

CMU 16-745 最优控制笔记

吕翔宇 (Xiangyu Lyu)
lvxiangyu11@gmail.com
TU Darmstadt, Germany
GitHub: lvxiangyu11

2025 年 7 月 31 日

目录

1 引言	6
2 动力学简介	7
2.1 连续时间的动力学系统	7
2.1.1 控制映射系统	9
2.1.2 Manipulator动力学	9
2.1.3 线性系统	9
2.1.4 平衡点 equilibria	10
2.1.5 平衡点的稳定性分析	10
2.2 离散时间的动力学系统	13
2.2.1 正向欧拉法	13
2.2.2 RK4方法	17
2.2.3 反向欧拉法	23
2.2.4 离散控制输入	24
2.3 接触动力学仿真	25
3 数值优化基础	26
3.1 符号声明	26
3.2 Root Finding问题	27
3.2.1 Newton 方法	29
3.2.2 牛顿方法在最小化问题中的应用	33
3.3 最小值存在的充分条件	36
3.4 正则化 (Regularization) 与强凸性	36
3.5 线性搜索 (Line Search) Backtracking方法	38
4 约束优化问题	40
4.1 等式约束问题	40
4.1.1 一阶必要条件	40
4.1.2 Gauss-Newton 方法的应用	42
4.2 不等式约束	45
4.2.1 一阶必要条件	45
4.2.2 直观理解	45
4.3 算法	45
4.3.1 活动集方法 (Active-Set Method)	45
4.3.2 障碍法 / 内点法 (Barrier / Interior-Point Method)	46
4.4 惩罚方法 / 外点法 (Penalty / Exterior-Point Method)	46
4.4.1 外部惩罚方法	47
4.4.2 增广拉格朗日方法 (Augmented Lagrangian Method)	47
4.4.3 二次规划示例 (Quadratic Program Example)	50
4.5 正规化和对偶	51
4.6 Merit Functions (for line search)解决超调问题	54
4.6.1 约束最小化的线搜索	55
5 最优控制	57
5.1 最优控制的历史回顾	57
5.2 变分法 Calculus of Variations	57
5.3 确定性最优控制 Deterministic Optimal Control	59

5.3.1 离散时间 Discrete Time	60
5.3.2 Pontryagin 最小值原理	60
5.4 线性二次调节器 Linear Quadratic Regulator (LQR)	65
5.4.1 间接射击法下的LQR（不推荐）	66
5.4.2 例子：双积分器系统	68
5.5 LQR 作为二次规划 (QP) 问题	71
5.5.1 例子	72
5.5.2 Riccati 递推：LQR QP 的结构	74
5.6 无限时域 LQR (Infinite Horizon LQR)	77
5.6.1 有限时域 LQR	77
5.6.2 无限时域 LQR	77
5.7 如何求解LQR控制律 (How to solve the policy)	77
5.7.1 非线性系统的扩展	78
5.8 可控性 (Controllability)	78
5.8.1 可控性 Gramian	79
5.8.2 Gramian 与可控性的关系	79
5.8.3 Gramian 的 SVD 分解与可控性方向	80
5.8.4 方向可控性的量化	80
5.8.5 计算举例	80
6 动态规划	81
6.1 贝尔曼原理	81
6.2 动态规划	81
6.2.1 动态规划算法	82
6.2.2 连续空间连续时间的动态规划 HJB	83
6.2.3 维数灾难 (The Curse)	85
6.2.4 我们为什么关心这些？	86
6.2.5 什么是拉格朗日乘子？	86
6.3 例子	86
6.3.1 QP 与 DP 实现对比示例	86
7 MPC 模型预测控制	87
7.1 凸优化与MPC简介	87
7.2 凸性基础	87
7.3 MPC基本思想	87
7.4 平面四旋翼示例	88
7.5 凸MPC示例	88
7.5.1 火箭着陆	88
7.5.2 足式机器人	88
7.6 非线性动力学怎么办？	88
7.7 一个四轴飞行器的例子	88
7.7.1 问题描述	88
7.7.2 系统模型	89
7.7.3 LQR解法	91
7.7.4 MPC解法	91
7.7.5 结果对比	93

8 轨迹优化	95
8.1 非线性轨迹优化问题	96
8.1.1 微分动态规划 (DDP/iLQR)	96
8.1.2 DDP/iLQR算法流程	104
8.1.3 DDP与iLQR的区别	104
8.1.4 优缺点总结	104
8.1.5 约束处理方法简介	105
8.1.6 最小时间问题处理	106
8.1.7 小结	106
8.1.8 ALTRQ 方法简介	106
8.2 直接轨迹优化	109
8.2.1 序列二次规划 (SQP)	109
8.2.2 直接方法 (Direct Methods)	112
8.3 间接法与直接法的对比	114
8.3.1 一个Dircol欠驱动双摆例子	114
8.4 LQR在SE3中	122
8.4.1 四元数的数值优化	122
8.4.2 四元数扰动的雅可比 (Attitude Jacobian)	122
8.4.3 Cost function关于扰动变量的一阶、二阶导	123
8.4.4 例子: Wahba问题的四元数优化	123
8.4.5 基于四元数LQR的四旋翼飞行器	125
9 混合系统	129
9.1 接触动力学仿真	129
9.2 砖块掉落案例	130
9.3 混合轨迹优化对于Legged Robot	131
10 前馈补偿与迭代学习控制	132
10.1 迭代学习控制 (ILC)	133
10.1.1 ILC的收敛性	135
10.2 ILC说明	135
10.3 倒立摆摆起控制案例: 处理模型不确定性和噪声扰动	136
11 随机最优控制与LQG理论	139
11.1 随机最优控制基础	139
11.2 LQG控制器部分	139
11.3 LQG最优观测器部分 (卡尔曼滤波)	139
11.4 KF与LQR的对偶性	139
11.4.1 数学形式的对偶性	139
11.4.2 物理意义的对偶性	139
11.4.3 设计参数的对偶关系	140
11.5 案例分析	140
11.5.1 系统模型与参数设置	140
11.5.2 LQR控制器设计	140
11.5.3 卡尔曼滤波器实现	140
11.6 附录: 多维高斯分布的性质	141
12 Robust Control	142

13 final part	143
13.1 凸松弛 (Convex Relaxation) 和如何降落火箭	143
13.1.1 Convex Relaxation	143
13.2 如何走路	144
13.3 自动驾驶和博弈论	144
13.4 数据驱动和行为克隆	144
13.5 强化学习	144
14 机器人中的数值优化引言	144
14.1 优化问题的基本形式	145
15	148
15.1 凸集与非凸集	148
15.1.1 凸集的基本性质	149
15.1.2 半正定锥的凸性与几何形状	149
15.2 高阶函数	150
15.3 高阶函数的信息与微分记号	151
15.4 凸函数及其性质	152
15.4.1 凸函数的上方图 (Epigraph)	152
15.5 为什么这些函数是凸的?	153
15.5.1 二阶条件 (Second-order conditions)	154
15.5.2 强凸性 (Strong Convexity)	154
15.5.3 Lipschitz 连续与上界 (Lipschitz Smoothness and Upper Bound)	155
15.5.4 凸函数的条件数 (Condition Number)	155
15.5.5 次微分 sub-differential	156
15.5.6 凸函数的单调性与一阶条件	157
15.6 无约束的非凸优化	157
15.6.1 最速下降 steepest gradient descent	157
15.6.2 步长选择与线搜索 (Step Size and Line Search)	157
15.6.3 Armijo 条件 (充分下降条件)	158
15.6.4 Backtracking/Armijo 线搜索算法	158
15.7 modified damped newton method (修正阻尼牛顿法)	158
15.7.1 Newton 法与二阶泰勒展开	158
15.7.2 Hessian 修正与阻尼牛顿法 (Modified/Damped Newton Method)	159
15.8 Practical Newton's Method 实用牛顿法	160
16 无约束优化理论	161

1 引言

这里是 CMU 16-745 最优控制课程的笔记。我将涵盖课程中的重要概念、定理和例子。课程目标

1. 分析动力系统的稳定性。
2. 设计稳定平衡和轨迹的LQR控制器。
3. 使用离线轨迹优化设计非线性系统的轨迹。
4. 使用在线凸优化实现模型预测控制。
5. 了解随机性和模型不稳定性的影响。
6. 无最优模型时直接优化反馈策略。

写作风格备注: (TODO: 终稿注释掉!) 额外的公式推导使用浅绿色背景框额外的例子使用浅蓝色背景框代码使用浅灰色背景

2 动力学简介

这一节面向不了解控制学、动力学的读者，以一个单摆系统为例，快速的介绍连续和离散动力学系统的基础，包括稳定性分析、线性系统和非线性系统。

2.1 连续时间的动力学系统

Continuous-Time Dynamics System (CTDS) 是一个描述系统状态随时间变化的数学模型。它通常由以下方程表示：

$$\dot{x} = f(x, u) \quad (1)$$

其中， $x(t) \in \mathbb{R}^n$ 是系统状态， $u(t) \in \mathbb{R}^m$ 是控制输入， f 是一个描述系统的动态函数，展示了系统如何根据 u 而演化。

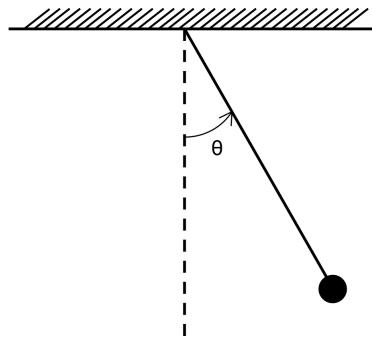


图 1：简单单摆的物理模型。图中展示了摆球的受力情况，包括重力 mg 和拉力 \vec{T} ，以及摆球的运动方向和角度 θ 的关系。

对于1中的单摆系统，其状态 x 可以用角度 q 和速度 v 来表示。我们可以将其状态表示为：

$$x = \begin{bmatrix} q \\ v \end{bmatrix} \quad (2)$$

q 并不总是一个vector。configuration并不必须是一个vector，并且速度并不必须是configuration的导数。并且，仅在对系统的动力学描述是平滑的时候，才可以说这个系统的描述是有效的。

连续动态系统的简单单摆例子 考虑1中的单摆系统。我们可以用以下方程来描述它的动力学：

$$ml^2\ddot{\theta} + mglsin(\theta) = \tau \quad (3)$$

其推导过程如下：

从拉格朗日方程推导动力学方程推导* 在这一节中，我们使用拉格朗日方法推导单摆系统的动力学方程。首先，单摆的动能为：

$$T = \frac{1}{2}ml^2\dot{\theta}^2$$

势能为：

$$V = mgl(1 - \cos \theta)$$

其中， m 是质量， l 是摆长， θ 是角度， $\dot{\theta}$ 是角速度， g 是重力加速度。拉格朗日量 L 为动能 V 与势能 L 之差，即：

$$L = T - V = \frac{1}{2}ml^2\dot{\theta}^2 - mgl(1 - \cos \theta)$$

拉格朗日方程的一般形式为：

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{\theta}} \right) - \frac{\partial L}{\partial \theta} = 0$$

我们计算拉格朗日量的偏导数：

1. 对于 $\frac{\partial L}{\partial \dot{\theta}}$ ，有：

$$\frac{\partial L}{\partial \dot{\theta}} = ml^2\dot{\theta}$$

2. 对于 $\frac{\partial L}{\partial \theta}$ ，有：

$$\frac{\partial L}{\partial \theta} = mgl \sin \theta$$

将这些代入拉格朗日方程，得到：

$$\frac{d}{dt} \left(ml^2\dot{\theta} \right) - mgl \sin \theta = 0$$

即：

$$ml^2\ddot{\theta} + mgl \sin \theta = 0$$

如果引入控制输入 τ ，方程变为：

$$ml^2\ddot{\theta} + mgl \sin \theta = \tau$$

这个方程描述了单摆的动力学行为，其中 τ 是控制输入，能够影响单摆的角加速度。最终，我们得到了单摆的动力学方程：

$$ml^2\ddot{\theta} + mgl \sin \theta = \tau$$

则系统的配置 q 可以通过单摆的角度 θ 来表示，速度 v 可以通过角速度 $\dot{\theta}$ 来表示。并且考虑到控制量 u ，通过施加到系统中的力矩(torque) τ 来控制单摆的运动。则状态可以表示为：

$$x = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} \quad (4)$$

则其速度 \dot{x} 可以表示为：

$$\dot{x} = \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{\theta} \\ -\frac{g}{l} \sin(\theta) + \frac{\tau}{ml^2} \end{bmatrix} \quad (5)$$

则系统动态方程可以写为 $f(x, u)$ ：

$$\dot{x} = f(x, u) = \begin{bmatrix} \dot{\theta} \\ -\frac{g}{l} \sin(\theta) + \frac{\tau}{ml^2} \end{bmatrix} \quad (6)$$

则这里状态是一个流形(manifold)的函数 $x \in \mathbb{S}^1 \times \mathbb{R}$ 是一个圆柱，表示了系统的状态随时间的变化。

2.1.1 控制映射系统

很多系统可以定义为一个控制映射系统。这是一个上述动力系统的特殊形式，其中控制输入通过一个映射矩阵 G 来影响系统。

$$\dot{x} = f_0(x) + G(x)u \quad (7)$$

对于上述简单单摆模型，可以将其表示为：

$$\dot{x} = \begin{bmatrix} \dot{\theta} \\ -\frac{g}{l} \sin(\theta) + \frac{\tau}{ml^2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix} \tau \quad (8)$$

2.1.2 Manipulator动力学

Manipulator是一个可以在空间中移动的物体。它的动力学可以用以下方程表示（参考《现代机器人学机构、规划与控制》的第八章，读者感兴趣可以自行阅读，此处略）：

$$M(q)\dot{v} + C(q, v) = B(q)u \quad (9)$$

其中， $M(q)$ 是质量矩阵， $C(q, v)$ 是科里奥利力矩阵和重力项， $B(q)$ 是控制输入矩阵。这个方程描述了Manipulator的动力学行为，其中 u 是控制输入，能够影响Manipulator的运动。则configuration的变化可以表示为：

$$\dot{q} = G(q)v \quad (10)$$

其中， $G(q)$ 是一个映射矩阵，将速度 v 映射到configuration的变化 \dot{q} 。则系统的运动学可以描述为：

$$\dot{x} = f(x, y) = \begin{bmatrix} G(q)v \\ -M(q)^{-1}(B(q) - C) \end{bmatrix} \quad (11)$$

在简单单摆系统中 $M(q) = ml^2$, $C(q, v) = mgl \sin(\theta)$, $B = I$, $G = I$ 。

所有机械系统都可以表述为这个形式，这是因为这个形式是一个描述Euler-Lagrange方程的不同形式(Kinetic energy - potential energy $L = T - V$)。

$$L = \frac{v^T M(q)v}{2} - V(q) \quad (12)$$

2.1.3 线性系统

线性系统(Linear System)是表述控制问题和设计控制器的通用表述方法。我们知道可以相对简单的解决一个线性系统，线性系统描述为：

$$\dot{x} = A(t)x + B(t)u \quad (13)$$

其中， $A(t)$ 和 $B(t)$ 是时间变化的矩阵。线性系统的解可以通过求解常微分方程来获得。线性系统的稳定性分析通常使用特征值和特征向量的方法。线性系统的控制器设计通常使用状态反馈和观测器设计的方法。

我们通常通过在现在状态附近的线性化来近似线性化非线性系统:

$$\dot{x} = f(x, u) \quad (14)$$

其中 $A = \frac{\partial f}{\partial x}$, $B = \frac{\partial f}{\partial u}$

2.1.4 平衡点 equilibria

equilibria指的是系统将会保持的位置，即一阶稳态。

$$\dot{x} = f(x, u) = 0 \quad (15)$$

在代数层面，这表示动力方程的根。对于简单单摆系统，平衡点可以表示为：

$$\dot{x} = f(x, u) = \begin{bmatrix} \dot{\theta} \\ -\frac{g}{l} \sin(\theta) + \frac{\tau}{ml^2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (16)$$

平衡点的求解 考虑一个简单的问题，我们如何找到一个平衡点？我们可以通过求解以下方程来找到平衡点：

$$\dot{x} = f(x, u) = \begin{bmatrix} \dot{\theta} \\ -\frac{g}{l} \sin(\theta) + \frac{\tau}{ml^2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow \frac{1}{ml^2} u = \frac{g}{l} \sin(\theta) \Rightarrow u = mgl \sin(\theta) \quad (17)$$

带入 $\theta = \pi/2$, 得到 $u = mgl$ 一般来说，可以将寻找平衡点的问题转化为求解一个非线性方程组的问题。

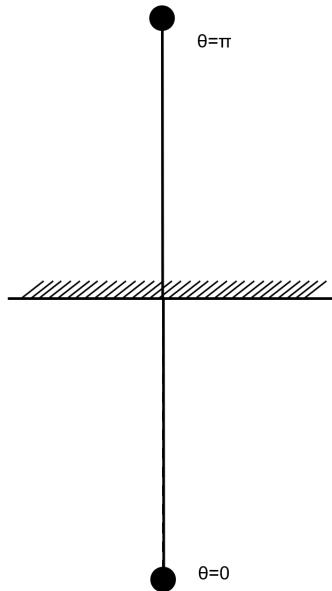


图 2: 图中展示了单摆系统的平衡点。平衡点是指系统在没有外部扰动的情况下，能够保持静止或匀速运动的状态。

2.1.5 平衡点的稳定性分析

在知道了如何求平衡点后，我们需要知道，如果对系统施加一个小的扰动，系统是否会回到平衡点。考虑一个1维系统，在这个例子中，当 $\frac{\partial f}{\partial x} < 0$ 时系统是稳定的。

当 $\frac{\partial f}{\partial x} > 0$ 时系统是不稳定的。我们可以通过线性化来分析系统的稳定性。对于一个线性系统，我们可以通过求解特征值来判断系统的稳定性。对于一个非线性系统，我们可以通过求解雅可比矩阵的特征值来判断系统的稳定性，因为从这个点越推越远。这个 $\frac{\partial f}{\partial x} < 0$ 的区域是一个吸引子(Attractor)又称basin of attraction of system(系统吸引盆地)，而 $\frac{\partial f}{\partial x} > 0$ 的区域是一个排斥子(Repeller)。

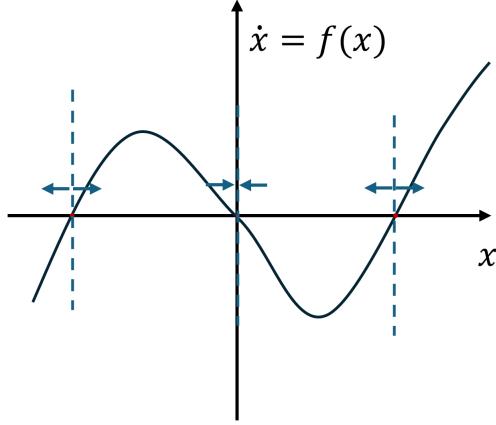


图 3: 图中展示了1维系统的稳定性分析。图中对于 $\dot{x} = 0$ 有三个解，也就是说有三个平衡点，但是只有中间的点满足 $\frac{\partial f}{\partial x} < 0$ ，而左右两侧 $f(x)$ 对 x 的偏导都是大于0，系统下一步将采取正向的行动，导致系统不稳定。

推广到高维系统，这个 $\frac{\partial f}{\partial x} < 0$ 则为是一个 Jacobian matrix。为了研究系统的稳定性，我们可以对 Jacobian matrix 进行特征值分解。对于一个线性系统，我们可以通过求解特征值来判断系统的稳定性。即，如果每个特征值的实部都小于0，则系统是渐近稳定的；如果有任何一个特征值的实部大于0，则系统是不稳定的；如果所有特征值的实部都等于0，则系统是边界稳定的。可以用公式描述为：

$$Re(eig(\frac{\partial f}{\partial x})) = \begin{cases} < 0 & \text{渐近稳定} \\ > 0 & \text{不稳定} \\ = 0 & \text{边界稳定} \end{cases} \quad (18)$$

Jacobian矩阵定义为：

$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \quad (19)$$

在上述简单单摆系统中

$$f(x) = \begin{bmatrix} \dot{\theta} \\ -\frac{g}{l} \sin(\theta) \end{bmatrix} \Rightarrow \frac{\partial f}{\partial x} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{l} \cos(\theta) & 0 \end{bmatrix} \quad (20)$$

考虑平衡点 $\theta = 0$ ，我们可以得到 Jacobian 矩阵的特征值：

$$\frac{\partial f}{\partial x} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{l} & 0 \end{bmatrix} \Rightarrow \lambda^2 + \frac{g}{l} = 0 \Rightarrow \lambda = \pm i \sqrt{\frac{g}{l}} \quad (21)$$

由于有一个特征值的实部大于0，表示系统在该点是一个鞍点，即系统在该点附近的运动是发散的。

考虑平衡点 $\theta = \pi$ ，我们可以得到Jacobian矩阵的特征值：

$$\frac{\partial f}{\partial x} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{l} & 0 \end{bmatrix} \Rightarrow \lambda^2 - \frac{g}{l} = 0 \Rightarrow \lambda = 0 \pm \sqrt{\frac{g}{l}} \quad (22)$$

该点特征值实部为0，表示系统在该点是一个中心点，即系统在该点附近的运动是周期性的（不考虑空气摩擦力）。对于纯虚数特征值系统，这个系统叫做边缘稳定（marginally stable）。可以添加damping来使得系统稳定，如 $u = -K_d\dot{\theta}$ （摩擦力）。

2.2 离散时间的动力学系统

在这一节中，我们将介绍

- 连续的常微分方程（Ordinary Differential Equations, ODEs）
- 离散动力系统稳定性分析

在通常情况下，我们无法直接从 $\dot{x} = f(x)$ 中求出 $x(t)$ 的解析解，则无法直接求出系统的状态随时间的变化。我们可以通过在离散时间内通过数值的方法近似表示 $x(t)$ 。并且，离散时间模型可以捕获连续ODEs无法捕获的动态行为（比如，离散时间模型可以捕获系统的周期性行为，接触）。

2.2.1 正向欧拉法

离散时间动力学系统（Discrete-Time Dynamics System, DTDS）是一个描述系统状态随时间变化的数学模型。通常用于模拟计算，如模拟器等。它通常由以下方程表示：

$$x_{k+1} = f_{discrete}(x_k, u_k) \quad (23)$$

正向欧拉法* 这里将介绍正向欧拉法，并举一个例子。

用一个最简单的离散化表示，即正向欧拉法（Forward Euler Method）：

$$x_{k+1} = x_k + h f_{continuous}(x_k, u_k) \quad (24)$$

其中， h 是离散化的时间步长。这个方程表示在时间步长 h 内，系统状态 x_k 和控制输入 u_k 的关系。 $f_{continuous} = \frac{\partial f(x,u)}{\partial t}$

手算例子： 考虑一个简单的线性系统 $\dot{x} = -2x$ ，其离散化形式为：

$$x_{k+1} = x_k + h(-2x_k) = (1 - 2h)x_k \quad (25)$$

假设初始条件为 $x_0 = 1$ 且步长 $h = 0.1$ ，则计算前几个离散时间点的状态：

- 对于 $k = 0$, $x_0 = 1$;
- 对于 $k = 1$, $x_1 = (1 - 2 \times 0.1) \times 1 = 0.8$;
- 对于 $k = 2$, $x_2 = (1 - 2 \times 0.1) \times 0.8 = 0.64$;
- 对于 $k = 3$, $x_3 = (1 - 2 \times 0.1) \times 0.64 = 0.512$ 。

我们可以看到，系统状态随着时间步的增加逐渐减小。

用一个最简单的离散化表示，即正向欧拉法（Forward Euler Method）：

$$x_{k+1} = x_k + h f_{continuous}(x_k, u_k) \quad (26)$$

其中， h 是离散化的时间步长。这个方程表示在时间步长 h 内，系统状态 x_k 和控制输入 u_k 的关系。 $f_{continuous} = \frac{\partial f(x,u)}{\partial t}$

离散时间动力学系统的简单单摆例子* 继续单摆系统

单摆的连续时间动力学系统可以通过以下方程描述：

$$\dot{x} = f(x, u) = \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{\theta} \\ -\frac{g}{l} \sin(\theta) + \frac{\tau}{ml^2} \end{bmatrix}$$

其中： θ 为单摆的角度（配置变量）， $\dot{\theta}$ 为角速度（速度变量）， $\ddot{\theta}$ 为角加速度。对于离散时间动力学系统，我们采用正向欧拉法进行离散化。设定离散时间步长为 h ，那么在每个时间步内，系统的状态 $x_k = [\theta_k, \dot{\theta}_k]$ 通过以下公式更新：

$$x_{k+1} = x_k + h \cdot f(x_k, u_k)$$

将连续时间的动力学方程 $f(x, u)$ 代入上式，我们得到：

$$x_{k+1} = \begin{bmatrix} \theta_k \\ \dot{\theta}_k \end{bmatrix} + h \cdot \begin{bmatrix} \dot{\theta}_k \\ -\frac{g}{l} \sin(\theta_k) + \frac{\tau_k}{ml^2} \end{bmatrix}$$

这个递推公式可以用来计算每个时间步的状态。

具体而言，我们可以分别更新角度和角速度：

$$\theta_{k+1} = \theta_k + h \cdot \dot{\theta}_k$$

$$\dot{\theta}_{k+1} = \dot{\theta}_k + h \left(-\frac{g}{l} \sin(\theta_k) + \frac{\tau_k}{ml^2} \right)$$

这个离散化公式描述了如何在每个时间步内，根据当前的角度和角速度，通过正向欧拉法更新单摆的状态。

针对简单单摆系统，设置 $l = m = 1, h = 0.1$ or 0.01 将会爆炸（blow up）。

以下实现了单摆系统的离散时间模拟代码，缩短模拟步长可以看到 blow up 被延缓了，但是仍会 blow up。

```

1 # [ discrete-time-dynamic-forward-euler.py ]
2 def pendulum_dynamics(x):
3     l = 1.0
4     g = 9.81
5
6     theta = x[0] # 角度
7     theta_dot = x[1] # 角速度
8     theta_ddot = -(g/l) * np.sin(theta) # 角加速度，由重力引起的
9
10    return np.array([theta_dot, theta_ddot])
11
12 def pendulum_forward_euler(fun, x0, Tf, h):
13     t = np.arange(0, Tf + h, h)
14
15     x_hist = np.zeros((len(x0), len(t))) # 状态历史记录
16     x_hist[:, 0] = x0 # 初始状态
17
18     for k in range(len(t) - 1): # 迭代计算每个时间步的状态
19         x_hist[:, k+1] = x_hist[:, k] + h * fun(x_hist[:, k]) # 使用欧拉法更新状态
20     return x_hist, t
21
22 if __name__ == "__main__":
23     # 初始条件和参数
24     x0 = np.array([0.1, 0]) # 初始角度和角速度
25     Tf = 10.0 # 模拟总时间(秒),
26     h = 0.01 # 时间步长。
27
28     # 计算单摆运动
29     x_hist, t_hist = pendulum_forward_euler(pendulum_dynamics, x0, Tf, h)
30

```

```

31     # 创建可视化对象并显示动画
32     visualizer = PendulumVisualizer(pendulum_length=1.0)
33     visualizer.visualize(x_hist, t_hist)

```

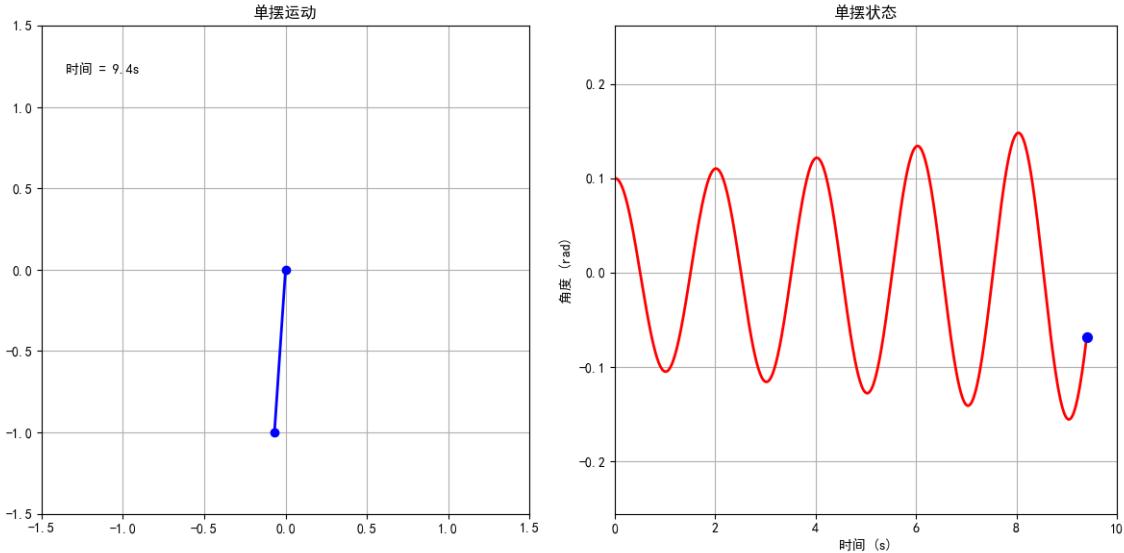


图 4: 图中展示了欧拉方法模拟的单摆系统的运动轨迹。可以看到，随着时间的推移，单摆逐渐blow up，说明该方法是数值不稳定的

正向欧拉法的稳定性分析 正向欧拉法的稳定性分析可以通过考虑离散时间系统的特征值来进行。对于线性系统，正向欧拉法的稳定性取决于离散化步长 h 和系统矩阵 A 的特征值。具体而言，如果所有特征值的模长小于1，则系统是渐近稳定的；如果有一个特征值的模长大于1，则系统是不稳定的。回顾之前判断系统是否稳定， $\text{Re}(\text{eig}(\frac{\partial f}{\partial x}))$ 与0的关系。对于离散时间，动力学系统可以迭代映射为：

$$x_N = f_d(f_d(f_d(\dots f_d(x_0))\dots)) \quad (27)$$

考虑线性和chain rule，有：

$$\frac{\partial x_N}{\partial x_0} = \left. \frac{\partial f_d}{\partial x_0} \frac{\partial f_d}{\partial x_1} \dots \frac{\partial f_d}{\partial x_{N-1}} \right|_{x_0} = A_d^N \quad (28)$$

其中 A_d 表示线性化的点的偏导 $\frac{\partial f_d}{\partial x}$ ，并且在迭代中，每个点的偏导数都是相同的。

稳定点是 $x = 0$ （可以通过修改坐标系达成），系统的稳定性意味着：

$$\lim_{N \rightarrow \infty} A_d^N x_0 = 0, \forall x_0 \in \mathbb{R}^n \Rightarrow \lim_{N \rightarrow \infty} A_d^N = 0 \Rightarrow |\text{eig}(A_d)| < 1 \quad (29)$$

如果 $|\text{eig}(A_d)| > 1$ ，则系统不稳定，迭代之后那些大于1的特征值将会占据主导地位，导致系统不稳定。

等价的， $\text{eig}(A_d)$ 必须在复数平面上位于单位圆内。

下面开始分析简单单摆系统在欧拉方法离散化后的稳定性， h 取 0.1, $g=9.81$, $l=1.0$ 。

$$A_d = \frac{f_d}{x_k} = I + h A_{\text{continuous}} = I + h \begin{bmatrix} 0 & 1 \\ -\frac{g}{l} \cos(\theta) & 0 \end{bmatrix} \quad (30)$$

$$\Rightarrow \text{eig}(A_d|_{\theta=0}) = 1 \pm 0.313i \Rightarrow |\text{eig}(A_d)| = 1.313 > 1 \Rightarrow \text{不稳定} \quad (31)$$

我们可以将h和eigenvalue的关系绘制成图表，其代码如下：

```

1 # [euler_eigenvalue_stability.py]
2 # 此处仅展示关键代码
3
4 h_values = np.linspace(0.01, 0.5, 100)
5 eigenvalues = []
6
7 for h in h_values:
8     A_d = A_discrete(h, theta)
9     eigs = eigvals(A_d)
10    eigenvalues.append(eigs)
11
12 # 转换为numpy数组便于处理
13 eigenvalues = np.array(eigenvalues)
14
15 # 计算特征值的模
16 magnitudes = np.abs(eigenvalues)

```

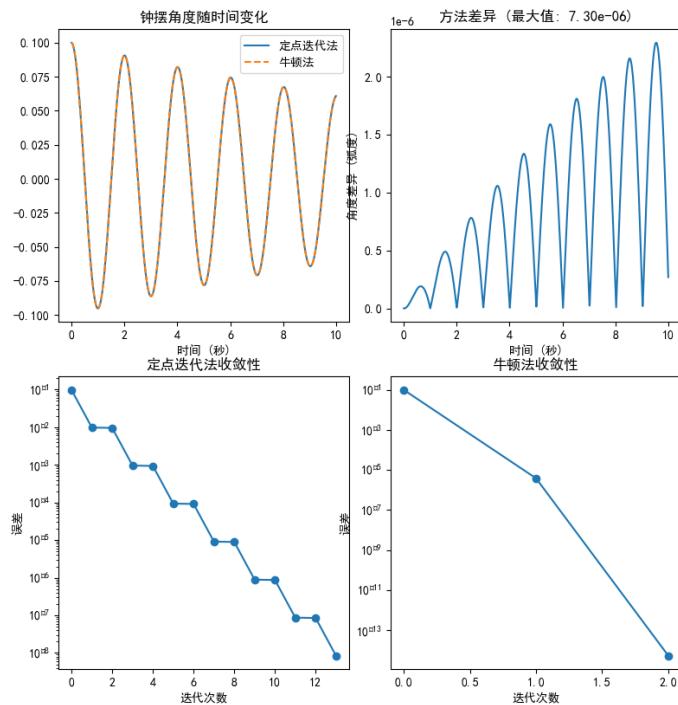


图 5：图中展了简单单摆系统在欧拉法离散化后的稳定性分析，通过观察不同时间步长下特征值的变化来判断系统的数值稳定性。

可以从5中看出

- 欧拉法对简单单摆系统的离散化在任何正的时间步长下都是不稳定的，因为特征值的模总是大于1。
- 时间步长越大，特征值的模越大，系统越不稳定。
- 系统的两个特征值在实部上逐渐分离
- 所有点都在单位圆外，表明系统在所有正的h值下都是不稳定的

结论

- 谨慎使用正向欧拉法，尤其是对于非线性系统。正向欧拉法在某些情况下可能会导致数值不稳定，尤其是在系统具有强非线性或刚性时。对于这些情况，可以考虑使用更高级的数值积分方法，如RK4方法或隐式欧拉法等。
- 在稳定点时谨慎检测的能量行为，是否是保守系统，是否有耗散。
- 因为欧拉方法说到底是在 t 时刻的导数 $f(x_k, u_k)$ ，相对于 x 总会有一个overshoot导致系统越来越blow up。所以避免使用正向欧拉法来模拟非线性系统的动力学行为。

2.2.2 RK4方法

4th order Runge-Kutta method (RK4) 是一种常用的数值积分方法，用于求解常微分方程 (ODEs)。它通过在每个时间步内计算多个斜率来提高精度。RK4方法的基本思想是通过四个斜率的加权平均来近似ODE的解。

RK4方法* 这里将介绍RK4方法，并举一个例子。

RK4方法（四阶龙格-库塔法）是一种常用的数值积分方法，用于求解常微分方程。它的基本思想是通过在每个时间步内计算四个斜率来估算系统状态的变化，进而得到更精确的解。

RK4方法的基本步骤如下：

1. 计算四个斜率：

$$\begin{aligned} k_1 &= hf(x_k, u_k) \quad (\text{evaluate at beginning}) \\ k_2 &= hf\left(x_k + \frac{1}{2}k_1, u_k\right) \quad (\text{evaluate at midpoint 1}) \\ k_3 &= hf\left(x_k + \frac{1}{2}k_2, u_k\right) \quad (\text{evaluate at midpoint 2}) \\ k_4 &= hf(x_k + k_3, u_k) \quad (\text{evaluate at end}) \end{aligned}$$

其中， h 是时间步长， $f(x_k, u_k)$ 是系统的动力学方程， x_k 和 u_k 分别表示在时刻 k 时的状态和控制变量。

2. 更新状态：

$$x_{k+1} = x_k + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

通过这一步，RK4方法将四个斜率加权平均，得到更精确的状态更新公式。

RK4方法通过计算四个斜率来提高精度，相比于正向欧拉法，RK4方法在每个时间步内计算了更多的信息，从而能够更准确地逼近常微分方程的解。RK4方法的误差是 $O(h^4)$ ，比欧拉法的 $O(h^2)$ 更高效，适合用于高精度要求的数值计算。

手算例子： 现在，我们通过一个简单的例子来手动计算RK4方法的应用。假设我们要求解以下简单的常微分方程：

$$\frac{dx}{dt} = -2x$$

初始条件为 $x(0) = 1$, 我们选择时间步长 $h = 0.1$ 。我们将使用RK4方法计算 x 在 $t = 0.1$ 时刻的值。

1. 计算四个斜率:

$$\begin{aligned}k_1 &= 0.1 \cdot (-2 \cdot 1) = -0.2 \\k_2 &= 0.1 \cdot \left(-2 \cdot \left(1 + \frac{1}{2} \cdot (-0.2)\right)\right) = -0.19 \\k_3 &= 0.1 \cdot \left(-2 \cdot \left(1 + \frac{1}{2} \cdot (-0.19)\right)\right) = -0.19 \\k_4 &= 0.1 \cdot \left(-2 \cdot \left(1 + (-0.19)\right)\right) = -0.162\end{aligned}$$

2. 更新状态:

$$x_1 = 1 + \frac{1}{6}(-0.2 + 2(-0.19) + 2(-0.19) + (-0.162)) = 0.818$$

因此, 使用RK4方法得到的 $x(0.1)$ 的近似值为0.818。

通过这个手算例子, 我们可以看到RK4方法比简单的欧拉法能够提供更精确的结果, 尤其是在步长较大的情况下。

```
1 # [ discrete_time_dynamic_RK4.py ]
2 import numpy as np
3 from src.PendulumVisualizer import *
4 def pendulum_dynamics(x):
5     l = 1.0
6     g = 9.81
7
8     theta = x[0]
9     theta_dot = x[1]
10    theta_ddot = -(g/l) * np.sin(theta)
11
12    return np.array([theta_dot, theta_ddot])
13
14 def pendulum_rk4(fun, x0, Tf, h):
15     t = np.arange(0, Tf + h, h)
16
17     x_hist = np.zeros((len(x0), len(t)))
18     x_hist[:, 0] = x0
19
20     for k in range(len(t) - 1):
21         x = x_hist[:, k]
22
23         k1 = fun(x)
24         k2 = fun(x + h/2 * k1)
25         k3 = fun(x + h/2 * k2)
26         k4 = fun(x + h * k3)
27
28         x_hist[:, k+1] = x + h/6 * (k1 + 2*k2 + 2*k3 + k4)
29
30     return x_hist, t
31
32 if __name__ == "__main__":
33     x0 = np.array([0.1, 0])
34     Tf = 10.0
35     h = 0.01
36
37     x_hist, t_hist = pendulum_rk4(pendulum_dynamics, x0, Tf, h)
38
```

```

39     visualizer = PendulumVisualizer(pendulum_length=1.0)
40     visualizer.visualize(x_hist, t_hist)

```

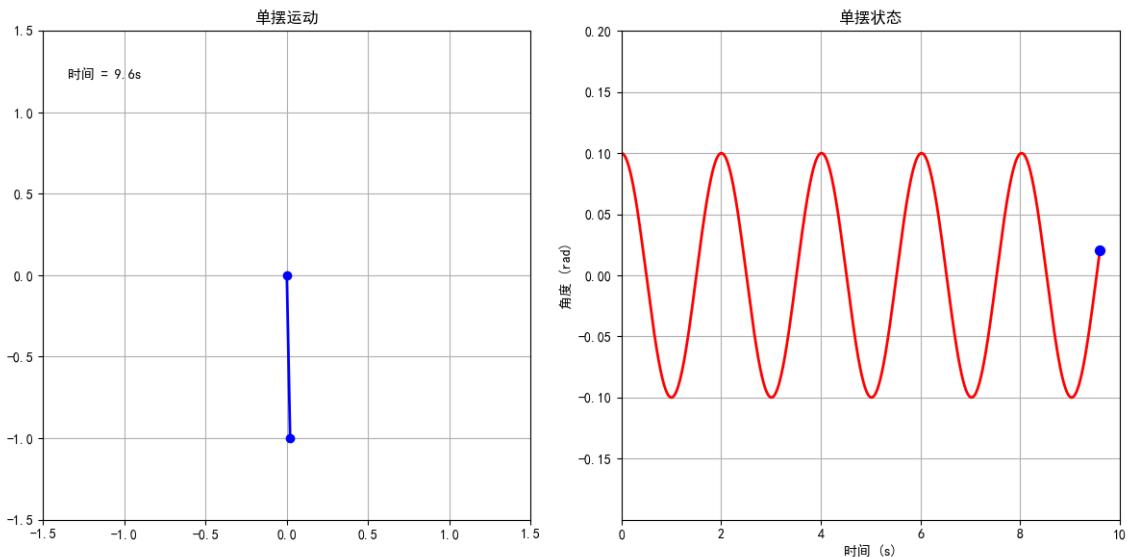


图 6: 图中展示了RK4模拟的单摆系统的运动轨迹。可以看到，随着时间的推移，单摆系统的运动轨迹更加平滑且稳定。

RK4稳定性分析 通过计算RK4的雅可比矩阵的特征值我们可以得到：

RK4 方法稳定性分析

RK4 方法稳定性分析实验结果

- 单摆线性化系统的特征值：

$$\lambda = \begin{bmatrix} 0 + 3.1321j \\ 0 - 3.1321j \end{bmatrix}$$

- 步长 $h = 0.01$ 下，RK4 方法的雅可比矩阵特征值：

$$\lambda_J = \begin{bmatrix} 0.9995 + 0.0313j \\ 0.9995 - 0.0313j \end{bmatrix}$$

- 放大因子（特征值绝对值）：

$$|\lambda_J| = \begin{bmatrix} 0.999999999993 \\ 0.999999999993 \end{bmatrix}$$

不同步长下的稳定性分析

- 对于不同的步长 h ，RK4 方法的稳定性如下所示：

RK4方法稳定域和单摆系统特征值

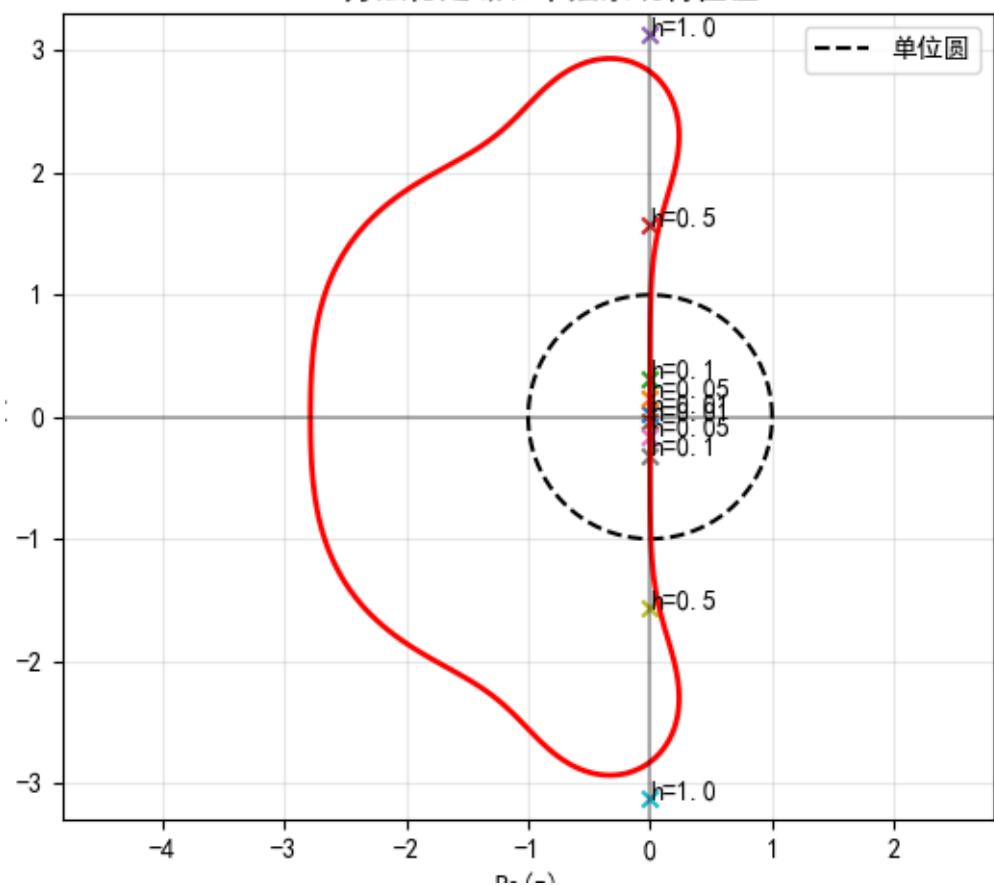


图 7: 图中展示了不同步长下, RK4方法的特征值, 可以看到落在单位圆中的量。

$h = 0.01$

$$\lambda = \begin{bmatrix} 0 + 3.1321j \\ 0 - 3.1321j \end{bmatrix}, \quad \lambda_J = \begin{bmatrix} 0.9995 + 0.0313j \\ 0.9995 - 0.0313j \end{bmatrix}, \quad |\lambda_J| = \begin{bmatrix} 0.999999999993 \\ 0.999999999993 \end{bmatrix}$$

$h = 0.05$

$$\lambda = \begin{bmatrix} 0 + 3.1321j \\ 0 - 3.1321j \end{bmatrix}, \quad \lambda_J = \begin{bmatrix} 0.9878 + 0.1560j \\ 0.9878 - 0.1560j \end{bmatrix}, \quad |\lambda_J| = \begin{bmatrix} 0.999999897875 \\ 0.999999897875 \end{bmatrix}$$

