $J(x^*, u^*, T)$, for this example it is easy to check that:

$$\tilde{H}(T) = e^{\alpha T} \left(\alpha x_0 - \frac{1}{4} e^{\alpha T} \right) \quad J(x^*, u^*, T) = x_0 e^{\alpha T} - \frac{e^{2\alpha T} - 1}{8\alpha} \quad (12)$$

However, as for the more complex case in the next section, we have numerically solved the equations using MATLAB. By applying the forward-backward sweep method, solving for $\tilde{H}$ and integrating for $J(x^*, u^*, T)$ using the composite Simpson's rule, we obtained the following plot:

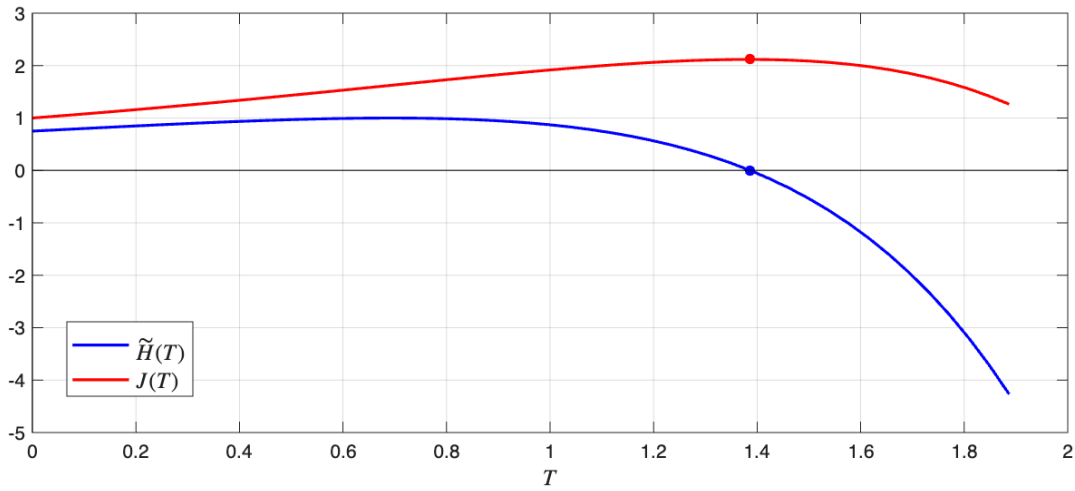


Figure 1: Plot of the Hamiltonian, $\tilde{H}$, and the cost function, J , as a function of the endtime, T . For this plot, we have set $\alpha = 1$ and $x_0 = 1$.

The marked points correspond to $\tilde{H}(T^*)$ (blue) and $J(T^*)$ (red), with an optimal end time $T_{num}^* = 1,3868$ (having set $\alpha = 1$ and $x_0 = 1$), which is a good approximation for the theoretical value of $T_{theo}^* = \ln(4)$. From the plot we can see that indeed $\tilde{H}(T_{num}^*) = -6.4762 \cdot 10^{-4} \approx 0$ and $J(T_{num}^*) = 2.1178$ which is approximately a local maximum¹. However, because we are looking for the end time (and control function) that minimizes the cost function, there does not exist a solution to the free end time OCP in this numerical range.

In fact, there is no solution globally, because by deriving the analytical solution in 12 we obtain:

$$\left. \frac{dJ}{dT} \right|_{T^*} = \tilde{H}(T^*) = 0 \quad \longrightarrow \quad T^* = \frac{1}{\alpha} \ln(4\alpha x_0), \quad \left. \frac{d^2 J}{dT^2} \right|_{T^*} = -4\alpha^3 x_0^2 < 0 \quad (\alpha > 0)$$

Therefore, there is only one maximum, which is the one we observe in the plot.

¹For this problem, solving analytically, $J(x^*, u^*, T^*) = \frac{(4\alpha x_0)^2 + 1}{8\alpha}$, so for $\alpha = 1$ and $x_0 = 1$ we get $J(T^*)_{theo} = 2 + \frac{1}{8}$.

