Continuous Time Optimal Control Problem

\[
\begin{align*}
\text{minimize} & \quad \phi(x(t_f)) + \int_0^{t_f} f_0(t, x(t), u(t)) dt \\
\text{subj.to} & \quad \dot{x}(t) = f(t, x(t), u(t)) \\
& \quad x(0) = x_0, \quad x(t_f) \in S_f, \quad u(t) \in \mathbb{R}^m
\end{align*}
\]

where

\[S_f = \{x \in \mathbb{R}^n \mid G(x) = 0\}, \quad G(x) = \begin{bmatrix} g_1(x) \\ \vdots \\ g_p(x) \end{bmatrix}\]

and all functions are twice continuously differentiable w.r.t. \(x\) and \(u\).

Representing functions

- \(x(t), u(t), \lambda(t)\) are functions on an interval
- In a computer only finitely many numbers can be stored
- There are two ways out:
  - Convert to a discrete time problem, e.g.
    \[x(t + h) = x(t) + hf(x(t), u(t))\]
  - Preserve the continuous nature of the problem but represent \(x, u\) and \(\lambda\) with a finite number of parameters, e.g. consistent approximations
**Consistent Approximations**

Instead of taking the control signal to be piecewise constant, take it as

\[ u(t) = \sum_{k=1}^{N} \mu_k \varphi(t)_k \]

where \( \varphi(t) \) is a scalar spline function or orthogonal basis function. Common choices of basis are Fourier series, Chebyshev series, Legendre series, and Walsh series.

This also results in finite dimensional optimization problems, but where gradients can be computed by numerically solving differential equations and computing integrals.

There are several professional software packages developed based on this method, e.g. Riots, Miser3, and PROPT.

---

**Boundary Conditions**

Assume that \( G \) and \( \phi \) depend on disjoint set of variables:
- \( G(x) = G(x_1, \ldots, x_p) \) (same \( p \) as number of components in \( G \))
- \( \phi(x) = \phi(x_{p+1}, \ldots, x_n) \)

Then boundary conditions for TPBVP at \( t = t_f \) are

\[
G(x_1(t_f), \ldots, x_p(t_f)) = 0; \quad \begin{bmatrix} \lambda_{p+1}(t_f) \\ \vdots \\ \lambda_n(t_f) \end{bmatrix} = \begin{bmatrix} \frac{\partial \phi(x(t_f))}{\partial x_{p+1}} \\ \vdots \\ \frac{\partial \phi(x(t_f))}{\partial x_n} \end{bmatrix}
\]

---

**Repetition: How to use PMP**

1. Define the Hamiltonian: \( H(t, x, u, \lambda) = f_0(t, x, u) + \lambda^T f(t, x, u) \)
2. Perform pointwise minimization: \( u(t) = \arg\min_{u \in U} H(t, x, u, \lambda) \); equivalent to \( H_u(t, x, u, \lambda) = 0 \) when \( U = \mathbb{R}^n \).
3. Solve TPBVP

\[
\begin{align*}
\dot{\lambda}(t) &= -H_x(t, x(t), u(t), \lambda(t)) \\
\dot{x}(t) &= H_u(t, x(t), u(t), \lambda(t)), \quad x(0) = x_0
\end{align*}
\]

with suitable boundary conditions at \( t = t_f \), which depend on problem setup.
4. Compare the candidate solutions obtained using PMP.

---

**Boundary Condition Iteration (Shooting)**

The problem is to find initial condition \( \lambda(0) = \lambda_0 \) such that

\[
0 = \mu(x(t_f), \lambda(t_f)) = \begin{bmatrix} G(x_1(t_f), \ldots, x_p(t_f)) \\ \lambda_{p+1}(t_f) - \frac{\partial \phi(x(t_f))}{\partial x_{p+1}} \\ \vdots \\ \lambda_n(t_f) - \frac{\partial \phi(x(t_f))}{\partial x_n} \end{bmatrix}
\]

The algorithm updates \( \lambda_0 \) such that the change \( \delta \mu(t_f) \) in \( \mu(t_f) \) is \( \delta \mu(t_f) = -\alpha \mu(t_f) \) or equivalently

\[
\frac{\partial \mu(t_f)}{\partial \lambda_0} \delta \lambda_0 = -\alpha \mu(t_f)
\]
Shooting Algorithm

1. Make an initial guess $\lambda(0) = \lambda_0$
2. Integrate the system
   \[
   \dot{\lambda}(t) = -H_u(t, x(t), u(t), \lambda(t)), \quad \lambda(0) = \lambda_0
   \]
   \[
   \dot{x}(t) = H_{\dot{x}}(t, x(t), u(t), \lambda(t)), \quad x(0) = x_0
   \]
   forward in time where $u(\cdot)$ is chosen such that
   \[
   H_u(t, x(t), u(t), \lambda(t)) = 0
   \]
3. Compute $\mu(t_f) = \mu(x(t_f), \lambda(t_f))$.
4. Update $\lambda_0 := \lambda_0 + \delta \lambda_0$, where
   \[
   \frac{\partial \mu(t_f)}{\partial \lambda_0} \delta \lambda_0 = -\alpha \mu(t_f).
   \]
   Repeat steps 2 to 4 until $\|\mu(t_f)\|$ is sufficiently small.

