轨迹优化方法与实践

刘成刚 (Frank Liu)

December 26, 2019

Contents

1 本文初衷

本文是对实践的一个总结，主要面向的是从事无人驾驶的工程技术人员和研究人员，旨在基于大数据的深度学习被广泛关注的今天，让更多的人熟悉和了解轨迹优化技术。轨迹优化技术与深度神经网络有非常紧密的联系[1, 2]，因此两者可以相互借鉴和发展。本文提到的方法不仅适用于无人驾驶的运动规划和控制问题，同时也适用于复杂动力学系统，如足式机器人、仿生机器人和飞行器的运动规划和控制问题。

2 概述

无人驾驶技术可以减少车祸发生，降低交通成本，让出行不便的人重获生活上的自由，因此具有巨大的市场价值。另一方面，对无人驾驶技术的投入将大大降低传感器和计算硬件的价格，促进人工智能算法的发展，从而带动整个机器人产业的发展并产生更加深远的社会影响。

[1] Neural Ordinary Differential Equations, Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, David Duvenaud

[2] Deep Learning Theory Review: An Optimal Control and Dynamical Systems Perspective, G.H. Liu, Evangelos, A. Theodorou

在无人驾驶系统的设计中经常会遇到以下一些极富挑战的问题：

1. 道路在线检测问题：在图像中检测到不连续的道路标志，如何确定道路的边界。
2. 其他车辆的运动预测问题：已知其他车辆的运动速度、朝向和所在的道路，如何确定其未来一段时间内的轨迹。
3. 隐含状态估计问题：已知车辆的位置、速度和朝向随时间变化的信息，如何确定其加速度、方向盘转角等不能直接观测的信息。
4. 地图路径平滑问题：已知道路中线，如何生成高阶平滑的轨迹。
5. 执行轨迹的生成问题：已知道路边界，其他车辆和行人的未来运动轨迹和所遵守的交通规则，如何确定车辆自身的执行轨迹。
6. 轨迹跟踪问题：已知车辆自身当前的状态和未来执行轨迹，如何生成油门和方向盘的控制指令？
7. 模仿学习问题：如何通过大量的人工驾驶数据，通过机器学习算法提高系统的性能。

这些看似互不相关的问题，其背后却有许多相似的实质，轨迹优化方法为这些问题提供了行之有效的解决方案。

3 轨迹优化方法

3.1 概述

轨迹优化方法是在满足特定约束的前提下寻找使评价函数最小（或最大）的轨迹的方法。轨迹优化源于变分法 (Calculus of Variations)。轨迹可以表示为系统状态变量和控制变量的函数，而评价函数是轨迹函数的函数，变分法是在当前轨迹附近进行轻微的扰动寻找使轨迹代价下降的方法。在变分法中一个最重要的结论是欧拉-拉格朗日方程 (Euler-Lagrange)。欧拉-拉格朗日方程给出了轨迹最优的必要条件。充分理解欧拉-拉格朗日方程对于充分理解和灵活应用轨迹优化非常有帮助。

严格地讲轨迹优化解决的是以下一类优化问题：

$$\min_{x,u} \phi(x(t_f)) + \int_{t_0}^{t_f} l(x(t), u(t), t) dt$$

s.t.

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

$$x(t_0) - x_0 = 0$$

$$h(x, u) \leq 0$$

其中 x 是系统的状态变量，如车辆的位置和速度，u 是系统的控制变量，如油门大小和方向盘的转角，f 是微分形式的系统动力学方程，描述了系统的状态的变化与当前状态和控制输入之间的关系。

根据变分法，无约束情况下轨迹最优的必要条件是：

$$l_x + \lambda^\top f_x + \dot{\lambda}^\top = 0$$

$$l_u + \lambda^\top f_u = 0$$

$$\phi_x(x(t_f)) - \lambda^\top(t_f) = 0$$

其中 λ 称为系统的协状态。

轨迹优化跟最优控制有紧密的联系。两者的区别在于：最优控制通常研究的是状态反馈控制律，即对于任意指定的系统状态计算对应的最优控制输出；而轨迹优化研究的是从某个指定状态开始的最优控制序列。两者的联系在于：动力学系统在最优反馈控制律的作用下形成向量场，最优轨迹是最优控制向量场中起始于某个特定状态的流。如果我们知道了最优状态反馈控制律就可以得到起始于任意状态的最优轨迹；反之，如果我们得到了始于任意状态的最优轨迹，就能够通过这些最优轨迹重构最优反馈控制律 [3,4]。也可以通过在线优化构成反馈控制律[5]，或者将离线规划的结果与在线优化相结合构成反馈控制律[6]