3 The parachute problem

Imagine we are free-falling under the influence of gravity while strapped to a parachute. The evolution of our velocity is subject to the laws of physics, more specifically, to gravity and friction. We would like our velocity function, $v(t)$, to be equal to a desired velocity, v_d , for most of our trajectory, so, in order to achieve this, we have a small motor to supply an extra force to help us stabilize. This extra force is denoted by $u(t)$, and we would like to minimize its usage as much as possible. Applying Newton's 2nd law, we obtain the following free end time OCP:

$$\begin{aligned} \min_{u, v, T} \quad & \left(\frac{1}{2} \int_0^T (v(t) - v_d)^2 dt + \frac{\alpha}{2} \int_0^T u(t)^2 dt \right) \\ \text{subject to} \quad & \dot{v}(t) = g + \frac{1}{m}u(t) - \frac{k}{m}v(t), \\ & v(0) = v_0 \end{aligned}$$

where $\alpha > 0$ is a cost function constant, $g = 9.8 \text{ m/s}^2$ is the gravity constant and $k > 0$ is the air viscosity coefficient. We are interested in computing the optimal end time and plotting $\tilde{H}(T)$ and $J(v^*, u^*, T)$. First, let's build the Hamiltonian. Because

$$\begin{aligned} f(t, v(t), u(t)) &= \frac{1}{2}[(v(t) - v_d)^2 + \alpha u(t)^2] = f(v(t), u(t)) \\ g(t, v(t), u(t)) &= g + \frac{1}{m}u(t) - \frac{k}{m}v(t) = g(v(t), u(t)) \end{aligned}$$

we have $H(t, v(t), u(t), \lambda(t)) = \frac{1}{2}[(v(t) - v_d)^2 + \alpha u(t)^2] + \lambda(t)[g + \frac{1}{m}u(t) - \frac{k}{m}v(t)]$. Also, because f and g are autonomous, we also know that $\frac{d}{dt}H|_* = 0$. Applying the necessary conditions knowing that $\phi(T, x(T)) \equiv 0$, we get the following equations:

$$\begin{aligned} u^*(t) &= -\frac{1}{\alpha m} \lambda(t) & \lambda(T^*) &= 0 \\ \dot{\lambda}(t) &= -v^*(t) + v_d + \frac{k}{m} \lambda(t) & H(T^*, x^*(T^*), u^*(T^*), \lambda(T^*)) &= 0 \end{aligned}$$

The adjoint and state equations form a system of equations that yield the following solution²

$$\begin{aligned} v^*(t) &= c_1(T^*)\left(\frac{k}{m} - a\right)e^{at} + c_2(T^*)\left(\frac{k}{m} + a\right)e^{-at} + v_p \\ \lambda(t) &= c_1(T^*)e^{at} + c_2(T^*)e^{-at} + \lambda_p \end{aligned}$$

where

$$c_1(T) = -e^{-aT}\lambda_p - c_2(T)e^{-2aT} \quad c_2(T) = \frac{m(v_0 - v_p) + e^{-aT}(k - ma)\lambda_p}{k + ma - (k - ma)e^{-2aT}} \quad a = \frac{1}{m}\sqrt{k^2 + \frac{1}{\alpha}}$$

and

$$v_p = \frac{\alpha m k g + v_d}{1 + \alpha k^2} = v_d + \frac{k}{m}\lambda_p \quad \lambda_p = \frac{\alpha m^2 g - \alpha m k v_d}{1 + \alpha k^2}$$

²Solution notation extracted from Álvarez Guardiola 2026.

