Optimal control as an extension of dynamics and the Pontryagin Maximum Principle

It may be surprising to hear that optimal control—the study of decision-making over time subject to constraints—is a natural extension of the study of dynamics. This is partly for historical reasons, since the derivation of dynamics predictions based on an optimization principle was itself based on a desire to make optimality a universal principle—for *studying* optics and mechanics and dynamics and for *designing* outcomes in these areas. In this regard science and engineering used to be much more entangled culturally than they are now.

The key point from our perspective is that we will be shifting from using optimization to *predict* natural phenomena to using optimization to *design* processes. There are lots of design optimizations one might want to do, but we will focus on decision-making, where a control vector can be chosen at every instant in time to influence the state of a system. For instance, one might have a motor that can exert a torque f_τ on a mechanical system at every time t, so that the goal becomes to choose $f_\tau(t)$ to achieve some goal.

To get from the ideas in variational principles as applied to dynamics to optimal control, a few key operations are needed. First, we will need to introduce an *objective*, which will look suspiciously like the action integral, and an *input* that will give us something to optimize over. Second, we need to introduce some operations that assist with differentiation—the *integral representation of an ordinary differential equation*, the *state transition matrix* for a linear ordinary differential equation, and the *convolution equation* for an affine linear ordinary differential equation (this will be how we talk about a linear system having an 'input' that can be controlled). These components, along with differentiation, will yield a relatively general form of the Pontryagin Maximum Principle, which is a set of ordinary differential equations governing optimality.

It is worth noting that the formulation we present here is not complete nor is it general. However, in a few pages one can get a relatively general view of optimal control as well as numerical methods that are genuinely useful. What one does not get is a) mechanisms for telling whether a given problem even has an optimal solution at all, and b) generalizations that allow one to take into account practical limitations (e.g., input saturation), not to mention other interesting things. These other interesting facets of optimal control are what make it a field of study, rather than simply an extension of dynamics like the view taken here.

Integral representations of ordinary differential equations

We will talk about derivatives of integral equations so that we can apply chain rule to an objective function $J(x(t), u(t))$, where $u(t)$ is the *input* to the system. First, we need to be able to represent an ordinary differential equation as an integral equation, to make it easier to differentiate with respect to state and control variables.

Let us remind ourselves of some things. Suppose we have a system described by the nonlinear ordinary differential equation:[162]

$$\dot{x} = f(x(t), u(t)) \quad x(0) = x_0.$$

[162] We will typically suppress the dependence on t to avoid too much notation.

An equivalent representation of that ordinary differential equation is:

$$x(t) = x_0 + \int_0^t f(x(\tau), u(\tau)) d\tau.$$

This latter equation is the *integral* form of the differential equation. To see that the two are equivalent, we can check if $x(t)$ satisfies the differential equation. By the Leibniz rule[163] we know that

[163] We haven't used this since we discussed impacts, but here it is again!

$$\frac{d}{dt}\left[x_0 + \int_0^t f(x(\tau), u(\tau)) d\tau\right] = f(x(t), u(t)).$$

So the expression for $x(t)$ satisfies the differential equation. Note that the differential equation incorporates the initial condition in a way that makes sense—at $t = 0$ the expression evaluated to x_0.

This integral representation of the ordinary differential equation is straight forward to differentiate using the same approaches we used before for differentating the action integral, though this time we will differentiate with respect to the input variable $u(t)$. As a result, we can apply chain rule to a cost function that depends on $x(t)$ and $u(t)$ and evaluate the first derivative of $J(x(t), u(t)$ with respect to $u(t)$. When that derivative is equal to zero (for all possible directions $v(t)$, the perturbations to $u(t)$), we know that we have a local minimizer, a local maximizer, or possibly a saddle point.