根据求解的过程，轨迹优化通常可分为间接方法和直接方法：

- 间接方法：依赖 Pontryagin 最大值原理，将轨迹优化问题转化为两点边值问题，然后利用求解两点边值的方法求解最优轨迹，最后对计算结果离散化。间接法在求解的最后进行离散化，因此结果的精度很高，但缺点是不容易收敛。

- 直接方法：直接方法首先将原问题离散化，从而将无限维连续优化问题转化为有限参数的非线性优化问题，然后利用常规的非线性优化方法求解最优轨迹。直接法在一开始对原问题进行离散化，其结果会存在一定的近似误差，但优点是比较容易收敛。

通常我们将把无限维连续轨迹优化问题转化为常规非线性优化问题的过程称为改编(transcription)。根据改编方法的不同，轨迹优化方法又可大致分成打靶法和配点法，两者的主要不同在于对状态方程的处理。

- 打靶法（shooting method）：打靶法只优化控制变量，状态变量和协状态通过对状态方程前向积分得到。基本过程是首先采用初始的控制轨迹 $u(0,\ldots,N-1)$ 生成初始的状态轨迹，根据末端边值误差 $e_f \triangleq \lambda(N) - V_x(x(N))^\top$ 采用非线性优化方法求解 $\lambda(0)$ 使得 e_f 接近于零。打靶法的优点是能够得到非常精确的最优轨迹。打靶法适用于控制变量的维数较低且初始估计比较准确的应用场合。打靶法存在的主要问题是对于有强约束或者动力学不稳定的系统优化不容易收敛。这种情况下可以采用多目标打靶法将轨迹分成若干轨迹片段，然后在每条轨迹片段上反复采用打靶法，直到所有断点满足连续条件。

[3] Random sampling of states in dynamic programming, C. G. Atkeson and B. J. Stephens, , in IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 38, no. 4, pp. 924-929, Aug. 2008. doi: 10.1109/TSMCB.2008.926610

[4] Trajectory-Based Dynamic Programming. Atkeson C.G., Liu C., In: Mombaur K., Berns K. (eds) Modeling, Simulation and Optimization of Bipedal Walking. Cognitive Systems Monographs, vol 18. Springer, Berlin, Heidelberg

[5] `https://en.wikipedia.org/wiki/Model_predictive_control`

[6] Biped walking control using offline and online optimization, C. LIU and J. SU, China Control Conference (CCC), 2011

- 配点法（collocation）：配点法采用条函数（spline）近似状态轨迹和控制轨迹，然后采用常规的非线性优化方法（NLP）求解样条函数的参数。伪谱方法（pseudospectral method）采用全局多项式函数近似状态轨迹和控制轨迹，系统状态方程约束由正交配置法近似，从而将原问题参数化。主要的伪谱方法有勒让德伪谱方法 (the Legendre pseudospectral method) 和切比雪夫伪谱方法(the Chebyshev pseudospectral method)。

除此之外，还有很多其他轨迹优化方法，例如 CHOMP 和 DDP 等。每种轨迹优化方法都有其自身的特点，适用于不同的场合。没有适合所有应用的最好的轨迹优化方法，实践中要根据对精度、对实时性的要求和优化问题的结构灵活选择。在非线性不强，对实时性要求较高的场合，微分动态规划（Differential Dynamic Programming）提供了一种行之有效的方法，下面重点介绍一下。微分动态规划方法大致属于直接打靶法。但和上面介绍的方法略有不同的是它不明确的区分改编和优化，而是沿轨迹反复地做前向和反向积分从而满足系统的动力学方程和轨迹最优的必要条件。微分动态规划法有接近二次型的收敛速度。此外，微分动态规划方法在收敛的同时能够返回最优控制律的一阶近似和值函数的二次阶近似，在很多分析中非常有用。

3.2 有限时域离散状态轨迹优化问题

为了简单起见，我们首先来介绍有限时域离散状态轨迹优化问题:

$$U, X = \arg\min_{X,U} \left\{ \phi(x_N) + \sum_{i=0}^{N-1} L(x_i, u_i, i) \right\}$$

s.t.