This allows us to construct the Hamiltonian at the optimal end time T^* :

$$\begin{aligned} H(T^*, x^*(T^*), u^*(T^*), \lambda(T^*)) &= f(v^*(T^*), u^*(T^*)) + \lambda(T^*)g(v^*(T^*), u^*(T^*)) \\ &= f(v^*(T^*), 0) = \frac{1}{2}[(v^*(T^*) - v_d)^2] = 0 \end{aligned}$$

where we have applied $\lambda(T^*) = 0$ and $u^*(T^*) = -\frac{1}{\alpha m} \lambda(T^*) = 0$. Now, computing the value of $v^*(T^*)$, we get $v^*(T^*) = v_d + a\lambda_p + 2ac_2(T^*)e^{-aT^*}$, therefore, for the Hamiltonian to be zero, it is required that

$$v^*(T^*) - v_d = a(\lambda_p + 2c_2(T^*)e^{-aT^*}) = 0 \quad \longrightarrow \quad 2c_2(T^*)e^{-aT^*} = -\lambda_p$$

after a few calculations, we can check that

$$2c_2(T^*)e^{-aT^*} = \frac{v_0 - v_d}{\frac{k}{m}\sinh(aT^*) + a\cosh(aT^*)} + \frac{(\frac{k}{m} - a)e^{-aT^*} - \frac{k}{m}}{\frac{k}{m}\sinh(aT^*) + a\cosh(aT^*)}\lambda_p$$

Because $k, m, a > 0$, $T \geq 0$, $\cosh(x) \geq 1$ and $\sinh(x) \geq 0$ for $x \geq 0$, the denominator is strictly positive. Then, for $2c_2(T^*)e^{-aT^*} = -\lambda_p$ we need

$$\frac{v_0 - v_d}{\frac{k}{m}\sinh(aT^*) + a\cosh(aT^*)} = 0 \quad \longrightarrow \quad v_0 = v_d \quad \text{or} \quad T^* \rightarrow \pm\infty$$

$$\frac{(\frac{k}{m} - a)e^{-aT^*} - \frac{k}{m}}{\frac{k}{m}\sinh(aT^*) + a\cosh(aT^*)} = -1 \quad \longrightarrow \quad T^* = 0$$

So, necessarily, $T^* = 0$ and $v_0 = v_d$ for a solution to exist. To check if this is correct, we will plot $\tilde{H}(T)$ and $J(v^*, u^*, T)$.

To numerically compute the solutions, we used a different approach from the one used in the cancer cell model. To start, we laid out the following recursions applying the forward Euler discretization algorithm on both the state and adjoint equations:

$$\begin{cases} \frac{v_{n+1} - v_n}{\Delta t} \approx \dot{v} = g(v_n, u_n(\lambda_n)) = g - \frac{1}{\alpha m^2} \lambda_n - \frac{k}{m} v_n \\ \frac{\lambda_{n+1} - \lambda_n}{\Delta t} \approx \dot{\lambda} = -\partial_x H|_{v_n, \lambda_n} = v_d + \frac{k}{m} \lambda_n - v_n \end{cases}$$

for $0 \leq n \leq N - 1$, such that $v^*(0) = v_0$, $v^*(T) = v_N$ and $\lambda(0) = \lambda_0$, $\lambda(T) = \lambda_N$, and rearranged them into

$$\begin{cases} v_{n+1} - (1 - \frac{k}{m}\Delta t)v_n + \frac{\Delta t}{\alpha m^2}\lambda_n = \Delta t g \\ \lambda_{n+1} - (\frac{k}{m}\Delta t + 1)\lambda_n + \Delta t v_n = \Delta t v_d \end{cases}$$

We have $2N + 2$ linear equations ($2N$ recursions and the initial and final conditions on the state and adjoint functions, respectively) and $2N + 2$ unknowns, so to find the numerical solution to the differential equations, it suffices to solve the system $M\tilde{z} = c$ where

$$M = \begin{pmatrix} e_0^\top & 0 \\ A & B \\ C & D \\ 0 & e_N^\top \end{pmatrix}, \quad \tilde{z} = (v_0, \dots, v_N, \lambda_0, \dots, \lambda_N)^\top, \quad c = (v_0, \Delta t g \cdot \mathbf{1}_N^\top, \Delta t v_d \cdot \mathbf{1}_N^\top, \lambda_N)^\top$$