Derivatives of Objective Functions with Dynamic Constraints

Suppose we have an optimal control system and we want to optimize the system by setting the derivative of a cost J equal to zero. We will minimize an objective function J with respect to the input u.[164] For reasons that largely have to do with the nice mathematical outcome it ensures, we will choose the cost function to be of the form:[165]

$$J(u) = \int_0^T \ell(x, u) dt + m(x(T))$$

Moreover, we will assume that the state $x(t)$ and the control $u(t)$ are constrained by their dynamic relationship

$$\dot{x} = f(x, u) \quad x(0) = x_0.$$

[164] What we should minimize with respect to is a key decision; we could also minimize with respect to the pair (x, u) that satisfies the dynamics, or, possibly, a pair (α, μ) that does not satisfy the dynamics. Each of these leads to different algorithms, but we will focus here on minimizing solely with respect to u—the most common choice.

[165] The function under the integral sign ℓ will also depend explicitly on time t, but we will not worry about that here; it does not impact the analysis.

To differentiate J with respect to u, we will take the same approach as we did when developing the least action principle, and differentiate using the directional derivative.

$$\frac{d}{d\epsilon}J(u+\epsilon v)|_{\epsilon=0} = \int_0^T \frac{d}{d\epsilon}\ell(x, u+\epsilon v)dt + \frac{d}{d\epsilon}m(x(T))|_{\epsilon=0}$$

This would be easy except for the fact that x depends on u, so we have to figure out how they are related, and the first bit of that includes carefully applying chain rule. The above equation is rewritten below adjusting the notation to make the dependence on u and its perturbations $v(t)$ more clear.

$$\frac{d}{d\epsilon}J(u+\epsilon v)|_{\epsilon=0} = \int_0^T \frac{d}{d\epsilon}\ell(x(u+\epsilon v), u+\epsilon v)dt + \frac{d}{d\epsilon}m(x(u+\epsilon v)|_{t=T})|_{\epsilon=0}$$

$$= \int_0^T D_1\ell(x(u+\epsilon v), u+\epsilon v)\frac{\partial x}{\partial u}v + D_2\ell(x(u+\epsilon v), u+\epsilon v)v dt + Dm(x(u+\epsilon v)\frac{\partial x(T)}{\partial u}|_{\epsilon=0}$$

$$= \int_0^T D_1\ell(x(t), u(t))\frac{\partial x}{\partial u}(t) + D_2\ell(x(t), u(t))v(t)dt + Dm(x)\frac{\partial x}{\partial u}(T)$$

This notation is pretty unwieldy, but we are going to need it below. The D_1 notation just means we are differentiating with respect to the first variable and the D_2 means we are differentiating with respect to the second variable. If there is only one variable, we just use D to denote differentiation. Moreover, the expression $\frac{\partial x}{\partial u}$ is a function of time—it tell us how x is affected, to first order, when u is changed...anywhere in the time evolution of the input u. This is important conceptually, because in a dynamical system (like a pendulum), changing u at any point will generally have an effect on all subsequent states x later in time. So, we don't yet know how to concretely compute $\frac{\partial x}{\partial u}(t)$, but this first-order sensitivity will be important for analyzing the derivatives of J with respec to u.

To simplify the notation further, define $a^T = D_1\ell(x(t), u(t))$ and $b^T = D_2\ell(x(t), u(t))$, and $z = \frac{\partial x}{\partial u}(t)$. This gives us

$$\frac{d}{d\epsilon}J(u+\epsilon v)|_{\epsilon=0} = \int_0^T a(t)^T z(t) + b^T v(t)dt + Dm(x)z(T).$$

This really is just simplifying the notation to make the equation less visually complex—nothing mathematical has happened yet. However, it does let us interpret each term. The first two terms are the derivative of ℓ with respect to the state (evolving in time) multiplying the perturbation of the state due to a perturbation in control (also evolving in time). The second term is the derivative of ℓ with respect to the control (evolving in time) multiplying the perturbation to the control. So the derivative of the objective has two parts—the sensitivity to the state and the sensitivity to the control. They add together as a consequence of chain rule.