$$x_{i+1} = F(x_i, u_i, i)$$

$$x_0 = \bar{x}_0$$

在介绍微分动态规划法之前，需要首先介绍一下贝尔曼方程和动态规划法（dynamic programming）。

3.3 贝尔曼方程

最优性原理（也称为嵌入原理）：不论一个问题的初始状态为何，以最优控制所形成的状态作为初始条件，余下的控制对余下的问题也必构成最优策略:

$$u^*(i) = \arg\min_{u_i} \left\{ L(x_i, u_i, i) + \min\{ \sum_{k=i+1}^{N-1} L(x_k, u_k, k) + \Phi(x_N) \} \right\}$$

在最优控制中，值函数是一个被提到的函数。对于有限时域轨迹优化问题，值函数是关于系统状态 x 和阶段 i 变化的时变函数:

$$V(x_i, i) = \min_{u_i} \left\{ \sum_{k=i}^{N-1} L(x_k, u_k, k) + \Phi(x_N) \right\}$$

对于无限时域问题，值函数是关于系统状态 x 的稳态函数:

$$V(x_i) = \min_{u_i} \Big\{ \sum_{k=i}^{\infty} L(x_k, u_k, k) \Big\}$$

定义了值函数以后，贝尔曼方程可描述为如下的嵌套形式:

$$V(x_i, i) = \min_{u_i} \big\{ L(x_i, u_i, i) + V(F(x_i, u_i), i+1) \big\}$$

3.4 动态规划法

动态规划采用分而治之的思路将多步或长期优化问题通过贝尔曼方程转化为单步或短期优化问题，从而将指数复杂度降低为多项式复杂度。求解动态规划的最常用的方法是“值迭代”法（Value iteration）。对于有限时域的最优控制问题，其最后阶段的值函数通常是已知的，即:

$$V(x_N, N) = \Phi(x_N, N)$$

其他阶段的值函数可以根据贝尔曼方程反向迭代得到:

$$V(x_i, i) = \min_{u} \big\{ L(x_i, u, i) + V(F(x_i, u_i), i+1) \big\}$$

在数值计算中，需要对值函数 $V(x_i, i)$ 进行近似，通常采用的方法是表格法（Tabular method）。表格法对状态空间进行离散化，然后对网格点处的状态运用贝尔曼方程求解值函数。在迭代的过程中，非网格点处的值函数通过函数差插值得到。

已知值函数，最优状态反馈控制律可以通过下面的方程得到:

$$u^*(x, i) = \arg\min_{u} \big\{ L(x, u, i) + V(F(x, u), i+1) \big\}$$

对于轨迹优化问题，可以采用得到的最优状态反馈控制律和系统状态方程生成起始于任意指定状态的最优轨迹。

如果状态变量的量化级数为 R_x ，控制变量的量化级数为 R_u ，状态变量的维数为 d_x ，控制变量的维数为 d_u ，那么动态规划的计算量将正比于 $R_x^{d_x} R_u^{d_u}$ ，而存储量正比于 $R_x^{d_x}$ 。动态规划方法的计算量和存储量随着状态变量维数成指数增长的问题称为“维数灾”问题。“维数灾”问题的存在限制了其在实际中的应用。

3.5 微分动态规划方法

在动态规划法中我们需要对整个状态空间的值函数进行近似，因此存在“维数灾”问题。为了克服这个问题，微分动态规划只对当前轨迹的邻域内的值函数进行近似，从而得到临域最优反馈控制律。通过临域最优反馈控制律不断更新轨迹，不断逼近最优轨迹。

设当前轨迹上一点的状态和控制分别为 x_i 和 u_i ，对当轨迹做很小的绕动 $x = x_i + \Delta x_i$ ， $u = u_i + \Delta u_i$ 。

定义一个中间函数: $Q(x, u) = L(x, u) + V(F(x, u))$ 。根据贝尔曼方程，阶段 i 的最优控制和值函数可以通过最小化 Q 函数得到。Q 函数的二阶近似可以表示为:

$$\begin{aligned} Q(x, u) = {} & Q(x_i, u_i) + Q_x(x_i, u_i)\Delta x_i + Q_u(x_i, u_i)\Delta u_i + \\ & \Delta x_i^\top Q_{xu}(x_i, u_i)\Delta u_i + \frac{1}{2}\Delta x_i^\top Q_{xx}(x_i, u_i)\Delta x_i + \frac{1}{2}\Delta u_i^\top Q_{uu}(x_i, u_i)\Delta u_i \end{aligned}$$

使得 Q 函数最小的控制扰动 Δu 为：

$$\Delta u_i^* = -Q_{uu}(i)^{-1}Q_u(i) - Q_{uu}(i)^{-1}Q_{ux}(i)\Delta x_i \tag{1}$$