$h = 0.1$

$$\lambda = \begin{bmatrix} 0 + 3.1321j \\ 0 - 3.1321j \end{bmatrix}, \quad \lambda_J = \begin{bmatrix} 0.9514 + 0.3081j \\ 0.9514 - 0.3081j \end{bmatrix}, \quad |\lambda_J| = \begin{bmatrix} 0.999993524289 \\ 0.999993524289 \end{bmatrix}$$

$h = 0.5$

$$\lambda = \begin{bmatrix} 0 + 3.1321j \\ 0 - 3.1321j \end{bmatrix}, \quad \lambda_J = \begin{bmatrix} 0.0244 + 0.9259j \\ 0.0244 - 0.9259j \end{bmatrix}, \quad |\lambda_J| = \begin{bmatrix} 0.9262 \\ 0.9262 \end{bmatrix}$$

$h = 1.0$

$$\lambda = \begin{bmatrix} 0 + 3.1321j \\ 0 - 3.1321j \end{bmatrix}, \quad \lambda_J = [0.1048 \pm 1.9889j], \quad |\lambda_J| = \begin{bmatrix} 1.9916 \\ 1.9916 \end{bmatrix}$$

符号说明

- λ : 线性化系统的特征值, 反映系统本身的动力特性;
- J : RK4 离散系统下的雅可比矩阵, 用于描述线性近似;
- λ_J : RK4 方法下的雅可比矩阵特征值;
- $|\lambda_J|$: 特征值的模 (放大因子), 表示数值解是否随时间增长或衰减。

```

1 # [discrete_time_dynamic_RK4.py]
2 # 此处仅展示关键代码
3 def compute_rk4_jacobian_eigenvalues(h):
4     # 线性系统  $x' = Ax$  的情况下, A的特征值为 lambda
5     # 对于单摆在小角度近似下, 线性化后的A矩阵为
6     l = 1.0
7     g = 9.81
8     A = np.array([[0, 1], [-g/l, 0]]) # 线性化的单摆动力学
9
10    # 计算A的特征值
11    eigen_A = np.linalg.eigvals(A)
12    print("单摆线性化系统的特征值:", eigen_A)
13
14    # 计算RK4方法的放大矩阵特征值
15    #  $R(z) = 1 + z + z^2/2 + z^3/6 + z^4/24$ 
16
17    # 计算不同特征值的放大因子
18    amplification_factors = []
19    eigenvalues = []
20
21    # 对每个特征值计算放大因子
22    for lam in eigen_A:
23        z = h * lam
24        R = 1 + z + z**2/2 + z**3/6 + z**4/24

```

```

25     amplification_factors.append(np.abs(R))
26     eigenvalues.append(R)
27
28     print(f'步长 h = h下RK4方法的雅可比矩阵特征值:', eigenvalues)
29     print(f'放大因子(特征值绝对值):', amplification_factors)
30
31     return eigen_A, eigenvalues, amplification_factors

```

稳定性判据与特征值分析 设系统矩阵 A 的特征值为 λ_i , 步长为 h , 则有:

$$z_i = h \cdot \lambda_i$$

RK4 迭代中相应的放大因子为:

$$\rho_i = R(z_i)$$

其模长:

$$|\rho_i| = |R(z_i)|$$

判断稳定性标准:

- 若 $|\rho_i| < 1$, 则该模态衰减 (数值稳定);
- 若 $|\rho_i| = 1$, 则该模态保持 (边界稳定);
- 若 $|\rho_i| > 1$, 则该模态发散 (不稳定)。

这是因为在数值方法 (如 Runge-Kutta 方法) 中, 我们并不是直接求解原始的微分方程 $\dot{x} = Ax$, 而是通过一种离散时间的迭代形式来逼近它。

我们逐步解释为何会有:

$$z_i = h \cdot \lambda_i, \quad \rho_i = R(z_i)$$

1. 原始系统的连续解形式 设系统为:

$$\dot{x}(t) = Ax(t)$$

其解析解为:

$$x(t) = e^{At}x(0)$$

也就是说, 系统演化由指数矩阵 e^{At} 控制。

2. 数值方法的离散迭代 RK4 等数值方法的本质是逼近这个指数矩阵的一种方式。对于一个离散步长 h , 我们用某个近似演化矩阵 $R(hA)$ 来代替 e^{hA} , 从而实现一步迭代:

$$x_{n+1} = R(hA)x_n$$

3. 为什么用特征值来简化? 矩阵 A 的特征值 λ_i 描述了系统在各特征方向上的增长或衰荡速率。在分析数值稳定性时, 我们关注每个模态 (也就是特征值对应的方向) 在一步后是否被放大或抑制。

由于特征值和矩阵函数的可交换性 (在 A 可对角化时):

$$\text{如果 } Av_i = \lambda_i v_i, \text{ 则 } R(hA)v_i = R(h\lambda_i)v_i$$

也就是说，数值迭代对每个特征模态的作用可以简化为一个复数放大因子：

$$\rho_i = R(h\lambda_i)$$

其中：

- $h\lambda_i = z_i$: 当前模态对应的无量纲时间步；
- $R(z_i)$: RK4 稳定性函数，在 z_i 的值，给出该模态的“放大比例”；
- $|R(z_i)|$: 判断该模态是否稳定（小于1表示衰减，等于1表示保持，大于1表示爆炸）。

4. 直觉类比 你可以将 λ_i 理解为一个“频率”或“增长率”，而 h 是我们离散模拟的“采样间隔”。那么：

$$z_i = h\lambda_i$$

用这个“频率率（时间步长）”来捕捉这个模态，数值方法能否稳定逼近它？如果 z_i 落在 RK4 的稳定区域，则该模态可以被良好模拟；否则，就会出现误差爆炸。

2.2.3 反向欧拉法

pybullet、mujoco使用的正是反向欧拉方法。对于隐式ODEs，例如：

$$f_d(x_{k+1}, x_k, u_k) = 0 \quad (32)$$

这表示在每个时间步内，系统的状态 x_{k+1} 和控制输入 u_k 之间的关系是隐式的。我们可以通过迭代的方法来求解这个方程。反向欧拉法（Backward Euler Method）是一种常用的数值积分方法，用于求解隐式ODEs。它的基本思想是通过在每个时间步内计算一个隐式方程来估算系统状态的变化，进而得到更精确的解。

$$x_{k+1} = x_k + hf(x_{k+1}) \quad (33)$$

相较于forward euler，这里的backward euler计算 $f(x)$ 是用的未来的时间 x_{k+1} 而不是当前的时间 x_k 。于是我们可以通过反向欧拉法来模拟系统：

$$f_d(x_{k+1}, x_k, u_k) = x_{k+1} - x_k - hf(x_{k+1}, u_k) = 0 \quad (34)$$

可以将这个问题转换为一个 x_{k+1} 时刻的root-finding问题，这个问题将在之后介绍。类似于机器学习中计算inverse dynamic的Newton-Euler方法？

反向欧拉法直觉理解 backward euler 通过离散化向系统添加了damping，即产生了undershoot（相对于euler方法的overshoot），使得系统在每个时间步内都能保持稳定。通过在每个时间步内计算一个隐式方程，反向欧拉法能够更好地捕捉系统的动态行为，尤其是在处理刚性系统时。反向欧拉法的误差是 $O(h)$ ，比欧拉法的 $O(h^2)$ 更高效，适合用于高精度要求的数值计算。但是这导致了由此构建的模拟器与现实存在很大的差距！值得注意的是：

- 显式方法通常比隐式方法更稳定。

- 对于forward simulation, 解决隐式ODEs代价很高。
- 对于很多直接的轨迹优化方法, 不再如此高代价。

2.2.4 离散控制输入

对于系统, 我们前文论述了状态通过 $x(t)$ 来描述, 并考虑了连续与离散时间的状态。现在我们来考虑系统的控制输入 $u(t)$ 的离散化表示。

$$u(t) = u_k, t_k \leq t < tk + 1 \quad (35)$$

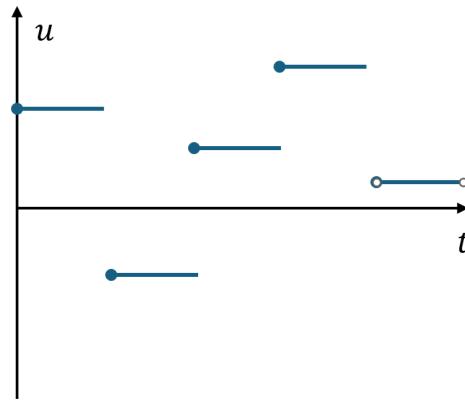


图 8: 这是一个zero-order hold的示意图。它表示在每个时间步内, 控制输入 $u(t)$ 保持不变, 直到下一个时间步到来。这个方法在数字控制系统中非常常见, 因为它可以简化控制器的设计和实现。

零阶保持 $u(t) = u_n, t_n \leq t \leq T_{n+1}$

而一阶保持 $u(t) = u_n + (\frac{u_{n+1}-u_n}{h}(t - t_n))$, 其中 h 是时间步长。一阶保持是一个更好的代替方法。它表示在每个时间步内, 控制输入 $u(t)$ 是一个线性函数, 直到下一个时间步到来。这个方法在数字控制系统中也很常见, 因为它可以更好地捕捉系统的动态行为。

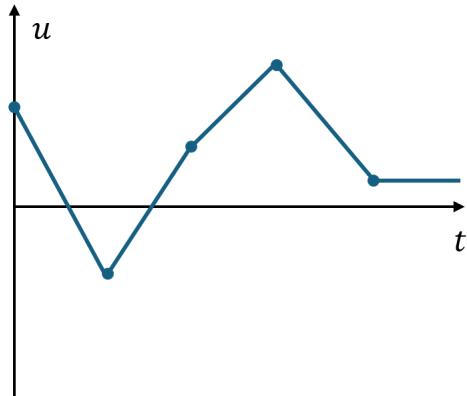


图 9: 这是一个first-order hold的示意图。它表示在每个时间步内，控制输入 $u(t)$ 是一个线性函数，直到下一个时间步到来。这个方法在数字控制系统中也很常见，因为它可以更好地捕捉系统的动态行为。通过使用一阶保持方法，我们可以更好地捕捉系统的动态行为，尤其是在处理快速变化的系统时。与零阶保持相比，一阶保持方法能够更准确地模拟系统的响应，并减少控制输入的突变，从而提高系统的稳定性和性能。

高阶保持，使用高阶的多项式来近似控制输入。然而在Bang-Bang控制中（比如说击球的这一瞬间，球换向快速弹回），两个极端值快速变换，其最优控制率不是光滑的，此时高阶保持甚至不如零阶保持

2.3 接触动力学仿真

为解决碰撞这一行为的仿真，要使用接触动力学仿真方法，这一节是后面的接触动力学的内容，可以阅读9.1节，不影响理解。

3 数值优化基础

这一节将介绍常用的数值优化方法。从一个 root-finding 问题开始，然后介绍

- 不动点迭代方法
- 牛顿方法
 - 最小化
 - 充分条件
 - 正则化
 - overshooting 问题

3.1 符号声明

这一节介绍多个符号规定，记住就好。

一元函数导数维度 考虑 $f(x): \mathbb{R}^n \rightarrow \mathbb{R}$, 则

$$\frac{\partial f}{\partial x} \in \mathbb{R}^{1 \times n} \Rightarrow \text{梯度是行向量}.$$

因为 $\frac{\partial f}{\partial x}$ 是将 Δx 映射到 Δf 的线性算子。并且通过泰勒展开我们可以得到：

$$f(x + \Delta x) \approx f(x) + \frac{\partial f}{\partial x} \Delta x,$$

因为 Δx 为列向量，所以如果要让 $f(x + \Delta x)$ 合法，则 $\frac{\partial f}{\partial x}$ 必然是行向量，

多元函数偏导（雅可比）维度 类似地，设 $g(y): \mathbb{R}^m \rightarrow \mathbb{R}^n$, 其雅可比为

$$J := \frac{\partial g(y)}{\partial y} \in \mathbb{R}^{n \times m}, \quad J = \begin{pmatrix} \frac{\partial g_1}{\partial y_1} & \frac{\partial g_1}{\partial y_2} & \cdots & \frac{\partial g_1}{\partial y_m} \\ \frac{\partial g_2}{\partial y_1} & \frac{\partial g_2}{\partial y_2} & \cdots & \frac{\partial g_2}{\partial y_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_n}{\partial y_1} & \frac{\partial g_n}{\partial y_2} & \cdots & \frac{\partial g_n}{\partial y_m} \end{pmatrix}$$

因为：

$$g(y + \Delta y) \approx g(y) + \frac{\partial g}{\partial y} \Delta y.$$

中 Δy 是 $m * 1$, 所以雅可比必然是 $n * m$ 。

链式法则（Chain Rule）：下面公式演示了复合函数的一阶泰勒展开

$$f(g(y + \Delta y)) \approx f(g(y)) + \frac{\partial f}{\partial x} \cdot \frac{\partial g}{\partial y} \cdot \Delta y.$$

nabla 记号 ∇

$$\nabla f(x) = \left(\frac{\partial f}{\partial x} \right)^\top \in \mathbb{R}^{n \times 1} \text{ 列向量,}$$

$$\nabla^2 f(x) = \frac{\partial}{\partial x}(\nabla f(x)) = \frac{\partial^2 f}{\partial x^2} \in \mathbb{R}^{n \times n}.$$

3.2 Root Finding 问题

考虑根问题 (Root Finding): 给定函数 $f(x)$, 希望找到 x^* 使得

$$f(x^*) = 0 \Rightarrow x^* \text{ 是根。}$$

例如: 连续时间动力系统的平衡点

$$\dot{x} = f(x), \quad \text{令 } f(x^*) = 0 \Rightarrow x^* \text{ 是平衡。}$$

这个问题与不动点问题密切相关, 即求解满足

$$f(x^*) = x^*$$

的 x^* , 例如离散时间动力系统的平衡点。

不动点迭代 (Fixed-Point Iteration) 最朴素的做法是不断迭代:

$$x_{k+1} = f(x_k),$$

若动态系统是稳定的, 则不断前向模拟该系统, 将会收敛到某个平衡点。

不动点收敛性分析 考虑映射 $x_{k+1} = f(x_k)$, 若存在不动点 x^* 满足 $f(x^*) = x^*$, 我们希望该迭代能收敛到 x^* 。

定理 (Banach 不动点定理): 设 $f: \mathbb{R}^n \rightarrow \mathbb{R}^n$ 是压缩映射, 即存在 $0 < L < 1$, 使得对任意 x, y , 有

$$\|f(x) - f(y)\| \leq L\|x - y\|,$$

则 f 有唯一不动点 x^* , 且对任意初始点 x_0 , 迭代 $x_{k+1} = f(x_k)$ 收敛于 x^* 。

收敛性证明: 记 x^* 为不动点, 考虑误差 $e_k := x_k - x^*$, 则

$$e_{k+1} = x_{k+1} - x^* = f(x_k) - f(x^*) \Rightarrow \|e_{k+1}\| \leq L\|e_k\| \leq L^2\|e_{k-1}\| \leq \dots \leq L^k\|e_0\|.$$

因此有:

$$\|x_k - x^*\| \leq L^k\|x_0 - x^*\| \rightarrow 0 \quad \text{当 } k \rightarrow \infty.$$

这说明只要映射满足 Lipschitz 常数 $L < 1$, 就能保证收敛性, 且为线性收敛。

梯度意义下的不动点 在梯度下降的意义下, 若将梯度流离散化:

1. 前向欧拉:

$$x_{k+1} = x_k - h\nabla f(x_k),$$

这就是梯度下降 (Gradient Descent) 的离散形式。

注: 可添加动画或直观演示。

2. 后向欧拉：类似于近似梯度法，也可看作是一种近似的「梯度流」方法（continuous limit）。

欧拉法中的根寻找问题：以非线性动力系统为例 我们考虑如下非线性常微分方程（ODE）：

$$\frac{dx}{dt} = f(x) = 0.5x \left(1 - \frac{x}{3}\right), \quad x(0) = 0.5.$$

该系统的平衡点满足 $f(x^*) = 0$ ，即 $x^* = 0$ 和 $x^* = 3$ ，这也正是一个‘根寻找（Root Finding）问题’。

数值方法一：前向欧拉法（Forward Euler）

$$x_{k+1} = x_k + hf(x_k)$$

收敛性分析：

若 f 在邻域内满足 Lipschitz 条件，即存在 $L > 0$ 使得

$$|f(x) - f(y)| \leq L|x - y|,$$

则前向欧拉法收敛。当 $hL < 1$ 时，具有稳定性。

对应 Python 代码： 关键部分的代码

```

1 # [optimization\forward_backward.py]
2 def forward_euler(x0, t_end, h):
3     t = np.arange(0, t_end, h)
4     x = np.zeros_like(t)
5     x[0] = x0
6
7     for i in range(1, len(t)):
8         x[i] = x[i-1] + h * f(x[i-1]) # Forward Euler update
9
10    return t, x

```

数值方法二：后向欧拉法（Backward Euler）

$$x_{k+1} = x_k + hf(x_{k+1})$$

我们需要在每一步中解非线性方程

$$x = x_k + h \cdot 0.5x \left(1 - \frac{x}{3}\right) \Rightarrow \frac{h}{6}x^2 + \left(1 - \frac{h}{2}\right)x - x_k = 0$$

收敛性分析：

设 $g(x) = x_k + hf(x)$ ，可视为不动点迭代 $x = g(x)$ 。

若 f 的导数有界，满足

$$|g'(x)| = |hf'(x)| < 1,$$

则根据 Banach 不动点定理，迭代过程收敛。

对应 Python 代码： 关键部分的代码

```

1 # [optimization\forward_backward.py]
2 def backward_euler(x0, t_end, h):
3     t = np.arange(0, t_end, h)
4     x = np.zeros_like(t)
5     x[0] = x0
6
7     for i in range(1, len(t)):
8         # Solve: x = x_prev + h * 0.5 * x * (1 - x / 3)

```

```

9     a = h / 6
0     b = (1 - h / 2)
1     c = -x[i-1]
2     discriminant = b**2 - 4*a*c
3     if discriminant >= 0:
4         x[i] = (-b + np.sqrt(discriminant)) / (2*a) # Take positive root
5     else:
6         x[i] = x[i-1] # No real root, fallback
7
8 return t, x

```

方法比较总结：

- 前向欧拉法：显式方法，简单高效，但稳定性对步长敏感
- 后向欧拉法：隐式方法，稳定性强，每步需解根问题
- 根问题体现：后向欧拉需解方程 $g(x) = x$ ，可视为不动点问题，若满足收敛条件可用不动点迭代或解析求解

为什么梯度下降在大规模问题中更受欢迎？在大规模机器学习中，梯度下降（Gradient Descent）相比牛顿方法（Newton's Method）更常用，原因包括：

- **计算更便宜：**牛顿法需要计算 Hessian 并求逆，代价为 $\mathcal{O}(n^3)$ ，而梯度下降只需计算一阶导数。（将在后面的公式中看到这个复杂度）
- **高维不可接受：**在高维情况下，牛顿方法由于计算复杂度过高，不适合使用。
- 此外，对于非凸函数（non-convex function）：
 - **多个平衡点 (equilibria)：**若希望找到全局最小值，需要在多个平衡点之间选择，等价于 NP 难问题。
 - **几乎需要穷举搜索 (exhaustive search)：**没有启发式或凸性结构支持时，只有遍历所有不动点才能确定最优解。
 - **仅适用于稳定的不动点：**牛顿迭代法仅在初始点足够靠近稳定的不动点时才有效；否则会发散。
 - **收敛速度慢：**特别在函数曲率变化大或存在鞍点时，牛顿法的收敛性无法保证。

3.2.1 Newton 方法

Newton 方法的基本思想是对 $f(x)$ 作线性近似，并设为零从而求解 Δx ，用于迭代更新。

由线性近似，在当前点 x 附近，采用一阶泰勒展开：

$$f(x + \Delta x) \approx f(x) + \frac{\partial f}{\partial x} \Delta x.$$

然后立即可以得到牛顿方法的两个步骤

- 令近似为零，转为 root-finding 问题，得到第一步，计算

$$f(x) + \frac{\partial f}{\partial x} \Delta x = 0 \Rightarrow \Delta x = - \left(\frac{\partial f}{\partial x} \right)^{-1} f(x)$$

注：这里求雅可比的逆是 $O(n^3)$

- 第二步，更新变量：

$$x \leftarrow x + \Delta x,$$

- 反复迭代直到收敛。

注：若导数矩阵不可逆或病态，可采用正则化策略：

如添加零平方项以求伪逆（pseudo-inverse）：即 Tikhonov regularization。

Backward Euler 步的固定点迭代示例 考虑使用 backward Euler 步来构造一个固定点迭代过程。目标是求解如下形式的不动点：

$$x_n = x_0 + hf(x_n),$$

其中 h 是步长参数。

- 初始化： $x_n = x_0$

- 误差定义：

$$\text{err} = \|x_0 + hf(x_n) - x_n\|_2$$

- 迭代至收敛：

$$\begin{aligned} \text{while } \text{err} > \epsilon : \quad &x_n \leftarrow x_0 + hf(x_n) \\ &\text{err} = \|x_0 + hf(x_n) - x_n\|_2^2 \end{aligned}$$

该迭代形式对应于隐式欧拉法或非线性方程的固定点迭代形式，可看作是牛顿方法的近似替代方案。

Backward Euler + Newton 方法联合迭代（backward_euler_step_newton_method）
目标：求解隐式方程

$$x_n = x_0 + hf(x_n)$$

这是 Backward Euler 步的形式，我们使用 Newton 方法求解该不动点方程。

- 定义残差函数：

$$r(x_n) := x_0 + hf(x_n) - x_n$$

- 迭代步骤：

$$\begin{aligned} &x_n \leftarrow x_0 \\ \text{while } &\|r(x_n)\| > \epsilon \text{ do:} \\ &\text{计算残差 } r = x_0 + hf(x_n) - x_n \\ &\text{计算雅可比 } \partial r = \frac{\partial r}{\partial x_n} = h \frac{\partial f}{\partial x}(x_n) - I \\ &\text{牛顿更新: } \quad x_n \leftarrow x_n - (\partial r)^{-1} r \end{aligned}$$

这个过程将 Backward Euler 构造的非线性方程，通过 Newton 方法迭代求解，适用于刚性系统或更稳定收敛的时间积分方法。

Backward Euler 与 Newton 方法的比较

- Backward Euler 方法虽然具有 **人工阻尼 (artificial damping)** 效果，但是收敛很慢（— $\text{Re}(\text{eig}(R))$ —可以看出来），较稳定。
- 固定点迭代方法（如 $x_{k+1} = f(x_k)$ ）通常 **不能直接找到极小值**，其收敛速度为 **线性收敛 (linear convergence)**，如图所示呈指数衰减但较慢。
- 相比之下，Newton 方法可在 **约 3 步迭代内** 快速接近最优解，具有 **二次收敛率 (quadratic convergence)**，即误差下降速度为：

$$\|x_{k+1} - x^*\| \leq C \|x_k - x^*\|^2.$$

Newton 方法的总结与优势

- Newton 方法配合 backward Euler 等技术，收敛非常快。
- 拥有 **二次收敛 (Quadratic Convergence)**，即误差下降速度极快。
- 可以达到 **机器精度 (machine precision)**，误差极小。
- 主要代价在于对 Jacobian 的因式分解 / 求逆操作，复杂度约为：

$$\mathcal{O}(n^3)$$

- 若能 **利用问题结构 (problem structure)**，可进一步降低复杂度，后续会展开说明。

隐式欧拉在非线性动力系统中的求根演示（单摆） 我们考虑经典非线性系统——单摆，其动力学形式为：

$$\dot{\theta} = \omega, \quad \dot{\omega} = -\frac{g}{l} \sin(\theta) \Rightarrow \dot{x} = f(x), \quad x = \begin{bmatrix} \theta \\ \omega \end{bmatrix}$$

使用 Backward Euler 离散化后，每一步需要求解如下形式的不动点问题：

$$x_{n+1} = x_n + h f(x_{n+1}) \quad \Rightarrow \quad \text{Root-Finding:} \quad r(x) = x_n + h f(x) - x = 0$$

方法一：固定点迭代 设 $g(x) = x_n + h f(x)$ ，则为不动点迭代：

$$x_{k+1} = g(x_k)$$

收敛条件：若 f 是 Lipschitz 连续，且 h 足够小使得 $|g'(x)| = |h f'(x)| < 1$ ，则根据 Banach 不动点定理，该迭代可收敛。

关键代码：

```

1 def backward_euler_step_fixed_point(fun, x0, h):
2     xn = x0.copy()
3     while np.linalg.norm(x0 + h * fun(xn) - xn) > tol:
4         xn = x0 + h * fun(xn) # Fixed-point iteration
5     return xn

```

方法二：Newton 法 将方程 $r(x) := x_n + hf(x) - x = 0$ 应用 Newton 方法：

$$x_{k+1} = x_k - \left(\frac{\partial r}{\partial x}(x_k) \right)^{-1} r(x_k)$$

其中：

$$\frac{\partial r}{\partial x} = h \frac{\partial f}{\partial x} - I$$

我们使用 `autograd` 自动求导，构造残差雅可比并执行线性求解。

关键代码：

```

1 def backward_euler_step_newton(fun, x0, h):
2     def residual(x): return x0 + h * fun(x) - x
3     J = jacobian(residual) # via autograd
4     x = x0.copy()
5     while np.linalg.norm(residual(x)) > tol:
6         x -= np.linalg.solve(J(x), residual(x)) # Newton step
7     return x

```

实验比较与结论： 我们对上述两种方法在单摆动力系统中进行比较。结果显示：

- **固定点法：** 收敛缓慢，对步长敏感，迭代次数较多；
- **Newton 法：** 收敛快，仅需少数迭代步，适用于刚性系统；
- **误差曲线：** Newton 法呈现典型的二次收敛，误差指数下降；
- **可视化：** 使用 PendulumVisualizer 工具包对系统动态进行了动画演示。

如图所示（略），两种方法在同一初值和时间步长下收敛路径基本一致，但效率差异明显。

（注：完整代码见 `optimization/pendulum_forward_backward_newton.py`）

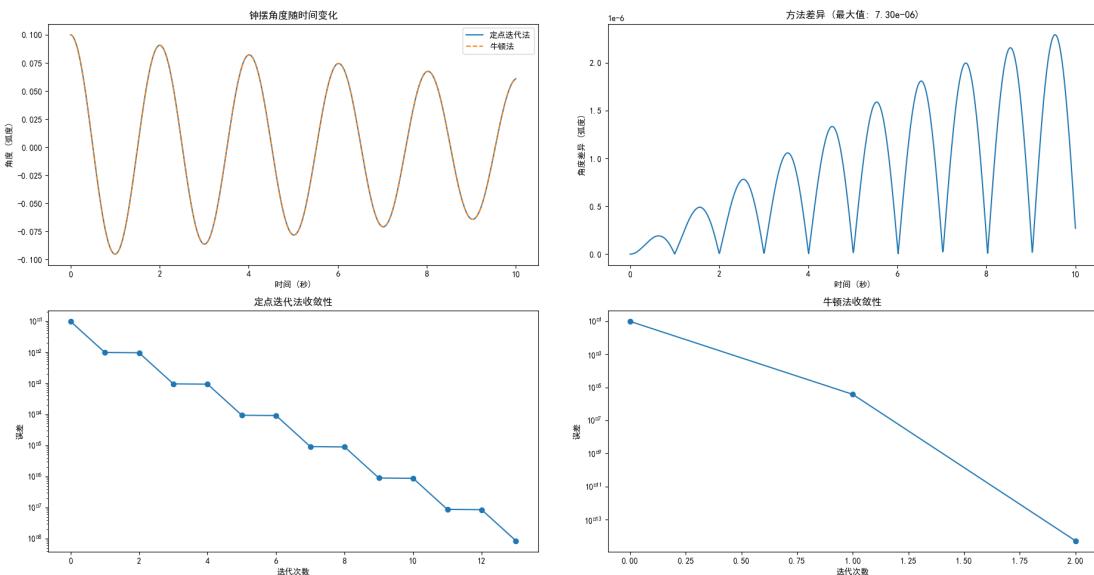


图 10：单摆系统中两种求根方法的比较：Fixed-Point 与 Newton 法

如图 10 所示，我们比较了 fixed-point 与 Newton 方法在隐式欧拉时间积分中的表现：

- **左上角:** 显示两种方法所模拟出的摆角随时间的演化曲线。两条曲线几乎重合，说明在相同步长和初始条件下，两者均可得到稳定且精确的系统轨迹。
- **右上角:** 显示两种方法得到的角度差异（误差），其最大值约为 7.3×10^{-6} ，属于数值误差范围，说明精度相当。
- **左下角:** 为固定点方法在某一步迭代中的误差衰减过程（以 0.1 步长模拟）。可以看出误差呈线性下降，符合固定点方法的线性收敛理论。
- **右下角:** 为牛顿方法的误差下降过程，仅用了 3 步就达到机器精度。呈现二次收敛特性，即误差以平方速度下降。

结论: 若系统非线性较强，或对精度/收敛速度有要求，建议使用 Newton 法；而在结构简单、实时性要求高时，fixed-point 方法仍具备价值。

3.2.2 牛顿方法在最小化问题中的应用

考虑优化问题

$$\min_x f(x), \quad f(x): \mathbb{R}^n \rightarrow \mathbb{R}$$

- 若 f 是光滑的，则要满足 $\frac{\partial f}{\partial x}|_{x^*} = 0$ 在局部最小值成立 (KKT)。
- 可将该问题转化为 root-finding 问题： $\nabla f(x) = 0$

应用 Newton 方法：

1. 计算更新量

$$\begin{aligned} \nabla f(x + \Delta x) &\approx \nabla f(x) + \nabla^2 f(x) \Delta x = 0 \\ \Rightarrow \Delta x &= -[\nabla^2 f(x)]^{-1} \nabla f(x) \end{aligned}$$

2. 更新

$$x \leftarrow x + \Delta x$$

解释: 牛顿方法 = 二次近似最小化 牛顿方法可以理解为在当前点 x 对 $f(x)$ 作局部二次近似：

$$f(x + \Delta x) \approx f(x) + \nabla f(x)^\top \Delta x + \frac{1}{2} \Delta x^\top \nabla^2 f(x) \Delta x$$

然后最小化该二次函数，得到的最优 Δx 就是 Newton 步。

例子：

$$f(x) = x^4 + x^3 - x^2 - x, \quad \nabla f(x) = 4x^3 + 3x^2 - 2x - 1, \quad \nabla^2 f(x) = 12x^2 + 6x - 2$$

例子：一维多项式函数的牛顿法最小化

我们考虑如下的实值一元多项式函数

$$f(x) = x^4 + x^3 - x^2 - x$$

其一阶导数（梯度）为

$$\nabla f(x) = 4x^3 + 3x^2 - 2x - 1$$

二阶导数 (Hessian, 在一维情况下即为标量) 为

$$\nabla^2 f(x) = 12x^2 + 6x - 2$$

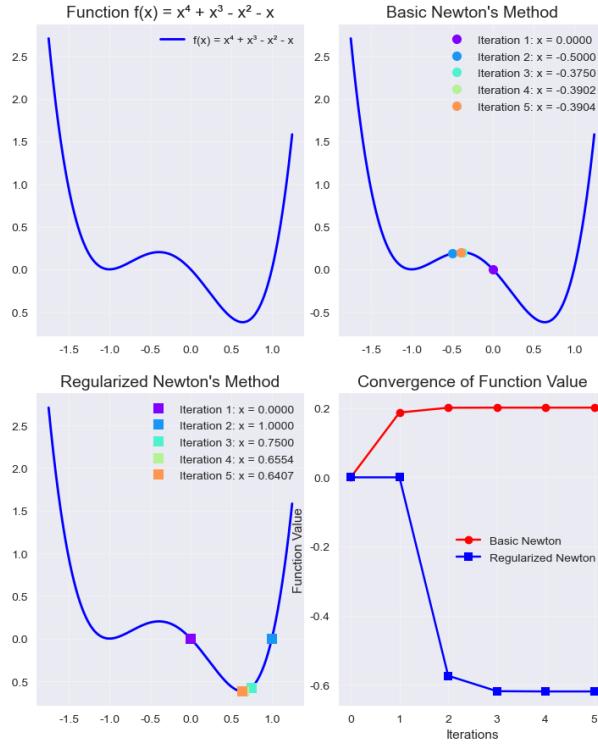
此函数为四次多项式，具有多个临界点。令 $\nabla f(x) = 0$ ，即对应牛顿法中所需求解的 root-finding 问题，其解可能为极小值点、极大值点或鞍点。Hessian 的符号决定了该点的曲率性质，若 $\nabla^2 f(x^*) > 0$ ，则该点为局部极小值。

图像分析表明该函数在 $x \approx -1.5, 0, 1.0$ 附近存在多个极值点，但牛顿法对初始点敏感，不同初值可能收敛至不同点。

核心代码实现 (Python) 我们对该函数进行数值实现，仅保留牛顿更新部分，具体如下：

```
1 def f(x):
2     return x**4 + x**3 - x**2 - x
3 def grad_f(x):
4     return 4*x**3 + 3*x**2 - 2*x - 1
5 def hessian_f(x):
6     return 12*x**2 + 6*x - 2
7
8 # basic newton method
9 def newton_step(x0):
10    return x0 - grad_f(x0) / hessian_f(x0)
11
12 # normalized newton method, to avoid Hessian matrix is negative
13 def regularized_newton_step(x0):
14     beta = 1.0
15     H = hessian_f(x0)
16     while H <= 0:
17         H += beta
18         beta *= 2
19     return x0 - grad_f(x0) / H
```

图像分析与比较 我们从初始点 $x_0 = 0$ 开始，分别使用基本牛顿法与正则化牛顿法进行 5 次迭代。可视化结果如下图所示：



并且我们绘制出此处的Hessian:

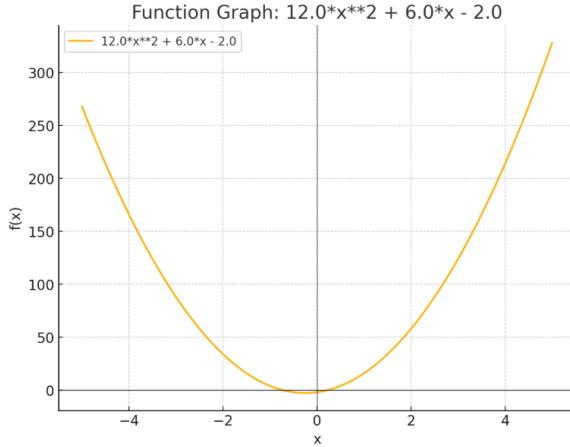


图 11: 单元多次函数的Hessian可视化，可以看到有负的情况，如果是矩阵则添加I的倍数，此处因为是单变量，添加1的倍数即可。

数值结果：

基本牛顿法: $x^* \approx -0.3904, f(x^*) \approx 0.2017$

正则化牛顿法: $x^* \approx 0.6404, f(x^*) \approx -0.6197$

结论：

- 函数 $f(x)$ 存在多个临界点，且 Hessian 可为负，说明基本牛顿法存在收敛到非极小值点的风险；
- 正则化牛顿法通过调整 Hessian，避免非正定性所带来的不稳定收敛问题；

- 在该例中，正则化方法显著提高了迭代稳定性与最终函数值质量。

结论：Newton 方法是 局部 root-finding 方法，对 $\nabla f(x) = 0$ 求解，其收敛结果取决于初始点：

- 可能收敛到极小值、极大值或鞍点。
- 要求初始点足够接近目标点。

3.3 最小值存在的充分条件

- 一阶必要条件： $\nabla f(x^*) = 0$ 是最小值的必要条件，但不是充分条件！
- 看一个标量情况：

$$\Delta x = -(\nabla^2 f)^{-1} \cdot \nabla f$$

梯度方向决定前进方向，Hessian 决定“步长”或“学习率”。

- 若 Hessian 为正：

$$\nabla^2 f > 0 \Rightarrow \text{descent (最小化)}$$

若 Hessian 为负：

$$\nabla^2 f < 0 \Rightarrow \text{ascent (最大化)}$$

- 更进一步，在 \mathbb{R}^n 中，若

$$\nabla^2 f \succ 0 \quad (\text{严格正定}) \Rightarrow \nabla^2 f \in \mathcal{S}_{++}$$

¹ 即所有特征值均为正，是局部极小的充分条件。因此，局部极小的二阶充分条件是：

在极小值点，Hessian 的所有特征值为正，即

\Rightarrow 方向为下降方向 (descent)

3.4 正则化 (Regularization) 与强凸性

- 若 $\nabla^2 f > 0$ 在整体上成立 $\Leftrightarrow f(x)$ 是强凸函数 (strongly convex)，则可用牛顿法全局求解。
- 但这对非凸或非线性问题通常不成立，因此不能总是依赖牛顿法。
- 正则化的动机：**为了让 Hessian 始终正定，从而确保下降方向，可以进行如下处理：

- 设 $H = \nabla^2 f$ ，若 $H \not\succ 0$ ，则执行：

$$\text{while } H \not\succ 0 : \quad H \leftarrow H + \beta I$$

其中 $\beta > 0$ ， I 为单位矩阵， β 可逐步增大。

¹关于矩阵正定的定义与性质请参考《矩阵分析》相关书籍与课程，此处不赘述。

- 更新步长:

$$\Delta x = -H^{-1}\nabla f, \quad x \leftarrow x + \Delta x$$

- 这种方法称为 **Tikhonov 正则化** (**Tikhonov regularization**), 在最小二乘 (least squares) 等问题中也常见, 体现为给权重加上二次惩罚项。

- **damped Newton** 方法, 通过调节步长或对 Hessian 添加扰动来控制更新方向, 使其总是一个下降步 (descent step):

- 它保证是下降方向, 同时还能让步长变得更小 (make step size smaller)。
- 给 Hessian 加上单位阵 (positive identity) 本质上等价于在优化目标中添加一个二次惩罚项 (quadratic penalty):

例如 $\beta\Delta x^\top \Delta x$ 被加到目标函数上 \Rightarrow 惩罚步长过大, 避免 overshoot

为什么给 Hessian 加上 βI 等价于加惩罚项? 在牛顿方法中, 我们在 x 附近用二阶泰勒展开近似目标函数:

$$f(x + \Delta x) \approx f(x) + \nabla f(x)^\top \Delta x + \frac{1}{2} \Delta x^\top \nabla^2 f(x) \Delta x$$

此时求解的是使该近似最小的 Δx 。

为了增强数值稳定性, 或者避免 Hessian 非正定带来的问题, 我们会对 Hessian 做如下正则化:

$$\tilde{H} = \nabla^2 f(x) + \beta I$$

此时对应的优化问题变为最小化:

$$\begin{aligned} & f(x) + \nabla f(x)^\top \Delta x + \frac{1}{2} \Delta x^\top (\nabla^2 f(x) + \beta I) \Delta x \\ &= f(x) + \nabla f(x)^\top \Delta x + \frac{1}{2} \Delta x^\top \nabla^2 f(x) \Delta x + \frac{1}{2} \beta \|\Delta x\|^2 \end{aligned}$$

因此, 给 Hessian 加 βI 等价于在原函数上添加一项二次惩罚项:

$$\frac{1}{2} \beta \|\Delta x\|^2$$

这样做的意义是:

- * 控制 Δx 的幅度, 防止步长过大 (overshooting)
- * 保证 Hessian 正定, 从而确保下降方向
- * 提高病态 Hessian 的数值稳定性

```

1 # aim function
2 def f(x): return x**4 + x**3 - x**2 - x
3 def grad_f(x): return 4*x**3 + 3*x**2 - 2*x - 1
4 def hessian_f(x): return 12*x**2 + 6*x - 2
5
6 # Damped Newton
7 def damped_newton_step(x0, beta=5.0):
8     H = hessian_f(x0)

```

```

9      # normalized Hessian
10     H_damped = H + beta
11     return x0 - grad_f(x0) / H_damped
12
13 # compare to different newton path
14 x0 = 0.0
15 path_beta_1 = [x0]
16 path_beta_5 = [x0]
17
18 for _ in range(5):
19     x0 = damped_newton_step(path_beta_1[-1], beta=1.0)
20     path_beta_1.append(x0)
21
22 x0 = 0.0
23 for _ in range(5):
24     x0 = damped_newton_step(path_beta_5[-1], beta=5.0)
25     path_beta_5.append(x0)

```

- 总结：这种方法的效果是 **抑制步长过大**，尤其适合 Hessian 非正定或发散风险高的情形。

3.5 线性搜索（Line Search） Backtracking方法

为了解决牛顿法中可能的 overshooting（步长过大）问题，引入线性搜索（Line Search）策略。

基本思想： 沿当前方向 Δx ，不直接采用全量更新，而是寻找一个缩放因子 $\alpha \in (0, 1]$ ，使得目标函数有足够的下降。

下降条件： 基于一阶泰勒展开，有如下下降近似：

$$f(x + \Delta x) \approx f(x) + \nabla f(x)^\top \Delta x$$

我们要求实际下降量不小于这个估计值的一定比例，即：

$$f(x + \alpha \Delta x) \leq f(x) + \beta \nabla f(x)^\top \Delta x$$

其中 $\beta \in (0, 1)$ 是容差系数（通常取 $\beta \approx 10^{-4}$ ），该条件称为 Armijo 条件。

算法伪代码： 下面展示了Armijo算法伪代码回溯线搜索是一种用于解决无约束优化问题的步长选择技术。该算法的主要步骤如下：

- 初始化：** 设置初始步长 $\alpha = 1$ ，以及减小系数 $\tau \in (0, 1)$ （通常为0.5）和Armijo条件参数 $\beta \in (0, 1)$ （通常为0.1）。
- 迭代过程：** 重复检查Armijo条件： $f(x + \alpha \Delta x) > f(x) + \beta \alpha \nabla f(x)^\top \Delta x$ 。如果条件成立，表示当前步长过大，需要减小步长： $\alpha \leftarrow \tau \cdot \alpha$ 。
- 终止条件：** 当找到合适的步长（即不满足上述条件）时，返回新的解 $x_{\text{new}} = x + \alpha \Delta x$ 。

在优化过程中， Δx 通常是负梯度方向 $-\nabla f(x)$ 或其他搜索方向。Armijo条件保证了函数值有足够的下降，从而确保算法的收敛性。参数 β 控制了对下降幅度的要求，而 τ 控制了步长减小的速度。

Backtracking Line Search Algorithm

```
Input :  $x, \Delta x, f(x), \nabla f(x)$ 
Output:  $x_{\text{new}} = x + \alpha \Delta x$ 
 $\alpha \leftarrow 1; \tau \in (0, 1); \beta \in (0, 1);$ 
while  $f(x + \alpha \Delta x) > f(x) + \beta \alpha \nabla f(x)^T \Delta x$  do
     $\quad \alpha \leftarrow \tau \cdot \alpha;$ 
return  $x + \alpha \Delta x$ 
```

简写记号：也可写成如下形式，强调目标函数下降满足一阶下降近似的比例容差：

$$f(x + \alpha \Delta x) \leq f(x) + \beta \nabla f(x)^T \Delta x \quad (\text{下降容忍})$$

或者在笔记中常见的简写方式：

$$f(x + \Delta x) \leq f(x) + \beta \nabla f(x)^T \Delta x$$

该条件是基于一阶导数的估计下降，也称作：

$$f(x + x) \leq f(x) + \beta \nabla f(x)^T \Delta x$$

4 约束优化问题

因为有约束的优化问题是一个复杂的问题，我们这里独立作为一章进行讲解。在section 3中，我们讨论了无约束的优化问题，然而现实中大多数是有约束的优化问题，比如说机械臂的速度不能过快、奇异位置限制等等。所以这一章，我们主要讲述有约束的优化问题。

4.1 等式约束问题

考虑以下最优化问题，受等式约束的限制：

$$\begin{aligned} \min_x f(x) : f(x) : \mathbb{R}^n &\rightarrow \mathbb{R} \\ \text{s.t. } c(x) = 0 : \mathbb{R}^n &\rightarrow \mathbb{R}^m \end{aligned}$$

4.1.1 一阶必要条件

在函数达到最小值的点 x 处的一阶必要条件：

- 需要 $\nabla f(x) = 0$ 在无约束的/自由方向上。
- 需要 $c(x) = 0$ 。

如果 $c(x)$ 是直线，那么它肯定与 $c(x)$ 的切线相切， $f(x)$ 在该点达到了最小值（图12）。与此相同的是，如果 $c(x)$ 是N维的，则最小化函数的切线意味着它们的切向量（导数）必须共线。因此， $\nabla f(x)$ 的任何非零分量都必须垂直于约束面（constraint surface）/流形（mainfold）。

$$\nabla f + \lambda \nabla c = 0 \quad \text{对于某个 } \lambda \in \mathbb{R}$$

其中 λ 是拉格朗日乘子或“对偶变量”。通常，考虑到多维情况，我们有：

$$\frac{\partial f}{\partial x} + \lambda^T \frac{\partial c}{\partial x} = 0 \quad \lambda \in \mathbb{R}^m$$

基于该梯度条件，我们定义拉格朗日函数：

$$L(x, \lambda) = f(x) + \lambda^T c(x)$$

该函数受到以下约束：

$$\text{s.t. } \begin{cases} \nabla_x L(x, \lambda) = \nabla f + \lambda^T \frac{\partial c}{\partial x} = 0 \\ \nabla_x L(x, \lambda) = c(x) = 0 \end{cases}$$

这些条件共同表示了KKT条件。我们现在可以使用牛顿法联合求解 x 和 λ 作为根求解问题：

$$\begin{aligned} \nabla_x L(x + \Delta x, \lambda + \Delta \lambda) &\approx \nabla_x L(x, \lambda) + \frac{\partial^2 L}{\partial x^2} \Delta x + \frac{\partial^2 L}{\partial x \partial \lambda} \Delta \lambda = 0 \\ \nabla_x L(x + \Delta x, \lambda + \Delta \lambda) &\approx c(x) + \frac{\partial c}{\partial x} \Delta x = 0 \Rightarrow \frac{\partial c}{\partial x} \Delta x = -c(x) \end{aligned}$$

这可以写成以下矩阵系统：

$$\begin{bmatrix} \frac{\partial^2 L}{\partial x^2} & \frac{\partial^2 L}{\partial x \partial \lambda} \\ -\nabla_x L(x, \lambda) & -c(x) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = 0$$