for $e_0^\top = (1, 0, \dots, 0)$, $e_N^\top = (0, \dots, 0, 1) \in \mathbb{R}^{1 \times (N+1)}$ and $\mathbf{1}_N^\top = (1, 1, \dots, 1) \in \mathbb{R}^{1 \times N}$. As for the matrix blocks conforming $M \in \mathbb{R}^{2N+2}$, we have $A, B, C, D \in \mathbb{R}^{N \times (N+1)}$ and:

$$A = \begin{pmatrix} -\left(1 - \frac{k}{m}\Delta t\right) & 1 & 0 & \dots & 0 \\ 0 & -\left(1 - \frac{k}{m}\Delta t\right) & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -\left(1 - \frac{k}{m}\Delta t\right) & 1 \end{pmatrix} \quad B = \begin{pmatrix} \frac{\Delta t}{\alpha m^2} & 0 & \dots & 0 & 0 \\ 0 & \frac{\Delta t}{\alpha m^2} & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & 0 \\ 0 & \dots & 0 & \frac{\Delta t}{\alpha m^2} & 0 \end{pmatrix}$$

$$C = \begin{pmatrix} \Delta t & 0 & \dots & 0 & 0 \\ 0 & \Delta t & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & 0 \\ 0 & \dots & 0 & \Delta t & 0 \end{pmatrix} \quad D = \begin{pmatrix} -\left(1 + \frac{k}{m}\Delta t\right) & 1 & 0 & \dots & 0 \\ 0 & -\left(1 + \frac{k}{m}\Delta t\right) & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -\left(1 + \frac{k}{m}\Delta t\right) & 1 \end{pmatrix}$$

Finally, using the *mldivide* function built into MATLAB we can easily find the solution. Also, the method applied to search for the optimal end time has been changed, as now, because the Hamiltonian is strictly positive, the secant method is not useful. In its place, a minimization method is used to find the root. The plots returned are the following.

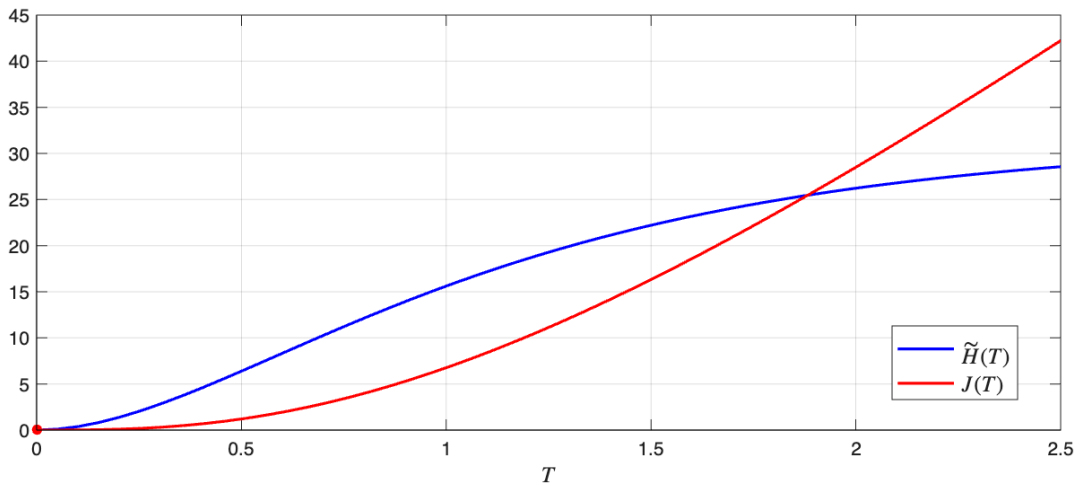


Figure 2: Plot of the Hamiltonian, $\tilde{H}$, and the cost function, J , as a function of the endtime, T . For this plot, we have set $\alpha = 1, v_0 = v_d = 1, k = 1, m = 1.25, g = 9.8$. The marked points correspond to the optimal end time in J and $\tilde{H}$.

We observe that, indeed, when $v_0 = v_d$, $T^* = 0$ is a solution to the free end time OCP, as it is a minimum of J (which is non-negative). More specifically, the computed optimal end time is $T_{num}^* = 0$ for which the functions take values $\tilde{H}(T_{num}^*) = 0$ and $J(T_{num}^*) = 0$.