阶段 i 的值函数在当前轨迹临域附近可以近似为：

$$\begin{aligned} V(x) &= \min_{\Delta u_i} Q(x_i + \Delta x_i, u_i + \Delta u_i) \\ &= Q(x_i + \Delta x_i, u_i + \Delta u_i^*) \\ &= Q(x_i, u_i) - \frac{1}{2}Q_u Q_{uu}^{-1} Q_u^\top + (Q_x - Q_u Q_{uu}^{-1} Q_{ux})\Delta x_i + \frac{1}{2}\Delta x_i^\top (Q_{xx} - Q_{xu} Q_{uu}^{-1} Q_{ux})\Delta x_i \end{aligned} \tag{2}$$

Q 函数的各阶导数如下：

$$Q_x(i) = L_x(i) + V_x(i+1)F_x(i) \tag{3}$$
$$Q_u(i) = L_u(i) + V_x(+1)F_u(i) \tag{4}$$
$$Q_{xx}(i) = L_{xx}(i) + F_x(i)^\top V_{xx}(i+1)F_x(i) + V_x(i+1)F_{xx}(i) \tag{5}$$
$$Q_{uu}(i) = L_{uu}(i) + F_u(i)^\top V_{xx}(i+1)F_u(i) + V_x(i+1)F_{uu}(i) \tag{6}$$
$$Q_{ux}(i) = L_{ux}(i) + F_u(i)^\top V_{xx}(i+1)F_x(i) + V_x(i+1)F_{ux}(i) \tag{7}$$
$$\tag{8}$$

其中 $F_x(i) := F_x|_{x_i,u_i}$, $F_u(i) := F_u|_{x_i,u_i}$, $F_{xx}(i) := F_{xx}|_{x_i,u_i}$, $F_{uu}(i) := F_{uu}|_{x_i,u_i}$, $F_{xu}(i) := F_{xu}|_{x_i,u_i}$, $L_x(i) := L_x|_{x_i,u_i}$, $L_u(i) := L_u|_{x_i,u_i}$, $L_{xx}(i) := L_{xx}|_{x_i,u_i}$, $L_{xu}(i) := L_{xu}|_{x_i,u_i}$, $L_{uu}(i) := L_{uu}|_{x_i,u_i}$, $V_x(i) := V_x(x)|x = x_i$, $V_{xx}(i) := V_{xx}(x)|_{x=x_i}$ 。

$$V_x(i) := V_x(x)|_{x=x_i} = Q_x(i) - Q_u(i)Q_{uu}(i)^{-1}Q_{ux}(i) \tag{9}$$
$$V_{xx}(i) := V_{xx}(x)|_{x=x_i} = Q_{xx}(i) - Q_{xu}(i)Q_{uu}(i)^{-1}Q_{ux}(i) \tag{10}$$

边界条件为：

$$V(N) = \Phi(N) \tag{11}$$
$$V_x(N) = \Phi_x(N) \tag{12}$$
$$V_{xx}(N) = \Phi_{xx}(N) \tag{13}$$

微分动态规化法可以描述为：

1. 根据控制变量的初始值，通过前向积分得到系统的状态轨迹。
2. 反向更新系统的值函数估计并计算临域最优反馈控制律。
3. 利用临域最优控制生成新的轨迹。
4. 计算新轨迹的代价，如果满足退出条件则退出；否则，如果代价的下降满足下降预期，返回步骤 2 并重复；否则算法报错。

因为上面得到的最优控制律是基于值函数近似得到的，所以在更新控制的时候需要选取一定的下降步长：

$$\Delta u_i^* = -\alpha Q_{uu}(i)^{-1} Q_u(i) - Q_{uu}(i)^{-1} Q_{ux}(i) \Delta x_i$$

在求解的过程中，可以采用置信域（trust region）方法或者线性回溯搜索（backtracking line search）方法[7]。线性回溯搜索法是非线性优化中常用的一种方法，其基本思想是首先采用一个较大的步长，然后对生成的轨迹进行评估。如果当前轨迹的代价没有达到下降预期，减小搜索步长和下降预期，重复上面的过程直到代价的实际下降达到或超过下降预期为止，否则算法报错。根据方程 2，代价函数的最大下降预期为：

$$\Delta Q = \frac{1}{2} Q_u Q_{uu}^{-1} Q_u^\top$$

微分动态规划法需要用到代价函数的 Hessian 矩阵，虽然在理论上能够提高收敛的速度，但是在实践中因为 Hessian 矩阵的维数通常比较高，计算比较耗时，有时反而会降低求解速度。如果系统的状态方程非线性不强，代价函数接近二次型，可以对其做进一步简化，采用一阶模型近似动力学方程，采用二次型近似代价函数，从而得一种新的优化方法，迭代线性二次型方法（iLQR）。iLQR 与 Gauss-Newton 方法类似，为保证收敛，在应用中需要注意以下两点：