其中第一矩阵是拉格朗日函数的Hessian矩阵，描述了KKT系统。

梯度计算与可视化 为了进行梯度可视化，我们首先计算目标函数和约束条件的梯度。在优化问题中，目标函数的梯度表示为：

$$\nabla f(x_1, x_2) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2} \right] = [2(x_1 - 2), 2(x_2 - 1)]$$

约束条件的梯度为：

$$\nabla c(x_1, x_2) = \left[\frac{\partial c}{\partial x_1}, \frac{\partial c}{\partial x_2} \right] = [1, 1]$$

接下来，我们用Python代码计算这些梯度，并通过箭头将其可视化。

```

1 # 计算目标函数的梯度
2 gradient_x1 = 2 * (opt_x1 - 2) # df/dx1
3 gradient_x2 = 2 * (opt_x2 - 1) # df/dx2
4 constraint_grad_x1 = 1 # dc/dx1
5 constraint_grad_x2 = 1 # dc/dx2

```

图像与说明 在下方的图像中，我们可以看到目标函数、约束条件以及梯度向量的可视化。图像由三个子图组成：

1. 目标函数图：左侧的图展示了目标函数 $f(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 1)^2$ 的三维图形（自行运行代码，可以用鼠标拖动方向）。在目标函数的表面上，最优点被标记为红色圆点，表示最小化的解。
2. 等高线与约束条件图：中间的图展示了目标函数的等高线以及约束条件 $x_1 + x_2 = 3$ 的线。最优点同样被标记在图中。
3. 梯度与约束图：右侧的图展示了目标函数的梯度（蓝色箭头）和约束条件的梯度（红色箭头）。最优点位于两条梯度相互平衡的位置，符合KKT条件。

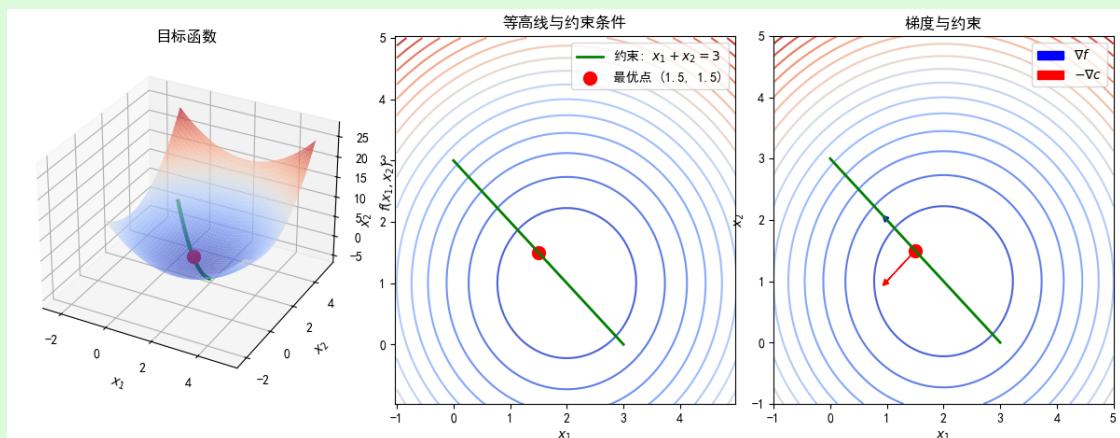


图 12：目标函数、等高线与约束条件以及梯度可视化图。

4.1.2 Gauss-Newton 方法的应用

Gauss-Newton原本是在牛顿法基础上修改，仅用于解决非线性最小二乘法问题，其不用计算二阶导数矩阵。

Gauss-Newton 方法在实际问题中广泛应用，特别是在优化问题和约束最优化中。相比于标准的牛顿法，Gauss-Newton 方法通过忽略二阶导数的影响（即忽略“约束曲率”）来简化计算，从而大大减少了每次迭代的计算量。尽管如此，Gauss-Newton 方法通常会导致稍慢的收敛速度，但每次迭代所需的计算量远小于标准牛顿法。

$$\frac{\partial^2 L}{\partial x^2} = \nabla^2 f + \frac{\partial}{\partial x} \left[\left(\frac{\partial c}{\partial x} \right)^T \lambda \right]$$

后面这一项是Hessian矩阵，计算复杂度太高！

应用示例： 我们考虑一个优化问题，目标函数为 $f(x)$ ，约束条件为 $c(x)$ ，Gauss-Newton 方法通过如下步骤来求解最优解：

$$\nabla_x L(x, \lambda) = \nabla f + \lambda^T \frac{\partial c}{\partial x} = 0$$

其中 $L(x, \lambda)$ 是拉格朗日乘子法的拉格朗日函数， λ 是拉格朗日乘子。

标准牛顿法与 Gauss-Newton 方法的对比 这里用公式对比两种方法的区别

标准牛顿法

目标函数和约束的数学公式：

$$\frac{\partial^2 L}{\partial x^2} = \nabla^2 f(x) + \frac{\partial}{\partial x} \left[\left(\frac{\partial c(x)}{\partial x} \right)^T \lambda \right]$$

更新步的 KKT 条件：

$$\begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} -\nabla f(x) - C^T \lambda \\ -c(x) \end{bmatrix}$$

Hessian 矩阵：

$$H = \nabla^2 f(x) + \frac{\partial}{\partial x} \left[\left(\frac{\partial c(x)}{\partial x} \right)^T \lambda \right]$$

更新公式：

$$\begin{bmatrix} x_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ \lambda_k \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix}$$

Gauss-Newton 方法

目标函数和约束的数学公式：

$$\frac{\partial^2 L}{\partial x^2} = \nabla^2 f(x)$$

更新步的 KKT 条件：

$$\begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} -\nabla f(x) - C^T \lambda \\ -c(x) \end{bmatrix}$$

Hessian 矩阵：

$$H = \nabla^2 f(x)$$

更新公式：

$$\begin{bmatrix} x_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ \lambda_k \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix}$$

```

1 # Newton方法实现 - 计算完整的Hessian矩阵
2 def newton_step(x0, lambda0):
3     # 确保x0是一维数组
4     x0 = np.ravel(x0)
5     lambda0 = np.array([lambda0]).ravel()
6
7     # 计算目标函数的Hessian矩阵 (包括二阶导数)
8     H = hessian_f(x0)
9
10    # 计算约束的雅可比矩阵
11    C = grad_c(x0).reshape(1, -1)

```

```

12
13     # 构建完整的KKT矩阵
14     KKT_matrix = np.block([
15         [H, C.T],
16         [C, np.array([[0]])]
17     ])
18
19     # 构建右侧向量
20     rhs = np.concatenate([
21         -grad_f(x0) - C.T @ lambda0,
22         -np.array([c(x0)])
23     ])
24
25     # 求解线性方程组
26     delta_z = np.linalg.solve(KKT_matrix, rhs)
27
28     delta_x = delta_z[:2]
29     delta_lambda = delta_z[2]
30
31     return x0 + delta_x, lambda0[0] + delta_lambda

```

```

1 # Gauss-Newton方法实现 - 忽略约束曲率 (二阶导数)
2 def gauss_newton_step(x0, lambda0):
3     # 确保x0是一维数组
4     x0 = np.ravel(x0)
5     lambda0 = np.array([lambda0]).ravel()
6
7     # 计算目标函数的Hessian矩阵 (仅使用一阶项)
8     H = hessian_f(x0)
9
10    # 计算约束的雅可比矩阵
11    C = grad_c(x0).reshape(1, -1)
12
13    # Gauss-Newton简化: 忽略约束的二阶导数影响
14    # 直接使用目标函数的Hessian而不添加约束的二阶导数影响
15
16    # 构建简化的KKT矩阵
17    KKT_matrix = np.block([
18        [H, C.T],
19        [C, np.array([[0]])]
20    ])
21
22    # 构建右侧向量
23    rhs = np.concatenate([
24        -grad_f(x0) - C.T @ lambda0,
25        -np.array([c(x0)])
26    ])
27
28    # 求解线性方程组
29    delta_z = np.linalg.solve(KKT_matrix, rhs)
30
31    delta_x = delta_z[:2]
32    delta_lambda = delta_z[2]
33
34    return x0 + delta_x, lambda0[0] + delta_lambda

```

实验结果与比较 我们设置初始猜测为 $[-1, -1]$ 和 $[-3, 2]$, 分别使用牛顿法和 Gauss-Newton 方法求解。实验结果表明, Gauss-Newton 方法在每次迭代中计算量较小, 但收敛速度略慢于标准牛顿法。我们比较了标准牛顿法和Gauss-Newton方法在约束优化问题中的收敛速度和表现。

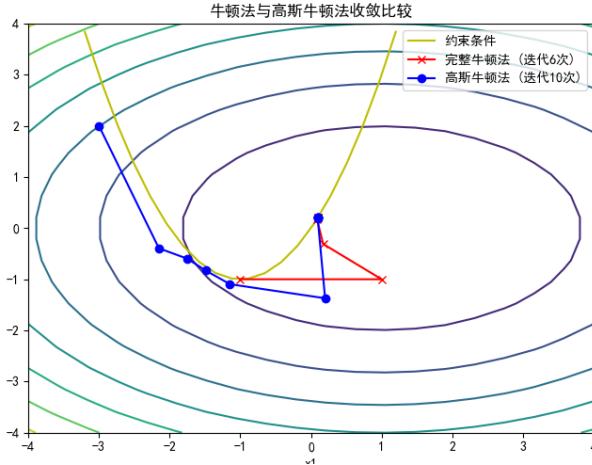


图 13: 牛顿法与高斯-牛顿法的优化路径对比（初始猜测为[-1, -1]）

从图13中可以看出，牛顿法的收敛速度较快，而Gauss-Newton方法则在迭代过程中收敛较慢，但计算量显著较低。

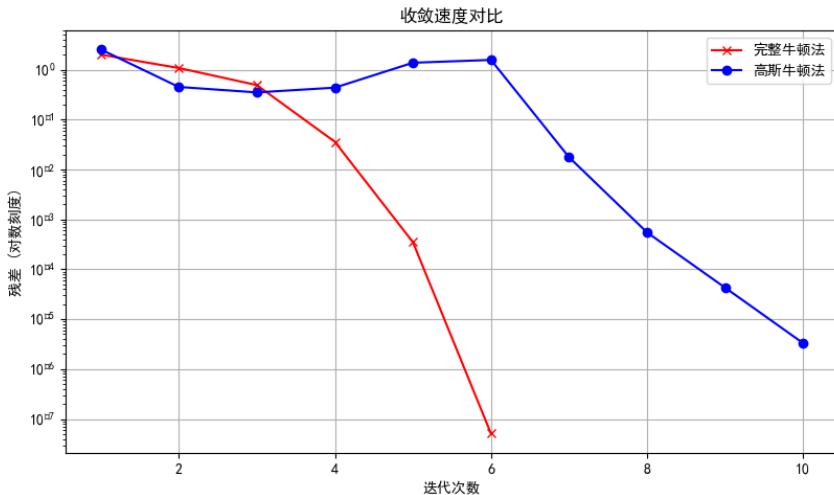


图 14: 标准牛顿法与Gauss-Newton方法的收敛速度对比

图14展示了标准牛顿法与Gauss-Newton方法在不同迭代次数下的收敛速度对比。可以看到，标准牛顿法在较少的迭代次数内达到了收敛，而Gauss-Newton方法则需要更多的迭代。

Take-Away Message:

- Gauss-Newton 方法在处理大规模优化问题时非常有效。
- 在一些情况下，可能需要对 $\frac{\partial^2 L}{\partial x^2}$ 进行正则化。
- 尽管 Gauss-Newton 方法的收敛速度较慢，但其每次迭代的计算量显著降低。
- 约束的曲率越大，由于忽略二次项而导致的偏差越大，从而性能越不好。

4.2 不等式约束

更一般地，我们必须调和不等式约束。考虑一个带有不等式约束的最小化问题：

$$\min_x f(x) \quad \text{s.t.} \quad c(x) \geq 0$$

让我们先看一下纯不等式约束的情况。通常，这些方法结合了等式约束方法，以处理在同一问题中同时包含不等式/等式约束的情况。

4.2.1 一阶必要条件

和之前一样，一阶必要条件要求：

- 在无约束/自由方向上，需要 $\nabla f(x) = 0$ 。
- 需要 $c(x) \geq 0$ ，注意这里的不等式。

从数学上讲，

$$\left\{ \begin{array}{ll} \nabla f(x) - \left(\frac{\partial c}{\partial x}\right)^T \lambda = 0 & \text{“Stationarity”} \quad (\text{驻点条件}) \\ c(x) \leq 0 & \text{“Primal Feasibility”} \quad (\text{原始可行性}) \\ \lambda \geq 0 & \text{“Dual Feasibility”} \quad (\text{对偶可行性}) \\ \lambda^T c(x) = 0 & \text{“Complementarity”} \quad (\text{互补条件}) \end{array} \right.$$

$\left(\frac{\partial c}{\partial x}\right)^T$ 是惩罚项，确保若 $c(x)$ 被违反，则对目标函数的惩罚项为正。这些条件一起指定了完整的KKT条件。注意，无论 $c(x)$ 或 λ 的符号如何，它都依赖于你习惯的方式，一条原则是它总是防止目标函数的变化。

4.2.2 直观理解

这些条件的直观理解如下：

- 如果约束是活动的（我们处于约束流形上），我们有 $c(x) = 0 \Rightarrow \lambda = 0$ 。这与等式约束的情况相同。
- 如果约束不活跃，我们有 $c(x) > 0 \Rightarrow \lambda = 0$ ，这与无约束的情况相同。
- 本质上，互补性确保了 λ 或 $c(x)$ 其中之一为零，或者约束的“开/关”切换。

4.3 算法

这里实现优化方法比等式约束问题更加复杂；我们不能直接将牛顿法应用到KKT条件下。我们有许多可供选择的方法，不同方法有不同的折衷。

4.3.1 活动集方法 (Active-Set Method)

使用场合： 当你知道哪些约束是活动的/不活动时，可以使用此方法。

适用问题： 解决等式约束问题（当它是活动时）。

优点： 如果你有一个好的启发式方法，它可以非常快速。

缺点： 如果你不知道哪些约束是活动的，它可能会非常复杂（组合性问题）。

活动约束的判定 判断一个约束是否是活动的，可以使用以下定义：

定义：如果约束 $c_i(x)$ 在某点 x^* 满足 $c_i(x^*) = 0$ ，则称该约束在点 x^* 是活动的。对于不等式约束 $c_i(x) \geq 0$ ，这意味着约束在边界上被严格满足。

定理（活动集判定）：在一个约束优化问题的最优解 x^* 处：

$$\min_x f(x) \text{ 满足 } c_i(x) \geq 0, i = 1, \dots, m,$$

活动约束的集合 \mathcal{A} 定义为：

$$\mathcal{A} = \{i \mid c_i(x^*) = 0\}.$$

对于所有 $i \notin \mathcal{A}$ ，有 $c_i(x^*) > 0$ ，并且对应的拉格朗日乘子 $\lambda_i = 0$ 。

例子：考虑以下优化问题：

$$\min_{x_1, x_2} f(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 2)^2$$

$$\text{满足 } c_1(x) = x_1 + x_2 - 3 \geq 0, \quad c_2(x) = x_1 \geq 0.$$

在候选解 $x^* = (1, 2)$ 处：

- $c_1(x^*) = 1 + 2 - 3 = 0$: c_1 是活动的。
- $c_2(x^*) = 1 \geq 0$: c_2 是非活动的。

因此，活动集为 $\mathcal{A} = \{1\}$ 。

4.3.2 障碍法 / 内点法 (Barrier / Interior-Point Method)

$$\min_x f(x) \quad \text{s.t.} \quad Cx \leq 0 \quad \Rightarrow \quad \min_x \left(f(x) - \frac{1}{\rho} \sum_{i=1}^m \log(-c_i(x)) \right)$$

因为添加了一个在边界处blow-up的项，所以不会越界。我们可以通过绘制 $f(x) = -\log(-x)$ 看到，越到边界0，函数值越大。我们可以通过调整 ρ 来控制惩罚的强度， ρ 越大，半径越大，约束越强，在实际中，常常从大到小来调整 ρ ，直到收敛为止

方法描述： 我们用一个“障碍函数”替换不等式约束，目标函数在约束边界处爆炸，见图14。

适用问题： 这是中等规模凸问题的标准方法。对MPC问题非常常用

缺点： 对于非凸问题，需要大量的技巧和技巧来使其工作。

4.4 惩罚方法 / 外点法 (Penalty / Exterior-Point Method)

外点法 (Penalty Method) 通过用惩罚项来替代约束条件，得到一个无约束的优化问题。具体来说，它通过惩罚违反约束的行为来进行优化。

4.4.1 外部惩罚方法

首先我们替换约束条件，使用惩罚项来代替，得到如下的优化问题：

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \text{s.t.} \quad c(\mathbf{x}) \leq 0$$

转化为

$$\min_{\mathbf{x}} f(\mathbf{x}) + \rho [\max(0, c(\mathbf{x}))]^2$$

其中， ρ 是惩罚系数，随着迭代的进行， ρ 通常逐渐增大，以增强对违反约束的惩罚。

惩罚方法易于实现，但存在一些问题，例如数值不稳定（由于 ρ 增大），并且无法达到高精度。特别是在 ρ 较大时，算法可能会变得非常不稳定。

$$\min_{\mathbf{x}} f(\mathbf{x}) + \rho [\max(0, c(\mathbf{x}))]^2$$

- 优点：

- 实现简单。
- 适用于一般的约束优化问题。

- 缺点：

- 数值不稳定。
- 无法达到高精度。
- 需要增加 ρ 来处理约束条件，这会导致问题的复杂度增加。
- 因为惩罚只是pull-back，所以会导致越界。

4.4.2 增广拉格朗日方法 (Augmented Lagrangian Method)

我们向惩罚方法中添加拉格朗日乘子估计，以解决惩罚方法所遇到的问题。具体而言，增广拉格朗日函数可以通过如下表达式表示：

$$\min_x f(x) + \tilde{\lambda}^T c(x) + \frac{\rho}{2} [\min(0, c(x))]^2$$

其中， $\tilde{\lambda}^T c(x)$ 称之为拉格朗日乘子估计值 (lagrangian multipliers estimation) 用于吸收约束，最终这个估计值 $\tilde{\lambda}$ 收敛于 λ 。 $\frac{\rho}{2} [\min(0, c(x))]^2$ 表示二次惩罚项， ρ 是惩罚参数。这两项结合在一起，形成了增广拉格朗日函数 $L_\rho(x, \lambda)$ 。首先，我们在固定的 λ 下对 x 进行最小化，然后每次迭代时通过“卸载”惩罚项来更新 λ 。

为了进一步更新拉格朗日乘子，我们知道如果是最小值，则其偏导一定为零，则有：

$$\begin{aligned} \frac{\partial f}{\partial x} - \tilde{\lambda} \frac{\partial c}{\partial x} + \rho c(x)^T \frac{\partial c}{\partial x} &= 0 \\ \Leftrightarrow \frac{\partial f}{\partial x} - \left[\tilde{\lambda} - \rho c(x) \right]^T \frac{\partial c}{\partial x} &= 0 \\ \Rightarrow \tilde{\lambda} &\leftarrow \tilde{\lambda} - \rho c(x) \end{aligned}$$

这说明， $\tilde{\lambda}$ 将会更新为 $\tilde{\lambda} - \rho c(x)$ ，对于那些活动约束，越来越多的约束将会吸入到lagrangian multipliers中。

在算法上，我们可以将增广拉格朗日方法表示为如下：

Algorithm 1: 增广拉格朗日方法

```
Input: 初始值  $x_0, \lambda_0, \rho_0$ 
Output: 最优解  $x$ , 拉格朗日乘子  $\lambda$ 
while 未收敛 do
    最小化  $L_\rho(x, \tilde{\lambda})$  以更新  $x$  ;
    更新拉格朗日乘子:  $\tilde{\lambda} \leftarrow \tilde{\lambda} - \max(0, \tilde{\lambda} - \rho c(x))$  ;
        /* 更新拉格朗日乘子, 确保非负性。 */
    增大惩罚参数:  $\rho \leftarrow \alpha\rho$  ;
        /* 增加惩罚参数, 一般  $\alpha \approx 10$ 。 */
end
```

该方法的优点在于：

- 修复了惩罚方法中可能出现的病态问题。
- 收敛速度较快（超线性），但精度适中。
- 在非凸问题中同样有效。

不等式约束的优化方法，牛顿增广拉格朗日法例

目标函数：

$$f(x) = \frac{1}{2} \left(x - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right)^T Q \left(x - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right)$$

其中， Q 是对角矩阵。

目标函数的梯度：

$$\nabla f(x) = Q \left(x - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right)$$

目标函数的 Hessian 矩阵：

$$\nabla^2 f(x) = Q$$

约束条件：

$$c(x) = Ax - b$$

其中， $A = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ 和 $b = -1$ 。

增广拉格朗日函数：

$$\mathcal{L}_\rho(x, \lambda) = f(x) + \lambda \cdot c(x) + \frac{\rho}{2} \cdot (c(x))^2$$

其中， λ 是拉格朗日乘子， ρ 是惩罚参数。

算法流程：牛顿法求解增广拉格朗日函数的最小值点

Algorithm 2: 牛顿法求解增广拉格朗日函数的最小值点

Input: 初始猜测 x_0 , 惩罚参数 ρ , 拉格朗日乘子 λ , 容忍度 ϵ
Output: 最优解 x

初始化 $x \leftarrow x_0$;
 $p \leftarrow \max(0, c(x))$;

while 满足收敛条件且迭代次数 < 最大迭代次数 **do**

计算梯度 $g = \nabla f(x) + (\lambda + \rho p) \cdot \nabla c(x)$;
计算Hessian矩阵 $H = \nabla^2 f(x) + \rho \cdot (\nabla c(x))^T \cdot \nabla c(x)$;
解线性方程组 $H\Delta x = -g$ 得到 Δx ;
更新 $x \leftarrow x + \Delta x$;
更新 $p \leftarrow \max(0, c(x))$;
更新拉格朗日乘子 $\lambda \leftarrow \lambda + \rho \cdot c(x)$;

return 最优解 x

代码如下

```
1 # 设置对角矩阵 Q
2 Q = np.diag([0.5, 1.0])
3
4 # 目标函数
5 def f(x):
6     """计算目标函数值"""
7     diff = x - np.array([1.0, 0.0])
8     return 0.5 * diff.T @ Q @ diff
9
10 # 目标函数的梯度
11 def grad_f(x):
12     """计算目标函数梯度"""
13     return Q @ (x - np.array([1.0, 0.0]))
14
15 # 目标函数的Hessian矩阵
16 def hessian_f(x):
17     """计算目标函数的Hessian矩阵"""
18     return Q
19
20 # 约束条件
21 A = np.array([1.0, -1.0])
22 b = -1.0
23
24 def c(x):
25     """计算约束条件值"""
26     return np.dot(A, x) - b
27
28 def grad_c(x):
29     """计算约束条件梯度"""
30     return A
31
32 # 增广拉格朗日函数
33 def La(x, lam, rho):
34     """计算增广拉格朗日函数值"""
35     p = max(0, c(x))
36     return f(x) + lam * p + (rho / 2) * (p ** 2)
37
38 # 牛顿法求解
39 def newton_solve(x0, lam, rho, tol=1e-8):
40     """使用牛顿法求解增广拉格朗日函数的最小值点"""
41     x = x0.copy()
42     p = max(0, c(x))
43
44     # 初始化约束梯度矩阵
45     C = np.zeros((1, 2))
46     if c(x) >= 0:
```

```

47     C = grad_c(x).reshape(1, 2)
48
49     # 计算目标函数梯度
50     g = grad_f(x) + (lam + rho * p) * C.T.flatten()
51
52     # 牛顿法迭代
53     iter_count = 0
54     max_iter = 100
55     while np.linalg.norm(g) >= tol and iter_count < max_iter:
56         # 计算Hessian矩阵
57         H = hessian_f(x) + rho * C.T @ C
58
59         # 计算牛顿方向
60         delta_x = np.linalg.solve(H, -g)
61
62         # 更新x
63         x = x + delta_x
64
65         # 更新约束相关量
66         p = max(0, c(x))
67         C = np.zeros((1, 2))
68         if c(x) >= 0:
69             C = grad_c(x).reshape(1, 2)
70
71         # 更新梯度
72         g = grad_f(x) + (lam + rho * p) * C.T.flatten()
73         iter_count += 1
74
75     return x

```

4.4.3 二次规划示例 (Quadratic Program Example)

考虑以下问题：

$$\min_x \frac{1}{2} x^T Q x + q^T x \quad \text{s.t.} \quad Ax \leq b, \quad Cx = d$$

这是一个二次规划问题，包含二次项和线性约束。这类问题在控制领域中非常常见并且有广泛的应用，通常可以快速求解。

机器人学中的二次规划示例 考虑一个机器人末端执行器的运动规划问题，我们希望最小化其关节速度的平方和，同时满足末端执行器的速度约束。问题可以表述为以下二次规划问题：

$$\begin{aligned} & \min_{\dot{q}} \frac{1}{2} \dot{q}^T W \dot{q} \\ \text{s.t. } & J \dot{q} = v, \quad \dot{q}_{\min} \leq \dot{q} \leq \dot{q}_{\max} \end{aligned}$$

其中：

- \dot{q} 是关节速度向量。
- W 是权重矩阵，通常为对角矩阵，用于平衡不同关节的速度优先级。
- J 是机器人雅可比矩阵，描述了关节速度与末端执行器速度之间的关系。
- v 是末端执行器的期望速度。
- \dot{q}_{\min} 和 \dot{q}_{\max} 是关节速度的上下限。

求解方法 该问题可以通过二次规划求解器来解决。以下是一个简单的 Python 实现：

```

1 import numpy as np
2 from scipy.optimize import minimize
3
4 # 定义问题参数
5 W = np.diag([1.0, 1.0]) # 权重矩阵
6 J = np.array([[1.0, 0.5], [0.5, 1.0]]) # 雅可比矩阵
7 v = np.array([0.5, 0.5]) # 期望末端速度
8 q_min = np.array([-1.0, -1.0]) # 关节速度下限
9 q_max = np.array([1.0, 1.0]) # 关节速度上限
10
11 # 定义目标函数
12 def objective(q_dot):
13     return 0.5 * q_dot.T @ W @ q_dot
14
15 # 定义约束
16 def equality_constraint(q_dot):
17     return J @ q_dot - v
18
19 constraints = {'type': 'eq', 'fun': equality_constraint}
20 bounds = [(q_min[i], q_max[i]) for i in range(len(q_min))]
21
22 # 求解二次规划问题
23 result = minimize(
24     objective,
25     x0=np.zeros(len(q_min)),
26     bounds=bounds,
27     constraints=constraints)
28
29 # 输出结果
30 if result.success:
31     print("Optimal joint velocities:", result.x)
32 else:
33     print("Optimization failed.")

```

结果与分析 通过上述代码，我们可以得到满足末端执行器速度约束的最优关节速度，同时保证关节速度的平方和最小。这种方法在机器人运动规划中非常常用，尤其是在实时控制中。

4.5 正规化和对偶

考虑以下问题：

$$\min_x f(x) \quad \text{s.t.} \quad c(x) = 0$$

我们可以将其表示为：

$$\min_x f(x) + P_\infty(c(x))$$

其中， $P_\infty(x)$ 定义为：

$$P_\infty(x) = \begin{cases} 0, & x = 0 \quad (\text{满足约束}) \\ +\infty, & x \neq 0 \quad (\text{不满足约束}) \end{cases}$$

从实践角度看，这样的表达式非常糟糕，但我们可以求解以下问题得到相同的效果：

$$\min_x \max_{\lambda} f(x) + \lambda^T c(x)$$

当 $c(x) \neq 0$ 时，内问题的结果为 $+\infty$ 。对于不等式约束的类似表达：

$$\begin{aligned} \min_x f(x) \quad \text{s.t.} \quad c(x) \geq 0 \Rightarrow \\ \min_x f(x) + P_{\infty}^+(c(x)) \end{aligned}$$

此时， $P_{\infty}^+(x)$ 定义为：

$$P_{\infty}^+(x) = \begin{cases} 0, & x \geq 0 \\ +\infty, & x < 0 \end{cases}$$

我们可以将其表示为：

$$\min_x \max_{\lambda \geq 0} f(x) - \lambda^T c(x)$$

其中， $f(x) - \lambda c(x)$ 为拉格朗日函数 $L(x, \lambda)$ 。

对于凸问题，可以交换最小值和最大值的顺序，解不会改变；这就是对偶问题的概念！然而，在一般情况下（如非凸问题），这种交换不成立。

该表达式的解释是，KKT 条件定义了 (x, λ) 空间中的鞍点。

KKT 系统在最优点时应该有 $\dim(x)$ 个正特征值和 $\dim(\lambda)$ 个负特征值，最优解下的系统称为“准定的”线性系统。quasi-definite 是指，一个具有已知正负特征值的鞍点系统。

总结： 当正规化 KKT 系统时，左下方的块应该是负的！

$$\begin{bmatrix} H + \beta I & C^T \\ C & -\beta I \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} -\nabla_x L \\ -c(x) \end{bmatrix} \quad \text{with } \beta > 0$$

这里左上角要正的，右下角要负的。

这使得系统成为准定的系统。

约束不等式牛顿法及其正则化 在处理约束优化问题时，牛顿法和正则化牛顿法可以有效地求解问题。牛顿法通过计算目标函数的Hessian矩阵和约束的雅可比矩阵，使用KKT条件构造线性系统并进行求解。正则化牛顿法通过引入正则化参数来确保解满足鞍点条件，从而避免计算中的奇异问题。

牛顿法 牛顿法的核心思想是通过最小化拉格朗日函数，计算目标函数和约束条件的梯度，利用以下的 KKT 条件来更新参数 x 和拉格朗日乘子 λ 。

首先，构造KKT矩阵 K ，包含目标函数的Hessian矩阵 H 和约束函数的雅可比矩阵 C ：

$$K = \begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix}$$

然后解以下线性系统，得到增量 Δx 和 $\Delta \lambda$ ：

$$K \cdot \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} -\nabla f(x) - C^T \lambda \\ -c(x) \end{bmatrix}$$

牛顿法的迭代过程为：

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \Delta x \\ \lambda^{(k+1)} &= \lambda^{(k)} + \Delta \lambda \end{aligned}$$

正则化牛顿法 正则化牛顿法通过引入正则化矩阵来保证系统的稳定性。通过修改KKT矩阵并引入正则化参数 β , 正则化后的KKT矩阵为:

$$K_{\text{reg}} = K + \beta \cdot \text{diag}(\mathbf{I}, -\mathbf{I})$$

其中, $\beta > 0$ 是正则化系数, 确保矩阵的特征值满足准定条件。最后, 求解以下线性系统得到更新增量:

$$K_{\text{reg}} \cdot \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} -\nabla f(x) - C^T \lambda \\ -c(x) \end{bmatrix}$$

正则化牛顿法的迭代过程为:

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \Delta x \\ \lambda^{(k+1)} &= \lambda^{(k)} + \Delta \lambda \end{aligned}$$

Python 代码展示 以下是实现牛顿法和正则化牛顿法的关键 Python 代码:

```

1 # [optimization_inequality_constraints_regularized.py]
2 # 牛顿步骤函数
3 def newton_step(x, lam):
4     H = hess_f(x) + lam * np.array([[2, 0], [0, 0]])
5     C = jacobian_c(x)
6
7     # 构建KKT矩阵
8     K = np.block([
9         [H, C.T],
10        [C, np.zeros((1, 1))],
11    ])
12
13     # 构建右侧向量
14     rhs = np.concatenate([-grad_f(x) - C.T @ lam, -np.array([c(x)])])
15
16     # 求解线性系统
17     delta_z = np.linalg.solve(K, rhs)
18
19     delta_x = delta_z[:2]
20     delta_lam = delta_z[2:]
21
22     return delta_x, delta_lam
23
24 # 正则化牛顿步骤函数
25 def regularized_newton_step(x, lam):
26     beta = 1.0
27     H = hess_f(x) + lam * np.array([[2, 0], [0, 0]])
28     C = jacobian_c(x)
29
30     # 构建KKT矩阵
31     K = np.block([
32         [H, C.T],
33         [C, np.zeros((1, 1))],
34     ])

```

```

35     # 计算特征值
36     e = np.linalg.eigvals(K)
37
38     # 正则化矩阵直到满足鞍点条件
39     while not (np.sum(e > 0) == len(x) and np.sum(e < 0) == len(lam)):
40         reg_matrix = np.zeros_like(K)
41         reg_matrix[:2, :2] = beta * np.eye(2)
42         reg_matrix[2:, 2:] = -beta * np.eye(1)
43         K = K + reg_matrix
44         e = np.linalg.eigvals(K)
45
46     # 求解线性系统
47     rhs = np.concatenate([-grad_f(x) - C.T @ lam, -np.array([c(x)])])
48     delta_z = np.linalg.solve(K, rhs)
49
50     delta_x = delta_z[:2]
51     delta_lam = delta_z[2:]
52
53     return delta_x, delta_lam
54
55
56 # 迭代方法
57 def iterate_newton_method(x_init, lam_init, max_iter=10, tol=1e-6):
58     x = x_init
59     lam = lam_init
60     for k in range(max_iter):
61         delta_x, delta_lam = newton_step(x, lam)
62         x += delta_x
63         lam += delta_lam
64
65         if np.linalg.norm(delta_x) < tol:
66             break
67     return x, lam
68
69 def iterate_regularized_newton_method(x_init, lam_init, max_iter=10, tol=1e-6):
70     x = x_init
71     lam = lam_init
72     for k in range(max_iter):
73         delta_x, delta_lam = regularized_newton_step(x, lam)
74         x += delta_x
75         lam += delta_lam
76
77         if np.linalg.norm(delta_x) < tol:
78             break
79     return x, lam

```

4.6 Merit Functions (for line search)解决超调问题

如何进行一个根寻找问题的线搜索？考虑如下问题：

$$\text{find } x^* \text{ s.t. } c(x^*) = 0$$

首先，我们定义一个标量“Merit 函数” $P(x)$ ，它度量距离解的距离。常用的 Merit 函数包括：

$$P(x) = \frac{1}{2} c(x)^T c(x) = \frac{1}{2} \|c(x)\|_2^2$$

或 $P(x) = \|c(x)\|_1$

注意：我们可以使用任何范数。现在我们可以对 $P(x)$ 使用阿米约规则（Armijo Rule）：

Armijo Rule on $P(x)$:

Algorithm 3: Armijo Rule

Input: 初始步长 $\alpha = 1$
Output: 优化的步长 α
while $p(x + \alpha\Delta x) > p(x) + b\nabla p(x)^T \Delta x$ **do**
 $\alpha \leftarrow \alpha/2$;
end
 $x \leftarrow x + \alpha\Delta x$

Step Length: $\alpha\nabla p(x)^T \Delta x$ 是期望的减少量，用于松弛防止来回反弹。
 β 是一个小的标量，通常 $0 < \beta < 1$ 。

4.6.1 约束最小化的线搜索

对于约束最小化问题，我们希望提出一个方案，指定我们违反约束的程度，以及我们距离最优解的远近。考虑问题：

$$\min_x f(x)$$

$$\begin{cases} c(x) \leq 0 \\ d(x) = 0 \end{cases}$$

我们可以定义拉格朗日函数为：

$$L(x, \lambda, \mu) = f(x) + \lambda^T c(x) + \mu^T d(x)$$

此时，我们有多种 Merit 函数的选项。其中一个是：

$$P(x, \lambda, \mu) = \frac{1}{2} \|r_{KKT} L(x, \lambda, \mu)\|_2^2$$

这里， $r_{KKT}(x, \lambda, \mu)$ 是 KKT 残差：

$$r_{KKT}(x, \lambda, \mu) = \begin{bmatrix} \nabla_x L(x, \lambda, \mu) \\ \min(0, c(x)) \\ d(x) \end{bmatrix}$$

然而，这不是最佳选择，因为评估 KKT 条件的梯度和牛顿步长的求解一样昂贵。
另一个选择是：

$$P(x, \lambda, \mu) = f(x) + \rho \left\| \begin{bmatrix} \min(0, c(x)) \\ d(x) \end{bmatrix} \right\|_1$$

这里， ρ 是目标函数最小化和约束满足之间的标量折衷。请记住，任何范数都可以用于此（图中使用的是 1 范数），但使用 1 范数是最常见的。这种方法提供了灵活性，因为我们可以在接近最优解时调整 ρ 。

又一个选择是：

$$P(x, \lambda, \mu) = f(x) - \lambda^T c(x) + \mu^T d(x) + \frac{\rho}{2} \|\min(0, c(x))\|_2^2$$

这实际上就是增广拉格朗日方法本身。可以看到 merrit-function.ipynb 中，未添加 KKT residual 的情况下，牛顿法将有非常大的过充问题。增加了 KKT residual 后将采取更保

守的步长。我们可以看到，牛顿法在没有 KKT 残差的情况下会有非常大的过冲问题。添加 KKT 残差后，牛顿法将采取更保守的步长。

```
1 ˜ function gauss_newton_step(x, λ)
2      H = ∇²f(x)                      # 目标函数 Hessian
3      C = ∂c(x)                      # 约束雅可比 (行向量)
4      # 解 KKT 线性系统 [H  C'; C  0]·[Δx; Δλ] = [-∇f - C'λ; -c]
5      Δz = [H  C'; C  0] \ (-∇f(x) - C' * λ; -c(x))
6      Δx = Δz[1:2]; Δλ = Δz[3]
7      return Δx, Δλ
8  end
9
10 # Merit 函数 (增广拉格朗日形式)
11 ρ = 1.0
12 ˜ function P(x, λ)
13     f(x) + λ' * c(x) + 0.5 * ρ * dot(c(x), c(x))
14 end
15
16 # Merit 梯度
17 ˜ function ∇P(x, λ)
18     g = ∇f(x) + ∂c(x)' * (λ .+ ρ * c(x))
19     return [g; c(x)]
20 end
21
22 # 带 Armijo 线搜索的一步更新
23 ˜ function update!(x, λ)
24     Δx, Δλ = gauss_newton_step(x, λ)
25     α = 1.0
26     # Armijo 判据
27     ˜ while P(x + α * Δx, λ + α * Δλ) > P(x, λ) + 0.01 * α * dot(∇P(x, λ), [Δx; Δλ])
28         α *= 0.5
29     end
30     x_new = x + α * Δx
31     λ_new = λ + α * Δλ
32     return x_new, λ_new
33 end
```

图 15: 更新后的不等式约束Merit函数可视化

结论

- 基于 KKT 残差的 $P(x)$ 计算代价较高。
- 过大的罚项权重可能导致问题。
- 增广拉格朗日方法可以与 Merit 函数一起使用。因此，如果我们使用增广拉格朗日来解决问题，只需将其作为 Merit 函数即可。

5 最优控制

5.1 最优控制的历史回顾

TODO: 完成这一部分

5.2 变分法 Calculus of Variations

这里我们将首先介绍变分法，因为最优控制的很多方法都可以看作是变分法的推广，例如动态规划（dynamic programming）和最优控制（optimal control）等。

我们希望寻找一个函数 $y(x)$ ，使得泛函

$$J[y] = \int_a^b L(x, y(x), y'(x)) dx$$

取得极值，其中 $L(x, y, y')$ 为已知函数，称为拉格朗日量（Lagrangian）， $y(x)$ 是待求函数，满足边界条件 $y(a) = y_a$, $y(b) = y_b$ 。

为此，引入扰动函数

$$y_\varepsilon(x) = y(x) + \varepsilon\eta(x),$$

其中 ε 为无穷小参数， $\eta(x)$ 为任意光滑函数，且满足端点条件 $\eta(a) = \eta(b) = 0$ 。

考虑变分：

$$\delta J = \frac{d}{d\varepsilon} J[y_\varepsilon] \Big|_{\varepsilon=0} = \frac{d}{d\varepsilon} \int_a^b L(x, y + \varepsilon\eta, y' + \varepsilon\eta') dx \Big|_{\varepsilon=0} = \int_a^b \left(\frac{\partial L}{\partial y} \eta + \frac{\partial L}{\partial y'} \eta' \right) dx.$$

对第二项积分分部，得：

$$\int_a^b \frac{\partial L}{\partial y'} \eta' dx = \left[\frac{\partial L}{\partial y'} \eta \right]_a^b - \int_a^b \frac{d}{dx} \left(\frac{\partial L}{\partial y'} \right) \eta dx.$$

因为 $\eta(a) = \eta(b) = 0$ ，边界项为零，故变分为：

$$\delta J = \int_a^b \left(\frac{\partial L}{\partial y} - \frac{d}{dx} \left(\frac{\partial L}{\partial y'} \right) \right) \eta(x) dx.$$

由于 $\eta(x)$ 是任意满足端点条件的光滑函数，根据变分法基本引理，必有：

$$\frac{\partial L}{\partial y} - \frac{d}{dx} \left(\frac{\partial L}{\partial y'} \right) = 0.$$

这就是著名的 Euler-Lagrange 方程。

例1：最短路径问题 考虑使下式最小的曲线：

$$J[y] = \int_a^b \sqrt{1 + y'^2} dx.$$

此时 $L = \sqrt{1 + y'^2}$ ，则

$$\frac{\partial L}{\partial y} = 0, \quad \frac{\partial L}{\partial y'} = \frac{y'}{\sqrt{1 + y'^2}},$$

代入 Euler-Lagrange 方程:

$$\frac{d}{dx} \left(\frac{y'}{\sqrt{1+y'^2}} \right) = 0 \Rightarrow \frac{y'}{\sqrt{1+y'^2}} = \text{常数} \Rightarrow y' = \text{常数},$$

因此 $y(x)$ 是直线: $y(x) = mx + c$ 。

例2: 最速降线问题 (Brachistochrone Problem) 设一个质点从原点 $(0, 0)$ 滑落至点 (x_1, y_1) , 在重力作用下沿某曲线运动。我们希望寻找使滑动时间最短的曲线 $y(x)$, 其中 $y(x) > 0$ 表示向下。

根据能量守恒, 质点在位置 y 处的速度为:

$$v = \sqrt{2gy}.$$

滑动时间为:

$$T[y] = \int_0^{x_1} \frac{ds}{v} = \int_0^{x_1} \frac{\sqrt{1+y'^2}}{\sqrt{2gy}} dx.$$

因此泛函为:

$$J[y] = \int_0^{x_1} L(y, y') dx, \quad \text{其中 } L(y, y') = \frac{\sqrt{1+y'^2}}{\sqrt{2gy}}.$$

接下来我们使用 Euler-Lagrange 方程进行推导:

$$\frac{d}{dx} \left(\frac{\partial L}{\partial y'} \right) - \frac{\partial L}{\partial y} = 0.$$

首先计算各个偏导数:

1. 对 y' 的偏导:

$$\frac{\partial L}{\partial y'} = \frac{1}{\sqrt{2gy}} \cdot \frac{y'}{\sqrt{1+y'^2}}.$$

2. 对 y 的偏导:

$$\frac{\partial L}{\partial y} = -\frac{1}{2} \cdot \frac{\sqrt{1+y'^2}}{\sqrt{2g} \cdot y^{3/2}}.$$

然后我们来求导数项:

$$\frac{d}{dx} \left(\frac{\partial L}{\partial y'} \right) = \frac{d}{dx} \left(\frac{y'}{\sqrt{2gy} \cdot \sqrt{1+y'^2}} \right).$$

将其视为乘积微分:

$$\frac{d}{dx} \left(\frac{y'}{\sqrt{y} \sqrt{1+y'^2}} \right) = \frac{1}{\sqrt{1+y'^2}} \cdot \frac{d}{dx} \left(\frac{y'}{\sqrt{y}} \right) + y' \cdot \frac{d}{dx} \left(\frac{1}{\sqrt{1+y'^2}} \right) \cdot \frac{1}{\sqrt{y}}.$$

计算这项非常繁琐, 因此我们转而采用简便的方法——**Beltrami 恒等式**, 它适用于 L 不显含 x 的情况。

Beltrami 恒等式:

若 L 不显含 x , 则:

$$L - y' \frac{\partial L}{\partial y'} = \text{常数.}$$

代入我们的问题:

先计算:

$$\frac{\partial L}{\partial y'} = \frac{y'}{\sqrt{2gy} \cdot \sqrt{1+y'^2}},$$

所以

$$L - y' \frac{\partial L}{\partial y'} = \frac{\sqrt{1+y'^2}}{\sqrt{2gy}} - y' \cdot \frac{y'}{\sqrt{2gy} \cdot \sqrt{1+y'^2}}.$$

合并为一个式子:

$$L - y' \frac{\partial L}{\partial y'} = \frac{1}{\sqrt{2gy}} \left(\sqrt{1+y'^2} - \frac{y'^2}{\sqrt{1+y'^2}} \right) = \frac{1}{\sqrt{2gy}} \cdot \frac{1+y'^2-y'^2}{\sqrt{1+y'^2}} = \frac{1}{\sqrt{2gy} \cdot \sqrt{1+y'^2}}.$$

因此:

$$\frac{1}{\sqrt{2gy(1+y'^2)}} = C.$$

两边取倒数并平方:

$$2gy(1+y'^2) = \frac{1}{C^2}.$$

整理得:

$$1+y'^2 = \frac{1}{2gC^2y} \Rightarrow y'^2 = \frac{1}{2gC^2y} - 1.$$

对这个微分方程进行变换求解。令:

$$\frac{1}{2gC^2} = A \Rightarrow y'^2 = \frac{A}{y} - 1.$$

——
参数化解法: 令

$$y = \frac{A}{2}(1 - \cos \theta), \quad x = \frac{A}{2}(\theta - \sin \theta),$$

则此摆线满足上述微分关系, 即为最速降线解。

5.3 确定性最优控制 Deterministic Optimal Control

这一节将介绍确定性最优控制, Pontryagin's Maximum Principle (庞特里亚金最大值原理) 和线性二次调节器 (Linear Quadratic Regulator, LQR) 问题。

首先我们考虑这样一个控制问题:

$$\begin{aligned} \min_{x(t), u(t)} \quad & J(x(t), u(t)) = \int_{t_0}^{t_f} L(x(t), u(t)) dt + L_F(x(t_f)) \\ \text{s.t.} \quad & \dot{x}(t) = f(x(t), u(t)) \\ & \text{and 其他约束条件...} \end{aligned} \tag{36}$$