Furthermore, let us analyze the optimal control and velocity as a function of time for different T s.

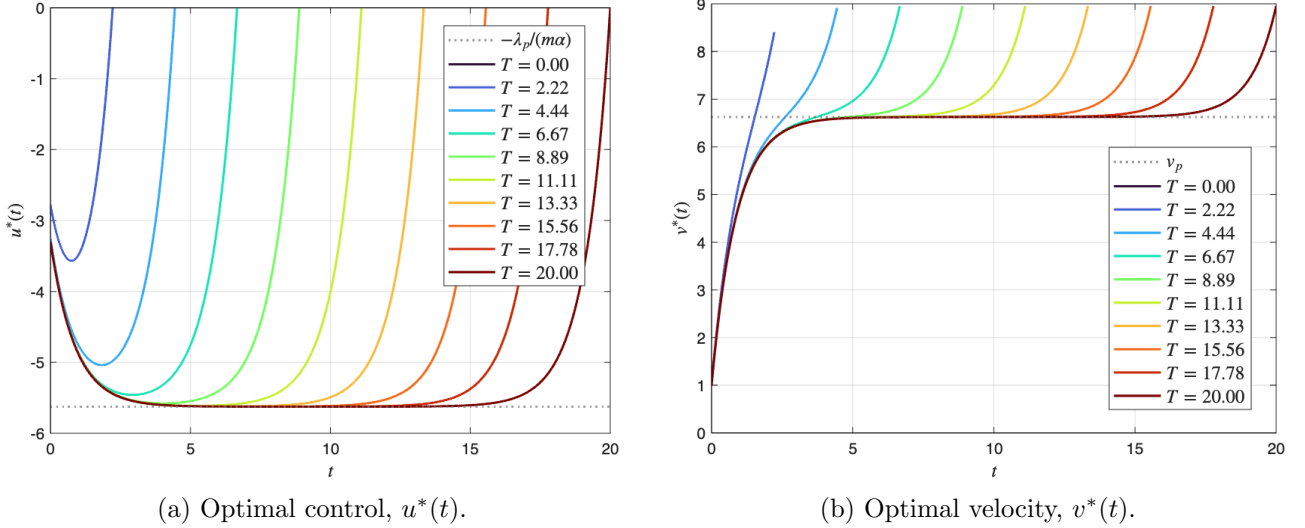


Figure 3: Plot of the optimal control and velocity for different values of the end time, T . The parameters are set to $\alpha = 1$, $v_0 = v_d = 1$, $k = 1$, $m = 1.25$, and $g = 9.8$.

As we can see, as the end time increases, both the control and velocity tend to stagnate to $-\lambda_p/m\alpha$ and v_p , respectively. This seems to indicate that for even bigger T s, the functions themselves do not change shape-wise but rather remain approximately constant for longer periods of time before turning to zero (for the control) or the end velocity. This behavior would explain why the cost function is monotonously increasing and eventually achieves a constant slope (as constant values for $v^*(t)$ and $u^*(t)$ along the majority of the trajectory yield linear dependencies on the cost function). This reinforces the idea that the only solution to our problem is for the end time to be zero.

Finally, let us interpret these results physically. First, we learn that in order to follow the desired velocity as close as possible and minimize the motor's usage, we need to begin with such velocity, to not use the motor and to not let time pass. Dynamically speaking, this is not a very interesting solution, therefore, if we let the end time increase, we must expect the motor to work against gravity ($u^*(t) \leq 0$ for all t), for it to eventually exert a constant force (which allows the system to reach equilibrium and the velocity to become constant), and for its usage to mitigate as the end time is approached (which causes the system to lose equilibrium and the velocity to increase). These dynamical effects that appear for non-zero end times contribute positively to the cost function, explaining the solution.

References

- Álvarez Guardiola, Esther (June 2026). "Structure-preserving discretisation of optimal control problems". Treball Final de Grau. URL: <https://hdl.handle.net/2117/465591>.
- Lenhart, Suzanne and John T. Workman (2007). *Optimal Control Applied to Biological Models*. Mathematical and Computational Biology Series. Chapman & Hall/CRC. ISBN: 9781584886402.