Now what we need is to understand the sensitivity of the state to the control $z(t) = \frac{\partial x}{\partial u}(t)$. This is where the integral form of the ordinary differential equation comes in. Assume that $\dot{x} = f(x, u)$ so that

$$x(t) = x_0 + \int_0^t f(x(\tau), u(\tau))d\tau.$$

We can differentiate this with respect to u.[166]

[166] Note the slight abuse of notation here, since I am allowing x to still take t as its argument.

$$\frac{\partial x}{\partial u}(t) \cdot v(t) = \frac{d}{d\epsilon}\left[x_0 + \int_0^t f(x(\tau), u(\tau) + \epsilon v(\tau))d\tau\right]_{\epsilon=0}$$

Evaluating this expression using chain rule, we get:

$$= \left[\frac{d}{d\epsilon}\int_0^t f(x(\tau), u(\tau))d\tau\right]_{\epsilon=0}$$

$$= \int_0^t D_1 f(x(\tau), u(\tau))\frac{\partial x}{\partial u}(\tau)e + D_2 f(x(\tau), u(\tau))v(\tau)d\tau$$

Looking at this can make one's head spin a bit the first time, so, as we did above with the derivative of the objective function, relabeling some variables can help. Setting $A(t) = D_1 f(x(t), u(t))$, and $B(t) = D_2 f(x(t), u(t))$ the equation above becomes

$$\frac{\partial x}{\partial u}(t) \cdot v(t) = \int_0^t A(\tau)z(\tau) + B(\tau)v(\tau)d\tau.$$

This *also* is just relabeling to make the equation a little easier to look at. We see that the sensitivity of the trajectory $x(t)$ to perturbations to $u(t)$ is a combination of how the dynamics evolve the sensitivity (the matrix A multiplies the sensitivity z so that changes in u early in time get integrated to future times) as well as the immediate effect of v on the trajectory.

Moreover, the equation above is the integral representation of the ordinary differential equation

$$\dot{z} = A(t)z(t) + B(t)v(t) \quad z(0) = 0.$$

That is, the function $z = \frac{\partial x}{\partial u}(t)$ satisfies a linear differential equation, which will help with analysis and will help identify what appears in the derivative of $J(u)$. Since this equation is linear, we need two ideas from linear systems theory: the *state transition matrix* and the *convolution equation*.

State transition matrices

The linear (time-varying) system above simplifies if we set $B(t) = 0$, which we do for the moment. This gives us a linear (time-varying) system

$$\dot{x} = A(t)x(t) \qquad x(t_0) = x_0.$$

The *state transition matrix* for this system is defined to be the matrix $\Phi(t, t_0)$ that satisfies the *matrix-valued* differential equation

$$\dot{\Phi} = A(t)\Phi \qquad \Phi(t_0, t_0) = Id_{n\times n}.$$

A remarkable and useful property of this matrix is that

$$x(t) = \Phi(t, t_0)x_0$$

holds for any linear system. In the special case where $A(t) = A$—the system is linear time-invariant (LTI)—it turns out that $\Phi(t, t_0) = e^{A(t-t_0)}$, the matrix exponential of $A(t - t_0)$.[167]

[167] So matrix exponentials show up in both linear systems theory and geometry. This isn't a coincidence, but I won't get into that here.

Convolution equations

Now we add $B(t)$ back in and consider the linear time-varying system

$$\dot{x} = A(t)x(t) + B(t)u(t) \qquad x(t_0) = x_0.$$

This does not have the same linear structure as the previous system, so the state transition matrix associated with $A(t)$ is not quite enough to compute the solution. Instead, we get the *convolution* equation

$$x(t) = \Phi(t, t_0)x_0 + \int_{t_0}^{t} \Phi(t, \tau)B(\tau)u(\tau)d\tau.$$

This equation is not quite as complex as it looks. Roughly speaking, it says that the control is coupled to what the system would have done based on its initial condition through the integral equation that weighs the input against the state transition matrix operating on the input. If $A = 0$, then we expect to be able to directly control the state; this is exactly what is predicted because for $A = 0$ we have $\Phi(t, t_0) = Id_{n\times n}\ \forall\ t$.