这里我们通过最小化cost functional J (是一个泛函)来求解控制问题。 $x(t)$ 是状态变量, $u(t)$ 是控制变量, $L(x(t), u(t))$ 是状态成本 (stage cost), $L_F(x(t_f))$ 是终端成本 (terminal cost), $f(x(t), u(t))$ 是系统动力学方程 (动力学的限制)。这是一个“无限维”问题, 即需要无限多个离散时间的控制点来完全描述要施加的控制, 也就是说

$$u(t)$$

是一个函数而不是一个数值。但对于计算机控制来说, 我们只能在有限个离散时间点上施加控制, 因此我们需要将这个问题离散化。我们可以通过控制时间间隔来无限逼近这个无限时间问题。

$$u(t) = \lim_{N \rightarrow \infty} u_n(t) = \lim_{N \rightarrow \infty} u_n(t_0 + n\Delta t), \quad n = 0, 1, \dots, N \quad (37)$$

- 该问题的解是开环 (open-loop) 轨迹。
- 目前只有极少数控制问题在连续时间下有解析解。
- 我们重点关注离散时间下的情形, 在这种情况下可以采用可行的算法。

5.3.1 离散时间 Discrete Time

考虑该问题的离散时间版本:

$$\begin{aligned} \min_{x_{1:N}, u_{1:N-1}} \quad & J(x_{1:N}, u_{1:N-1}) = \sum_{k=1}^{N-1} L(x_k, u_k) + L_F(x_N) \\ \text{s.t.} \quad & x_{n+1} = f(x_n, u_n) \\ & u_{\min} \leq u_k \leq u_{\max} \quad \text{力矩约束 (Torque limits)} \\ & c(x_k) \leq 0 \quad \forall k \quad \text{障碍/碰撞约束 (Obstacle / collision constraints)} \end{aligned}$$

- 这个问题的离散时间版本现在是一个有限维问题。
- 采样点 x_k, u_k 常被称为“节点” (knot points)。
- 我们可以使用如 Runge-Kutta 方法等积分方法, 将连续系统转换为离散时间问题。(见2.2.2节)
- 最后, 也可以通过插值法将离散时间问题转回连续时间问题。

5.3.2 Pontryagin 最小值原理

- Pontryagin 最小值原理 (Pontryagin's Minimum Principle), 如果我们最大化奖励函数则又称为“最大值原理”。
- 本质上为确定性最优控制问题提供了一阶必要条件。
- 在离散时间下, 它其实是 KKT 条件的一个特例。

考虑我们之前的问题：

$$\begin{aligned} \min_{x_{1:N}, u_{1:N-1}} \quad & J(x_{1:N}, u_{1:N-1}) = \sum_{k=1}^{N-1} L(x_k, u_k) + L_F(x_N) \\ \text{s.t.} \quad & x_{n+1} = f(x_n, u_n) \end{aligned}$$

在本设置中，我们主要考虑控制约束（如力矩限制），但难以处理状态约束（如碰撞约束）。

我们可以写出该问题的 Lagrangian 如下：

$$L = \sum_{k=1}^{N-1} [l(x_k, u_k) + \lambda_{k+1}^T (f(x_k, u_k) - x_{k+1})] + L_F(x_N)$$

这通常以“哈密顿量”形式表达：^{2 3}

$$H(x, u, \lambda) = l(x, u) + \lambda^T f(x, u)$$

将 H 带入 L 中，得

$$L = H(x_1, u_1, \lambda_2) + \left[\sum_{k=2}^{N-1} H(x_k, u_k, \lambda_{k+1}) - \lambda_k^T x_k \right] + L_F(x_N) - \lambda_N^T x_N$$

对 x 和 λ 求偏导数（必要条件）：

$$\begin{aligned} \frac{\partial L}{\partial x_k} &= \frac{\partial H}{\partial x_k} + \frac{\partial f}{\partial x_k}^T \lambda_{k+1} - \lambda_k = 0 \\ \frac{\partial L}{\partial \lambda_k} &= f(x_k, u_k) - x_{k+1} = 0 \\ \frac{\partial L}{\partial x_N} &= \frac{\partial L_F}{\partial x_N} - \lambda_N = 0 \end{aligned}$$

对控制量 u ，我们显式写出极小化过程（考虑控制约束）：

$$u_k = \arg \min_{\tilde{u}} H(x_k, \tilde{u}, \lambda_{k+1}), \quad \text{s.t. } \tilde{u} \in \mathcal{U}$$

其中 \mathcal{U} 是可行控制集合，比如 $u_{\min} \leq \tilde{u} \leq u_{\max}$ 。

²Hamiltonian 是一个物理学术语，通常用于描述系统的总能量。在最优控制中指的是“当前成本+未来影响的总和”，有点像 RL 中的“价值函数”的意思。公式中的

- $l(x, u)$ 表示当前时刻的损失，如 u^2 能量耗散。
- $f(x, u)$ 是系统方程描述 x 如何变化，
- λ 是协态变量（co-state variable），表示对未来影响的权重。
- $\lambda^T f(x, u)$ 是“当前状态变化对未来目标的影响”——像是提前把未来的损失加进来。

³协态变量（co-state variable），也叫伴随变量（adjoint variable）在最优化问题里，有约束时我们会引入拉格朗日乘子，衡量目标函数对约束的“敏感度”。协态变量其实就是动态系统里的“拉格朗日乘子”，用来处理动力学约束 $\dot{x} = f(x, u)$ 。直观来说，协态变量告诉你“如果当前状态发生微小变化，对最终目标的影响有多大”。

总结如下：

$$\begin{cases} x_{k+1} = \nabla_\lambda H(x_k, u_k, \lambda_{k+1}) = f(x_k, u_k) \\ \lambda_k = \nabla_x H(x_k, u_k, \lambda_{k+1}) = \nabla_x l(x_k, u_k) + \left(\frac{\partial f}{\partial x}\right)^T \lambda_{k+1} \\ u_k = \arg \min_{\tilde{u}} H(x_k, \tilde{u}, \lambda_{k+1}), \quad \text{s.t. } \tilde{u} \in \mathcal{U} \\ \lambda_N = \frac{\partial L_F}{\partial x_N} \end{cases}$$

连续时间形式：

$$\begin{aligned} \dot{x} &= \nabla_\lambda H(x, u, \lambda) = f_{continuous}(x, u) \\ \dot{\lambda} &= \nabla_x H(x, u, \lambda) = \nabla_x l(x, u) + \left(\frac{\partial f_{continuous}}{\partial x}\right)^T \lambda \\ u &= \arg \min_{\tilde{u}} H(x, \tilde{u}, \lambda), \quad \text{s.t. } \tilde{u} \in \mathcal{U} \\ \lambda_N &= \frac{\partial L_F}{\partial x_N} \end{aligned}$$

Pontryagin's 实际例子 下面用一个经典最优控制问题，详细说明 Pontryagin 最小值原理的求解过程。

1. 问题表述（离散形式）

控制一维质量点的运动，在 N 个时间步内从已知初态 (x_1, v_1) 移动到已知终态 (x_N, v_N) ，使控制能量最小。

$$\begin{aligned} \min_{x_{1:N}, v_{1:N}, u_{1:N-1}} \quad & J = \sum_{k=1}^{N-1} u_k^2 \\ \text{s.t. } & \begin{cases} x_{k+1} = x_k + hv_k \\ v_{k+1} = v_k + hu_k \end{cases} \end{aligned}$$

其中 h 是步长， u_k 是控制输入（加速度）， x_k 是位置， v_k 是速度。

2. 写出Lagrangian

为了引入动力学约束，引入协态变量 $\lambda_{k+1}^x, \lambda_{k+1}^v$ ，Lagrangian为：

$$L = \sum_{k=1}^{N-1} [u_k^2 + \lambda_{k+1}^x(x_k + hv_k - x_{k+1}) + \lambda_{k+1}^v(v_k + hu_k - v_{k+1})] + 0$$

（这里认为final cost=0）

3. 写出Hamiltonian

$$H(x_k, v_k, u_k, \lambda_{k+1}^x, \lambda_{k+1}^v) = u_k^2 + \lambda_{k+1}^x(v_k) + \lambda_{k+1}^v(u_k)$$

其中， $l(x_k, u_k) = u_k^2$ ， $f(x_k, v_k, u_k) = (v_k, u_k)$ 。

4. 将Hamiltonian带入Lagrangian

结合模板公式，Lagrangian可以表达为：

$$L = H(x_1, v_1, u_1, \lambda_2^x, \lambda_2^v) + \sum_{k=2}^{N-1} [H(x_k, v_k, u_k, \lambda_{k+1}^x, \lambda_{k+1}^v) - (\lambda_k^x x_k + \lambda_k^v v_k)] - (\lambda_N^x x_N + \lambda_N^v v_N)$$

5. 对 x_k, v_k, λ_k 求偏导（必要条件）

- 对 x_k 求导：

$$\frac{\partial L}{\partial x_k} = \lambda_{k+1}^x - \lambda_k^x = 0 \implies \lambda_{k+1}^x = \lambda_k^x$$

- 对 v_k 求导：

$$\frac{\partial L}{\partial v_k} = h\lambda_{k+1}^x + \lambda_{k+1}^v - \lambda_k^v = 0 \implies \lambda_k^v = \lambda_{k+1}^v + h\lambda_{k+1}^x$$

- 对 $\lambda_{k+1}^x, \lambda_{k+1}^v$ 求导得到动力学约束：

$$x_{k+1} = x_k + hv_k$$

$$v_{k+1} = v_k + hu_k$$

6. 对控制 u_k 求极小（Pontryagin条件）

$$u_k = \arg \min_u H(x_k, v_k, u, \lambda_{k+1}^x, \lambda_{k+1}^v)$$

具体为：

$$\frac{\partial H}{\partial u_k} = 2u_k + \lambda_{k+1}^v = 0 \implies u_k^* = -\frac{1}{2}\lambda_{k+1}^v$$

7. 配合边界条件解联立方程

$$x_1 = x_0, \quad v_1 = v_0, \quad x_N = x_f, \quad v_N = v_f$$

λ_N^x, λ_N^v 可由终点条件确定

联立以上递推关系与边界条件，即可数值求解 $x_k, v_k, \lambda_k^x, \lambda_k^v, u_k$ 的最优轨迹。

7. Python 数值解法（简要代码说明）

1. 定义微分方程，将 λ_1 和 λ_2 作为扩展变量，一起求解。

2. 将 u 用 λ_2 表达，代入方程。

3. 用 `scipy.integrate.solve_bvp` 设置微分方程和边界条件，数值求解。

4. 画出 $x(t), v(t), u(t)$ 的最优轨迹。

8. 代码完整见[OptimalControl/PMP.py] 本代码实现的离散最优控制问题数学描述如下：

1. 优化目标:

$$\min_{u_0, \dots, u_{N-2}} J = \sum_{k=0}^{N-2} u_k^2$$

2. 系统动力学 (离散):

$$\begin{cases} x_{k+1} = x_k + hv_k \\ v_{k+1} = v_k + hu_k \end{cases} \quad \text{for } k = 0, 1, \dots, N-2$$

3. 边界条件:

$$x_0 = x_{\text{init}}, \quad v_0 = v_{\text{init}}, \quad x_{N-1} = x_{\text{final}}, \quad v_{N-1} = v_{\text{final}}$$

4. 协态变量递推 (伴随方程, 反向递推):

$$\lambda_k^x = \lambda_{k+1}^x$$

$$\lambda_k^v = \lambda_{k+1}^v + h\lambda_{k+1}^x$$

(终点条件: $\lambda_{N-1}^x, \lambda_{N-1}^v$ 用射击法确定)

5. 最优性条件 (Pontryagin极小):

$$u_k^* = \arg \min_u (u^2 + \lambda_{k+1}^v u) \implies u_k^* = -\frac{1}{2}\lambda_{k+1}^v$$

6. 整体递推流程 (数值实现):

已知 x_0, v_0 , 猜测 $\lambda_{N-1}^x, \lambda_{N-1}^v$

$$\text{for } k = N-2 \rightarrow 0 : \quad \begin{cases} \lambda_k^x = \lambda_{k+1}^x \\ \lambda_k^v = \lambda_{k+1}^v + h\lambda_{k+1}^x \end{cases}$$

$$\text{for } k = 0 \rightarrow N-2 : \quad \begin{cases} u_k = -\frac{1}{2}\lambda_{k+1}^v \\ x_{k+1} = x_k + hv_k \\ v_{k+1} = v_k + hu_k \end{cases}$$

调整 $\lambda_{N-1}^x, \lambda_{N-1}^v$ 使得 $x_{N-1} = x_{\text{final}}, v_{N-1} = v_{\text{final}}$

8. shooting法理解 通俗理解: 射击法就像打靶: 你站在起点 (已知初始状态), 目标是命中远处的靶心 (终点约束), 但你不知道子弹出膛时该瞄准哪个方向 (未知的初始/终点“协态”参数)。所以你“先猜一下方向”, 打出去, 看结果离靶心有多远。然后不断调整你的“瞄准角度”, 直到正中靶心。

数学理解: 在两点边值问题里, 只有初值 (如 x_0, v_0) 是已知的, 协态变量的终点值 (如 λ_N^x, λ_N^v) 是未知的。我们将协态变量的终点值当作“待定参数”来猜测。用这些猜测值, 正向/反向递推系统状态和协态, 看看末端 x_N, v_N 距离目标还有多远 (残差)。用数值优化器 (比如 root) 不断调整协态的终点猜测, 使残差收敛为0, 就得到了完整的最优轨迹。

7. 9. 数值结果

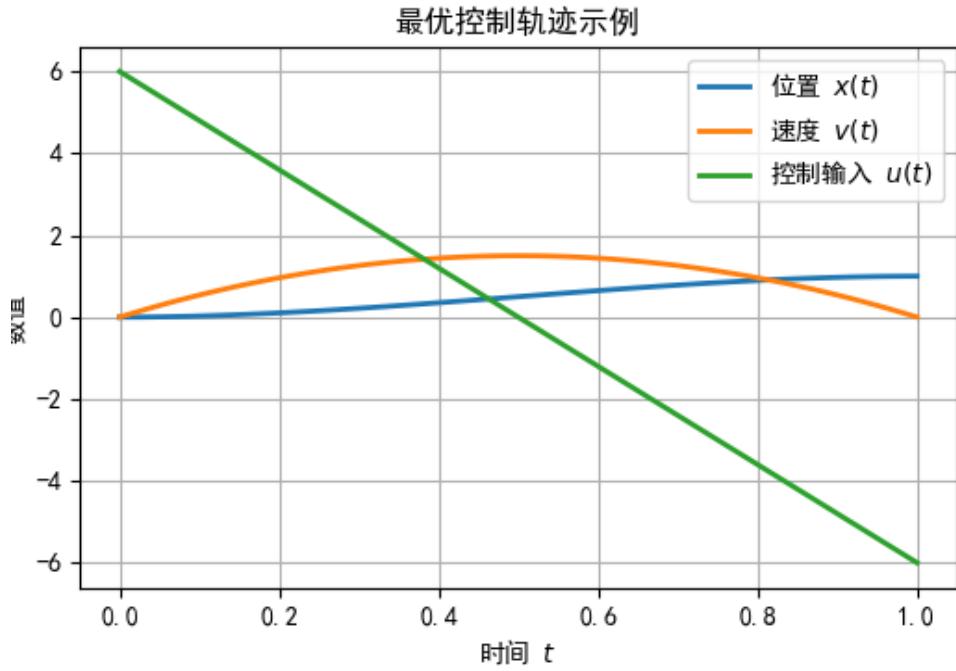


图 16: 离散最优控制问题的数值解与控制轨迹示例。图中显示了位置 $x(t)$ 、速度 $v(t)$ 和控制输入 $u(t)$ 随时间的变化。

由图 16 可见:

- 位置 $x(t)$ 从初始点 0 平滑地到达终点 1，轨迹为一条凸二次曲线；
- 速度 $v(t)$ 在中途达到最大值，末端回到 0，实现了“平滑启动和平滑停止”；
- 控制输入 $u(t)$ 随时间线性减小，为一条斜率为负的直线，这符合 $u(t) = at + b$ 的结构，是能量最优问题的典型特征。

这种结果显示了在能量最小的约束下，最优控制策略是**“温和施加加速度”**，而不是突变或极值切换。控制量 $u(t)$ 的线性变化，保证了既满足动态约束，又能使能量消耗最小。整体轨迹平滑，末端严格达到指定位置与速度。

5.4 线性二次调节器 Linear Quadratic Regulator (LQR)

LQR（线性二次调节器）是一类非常重要的最优控制方法，广泛应用于线性系统。LQR问题的目标是在满足线性动力学的约束下，使得系统的某一二次型性能指标最小化。数学表达如下：

$$\min_{x_{1:N}, u_{1:N-1}} \sum_{k=1}^{N-1} \left[\frac{1}{2} x_k^T Q x_k + \frac{1}{2} u_k^T R u_k \right] + \frac{1}{2} x_N^T Q_N x_N \quad (38)$$

$$\text{s.t. } x_{k+1} = A_k x_k + B_k u_k \quad (39)$$

其中：

- $Q \succeq 0$ 用于惩罚系统状态， $R \succ 0$ 用于惩罚控制输入。
- Q_N 用于终端状态惩罚。

- A_k, B_k 分别为系统的离散时间动力学矩阵，若对所有时刻均相同，则称为时不变LQR，否则为时变LQR。

LQR的一些性质：

- LQR的目标通常是将系统状态驱动至原点 ($x = 0$)。
- 当 A_k, B_k, Q_k, R_k 全部为常数时，称为“时不变LQR”；否则为“时变LQR”(TVLQR)。
- LQR常用于平衡点的稳定，TVLQR适用于轨迹跟踪问题。
- LQR理论可以（在局部）近似描述许多非线性问题，因此非常常用。
- LQR还有很多扩展形式，如无限时域LQR、随机LQR等。
- LQR被称为“控制理论皇冠上的明珠”。

5.4.1 间接射击法下的LQR（不推荐）

根据哈密顿量理论，LQR问题可用如下递推公式求解：

$$x_{k+1} = A_k x_k + B_k u_k \quad (31)$$

$$\lambda_k = Q x_k + A_k^T \lambda_{k+1}, \quad \lambda_N = Q_N x_N \quad (32)$$

$$u_k = -R^{-1} B_k^T \lambda_{k+1} \quad (33)$$

上述公式给出了：

- x_k 的正向递推（动力学方程）；
- λ_k 的反向递推（伴随方程）；
- u_k 的解析最优解（最优化条件）。

间接射击法求解LQR的基本流程：

1. 初始猜测 $u_{1:N-1}$ 及 x_0 ，用系统动力学正向递推出所有 $x_{1:N}$ 。
2. 反向递推伴随变量 λ_k ，即 $\lambda_k = Q x_k + A_k^T \lambda_{k+1}$ ，终点 $\lambda_N = Q_N x_N$ 。
3. 用 $\Delta u_k = (u_k - u_{\text{old}})$ 修正输入，并用 $u_{\text{new}} = u_{\text{old}} + \alpha \Delta u$ 做线搜索，更新控制律。
4. 重复步骤2-3，直至 Δu 收敛。

显然我们要证明反向地推伴随变量的公式及其终点递推式，和 Δu 的更新公式。下面我们可以加以证明：让我们从LQR问题的Hamiltonian入手来推导这些公式。

间接射击法求解LQR证明

推导1： 协态方程 $\lambda_k = Q x_k + A_k^T \lambda_{k+1}$ 与终点条件 $\lambda_N = Q_N x_N$

回顾LQR问题的Lagrangian:

$$L = \sum_{k=1}^{N-1} \left[\frac{1}{2} x_k^T Q x_k + \frac{1}{2} u_k^T R u_k + \lambda_{k+1}^T (A_k x_k + B_k u_k - x_{k+1}) \right] + \frac{1}{2} x_N^T Q_N x_N$$

对应的Hamiltonian为:

$$H(x_k, u_k, \lambda_{k+1}) = \frac{1}{2} x_k^T Q x_k + \frac{1}{2} u_k^T R u_k + \lambda_{k+1}^T (A_k x_k + B_k u_k)$$

通过最优化条件 (参考附页??), 对 x_k 求导并令其等于零:

$$\frac{\partial L}{\partial x_k} = 0$$

具体计算: 当 $1 \leq k \leq N-1$ 时, L 中只有两项与 x_k 有关, 则 L 关于 x_k 的偏导为:

$$\begin{aligned} \frac{\partial L}{\partial x_k} &= \frac{\partial}{\partial x_k} \left[\frac{1}{2} x_k^T Q x_k + \lambda_{k+1}^T A_k x_k - \lambda_k^T x_k \right] \\ &= Q x_k + A_k^T \lambda_{k+1} - \lambda_k = 0 \end{aligned}$$

a

解得:

$$\lambda_k = Q x_k + A_k^T \lambda_{k+1}$$

对于终点条件, 当 $k = N$ 时,

$$\begin{aligned} \frac{\partial L}{\partial x_N} &= \frac{\partial}{\partial x_N} \left[\frac{1}{2} x_N^T Q_N x_N - \lambda_N^T x_N \right] \\ &= Q_N x_N - \lambda_N = 0 \end{aligned}$$

解得:

$$\lambda_N = Q_N x_N$$

推导2: 控制更新量 $\Delta u_k = (u_k - u_{\text{old}})$

对Hamiltonian关于 u_k 求导并令其等于零:

$$\frac{\partial H}{\partial u_k} = R u_k + B_k^T \lambda_{k+1} = 0$$

解得最优控制:

$$u_k^* = -R^{-1} B_k^T \lambda_{k+1}$$

在间接射击法的迭代过程中, 我们需要更新控制输入。假设当前迭代的控制输入为 u_k^{old} , 根据最优化条件, 新的控制输入应为:

$$u_k^{\text{new}} = -R^{-1} B_k^T \lambda_{k+1}$$

因此, 控制更新量为:

$$\Delta u_k = u_k^{\text{new}} - u_k^{\text{old}} = -R^{-1} B_k^T \lambda_{k+1} - u_k^{\text{old}}$$

这表示在每次迭代中，控制更新量是基于协态变量 λ_{k+1} 计算的理论最优控制与当前控制值之间的差。

在实际的间接射击法算法中，我们通常会引入步长参数 α 来控制更新速度：

$$u_k^{\text{new}} = u_k^{\text{old}} + \alpha \cdot \Delta u_k$$

这样可以确保算法更稳定地收敛到最优解。

^a在矩阵微分中，我们使用了以下公式： $\frac{\partial}{\partial x}(x^T Q x) = (Q + Q^T)x$ ；当 Q 是对称矩阵时，这等于 $2Qx$ ，在LQR中 Q 通常是对称的，所以实际上是 Qx （因为 $\frac{1}{2}$ 的系数）。以及 $\frac{\partial}{\partial x}(a^T B x) = B^T a$ ，其中 a 和 x 是向量， B 是矩阵。

5.4.2 例子：双积分器系统

以双积分器（double integrator）为例，其动力学连续时间表达式为：

$$\dot{x} = \begin{bmatrix} q \\ \dot{q} \end{bmatrix}, \quad \ddot{x} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} q \\ \dot{q} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

将其离散化，得到：

$$x_{k+1} = \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} \frac{1}{2}h^2 \\ h \end{bmatrix} u_k$$

其中， $\begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix}$ 为系统矩阵 A ， $\begin{bmatrix} \frac{1}{2}h^2 \\ h \end{bmatrix}$ 为控制矩阵 B ， h 为离散时间步长。

Algorithm 4: 代价函数计算 (J)

Input: 状态轨迹 $x_{hist} \in \mathbb{R}^{n \times N}$, 控制序列 $u_{hist} \in \mathbb{R}^{N-1}$

Output: 总代价函数值 $cost$

算法描述: 该函数计算最优控制问题的总代价函数值。代价函数由终端状态代价和累积状态及控制代价两部分组成。首先计算终端状态代价，然后通过循环累加每个时间步的状态代价和控制代价。

```

cost ← 0.5 ·  $x_{hist}[:, N - 1]^T \cdot Q_n \cdot x_{hist}[:, N - 1]$  // 计算终端状态代价，使用终
端权重矩阵  $Q_n$ 
for  $k \leftarrow 0$  to  $N - 2$  do
    cost ← cost + 0.5 ·  $x_{hist}[:, k]^T \cdot Q \cdot x_{hist}[:, k] + 0.5 \cdot u_{hist}[k] \cdot R \cdot u_{hist}[k]$  // 累加
    状态代价和控制代价
end
return cost;

```

Algorithm 5: 系统前向模拟 (Rollout)

Input: 初始状态 $x_0 \in \mathbb{R}^n$, 控制序列 $u_{hist} \in \mathbb{R}^{N-1}$

Output: 状态轨迹 $x_{hist} \in \mathbb{R}^{n \times N}$

算法描述: 这里使用的是前向欧拉法, 我们可以换为RK4。该函数根据给定的初始状态和控制序列, 前向模拟系统的动态演化过程。系统状态方程为线性时不变系统 $x_{k+1} = A \cdot x_k + B \cdot u_k$, 其中 A 和 B 是系统和控制矩阵。函数返回包含所有时间步状态的完整状态轨迹。

初始化 $x_{hist} \in \mathbb{R}^{n \times N}$, $x_{hist}[:, 0] \leftarrow x_0$ // 设置初始状态

for $k \leftarrow 0$ **to** $N - 2$ **do**

$x_{hist}[:, k + 1] \leftarrow A \cdot x_{hist}[:, k] + B \cdot u_{hist}[k]$ // 根据系统动态方程更新下一个状态

end

return x_{hist} ;

Algorithm 6: 基于梯度下降的最优控制算法

Input: 初始状态 x_0 , 终止时间 T_{final} , 步长 h , 矩阵 A, B, Q, R, Q_n
Output: 最优控制序列 u_{hist} , 最优状态轨迹 x_{hist}
算法描述: 该算法通过梯度下降法求解线性二次型最优控制问题。首先初始化控制序列为零向量, 然后迭代更新控制输入以最小化代价函数。在每次迭代中, 算法从终端状态向初始状态反向计算协状态 (λ) 和控制梯度, 然后使用Armijo线搜索确定步长, 更新控制输入和系统状态。迭代过程持续到控制变化量小于阈值或达到最大迭代次数为止。

```
计算总时间步数  $N = \lfloor T_{final}/h \rfloor + 1$ ;
// 初始化变量
初始化  $x_{hist} \in \mathbb{R}^{n \times N}$ ,  $x_{hist}[:, 0] \leftarrow x_0$ ;
初始化  $u_{hist} \in \mathbb{R}^{N-1} \leftarrow 0$ ;
初始化  $\Delta u \in \mathbb{R}^{N-1} \leftarrow 1$ ;
初始化  $\lambda_{hist} \in \mathbb{R}^{n \times N} \leftarrow 0$ ;
 $x_{hist} \leftarrow \text{Rollout}(x_0, u_{hist})$  // 使用初始控制序列模拟系统
 $b \leftarrow 10^{-2}$  // Armijo线搜索参数
 $\alpha \leftarrow 1.0$  // 初始步长
iter_count  $\leftarrow 0$  // 迭代计数器
while  $\max |\Delta u| > 10^{-2}$  且  $iter\_count < 100$  do
     $\lambda_{hist}[:, N-1] \leftarrow Q_n \cdot x_{hist}[:, N-1]$  // 设置终端协状态
    for  $k = N-2$  downto 0 do
         $\Delta u[k] \leftarrow -(u_{hist}[k] + \frac{1}{R} B^T \cdot \lambda_{hist}[:, k+1])$  // 计算控制梯度方向
         $\lambda_{hist}[:, k] \leftarrow Q \cdot x_{hist}[:, k] + A^T \cdot \lambda_{hist}[:, k+1]$  // 向后更新协状态
    end
     $\alpha \leftarrow 1.0$  // 重置步长为初始值
     $u_{new} \leftarrow u_{hist} + \alpha \cdot \Delta u$ ;
     $x_{new} \leftarrow \text{Rollout}(x_0, u_{new})$  // 使用新控制序列模拟系统
    // Armijo线搜索条件
    while  $J(x_{new}, u_{new}) > J(x_{hist}, u_{hist}) - b \cdot \alpha \cdot \Delta u^T \Delta u$  do
         $\alpha \leftarrow 0.5 \cdot \alpha$  // 减小步长
         $u_{new} \leftarrow u_{hist} + \alpha \cdot \Delta u$ ;
         $x_{new} \leftarrow \text{Rollout}(x_0, u_{new})$  // 重新模拟系统
    end
     $u_{hist} \leftarrow u_{new}$  // 更新控制历史
     $x_{hist} \leftarrow x_{new}$  // 更新状态历史
     $iter\_count \leftarrow iter\_count + 1$  // 更新迭代计数
end
```

5.5 LQR 作为二次规划 (QP) 问题

假设初始状态 x_1 已知 (不是优化变量)。定义整个轨迹为决策变量 z :

$$z = \begin{bmatrix} u_1 \\ x_2 \\ u_2 \\ \vdots \\ x_N \end{bmatrix}$$

再定义 Hessian 矩阵 H , 与 z 的维度相匹配:

$$H = \begin{bmatrix} R_1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & Q_2 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & R_2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & Q_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & R_{N-1} & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & Q_N \end{bmatrix}$$

于是目标函数可写为 $J = \frac{1}{2}z^T Hz$ 。再定义约束矩阵 C 和向量 d :

$$C = \begin{bmatrix} B_1 & -I & 0 & \cdots & 0 & 0 \\ 0 & A_2 & B_2 & -I & \cdots & 0 \\ 0 & 0 & A_3 & B_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -I \end{bmatrix}$$

$$d = \begin{bmatrix} -A_1 x_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

这些矩阵强制执行系统动力学约束 $x_{k+1} = A_k x_k + B_k u_k$, 其中第一行处理初始条件 x_1 的特殊情况。矩阵 C 的结构确保了决策变量 z 必须满足系统动力学方程:

$$Cz = \begin{bmatrix} B_1 & -I & 0 & \cdots & 0 & 0 \\ 0 & A_2 & B_2 & -I & \cdots & 0 \\ 0 & 0 & A_3 & B_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -I \end{bmatrix} \begin{bmatrix} u_1 \\ x_2 \\ u_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} -A_1 x_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = d$$

4

⁴这里第一行 $-A_1 x_1$ 是一个常数向量, 表示初始状态的约束, 不参与优化。

于是，LQR 问题可以写成标准的二次规划问题 (QP)：

$$\min_z \quad \frac{1}{2} z^T H z \quad (40)$$

$$\text{s.t.} \quad Cz = d \quad (41)$$

这样其 Lagrangian 为

$$L(z, \lambda) = \frac{1}{2} z^T H z + \lambda^T [Cz - d]$$

KKT 条件为

$$\nabla_z L = Hz + C^T \lambda = 0 \quad (42)$$

$$\nabla_\lambda L = Cz - d = 0 \quad (43)$$

可整理为一个大的线性系统

$$\begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} z \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ d \end{bmatrix}$$

只需解这个线性方程组即可得到LQR的精确解！不需要迭代，因为这是线性系统！

5.5.1 例子

同样以双积分器 (sliding brick) 为例，可以直接比较 QP 解和间接射击法的效果，QP解往往收敛性更好、效率更高。

LQR QP 的代码实现

LQR QP问题描述： 本问题对应的物理背景为：给定一个一维质点（如小车）在无摩擦轨道上运动，系统状态为位置 x 和速度 v ，控制输入为加速度 u 。初始状态为 $x_0 = 1.0$, $v_0 = 0$ ，目标是在 $T_{\text{final}} = 100.0$ 秒内，通过合理设计每个时刻的加速度 u_k ，使得整个运动过程中状态和控制的加权二次型代价之和最小。

系统动力学采用离散时间双积分器模型，步长 $h = 0.1$ ，系统矩阵为

$$A = \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} \frac{1}{2}h^2 \\ h \end{bmatrix}$$

其中 A 表示在无控制输入时，位置 x 由速度 v 推动，速度 v 保持不变； B 表示控制输入 u 对状态的影响： u 通过二次积分影响位置 (x)，通过一次积分影响速度 (v)。该模型准确反映了加速度控制下的质点运动特性。

系统转移方程为：

$$x_{k+1} = Ax_k + Bu_k, \quad k = 0, 1, \dots, N-2$$

其中 $x_k = \begin{bmatrix} x_k \\ v_k \end{bmatrix}$ ， u_k 为标量控制输入。

状态维数 $n = 2$ ，控制维数 $m = 1$ ，总步数 $N = 1001$ 。

代价函数为

$$J = \frac{1}{2}x_N^\top Q_n x_N + \sum_{k=0}^{N-2} \left(\frac{1}{2}x_k^\top Q x_k + \frac{1}{2}u_k^\top R u_k \right)$$

其中 $Q = I_2$, $R = 0.1I_1$, $Q_n = I_2$ 。

物理意义为：在保证系统动力学约束 $x_{k+1} = Ax_k + Bu_k$ 的前提下，最小化位置、速度的偏离和控制能量的加权和，实现能量最优、平滑且高效的轨迹。通过将所有状态和控制作为优化变量，并将动力学写成等式约束，最终转化为带等式约束的二次规划（QP）问题。求解该QP即可得到最优控制序列和状态轨迹。

LQR QP核心数学公式：

$$\begin{aligned} \min_z \quad & \frac{1}{2}z^T H z \\ \text{s.t.} \quad & Cz = d \end{aligned}$$

其中 $z = [u_1, x_2, u_2, \dots, x_N]^T$, H 为块对角矩阵, C 强制系统动力学约束, d 包含初始条件。

Python代码实现： 如下

```

1  #[ OptimalControl/3_LQR_QuadraticProgramming.py ]
2  # 代价权重矩阵（使用稀疏矩阵）
3  Q = sparse.eye(2)                      # 状态代价矩阵
4  R = 0.1 * sparse.eye(1)                 # 控制代价矩阵
5  Qn = sparse.eye(2)                     # 最终状态代价矩阵
6
7  # 代价函数定义
8  def J(xhist, uhist):
9      """计算控制系统的总代价"""
10     cost = 0.5 * xhist[:, -1].T @ Qn @ xhist[:, -1]
11     for k in range(N-1):
12         state_cost = 0.5 * xhist[:, k].T @ Q @ xhist[:, k]
13         control_cost = 0.5 * uhist[k].T @ R @ uhist[k]
14         cost += state_cost + control_cost
15     return cost[0, 0]
16
17
18 # 构建H
19 H_blocks = [R]
20 for _ in range(N-2): # N=1001
21     H_blocks.append(Q) # H size = (2, 2)
22     H_blocks.append(R) # H size = (1, 1)
23 H_blocks.append(Qn)
24 H = sparse.block_diag(H_blocks) # H size = (3*(N-1), 3*(N-1)) = (3000, 3000)
25
26 # 构建C
27 B_block = np.hstack((B, -np.eye(2))) # B size = (2, 3)
28 C = kron(sparse.eye(N-1), B_block) # 构建了只构建了部分约束关系，仅包含了控制输入uk和状态 xk+1
29 C_dense = C.toarray() # 转为密集矩阵以便更新元素
30 # 在C矩阵中插入A矩阵
31 for k in range(N-2):
32     row_idx = (k*n) + np.arange(n)
33     col_idx = (k*(n+m)-n) + np.arange(n)
34     C_dense[np.ix_(row_idx, col_idx)] = A
35 C = sparse.csr_matrix(C_dense)
36
37 # 构建d
38 d_top = -A @ x0
39 d_bottom = np.zeros((C.shape[0]-n, 1))
40 d = np.vstack((d_top, d_bottom))
41
42 # 求解KKT系统
43 H_dense = H.toarray() # [3000, 3000]
44 C_dense = C.toarray() # [2000, 3000]
45 zeros_block = np.zeros((C.shape[0], C.shape[0]))
46 zeros_H = np.zeros((H.shape[0], 1))

```

```

47 left_matrix = np.block([
48     [H_dense, C_dense.T],
49     [C_dense, zeros_block]
50 ]) # [5000, 5000]
51 right_vector = np.vstack((zeros_H, d))
52 y = np.linalg.solve(left_matrix, right_vector) # 求解KKT系统
53
54 # 提取轨迹
55 z = y[:H.shape[0]]
56 Z = z.reshape((n+m, N-1), order='F') # z的维度为(3*(N-1), 1), 重塑为(3, N-1)的矩阵
57 xhist = Z[m:, :]
58 xhist = np.hstack((x0, xhist))
59 uhist = Z[:m, :]

```

定义 5.1 (Kronecker 乘积). 设 A 是 $m \times n$ 矩阵, B 是 $p \times q$ 矩阵, 则它们的 Kronecker 乘积 $A \otimes B$ 是一个 $mp \times nq$ 矩阵, 定义为:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix}$$

5.5.2 Riccati 递推: LQR QP 的结构

LQR的QP KKT线性系统很稀疏, 结构高度规则。以如下块状矩阵为例:

$$\left[\begin{array}{cccccc|ccc} R & 0 & 0 & 0 & 0 & 0 & B^T & 0 & 0 \\ 0 & Q & 0 & 0 & 0 & 0 & -I & A^T & 0 \\ 0 & 0 & R & 0 & 0 & -I & 0 & B^T & 0 \\ 0 & 0 & 0 & Q & 0 & 0 & 0 & -I & A^T \\ 0 & 0 & 0 & 0 & R & 0 & 0 & 0 & B^T \\ 0 & 0 & 0 & 0 & 0 & Q_N & 0 & 0 & -I \\ \hline B & -I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & A & B & -I & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & A & B & -I & 0 & 0 & 0 \end{array} \right] \begin{bmatrix} u_1 \\ x_2 \\ u_2 \\ x_3 \\ u_3 \\ x_4 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -A_1 x_1 \\ 0 \\ 0 \end{bmatrix}$$

考虑上三角的最后一行 (即第六行): 从Pontryagin的终端条件出发, 得到

$$Q_N x_4 - \lambda_4 = 0 \implies \lambda_4 = Q_N x_4$$

再看倒数第二行, 同时考虑到 $x_4 = Ax_3 + Bu_3$:

$$Ru_3 + B^T \lambda_4 = 0 \implies u_3 = -(R + B^T Q_N B)^{-1} B^T Q_N A x_3$$

递推出一个反馈增益 $K_3 = (R + B^T Q_N B)^{-1} B^T Q_N A$ 。

再看倒数第三行:

$$Qx_3 - \lambda_3 + A^T \lambda_4 = 0$$

代入 λ_4 和 u_3 后, 可以写成

$$Qx_3 - \lambda_3 + A^T Q_N (Ax_3 + Bu_3) = 0$$

进一步递推, 可以定义

$$P_3 = Q + A^T Q_N (A - BK_3)$$

最终，得到 Riccati 方程递推形式：

$$P_N = Q_N \quad (44)$$

$$K_n = (R + B^T P_{n+1} B)^{-1} B^T P_{n+1} A \quad (45)$$

$$P_n = Q + A^T P_{n+1} (A - B K_n) \quad (46)$$

Riccati 递推的含义与递推过程说明：

- P_n 是“代价到终点的二次型权重矩阵”，即从第 n 步出发到终点的最小总代价可以写为 $x_n^T P_n x_n$ 。 P_n 递推反映了未来代价对当前的影响。
- K_n 是最优反馈增益矩阵，给出每一步的最优控制律 $u_n = -K_n x_n$ 。
- Riccati 递推的过程是：从终点向前递推，先用终端权重 $P_N = Q_N$ 初始化，然后依次计算 $K_{N-1}, P_{N-1}, K_{N-2}, P_{N-2}, \dots$ ，直到 P_1 。
- 递推公式如下：

$$\begin{cases} P_N = Q_N \\ K_n = (R + B^T P_{n+1} B)^{-1} B^T P_{n+1} A \\ P_n = Q + A^T P_{n+1} (A - B K_n) \end{cases}$$

- 这样，先反向递推出所有 K_n ，再正向用

$$\begin{aligned} u_n &= -K_n x_n \\ x_{n+1} &= Ax_n + Bu_n \end{aligned}$$

生成最优轨迹。

Riccati 递推的代码实现

与5.5.1同一个例子，我们可以用Riccati递推来求解LQR，从而高效地求解LQR的QP问题。Riccati 递推的数值实现（Python 伪代码）：

```

1 # 初始化
2 P = np.zeros((n, n, N)) # [2, 2, 101]
3 K = np.zeros((m, n, N-1)) # [1, 2, 100]
4 P[:, :, N-1] = Qn
5
6 # 反向递推
7 for k in range(N-2, -1, -1):
8     temp = R + B.T @ P[:, :, k+1] @ B
9     K[:, :, k] = np.linalg.inv(temp) @ (B.T @ P[:, :, k+1] @ A)
10    P[:, :, k] = Q + A.T @ P[:, :, k+1] @ (A - B @ K[:, :, k])
11
12 # 正向生成最优轨迹
13 xhist = np.zeros((n, N))
14 xhist[:, 0:1] = x0
15 uhist = np.zeros((m, N-1))
16 for k in range(N-1):
17     uhist[:, k:k+1] = -K[:, :, k] @ xhist[:, k:k+1]
18     xhist[:, k+1:k+2] = A @ xhist[:, k:k+1] + B @ uhist[:, k:k+1]
```

说明：

- 首先用终端权重 Q_n 初始化 P_N ，然后从 $k = N - 2$ 递推到 $k = 0$ ，依次计算所有 K_k 和 P_k 。

- 得到所有 K_k 后，从初始状态 x_0 出发，正向递推生成最优控制 u_k 和状态 x_k 。
- 这种方法高效且数值稳定，适用于大规模LQR问题。

带噪声仿真与增益分析 在实际系统中，状态演化常常受到噪声影响。我们可以在前向仿真时加入高斯噪声，考察LQR控制的鲁棒性。下例展示了带噪声的状态轨迹，以及控制增益 K_1, K_2 随时间的变化。

```

1 # 有随机噪声的前向迭代
2 xhist_noise = np.zeros((n, N))
3 xhist_noise[:, 0:1] = x0
4 uhist_noise = np.zeros((m, N-1))
5 for k in range(N-1):
6     uhist_noise[:, k:k+1] = -K[:, :, k] @ xhist_noise[:, k:k+1]
7     noise = 0.01 * np.random.randn(2, 1)
8     xhist_noise[:, k+1:k+2] = A @ xhist_noise[:, k:k+1]
9         + B @ uhist_noise[:, k:k+1]
10        + noise
11
12 # 绘制控制增益随时间变化
13 plt.plot(range(N-1), K[0, 0, :], label='K1')
14 plt.plot(range(N-1), K[0, 1, :], label='K2')
15 plt.legend()

```

增益收敛与无穷时域LQR LQR的有限时域反馈增益 K_k 会随时间变化，但在步数足够大时， K_k 会收敛到一个极限 K_∞ 。可以用 `control.dlqr` 直接求解无穷时域LQR的最优增益：

```

1 # 无穷时域LQR增益计算
2 import control
3 Kinf = control.dlqr(A, B, Q, R)[0]
4 print("无穷时域增益矩阵 Kinf:\n", Kinf)
5 print("K[:, :, 0] --> Kinf =\n", K[:, :, 0] - Kinf)

```

闭环特征值 无穷时域LQR的闭环系统特征值决定了系统的收敛速度和稳定性：

```

1 eig_vals = np.linalg.eigvals(A - B @ Kinf)
2 print("闭环特征值:", eig_vals)

```

数值结果示例：

无穷时域增益矩阵 K_{inf} :

$$[2.5857009 \quad 3.44343592]$$

$$K[:, :, 0] - K_{\text{inf}} = [-6.729 \times 10^{-9} \quad -2.288 \times 10^{-9}]$$

闭环特征值: [0.899, 0.744]

这些结果表明，有限时域LQR的初始增益已非常接近无穷时域极限，闭环系统收敛且稳定。

要点总结：

- 这就是著名的 Riccati 方程或 Riccati 递推。
- 我们可以通过先进行一次**向后Riccati递推**，然后再进行一次**向前回代**，从初始条件出发依次计算出所有的 $x_{1:N}$ 和 $u_{1:N-1}$ ，从而高效地求解LQR的QP问题。
- 由于 QP 结构稀疏，递推可高效实现，复杂度 $O(Nn^3)$ 而不是 $O(N^3(n+m)^3)$ 。
- 这让我们可以得到一个反馈律 $u = -Kx$ ，而不只是开环最优轨迹。

5.6 无限时域 LQR (Infinite Horizon LQR)

下面考虑无限时域的LQR问题。

- 对于时不变LQR, K 增益矩阵在无限时域下会收敛到一个常数值。
- 对于稳定化问题, 几乎总是使用这个常数 K 。
- 对于 P 的递推, 可以用不动点迭代法或求根法数值求解。Julia/Matlab/Python 中的 `dare` 函数都可以直接求解。

5.6.1 有限时域 LQR

$$\min_{x_{1:N}, u_{1:N-1}} \sum_{k=1}^{N-1} \left[\frac{1}{2} x_k^T Q x_k + \frac{1}{2} u_k^T R u_k \right] + \frac{1}{2} x_N^T Q_N x_N \quad (47)$$

$$\text{s.t. } x_{k+1} = A_k x_k + B_k u_k \quad (48)$$

其中 A_k, B_k 可以随时间变化。

5.6.2 无限时域 LQR

$$\min_{x_{1:\infty}, u_{1:\infty}} \sum_{k=1}^{\infty} \left[\frac{1}{2} x_k^T Q x_k + \frac{1}{2} u_k^T R u_k \right] \quad (49)$$

$$\text{s.t. } x_{k+1} = A x_k + B u_k \quad (50)$$

其中 A 和 B 是时不变的。

5.7 如何求解LQR控制律 (How to solve the policy)

有两种常见求解策略:

- $\pi(x)$: 每次在线求解QP问题 (太慢)。
- $\pi(x) = -Kx$: 对于无限时域LQR, 只需一次性计算出反馈增益 K , 即可通过线性反馈生成所有的 U_k 。