A Simple cancer cell model code

```

1 function MainFETOCP_cellmodel()
2
3     a=1; % exponential growth factor
4     x0=1; %initial number of tumor cells
5     lT=1; %final condition in lambda
6
7     T0=1;T1=2; %optimal end time initial guesses
8     tol=0.001;
9     N=1000;
10
11     % Evaluate H for the first two times
12     [x,l,u]=SolveODEadj(a,x0,lT,T0,N);
13     HT0=H(x(end),l(end),u(end),a);
14     T=T0;
15
16     if abs(HT0)>tol
17         [x,l,u]=SolveODEadj(a,x0,lT,T1,N);
18         HT1=H(x(end),l(end),u(end),a);
19         T=T1;
20
21         %update T until HT=0 using the secant method
22         while min(abs(HT0), abs(HT1))>tol
23
24             if abs(HT0-HT1)<tol
25                 error('Invalid T0 and T1 values\n');
26             end
27
28             T = T1-((HT1*(T1-T0))/(HT1-HT0));
29             [x,l,u] = SolveODEadj(a,x0, lT, T, N);
30             HT = H(x(end),l(end),u(end),a);
31             T0=T1; T1=T;
32             HT0=HT1; HT1=HT;
33         end
34     end
35
36     fprintf('The optimal end time is %.4f\n', T);
37
38     %is the hamiltonian constant along t?
39     Hn=H(x,l,u,a);
40     deltaH = max(abs(diff(Hn)));
41     fprintf('The maximum difference between consecutive Hns is %.4e\n',
42             deltaH);
43
44     %J(T) and H(T) representation
45     Topt=T; %rename the optimal end time
46     Hopt=H(x(end), l(end), u(end),a);

```

```

46     Jopt=J(x, u, Topt, N);
47     fprintf('The optimal values are Hopt=%.4e and Jopt=%.4e\n',Hopt, Jopt);
48     M=50; %number of evaluated points in the plot
49
50     % Evaluate H(T) and J(T) around the optimal end time
51     T_vals = linspace(0, Topt + 0.5, M);
52     H_vals = zeros(size(T_vals));
53     J_vals = zeros(size(T_vals));
54
55     for i = 1:length(T_vals)
56         [x, l, u] = SolveODEadj(a,x0, lT, T_vals(i), N);
57         H_vals(i) = H(x(end), l(end), u(end),a);
58         J_vals(i) = J(x, u, T_vals(i), N);
59     end
60
61     %Plot the values
62     plot(T_vals,H_vals,'b-','LineWidth',1.5);
63     hold on;
64     plot(T_vals,J_vals,'r-','LineWidth',1.5);
65
66     %Plot the optimal values
67     plot(Topt,Hopt,'bo','MarkerSize',5,'MarkerFaceColor','b');
68     plot(Topt,Jopt,'ro','MarkerSize',5,'MarkerFaceColor','r');
69
70     %Plot properties and creation of x and y axis
71     grid on;
72     box on;
73     xlabel('$T$', 'Interpreter','latex','FontSize',12);
74     xline(0,'k-','LineWidth',0.75);
75     yline(0,'k-','LineWidth',0.75);
76
77     %legend of the graph
78     legend({'$\tilde{H}(T)$','$J(T)$'}, 'Interpreter','latex', ...
79           'FontSize',12,'Location','best');
80
81 end
82
83 %Function definitions
84
85 %Integrate f using the composite Simpson's rule
86 function y=J(x,u,T,N)
87
88     h=T/N;
89     I=0;
90     for i=1:((N/2)-1)
91         I=I+((h/3)*(f(u(2*i))+4*f(u(2*i+1))+f(u(2*i+2))));
92     end
93