The Pontryagin Maximum Principle as a Variational Principle

$$J(u) = \int_{t_0}^{t_f} \ell(x, u)dt + m(x(t_f))$$

$$\dot{x} = f(x, u)$$

$$x(t_0) = x_0$$

$$DJ(u)\cdot v = \int_{t_0}^{t_f} \underbrace{D_1\ell(x,u)}_{a(t)^T}\underbrace{\frac{\partial x}{\partial u}}_{z(t)} + \underbrace{D_2\ell(x,u)}_{b(t)^T} v dt + \underbrace{Dm(x(t_f))}_{p_1^T}\underbrace{\frac{\partial x(t_f)}{\partial u}}_{z(t_f)}$$

We know that $z(t)$ and $v(t)$ are related through the linear differential equation

$$\dot{z} = \underbrace{D_1 f(x,u)}_{A(t)} z + \underbrace{D_2 f(x,u)}_{B(t)} v$$

and

$$z(t_0) = 0$$

because x_0 is given, so cannot be variable.

Using this notation,

$$DJ(u) = \int_{t_0}^{t_f} a^T z + b^T v dt + p_1^T z(t_f)$$

$$= \int_{t_0}^{t_f} a^T \left(\int_{t_0}^{t} \phi(t,\tau) B(\tau) v(\tau) d\tau \right) + b^T v dt + p_1^T \left(\int_{t_0}^{t_f} \phi(t_f,\tau) B(\tau) v(\tau) d\tau \right)$$

now, since $a(t)^T$ does not depend on τ, we can bring it in the second integral.

$$= \int_{t_0}^{t_f} \int_{t_0}^{t} a^T \phi(t,\tau) B(\tau) v(\tau) d\tau dt + \int_{t_0}^{t_f} b^T v dt + p_1^T \left(\int_{t_0}^{t_f} \phi(t_f,\tau) B(\tau) v(\tau) d\tau \right)$$

Now change the order of integration, noting in Fig. 48 the change in boundary conditions on the integrals.

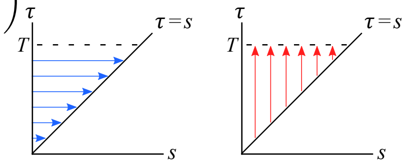

Figure 46: Changing order of integration.

$$= \int_{t_0}^{t_f} \int_{\tau}^{t_f} a^T \phi(t,\tau) B(\tau) v(\tau) dt d\tau + \int_{t_0}^{t_f} b^T v dt + p_1^T \left(\int_{t_0}^{t_f} \phi(t_f,\tau) B(\tau) v(\tau) d\tau \right)$$

factor out the terms that depend on τ to the right

$$= \int_{t_0}^{t_f} \left[\int_{\tau}^{t_f} a^T \phi(t,\tau) dt \right] B(\tau) v(\tau) d\tau + \int_{t_0}^{t_f} b^T v dt + p_1^T \left(\int_{t_0}^{t_f} \phi(t_f,\tau) B(\tau) v(\tau) d\tau \right)$$

now combine the left term with the right term

$$= \int_{t_0}^{t_f} \left[\underbrace{\int_{\tau}^{t_f} a^T \phi(t,\tau) dt + p_1^T \phi(t_f,\tau)}_{p(\tau)^T} \right] B(\tau) v(\tau) d\tau + \int_{t_0}^{t_f} b^T v dt$$

and, ignoring how we actually compute $p(\tau)$ for a moment...and replacing τ with t because it doesn't matter what we *label* the integration variable, we get

$$= \int_{t_0}^{t_f} \left[p(t)^T B(t) + b(t)^T \right] v(t) dt = 0$$

This lets us conclude that the term in the brackets must be equal to zero.

$$p(t)^T B(t) + b(t)^T = 0$$

Substituting in our notation, we get

$$p(t)^T D_2 f(x,u) + D_2 \ell(x,u) = 0$$

But we still need to figure out how to evaluate $p(\tau)$. Coming back to the expression for $p(\tau)$.

$$p(\tau)^T = \int_\tau^{t_f} a^T \phi(t,\tau)dt + p_1^T \phi(t_f,\tau)$$

This is similar to the convolution equation we saw before, but it isn't quite the same. The time variable for p is τ and τ appears at the beginning of the integral instead of the end of the integral, suggesting that this system is running backwards in time. This is at least consistent with the fact that $\phi(T,T) = I_{n\times n}$, and that p_1 is evaluated at time T.