1. 代价函数在最优轨迹附近趋近于零。

2. 代价函数的非线性不能太强。

DDP 要求在迭代的过程中 Hessian 矩阵必须是正定的。如果在求解的过程中 Hessian 出现非正定，可以采用 Trust region 方法或者 Levenberg-Marquardt 方法[8, 9]。Levenberg-Marquardt 方法是一种在梯度下降方法和牛顿方法之间进行自适应的方法，即如果代价下降，减小阻尼系数使得算法趋向于牛顿方法；否则提高阻尼系数使算法趋向于梯度方法。如果将函数的优化过程看成是动力学系统的镇定控制过程，Levernberg-Marquardt 方法非常类似于变增益控制，两者之间存在很多有趣的联系值得我们去探索。

3.6 在应用中需解决的问题和方法

1. 初始轨迹的生成问题: 轨迹优化是一种局部优化方法，初始轨迹的选取对于优化的结果和收敛的速度至关重要。常用的初始轨迹生成方法有：

    - 反馈控制法：在实际中常常会遇到使系统稳定于某个指定的状态的控制和规划问题，例如倒立摆摆起控制、仿人机器人的站立平衡控制和泊车控制等。对于这类问题，一个常用的方法是对系统在稳定状态附近线性化，设计线性二次型控制器（LQR），然后利用 LQR 控制器生成初始轨迹。另一类常见的问题是跟踪问题。例如在无人驾驶的车道内的控制问题（in-lane control），可以通过路径跟踪控制器来生成初始轨迹，如 pure pursued controller, gain scheduling controller, Stanley controller 等。

[7] https://en.wikipedia.org/wiki/Backtracking_line_search

[8] http://people.duke.edu/~hpgavin/ce281/lm.pdf

[9] http://www.applied-mathematics.net/LMvsTR/LMvsTR.pdf

- 由简入繁法：将要解决的问题进行简化，去掉计算耗时或者非线性很强代价函数和约束条件，然后对简化后的问题进行优化从而得到初始轨迹。例如，在无人驾驶中车道内的控制问题可以忽略所有运动目标和边界约束，对静止的交通环境进行优化，将优化的结果作为初始轨迹。
- 优势互补法：每一种轨迹优化方法都有其自身的优缺点，在实际应用中可以将多种方法结合从而达到最好的效果。例如直接配点法的优势在于约束处理的能力比较强，对非线性比较强的系统收敛性比较好。因此在 DDP 不能收敛的情况可以考虑于直接配点法结合。在结构复杂的机器人的控制中，有些问题的维数高、约束复杂，找到可行解本身就非常难，此时可以考虑与随机采样方法结合，如 RRT [10], Probabilistic roadmap [11, 12]。
- 控制律重用法：上面介绍的几种生成初始轨迹的方法属于“冷启动”方法，控制律重用法则属于一种“热启动”方法。轨迹优化经常用于实现模型预测控制（Model Predictive Control）或滚动优化（Receding Horizon Optimization），在上述应用中通常我们有上一步的优化结果，如果每一步之间代价函数的变化不大，可以采用上次优化得到的临域最优控制律 (neighboring optimal control law) 生成初始轨迹。如果初始状态距离上一次的最优轨迹很近，而且代价函数和上次相比变化不大，这种方法可以大大提升优化的收敛速度。例如在无人驾驶的应用中采用此方法我们将计算时间缩短近 30% 。不过需要注意的是虽然该方法能够大大提升 best case 的计算速度，却会降低 worst case 的计算速度，在某些应用中往往 worst case 的性能更为重要，此时需要考虑与其他方法结合来解决这个问题。

2. 错误局部最优的规避问题:

   轨迹优化方法是一种局部优化方法，在实际应用中可能会收敛到不期望的局部最优，例如在自动驾驶应用中，车辆可能在错误的一侧躲避障碍。避免”错误“的局部最优解的方法有：

   - 代价函数法：在设计代价函数的时候尽量减少不希望的局部最优，例如，在求取点到线的距离的时候应该采用有符号距离。在设计代价函数的时候还可以通过增加 regularization term 和 shaping terms 来避免错误收敛的发生。
   - 初始轨迹法：如果代价函数存在多个局部最优，如车辆在躲避障碍物的时候可以从两侧中的任意一侧通过，应该保证初始轨迹在期望一侧。
   - 约束加强法：很多收敛问题是约束造成的，即系统无法跨越约束造成的障碍，最终收敛到了“错误”的局部最优。此时可以采用的方法是约束加强法，即在优化的初始阶段将约束松弛，然后在优化的过程中逐渐加强并最终满足所有约束。