如果我们想让机器人到达任意目标点, 该如何做? 已知目标点 x_{goal} , 有

$$x_{\text{goal}} = A x_{\text{goal}} + B u_{\text{goal}} \quad (51)$$

$$\tilde{x} = x - x_g, \quad \tilde{U} = U - U_g \quad (52)$$

则有

$$x_{k+1} - x_g = A x_k + B U_k - (A x_g + B U_g) \quad (53)$$

$$x_{k+1} - x_g = A(x_k - x_g) + B(U_k - U_g) \quad (54)$$

这样，只需对 \tilde{x}, \tilde{U} 求解LQR，得到反馈增益

$$K = \text{dLQR}(A, B, Q, R) \quad (55)$$

$$\tilde{U} = -K\tilde{x} \quad (56)$$

$$U - U_g = -K(x - x_g) \quad (57)$$

$$U = -K(x - x_g) + U_g \quad (58)$$

因此，我们可以把机器人驱动到任意目标。如果 A, B, Q, R 都是常数， K 也是常数。如果 A, B 是时变的，就需要每个时刻都重新计算 K （可离线提前算好）。

5.7.1 非线性系统的扩展

对某个定点进行泰勒展开，设 $\bar{x} = f(\bar{x}, \bar{u})$ ，则动态函数可线性化为：

$$x_{k+1} \approx f(\bar{x}, \bar{u}) + \frac{\partial f}{\partial x}\Big|_{\bar{x}, \bar{u}}(x - \bar{x}) + \frac{\partial f}{\partial u}\Big|_{\bar{x}, \bar{u}}(u - \bar{u}) \quad (59)$$

$$\bar{x} + \Delta x_{k+1} = f(\bar{x}, \bar{u}) + A\Delta x + B\Delta u \quad (60)$$

$$\Delta x_{k+1} = \bar{A}\Delta x + \bar{B}\Delta u \quad (61)$$

5.8 可控性 (Controllability)

如何判断 LQR 能否工作？对于时不变系统，有一个简单的答案。对任意初始状态 x_0 ， x_n 的表达式为：

$$x_n = Ax_{n-1} + Bu_{n-1} \quad (62)$$

$$= A(Ax_{n-2} + Bu_{n-2}) + Bu_{n-1} \quad (63)$$

$$= A^n x_0 + A^{n-1}Bu_0 + \dots + Bu_{n-1} \quad (64)$$

$$= [B \ AB \ A^2B \ \dots \ A^{n-1}B] \begin{bmatrix} u_{n-1} \\ u_{n-2} \\ \vdots \\ u_0 \end{bmatrix} + A^n x_0 \quad (65)$$

我们的目标是让 $x_n = 0$ ，则可以定义可控性矩阵 C ：

定义 5.2 (可控性矩阵 (Controllability Matrix))。

$$C = [B \ AB \ A^2B \ \dots \ A^{n-1}B]$$

要想从任意 x_0 驱动到任意 x_n ，可控性矩阵必须满秩：

$$\text{rank}(C) = n, \quad n = \dim(x)$$

这等价于对 u_0, \dots, u_{n-1} 求解如下最小二乘问题，直接使用PseudoInverse ??：

$$\begin{bmatrix} u_{n-1} \\ u_{n-2} \\ \vdots \\ u_0 \end{bmatrix} = [C^T(CC^T)^{-1}] [x_n - A^n x_0]$$

其中 $[C^T(CC^T)^{-1}]$ 是 C 的伪逆，需要 CC^T 可逆，这个 CC^T 在这里称作 Controllability Gramian (可控性Gramian，参考 Steve Brunton 的讲解⁵)。由于 Cayley-Hamilton 定理 (见附页??)， A^n 可以写成更小幂次 A 的线性组合：

$$A^n = \sum_{k=0}^{n-1} \alpha_k A^k$$

因此，继续增加步数/增加 C 的列数，并不能提高 C 的秩。在时变系统情况下，用QP方法比上述解法更好。

5.8.1 可控性 Gramian

对于系统：

$$\dot{x} = Ax + Bu \quad y = Cx + Du, x \in \mathbb{R}^n$$

我们有

$$C = [B \ AB \ A^2B \ \dots \ A^{n-1}B]$$

若 $\text{rank}(C) = n$ ，则系统可控。但是这是一个T/F问题，我们无法判断是哪个方向是否可控。我们需要衡量在不同方向的可控性。回顾矩阵分析中的SVD分解。首先我们考虑这个ODE的解：

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\tau)}Bu(\tau)d\tau$$

定义 5.3 (可控性 Gramian). 线性系统的可控性 Gramian 矩阵定义为：

$$W_c(t_0, t_f) = \int_{t_0}^{t_f} e^{A(t_f-\tau)}BB^Te^{A^T(t_f-\tau)}d\tau$$

对于时不变系统，若取 $t_0 = 0, t_f = T$ ，则可简写为：

$$W_c(T) = \int_0^T e^{A(T-\tau)}BB^Te^{A^T(T-\tau)}d\tau$$

在无穷时域情况下，若系统稳定（即 A 的所有特征值实部均为负），则有：

$$W_c = \int_0^\infty e^{At}BB^Te^{A^Tt}dt$$

此时， W_c 满足 Lyapunov 方程：

$$AW_c + W_cA^T + BB^T = 0$$

5.8.2 Gramian 与可控性的关系

Gramian 矩阵 W_c 与可控性有着深刻的联系：

- 系统可控的充要条件为 W_c 正定，或等价地， $\text{rank}(W_c) = n$
- W_c 的行列式大小表示系统整体的可控性强弱
- W_c 的最小特征值表示最难控制的方向的可控性

⁵<https://www.youtube.com/watch?v=ZNHx62HbKNA>

5.8.3 Gramian 的 SVD 分解与可控性方向

对可控性 Gramian 进行奇异值分解:

$$W_c = V\Sigma V^T$$

其中 $V = [v_1, v_2, \dots, v_n]$ 是正交矩阵, $\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_n)$ 是奇异值矩阵, 且 $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n \geq 0$ 。

这种分解揭示了系统可控性的重要信息:

- 奇异值 σ_i 表示在方向 v_i 上的可控性大小
- 最大奇异值 σ_1 对应最容易控制的方向 v_1
- 最小奇异值 σ_n 对应最难控制的方向 v_n
- 如果 $\sigma_n = 0$, 则系统在方向 v_n 不可控

5.8.4 方向可控性的量化

要定量分析在特定方向 η (单位向量) 上的可控性, 可计算:

$$\gamma_\eta = \eta^T W_c \eta$$

γ_η 度量了将状态从原点移动到方向 η 所需的能量, 较大的 γ_η 表示该方向更容易控制。

最难控制的方向对应于 W_c 最小特征值的特征向量:

$$\min_{\|\eta\|=1} \eta^T W_c \eta = \sigma_n$$

系统总体可控性可通过下列指标度量:

- 最小可控性: $\min_{\|\eta\|=1} \eta^T W_c \eta = \sigma_n$
- 可控性的度量: $\text{trace}(W_c) = \sum_{i=1}^n \sigma_i$
- 可控容积: $\det(W_c) = \prod_{i=1}^n \sigma_i$

5.8.5 计算举例

考虑一个简单的二维系统:

$$\dot{x} = \begin{bmatrix} -1 & 1 \\ 0 & -2 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u$$

我们可以通过求解 Lyapunov 方程或直接计算积分来获得可控性 Gramian W_c 。然后通过 SVD 分解, 确定最易控制和最难控制的方向。

这种可视化称为可控性椭球 (图17), 直观展示了系统在各个方向上的可控性强弱, 为控制系统设计提供了重要参考。

TODO: 绘制图像

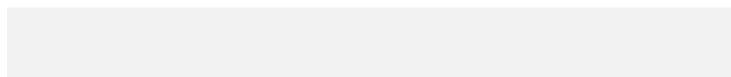


图 17: 可控性椭球示意图。椭球的主轴方向由可控性 Gramian 的特征向量决定，轴长由特征值决定。较长的轴表示该方向更容易控制。

6 动态规划

从LQR的QP推导Riccati方法中，我们看到那种可以递推出来的方法。过去的控制输入会影响未来的状态，但未来的控制输入不能影响过去的状态。我们可以从轨迹的终点开始倒推，计算出最优轨迹。这个方法就是动态规划。

6.1 贝尔曼原理

- 最优控制问题本质上具有顺序结构。
- 过去的控制输入会影响未来的状态，但未来的控制输入不能影响过去的状态。
- 贝尔曼原理（即最优化原理）指出了这一点对最优轨迹的影响。
- 最优轨迹的子轨迹对于相应定义的子问题也必须是最优的。

6.2 动态规划

动态规划（Dynamic Programming, DP）是一种将复杂问题分解为更简单子问题的优化方法。在最优控制中，动态规划自底向上地从终点递推最优轨迹。其核心思想由贝尔曼最优化原理（Bellman's Principle of Optimality）给出：

定义 6.1 (贝尔曼最优化原理). 对于任意最优轨迹，其任意子轨迹在相应的子问题下也必须是最优的。

在最优控制问题中，定义“最优代价到达函数”(optimal cost-to-go function)，即价值函数 (value function) $V_k(x)$ ，表示从时刻 k 的状态 x 出发，按照最优策略所能获得的累计最小代价。对于终点，有

$$V_N(x) = \frac{1}{2}x^T Q_N x = \frac{1}{2}x^T P_N x. \quad (1)$$

对于 $k = N - 1$, 有如下递推关系:

$$V_{N-1}(x) = \min_u \left\{ \frac{1}{2}x^T Q_{N-1}x + \frac{1}{2}u^T R_{N-1}u + V_N(A_{N-1}x + B_{N-1}u) \right\}. \quad (66)$$

将 V_N 代入并展开, 有

$$\begin{aligned} V_{N-1}(x) = \min_u & \left\{ \frac{1}{2}x^T Q_{N-1}x + \frac{1}{2}u^T R_{N-1}u \right. \\ & \left. + \frac{1}{2}(A_{N-1}x + B_{N-1}u)^T Q_N(A_{N-1}x + B_{N-1}u) \right\}. \end{aligned} \quad (67)$$

对 u 求极值, 令梯度为零, 得到最优控制律:

$$u_{N-1}^* = -(R_{N-1} + B_{N-1}^T Q_N B_{N-1})^{-1} B_{N-1}^T Q_N A_{N-1} x. \quad (5)$$

定义

$$K_{N-1} = (R_{N-1} + B_{N-1}^T Q_N B_{N-1})^{-1} B_{N-1}^T Q_N A_{N-1}, \quad (68)$$

则 $u_{N-1}^* = -K_{N-1}x$ 。

将最优 u 代入 $V_{N-1}(x)$, 得到

$$V_{N-1}(x) = \frac{1}{2}x^T P_{N-1}x, \quad (7)$$

其中

$$P_{N-1} = Q_{N-1} + K_{N-1}^T R_{N-1} K_{N-1} + (A_{N-1} - B_{N-1} K_{N-1})^T Q_N (A_{N-1} - B_{N-1} K_{N-1}). \quad (69)$$

如此递推, 可得到 K_k 和 P_k 的递推关系, 这即为 Riccati 方程的动态规划推导。

6.2.1 动态规划算法

动态规划 (DP) 反向递推算法

其中, $l(x, u)$ 表示一步的即时代价函数, $l_N(x)$ 为终端代价函数。

反向 DP 算法:

1. $V_N(x) \leftarrow l_N(x)$
 2. $K \leftarrow N$
 3. **while** $K > 1$ **do**
 - (a) $V_{k-1} = \min_u [l(x, u) + V_k(f(x, u))]$ \triangleright 贝尔曼方程
 - (b) $k \leftarrow k - 1$
- 已知 $V_k(x)$ 时, 最优反馈策略为:

$$u_k^*(x) = \arg \min_u [l(x, u) + V_{k+1}(f(x, u))] \quad (8)$$

- DP方程也可用行动-价值 (action-value) 函数 Q (或 S) 等价表示:

$$Q_k(x, u) = l(x, u) + V_{k+1}(f(x, u)) \quad (9)$$

$$u_k^*(x) = \arg \min_u Q_k(x, u) \quad (10)$$

- 在LQR等问题中, Q 有时用于状态加权矩阵, 这里用 $Q_k(x, u)$ 或 $S_k(x, u)$ 均可。

与强化学习中的贝尔曼方程对比

- 在强化学习 (Reinforcement Learning, RL) 中, 贝尔曼方程通常写为:

$$V^\pi(x) = \mathbb{E}_{u \sim \pi(x)} [l(x, u) + \gamma V^\pi(f(x, u))]$$

$$Q^\pi(x, u) = l(x, u) + \gamma \mathbb{E}_{x' \sim f(x, u)} [V^\pi(x')]$$

其中 γ 为折扣因子, π 为策略, \mathbb{E} 表示对随机性的期望。

- RL中的贝尔曼方程本质上是DP的推广, 允许系统动力学和奖励函数为随机过程, 并引入折扣因子。
- RL算法 (如Q-learning、价值迭代等) 正是基于对贝尔曼方程的近似或迭代求解。

6.2.2 连续空间连续时间的动态规划 HJB

在连续时间、连续状态空间的最优控制问题中, Bellman 原理导出的值函数满足一个非线性偏微分方程, 称为 **Hamilton–Jacobi–Bellman (HJB)** 方程。它是动态规划在连续设置下的推广。

系统设定 考虑如下控制系统:

$$\dot{x}(t) = f(x(t), u(t)), \quad x(t) \in \mathbb{R}^n, \quad u(t) \in \mathcal{U} \subseteq \mathbb{R}^m \quad (70)$$

给定状态的初始值 $x(0) = x_0$, 控制目标是最小化如下性能指标:

$$J(x_0) = \int_0^T L(x(t), u(t)) dt + \phi(x(T)) \quad (71)$$

其中 $L(x, u)$ 是运行代价, $\phi(x)$ 是终端代价。

最优值函数定义 定义最优值函数为:

$$V(x, t) = \min_{u(\cdot)} \left[\int_t^T L(x(s), u(s)) ds + \phi(x(T)) \right] \quad (72)$$

HJB 方程形式 则 $V(x, t)$ 满足的 Hamilton–Jacobi–Bellman 方程为:

$$\frac{\partial V(x, t)}{\partial t} + \min_{u \in \mathcal{U}} \{L(x, u) + \nabla_x V(x, t)^T f(x, u)\} = 0 \quad (73)$$

终端条件为:

$$V(x, T) = \phi(x) \quad (74)$$

若考虑无限时域 $T \rightarrow \infty$ 且无终端代价 (如 LQR), 则 HJB 简化为稳态形式:

$$0 = \min_{u \in \mathcal{U}} \{L(x, u) + \nabla_x V(x)^T f(x, u)\} \quad (75)$$

—

HJB 方程求解示例: 线性系统 + 二次代价 考虑一维线性系统:

$$\dot{x}(t) = ax(t) + bu(t) \quad (76)$$

目标是最小化如下无限时域性能指标:

$$J(x_0) = \int_0^\infty (qx^2(t) + ru^2(t)) dt \quad (77)$$

其中 $q > 0, r > 0$ 为权重常数。

HJB 方程写法 定义值函数 $V(x)$ 为从状态 x 开始的最小代价, 则 HJB 方程为:

$$0 = \min_u \{qx^2 + ru^2 + V'(x)(ax + bu)\} \quad (78)$$

最优控制律 为求最优控制, 构造代价项:

$$\mathcal{H}(x, u) = qx^2 + ru^2 + V'(x)(ax + bu) \quad (79)$$

对 u 求偏导并令其为零:

$$\frac{\partial \mathcal{H}}{\partial u} = 2ru + bV'(x) = 0 \Rightarrow u^*(x) = -\frac{b}{2r}V'(x) \quad (80)$$

猜测值函数形式 设值函数为二次型:

$$V(x) = Px^2 \Rightarrow V'(x) = 2Px \quad (81)$$

代入最优控制律得:

$$u^*(x) = -\frac{b}{2r} \cdot 2Px = -\frac{bP}{r}x \quad (82)$$

将 $u^*(x)$ 代入 HJB 方程:

$$0 = qx^2 + r \left(-\frac{bP}{r}x \right)^2 + 2Px \cdot \left(ax - b \cdot \frac{bP}{r}x \right) \quad (83)$$

$$= x^2 \left[q + \frac{b^2 P^2}{r} + 2aP - \frac{2b^2 P^2}{r} \right] \quad (84)$$

$$= x^2 \left[q + 2aP - \frac{b^2 P^2}{r} \right] \quad (85)$$

因此 P 满足如下代数 Riccati 方程:

$$\frac{b^2}{r}P^2 - 2aP - q = 0 \quad (86)$$

数值例子 设:

$$a = 1, \quad b = 1, \quad q = 1, \quad r = 1$$

则 Riccati 方程变为:

$$P^2 - 2P - 1 = 0 \quad \Rightarrow \quad P = 1 + \sqrt{2} \quad (\text{取正根}) \quad (87)$$

得到最优控制律:

$$u^*(x) = -\left(1 + \sqrt{2}\right)x \quad (88)$$

最终结果总结

- 状态方程: $\dot{x} = x + u$
- 性能指标: $J = \int_0^\infty (x^2 + u^2) dt$
- 最优值函数: $V(x) = (1 + \sqrt{2})x^2$
- 最优控制律: $u^*(x) = -(1 + \sqrt{2})x$

6.2.3 维数灾难 (The Curse)

- 对于全局最优, DP方法是充分的。
- 但它只对简单问题 (如LQR或低维问题) 可行。
- $V(x)$ 在LQR问题中保持二次型, 但对一般非线性问题则难以解析表达。
- 即使可以表达, $\min_u S(x, u)$ 通常是非凸的, 单独求解也可能很困难。
- 随着状态维数增加, DP的计算量急剧上升, 因为 $V(x)$ 难以表示。

6.2.4 我们为什么关心这些？

- 近似DP（用函数逼近来表示 $V(x)$ 或 $S(x, u)$ ）非常有效。
- 是现代强化学习方法的基础。
- DP很好地推广到随机问题（只需将所有内容包裹在期望算子下），而Pontryagin方法则不能。

6.2.5 什么是拉格朗日乘子？

- 回忆QP中的Riccati推导：

$$\lambda_n = P_n x_n \quad (11)$$

$$\begin{aligned} P_n &= Q + A^T P_{n+1} (A - BK) \\ &= Q + K^T R K + (A - BK)^T P_{n+1} (A - BK) \end{aligned} \quad (12)$$

- 对于DP：

$$V_n(x) = \frac{1}{2} x^T P_n x \quad (14)$$

$$\lambda_n = \nabla_x V_n(x) \quad (15)$$

- 动力学的乘子就是 cost-to-go 的梯度。
- 这种思想可推广到一般非线性情形，不仅限于LQR。

6.3 例子

与5.5.1同一个例子，这里我们对比QP和DP的方法。

6.3.1 QP 与 DP 实现对比示例

下面给出一个简化版的 Python 代码片段，演示如何分别使用二次规划（QP）和动态规划（DP）求解同一个离散 LQR 问题，并对比它们的核心步骤。

核心差异

- QP 方法将整个时间序列一次性拼成一个大规模的二次规划问题，依赖通用求解器求解 KKT 系统，适用于问题规模较小或需要精确解的场景。
- DP 方法通过反向 Riccati 递推，逐步计算每一步的最优反馈增益，计算量随时间步线性增长，更加高效且具有在线更新能力。
- QP 可直接获得拉格朗日乘子，从而得到值函数在每一步的梯度；DP 则显式构造 P 矩阵，通过 $\lambda_k = P_k x_k$ 恢复梯度信息。

7 MPC 模型预测控制

7.1 凸优化与MPC简介

- 线性二次调节器（LQR）非常强大，但通常需要显式处理约束。
- 这些约束（如力矩限制）通常可以被编码为凸集。在机械臂轨迹规划中，常见约束包括：
 - **关节速度限制：** $\dot{q}_{\min} \leq \dot{q} \leq \dot{q}_{\max}$ ，其中 \dot{q} 为关节速度向量， $\dot{q}_{\min}, \dot{q}_{\max}$ 为上下界，均为线性不等式，构成凸集。
 - **末端执行器工作空间约束：** $Ax \leq b$ ，其中 x 为末端位姿， A, b 定义多面体工作空间，也是凸集。
 - **力/力矩限制：** $\|u\|_2 \leq u_{\max}$ ， u 为控制输入（如关节力矩），该约束为椭球型凸集。

这些约束之所以是凸的，是因为：线性不等式（如 $\dot{q}_{\min} \leq \dot{q} \leq \dot{q}_{\max}$ 和 $Ax \leq b$ ）的可行域是凸集，任意两点的连线也在集合内；而 ℓ_2 范数约束（如 $\|u\|_2 \leq u_{\max}$ ）对应于原点为中心的球体，也是凸集。这保证了优化问题的可解性和全局最优性。详见《现代机器人学：力学，规划，控制》第13章。

- 约束会破坏Riccati解析解，但我们仍然可以在线求解二次规划（QP）问题。
- 随着计算能力提升，凸MPC变得非常流行。

7.2 凸性基础

- **凸集：**连接集合中任意两点的线段仍在集合内。
- 常见例子：线性子空间、半空间/多面体、椭球、锥等。
- **凸函数：**其上方图是凸集。例如线性函数、二次型、范数等。
- **凸优化问题：**在凸集上最小化凸函数。
- 典型问题：线性规划（LP）、二次规划（QP）、带二次约束的QP（QCQP）、二阶锥规划（SOCP）。
- 凸优化问题没有虚假的局部最优解，KKT条件下的解即为全局最优。
- 牛顿法等数值方法收敛快，适合实时控制。

7.3 MPC基本思想

- MPC本质上是“带约束的LQR”。
- 若已知代价到终点的函数 $V(x)$ ，可通过动态规划将控制问题转化为一步优化问题。
- LQR难以处理约束，MPC通过在有限时域内显式加入约束，提升了控制效果。
- MPC思想：将一步优化扩展为多步优化（预测时域），每次只执行第一个控制量，滚动优化。

- 当无额外约束时，MPC与LQR等价。
- 预测时域越长，对 $V(x)$ 的依赖越小； $V(x)$ 越好，时域可短。

7.4 平面四旋翼示例

- 对于某些非线性系统，MPC优于LQR，因为MPC能显式处理约束。
- **技巧：**对非线性系统线性化后，可加约束保证变量处于线性化有效区间，从而保证近似精度。

7.5 凸MPC示例

7.5.1 火箭着陆

推力矢量约束呈锥形（喷火方向的极限）：我们可以看到，火箭推进的尾焰在喷嘴处形成一个锥形的推力矢量约束。这个约束可以用来限制火箭的推进方向和推力大小，以确保火箭在着陆时能够保持稳定的姿态和速度。

图：火箭着陆

SpaceX和JPL的火星着陆任务采用基于SOCP的MPC，并对动力学进行线性化处理。

7.5.2 足式机器人

接触力需满足“摩擦锥”约束：

图：摩擦锥

摩擦锥常被近似为金字塔形状，此时约束变为线性 ($Ax \leq \rho$)。
MIT Cheetah等四足机器人采用基于QP的MPC，并对动力学线性化。

7.6 非线性动力学怎么办？

- 线性化方法在实际中常常效果很好，能用就用。
- 非线性动力学会导致MPC问题变为非凸，无法保证收敛。
- 实践中经过一定调整（如自动驾驶），非凸MPC也能取得不错效果。

7.7 一个四轴飞行器的例子

7.7.1 问题描述

考虑一个平面四旋翼飞行器的控制问题，该飞行器在二维平面内运动，具有3个自由度：水平位置 x 、垂直位置 y 和姿态角 θ 。

系统参数：

- 质量： $m = 1.0 \text{ kg}$
- 臂长： $\ell = 0.3 \text{ m}$

- 转动惯量: $J = 0.2m\ell^2$
- 重力加速度: $g = 9.81 \text{ m/s}^2$

控制约束: 每个推进器的推力限制为:

$$0.2mg \leq u_i \leq 0.6mg, \quad i = 1, 2$$

控制目标: 将四旋翼从初始状态 $x_0 = [1.0, 2.0, 0, 0, 0, 0]^T$ (位置(1, 2), 零姿态角和零速度) 控制到目标悬停点 $x_{ref} = [0, 1.0, 0, 0, 0, 0]^T$ (位置(0, 1), 零姿态角和零速度)。

性能指标: 比较LQR和MPC两种控制方法在存在推力约束情况下的控制性能, 包括轨迹跟踪精度、控制输入的约束满足情况以及收敛速度。

7.7.2 系统模型

四旋翼飞行器在二维平面内的动力学建模需要考虑其物理结构和受力分析。

坐标系定义:

- 状态向量: $x = [x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}]^T$
- 其中 (x, y) 为质心位置, θ 为机体相对水平面的姿态角 (正值表示逆时针转动)
- $(\dot{x}, \dot{y}, \dot{\theta})$ 分别为线速度和角速度
- 控制输入: $u = [u_1, u_2]^T$, 分别为左、右推进器推力

物理模型推导:

1. 几何关系分析: 四旋翼机体倾斜角度为 θ 时, 每个推进器产生的推力矢量相对于惯性坐标系的方向为:

- 水平方向 (x 轴): 推力在水平方向的投影为 $\sin \theta$
- 垂直方向 (y 轴): 推力在垂直方向的投影为 $\cos \theta$

2. 牛顿第二定律应用:

水平方向运动方程: 根据牛顿第二定律, 水平方向的合外力等于质量乘以加速度:

$$\sum F_x = m\ddot{x}$$

水平方向只有推力的水平分量:

$$F_x = (u_1 + u_2) \sin \theta$$

因此:

$$m\ddot{x} = (u_1 + u_2) \sin \theta$$

$$\ddot{x} = \frac{1}{m}(u_1 + u_2) \sin \theta$$

垂直方向运动方程: 垂直方向受到推力的垂直分量和重力作用:

$$\sum F_y = m\ddot{y}$$

垂直方向的合外力为:

$$F_y = (u_1 + u_2) \cos \theta - mg$$

其中 $(u_1 + u_2) \cos \theta$ 为总推力在垂直方向的分量, mg 为重力(向下为负)。
因此:

$$m\ddot{y} = (u_1 + u_2) \cos \theta - mg$$

$$\ddot{y} = \frac{1}{m}(u_1 + u_2) \cos \theta - g$$

3. 转动方程推导:

力矩分析: 以质心为转动中心, 分析各推进器产生的力矩。设左推进器距质心距离为 $\ell/2$, 右推进器距质心距离也为 $\ell/2$ 。

左推进器产生的力矩(绕质心, 逆时针为正):

$$M_1 = -u_1 \cdot \frac{\ell}{2}$$

右推进器产生的力矩:

$$M_2 = u_2 \cdot \frac{\ell}{2}$$

总力矩:

$$M = M_1 + M_2 = -u_1 \cdot \frac{\ell}{2} + u_2 \cdot \frac{\ell}{2} = \frac{\ell}{2}(u_2 - u_1)$$

根据转动定律:

$$M = J\ddot{\theta}$$

其中转动惯量 $J = 0.2m\ell^2$ (给定), 因此:

$$J\ddot{\theta} = \frac{\ell}{2}(u_2 - u_1)$$

$$\ddot{\theta} = \frac{1}{J}\frac{\ell}{2}(u_2 - u_1)$$

4. 状态空间表示: 将二阶微分方程组转换为一阶状态方程组。定义状态向量 $x = [x_1, x_2, x_3, x_4, x_5, x_6]^T = [x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}]^T$:

$$\dot{x} = f(x, u) = \begin{bmatrix} x_4 \\ x_5 \\ x_6 \\ \frac{1}{m}(u_1 + u_2) \sin x_3 \\ \frac{1}{m}(u_1 + u_2) \cos x_3 - g \\ \frac{1}{J}\frac{\ell}{2}(u_2 - u_1) \end{bmatrix}$$

6

5. 平衡点线性化: 在悬停状态($x_{hover} = [0, 0, 0, 0, 0, 0]^T$, $u_{hover} = [0.5mg, 0.5mg]^T$)处, 推力平衡重力, 无净力矩。

对非线性函数 $f(x, u)$ 在平衡点处进行泰勒展开并保留一阶项:

$$f(x, u) \approx f(x_{hover}, u_{hover}) + \left. \frac{\partial f}{\partial x} \right|_{hover} (x - x_{hover}) + \left. \frac{\partial f}{\partial u} \right|_{hover} (u - u_{hover})$$

⁶这里 x_4, x_5, x_6 分别代表 $\dot{x}, \dot{y}, \dot{\theta}$, 即位置和角度的导数。 $\dot{x}_1 = x_4$, $\dot{x}_2 = x_5$, $\dot{x}_3 = x_6$, 而不是 x_1, x_2 , 因为 x_1, x_2, x_3 本身就是位置和角度变量。

由于 $f(x_{\text{hover}}, u_{\text{hover}}) = 0$ (平衡点性质), 得到线性化系统:

$$\dot{x} \approx A(x - x_{\text{hover}}) + B(u - u_{\text{hover}})$$

其中:

$$A = \frac{\partial f}{\partial x} \Big|_{x_{\text{hover}}, u_{\text{hover}}}, \quad B = \frac{\partial f}{\partial u} \Big|_{x_{\text{hover}}, u_{\text{hover}}}$$

注意: A 矩阵中(4, 3)元素来自 $\frac{\partial}{\partial \theta}[(u_1 + u_2) \sin \theta]|_{\theta=0} = (u_1 + u_2) \cos(0) = mg$ 。

采用RK4方法对非线性系统进行数值积分和离散化, 得到离散线性系统 $x_{k+1} = A_d x_k + B_d u_k$, 用于LQR和MPC控制器设计。

7.7.3 LQR解法

对于线性化后的离散系统, LQR控制器设计遵循标准的最优控制理论框架。

控制目标函数: 设计有限时域二次型性能指标:

$$J = \frac{1}{2} x_N^T Q_f x_N + \frac{1}{2} \sum_{k=0}^{N-1} (x_k^T Q x_k + u_k^T R u_k)$$

其中:

- $Q = I_{6 \times 6}$ 为状态权重矩阵
- $R = 0.01 \cdot I_{2 \times 2}$ 为控制权重矩阵
- Q_f 为终端状态权重矩阵

离散代数Riccati方程 (DARE): 对于无约束情况, 最优控制增益 K 通过求解DARE获得:

$$P = Q + A^T P A - A^T P B (R + B^T P B)^{-1} B^T P A$$

终端权重矩阵 $Q_f = P$ 选择为该Riccati方程的解, 保证了性能指标的最优化。

控制律: LQR控制器采用线性反馈形式:

$$u_k = u_{\text{hover}} - K(x_k - x_{\text{ref}})$$

其中 $K = (R + B^T P B)^{-1} B^T P A$ 为最优反馈增益矩阵。

约束处理: LQR本身无法直接处理推力约束 $u_{\min} \leq u \leq u_{\max}$ 。在实现中采用饱和函数进行约束:

$$u_k^{\text{actual}} = \text{clamp}(u_k^{\text{LQR}}, u_{\min}, u_{\max})$$

这种饱和处理会破坏LQR的最优化, 可能导致性能下降或系统不稳定。

7.7.4 MPC解法

MPC (模型预测控制) 通过将约束优化问题表述为二次规划 (QP) 来直接处理系统约束。

MPC优化问题表述: 在每个时刻 k , MPC求解以下有限时域优化问题:

$$\min_{u_k, \dots, u_{k+N-1}} \frac{1}{2} x_{k+N}^T Q_f x_{k+N} + \frac{1}{2} \sum_{j=0}^{N-1} (x_{k+j}^T Q x_{k+j} + u_{k+j}^T R u_{k+j})$$

约束条件：

$$x_{k+j+1} = Ax_{k+j} + Bu_{k+j}, \quad j = 0, \dots, N-1 \quad (89)$$

$$u_{min} \leq u_{k+j} \leq u_{max}, \quad j = 0, \dots, N-1 \quad (90)$$

$$x_{k+0} = x_k \text{ (当前状态)} \quad (91)$$

其中 $N = 20$ 为预测时域长度（1秒，20Hz采样频率）。

QP标准形式转换： 将MPC问题重新组织为QP标准形式：

$$\min_z \frac{1}{2} z^T H z + b^T z$$

约束： $Dz = d$ (等式约束) 和 $l \leq z \leq u$ (不等式约束)

决策变量 z 包含所有时域内的控制和状态变量：

$$z = [u_k^T, x_{k+1}^T, u_{k+1}^T, x_{k+2}^T, \dots, u_{k+N-1}^T, x_{k+N}^T]^T$$

约束矩阵构建：

- 动力学约束矩阵 C : 编码状态转移方程 $x_{k+j+1} = Ax_{k+j} + Bu_{k+j}$
- 控制选择矩阵 U : 从决策变量中提取控制输入
- 组合约束矩阵: $D = [C; U]$
- 约束边界: $lb = [Ax_k; u_{min} - u_{hover}]$, $ub = [Ax_k; u_{max} - u_{hover}]$

滚动时域实现：

1. 获取当前状态 x_k
2. 更新QP约束边界 ($lb[1:6] = -Ax_k$, $ub[1:6] = -Ax_k$)
3. 使用OSQP求解器求解QP问题
4. 提取第一个控制输入 u_k 并应用到系统
5. 移动到下一时刻，重复上述过程

7.7.5 结果对比

LQR方法	MPC方法
优化问题: 无约束二次型优化 $\min_u \sum_{k=0}^{\infty} (x_k^T Q x_k + u_k^T R u_k)$	优化问题: 带约束的二次规划 (QP) $\min_{u_{k:k+N-1}} \sum_{j=0}^{N-1} (x_{k+j}^T Q x_{k+j} + u_{k+j}^T R u_{k+j})$ $+ x_{k+N}^T Q_f x_{k+N}$
控制律: 线性反馈 $u_k = u_{hover} - K(x_k - x_{ref})$	控制律: 在线优化求解 $u_k^* = \arg \min \text{QP problem}$
其中 $K = (R + B^T P B)^{-1} B^T P A$ 约束处理: 无法直接处理，采用饱和函数 $u_k^{actual} = \text{clamp}(u_k, u_{min}, u_{max})$	每步只执行 u_k^* 的第一个元素 约束处理: 显式纳入约束条件 $u_{min} \leq u_{k+j} \leq u_{max}, \quad j = 0, \dots, N-1$
破坏最优性: 计算复杂度: 极低，仅矩阵乘法 $\mathcal{O}(n)$	保持最优性: 计算复杂度: 较高，需在线求解QP $\mathcal{O}(n^3)$
稳定性: 理论保证（无约束时） 约束时可能失稳	稳定性: 需要特殊设计保证 终端约束/代价函数设计

表 1: LQR与MPC方法对比

仿真结果分析:

从四旋翼控制仿真结果可以看出:

- **轨迹跟踪性能:** MPC在 x 和 y 方向都表现出更平滑的收敛特性，而LQR由于约束饱和导致振荡较大。
- **约束满足:** MPC严格满足推力约束 $0.2mg \leq u_i \leq 0.6mg$ ，而LQR通过饱和函数强制满足约束，破坏了控制器的最优设计。

- **姿态角控制：** MPC能更好地控制姿态角 θ 保持在小角度范围内，验证了线性化假设的有效性。
- **控制输入：** MPC的控制输入更加平滑，避免了LQR中由于约束饱和导致的突变。

优劣势总结：

LQR优势：

- 计算效率极高，适合高频控制
- 理论成熟，稳定性有保证
- 实现简单，调参容易

LQR劣势：

- 无法处理约束，饱和处理破坏最优性
- 约束激活时可能导致性能严重下降
- 对非线性系统适应性差

MPC优势：

- 显式处理各种约束（状态、控制、输出约束）
- 预测性控制，对模型不确定性有一定鲁棒性
- 可处理多目标优化问题
- 约束下仍保持最优性

MPC劣势：

- 计算量大，需要实时求解优化问题
- 参数调节复杂（预测时域、权重矩阵等）
- 稳定性需要特殊设计（终端约束、终端代价）
- 对求解器性能依赖较大

8 轨迹优化

在这一节，我们将介绍轨迹优化的几种方式。

- 非线性轨迹优化 (Non-linear Trajectory optimization)，主要讲述使用DDP/iLQR等方法进行非线性的轨迹优化，并且考虑将时间步长作为优化变量。
- 直接轨迹优化 (Direct Trajectory optimization)

8.1 非线性轨迹优化问题

$$\min_{x_1, x_1, u_1, \dots, J} = \sum_{k=1}^{N-1} l_k(x_k, u_k) + l_N(x_N) \quad (92)$$

$$\text{s.t. } x_{k+1} = f(x_k, u_k) \quad (93)$$

$$x_k \in X_k \quad (94)$$

$$u_k \in U_k \quad (95)$$

其中成本函数是非凸的，动力学模型为非线性动力学，约束条件为非凸约束。通常假设成本和约束是 C^2 连续的。

8.1.1 微分动态规划 (DDP/iLQR)

- 对于Cost-to-go函数 $V_k(x)$ ，我们可以使用二阶泰勒展开来近似那些非线性函数。
- 在强化学习中，通常使用神经网络拟合价值函数
- 可能实现非常快速的收敛

求解DDP/iLQR 非线性cost-to-go函数 $V_k(x)$ 的泰勒展开：

$$V_k(x + \Delta x) \approx V_k(x) + p_k^T \Delta x + \frac{1}{2} \Delta x^T P_k \Delta x \quad (96)$$

其中 $p_N = \nabla_x l_N(x)$, $P_N = \nabla_{xx}^2 l_N(x)$

同样，action-value函数 $S_k(x_k, u_k) = l_k(x_k, u_k) + V_{k+1}(f(x_k, u_k))$ 的泰勒展开为：

$$S_k(x + \Delta x, u + \Delta u) \approx S_k(x, u) + \begin{bmatrix} g_x \\ g_u \end{bmatrix}^T \begin{bmatrix} \Delta x \\ \Delta u \end{bmatrix} + \quad (97)$$

$$\frac{1}{2} \begin{bmatrix} \Delta x \\ \Delta u \end{bmatrix}^T \begin{bmatrix} G_{xx} & G_{xu} \\ G_{ux} & G_{uu} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta u \end{bmatrix} \text{ if } (G_{xu} = G_{ux}^T) \quad (98)$$

为什么我们有动作价值函数（在强化学习中是Q函数）？因为Q函数是动态规划在回溯过程中的方程。

根据贝尔曼原理，我们可以从 V_N 向后计算到 V_{N-1} ：

$$V_{N-1}(x + \Delta x) := \min_{\Delta u} \left[S(x, u) + g_x^T \Delta x + g_u^T \Delta u + \frac{1}{2} \Delta x^T G_{xx} \Delta x \right] \quad (99)$$

$$+ \frac{1}{2} \Delta u^T G_{uu} \Delta u + \frac{1}{2} \Delta x^T G_{xu} \Delta u + \frac{1}{2} \Delta u^T G_{ux} \Delta x \quad (100)$$

对控制输入 Δu 求导以计算最优控制输入：

$$\nabla_u V_{N-1}(x + \Delta x) = g_u + G_{uu} \Delta u + G_{ux} \Delta x = 0 \quad (101)$$

$$\Rightarrow \Delta u_{N-1} = -G_{uu}^{-1} g_u - G_{uu}^{-1} G_{ux} \Delta x = -d_{N-1} - K_{N-1} \Delta x \quad (102)$$

可以看到，除了反馈项 K ，它还有一个前馈项 d 。⁷ 这是因为原始LQR问题是在原点求解，而现在我们想在任意初始状态下解决这个问题。在使用Riccati方程解LQR的过程中，我们还可以将最优输入项插入回 V_{N-1} 中并获得新的 $V(x)$ ，用于迭代：

$$V_{N-1}(x + \Delta x) = V_{N-1}(x) + g_x^T \Delta x + g_u^T (-d_{N-1} - K_{N-1} \Delta x) + \dots \quad (103)$$

$$\frac{1}{2} \Delta x^T G_{xx} \Delta x + \frac{1}{2} (d_{N-1} + K_{N-1} \Delta x)^T G_{uu} (d_{N-1} + K_{N-1} \Delta x) - \dots \quad (104)$$

$$\frac{1}{2} \Delta x^T G_{xu} (d_{N-1} + K_{N-1} \Delta x) - \frac{1}{2} (d_{N-1} + K_{N-1} \Delta x)^T G_{ux} \Delta x \quad (105)$$

适当整理后，我们得到与”Riccati”方程相同的结构：

$$P_{N-1} = G_{xx} + K_{N-1}^T G_{uu} K_{N-1} - G_{xu} K_{N-1} - K_{N-1}^T G_{ux} \quad (106)$$

$$p_{N-1} = g_x - K_{N-1}^T g_u + K_{N-1}^T G_{uu} d_{N-1} - G_{xu} d_{N-1} \quad (107)$$

则有：

$$V_{N-1}(x + \Delta x) = V_{N-1}(x) + p_{N-1}^T \Delta x + \frac{1}{2} \Delta x^T P_{N-1} \Delta x \quad (108)$$

该公式表示对函数 $V_{N-1}(x)$ 在点 x 处的二阶泰勒展开，其中 p_{N-1} 是梯度， P_{N-1} 是海森矩阵。用于近似状态扰动 Δx 下的值函数变化。矩阵偏导计算需要参考??节。根据这些数学技巧，可以推导stage-action函数 $S(x, u)$ 的雅可比矩阵：

$$S_k(x, u) = l_k(x, u) + V_{k+1}(f(x, u)) \quad (109)$$

$$\begin{aligned} \frac{\partial S}{\partial x} &= \frac{\partial l}{\partial x} + \frac{\partial V}{\partial x} \frac{\partial f}{\partial x} \\ \frac{\partial S}{\partial u} &= \frac{\partial l}{\partial u} + \frac{\partial V}{\partial x} \frac{\partial f}{\partial u} \end{aligned}$$

$$G_{xx} = \frac{\partial g_x}{\partial x} = \nabla_{xx}^2 l(x, u) + A_k^T \nabla^2 V_{k+1} A_k + \left(\frac{\partial p_{k+1}^T}{\partial x} \otimes I \right) \frac{\partial A_k}{\partial x}$$

$$G_{uu} = \frac{\partial g_u}{\partial u} = \nabla_{uu}^2 l(x, u) + B_k^T \nabla^2 V_{k+1} B_k + \left(\frac{\partial p_{k+1}^T}{\partial x} \otimes I \right) \frac{\partial B_k}{\partial u}$$

$$G_{xu} = \frac{\partial g_x}{\partial u} = \nabla_{xu}^2 l(x, u) + A_k^T \nabla^2 V_{k+1} B_k + \left(\frac{\partial p_{k+1}^T}{\partial x} \otimes I \right) \frac{\partial B_k}{\partial x}$$

G_{xx} 正定性分析：

从前面的推导可以看到， G_{xx} 的表达式为：

$$G_{xx} = \nabla_{xx}^2 l(x, u) + A_k^T \nabla^2 V_{k+1} A_k + \left(\frac{\partial p_{k+1}^T}{\partial x} \otimes I \right) \frac{\partial A_k}{\partial x}$$

7

- 前馈（feedforward）并不是说它作用在“前面”的时刻，而是说它“预先”计算，只依赖于参考/目标，不依赖于实际偏差。
- 反馈（feedback）并不是说它作用在“后面”的时刻，而是说它“事后根据实际偏差做调整”，依赖于当前的状态与目标之间的误差。

其中各项的正定性分析如下：

- **第一项 $\nabla_{xx}^2 l(x, u)$** : 这是代价函数的Hessian矩阵。若代价函数为凸函数（如二次型），则此项半正定。对于常见的二次代价函数 $l(x, u) = \frac{1}{2}(x - x_{ref})^T Q(x - x_{ref}) + \frac{1}{2}u^T R u$, 有 $\nabla_{xx}^2 l = Q \succeq 0$ 。
- **第二项 $A_k^T \nabla^2 V_{k+1} A_k$** : 由于 $\nabla^2 V_{k+1} = P_{k+1}$ 在算法递推中保持半正定（这是Riccati递推的性质），且对于任意矩阵 A_k ，二次型 $A_k^T P_{k+1} A_k$ 也保持半正定。
- **第三项（张量项）**: $\left(\frac{\partial p_{k+1}^T}{\partial x} \otimes I\right) \frac{\partial A_k}{\partial x}$ 是包含动力学二阶导数的张量项，其符号不定。

正定性保证的策略：

1. **iLQR方法**: 忽略张量项，仅保留前两项，确保 $G_{xx} \succeq 0$:

$$G_{xx}^{\text{iLQR}} = \nabla_{xx}^2 l(x, u) + A_k^T P_{k+1} A_k \succeq 0$$

2. **DDP方法**: 保留所有项，但需要正则化处理:

$$G_{xx}^{\text{DDP}} = G_{xx} + \lambda I$$

其中 $\lambda > 0$ 是正则化参数，确保矩阵正定。

3. **自适应正则化**: 根据 G_{xx} 的最小特征值动态调整 λ :

$$\lambda = \max(0, -\lambda_{\min}(G_{xx}) + \epsilon)$$

其中 $\epsilon > 0$ 是小的正数。

注意： G 的最后一项为张量项，若忽略该项，则称为iLQR，否则为DDP。该处理方式与牛顿法和高斯-牛顿法的关系类似。若保留张量项，则Hessian矩阵可能不正定，因此需要进行正则化以保证DDP算法收敛到局部极小值；若去掉张量项，则iLQR方式的Hessian矩阵必定为半正定，可以不进行正则化（但实际迭代过程中存在数值舍入误差，建议仍进行正则化处理）。

Acrobot双摆的DDP/iLQR 非线性轨迹优化实例

1.问题描述 Acrobot系统是一个典型的欠驱动双摆系统，其动力学具有高度非线性。系统状态通常为 $x = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]^T$ ，其中 θ_1 和 θ_2 分别为两个关节的角度， $\dot{\theta}_1$ 和 $\dot{\theta}_2$ 为角速度。控制输入 u 仅作用于第二个关节。

目标：从初始悬垂状态（如 $\theta_1 = 0, \theta_2 = 0, \dot{\theta}_1 = 0, \dot{\theta}_2 = 0$ ）控制到倒立目标状态（如 $\theta_1 = \pi, \theta_2 = 0, \dot{\theta}_1 = 0, \dot{\theta}_2 = 0$ ）。

优化问题形式：

$$\begin{aligned} \min_{\{u_k\}_{k=1}^{N-1}} \quad & J = \sum_{k=1}^{N-1} l(x_k, u_k) + l_N(x_N) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\ & u_{\min} \leq u_k \leq u_{\max} \end{aligned}$$

其中 $l(x_k, u_k)$ 为阶段代价（如状态误差和控制能量）， $l_N(x_N)$ 为终端代价（如与目标状态的距离）， $f(x_k, u_k)$ 为 Acrobot 的离散动力学。