Now, $p(t)$ satisfies a nice but somewhat unexpected differential equation. You can think of the integral expression that defines $p(s)^T$ as being the convolution equation, except that in this case the state transition matrix $\Phi(\tau, s)$ operates on a boundary condition at the final time T instead of the initial time 0. So let's examine $p(\tau)$ and differentiate it to find out what ordinary differential equation describes it.[168]

[168] Explain here that $\frac{d}{d\tau}\phi(t,\tau)$=$\phi(t,\tau)(-A)$, and why it is it true.

$$p(\tau)^T = \int_\tau^{t_f} a(s)^T \Phi(s,\tau)ds + p_1^T \Phi(t_f,\tau)$$

so

$$\frac{d}{d\tau}p(\tau)^T = \frac{d}{d\tau}\left[\int_\tau^{t_f} a(s)^T\Phi(s,\tau)ds + p_1^T\Phi(t_f,\tau)\right]$$

$$= -a(\tau)^T\phi(\tau,\tau) + p_1^T\phi(t_f,\tau)(-A) + \int_\tau^{t_f} a(s)^T\Phi(s,\tau)(-A)ds$$

$$= -a(\tau)^T + \left[\int_\tau^{t_f} a(s)^T\Phi(s,\tau)ds + p_1^T\phi(t_f,\tau)\right](-A)$$

$$= -a(\tau)^T - p^T A.$$

So we get that

$$\dot{p}^T = -p^T A - a^T$$

or, in column format,

$$\dot{p} = -A^T p - a = -D_1 f(x,u)^T p - D_1 \ell(x,u)^T.$$

What boundary condition do we know for p? Only what its value is as t_f, so instead of an initial condition we have the terminal condition

$$p(t_f) = p_1$$

Let's collect what we know so far.

$$\begin{aligned}
&\dot{x} = f(x,u) \quad x(t_0) = x_0 \\
&\dot{p} = -D_1 f(x,u)^T p - D_1\ell(x,u) \quad p(t_f) = Dm(x(t_f)) \\
&0 = p^T D_2 f(x,u) + D_2\ell(x,u)
\end{aligned}$$

These three equations are a version of *Pontryagin's Maximum Principle*. If one can solve the third equation for u and substitute into the first two equations, what results is a two point boundary value problem that can be numerically solved (including in python).

Example Consider a linear spring-mass system with a force applied. The ODE governing this system is

$$\ddot{x} = -kx + u,$$

where u is the force applied. In first-order form, this is

$$\begin{bmatrix} \dot{x} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k & 0 \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \qquad \begin{bmatrix} x(t_0) \\ w(t_0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Now let's assume that we want to the state to move from the initial condition to some desired state, such as $(x_ref, w_ref) = (2, 0)$. Then we could create an objective function

$$J = \int_{t_0}^{t_f} \ell(x,u)dt + m(x(t_f))$$

where

$$\ell = \frac{d_1}{2}(x(t)-2)^2 + \frac{d_2}{2}u(t)^2$$

and

$$m(x(t_f)) = \frac{c_1}{2}(x(t_f)-2)^2 + \frac{c_2}{2}w(t)^2.$$

Then the equation governing p would be

$$\begin{bmatrix} \dot{p}_1 \\ \dot{p}_2 \end{bmatrix} = -\begin{bmatrix} 0 & -k \\ 1 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} - \begin{bmatrix} d_1(x(t)-2) \\ 0 \end{bmatrix} \qquad \begin{bmatrix} p_1(t_f) \\ p_2(t_f) \end{bmatrix} = \begin{bmatrix} c_1(x(t_f)-2) \\ 0 \end{bmatrix}.$$