Remark: The transition matrix $\frac{\partial \mu(t_f)}{\partial \lambda_0}$ can be computed either using numerical differentiation or linearization.

Gradient Methods

Consider case when no terminal constraint. First order variation:
\[
\delta J = (\phi_x(x^*(t_f)) - \lambda(t_f))^T \delta x(t_f) + \int_0^{t_f} H_u(t, x^*(t), u^*(t), \lambda(t))^T \delta u(t) dt
\]
\[
+ \int_0^{t_f} \left[ H_{\dot{x}}(t, x^*(t), u^*(t), \lambda(t)) + \dot{\lambda}(t) \right]^T \delta x(t) dt
\]
It can be shown that the gradient of the cost function with respect to $u$ is
\[
H_u(t, x(\cdot), u(\cdot), \lambda(\cdot))
\]
as long as $\lambda(\cdot)$ satisfies the adjoint equation.

Pros and Cons for Shooting Methods

(+) Conceptually simple. It was used to launch satellites in the 1950s

(+) Control constraints are easy to deal with. The only modification is that in step 2 we must compute the control pointwise from
\[
\min_{v \in U} H(t, x(t), v, \lambda(t)).
\]

(-) It can be crucial to find a good initial estimate of $\lambda_0$.

(-) Integration of the system may be severely unstable.

Gradient Algorithm

1. Guess $u(t), 0 \leq t \leq t_f$
2. Integrate $\dot{x}(t) = f(t, x(t), u(t)), x(0) = x_0$ forward in time.
3. Integrate the adjoint equation $\dot{\lambda}(t) = -H_x(t, x(t), u(t), \lambda(t)), \lambda(t_f) = \phi_x(x(t_f))$ backward in time.
4. Update $u(t) := u(t) - \alpha H_u(t, x(t), u(t), \lambda(t))$, where $\alpha$ is the step length.

Repeat steps 2 to 4 until $\int_0^{t_f} \|H_u(t, x(t), u(t), \lambda(t))\| dt$ is sufficiently small.
Pros and Cons for Gradient Methods

(+) It gives good improvement in the first iterations.
(+) Stability is generally improved compared to the shooting method since the integration of \( x(\cdot) \) and \( \lambda(\cdot) \) is performed in the stable direction.
(+) Control constraints can be taken into account by projecting onto the control constraint set.
(+) It was used to solve a large number of aeronautical problems in the 1960s.
(-) Convergence tends to be slow.

Riccati Equation

\[ \dot{P} + PA + A^T P + Q = (PB + S)R^{-1}(PB + S)^T, \quad P(t_f) = \Phi_{xx}(x(t_f)) \]

and linear differential equation

\[ \dot{q} + A^T q = (PB + S)R^{-1}(B^T q + r), \quad q(t_f) = 0 \]

where

\[
\begin{align*}
A(t) &= f_x(t, x(t), u(t)) \\
B(t) &= f_u(t, x(t), u(t)) \\
Q(t) &= H_{xx}(t, x(t), u(t), \lambda(t)) \\
S(t) &= H_{xu}(t, x(t), u(t), \lambda(t)) \\
R(t) &= H_{uu}(t, x(t), u(t), \lambda(t)) \\
r(t) &= H_u(t, x(t), u(t), \lambda(t))
\end{align*}
\]

Newton’s Method

The idea is to iteratively locally approximate the Lagrangian with a second order Taylor expansion and the dynamic constraints with a first order Taylor expansion (linearization), and then solve the resulting approximate Linear Quadratic Control problem to get a step.

\[
\begin{align*}
\text{minimize} \quad & \quad \delta x(t_f)^T \Phi_{xx}(x(t_f)) \delta x(t_f) + \int_0^{t_f} \left( 2H_u^T \delta u + \left[ \begin{array}{c} \delta x^T \\ \delta \lambda^T \\ \delta u \end{array} \right] \left[ \begin{array}{ccc} H_{xx} & H_{xu} & \delta \lambda \\ H_{xu} & H_{uu} & \delta u \end{array} \right] \left[ \begin{array}{c} \delta x \\ \delta \lambda \\ \delta u \end{array} \right] \right) dt \\
\text{subject to} \quad & \quad \dot{\delta x}(t) = f_x \delta x + f_u \delta u, \quad \delta x(0) = 0
\end{align*}
\]

Newton’s Algorithm

1. Guess \( u(t), 0 \leq t \leq t_f \).
2. Integrate \( x(t) = f(t, x(t), u(t)), x(0) = x_0 \) forward in time.
3. Integrate the adjoint equation \( \dot{\lambda}(t) = -H_x(t, x(t), u(t), \lambda(t)), \quad \lambda(t_f) = \Phi_{xx}(x(t_f)) \) backward in time.
4. Solve for \( P(\cdot) \) and \( q(\cdot) \) on the previous frame.
5. Update

\[
u_{new} := u_{old} - R^{-1}[(B^T P + S^T)(x_{new} - x_{old}) + (r + B^T q)],
\]

where \( R \in (0, 1) \) is the step length.

Repeat steps 2 to 5 until \( \|H_u(t, x(t), u(t), \lambda(t))\| \) is sufficiently small.
Pros and Cons for Newton’s Method

(+) Fast convergence
(+) Solid theoretical justification
(-) Good initial guess is needed. Can be achieved by using gradient method initially
(-) Each iteration is computationally quite expensive