```

```

94     y=x(end)+I;
95
96 end
97
98 %Solve the state and adjoint equations using the
99 %forward-backward sweep method
100 function [x,l,u]=SolveODEadj(a,x0,lT,T,N)
101
102     x=zeros(N+1,1);
103     l=zeros(N+1,1);
104     u=zeros(N+1,1);
105
106     dt=T/N;
107     x(1)=x0;
108     l(end)=lT;
109
110     % Solution of the adjoint equation (backwards)
111     for i=N+1:-1:2
112         l(i-1)=l(i)+dt*Hx(l(i),a);
113     end
114
115     u=Hu(l);
116
117     % Solution of the state equation (forwards)
118     for i =1:N
119         x(i+1)=x(i)+dt*g(x(i),u(i),a);
120     end
121
122 end
123
124 %Function definitions
125 function y=f(u)
126     y=u.^2;
127 end
128 function y=g(x,u,a)
129     y=a*x-u;
130 end
131 function y=H(x,l,u,a)
132     y=f(u)+l.*g(x,u,a);
133 end
134 function y=Hx(l,a)
135     y=a*l;
136 end
137 function y=Hu(l)
138     y=l/2;
139 end

```

B Parachute model code

```

1 function MainFETOCP_parachute()
2
3     %parachute problem's parameters
4     lT=0; %final condition in lambda
5     v0=1; %initial condition in v
6     vd=1; %target velocity
7     k=1; %viscosity coefficient
8     m=1.25; %mass
9     g0=9.8; %gravity constant
10    alpha=1; %cost function constant
11
12    tol=1.e-8;
13    N=1000;
14
15    % Find T such that H(T)=0 knowing that H(T)>=0
16
17    %Start by defining a search interval
18    Tmin = 0; Tmax = 1;
19
20    %Define \tilde{H}(T) and find its minimum in the interval
21    Hfinal = @(T) Hamiltonian_at_end(T,v0,lT,N,g0,k,m,alpha,vd);
22    Tc = fminbnd(Hfinal,Tmin,Tmax);
23
24    % Compare with endpoints to find minimum
25    T_candidates = [Tmin,Tc,Tmax];
26    H_candidates = zeros(size(T_candidates));
27
28    for i = 1:length(T_candidates)
29        H_candidates(i) = Hfinal(T_candidates(i));
30    end
31
32    [Hmin,idx] = min(H_candidates);
33
34    % Check if the minimum Hamiltonian is within tolerance
35    if abs(Hmin) >= tol
36        error(['Time at minimum Hamiltonian does not yield' ...
37            ' a Hamiltonian close to zero.']);
38    end
39
40    %Optimal end time solution
41    T = T_candidates(idx);
42    [v,l,u] = SolveODEadj(v0,lT,T,N,g0,k,m,alpha,vd);
43    fprintf('The optimal end time is %.4e\n', T);
44
45    %is the hamiltonian constant?
46    Hn=H(v,l,u,alpha,vd,g0,k,m);

```

```

47     deltaH = max(abs(diff(Hn)));
48     fprintf('The maximum difference between consecutive Hns is %.4e\n',
49           deltaH);
50
51     %J(T) and H(T) representation
52     Topt=T; %rename the optimal end time
53     Hopt=H(v(end), l(end), u(end),alpha,vd,g0,k,m);
54     Jopt=J(v, u, Topt, N,alpha,vd);
55     fprintf('The optimal values are Hopt=%.4e and Jopt=%.4e\n',Hopt, Jopt);
56     M=50; %number of evaluated points in the plot
57
58     % Evaluate H(T) and J(T) around the optimal end time
59     T_vals = linspace(0, Topt + 2.5, M);
60     H_vals = zeros(size(T_vals));
61     J_vals = zeros(size(T_vals));
62
63     for i = 1:length(T_vals)
64         [v, l, u] = SolveODEadj(v0, lT, T_vals(i), N,g0,k,m,alpha,vd);
65         H_vals(i) = H(v(end), l(end), u(end),alpha,vd,g0,k,m);
66         J_vals(i) = J(v, u, T_vals(i), N,alpha,vd);
67     end
68
69     %Plot the values
70     plot(T_vals,H_vals,'b-','LineWidth',1.5);
71     hold on;
72     plot(T_vals,J_vals,'r-','LineWidth',1.5);
73
74     %Plot the optimal values
75     plot(Topt,Hopt,'bo','MarkerSize',5,'MarkerFaceColor','b');
76     plot(Topt,Jopt,'ro','MarkerSize',5,'MarkerFaceColor','r');
77
78     %Plot properties and creation of x and y axis
79     grid on;
80     box on;
81     xlabel('$T$', 'Interpreter','latex','FontSize',12);
82     xline(0,'k-','LineWidth',0.75);
83     yline(0,'k-','LineWidth',0.75);
84
85     %legend of the graph
86     legend({'$\tilde{H}(T)$','$J(T)$'}, 'Interpreter','latex', ...
87           'FontSize',12,'Location','best');
88 end
89
90 %Function definitions
91
92 %For a given T and parameters, solves the equations
93 %and evaluates H(T)