Lastly, looking at

$$p^T D_2 f + D_2 \ell = 0$$

we get that

$$u(t) = -p_2(t).$$

This gives us the following two point boundary value problem to solve.

$$\begin{aligned} \dot{x} &= -w \\ \dot{w} &= -kx - p_2 \\ \dot{p}_1 &= -d_1 x + k p_2 + 2 \\ \dot{p}_2 &= -p_1 \end{aligned}$$

with the boundary conditions

$$\begin{aligned} x(t_0) &= 1 \\ w(t_0) &= 0 \\ p_1(t_f) &= c_1(x(t_f) - 2) \\ p_2(t_f) &= 0 \end{aligned}$$

Solving this ODE will give us p_2 which in turn gives us the optimal $u(t)$.

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_bvp  # import the TPBVP solver

# initialize parameters
N = 100
T = 10.0
k = 1.6
d1 = 2.0
d2 = 0.1
c1 = 100.0
c2 = 0.0 #weight on terminal velocity equals zero in all code below #not
     included in any equations of motion#
xref = 2.0 #desired position at the end of the time horizon
x0 = np.array([1., 0.])
t_axis = np.linspace(T, 0, N)  # forward time steps
t_axis2 = np.linspace(0, T, N)  # backward time steps

# initial guess for the time series (x,p)
z = np.zeros((4, N))

# TPBVP system dynamics (in vector form)
def fun(t, z):
    return np.vstack((z[1],
                      -k*z[0]-(1/d2)*z[3],
                      -d1*z[0]+k*z[3]+d1*xref,
                      -1.0*z[2]))

# boundary condition specification based on usage of solve_bvp, see scipy
     documentation
def bc(za, zb):
    xT = zb[0:2]
    p1 = np.vstack((c1*(xT[0]-xref), c2*xT[1]))
    return np.array([za[0]-x0[0], za[1]-x0[1], zb[2]-p1[0], zb[3]-p1[1]])

# solve TPBVP
res = solve_bvp(fun, bc, t_axis2, z)
x1 = res.sol(t_axis2)[0:2]
p1 = res.sol(t_axis2)[2:4]
u1 = -1/d2 * p1[1]
print("Solver success: ", res.success)

# plot
fig = plt.figure()
ax = plt.gca()
ax.plot(t_axis2, x1[0,:], 'r')
ax.plot(t_axis2, x1[1,:], 'b')
ax.plot(t_axis2, u1.reshape(N), 'g')
ax.legend(['$x_1$(t)', '$x_2$(t)', 'u(t)'])
ax.set_title("Solution From TPBVP")
ax.set_xlabel("Time Steps")
ax.set_ylabel("System State")
plt.grid()
plt.show()
plt.close()
```

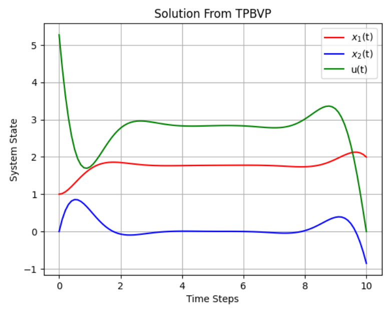

Figure 47: The solution to the TPBVP, including the optimal u, and states x and $\dot{x}$.

Note that this code can remain nearly unchanged if one replaces the system with a nonlinear pendulum, and the scipy functions will work just as well if one defines the dynamics to be the following.