约束：仅第二关节可控，输入受限，状态可加软约束。

求解方法：采用 DDP/iLQR 算法，通过对动力学和代价函数的二阶近似，迭代优化控制序列，实现从悬垂到倒立的轨迹生成与最优控制。

2. 动态系统 内容对于学过机器人的同学来说应该不陌生，如没了解过机器人学，可以参考《现代机器人学：机构、规划与控制》动力学一章忽略掉旋量系统不影响理解。这里我们简单总结一下 Acrobot 系统的连续时间动力学可用如下微分方程描述：

$$\dot{x} = f(x, u) = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix}$$

其中，状态 $x = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]^T$ ，控制输入 $u = [\tau_2]$ （仅第二关节有驱动力矩）。动力学方程由如下结构组成：

$$M(\theta) \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} = \begin{bmatrix} 0 \\ \tau_2 \end{bmatrix} - B(\theta, \dot{\theta}) - G(\theta) - C(\dot{\theta})$$

其中：

- $M(\theta)$: 质量（惯性）矩阵，依赖于关节角度
- $B(\theta, \dot{\theta})$: 科里奥利力和离心力项
- $G(\theta)$: 重力项
- $C(\dot{\theta})$: 摩擦项

各项具体表达式如下：

质量矩阵：

$$M(\theta) = \begin{bmatrix} m_1 l_1^2 + J_1 + m_2(l_1^2 + l_2^2 + 2l_1 l_2 \cos \theta_2) + J_2 & m_2(l_2^2 + l_1 l_2 \cos \theta_2) + J_2 \\ m_2(l_2^2 + l_1 l_2 \cos \theta_2) + J_2 & m_2 l_2^2 + J_2 \end{bmatrix}$$

科里奥利/离心力项：

$$B(\theta, \dot{\theta}) = \begin{bmatrix} -(2\dot{\theta}_1 \dot{\theta}_2 + \dot{\theta}_2^2) m_2 l_1 l_2 \sin \theta_2 \\ m_2 l_1 l_2 \sin \theta_2 \dot{\theta}_1^2 \end{bmatrix}$$

摩擦项（粘性摩擦）：

$$C(\dot{\theta}) = \begin{bmatrix} c\dot{\theta}_1 \\ c\dot{\theta}_2 \end{bmatrix}$$

重力项:

$$G(\theta) = \begin{bmatrix} [(m_1 + m_2)l_1 \cos \theta_1 + m_2 l_2 \cos(\theta_1 + \theta_2)]g \\ m_2 l_2 \cos(\theta_1 + \theta_2)g \end{bmatrix}$$

输入力矩:

$$\tau = \begin{bmatrix} 0 \\ \tau_2 \end{bmatrix}$$

最终状态空间表达式:

$$\dot{x} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ M(\theta)^{-1} (\tau - B(\theta, \dot{\theta}) - G(\theta) - C(\dot{\theta})) \end{bmatrix}$$

上述动力学模型高度非线性，体现了欠驱动系统的复杂性，适用于DDP/iLQR等非线性轨迹优化算法的研究与实现。acrobot dynamics函数的输入是 $((\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2)^T, (0, \tau_2)^T)$ ，其输出为 $\dot{\theta}_1, \dot{\theta}_2, \ddot{\theta}_1, \ddot{\theta}_2$ 。

3. 数值积分 这里使用RK4方法进行离散化。前面已经讨论过，这里简单列举下公式。四阶龙格-库塔法（RK4）公式:

$$\begin{aligned} k_1 &= f(x_k, u_k) \\ k_2 &= f\left(x_k + \frac{h}{2}k_1, u_k\right) \\ k_3 &= f\left(x_k + \frac{h}{2}k_2, u_k\right) \\ k_4 &= f(x_k + h k_3, u_k) \\ x_{k+1} &= x_k + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}$$

4. 数值微分 在DDP/iLQR算法中，需要对动力学离散映射 $f(x, u)$ 分别对状态 x 和控制 u 求雅可比矩阵，即

$$A = \frac{\partial f}{\partial x}, \quad B = \frac{\partial f}{\partial u}$$

对于复杂的非线性动力学，解析求导往往较为繁琐，因此常用自动微分（Automatic Differentiation, AD）工具进行数值计算。

以Julia语言为例，利用ForwardDiff库，可以通过如下方式实现： - 对状态 x 求

雅可比矩阵：

$$A = \frac{\partial f}{\partial x} \approx \text{ForwardDiff.jacobian}(x \mapsto f(x, u), x)$$

- 对控制 u 求雅可比矩阵（若 u 为标量，则为导数）：

$$B = \frac{\partial f}{\partial u} \approx \text{ForwardDiff.jacobian}(u \mapsto f(x, u), u)$$

其中 $f(x, u)$ 可为如RK4离散化后的动力学映射。

自动微分方法相比有限差分具有更高的精度和效率，适用于高维系统的雅可比矩阵计算，是现代轨迹优化实现中的常用技术手段。

5. cost QP

- 阶段代价 (stage cost)：

$$l(x_k, u_k) = \frac{1}{2}(x_k - x_{\text{goal}})^T Q(x_k - x_{\text{goal}}) + \frac{1}{2} R u_k^2$$

其中

- $Q = \text{diag}(1, 1, 0.1, 0.1)$: 位置权重大，速度权重小
- $R = 0.01$: 控制输入权重（能耗惩罚）

- 终端代价 (terminal cost)：

$$l_N(x_N) = \frac{1}{2}(x_N - x_{\text{goal}})^T Q_N(x_N - x_{\text{goal}})$$

其中 $Q_N = 100 \cdot I$ ，为终端状态权重矩阵，通常取较大值以强化终端约束。

- 总代价函数：

$$J = \sum_{k=1}^{N-1} l(x_k, u_k) + l_N(x_N)$$

其中 $l(x_k, u_k)$ 和 $l_N(x_N)$ 分别如上定义， x_{goal} 为目标状态。

6. 初始、目标状态并初始化轨迹 初始状态与目标状态:

$$x_0 = \begin{bmatrix} -\frac{\pi}{2} \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (\text{初始: 悬垂位置})$$

$$x_{\text{goal}} = \begin{bmatrix} \frac{\pi}{2} \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (\text{目标: 倒立位置})$$

初始轨迹猜测:

令 N 为离散时间步数, 初始状态轨迹 $\{x_k\}$ 可设为

$$x_k = x_0, \quad k = 1, \dots, N$$

即将初始状态复制 N 次作为初始轨迹。

控制序列 $\{u_k\}$ 可随机初始化:

$$u_k \sim \mathcal{N}(0, 1), \quad k = 1, \dots, N - 1$$

初始前向积分:

根据初始控制序列, 利用RK4方法递推计算状态轨迹:

$$x_{k+1} = f_{\text{RK4}}(x_k, u_k), \quad k = 1, \dots, N - 1$$

其中 f_{RK4} 为 RK4 离散动力学映射。

初始总代价:

$$J = \sum_{k=1}^{N-1} l(x_k, u_k) + l_N(x_N)$$

其中 $l(x_k, u_k)$ 和 $l_N(x_N)$ 如前所述。

7. DDP/iLQR算法 DDP/iLQR 算法的核心思想是通过二阶泰勒展开和 Riccati 方程迭代优化控制序列。以下是算法的数学流程:

算法步骤说明:

- **后向传播 (Backward Pass):** 从终端状态向前递推, 利用当前轨迹和控制, 计算每一步的反馈增益 K_k 和前馈项 d_k , 并更新 Riccati 变量 p_k, P_k 。这一步本质上是对二次近似问题的解析求解。
- **前向传播 (Forward Pass):** 用更新后的控制律 (包含前馈和反馈项) 生成新的控制序列, 并用 RK4 积分得到新的状态轨迹。通过线搜索调整步长 α , 确保代价函数下降。
- **收敛判据:** 判断前馈项 d_k 是否足够小或总代价 J 是否收敛, 决定是否终止迭代。

- 雅可比与Hessian计算: A_k, B_k 为动力学对状态和控制的雅可比矩阵, Q_k, R_k 为代价函数的二阶导数 (Hessian), 实际实现中可用自动微分工具获得。

Algorithm 7: DDP/iLQR算法数学流程 (主流程)

Input: 初始状态 x_1 , 初始控制序列 $\{u_k\}_{k=1}^{N-1}$, 收敛阈值 ϵ

Output: 最优控制序列 $\{u_k^*\}$, 最优状态轨迹 $\{x_k^*\}$

1. 终端条件初始化:

$$p_N = Q_N(x_N - x_{goal}), \quad P_N = Q_N$$

while 未收敛 **do**

2. 后向传播 (Backward Pass):

调用Algorithm 8完成后向传播, 计算 d_k, K_k, p_k, P_k 的序列 (即控制律和Riccati变量的轨迹)。

3. 前向传播与线搜索 (Forward Pass & Line Search):

调用Algorithm 9完成前向传播与线搜索, 更新轨迹 $\{u_k\}, \{x_k\}$ 。

4. 收敛判据:

if $\max_k |d_k| < \epsilon$ 或 J 收敛 **then**

break;

// 满足收敛条件则终止

5. 迭代信息输出 (可选)

Algorithm 8: DDP/iLQR算法后向传播 (Backward Pass)

for $k = N - 1$ **to** 1 **do**

 计算cost function函数一阶导数

$$q_k = Q(x_k - x_{goal}), \quad r_k = Ru_k; \quad // \frac{\partial l}{\partial x}, \frac{\partial l}{\partial u}$$

 计算system dynamic雅可比

$$A_k = \frac{\partial f}{\partial x}(x_k, u_k), \quad B_k = \frac{\partial f}{\partial u}(x_k, u_k); \quad // \frac{\partial f}{\partial x}, \frac{\partial f}{\partial u}$$

 计算stage-value function一阶导

$$g_x = q_k + A_k^\top p_{k+1}, \quad g_u = r_k + B_k^\top p_{k+1}; \quad // \frac{\partial S}{\partial x}, \frac{\partial S}{\partial u}$$

 计算stage-value function二阶导

$$G_{xx} = Q_k + A_k^\top P_{k+1} A_k; \quad // \frac{\partial^2 S}{\partial x^2}$$

$$G_{uu} = R_k + B_k^\top P_{k+1} B_k; \quad // \frac{\partial^2 S}{\partial u^2}$$

$$G_{xu} = A_k^\top P_{k+1} B_k, \quad G_{ux} = B_k^\top P_{k+1} A_k; \quad // \frac{\partial^2 S}{\partial x \partial u}, \frac{\partial^2 S}{\partial u \partial x}$$

 计算最优控制律

$$d_k = G_{uu}^{-1} g_u; \quad // \text{前馈项}$$

$$K_k = G_{uu}^{-1} G_{ux}; \quad // \text{反馈增益}$$

 Riccati变量递推

$$p_k = g_x - K_k^\top g_u + K_k^\top G_{uu} d_k - G_{xu} d_k; \quad // \text{一阶项递推}$$

$$P_k = G_{xx} + K_k^\top G_{uu} K_k - G_{xu} K_k - K_k^\top G_{ux}; \quad // \text{二阶项递推}$$

 记录代价变化

$$\Delta J += g_u^\top d_k; \quad // \text{累计代价下降量}$$

^aS即state action function (动作-价值函数) $S_k(x_k, u_k) = l_k(x_k, u_k) + V_{k+1}(f(x_k, u_k))$, 它在动态规划中对应于Q函数, 表示在当前状态 x_k 和控制 u_k 下的瞬时代价与后续最优代价之和。 g_x 和 g_u 分别为S对 x 和 u 的梯度, 用于后向传播递推Riccati变量。

^b $V_k(x)$ 为从时刻 k 状态 x 出发到终点的最小累计代价 (即cost-to-go function), 在动态规划中递归定义: $V_k(x) = \min_{u_k} [l_k(x, u_k) + V_{k+1}(f(x, u_k))]$ 。

Algorithm 9: DDP/iLQR算法前向传播与线搜索 (Forward Pass & Line Search) 使用RK4, Armijo方法

```

设 $\alpha \leftarrow 1$ 
repeat
    for  $k = 1$  to  $N - 1$  do
         $u_k^{\text{new}} = u_k - \alpha d_k - K_k(x_k^{\text{new}} - x_k)$ ;           // 更新控制输入
         $x_{k+1}^{\text{new}} = f_{\text{RK4}}(x_k^{\text{new}}, u_k^{\text{new}})$ ;           // RK4积分得到新状态
        计算 $J_{\text{new}}$ ;           // 计算新总代价
        ;
        if  $J_{\text{new}}$ 未满足下降条件 then
             $\alpha \leftarrow 0.5\alpha$ ;           // 步长减半, 继续线搜索
    until 满足充分下降;

```

注: 实际实现时, 需注意数值稳定性 (如 G_{uu} 奇异时可加正则项), 以及约束处理 (如输入饱和、软约束等)。

8.1.2 DDP/iLQR算法流程

1. **初始化:** 给定初始状态 x_1 , 初始控制序列 $\{u_k\}_{k=1}^{N-1}$ 。
2. **前向传播 (Forward Rollout):** 根据当前控制序列, 利用系统动力学递推计算状态轨迹 $\{x_k\}_{k=1}^N$ 。
3. **反向传播 (Backward Pass):** 从终点 $k = N$ 向前递推, 利用泰勒展开和Riccati方程计算反馈增益 K_k 和前馈项 d_k , 并更新 P_k, p_k 。
4. **控制更新:** 用线性反馈律更新控制输入:

$$u_k^{\text{new}} = u_k + d_k + K_k(x_k^{\text{new}} - x_k)$$

5. **线搜索 (Line Search):** 对步长 α 进行线搜索, 选择使代价函数下降的合适步长。
6. **收敛判据:** 若代价函数收敛或最大迭代次数达到, 算法终止; 否则返回第2步。

8.1.3 DDP与iLQR的区别

- DDP在反向传播时保留了二阶项 (Hessian的张量项), 因此理论上收敛速度更快, 但计算量更大。
- iLQR忽略了二阶张量项, 仅保留高斯-牛顿近似, 计算更高效, 适用于大规模系统。
- DDP对数值条件更敏感, iLQR更稳定。

8.1.4 优缺点总结

优点:

- 能高效求解大规模非线性轨迹优化问题。

- 充分利用动态规划结构，收敛速度快。
- 生成的控制律具有反馈结构，鲁棒性好。
- DDP在收敛的时候，能提供一个TVLQR（Time-Varying Linear Quadratic Regulator）控制律，适用于在线控制，因为Q和R可以在线更新。
- 基于DP的方法，求出的轨迹更接近全局最优解。

缺点：

- 不原生支持约束处理，需结合其他方法（如罚函数、障碍函数等）。
- 对初始轨迹依赖较大，易陷入局部最优。（后面说的基于采样的控制器能解决这个）
- 因为Riccati的 backward pass在数值上是不稳定的，长时间轨迹易出现数值不稳定，浮点数累积错误。
- 遇到病态问题时（如Hessian矩阵接近奇异、A或B矩阵有一个特别大、动态条件数特别多），可能导致收敛失败。

8.1.5 约束处理方法简介

在实际轨迹优化问题中，常常需要处理输入约束（如力/力矩限制）、状态约束（如障碍物、速度限制）等。常见的约束处理方法包括：

- “**压缩函数**”法（**Squashing Function**）：例如通过tanh等函数将控制输入压缩到允许范围内，简单但会引入非线性，可能导致收敛变慢。

$$u_k = u_{\max} \cdot \tanh(\tilde{u}_k)$$

其中 \tilde{u}_k 为未约束的优化变量。

- **罚函数法（Penalty Method）**：将约束违背程度作为罚项加入到目标函数中，适用于软约束，但可能导致病态优化问题。

$$J_{\text{penalty}} = J + \rho \sum_k (\max(0, u_k - u_{\max})^2 + \max(0, u_{\min} - u_k)^2)$$

其中 ρ 为罚因子。

- **障碍函数法（Barrier Method）**：在约束边界附近引入无穷大代价，防止解越界，适合严格约束，但数值实现需谨慎。

$$J_{\text{barrier}} = J - \mu \sum_k (\log(u_k - u_{\min}) + \log(u_{\max} - u_k))$$

其中 μ 为障碍参数。

- **增广拉格朗日法（Augmented Lagrangian）**：将拉格朗日乘子和二次罚项结合，适合处理等式和不等式约束，收敛性较好。

$$\mathcal{L}(x, u, \lambda) = J + \lambda^T c(x, u) + \frac{\rho}{2} \|c(x, u)\|^2$$

其中 $c(x, u)$ 为约束函数， λ 为拉格朗日乘子。

- **盒约束QP (Box-constrained QP):** 在反向传播阶段直接对 Δu 做盒约束优化，适合输入约束。

$$\min_{\Delta u} \quad \frac{1}{2} \Delta u^T G_{uu} \Delta u + g_u^T \Delta u \quad \text{s.t.} \quad u_{\min} \leq u_k + \Delta u \leq u_{\max}$$

注意：状态约束通常更难处理，常见做法是将其转化为软约束加入目标函数，或采用投影等方法。

8.1.6 最小时间问题处理

最小时间问题的目标是使系统在最短时间内到达目标状态。其优化形式为：

$$\min_{x(t), u(t), T} \quad J = \int_0^T 1 dt \quad (110)$$

$$\text{s.t.} \quad \dot{x} = f(x, u) \quad (111)$$

$$x(T) = x_{\text{goal}} \quad (112)$$

$$U_{\min} \leq u(t) \leq U_{\max} \quad (113)$$

常用处理方法包括：

- 将时间步长 h 作为控制变量：通过调整每步的 h_k ，间接优化终止时间 T 。
- 在RK离散化中引入 h_k ：如 $x_{k+1} = f(x_k, u_k, h_k)$ ，并将 h_k 加入优化变量。
- 在目标函数中加权 h_k ：如 $J(x, u, h) = \sum_{k=1}^{N-1} h_k l_k(x_k, u_k) + l_N(x_N)$ 。
- 对 h_k 加约束：防止 h_k 过大或为负，避免“作弊”物理规律。

最小时间问题通常是非线性/非凸优化，即使动力学是线性的。因为融入了 h 的优化，导致问题变为非线性规划 (NLP)。因此，DDP/iLQR等方法仍然适用，但需要对时间步长进行特殊处理

8.1.7 小结

DDP/iLQR为非线性轨迹优化提供了高效的数值方法，但在约束处理和特殊目标（如最短时间）问题上需结合其他优化技术。实际应用中需根据问题特点选择合适的约束处理方式和算法变体。

8.1.8 ALTRo 方法简介

论文信息：

- 标题：ALTRo: A Fast Solver for Constrained Trajectory Optimization
- 作者：Taylor A. Howell, Brian E. Jackson, Zachary Manchester
- 机构：Stanford University
- 代码链接：<https://github.com/RoboticExplorationLab/TrajectoryOptimization.jl>

问题形式化 轨迹优化问题定义如下：

$$\min_{x_0^N, u_0^{N-1}, \Delta t} \ell_N(x_N) + \sum_{k=0}^{N-1} \ell_k(x_k, u_k, \Delta t) \quad (114)$$

$$\text{s.t. } x_{k+1} = f(x_k, u_k, \Delta t) \quad (115)$$

$$g_k(x_k, u_k) \leq 0 \quad (116)$$

$$h_k(x_k, u_k) = 0 \quad (117)$$

增广拉格朗日 iLQR (AL-iLQR) ALTRO 使用 iLQR 作为内核，并通过增广拉格朗日法引入约束处理：

$$\mathcal{L}_A = \ell_N(x_N) + \sum_{k=0}^{N-1} \ell_k(x_k, u_k) + \left(\lambda_k + \frac{1}{2} I_{\mu_k} c_k(x_k, u_k) \right)^T c_k(x_k, u_k) \quad (118)$$

其中：

- λ_k : 拉格朗日乘子
- μ_k : 惩罚因子
- c_k : 联合约束函数（包含不等式 g_k 与等式 h_k ）

乘子更新规则：

$$\lambda_{ki}^+ = \begin{cases} \lambda_{ki} + \mu_k c_{ki}, & i \in \mathcal{E} \\ \max(0, \lambda_{ki} + \mu_k c_{ki}), & i \in \mathcal{I} \end{cases} \quad (119)$$

数值稳定性：平方根形式 为提高数值鲁棒性，引入平方根卡尔曼滤波器思想，通过 QR 或 Cholesky 分解处理：

$$S_N = \text{QR} \left(\left(\sqrt{\ell_{xx}^N}, \sqrt{I_{\mu_N}} c_{xN} \right) \right) \quad (120)$$

$$Z_{xx} = \text{QR} \left(\left(\sqrt{\ell_{xx}}, S'A, \sqrt{I_\mu} c_x \right) \right) \quad (121)$$

$$Z_{uu} = \text{QR} \left(\left(\sqrt{\ell_{uu}}, S'B, \sqrt{I_\mu} c_u, \sqrt{\rho I} \right) \right) \quad (122)$$

不可行轨迹初始化 引入松弛变量 s_k ，修改动力学为：

$$x_{k+1} = f(x_k, u_k) + s_k \quad (123)$$

并添加惩罚项：

$$\sum_{k=0}^{N-1} \frac{1}{2} s_k^T R_s s_k \quad \text{且约束 } s_k = 0 \quad (124)$$

时间惩罚问题 引入时间变量 $\tau_k = \sqrt{\Delta t_k}$:

$$\begin{bmatrix} x_{k+1} \\ \delta_{k+1} \end{bmatrix} = \begin{bmatrix} f(x_k, u_k, \tau_k) \\ \tau_k \end{bmatrix} \quad (125)$$

$$\sum_k R_\tau \tau_k^2 \quad \text{且约束 } \delta_k = \tau_k \quad (126)$$

主动集投影 使用 Hessian 加权投影求解以提升约束精度:

$$S = \sqrt{DH^{-1}D^T} \quad (127)$$

$$\delta Y = H^{-1}D^T(S^{-1}S^{-T}d) \quad (128)$$

其中 D 为活跃约束的雅可比矩阵, d 为残差向量。

实验摘要

- 系统包括: Block Move, Pendulum, Acrobot, Cartpole, Reeds-Shepp Car, Kuka 机械臂, 四旋翼等;
- ALTR0 比 Ipopt/SNOPT 更快且能处理更复杂的约束;
- 支持不可行轨迹初始化;

总结与展望 ALTR0 结合 DDP 的高效性与直接方法的鲁棒性。未来方向包括并行化、使用 ADMM 和解决双层优化问题等。

8.2 直接轨迹优化

8.2.1 序列二次规划 (SQP)

SQP (Sequential Quadratic Programming) 是一种用于求解非线性优化 (NLP) 问题的迭代方法。可以处理任何程度的非线性，包括约束中的非线性。但其主要缺点是，该方法包含多个导数，这些导数可能需要在迭代求解之前进行解析计算，因此，对于包含许多变量或约束的大型问题时，SQP会变得相当繁琐。

定义 8.1 (SQP问题). SQP问题是一个迭代优化问题，形式如下：

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad (129)$$

$$\text{s.t. } h_i(\mathbf{x}) = 0, \quad i = 1, \dots, m \quad (130)$$

$$g_j(\mathbf{x}) \leq 0, \quad j = 1, \dots, p \quad (131)$$

其中， f 是目标函数， h_i 是等式约束， g_j 是不等式约束。

拉格朗日函数和KKT条件 对于上式的拉格朗日和KKT条件为：拉格朗日函数将所有约束的信息组合成一个函数，对等式约束引入 λ ，对不等式约束引入 μ ：

$$L(x, \lambda, \mu) = f(x) + \sum_i \lambda_i h_i(x) + \sum_i \mu_i g_i(x) \quad (132)$$

KKT条件可写为：

$$\nabla L = \begin{bmatrix} \frac{dL}{dx} \\ \frac{dL}{d\lambda} \\ \frac{dL}{d\mu} \end{bmatrix} = \begin{bmatrix} \nabla f + \lambda \nabla h + \mu \nabla g^* \\ h \\ g^* \end{bmatrix} = 0 \quad (133)$$

其中 g^* 表示有效（活跃）不等式约束。

第二个KKT条件仅是可行性条件， $h(x)$ 被限制为零。第三个KKT条件表示只有有效的不等式约束需要满足该等式，非活动约束的 μ 为零。

SQP两个基本方法 SQP结合了两种用于解决NLP问题的基本方法，前面章节我们已经讨论过，这里简要概述一下：

- **有效集方法 (Active Set Method):** 有效集方法通过猜测和检查来求解KKT条件，以找到临界点。首先猜测哪些不等式约束是有效的（即等式成立），然后将这些约束视为等式约束，求解相应的子问题。如果某个约束的拉格朗日乘子为负，则将其移出有效集；如果某个不等式约束在当前解处被激活，则将其加入有效集。该方法适用于约束数量较少或结构较简单的问题，但对于大规模问题，因KKT条件的线性性限制，效率较低。
- **牛顿法 (Newton's Method):** 牛顿法的核心思想是利用目标函数的二阶导数信息，通过泰勒展开近似目标函数，并迭代更新变量以快速收敛到临界点。每一步迭代形式为：

$$x_{k+1} = x_k - \frac{\nabla f}{\nabla^2 f} \quad (134)$$

牛顿法在接近最优解时具有二次收敛速度，但需要计算和存储Hessian矩阵，且每步需解线性方程组。对于非凸问题，Hessian可能不是正定的，此时需结合修正策略或信赖域方法。

SQP算法 目标函数的临界点也积极拉格朗日函数的临界点。反之亦然，因为拉格朗日函数储存 KKT 条件于目标函数：所有这些都多余，复杂无效。因此，该算法只是简单地迭代牛顿调优求拉格朗日函数的临界点。由于拉格朗日函数提供了改变，因此该方法形成一个系统：

$$\begin{bmatrix} x_{k+1} \\ \lambda_{k+1} \\ \mu_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ \lambda_k \\ \mu_k \end{bmatrix} - (\nabla^2 L_k)^{-1} \nabla L_k \quad (135)$$

记忆： $\nabla L = \begin{bmatrix} \frac{dL}{dx} \\ \frac{dL}{dh} \\ \frac{dL}{d\mu} \end{bmatrix} = \begin{bmatrix} \nabla f + \lambda \nabla h + \mu \nabla g^* \\ h \\ g^* \end{bmatrix}$

然后 $\nabla^2 L = \begin{bmatrix} \nabla_{xx}^2 L & \nabla h & \nabla g \\ \nabla h & 0 & 0 \\ \nabla g & 0 & 0 \end{bmatrix}$

与有效集方法不同，无论目标函数导数的非线性程度如何，SQP 都完全无需求解线性问题。理论上，如果线性方案适度可以解析地提取出来，然后进行编码，那么软件就可以非常快速地迭代。因为系统不会发生变化。然而，在实践中，数据很可能不是可分定的，因而变量成为受主下线性路径。因此，牛顿法可能的改动方向”p”通常从预标定的方式求解：使用二次优化求解二次最小化问题。该方程解释如下：

$$p = -\frac{\nabla L}{\nabla^2 L} = -\frac{(\nabla L)p}{(\nabla^2 L)p} \quad (136)$$

由于 p 是目标函数的增量变化，因此该方程类似于目标函数导数的两项泰勒级数，这意味着 p 方变更的条件等效于牛顿迭代。分解这方程组的不同方程，并将二阶项假设以符合泰勒级数观念，即可得到一个最小化问题。该问题是二次项数，因此必须用非线性优化方法求解，这意味着将牛顿法解线性问题中的需要引入算法中，包含个可预期的单变量问题即问题复杂解读式。

$$\min_p f_k(x) + \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 L_k p \quad (137)$$

$$\text{s.t. } \nabla h_k p + h_k = 0 \quad (138)$$

$$\nabla g_k p + g_k = 0 \quad (139)$$

SQP实际案例 假设我们有一个简单的非线性优化问题：

$$\text{Minimize } f(\vec{x}) = x_1^3 + 2x_2^2 - 8x_1x_2 \quad (140)$$

$$\text{Subject to } h_1(\vec{x}) : x_2^2 - 1 = 0 \quad (141)$$

$$g_1(\vec{x}) : x_1x_2 - 4 \leq 0 \quad (142)$$

这里我们有等式约束，不等式约束和目标函数，他们都是非线性的。

①构建局部QP问题：假设初始点为 $\vec{x}_0 = [4, 2]$ ，评估目标函数和约束的梯度：

$$f(4, 2) = (4)^3 + 2 \cdot (2)^2 - 8 \cdot (4) \cdot (2) = 64 + 8 - 64 = 8 \text{ 检测约束条件：}$$

$$\text{等式约束: } h_1(4, 2) = (2)^2 - 1 = 3 \text{ (违反约束), } \nabla h_1 = \begin{bmatrix} 0 \\ 2x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$$

$$\text{不等式约束: } g_1(4, 2) = (4)(2) - 4 = 4 > 0 \text{ (违反约束), } \nabla g_1 = \begin{bmatrix} x_2 \\ x_1 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

构建局部QP问题目标函数梯度与Hessian：

$$\nabla f(\vec{x}) = \begin{bmatrix} 3x_1^2 - 8x_2 \\ 4x_2 - 8x_1 \end{bmatrix} \Rightarrow \nabla f(4, 2) = \begin{bmatrix} 3 \times 16 - 16 \\ 8 - 32 \end{bmatrix} = \begin{bmatrix} 32 \\ -24 \end{bmatrix}$$

$$\nabla_{xx}^2 f(\vec{x}) = \begin{bmatrix} 6x_1 & -8 \\ -8 & 4 \end{bmatrix} \Rightarrow \nabla_{xx}^2 f(4, 2) = \begin{bmatrix} 24 & -8 \\ -8 & 4 \end{bmatrix}$$

②局部二次规划 (QP) 问题展开：

我们将目标函数在点 $\vec{x}_0 = [4, 2]$ 处进行二阶泰勒展开，令 $\vec{d} = [d_1, d_2]^T$ 为步长变量：

$$\begin{aligned} f_L &= f(\vec{x}_0) + \nabla f(\vec{x}_0)^T \vec{d} \\ &= 8 + [32 \quad -24] \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \\ &= 8 + 32d_1 - 24d_2 \\ f_Q &= \frac{1}{2} \vec{d}^T \nabla_{xx}^2 f(\vec{x}_0) \vec{d} \\ &= \frac{1}{2} [d_1 \quad d_2] \begin{bmatrix} 24 & -8 \\ -8 & 4 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \\ &= \frac{1}{2} (24d_1^2 + 4d_2^2 - 16d_1d_2) \end{aligned}$$

因此，局部QP问题为：

$$\begin{aligned} \min_{\vec{d}} \quad & 8 + 32d_1 - 24d_2 + \frac{1}{2}(24d_1^2 + 4d_2^2 - 16d_1d_2) \\ \text{s.t.} \quad & h_1(\vec{x}_0) + \nabla h_1(\vec{x}_0)^T \vec{d} = 0 \\ & g_1(\vec{x}_0) + \nabla g_1(\vec{x}_0)^T \vec{d} \leq 0 \end{aligned}$$

将约束线性化：

$$h_1(\vec{x}_0) + \nabla h_1(\vec{x}_0)^T \vec{d} = 0 \implies 3 + 4d_2 = 0 \implies d_2 = -\frac{3}{4}$$

$$g_1(\vec{x}_0) + \nabla g_1(\vec{x}_0)^T \vec{d} \leq 0 \implies 4 + 2d_1 + 4d_2 \leq 0 \implies 2d_1 + 4d_2 \leq -4$$

最终，得到如下QP问题， Local Object Function:

$$\begin{aligned} \min_{d_1, d_2} \quad & 8 + 32d_1 - 24d_2 + 12d_1^2 + 2d_2^2 - 8d_1d_2 \\ \text{s.t.} \quad & d_2 = -\frac{3}{4} \\ & 2d_1 + 4d_2 \leq -4 \end{aligned}$$

这样就将原始非线性优化问题在当前点线性化/二次化，转化为一个标准的二次规划（QP）问题，可以用QP求解器直接求解。

③拉格朗日函数与KKT条件求解

我们可以进一步写出该QP问题的拉格朗日函数：

$$\begin{aligned} L(\vec{d}, U, \nu) = & 8 + 32d_1 - 24d_2 + 12d_1^2 + 2d_2^2 - 8d_1d_2 \\ & + U(4 + 2d_1 + 4d_2) + \nu(3 + 4d_2) \end{aligned}$$

对 d_1, d_2, U, ν 分别求偏导并令其为零，得到KKT条件，可以解得最优解：

$$d^* = \begin{bmatrix} d_1^* \\ d_2^* \end{bmatrix} = \vec{x}_0 = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$$

这说明通过KKT或牛顿法可以直接求得该QP问题的最优步长，从而更新原始变量，进入下一轮SQP迭代。

8.2.2 直接方法 (Direct Methods)

直接方法 (Direct Methods) 是轨迹优化中的一种重要方法，它通过将轨迹优化问题转化为一个非线性规划 (NLP) 问题来求解。直接方法的核心思想是将轨迹表示为一系列离散的控制输入和状态变量，然后通过优化这些变量来找到最优轨迹。

在之前的章节中，我们一直使用RK4等数值积分方法来显示地表示系统的动态方程：

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \rightarrow \mathbf{x}_{k+1} = \mathbf{x}_k + h f(\mathbf{x}_k, \mathbf{u}_k) \quad (143)$$

对于这种策略来说Rollout是必须的，例如在DDP、之前的Riccati迭代中，我们需要从初始状态 x_1 开始，不断地执行**最优控制Rollout**来得到优化后的整条轨迹。这近乎于一种MRF (Markov Random Field) 方法，时间序列之间满足马尔可夫性质。

对于直接方法，没有rollout这个过程，并不假设系统是马尔科夫的，而是将所有变量都放在一个大矩阵中进行迭代优化，也就是说， x_k, x_{k+1} 是同时存在的，这就需要一种显示数值积分的方法来解决。

我们首先寻找一个更加稳定的方法来表示系统约束方程，即找到这样一条系统约束：

直接方法的系统约束的一般形式：

$$c_k(x_k, u_k, x_{k+1}, u_{k+1}) = 0 \quad (144)$$

状态轨迹的近似 经典的直接方法使用cubic splines(三次样条插值)来近似状态轨迹

输入轨迹的近似 使用piecewise constant control inputs(分段常数控制输入)来近似输入轨迹, 即在每个时间段内, 控制输入变化率保持不变。

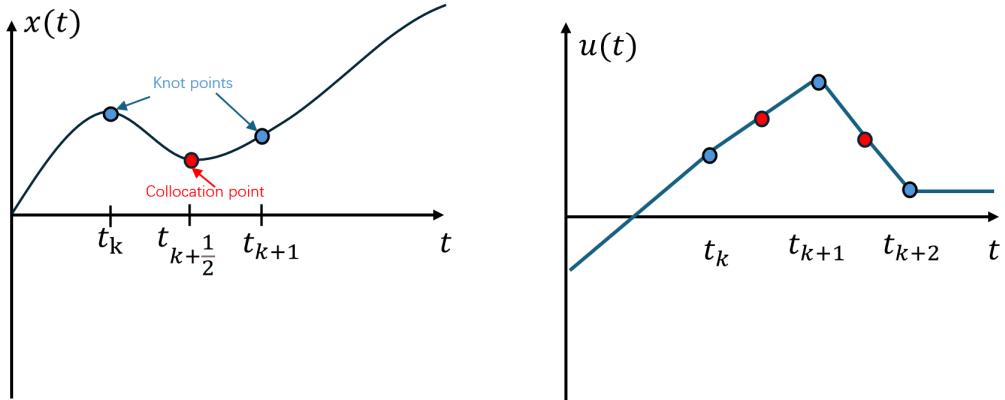


图 18: 直接法轨迹优化中的轨迹近似示意图。左图: 状态轨迹 $x(t)$ 通过三次样条插值(蓝色圆点为节点点, 红色圆点为配点)进行近似; 右图: 控制输入 $u(t)$ 采用分段常数或分段线性方式近似(蓝色圆点为节点点, 红色圆点为配点)。

三阶多项式轨迹插值 在三阶多项式的前提下, 轨迹可以完全由前后两点的一阶导数来确定。假设在区间 $[t_k, t_{k+1}]$ 上, 状态 $x(t)$ 用三阶多项式表示:

$$x(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 \quad (145)$$

$$\dot{x}(t) = c_1 + 2c_2 t + 3c_3 t^2 \quad (146)$$

令 $t_k = 0$, $t_{k+1} = h$, 则有

$$\begin{bmatrix} x_k \\ \dot{x}_k \\ x_{k+1} \\ \dot{x}_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & h & h^2 & h^3 \\ 0 & 1 & 2h & 3h^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad (147)$$

因此, 系数 \mathbf{c} 可由状态和导数唯一确定:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{3}{h^2} & -\frac{2}{h} & \frac{3}{h^2} & -\frac{1}{h} \\ \frac{2}{h^3} & \frac{1}{h^2} & -\frac{2}{h^3} & \frac{1}{h^2} \end{bmatrix} \begin{bmatrix} x_k \\ \dot{x}_k \\ x_{k+1} \\ \dot{x}_{k+1} \end{bmatrix} \quad (148)$$

带入 $t = \frac{h}{2}$, 得到中点处的状态表达式

$$x_{k+1/2} = \frac{1}{2}(x_k + x_{k+1}) + \frac{h}{8}(\dot{x}_k - \dot{x}_{k+1}) + \frac{h}{8}(f(x_k, u_k) - f(x_{k+1}, u_{k+1})), \quad (149)$$

$$\dot{x}_{k+1/2} = \frac{1}{2}(\dot{x}_k + \dot{x}_{k+1}) - \frac{3h}{2}(x_k - x_{k+1}) - \frac{h}{4}(f(x_k, u_k) + f(x_{k+1}, u_{k+1})), \quad (150)$$

$$u_{k+1/2} = \frac{1}{2}(u_k + u_{k+1}). \quad (151)$$

接着在中点处施加系统约束方程 $\dot{x}_{k+\frac{1}{2}} = f(x_{k+\frac{1}{2}}, u_{k+\frac{1}{2}})$, 得到前后状态的与输入约束:

$$\begin{aligned} c_i(x_k, u_k, x_{k+1}, u_{k+1}) &= f\left(x_{k+\frac{1}{2}}, u_{k+\frac{1}{2}}\right) \\ &\quad - \left(-\frac{3}{2}h(x_k - x_{k+1}) - \frac{1}{4}[f(x_k, u_k) + f(x_{k+1}, u_{k+1})] \right) \\ &= 0 \quad (152) \end{aligned}$$

综上所述, 即是Hermite-Simpson方法的核心思想。它通过在每个时间步内计算初始点、中点和终点的导数值, 来更精确地逼近状态轨迹。这种方法不仅提高了数值积分的精度, 还增强了对系统动态约束的处理能力。

Hermite-Simpson方法 在直接法中, Hermite-Simpson方法是一种常用的数值积分方法, 用于提高系统动态方程的近似精度。该方法通过在每个时间步中引入中点状态和中点控制输入, 结合三次插值多项式, 构造出更精确的系统约束。

Hermite-Simpson方法的公式如下:

$$y_{n+1} = y_n + \frac{h}{6} [k_1 + 4k_2 + k_3], \quad (153)$$

$$k_1 = f(x_n, y_n), \quad (154)$$

$$k_2 = f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right), \quad (155)$$

$$k_3 = f(x_n + h, y_n - hk_1 + 2hk_2). \quad (156)$$

在Hermite-Simpson方法中, k_1 表示初始点的导数值, k_2 表示中点的导数值, k_3 表示终点的导数值。通过这三个点的导数值, Hermite-Simpson方法能够在每个时间步内更精确地逼近状态轨迹。

这种方法的优点在于其高阶精度和稳定性, 特别适用于直接轨迹优化中的系统动态约束。通过将Hermite-Simpson方法引入到直接法的优化框架中, 可以显著提高轨迹优化的精度和收敛速度。

8.3 间接法与直接法的对比

如下:

一个Dircol的例子如下:

8.3.1 一个Dircol欠驱动双摆例子

1. 问题转换: 连续到离散 直接配点法的核心思想是将连续时间最优控制问题转化为有限维非线性规划问题。

定义 8.2 (连续时间最优控制问题). 原始的连续时间最优控制问题可以表述为:

$$\min_{x(t), u(t)} \int_0^T L(x(t), u(t)) dt \quad (157)$$

$$\text{s.t. } \dot{x}(t) = f(x(t), u(t)) \quad (158)$$

$$x(0) = x_0, \quad x(T) = x_{goal} \quad (159)$$

表 2: DDP/iLQR（间接法）与 Direct Collocation（直接法）比较

指标	DDP/iLQR（间接法）	Direct Collocation（直接法）
动力学可行性	在整个迭代过程中，即使未完全收敛到局部极小值，系统动力学是一直可行的，因此即使优化未完全收敛，也可以将优化的结果给机器人执行，利于实时控制	在迭代过程中，由于不存在前向rollout的过程，因此，只有当优化完全收敛后，优化的变量 x_k 、 u_k 才是满足系统动力学约束的，不利于实时控制
初始状态轨迹估计	也正是由于rollout过程会一直保持系统动力学可行，因此无法给一条粗糙初始状态轨迹，只能给关于控制率 u 的初始估计。因此不利于与一些粗糙的前端几何搜索或启发搜索结合（只能通过Cost方式结合）	必须给关于 x 和 u 的初始估计，可以通过一些启发搜索或者一些RRT方案来搜索一条粗糙的初始轨迹作为优化的初始值，利于在复杂的环境中探索（可以方便的让优化的局部极小值保持在期望的附近）
约束的处理	初始版本不能处理状态或输入上的约束。 u 的约束较好处理，但是 x 的约束一般来说要将整个DDP问题写成增广拉格朗日函数的形式来处理 x 约束，再结合一些active set的trick也可以做到处理约束。但是，由于增广拉格朗日法的一些数值问题，在约束很苛刻时，问题迭代会变得病态	可以方便地处理 x 、 u 约束，实际上是把问题抛给商用/开源的非线性求解器。底层实现一般是对非线性优化问题利用SQP迭代求解。
速度	在无约束的情况下非常快，因为整个问题迭代过程的矩阵维度都很小，在有约束但是约束不苛刻的情况下，通过增广拉格朗日法收敛也比较快。时间复杂度是 $O(N(n+m)^3)$ 级别的	通常来说，构建成一个大矩阵的方式，即使利用了矩阵的稀疏特性，求解速度依旧会慢一些，时间复杂度是 $O(N^3(n+m)^3)$ 级别的
嵌入式平台实现	比较容易在嵌入式平台实现（问题维度较小）	难以在嵌入式平台求解复杂的非线性优化或者SQP问题
数值问题	由于DDP法本质上算是一种shooting方法，要依赖串级的梯度和Hessian的传导，因此，会随着Backward pass迭代过程的积累的舍入和截断误差和矩阵的幂乘使矩阵的条件数变大，所以在Horizon大时，会使问题变得病态	通常来说是数值鲁棒的，因为不需要有这样的一个很长传导和误差积累过程
控制器	在迭代时，可以获得一个前馈+反馈的控制率，在收敛后，前馈控制率为0，免费获得了一个时变LQR问题的最优反馈控制率，因此不需要额外设计控制器来跟踪	优化收敛后得到的是仅仅是一条前馈轨迹，需要额外设计一个控制器来跟踪这条轨迹。

离散化实现

```
# 将连续时间 [0,T] 离散化为N个时间节点  
Nt = Int(Tfinal/h)+1      # 51个时间节点  
n_nlp = (Nx+Nu)*Nt       # 255个决策变量 = (4状态+1控制)×51  
m_nlp = Nx*(Nt+1)         # 208个约束 = 4×52
```

转换后的离散非线性规划问题：

$$\min_{x_k, u_k} \sum_{k=1}^{N_t} L(x_k, u_k) \quad (160)$$

$$\text{s.t. 配点约束、边界约束} \quad (161)$$

2. 核心：Hermite-Simpson配点约束 Hermite-Simpson配点是算法的数学核心，它使用三点 (x_1, x_m, x_2) 来近似微分方程。

定义 8.3 (Hermite-Simpson配点约束). 对每个时间区间 $[t_k, t_{k+1}]$ ，配点约束定义为：

$$x_m = \frac{1}{2}(x_1 + x_2) + \frac{h}{8}(f_1 - f_2) \quad (162)$$

$$u_m = \frac{1}{2}(u_1 + u_2) \quad (163)$$

$$\dot{x}_m = \frac{-3}{2h}(x_1 - x_2) - \frac{1}{4}(f_1 + f_2) \quad (164)$$

$$f_m = f(x_m, u_m) \quad (165)$$

$$0 = f_m - \dot{x}_m \quad (166)$$

其中 $f_1 = f(x_1, u_1)$, $f_2 = f(x_2, u_2)$ 。

配点约束实现

```
function dircol_dynamics(x1,u1,x2,u2)
    f1 = RZ.dynamics(a, x1, u1)      # 左端点动力学
    f2 = RZ.dynamics(a, x2, u2)      # 右端点动力学

    # 中点状态 (不是简单平均!)
    xm = 0.5*(x1 + x2) + (h/8.0)*(f1 - f2)
    um = 0.5*(u1 + u2)

    # 中点导数 (Simpson公式)
    □m = (-3/(2.0*h))*(x1 - x2) - 0.25*(f1 + f2)

    # 中点力学
    fm = RZ.dynamics(a, xm, um)

    return fm - □m  # 强制中点满足微分方程
end
```

注释 8.4. Hermite-Simpson积分提供4阶精度，远高于简单的欧拉法（1阶）或中点法（2阶）。数学原理：对每个区间 $[t_k, t_{k+1}]$ ，要求 $f(x_m, u_m) = \dot{x}_m$ （中点处的动力学一致性）。

3. 构建大型NLP问题 决策变量向量包含所有时刻的状态和控制：

$$z = [x_1; u_1; x_2; u_2; \dots; x_{51}; u_{51}] \in \mathbb{R}^{255} \quad (167)$$

约束函数包含初始条件、动力学约束和终端条件：

约束函数实现

```
function con!(c,ztraj)
    z = reshape(ztraj,Nx+Nu,Nt)
    c[1:Nx] .= z[1:Nx,1] - x0                      # 初始条件约束
    @views dynamics_constraint!(c[(Nx+1):(end-Nx)],ztraj) # 动力学约束
    c[(end-Nx+1):end] .= z[1:Nx,end] - xgoal        # 终端条件约束
end

function dynamics_constraint!(c,ztraj)
    d = reshape(c,Nx,Nt-1)      # 重塑约束向量
    z = reshape(ztraj,Nx+Nu,Nt) # 重塑决策变量
    for k = 1:(Nt-1)
        # 提取相邻时刻的状态和控制
        x1 = z[1:Nx,k]
        u1 = z[(Nx+1):(Nx+Nu),k]
        x2 = z[1:Nx,k+1]
        u2 = z[(Nx+1):(Nx+Nu),k+1]
        # 计算配点约束
        d[:,k] = dircol_dynamics(x1,u1,x2,u2)
    end
    return nothing
end
```

4. 自动微分计算梯度 现代优化算法的一个重要优势是使用自动微分技术，无需手工推导复杂的梯度公式。

自动微分实现

```
# 目标函数梯度
function MOI.eval_objective_gradient(prob::MOI.AbstractNLPEvaluator, grad_f, x)
    ForwardDiff.gradient!(grad_f, cost, x) # 自动微分计算梯度
    return nothing
end

# 约束雅可比矩阵
function MOI.eval_constraint_jacobian(prob::MOI.AbstractNLPEvaluator, jac, x)
    # 自动微分计算约束雅可比矩阵
    ForwardDiff.jacobian!(reshape(jac, prob.m_nlp, prob.n_nlp), cons,
        zeros(prob.m_nlp), x)
    return nothing
end
```

命题 8.5 (自动微分的优势). 自动微分技术提供:

1. 机器精度的梯度计算
2. 避免数值微分的误差
3. 无需手工推导复杂的梯度表达式
4. 高效的计算复杂度

5. 使用Ipopt内点法求解 :

Algorithm 10: Ipopt内点法求解流程

Result: 最优轨迹 $x^*(t)$, $u^*(t)$
初始化Ipopt求解器参数;
设置收敛容差: $\text{tol} = 10^{-6}$;
设置最大迭代次数: $\text{max_iter} = 10000$;
while 未收敛且迭代次数 $< \text{max_iter}$ **do**
 计算目标函数梯度 $\nabla f(z)$;
 计算约束雅可比矩阵 $\nabla c(z)$;
 求解KKT系统获得搜索方向 Δz ;
 线搜索确定步长 α ;
 更新变量: $z \leftarrow z + \alpha \Delta z$;
 检查收敛条件;
end

Ipopt求解器配置

```
# 求解器函数
function solve(x0,prob::MOI.AbstractNLPEvaluator;
    tol=1.0e-6,c_tol=1.0e-6,max_iter=10000)
    x_l, x_u = prob.primal_bounds
    c_l, c_u = prob.constraint_bounds

    nlp_bounds = MOI.NLPBoundsPair.(c_l,c_u)
    block_data = MOI.NLPBlockData(nlp_bounds,prob,true)

    # 配置Ipopt求解器
    solver = Ipopt.Optimizer()
    solver.options["max_iter"] = max_iter      # 最大迭代次数
    solver.options["tol"] = tol                  # 收敛容差
    solver.options["constr_viol_tol"] = c_tol   # 约束违反容差

    x = MOI.add_variables(solver,prob.n_nlp)

    # 设置变量边界和初值
    for i = 1:prob.n_nlp
        MOI.add_constraint(solver, x[i], MOI.LessThan(x_u[i]))
        MOI.add_constraint(solver, x[i], MOI.GreaterThan(x_l[i]))
        MOI.set(solver, MOI.VariablePrimalStart(), x[i], x0[i])
    end

    # 求解优化问题
    MOI.set(solver, MOI.NLPBlock(), block_data)
    MOI.set(solver, MOI.ObjectiveSense(), MOI.MIN_SENSE)
    MOI.optimize!(solver)

    # 获取解
    res = MOI.get(solver, MOI.VariablePrimal(), x)
    return res
end
```

Ipopt算法特点：内点法处理约束、牛顿法求解KKT条件、BFGS近似海塞矩阵、线搜索保证收敛。

6. 求解过程详解 良好的初始猜测对收敛性至关重要：

初始猜测设置

```
# 初始猜测
xguess = kron(ones(Nt)', x0) # 状态初始猜测（保持初始状态）
uguess = zeros(Nt)'          # 控制初始猜测（零控制）
z0 = reshape([xguess; uguess], (Nx+Nu)*Nt, 1); # 组合成决策变量向量
```

Ipopt内部执行以下迭代步骤：

1. **线性化**: 在当前点计算梯度和雅可比矩阵
2. **求解QP子问题**: 获得搜索方向
3. **线搜索**: 确定合适的步长
4. **更新**: $x \leftarrow x + \alpha \cdot \Delta x$
5. **检查收敛**: $\|\nabla \mathcal{L}\| < \text{tol}$

算法在满足以下条件时收敛：

$$\|\nabla \mathcal{L}\| < 10^{-6} \quad (\text{一阶最优性}) \quad (168)$$

$$\|c(x)\| < 10^{-6} \quad (\text{约束违反}) \quad (169)$$

$$\text{迭代次数} < 10000 \quad (\text{最大迭代限制}) \quad (170)$$

7. 算法有效性分析

[直接配点法的优势] 直接配点法相比其他方法具有以下优势：

1. **全局视角**: 同时优化整条轨迹，避免打靶法的初值敏感性
2. **高精度**: Hermite-Simpson提供4阶积分精度
3. **稀疏性**: 约束矩阵具有带状稀疏结构，求解高效
4. **鲁棒性**: Ipopt对初值选择不敏感

注释 8.6 (计算复杂度). 算法的计算复杂度特征：

- 变量数: $\mathcal{O}(N)$ (N 为时间节点数)
- 约束数: $\mathcal{O}(N)$
- 雅可比稀疏度: 每行最多几个非零元素

例子 8.7 (Acrobot系统求解). 对于Acrobot (双摆) 系统：

- 状态维数: $N_x = 4$ (两个关节角度和角速度)
- 控制维数: $N_u = 1$ (单个关节扭矩)

- 时间节点: $N_t = 51$
- 决策变量总数: $(4 + 1) \times 51 = 255$
- 约束总数: $4 \times 52 = 208$

完整求解流程

```
# 创建并求解优化问题
prob = ProblemMOI(n_nlp,m_nlp)
z_sol = solve(z0,prob)           # 求解最优控制问题
ztraj = reshape(z_sol,Nx+Nu,Nt) # 重塑解轨迹
xtraj = ztraj[1:Nx,:]          # 提取状态轨迹
utraj = ztraj[(Nx+1):(Nx+Nu),:] # 提取控制轨迹
```

总结而言，直接配点法实现了“变分法→配点法→NLP→内点法”的完整求解链条，将复杂的最优控制问题转化为标准的非线性规划问题求解。

8.4 LQR在SE3中

对于旋转、四元数部分请阅读??