```

```

94 function y=Hamiltonian_at_end(T,v0,lT,N,g0,k,m,alpha,vd)
95
96     [v,l,u] = SolveODEadj(v0,lT,T,N,g0,k,m,alpha,vd);
97     y=H(v(end),l(end),u(end),alpha,vd,g0,k,m);
98
99 end
100
101 %Integrate f using the composite Simpson's rule
102 function y=J(v,u,T,N,alpha,vd)
103
104     h=T/N;
105     I=0;
106     for i=1:((N/2)-1)
107         I=I+((h/3)*(f(v(2*i),u(2*i),alpha,vd) ...
108             +4*f(v(2*i+1),u(2*i+1),alpha,vd) ...
109             +f(v(2*i+2),u(2*i+2),alpha,vd)));
110     end
111
112     y=I;
113
114 end
115
116 %Solve the state and adjoint equations using
117 %the method detailed in section 3 and mldivide
118 function [v,l,u] = SolveODEadj(v0,lT,T,N,g0,k,m,alpha,vd)
119
120     dt = T/N;
121
122     %Solve Az=b (*) where z=[v,l]
123     % (*) This notation differs from the one used in section 3,
124     %A, z and b correspond to M, \tilde{z} and c, respectively.
125     A = sparse(2*(N+1),2*(N+1));
126     b = zeros(2*(N+1),1);
127
128     %v0 input
129     A(1,1) = 1;
130     b(1) = v0;
131
132     for n = 0:N-1
133
134         %vn input where
135         %  $v_{n+1} - (1-dt*k/m)*v_n + (dt/(alpha*m^2))*l_n = dt*g0$ 
136         A(2*(n+1), n+1+1) = 1;
137         A(2*(n+1), n+1) = -(1-dt*(k/m));
138         A(2*(n+1), N+1+n+1) = dt/(alpha*m^2);
139         b(2*(n+1)) = dt*g0;
140
141         %ln input where

```

```

142         %  $l_{n+1} - (1+dt*k/m)l_n + dt*v_n = dt*vd$ 
143         A(2*(n+1)+1, N+1+n+1+1) = 1;
144         A(2*(n+1)+1, N+1+n+1) = -(1+dt*(k/m));
145         A(2*(n+1)+1, n+1) = dt;
146         b(2*(n+1)+1) = dt*vd;
147     end
148
149     %lT input
150     A(2*(N+1), 2*(N+1)) = 1;
151     b(2*(N+1)) = lT;
152
153     z = mldivide(A,b);
154
155     v = z(1:N+1);
156     l = z(N+2:2*(N+1));
157     u = Hu(l,alpha,m);
158
159 end
160
161 %Function definitions
162 function y=f(v,u,alpha,vd)
163     y=(1/2)*((v-vd).^2+alpha*u.^2);
164 end
165
166 function y=g(v,u,g0,k,m)
167     y=g0+(u/m)-(k/m)*v;
168 end
169
170 function y=H(v,l,u,alpha,vd,g0,k,m)
171     y=f(v,u,alpha,vd)+l.*g(v,u,g0,k,m);
172 end
173
174 function y=Hu(l, alpha, m)
175     y=(-1)/(alpha*m);
176 end

```