3. 实时性问题：实时性是轨迹优化在应用中需要考虑的最核心问题之一。近年来在线优化方法被越来越多地用于解决控制和规划问题，一个最重要的原因是优化算法实时性得到了很大的提高。提高算法实时性需要较多的经验，例如针对不同的问题和要求，选择合适的模型、代价函数、时域长度、优化步长、约束处理方法、初始轨迹

[10] https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree

[11] https://en.wikipedia.org/wiki/Probabilistic_roadmap

[12] Full-body motion planning and control for the car egress task of the DARPA robotics challenge, C. Liu, C. G. Atkeson, Siyuan Feng, and X Xinjilefu, 2015 humanoid Robots

生成方法和函数近似方法等。不同的建模和实现方式，即使是同样一种算法的性能也会有几个数量级的差别。例如在无人驾驶的应用中，通过对算法和实现方法的优化我们将收敛时间降低了近 80%。

提高优化算法的实时性的技巧大致归纳如下：

- 提高初始轨迹的质量，例如控制律重用法。
- 软件实现中采用数据导向的编程方法（Data-orientated Programming），指导思想是将跟计算相关的数据存放在相邻地址，避免内存动态分配，缓存尽量多的中间变量，充分利用 CPU 和缓存实现高效的流水操作。
- 避免代价函数中的非二次型成分，例如线性代价函数的引入会导致收敛速度从二阶收敛退化为一阶收敛。
- 代价函数分层计算，例如在线性搜索的过程中，可以首先计算计算量较小的代价从而可以对当前的轨迹进行快速评估以避免不必要的计算。
- 选择合适的模型：通过一定的方法，在保证精度的同时尽量减少模型的复杂度和非线性。
- 在轨迹优化中计算量最大的是常微分方程积分，在应用中可以在积分精度和计算速度间进行折中，选择合适的积分方法。
- 时间步长和时域的选取：在时域不变的情况下，增加计算步长可以减少优化的阶段数，提高收敛速度。但是大步长会降低结果的质量，降低满足约束的精度，并更容易出现“震铃”现象（在平衡点附近反复震荡）。一种有效的方法是采用变步长方法，近处用小步长，远处用大步长。
- 优化时域的选取：缩短优化时域长度可以等比例地缩短计算时间，但同时会降低结果的最优性。在模型预测控制中，可能会出现低频震荡问题，如在无人驾驶的轨迹生成中可能会产生摇摆问题。一种可行的方法是通过提高终端代价函数的质量，在缩短时域长度的同时，不降低优化结果的质量。
- 约束的处理。微分动态规划方法主要用来处理无约束的轨迹优化问题，在应用中可以通过惩罚函数法近似地处理约束。通常状态约束比控制约束更难处理。在算法设计的过程中，可以采用不同的方法对两种约束进行处理，从而极大地提升优化收敛速度。另外，如果初始轨迹严重违反约束，可以通过约束加强法使算法更加稳定、收敛更快。
- 梯度和 Hessian 矩阵的计算：为了提高计算速度，应该为算法提供梯度和 Hessian 矩阵，避免使用有限微分法在线计算。
- 并行计算和硬件加速：线性搜索、ODE 积分和值函数更新等都可以采用硬件加速。
- 代码深度优化：在计算的过程中有大量的稀疏矩阵运算，可以将算法对特定问题做深度优化，从而避免不必要的计算和内存开销。
- 制定正确的优化退出条件：因为优化算法本身对原问题的进行了很多近似，可以通过制定合适的优化退出条件提前退出迭代，从而换取更高的运行速度。虽然可能牺牲了一定的最优性，但在实时控制中有时反而会带来更大的性能提升。

3.7 总结

我们首先介绍了轨迹优化的定义，分类及与最优控制之间的关系。然后介绍了最优控制中一种全局方法（动态规划方法）和局部方法（微分动态规划法）。最后介绍了轨迹优化在实际应用中会遇到的问题和解决方法。在下一章，我们将介绍迹优化的概率解释，从而建立起轨迹优化与机器学习之间的联系。