8.4.1 四元数的数值优化

对于四元数表示的系统下，数值优化并不能直接优化四元数，因为四元数必须保持单位长度。即，不能直接使用LQR、Raccati方法，因为这些方法是欧式空间上自由变化的。

在最优控制问题中，直接对四元数 q 进行优化会遇到单位模约束的问题，因为四元数必须始终保持单位长度。为了解决这个问题，常用的方法是采用扰动参数化：不是直接优化 q ，而是优化一个较小的旋转扰动 $\delta\phi \in \mathbb{R}^3$ ，然后通过指数映射将其转换为四元数的增量 δq 。

具体做法如下：

1. 设当前的参考四元数为 q ，我们用一个小扰动 $\delta\phi$ 表示姿态的微小变化。
2. 将 $\delta\phi$ 通过指数映射转为四元数增量 δq ：

$$\delta q = \exp\left(\frac{1}{2}\delta\phi\right) \quad (171)$$

其中 $\exp(\cdot)$ 表示四元数的指数映射。

3. 用新的四元数表示更新后的姿态：

$$q_{\text{new}} = \delta q \otimes q \quad (172)$$

其中 \otimes 表示四元数乘法。

这样，优化变量始终是三维的 $\delta\phi$ ，避免了单位模约束的问题。LQR等线性方法可以直接在 $\delta\phi$ 上进行，而每次状态更新时再通过四元数运算保证姿态的正确性。

常用的近似有：

$$\delta q \approx \begin{bmatrix} 1 \\ \frac{1}{2}\delta\phi \end{bmatrix} \quad (173)$$

适用于 $\delta\phi$ 较小时。另外还有如下常用近似：

$$\delta q \approx \begin{bmatrix} \cos\left(\frac{\|\delta\phi\|}{2}\right) \\ \sin\left(\frac{\|\delta\phi\|}{2}\right) \frac{\delta\phi}{\|\delta\phi\|} \end{bmatrix} \quad (\text{axis-angle 形式}) \quad (174)$$

$$\delta q \approx \begin{bmatrix} 1 \\ \frac{1}{2}\delta\phi \end{bmatrix} \approx \begin{bmatrix} 1 \\ \mathbf{v} \end{bmatrix} \quad (\text{四元数虚部/向量部分近似}) \quad (175)$$

这种方法在SLAM、轨迹优化等问题中被广泛采用，可以有效结合四元数的优点和欧式空间优化的便利性。

8.4.2 四元数扰动的雅可比（Attitude Jacobian）

在优化过程中，我们需要计算关于扰动 $\delta\phi$ 的梯度。由于优化变量是 $\delta\phi$ ，而实际状态是四元数 q ，因此需要用链式法则将 $\frac{\partial J}{\partial q}$ 转换为 $\frac{\partial J}{\partial \delta\phi}$ 。

对于小扰动 $\delta\phi$, 有如下近似:

$$\delta q \approx \begin{bmatrix} 1 \\ \frac{1}{2}\delta\phi \end{bmatrix} \quad (176)$$

四元数扰动对 $\delta\phi$ 的雅可比为

$$G(q) = \frac{\partial q}{\partial \delta\phi} = \frac{1}{2}L(q)H \quad (177)$$

其中 $L(q)$ 为左乘四元数矩阵, H 为 $\delta\phi$ 到四元数虚部的映射矩阵。

最终, 目标函数 J 对 $\delta\phi$ 的梯度为

$$\frac{\partial J}{\partial \delta\phi} = \frac{\partial J}{\partial q}G(q) \quad (178)$$

这种方法可以方便地将四元数优化问题转化为欧式空间的优化问题, 便于数值算法实现。

8.4.3 Cost function关于扰动变量的一阶、二阶导

在实际优化问题中, 除了需要一阶梯度, 还常常需要计算cost function关于扰动变量 $\delta\phi$ 的二阶导 (Hessian), 以便于牛顿法、二次规划等二阶优化方法。

设目标函数 $J(q(\delta\phi))$, 其一阶、二阶导数分别为:

$$\frac{\partial J}{\partial \delta\phi} = \frac{\partial J}{\partial q}G(q) \quad (179)$$

$$\frac{\partial^2 J}{\partial \delta\phi^2} = G(q)^\top \frac{\partial^2 J}{\partial q^2} G(q) + \frac{\partial J}{\partial q} \frac{\partial G(q)}{\partial \delta\phi} \quad (180)$$

其中, $\frac{\partial^2 J}{\partial q^2}$ 为cost function关于四元数 q 的Hessian, $\frac{\partial G(q)}{\partial \delta\phi}$ 为雅可比 $G(q)$ 关于 $\delta\phi$ 的导数项。对于小扰动近似, 第二项通常较小, 有时可忽略。

这种链式法则的推导方式, 使得我们可以方便地将四元数相关的优化问题转化为三维扰动变量 $\delta\phi$ 上的优化问题, 便于数值实现和理论分析。则对于系统方程, 我们可以想要推导四元数表达下的演化 $q_k \rightarrow q_{k+1}$ 变为 $q_k \rightarrow \phi_k \rightarrow \phi_{k+1} \rightarrow q_{k+1}$, 其中 ϕ_k 为扰动变量。

下面我们考虑一个最小化距离代价的例子, 用来定位。

8.4.4 例子: Wahba问题的四元数优化

考虑一个经典的姿态估计问题 (Wahba问题): 给定一组在世界坐标系 $\{N\}$ 中的单位向量 \mathbf{v}_i^N 和对应的在机体坐标系 $\{B\}$ 中的观测向量 \mathbf{v}_i^B , 求解从机体坐标系到世界坐标系的旋转矩阵 \mathbf{R} 。

问题描述: 最小化如下代价函数:

$$J(\mathbf{q}) = \frac{1}{2} \sum_{i=1}^n \|\mathbf{v}_i^N - \mathbf{Q}(\mathbf{q})\mathbf{v}_i^B\|^2 \quad (181)$$

其中 $\mathbf{Q}(\mathbf{q})$ 为四元数 \mathbf{q} 对应的旋转矩阵:

$$\mathbf{Q}(\mathbf{q}) = \mathbf{H}^T \mathbf{R}(\mathbf{q})^T \mathbf{L}(\mathbf{q}) \mathbf{H} \quad (182)$$

这里 $\mathbf{L}(\mathbf{q})$ 和 $\mathbf{R}(\mathbf{q})$ 分别为四元数的左乘和右乘矩阵, $\mathbf{H} = \begin{bmatrix} \mathbf{0}_{1 \times 3} \\ \mathbf{I}_{3 \times 3} \end{bmatrix}$ 为选择矩阵。

扰动参数化: 使用三维扰动参数 $\boldsymbol{\phi} \in \mathbb{R}^3$ 进行优化, 当前四元数的更新规则为:

$$\mathbf{q}_{\text{new}} = \mathbf{L}(\mathbf{q}) \begin{bmatrix} \sqrt{1 - \boldsymbol{\phi}^T \boldsymbol{\phi}} \\ \boldsymbol{\phi} \end{bmatrix} \quad (183)$$

Gauss-Newton 算法: 定义残差向量 $\mathbf{r}(\mathbf{q}) = [\mathbf{v}_1^N - \mathbf{Q}(\mathbf{q})\mathbf{v}_1^B; \dots; \mathbf{v}_n^N - \mathbf{Q}(\mathbf{q})\mathbf{v}_n^B]$, 则:

1. 计算残差对四元数的雅可比: $\mathbf{J}_r = \frac{\partial \mathbf{r}}{\partial \mathbf{q}}$
2. 转换为对扰动参数的雅可比: $\nabla_{\boldsymbol{\phi}} \mathbf{r} = \mathbf{J}_r \mathbf{G}(\mathbf{q})$, 其中 $\mathbf{G}(\mathbf{q}) = \mathbf{L}(\mathbf{q}) \mathbf{H}$
3. Gauss-Newton 更新步骤:

$$\boldsymbol{\phi} = -(\nabla_{\boldsymbol{\phi}} \mathbf{r}^T \nabla_{\boldsymbol{\phi}} \mathbf{r})^{-1} \nabla_{\boldsymbol{\phi}} \mathbf{r}^T \mathbf{r} \quad (184)$$

4. 更新四元数: $\mathbf{q} \leftarrow \mathbf{L}(\mathbf{q}) \begin{bmatrix} \sqrt{1 - \boldsymbol{\phi}^T \boldsymbol{\phi}} \\ \boldsymbol{\phi} \end{bmatrix}$

5. 重复步骤1-4直到收敛 ($\|\boldsymbol{\phi}\| < \epsilon$)

这种方法避免了直接对四元数进行优化时的单位模约束问题, 同时保持了四元数表示的数值稳定性。算法收敛后, 估计的四元数 \mathbf{q} 与真实四元数 \mathbf{q}_{true} 相差一个符号 (因为 \mathbf{q} 和 $-\mathbf{q}$ 表示同一旋转)。

Gauss-Newton 更新步公式的推导:

对于非线性最小二乘问题 $\min_{\boldsymbol{\phi}} \frac{1}{2} \|\mathbf{r}(\boldsymbol{\phi})\|^2$, 我们需要求解:

目标函数为:

$$J(\boldsymbol{\phi}) = \frac{1}{2} \mathbf{r}^T(\boldsymbol{\phi}) \mathbf{r}(\boldsymbol{\phi}) \quad (185)$$

一阶导数 (梯度):

$$\frac{\partial J}{\partial \boldsymbol{\phi}} = \mathbf{r}^T(\boldsymbol{\phi}) \frac{\partial \mathbf{r}}{\partial \boldsymbol{\phi}} = (\nabla_{\boldsymbol{\phi}} \mathbf{r})^T \mathbf{r}(\boldsymbol{\phi}) \quad (186)$$

二阶导数 (Hessian):

$$\frac{\partial^2 J}{\partial \boldsymbol{\phi}^2} = (\nabla_{\boldsymbol{\phi}} \mathbf{r})^T (\nabla_{\boldsymbol{\phi}} \mathbf{r}) + \mathbf{r}^T(\boldsymbol{\phi}) \frac{\partial^2 \mathbf{r}}{\partial \boldsymbol{\phi}^2} \quad (187)$$

在 Gauss-Newton 方法中, 我们忽略二阶项 $\mathbf{r}^T(\boldsymbol{\phi}) \frac{\partial^2 \mathbf{r}}{\partial \boldsymbol{\phi}^2}$ (假设在最优点附近残差足够小), 得到近似 Hessian:

$$\mathbf{H}_{\text{GN}} \approx (\nabla_{\boldsymbol{\phi}} \mathbf{r})^T (\nabla_{\boldsymbol{\phi}} \mathbf{r}) \quad (188)$$

根据Newton法的更新公式 $\phi_{\text{new}} = \phi - \mathbf{H}^{-1}\nabla J$, 有:

$$\Delta\phi = -\mathbf{H}_{\text{GN}}^{-1} \frac{\partial J}{\partial \phi} \quad (189)$$

$$= -[(\nabla_\phi \mathbf{r})^T (\nabla_\phi \mathbf{r})]^{-1} (\nabla_\phi \mathbf{r})^T \mathbf{r}(\phi) \quad (190)$$

因此, Gauss-Newton更新步为:

$$\phi = -(\nabla_\phi \mathbf{r}^T \nabla_\phi \mathbf{r})^{-1} \nabla_\phi \mathbf{r}^T \mathbf{r} \quad (191)$$

这就是上述步骤3中使用的更新公式。该方法的优势在于:

- 避免计算复杂的二阶导数
- 保证 \mathbf{H}_{GN} 为正定矩阵 (当 $\nabla_\phi \mathbf{r}$ 满秩时)
- 在残差较小时具有超线性收敛速度

标准牛顿法回顾:

对于无约束优化问题 $\min_{\mathbf{x}} f(\mathbf{x})$, 标准牛顿法基于二阶泰勒展开:

$$f(\mathbf{x} + \Delta\mathbf{x}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^T \Delta\mathbf{x} + \frac{1}{2} \Delta\mathbf{x}^T \mathbf{H}(\mathbf{x}) \Delta\mathbf{x} \quad (192)$$

其中 $\nabla f(\mathbf{x})$ 是梯度, $\mathbf{H}(\mathbf{x})$ 是Hessian矩阵。

对 $\Delta\mathbf{x}$ 求导并令其为零, 得到最优步长:

$$\frac{\partial}{\partial \Delta\mathbf{x}} [\nabla f(\mathbf{x})^T \Delta\mathbf{x} + \frac{1}{2} \Delta\mathbf{x}^T \mathbf{H}(\mathbf{x}) \Delta\mathbf{x}] = 0 \quad (193)$$

$$\nabla f(\mathbf{x}) + \mathbf{H}(\mathbf{x}) \Delta\mathbf{x} = 0 \quad (194)$$

因此, 牛顿法的更新公式为:

$$\Delta\mathbf{x} = -\mathbf{H}(\mathbf{x})^{-1} \nabla f(\mathbf{x}) \quad (195)$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k) \quad (196)$$

与Gauss-Newton的对比:

- 牛顿法: 使用真实Hessian $\mathbf{H}(\mathbf{x}) = \frac{\partial^2 f}{\partial \mathbf{x}^2}$
- Gauss-Newton法: 对于 $f(\mathbf{x}) = \frac{1}{2} \|\mathbf{r}(\mathbf{x})\|^2$, 使用近似Hessian $\mathbf{H}_{\text{GN}} = \mathbf{J}_r^T \mathbf{J}_r$

牛顿法具有二次收敛性, 但需要计算和求逆Hessian矩阵, 计算成本较高。Gauss-Newton法虽然收敛速度略慢, 但计算更高效, 特别适用于非线性最小二乘问题。

8.4.5 基于四元数LQR的四旋翼飞行器

在这一节中, 我们将展示如何将前面介绍的四元数扰动方法应用到四旋翼飞行器的LQR控制中。四旋翼系统是一个典型的欠驱动系统, 其姿态控制需要处理四元数的单位约束问题。

四旋翼飞行器动力学模型

考虑一个四旋翼飞行器系统，其状态向量包含位置、姿态（四元数）、线速度和角速度：

$$\mathbf{x} = \begin{bmatrix} \mathbf{r} \\ \mathbf{q} \\ \mathbf{v} \\ \boldsymbol{\omega} \end{bmatrix} \in \mathbb{R}^{13} \quad (197)$$

其中 $\mathbf{r} \in \mathbb{R}^3$ 为位置， $\mathbf{q} \in \mathbb{R}^4$ 为单位四元数， $\mathbf{v} \in \mathbb{R}^3$ 为体坐标系下的线速度， $\boldsymbol{\omega} \in \mathbb{R}^3$ 为体坐标系下的角速度。

动力学方程为：

$$\begin{cases} \dot{\mathbf{r}} = \mathbf{Q}(\mathbf{q})\mathbf{v} \\ \dot{\mathbf{q}} = \frac{1}{2}\mathbf{L}(\mathbf{q})\mathbf{H}\boldsymbol{\omega} \\ \dot{\mathbf{v}} = \mathbf{Q}(\mathbf{q})^T\mathbf{g} + \frac{1}{m}\mathbf{B}_f\mathbf{u} - \boldsymbol{\omega} \times \mathbf{v} \\ \dot{\boldsymbol{\omega}} = \mathbf{J}^{-1}(-\boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} + \mathbf{B}_\tau\mathbf{u}) \end{cases} \quad (198)$$

其中： - $\mathbf{Q}(\mathbf{q})$ 为四元数对应的旋转矩阵 - $\mathbf{L}(\mathbf{q})$ 为四元数左乘矩阵 - $\mathbf{H} = \begin{bmatrix} \mathbf{0}_{1 \times 3} \\ \mathbf{I}_{3 \times 3} \end{bmatrix}$ 为选择矩阵 - $\mathbf{g} = [0, 0, -g]^T$ 为重力向量 - \mathbf{B}_f 和 \mathbf{B}_τ 分别为力和力矩分配矩阵 - $\mathbf{u} \in \mathbb{R}^4$ 为四个电机的推力输入

- $\dot{\mathbf{v}}$ 公式推导如下：由牛顿第二定律，机体坐标系下的线速度动力学推导如下：

$$m\dot{\mathbf{v}}^B = \mathbf{F}_{\text{total}}^B - m\boldsymbol{\omega}^B \times \mathbf{v}^B \quad (199)$$

$$\Rightarrow \dot{\mathbf{v}}^B = \frac{1}{m}\mathbf{F}_{\text{total}}^B - \boldsymbol{\omega}^B \times \mathbf{v}^B \quad (200)$$

其中， $\mathbf{F}_{\text{total}}^B$ 为机体坐标系下的总力。通常包括重力和螺旋桨推力：

$$\mathbf{F}_{\text{total}}^B = \mathbf{Q}^T(\mathbf{q})m\mathbf{g} + \mathbf{B}_f\mathbf{u} \quad (201)$$

因此，完整表达式为：

$$\dot{\mathbf{v}}^B = \mathbf{Q}^T(\mathbf{q})\mathbf{g} + \frac{1}{m}\mathbf{B}_f\mathbf{u} - \boldsymbol{\omega}^B \times \mathbf{v}^B \quad (202)$$

其中 $\mathbf{Q}^T(\mathbf{q})\mathbf{g}$ 为重力在机体坐标系下的分量， $\mathbf{B}_f\mathbf{u}$ 为推力分配， $-\boldsymbol{\omega}^B \times \mathbf{v}^B$ 为科氏力项。

- $\dot{\mu}$ 是由欧拉公式给出的:

$$\mathbf{J}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} = \mathbf{B}_\tau \mathbf{u} \quad (203)$$

其中 \mathbf{J} 为转动惯量矩阵, $\boldsymbol{\omega}$ 为机体角速度, $\mathbf{B}_\tau \mathbf{u}$ 为输入力矩。该方程即为欧拉方程 (Euler's Equation), 描述了刚体的角加速度与输入力矩、陀螺力之间的关系。

为什么转换到四元数表示

使用四元数表示姿态具有以下优势:

- **避免奇点:** 与欧拉角不同, 四元数没有万向锁问题
- **数值稳定性:** 四元数的插值和积分更加稳定
- **计算效率:** 旋转合成的计算复杂度更低
- **全局有效性:** 四元数可以表示任意旋转而无奇点

但是, 四元数必须满足单位长度约束 $\|\mathbf{q}\| = 1$, 这使得传统的线性控制方法 (如LQR) 无法直接应用。

四元数约束下的线性化

首先给定一个离散时间下的动态系统的参考点 \bar{x}_n, \bar{u}_n 使用一阶泰勒展开对系统进行线性化, 可以得到:

$$\bar{x}_{n+1} + \Delta x_{n+1} = f(\bar{x}_n + \Delta x_n, \bar{u}_n + \Delta u_n) \approx f(\bar{x}_n, \bar{u}_n) + A_n \Delta x_n + B_n \Delta u_n \quad (204)$$

其中 $A_n = \frac{\partial f}{\partial x}|_{(\bar{x}_n, \bar{u}_n)}$, $B_n = \frac{\partial f}{\partial u}|_{(\bar{x}_n, \bar{u}_n)}$ 。

由于四元数约束的存在, 矩阵 $\mathbf{A} \in \mathbb{R}^{13 \times 13}$ 总是奇异的 (秩为12), 导致系统不可控。

约简系统与可控性分析

为了解决这个问题, 我们将四元数误差参数化为三维Rodrigues参数 ϕ :

$$\mathbf{q}_{\text{error}} = \mathbf{L}(\mathbf{q}_0)^T \mathbf{q} \approx \begin{bmatrix} 1 \\ \frac{1}{2}\phi \end{bmatrix} \quad (205)$$

定义约简状态 $\tilde{\mathbf{x}} \in \mathbb{R}^{12}$:

$$\tilde{\mathbf{x}} = \begin{bmatrix} \Delta \mathbf{r} \\ \phi \\ \Delta \mathbf{v} \\ \Delta \boldsymbol{\omega} \end{bmatrix} \quad (206)$$

通过变换矩阵 $\mathbf{E}(\mathbf{q})$:

$$\mathbf{E}(\mathbf{q}) = \text{blkdiag}(\mathbf{I}_3, \mathbf{G}(\mathbf{q}), \mathbf{I}_6) \quad (207)$$

其中 $\mathbf{G}(\mathbf{q}) = \mathbf{L}(\mathbf{q})\mathbf{H}$ 。

约简系统为:

$$\Delta \dot{\tilde{\mathbf{x}}} = \tilde{\mathbf{A}} \Delta \tilde{\mathbf{x}} + \tilde{\mathbf{B}} \Delta \mathbf{u} \quad (208)$$

其中:

$$\tilde{\mathbf{A}}_k = \mathbf{E}(\mathbf{x}_{k-1}^-)^T \mathbf{A} \mathbf{E}(\mathbf{x}_n^-) \quad (209)$$

$$\tilde{\mathbf{B}}_k = \mathbf{E}^T(\mathbf{x}_{n-1}^-)^T \mathbf{B} \quad (210)$$

可控性证明

约简系统($\tilde{\mathbf{A}}$, $\tilde{\mathbf{B}}$)的可控性矩阵为:

$$\mathcal{C} = \begin{bmatrix} \tilde{\mathbf{B}} & \tilde{\mathbf{A}}\tilde{\mathbf{B}} & \cdots & \tilde{\mathbf{A}}^{11}\tilde{\mathbf{B}} \end{bmatrix} \quad (211)$$

对于四旋翼系统, 可以证明 $\text{rank}(\mathcal{C}) = 12$, 即约简系统是完全可控的。这是因为:

- 位置通过推力控制间接可控
- 姿态通过力矩直接可控
- 速度和角速度通过控制输入直接影响

LQR控制器设计

对约简系统设计LQR控制器, 求解代数Riccati方程:

$$\tilde{\mathbf{A}}^T \mathbf{P} + \mathbf{P} \tilde{\mathbf{A}} - \mathbf{P} \tilde{\mathbf{B}} \mathbf{R}^{-1} \tilde{\mathbf{B}}^T \mathbf{P} + \mathbf{Q} = 0 \quad (212)$$

最优反馈增益为:

$$\mathbf{K} = \mathbf{R}^{-1} \tilde{\mathbf{B}}^T \mathbf{P} \quad (213)$$

控制律为:

$$\mathbf{u} = \mathbf{u}_0 - \mathbf{K} \Delta \tilde{\mathbf{x}} \quad (214)$$

其中姿态误差 ϕ 通过四元数误差计算:

$$\phi = 2 \cdot \text{vec}(\mathbf{L}(\mathbf{q}_0)^T \mathbf{q}) \quad (215)$$

这种方法成功地将四元数约束问题转化为无约束的线性控制问题, 使得传统的LQR方法可以直接应用于四旋翼姿态控制, 同时保持了四元数表示的所有优势。

9 混合系统

9.1 接触动力学仿真

接触模型，仅考虑单一动力学模型和连续状态的仿真（ODE）不足以描述接触力的复杂性。在机器人学中，我们学过，将这些力建模成弹簧和阻尼器的组合是有用的。接触力可以通过以下方式建模：

$$f_c = k \cdot x + b \cdot \dot{x} \quad (216)$$

其中 f_c 是接触力， k 是弹簧常数， b 是阻尼系数， x 是位置偏移， \dot{x} 是速度。

但这里我们讨论接触力如何进行模拟，在第二章和第三章，我们讲述了对于线性系统，使用ODE和PDE进行模拟。对于接触力，我们需要使用混合系统来描述。

在机器人仿真和优化中，存在一类涉及接触的复杂问题，包括四足机器人的运动控制（locomotion）、机械臂的操作优化（manipulation）以及弹性碰撞仿真等。这类问题由于接触现象的存在，使得传统的基于单一动力学模型和连续状态变化的积分仿真方法无法直接适用，需要采用特殊的处理技术。

考虑球体撞击刚性地面的情形：在接触瞬间，球体的速度会在极短时间内从向下的非零值突变为向上的非零值。这种物理过程具有不连续性，难以用传统方法建模。类似地，在四足机器人行走过程中，足端着地瞬间速度会突变为零，同时系统的状态方程也会发生相应变化。这些不连续的状态跳变是控制系统设计中必须考虑的关键因素。

目前机器人学领域主要采用两种方法处理此类问题：

- **基于事件的方法 (Event-based/Hybrid Method) :** 采用连续的常微分方程 (ODE) 进行系统积分，在积分过程中通过守护函数 (guard function) 持续监测接触条件。当接触发生时，执行跳跃映射 (jump map) 处理所有不连续操作 (状态跳变、方程切换)，随后继续连续时间ODE积分。

该方法将系统建模为混合动力学系统，明确区分连续和离散的动态过程：

- 连续积分阶段：使用ODE积分器推进系统状态

$$\dot{x} = f(x, u, t) \quad (217)$$

- 事件检测：Guard函数持续监测接触条件

$$g(x) = 0 \quad (218)$$

- 状态跳变：当检测到接触时，执行Jump Map

$$x^+ = J(x^-) \quad (219)$$

- **时间步进/接触隐式方法 (Time-Stepping/Contact Implicit Method) :** 将连续和不连续过程统一描述为带约束的优化问题。每个迭代步骤不使用原始连续ODE方程，而是根据优化问题结果进行迭代，在优化框架中统一考虑接触约束现象。

该方法在每个时间步 k 内求解以下优化问题：

$$\min_{x_{k+1}, \lambda} \quad \frac{1}{2} \|x_{k+1} - x_{\text{pred}}\|^2 \quad (220)$$

$$\text{s.t.} \quad M(x_{k+1} - x_k) = h \cdot f(x_k, u_k) + h \cdot \lambda \quad (221)$$

$$\phi(x_{k+1}) \geq 0 \quad (222)$$

$$\phi(x_{k+1}) \perp \lambda \geq 0 \quad (223)$$

其中 $\phi(x)$ 是接触距离函数， λ 是接触力， \perp 表示互补性条件。

前者广泛应用于控制领域（如四足机器人运动控制），后者主要用于机器人仿真平台（Gazebo、PyBullet、MuJoCo等）。

在控制领域，混合方法能够很好地兼容传统的直接配点法（DIRCOL）或微分动态规划（DDP）类优化算法，仅需在接触点处添加约束条件和优化变量。然而，该方法的局限性在于通常需要预先指定接触时间和切换序列（对运动问题而言即给定步态），虽然存在优化接触时间的研究工作，但整体优化空间有限。

接触隐式方法虽然理论上可应用于控制领域，但本质上将每步的无约束ODE问题转换为有约束问题。若直接应用于传统DDP或序列二次规划（SQP）框架，会显著增加优化复杂度，效果欠佳。因此，采用此方法需要从根本上重新构建问题表述，这也是当前的一个活跃研究方向。

9.2 砖块掉落案例

考虑一个简单的砖块自由下落并与地面发生接触的动力学系统。我们用如下符号描述系统状态：

- q : 砖块的位置
- v : 砖块的速度
- m : 砖块质量
- g : 重力加速度
- λ : 地面对砖块的接触力

系统动力学模型为：

$$m \frac{v_{k+1} - v_k}{h} = -mg + J^T \lambda_k \quad (224)$$

$$q_{k+1} = q_k + h v_{k+1} \quad (225)$$

其中 h 为步长， J 为接触雅可比矩阵（本例为 $J = 1$ ）， λ_k 为接触力。

接触约束为：

$$\phi(q_{k+1}) \geq 0 \quad (226)$$

$$\lambda_k \geq 0 \quad (227)$$

$$\phi(q_{k+1}) \lambda_k = 0 \quad (228)$$

其中 $\phi(q_{k+1})$ 表示砖块与地面的距离。

证明该问题可转化为QP问题 上述动力学和接触约束实际上是一个典型的互补条件（KKT条件）问题。我们可以将其转化为如下带等式约束的二次规划（QP）问题：

$$\min_{v_{k+1}} \frac{1}{2}mv_{k+1}^2 + mv_{k+1}(hg - v_k) \quad (229)$$

$$\text{s.t. } J(q_k + hv_{k+1}) = 0 \quad (230)$$

证明过程：

1. 由动力学方程

$$m\frac{v_{k+1} - v_k}{h} = -mg + J^T \lambda_k$$

可得

$$mv_{k+1} = mv_k - mhg + hJ^T \lambda_k$$

$$v_{k+1} = v_k - hg + \frac{h}{m}J^T \lambda_k$$

2. 由接触约束 $J(q_k + hv_{k+1}) = 0$, 可解出 v_{k+1} 的取值范围。

3. 结合互补条件 $\lambda_k \geq 0, \phi(q_{k+1}) \geq 0, \phi(q_{k+1})\lambda_k = 0$, 该问题等价于求解如下优化问题：

$$\begin{aligned} \min_{v_{k+1}, \lambda_k} \quad & \frac{1}{2}m(v_{k+1} - v_k + hg)^2 \\ \text{s.t.} \quad & J(q_k + hv_{k+1}) = 0 \\ & \lambda_k \geq 0 \end{aligned}$$

其中目标函数来自于对动力学方程的最小二乘化，约束来自于接触条件。

因此，该带互补约束的动力学问题可以被等价地转化为带等式约束的QP问题进行求解。

结论： 砖块掉落接触问题的每一步都可以通过求解一个带等式约束的QP问题得到最优速度 v_{k+1} 和接触力 λ_k , 从而实现物理一致的仿真。

优点：

- 统一考虑了接触和不接触情况
- 显式计算了接触力

缺点：

- 无法直接获得准确的接触时间（虽然在这个问题中我们可以很直接地 check, 但在复杂问题中, 实际 contact 的时间是无法直接得到的, 而在某些情况下（如操作时）, 又需要这个时间）
- 只能采用欧拉积分的方式（多为后向欧拉积分, 正如 Lecture 2 中讨论的, 这种方法比前向的更稳定), 无法用更高阶的方式如 RK4。这造成的结果是, 采用较大的仿真步长时, 仿真的精度会下降。

9.3 混合轨迹优化对于Legged Robot

one-legged hopper

10 前馈补偿与迭代学习控制

在轨迹优化问题中，我们通常基于一个假设的系统模型进行优化。然而，实际系统与优化时所用模型之间往往存在误差。这种模型误差可能导致轨迹优化阶段得到的解并不是实际系统的局部最优点，即优化结果在真实系统中表现不佳。模型误差的来源主要有两方面：一是为了简化优化过程，常常采用线性化或简化的模型；二是即使采用了较为完整的动力学模型，实际参数值也可能存在偏差，或者环境中存在不可预知的扰动（如阵风、摩擦等）。

通常情况下，可以通过反应式/反馈控制器（如MPC或LQR）来跟踪规划得到的轨迹，利用反馈机制补偿模型误差，使系统 x_{real} 尽量接近期望轨迹 x_{ref} 。但对于需要反复执行同一轨迹的场景（如流水线机械臂、巡航无人机等），还可以采用前馈⁸补偿的方法：通过多次试验和修正，使得系统在实际环境中开环执行 u_{ref} 时就能很好地跟踪目标轨迹 x_{ref} 。与反馈控制相比，前馈补偿理论上具有更低的延迟和更好的跟踪性能。本节将介绍一种实现前馈补偿的方法——迭代学习控制（Iterative Learning Control, ILC）。

这种方式有点类似于强化学习中的 sim2real（从仿真到现实）过程：我们首先在假设的模型上进行轨迹优化，得到一条参考轨迹，然后将其应用到实际系统中执行。根据实际执行后的 rollout error（轨迹跟踪误差），对前馈轨迹进行修正。经过 5-6 次这样的迭代修正后，通常可以获得非常好的跟踪效果。

那么，为什么不直接在真实系统上进行 trajectory optimization 呢？主要原因是这样做的代价太高、耗时太久。实际系统的试验成本、风险和时间消耗都远大于仿真环境，因此通常采用“先仿真优化、再实物修正”的策略。为了进一步提升控制精度，可以采用数据驱动（data-driven）的方法，主要从两大方向入手：一是提升系统模型的准确性，二是优化控制律的设计。

一、系统模型的改进

- **系统参数辨识（Grey-box Identification）：**在已知系统结构的前提下，通过采集输入输出数据，利用最小二乘法等参数估计算法，获得最优的模型参数。

优点：数据利用效率高（sample efficient），模型具有良好的可解释性和泛化能力。

缺点：模型结构需事先假定，难以覆盖所有实际复杂性。

例：已知机械臂的动力学方程，但质量、摩擦系数未知，通过实验测量输入输出数据，利用最小二乘法估算这些参数。

- **黑盒建模（Black-box Modeling）：**无需预设系统结构，直接利用神经网络等函数逼近器对系统动力学或未建模动态（如扰动）进行建模。

优点：适用范围广，无需结构假设。

缺点：对数据量要求高，样本效率低，可解释性较差。

例：用深度神经网络学习四旋翼无人机的动力学，无需显式建模空气动力学细节。

二、控制律的优化

- **基于数据的控制律学习：**典型如强化学习（Policy Gradient 等），直接用函数逼近器近似最优反馈控制律。

⁸前馈（Feedforward）与反馈（Feedback）的区别在于：前馈控制根据已知的系统模型和期望输入，提前计算出控制量，直接作用于系统，以抵消已知的扰动或实现期望输出，不依赖于系统的实际输出。例如，在机械臂控制中，根据期望轨迹和动力学模型计算出所需的力矩作为前馈补偿。反馈控制则根据系统的实际输出与期望输出之间的误差，实时调整控制量以修正偏差。例如，PID 控制器根据位置误差调整输入，实现轨迹跟踪。前馈适用于模型已知且扰动可预测的场景，反馈则能补偿模型不确定性和外部扰动，提升系统的鲁棒性。

优点：理论上不依赖于系统模型，适用于复杂或未知系统。

缺点：泛化能力有限（难以迁移到新场景），对数据和交互次数需求大。

例：用深度强化学习训练机械臂抓取物体，直接输出关节力矩，无需动力学模型。

- **基于数据的轨迹修正：**在已有名义模型生成的初始轨迹基础上，通过实际系统的执行数据对轨迹进行迭代修正。

优点：样本效率高，适合重复性任务。

缺点：依赖于初始模型的合理性，难以泛化到不同任务或场景。

例：流水线机械臂反复搬运同一物体，通过多次执行和误差修正，逐步优化前馈轨迹，实现高精度跟踪。

10.1 迭代学习控制 (ILC)

迭代学习控制 (ILC) 是一种用于重复性任务的控制方法，旨在通过多次执行同一任务来逐步改进控制输入，从而提高系统的跟踪精度。ILC 的核心思想是利用每次执行任务后获得的误差信息来调整下一次的控制输入，使得系统在多次迭代中逐渐接近期望轨迹。

下面我们开始推导迭代学习控制

ILC可以看作是一个特殊的 model-based policy gradient 方法。假设我们通过轨迹规划的到一条参考轨迹：

$$u_k = \bar{u}_k - K_k(x_k - \bar{x}_k) \quad (231)$$

其中 \bar{u}_k 是参考轨迹的控制输入， K_k 是反馈增益矩阵， x_k 是系统状态， \bar{x}_k 是参考状态。

通过ILC，我们只修正参考轨迹的控制输入 \bar{u}_k ，不修改参考状态 \bar{x}_k ，最终的目标是完全跟踪 \bar{x}_k 。因此，ILC的目标是通过多次迭代来改进控制输入，使得系统在每次迭代中都能更好地跟踪参考轨迹。

ILC的问题formulation与轨迹优化问题很像，可以用任何方式的cost function，这里我们设成二次形式

$$\begin{aligned} \min_{x_{1:N}, u_{1:N}} \quad & J = \sum_{n=1}^{N-1} \left[\frac{1}{2}(x_n - \bar{x}_n)^T Q(x_n - \bar{x}_n) + \frac{1}{2}(u_n - \bar{u}_n)^T R(u_n - \bar{u}_n) \right] \\ & + \frac{1}{2}(x_N - \bar{x}_N)^T Q_N(x_N - \bar{x}_N) \\ \text{s.t.} \quad & x_{n+1} = f_{nominal}(x_n) \end{aligned} \quad (232)$$

其中， x_n 为系统在第 n 步的实际状态， \bar{x}_n 为参考状态， u_n 为实际控制输入， \bar{u}_n 为参考控制输入。 Q 、 Q_N 、 R 分别为状态、终端状态和控制输入的权重矩阵。 $f_{nominal}(\cdot)$ 表示名义系统模型的状态转移函数。在 ILC 中， x_n 由实际系统 roll-out 得到，优化变量仅为 u_n ， \bar{x}_n 和 \bar{u}_n 为参考轨迹，不参与优化。

与普通轨迹优化问题的区别是：

- 这里的状态更新要在实际rollout⁹得到。
- 更新过程只会更新 u 而不更新 \bar{x} 。

⁹rollout指的是将当前的控制输入序列应用到真实系统（或高保真仿真环境）中，记录系统实际产生的状态轨迹。与在模型中“推演”不同，rollout强调在真实物理环境下执行，能够反映模型误差、外部扰动等实际因素。

为了进一步去理解ILC，我们写出(2)的KKT条件（根据Lecture 4 (6)），其中

$$L = J + \lambda^T c,$$

c 是约束对应本问题则是系统方程

$$\begin{aligned}\nabla_x L(x + \Delta x, \lambda + \Delta \lambda) &\approx \nabla_x L(x, \lambda) + \frac{\partial^2 L}{\partial x^2} \Delta x + \frac{\partial^2 L}{\partial x \partial \lambda} \Delta \lambda \\&= \nabla_x J(x) + \nabla_x \lambda^T c(x) + \frac{\partial^2 L}{\partial x^2} \Delta x + \frac{\partial c}{\partial x} \Delta \lambda \\&= \nabla_x J(x) + \frac{\partial c}{\partial x} \lambda + \frac{\partial^2 L}{\partial x^2} \Delta x + \frac{\partial c}{\partial x} \Delta \lambda \\&= \nabla_x J(x) + \frac{\partial c}{\partial x} \lambda_{\text{new}} + \frac{\partial^2 L}{\partial x^2} \Delta x\end{aligned}\tag{3}$$

$$\nabla_\lambda L(x + \Delta x, \lambda) \approx c(x) + \frac{\partial c}{\partial x} \Delta x\tag{233}$$

步骤解释

- 第一步：对 $L(x, \lambda)$ 关于 x 和 λ 进行一阶泰勒展开，得到增量近似。
- 第二步：将拉格朗日函数 $L(x, \lambda) = J(x) + \lambda^T c(x)$ 的梯度展开， $\nabla_x L(x, \lambda) = \nabla_x J(x) + \nabla_x (\lambda^T c(x))$ ，其中 $\nabla_x (\lambda^T c(x)) = \frac{\partial c}{\partial x} \lambda$ 。
- 第三步：将 $\nabla_x \lambda^T c(x)$ 写成 $\frac{\partial c}{\partial x} \lambda$ ，并将 $\frac{\partial c}{\partial x} \Delta \lambda$ 保留。
- 第四步：将 $\lambda + \Delta \lambda$ 合并为 λ_{new} ，即 $\lambda_{\text{new}} = \lambda + \Delta \lambda$ ，从而合并相关项。

与Lecture4中不同的是，我们将 $\Delta \lambda$ 换成了 λ 来方便将右边的 ∇L 换成 ∇J 方便理解。

将KKT系统中的各个变量（如 H 、 $c(z)$ 等）用本问题的具体表达式替换进去。也就是把一般形式的KKT系统，具体化为ILC问题的形式。比如 H 是二次代价的Hessian， $c(z)$ 是动力学约束， $\nabla_x J$ 是代价函数对状态的梯度等。并且采用高斯牛顿近似L的Hessian阵，得到KKT系统

$$\begin{bmatrix} H & \frac{\partial c(z)}{\partial z}^T \\ \frac{\partial c(z)}{\partial z} & 0 \end{bmatrix} \begin{bmatrix} \Delta z \\ \lambda \end{bmatrix} = \begin{bmatrix} -\nabla_x J \\ -c(z) \end{bmatrix}$$