```
# TPBVP system dynamics for nonlinear pendulum (treat k as the gravity term,
     also in vector form)
def fun(t, z):
    return np.vstack((z[1],
                      -k*np.sin(z[0])-c1*z[3],
                      -d1*z[0]+k*np.cos(z[0])*z[3]+xref,
                      -1.0*z[2]))
```

Exercise Use the code above to find an optimal control that controls the bead on the hoop to a particular angle away from vertical.

How do we tell if a solver has given us a good solution?

As the code above indicates, one can *state* the conditions for optimality for a general nonlinear system easily, and *specifying code* that nominally solves for the optimal solution is not much more challenging. However, unless the solver explicitly returns an error, how do we assess the numerically-obtained solution? One possibility is to evaluate the directional derivative of J in sample directions $v(t)$, though this is going to numerically sensitive, hard to interpret (how close to zero are we expecting?), and the set of $v(t)$ one chooses may not capture the worst-case directions. Alternatively one can look at the structure of the optimality conditions, which will lead to a surprising[169] connection to dynamics.

[169] At least to me!

Look again at the conditions in Pontryagin's Maximum Principle.

$$\begin{aligned}
&\dot{x} = f(x,u) \quad x(t_0) = x_0\\
&\dot{p} = -D_1 f(x,u)^T p - D_1\ell(x,u) \quad p(t_f) = Dm(x(t_f))\\
&0 = p^T D_2 f(x,u) + D_2\ell(x,u)
\end{aligned}$$

A notational restatement of these can be achieved by using the Hamiltonian formalism.

$$H = \ell + p^T f$$

The conditions for optimality are equivalent to Hamilton's equations, which we saw earlier in class.

$$\begin{aligned}\dot{x} &= D_p H^T \\ \dot{p} &= -D_x H^T \\ 0 &= D_u H^T\end{aligned}$$

Note that these equations have nothing to do with mechanical energy. Instead, from a purely formal perspective, they provide a convenient way of remembering the maximum principle. Moreover, one now sees why the variable p is labeled with a p—it shares the same role as momentum in the mechanical setting. But despite *not* being a mechanical Hamiltonian representing total energy[170], this Hamiltonian *is* conserved. After all, the only thing the proof of conservation relied on is Hamilton's equations, and the specifics of energy never played a role.[171] The upshot is that when we obtain an optimal $u(t)$, it should be a conserved quantity.

[170] ...or something like total energy

[171] This is not *quite* true—where did *an* aspect of mechanics play a role in proving that the Hamiltonian is conserved?

For the spring-mass-force control problem above,

$$H = \frac{1}{2}d_1(x - x_{ref})^2 + \frac{1}{2}d_2u^2 + [p_1, p_2] \cdot \begin{bmatrix} w \\ -kx + u \end{bmatrix}$$

where $u = \frac{-1}{d_2}p_2$. Plotting this from the solution to the TPBVP gives the plot below, which is indeed very close to conserved, suggesting that for this problem the solver did a good job approximating the solution.

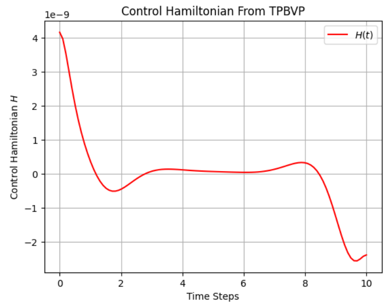

Figure 48: The control Hamiltonian $H$ for the spring-mass-force system. I have plotted the difference from the overall average value of $H$.

```
# plot Hamiltonian
H = 0.5*d1*(x1[0]-xref)**2+0.5*(1/d2)*p1[1]**2+p1[0]*x1[1]-(1/d2)*p1[1]**2-k
    *x1[0]*p1[1]
fig = plt.figure()
ax = plt.gca()
ax.plot(t_axis2, H[:]-sum(H)/len(H), 'r')
ax.legend(['$H(t)$'])
ax.set_title("Control Hamiltonian From TPBVP")
ax.set_xlabel("Time Steps")
ax.set_ylabel("Control Hamiltonian $H$")
plt.grid()
plt.show()
plt.close()
```

You will not be surprised to hear that as systems become more complex,[172] black-box solvers struggle to provide good solutions.

[172] higher dimensional, nonlinear, and even non-differentiable...