其中

$$H = \begin{bmatrix} Q & 0 & \cdots & 0 \\ 0 & R & \cdots & 0 \\ 0 & 0 & Q & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ 0 & 0 & 0 & Q_N \end{bmatrix}, \quad c(z) = \begin{bmatrix} f(x_1, u_1) - x_2 \\ f(x_2, u_2) - x_3 \\ \vdots \\ f(x_{N-1}, u_{N-1}) - x_N \end{bmatrix}\tag{4}$$

从(4)中我们可以看出，ILC实际做的事跟普通轨迹优化问题很像，但是为了简化，我们计算

$$\left. \frac{\partial c}{\partial z} \right|_{x_n, u_n} \approx \left. \frac{\partial c}{\partial z} \right|_{\bar{x}_n, \bar{u}_n}$$

即可以一直在参考轨迹附近近线性化系统，这样子左边的矩阵就会变成一个常值矩阵了。可以这样做是因为我们假设，最优的轨迹会在参考轨迹附近，ILC只是做一些稍微的修正作用而已。

注意(4)中使用的 f 其实是nominal model，因为我们实际上不知道 f_{real} ，也不知道其梯度，所以我们用一个粗糙的model来模拟这个过程，但是在evaluate J 的梯度时，需要一个特定的 x, u 这个值就是实机rollout出来的。

备注

假如 $c(z)$ 中使用的就是 f_{real} 而不是 $f_{nominal}$ 那么 $c(z) = 0$ 。可以在ILC过程中额外加入一些约束，如力矩，然后继续修正它。

Algorithm 11: Iterative Learning Control (ILC)

```

Input: Nominal trajectory  $\bar{x}, \bar{u}$ 
while  $\|x - \bar{x}\| > tol$  do
     $(x_n, u_n) \leftarrow \text{rollout}(x_0, \bar{u});$  // On real system
     $(\Delta x, \Delta u) \leftarrow \arg \min_{\Delta x, \Delta u} J(\Delta x, \Delta u)$ 
        s.t.  $\Delta x_{k+1} = A_k \Delta x_k + B_k \Delta u_k, u_{min} \leq u \leq u_{max};$ 
    // Linearized Dynamics, QP
     $\bar{u} \leftarrow \bar{u} + \Delta u;$  // Only update  $\bar{u}$ 

```

10.1.1 ILC的收敛性

在ILC中，我们每次迭代都用实际系统的rollout误差来修正控制输入。虽然每次修正用到的模型是近似的（如线性化模型或名义模型），但只要每次更新的方向大致沿着下降方向，ILC通常都能收敛到一个较优的解。

更正式地说，假设我们每次迭代的更新满足如下条件（类似于信赖域或Armijo条件）：

$$\left\| f(x) + \frac{\partial f}{\partial x} \Delta x \right\| \leq \eta \|f(x)\| \quad (234)$$

其中 $f(x)$ 为轨迹误差， $\frac{\partial f}{\partial x}$ 为模型的雅可比， Δx 为本次修正量， $\eta \in (0, 1)$ 为收敛因子。

只要每次修正都能保证误差有收敛趋势（即 $\eta < 1$ ），则经过多次迭代后，误差将逐步减小，最终收敛到一个较小的范围内。实际应用中，ILC常与line search等技术结合，以确保每步更新都能带来误差的下降，从而保证整体的收敛性。

结论： ILC的收敛性依赖于每次迭代的修正方向和步长选择。只要修正方向大致正确，并通过步长控制避免发散，ILC通常都能在有限次迭代内收敛到较优的跟踪性能。

10.2 ILC说明

案例：基于OSQP的ILC实现注意事项

在实际代码中，ILC的实现通常采用类似SQP（Sequential Quadratic Programming，序列二次规划）的方法。每次迭代都需要求解一个最优的QP（Quadratic Programming，二次规划）问题，然后根据QP的结果继续修正（在实际运行时，通常需要多次执行OSQP优化，直到最终获得满意的效果）。

ILC实现中的关键注意点

- **线性化点的选择:** OSQP^a只能处理线性约束，因此需要对系统进行线性化。线性化点始终选在 $x_{nominal}$ ，即名义轨迹附近进行修正。
- **优化变量的定义:** OSQP问题的优化变量为 Δz ，一次项对应的值为 $x_{traj} - x_{opt}$ 。结合二次项，可以保证最优解 $\Delta x = x_{opt} - x_{traj}$ ，约束为 $x_{next} = x_{traj} + \Delta x = x_{opt}$ 。
- **初始值设定:** 由于优化变量为 Δz ，初始值应设为 $\Delta x_0 = 0$ 。
- **约束的正确性:** 优化变量为 Δz ，但对 u 的约束是在非 Δ 下的，因此构建OSQP问题时要对约束进行正确修正。

^aOperator Splitting Quadratic Program (OSQP) 是一种用于求解二次规划 (Quadratic Programming, QP) 问题的优化算法和开源求解器。其核心思想是采用运算符分裂 (Operator Splitting) 技术，将原始的QP问题分解为更易于处理的子问题，通过交替方向乘子法 (ADMM, Alternating Direction Method of Multipliers) 等方法高效迭代求解。OSQP特别适合处理带有稀疏结构和大规模约束的QP问题，具有数值稳定、收敛性好、易于嵌入等优点，因此在控制、机器学习和信号处理等领域被广泛应用。

简要流程总结:

1. 在名义轨迹 $x_{nominal}, u_{nominal}$ 附近线性化系统动力学，构建QP问题。
2. 以 Δz 为优化变量，设定目标和约束，利用OSQP等QP求解器获得 Δz^* 。
3. 用 Δz^* 修正名义轨迹，得到新的控制输入。
4. 在真实系统上rollout，记录实际轨迹，重复上述过程，直至收敛。

这种方法能够有效地利用模型和实际数据的结合，逐步提升轨迹跟踪精度，是工业机器人等重复性任务中常用的高效前馈补偿手段。

10.3 倒立摆摆起控制案例：处理模型不确定性和噪声扰动

考虑一个倒立摆摆起问题，其中存在显著的模型不确定性和非线性摩擦扰动。系统状态为 $x = [p, \theta, \dot{p}, \dot{\theta}]^T$ ，其中 p 为小车位置， θ 为摆杆角度，控制输入为作用在小车上的水平力 u 。目标是将摆杆从下垂位置摆起至倒立位置。

问题描述:

$$\begin{aligned} \min_{x_{1:N}, u_{1:N}} \quad & J = \sum_{k=1}^{N-1} \left[\frac{1}{2} (x_k - x_{ref,k})^T Q (x_k - x_{ref,k}) + \frac{1}{2} u_k^T R u_k \right] \\ & + \frac{1}{2} (x_N - x_{ref,N})^T Q_f (x_N - x_{ref,N}) \end{aligned} \quad (235)$$

$$\text{s.t. } x_{k+1} = f(x_k, u_k), \quad k = 1, \dots, N-1 \quad (236)$$

$$|u_k| \leq u_{max} = 5.0 \text{ N}, \quad k = 1, \dots, N-1 \quad (237)$$

$$x_0 = [0, 0, 0, 0]^T, \quad x_{ref,N} = [0, \pi, 0, 0]^T \quad (238)$$

其中 $Q = \text{diag}(1.0, 1.0, 1.0, 1.0)$, $Q_f = 100 \cdot I_4$, $R = 0.1$ 。

真实系统与名义模型的差异:

名义动力学模型采用标准倒立摆参数，而真实系统包含以下扰动：

- 参数不确定性: $m_c = m_{c,nom} + 0.02$, $m_p = m_{p,nom} - 0.01$, $l = l_{nom} + 0.005$
- 非线性摩擦力: $F_{friction} = [0.01; 0.01] \cdot \tanh(5.0 \cdot \dot{q})$

真实动力学为:

$$H(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) + F_{friction}(\dot{q}) = Bu \quad (239)$$

其中 $F_{friction}(\dot{q}) = [0.01; 0.01] \cdot \tanh(5.0 \cdot \dot{q})$

方法一：直接RK4积分

使用四阶龙格-库塔法对名义动力学模型进行数值积分:

$$\begin{aligned} f_1 &= f_{nom}(x_k, u_k) \\ f_2 &= f_{nom}\left(x_k + \frac{h}{2}f_1, u_k\right) \\ f_3 &= f_{nom}\left(x_k + \frac{h}{2}f_2, u_k\right) \\ f_4 &= f_{nom}(x_k + hf_3, u_k) \\ x_{k+1} &= x_k + \frac{h}{6}(f_1 + 2f_2 + 2f_3 + f_4) \end{aligned} \quad (240)$$

直接应用最优控制序列到含噪声的真实系统时，由于模型失配和摩擦扰动，会产生显著的跟踪误差。

方法二：LQR跟踪控制

基于名义模型设计反馈控制器:

$$K_k = (R + B_k^T P_{k+1} B_k)^{-1} B_k^T P_{k+1} A_k \quad (241)$$

$$P_k = Q + K_k^T R K_k + (A_k - B_k K_k)^T P_{k+1} (A_k - B_k K_k) \quad (242)$$

其中 $A_k = \frac{\partial f_{nom}}{\partial x} \Big|_{x_{ref,k}, u_{ref,k}}$, $B_k = \frac{\partial f_{nom}}{\partial u} \Big|_{x_{ref,k}, u_{ref,k}}$

控制律为:

$$u_k = \text{sat}(u_{ref,k} - K_k(x_k - x_{ref,k}), -u_{max}, u_{max}) \quad (243)$$

虽然反馈能部分补偿扰动，但由于模型失配，仍存在稳态误差。

方法三：迭代学习控制 (ILC)

ILC通过多次rollout学习扰动模式，逐步修正前馈轨迹。每次迭代求解:

$$\min_{\Delta z} \quad \frac{1}{2} \Delta z^T H \Delta z + q^T \Delta z \quad (244)$$

$$\text{s.t.} \quad D \Delta z = 0 \quad (245)$$

$$u_{min} \leq u_{current} + \Delta u \leq u_{max} \quad (246)$$

其中线性项基于实际rollout误差构建:

$$q_k = \begin{bmatrix} 0 \\ Q_{ilc}(x_{actual,k} - x_{ref,k}) \end{bmatrix} \quad (247)$$

ILC权重矩阵针对摩擦扰动进行调整:

$$Q_{ilc} = \text{diag}(0.01, 0, 1.0, 0), \quad R_{ilc} = 0.1 \quad (248)$$

其中角度误差权重较大，而位置和速度权重较小，以适应摩擦主要影响速度状态的特点。

ILC更新规则：

$$u_{new} = u_{current} + \Delta u \quad (249)$$

经过5-6次ILC迭代后，前馈轨迹能够有效补偿模型不确定性和非线性摩擦，实现高精度轨迹跟踪，即使在开环执行时也能维持良好的跟踪性能。

11 随机最优控制与LQG理论

本章介绍了随机最优控制的基本思想及LQG问题的解析推导，包括最优控制器与最优观测器（Kalman滤波），并讨论了分离原理及其实际应用。

11.1 随机最优控制基础

11.2 LQG控制器部分

11.3 LQG最优观测器部分（卡尔曼滤波）

由于**Kalman Filter**是一个非常大的话题，这里我将KF/EKF/UKF等放在了附录中，包含证明和使用，请参看??。

11.4 KF与LQR的对偶性

在控制理论中，卡尔曼滤波（KF）与线性二次调节器（LQR）之间存在着深刻的合作关系。这种对偶性不仅在数学形式上表现出来，更在设计思想和求解方法上体现了内在的一致性。

11.4.1 数学形式的对偶性

考虑LQR问题的代价函数：

$$J = \frac{1}{2} \int_0^T (x^T Q x + u^T R u) dt + \frac{1}{2} x_T^T S x_T \quad (250)$$

其中，状态方程为：

$$\dot{x} = Ax + Bu \quad (251)$$

对应的Riccati方程为：

$$-\dot{P} = A^T P + PA - PBR^{-1}B^T P + Q \quad (252)$$

而在卡尔曼滤波中，协方差传播方程为：

$$\dot{\Sigma} = A\Sigma + \Sigma A^T + GQG^T - \Sigma C^T R^{-1} C \Sigma \quad (253)$$

通过变量替换 $P \leftrightarrow \Sigma$, $A \leftrightarrow A^T$, $B \leftrightarrow C^T$, $Q \leftrightarrow GQG^T$, 可以看出两个方程在结构上完全对偶。

11.4.2 物理意义的对偶性

- **LQR:** 寻找最优控制输入，使系统状态的偏差最小
- **KF:** 寻找最优状态估计，使估计误差的协方差最小

这种对偶性表明，控制问题中的”向前传播”（前向时间）与估计问题中的”向后传播”（信息流向）在数学本质上是等价的。

参数类型	LQR控制器	Kalman滤波器
状态权重	Q	R^{-1} (测量噪声的逆)
控制权重	R	Q^{-1} (过程噪声的逆)
系统矩阵	A	A^T
输入矩阵	B	C^T

表 3: LQR与KF参数的对偶关系

11.4.3 设计参数的对偶关系

这种对偶性在LQG问题的分离原理中得到了完美体现：最优控制器和最优观测器可以独立设计，然后简单地串联使用。

11.5 案例分析

本节通过一个具体的Julia实现来演示LQG控制器的设计与仿真。我们考虑一个二维离散时间积分器系统，其中只能观测到位置信息，需要同时进行状态估计和最优控制。

11.5.1 系统模型与参数设置

考虑离散时间系统：

$$x_{k+1} = Ax_k + Bu_k + w_k \quad (254)$$

$$y_k = Cx_k + v_k \quad (255)$$

其中状态转移矩阵 $A = \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix}$ 表示离散化的积分器模型，控制输入矩阵 $B = \begin{bmatrix} 0.5h^2 \\ h \end{bmatrix}$ ，观测矩阵 $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$ （仅观测位置）。

系统噪声协方差 $W = BB^T \cdot 0.1 + 10^{-5}I$ ，观测噪声协方差 $V = 0.1$ 。

11.5.2 LQR控制器设计

通过求解离散代数Riccati方程获得最优反馈增益：

$$P = A^T PA - A^T PB(R + B^T PB)^{-1}B^T PA + Q \quad (256)$$

最优控制律为： $u_k = -Kx\Theta_k$ ， 其中 $K = (R + B^T PB)^{-1}B^T PA$ 。

11.5.3 卡尔曼滤波器实现

卡尔曼滤波包含预测和更新两个步骤：

预测步骤：

$$\tilde{x}_k = A\hat{x}_{k-1} + Bu_{k-1} \quad (257)$$

$$\tilde{\Sigma}_k = A\Sigma_{k-1}A^T + W \quad (258)$$

更新步骤：

$$z_k = y_k - C\tilde{x}_k \quad (259)$$

$$S_k = C\tilde{\Sigma}_k C^T + V \quad (260)$$

$$L_k = \tilde{\Sigma}_k C^T S_k^{-1} \quad (261)$$

$$\hat{x}_k = \tilde{x}_k + L_k z_k \quad (262)$$

$$\Sigma_k = (I - L_k C) \tilde{\Sigma}_k (I - L_k C)^T + L_k V L_k^T \quad (263)$$

仿真验证了LQG控制器的分离原理：卡尔曼滤波器和LQR控制器可以独立设计并串联使用，同时保持整体系统的最优性。协方差矩阵的收敛过程展现了估计精度的逐步提升，而控制性能则体现在状态轨迹向目标状态的稳定收敛。

11.6 附录：多维高斯分布的性质

12 Robust Control

因为LQR+KF对噪声敏感，我们需要一个能处理噪声同时又能保证系统稳定的控制器，这就是鲁棒控制（Robust Control）。2025年的CMU 16-745课程中已经将该章节移除。

作为关键词记录几种Robust Control的算法：

- H-infinity Control
- μ -Synthesis
- Loop Shaping
- Sliding Mode Control

Robust Control是一个非常大的子领域，其复杂程度不亚于最优控制，所以我们只介绍其中的一种算法：Minimax DDP（Minimax Dynamic Programming）。

13 final part

13.1 凸松弛 (Convex Relaxation) 和如何降落火箭

火箭建模

- 刚体模型 (Rigid-body model) 用于慢速控制位置:

$$\begin{aligned}\dot{v} &= -g + \frac{T}{m} \\ \dot{m} &= -\alpha T\end{aligned}$$

其中 v 为速度, g 为重力加速度, T 为推力, m 为质量, α 为燃料消耗系数。

- 姿态动力学 (Attitude dynamics) 用于高速控制姿态:

$$J\dot{\omega} + \omega \times J\omega = l \times T$$

其中 J 为转动惯量, ω 为角速度, l 为力臂。

Fluid Sloshing (液体晃动) 是火箭在降落过程中燃料晃动导致的姿态扰动, 是一个非常复杂的非线性问题。将其建模为一个 l_p 和 m_p 的倒立摆比建模成一个弹簧更有效。

柔性模态 (Flexible modes): 火箭为了减重, 结构通常较轻, 因此整体刚度较低, 容易出现低频的弯曲模态。这些柔性模态会影响火箭的姿态和轨迹控制, 尤其在高速飞行和降落阶段更为明显。需要通过结构设计和控制算法 (如模态阻尼、主动控制等) 来抑制这些低频振动, 保证飞行稳定性。

气动力 (Aerodynamic Forces): 通常在火箭降落问题中被忽略。可以通过在位置控制器中加入速度约束, 确保速度较小, 从而使气动力影响减小。

13.1.1 Convex Relaxation

使用一个更大的凸集来替换原来的非凸集, 叫做**凸松弛** (Convex Relaxation)。

例如对于 $S_1 = \{x \in \mathbb{R}^n | x^2 \leq 1\}$, 可以用 $S_2 = \{x \in \mathbb{R}^n | x \leq 1\}$ 来替换, 记 $S_1 = \partial S_2$ 是 S_2 的边界。

有时候, 如果目标函数是“好的”(如线性或凸), 通过求解松弛后的问题, 仍然可以得到原问题的解。

例如, 考虑如下优化问题:

$$\begin{aligned}\min \quad & c^T x \\ \text{s.t.} \quad & \|x\| = 1\end{aligned}$$

我们可以将约束 $\|x\| = 1$ 松弛为 $\|x\| \leq 1$, 得到凸松弛问题:

$$\begin{aligned}\min \quad & c^T x \\ \text{s.t.} \quad & \|x\| \leq 1\end{aligned}$$

此时, 最优解依然会落在 $\|x\| = 1$ 的边界上, 因此松弛后的解与原问题一致。这种情况称为**tight relaxation** (紧松弛)。

当松弛后的最优解恰好也满足原约束时, 凸松弛是有效的; 否则只能得到原问题的下界或近似解。

对于上述火箭问题, 我们有如下几种限制:

- 最大推力限制: $\|T\| \leq T_{\max}$, 这是一个凸集。
- 推力角度限制: $\frac{n^T T}{\|T\|} \leq \cos(\theta_{\max})$, 这是一个SOCP (Second Order Cone Programming) 限制, 也是一个凸集。
- 最小推力限制: $\|T\| \geq T_{\min}$, 这是一个非凸集。

Slack Variable (松弛变量) 可以用来将非凸集转化为凸集。加入一个新的松弛变量 Γ , 使得

$$\begin{aligned}\|T\| &= \Gamma \\ T_{\min} &\leq \Gamma \leq T_{\max} \\ n^T T &\leq \Gamma \cos(\theta_{\max})\end{aligned}$$

其中第一个等式 $\|T\| = \Gamma$ 是非凸的。我们可以将其松弛为 $\|T\| \leq \Gamma$, 这样所有约束都变成了凸集:

$$\begin{aligned}\|T\| &\leq \Gamma \\ T_{\min} &\leq \Gamma \leq T_{\max} \\ n^T T &\leq \Gamma \cos(\theta_{\max})\end{aligned}$$

这样就完成了凸松弛, 所有约束都可以用SOCP等凸优化方法求解。

论文中, 他们使用的某些cost function, 例如燃料最小化的这些, 结果证明是tight的, 注意看论文中的证明!

13.2 如何走路

略, 过程过于抽象且没有给出例子。简而言之是将非凸的运动学约束转化为QP问题。

13.3 自动驾驶和博弈论

13.4 数据驱动和行为克隆

13.5 强化学习

在2023之后的课程已经被移除, RL作为非常大的内容, 不适合在最优控制中进行讨论。2025的课程作为Guest Talk简述了一下四足机器人的MPPI (Model Predictive Path Integral Control) 方法: 从某个分布中采样一组动作, 评估这些采样并选择表现最好的一个, 用最优样本更新分布的均值, 重复上述过程。

14 机器人中的数值优化引言

从这一节开始, 我们将转向机器人中的数值优化。我们将介绍一些基本的概念和方法, 继续前面所论述的最优控制内容。

本课程将围绕以下几个主题展开:

- **数值优化基础:** 介绍优化问题的基本形式、约束类型、凸性等核心概念。

- **无约束优化方法:** 包括梯度下降法、牛顿法、拟牛顿法等常用算法及其收敛性分析。
- **约束优化方法:** 涵盖拉格朗日乘子法、KKT条件、罚函数法、投影法等。
- **凸优化与非凸优化:** 讲解凸优化的理论基础及其在机器人中的应用，讨论非凸问题的挑战与常用求解策略。
- **数值优化在机器人中的应用:** 如轨迹规划、参数估计、点云配准、路径平滑等实际案例。
- **现代优化工具与软件:** 介绍常用的数值优化库和求解器（如CVX、Gurobi、OSQP等），并结合实际问题进行演示。

推荐书目：

- **《最优化：建模算法与理论》:** 适合初学者，涵盖线性和非线性优化问题的求解方法。
- **Numerical Optimization,** Nocedal 和 Wright: 深入探讨数值优化的理论和算法，包括梯度下降、牛顿法等经典方法。
- **Lectures on Convex Optimization,** Bertsekas: 理论清晰，涵盖光滑与非光滑优化。
- **Lectures On Modern Convex Optimization:** 分析、算法与工程应用。

14.1 优化问题的基本形式

优化问题通常可以表述为：

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & g(x) \leq 0 \\ & h(x) = 0 \end{aligned}$$

其中，

- $x = (x_1, \dots, x_n) \in \mathbb{R}^n$: 优化变量
- $f : \mathbb{R}^n \rightarrow \mathbb{R}$: 目标函数
- $g : \mathbb{R}^n \rightarrow \mathbb{R}^{m_g}$: 不等式约束函数
- $h : \mathbb{R}^n \rightarrow \mathbb{R}^{m_h}$: 等式约束函数

最优解 x^* 是在所有满足约束条件的向量中，使目标函数 f 取得最小值的解。其中 $f(x)$ 要满足 1. lower bounded, 2. 有 bounded level set, 即 $f(x) \leq \beta$ 的解集是有界的。

在机器人学中，最优化常用于：平滑和映射（用非线形最小二乘），轨迹规划（使用非线形规划），点云配准（半正定规划），时间优化的参数化（二阶Conic Program）。

下面是课程安排：

1. 第一次课：数值优化基础

- (a) 数学规划与机器人
- (b) 非凸优化中的凸性
- (c) 凸集与凸函数
- (d) 无约束非凸优化
- (e) 线搜索最速梯度下降
- (f) 修正阻尼牛顿法

2. 第二次课：无约束优化进阶

- (a) 拟牛顿法工程上有效的无约束优化方法
- (b) BFGS 更新
- (c) 强/弱曲率条件
- (d) Li-Fukushima 全局收敛性：介绍Li-Fukushima方法在无约束优化中的全局收敛性理论，分析其与传统牛顿法的异同及在实际问题中的应用。
- (e) 有限记忆 BFGS 轻量化
- (f) Lewis-Overton 非光滑线搜索
- (g) 线性共轭梯度下降绕过大矩阵
- (h) 牛顿-CG 方法
- (i) 应用：平滑导航路径生成

3. 第三次课：约束优化方法

- (a) 约束优化问题的分类与复杂性
- (b) 低维线性规划：Seidel 算法
- (c) 低维严格凸二次规划
- (d) 顺序无约束极小化技术 (SUMT)
- (e) SUMT：罚函数法与障碍法
- (f) SUMT：拉格朗日松弛与Uzawa方法
- (g) Karush-Kuhn-Tucker (KKT) 条件
- (h) 增广拉格朗日方法 (PHR)
- (i) 应用1：控制分配问题
- (j) 应用2：碰撞距离计算
- (k) 应用3：非线性模型预测控制

4. 第四次课：对称锥优化

- (a) 锥与对称锥
- (b) 对称锥与欧几里得Jordan代数
- (c) 锥的可表性
- (d) 对称锥的增广拉格朗日法
- (e) 半光滑牛顿法

(f) 应用：时间最优路径参数化

5. 第五次课：问题建模与求解技巧

- (a) 平滑化技术
- (b) 自由度与伴随法
- (c) 线性求解器的分类与特性
- (d) 项目：密集障碍环境下的安全导航

15

机器人中的数值优化第一章——数值优化基础

15.1 凸集与非凸集

定义 15.1 (凸集). 如果对于任意两个点 $x, y \in C$, 以及任意 $\lambda \in [0, 1]$, 都有 $\lambda x + (1 - \lambda)y \in C$, 则称 C 是一个凸集。

更一般的, 凸集中的所有点的任意凸组合也在该凸集中:

$$\sum \theta_i x_i, \sum \theta_i = 1, \theta_i \leq 0 \quad (264)$$

convex hull: 选一个最小的凸集能将原来非凸集包起来。

常见集合的凸性举例

- 超平面 (Hyperplane): $\{x \mid a^T x = b\}$ 是凸集。因为任意两点的凸组合仍满足线性等式。
- 半空间 (Half-space): $\{x \mid a^T x \geq b\}$ 是凸集。因为线性不等式的解集是凸的。
- 球体 (Sphere): $\{x \mid \|x - x_0\| = b\}$ 不是凸集。因为两个在球面上的点的连线一般不在球面上。
- 球 (Ball): $\{x \mid \|x - x_0\| \leq b\}$ 是凸集。因为范数是凸函数, 其下水平集是凸的。
- 多项式零点集 (Polynomials): $\{f \mid f = \sum_i a_i x^i\}$ 一般不是凸集, 除非对系数或函数有特殊限制。

定义 15.2 (锥 (Cone)). 如果集合 C 满足: 对于任意 $x \in C$ 和任意 $a \geq 0$, 都有 $ax \in C$, 则称 C 是一个锥集 (cone)。

例子 15.3 (二阶锥). 二阶锥 (Second-order cone) 定义为

$$C_2 = \{(x, t) \mid \|x\| \leq t\} \subset \mathbb{R}^{n+1}$$

其中 $x \in \mathbb{R}^n, t \in \mathbb{R}$ 。

定义 15.4 (半正定锥 (Semi-definite cone)). 半正定锥 S_+^n 定义为所有对称且半正定的 $n \times n$ 实矩阵的集合:

$$S_+^n = \{A \in \mathbb{R}^{n \times n} \mid A = A^T, A \succeq 0\}$$

¹⁰ 其中 $A \succeq 0$ 表示 A 是半正定的, 即对任意 $x \in \mathbb{R}^n$, 有 $x^T A x \geq 0$ 。

注释 15.5. 半正定锥是一个凸锥: 若 $A, B \in S_+^n$, 且 $a, b \geq 0$, 则 $aA + bB \in S_+^n$ 。

证明. 由于 A, B 都是对称半正定矩阵, $a, b \geq 0$, 则 $aA + bB$ 也是对称矩阵。对任意 $x \in \mathbb{R}^n$, 有

$$x^T(aA + bB)x = a x^T A x + b x^T B x \geq 0$$

因为 $x^T A x \geq 0$, $x^T B x \geq 0$, 且 $a, b \geq 0$ 。因此 $aA + bB$ 也是半正定矩阵, 即 $aA + bB \in S_+^n$ 。 \square

¹⁰ 这里矩阵半正定等定义和判定见矩阵分析课程, 此处略。

15.1.1 凸集的基本性质

- 任意个凸集的交集仍为凸集。即 $\bigcap_i C_i$ 是凸集，只要每个 C_i 都是凸集。
- 凸集的并集一般不是凸集。只有在所有集合两两包含时，才有可能为凸集。
- 仿射变换保持凸性。若 C 是凸集， A 为线性变换， b 为向量，则 $AC+b = \{Ax+b \mid x \in C\}$ 也是凸集。
- 凸集的闭包、内部、相对内部仍为凸集。
- 凸锥的非负缩放仍为凸锥。
- 凸集的Minkowski和（加法）一般是凸集。设 C_1, C_2 是凸集，则 $C_1 + C_2 = \{x + y \mid x \in C_1, y \in C_2\}$ 也是凸集。
- 笛卡尔积保持凸性。若 $A \subset \mathbb{R}^n$ 、 $B \subset \mathbb{R}^m$ 都是凸集，则 $A \times B$ 也是凸集。

证明. 设 $(a_1, b_1), (a_2, b_2) \in A \times B$, 任取 $\lambda \in [0, 1]$, 则

$$\lambda(a_1, b_1) + (1 - \lambda)(a_2, b_2) = (\lambda a_1 + (1 - \lambda)a_2, \lambda b_1 + (1 - \lambda)b_2)$$

由于 A 是凸集, $\lambda a_1 + (1 - \lambda)a_2 \in A$; B 是凸集, $\lambda b_1 + (1 - \lambda)b_2 \in B$ 。因此

$$(\lambda a_1 + (1 - \lambda)a_2, \lambda b_1 + (1 - \lambda)b_2) \in A \times B$$

所以 $A \times B$ 是凸集。 \square

- 凸集的Hadamard乘积（逐元素乘法）一般不是凸集。只有在特殊情况下（如所有元素非负等）才可能保持凸性。

15.1.2 半正定锥的凸性与几何形状

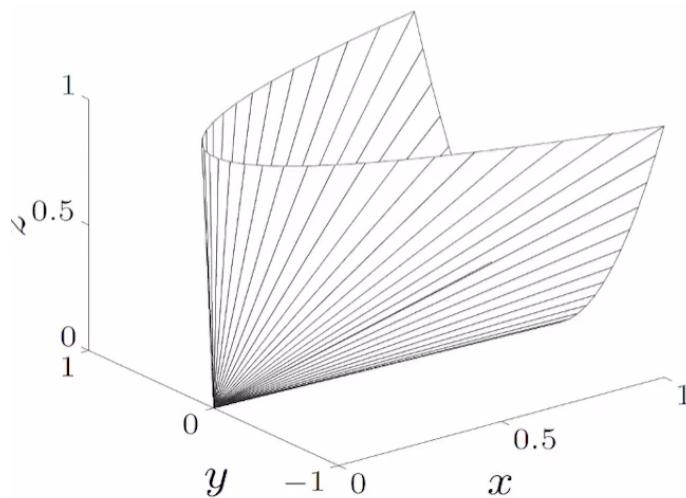


图 19: 半正定锥 S^n_+ 的几何形状示意图。

半正定锥的定义与凸性 半正定锥 S_+^n 可以用如下方式刻画：

$$S_+^n = \bigcap_{x \in \mathbb{R}^n} \{A \mid x^T A x \geq 0\}$$

即所有对称矩阵 A , 使得对任意 x 都有 $x^T A x \geq 0$ 。

凸性分析 半正定锥是凸集。因为对于任意 $A, B \in S_+^n$, 以及任意 $\theta \in [0, 1]$, 有

$$x^T (\theta A + (1 - \theta)B) x = \theta x^T A x + (1 - \theta)x^T B x \geq 0$$

因此 $\theta A + (1 - \theta)B \in S_+^n$ 。

几何直观 如上图所示, 半正定锥在低维空间中呈现“锥形”结构, 是一个凸锥。该集合在优化中非常重要, 广泛用于半正定规划 (SDP) 等问题。

15.2 高阶函数

梯度、Jacobian与Hessian

- **梯度 (Gradient):** 对于标量函数 $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$, 梯度是所有一阶偏导数组成的向量:

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

- **Jacobian矩阵 (Jacobian):** 对于向量值函数 $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m$, Jacobian是所有一阶偏导数组成的 $m \times n$ 矩阵:

$$J_f(x) = \frac{\partial f}{\partial x^T} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

- **Hessian矩阵 (Hessian):** 对于标量函数 $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$, Hessian是所有二阶偏导数组成的对称矩阵:

$$H_f(x) = \nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

高阶函数的泰勒展开 设 $f : \mathbb{R}^n \rightarrow \mathbb{R}$ 是光滑函数, $x \in \mathbb{R}^n$, 其泰勒展开为:

$$f(x) = f(0) + x^T \nabla f(0) + \frac{1}{2} x^T \nabla^2 f(0) x + O(\|x\|^3)$$

该展开在数值优化中用于函数的线性和二次近似。

15.3 高阶函数的信息与微分记号

矩阵与向量的导数类型 对于不同类型的输入输出，常见的导数记号如下表所示：

X	Y	G	记号	名称
\mathbb{R}	\mathbb{R}	\mathbb{R}	$f'(x)$	导数
\mathbb{R}^n	\mathbb{R}	\mathbb{R}^n	$\frac{\partial f}{\partial x_i}$	梯度
\mathbb{R}^n	\mathbb{R}^m	$\mathbb{R}^{n \times m}$	$\frac{\partial f_i}{\partial x_j}$	Jacobian
$\mathbb{R}^{m \times n}$	$\mathbb{R}^{m \times n}$	$\mathbb{R}^{m \times n}$	$\frac{\partial f}{\partial x_{ij}}$	元素导数

微分记号的常用性质 在矩阵分析和优化中，微分记号 d 有如下常用性质：

$$\begin{aligned}
 dA &= 0 \\
 d(\alpha X) &= \alpha dX \\
 d(AXB) &= A dX B \\
 d(X + Y) &= dX + dY \\
 d(X^\top) &= (dX)^\top \\
 d(XY) &= (dX)Y + X(dY) \\
 d\langle X, Y \rangle &= \langle dX, Y \rangle + \langle X, dY \rangle \\
 d\left(\frac{X}{\phi}\right) &= \frac{\phi dX - (d\phi)X}{\phi^2} \\
 d \operatorname{tr} X &= I \\
 df(g(x)) &= \frac{df}{dg} \cdot dg(x)
 \end{aligned}$$

这些性质在推导链式法则、矩阵函数的导数等问题时非常有用。

下面给出例子，使用逐元素求导和使用公式求导：

例子 15.6 (逐元素求导). 已知 $f(x) = \frac{1}{2}x^T Ax + b^T x + c$, 其中 $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$, $c \in \mathbb{R}$ 。求 $\nabla f(x)$ 。

解：展开 $f(x)$ 得

$$f(x) = \frac{1}{2} \sum_{i,j=1}^n x_i a_{ij} x_j + \sum_{i=1}^n b_i x_i + c$$

对 x_i 求偏导：

$$\frac{\partial f(x)}{\partial x_i} = \frac{1}{2} \sum_{j=1}^n (a_{ij} + a_{ji}) x_j + b_i$$

因此

$$\nabla f(x) = \frac{1}{2}(A + A^T)x + b$$

例子 15.7 (利用公式求导). 已知 $f(X) = \text{tr}(AX)$, 其中 $A, X \in \mathbb{R}^{n \times n}$ 。求 $\nabla f(X)$ 。

解: 利用矩阵求导公式 $\nabla_X \text{tr}(AX) = A^T$, 所以

$$\nabla f(X) = A^T$$

15.4 凸函数及其性质

凸函数是满足Jesen不等式的函数, 即

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y), \quad \forall x, y, \theta \in [0, 1] \quad (265)$$

即对函数任意两点连起来, 两点之间的函数值不超过两点函数值的线性组合。如果严格取不等号则为strictly convex function (严格凸函数)

对于不等式方向, 又有concave function (凹函数)。

15.4.1 凸函数的上方图 (Epigraph)

定义 15.8 (上方图 (Epigraph)). 给定函数 $f : \mathbb{R}^n \rightarrow \mathbb{R}$, 其上方图定义为

$$\text{epi}(f) = \{(x, y) \mid f(x) \leq y\}$$

即所有在图像上方 (含图像本身) 的点的集合。

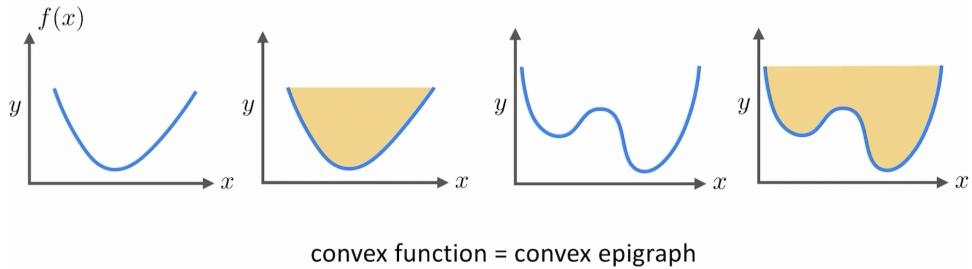


图 20: 凸函数的上方图 (Epigraph)。左: 凸函数的上方图是凸集; 右: 非凸函数的上方图不是凸集。

性质

- 函数 f 是凸函数, 当且仅当其上方图 $\text{epi}(f)$ 是凸集。
- 直观理解: 凸函数的图像“开口向上”, 其上方图不会出现“凹陷”。

凸函数的sub level set (下水平集) 定义为

$$\text{sub}(f) = \{x \mid f(x) \leq c\}$$

是凸集当且仅当 f 是凸函数。

具有凸的下水平集的函数不一定是凸函数, 比如 $f(x) = \log(|x| + .1)$, 这是quasi-convex函数, 但不是凸函数。quasi-convex函数的下水平集是凸集, 但上方图不一定是凸集。

凸的性质很容易被保留下来, 而quasi-convex的性质不容易被保留下来。

凸函数的局部最小值也是全局最小值，这就是为什么我们要研究凸函数的原因。

例： $f(x) = x + x^2 + x^3 + |x|$ 是凸集。

各种范数也是凸的，比如 L_1 范数、 L_2 范数等。范数满足三角不等式，即 $\|x+y\| \leq \|x\| + \|y\|$ ，则有 $\theta a + (1-\theta)b \leq \theta\|a\| + (1-\theta)\|b\|$ ，因此范数是凸函数。

仿射变换也是凸的，即若 $f(x)$ 是凸函数，则 $g(x) = f(Ax+b)$ 也是凸函数，其中 A 是线性变换， b 是平移向量。

点对点最大值操作与凸性保持 点对点最大值操作 (point-wise max) 可以保持凸性。设有一组凸函数 $f_i(x)$ ，定义

$$g(x) = \max_i f_i(x)$$

则 $g(x)$ 也是凸函数。

常见例子

- **绝对值：** $|x| = \max\{x, -x\}$ ，是两个线性函数的最大值，因此是凸函数。
- **无穷范数：** $\|x\|_\infty = \max_i |x_i|$ ，是若干个凸函数 ($|x_i|$) 的最大值，因此是凸函数。
- **最大特征值：** $\|A\|_2 = \max_v v^T A v$ ，是关于 A 的凸函数 (A 为对称矩阵时)。

上方图的凸性是否保持？ 点对点最大值操作不仅保持函数的凸性，也保持上方图 (epigraph) 的凸性。因为若 $f_i(x)$ 的上方图都是凸集，则

$$\text{epi}(g) = \bigcap_i \text{epi}(f_i)$$

交集仍为凸集，因此 $g(x)$ 的上方图也是凸集。

注释 15.9. 点对点最大值操作是构造新凸函数的常用方法，常见于范数、绝对值、最大特征值等场景。

15.5 为什么这些函数是凸的？

- **Trace:** $f(X) = \text{trace}(A^T X)$ 是线性算子，线性函数总是凸的（也是凹的）。
- **Distance over set:** $f(x) = \max_{y \in C} \|x - y\|$ ，对凸集 C ， $\|x - y\|$ 关于 x 是凸函数，对 y 取最大值仍保持凸性（最大值操作保持凸性）。
- **Distance to convex set:** $f(x) = \min_{y \in C} \|x - y\|$ ，这是特殊情形： C 为凸集时， $\|x - y\|$ 关于 x 是凸函数，对 y 取最小值，结果也是凸函数。
- **Affine Norm:** $f(x) = \|b + \sum_i A_i x_i\|_2$ ，仿射变换加范数，范数是凸函数，仿射变换不会破坏凸性，因此整体是凸函数。

一般结论 如果 $g(x, y)$ 关于 x 是凸函数，则对 y 取最大值或最小值（在 y 上凸/凹时）通常能保持凸性。即

$$f(x) = \sup_{y \in C} g(x, y)$$

若 $g(x, y)$ 关于 x 是凸函数，则 $f(x)$ 也是凸函数。

注释 15.10. 这些性质在优化建模中非常重要，常用于构造新的凸目标函数或约束。

对于凸函数，

凸函数的切线性质 凸函数在任意点的切线（线性近似）都在函数图像的下方，即

$$f(y) \geq f(x) + \nabla f(x)^T(y - x), \quad \forall x, y \in \text{dom } f$$

这意味着：凸函数总是位于其线性近似的上方。

该性质是凸优化理论的基础，保证了梯度下降等方法的收敛性和全局最优性。

则有：

一阶最优性条件与全局最优性 对于凸函数，若 x^* 是可微函数 $f(x)$ 的极小点，则有

$$\nabla f(x^*) = 0$$

结合凸函数的切线性质，得到

$$f(y) \geq f(x^*) + \nabla f(x^*)^T(y - x^*) = f(x^*)$$

即对于所有 y , $f(y) \geq f(x^*)$, 因此 x^* 是全局最小点。

注释 15.11. 对于非凸函数, $\nabla f(x^*) = 0$ 只保证 x^* 是驻点，可能是鞍点、局部极小或极大值点。只有凸函数才保证一阶最优性条件对应全局最优。

结论 对于凸函数，任何极小点都是全局极小点。

15.5.1 二阶条件 (Second-order conditions)

凸函数的二阶判据 一个光滑函数 $f(x)$ 是凸函数，当且仅当其Hessian矩阵处处半正定：

$$\nabla^2 f(x) \succeq 0, \quad \forall x$$

即Hessian为半正定矩阵 (semi-definite)。

非凸函数的极小值二阶条件 对于非凸函数，极小点 x^* 满足：

$$\nabla^2 f(x^*) \succeq 0$$

即在极小点处Hessian半正定。

Hessian的作用 Hessian矩阵是光滑函数的良好局部二次近似模型。二阶条件在优化理论和算法中非常重要。

15.5.2 强凸性 (Strong Convexity)

定义 函数 f 是 m -强凸 (m -strongly convex) 的，如果存在 $m > 0$, 使得对任意 x, y , 有

$$f(y) \geq f(x) + (y - x)^T \nabla f(x) + \frac{m}{2} \|y - x\|^2$$

其中 m 被称为最小曲率 (min curvature)，该条件对任意凸函数都成立。

Hessian 存在时的等价条件 当 f 二阶可微时，泰勒展开有

$$f(y) \approx f(x) + (y - x)^T \nabla f(x) + \frac{1}{2}(y - x)^T \nabla^2 f(x)(y - x)$$

强凸性要求

$$f(y) \geq f(x) + (y - x)^T \nabla f(x) + \frac{\lambda_{\min}}{2} \|y - x\|^2$$

其中 λ_{\min} 是 Hessian $\nabla^2 f(x)$ 的最小特征值。

矩阵判据 因此， f 是 m -强凸的当且仅当

$$\nabla^2 f(x) \succeq mI$$

即 Hessian 处处大于等于 m 倍单位阵。

注释 15.12. 强凸函数不仅保证唯一极小点，还能保证优化算法更快收敛（如梯度下降线性收敛）。

强凸用于判别函数的收敛性和唯一性。强凸函数的梯度下降法具有更好的收敛速度。强凸这里构建了一个二次函数的下界，保证了函数的“弯曲程度”，使得优化问题更易处理。

这里构建上界：

15.5.3 Lipschitz 连续与上界 (Lipschitz Smoothness and Upper Bound)

定义 若函数 f 的梯度 $\nabla f(x)$ 是 M -Lipschitz 连续的，即存在常数 $M > 0$ ，使得

$$\|\nabla f(x) - \nabla f(y)\| \leq M\|x - y\|, \quad \forall x, y$$

则称 f 是 M -smooth 的， M 称为 Lipschitz 常数。

二次上界 M -smooth 的函数 f 满足如下二次上界：

$$f(y) \leq f(x) + (y - x)^T \nabla f(x) + \frac{M}{2} \|y - x\|^2$$

即函数值不会超过其在 x 点的线性近似加上一个二次项。

推论 该上界在分析梯度下降等优化算法时非常重要，常用于收敛性证明和步长选择。

目标误差与最优解距离的界 对于 M -smooth、 m -强凸的函数，有如下界：

$$\frac{m}{2} \|y - x^*\|^2 \leq f(y) - f(x^*) \leq \frac{M}{2} \|y - x^*\|^2$$

其中 x^* 为最小点。这说明目标函数值与最优解的距离平方成正比。

15.5.4 凸函数的条件数 (Condition Number)

定义 条件数 κ 衡量函数的“弯曲程度”或“各向异性”，反映优化问题的难易程度。常见定义如下：

- 一般函数: $\kappa = \frac{\text{major axis}}{\text{minor axis}}$, 即等高线主轴与次轴之比。
- 光滑函数: $\kappa \approx \text{cond}(\nabla^2 f(x))$, 即 Hessian 的条件数 (最大特征值与最小特征值之比)。
- 可微函数: $\kappa = \frac{M}{m}$, 其中 M 是 Hessian 最大特征值, m 是最小特征值。

几何意义 条件数越大, 等高线越“扁”, 优化算法收敛越慢; 条件数越小, 等高线越“圆”, 优化更容易。

优化算法中的作用 条件数决定了梯度下降等一阶方法的收敛速度。条件数越大, 收敛越慢, 此时需要考虑更高阶的信息; 条件数越小, 收敛越快。实际问题中常通过预处理 (如预条件化) 降低条件数以加速优化。

15.5.5 次微分 sub-differential

对于不可微的凸函数, 梯度的概念可以推广为次微分。在点 x 处, $f(x)$ 的次微分定义为:

$$\partial f(x) = \{g \mid f(y) \geq f(x) + (y - x)^T g, \forall y\}$$

即所有使得该不等式成立的向量 g 构成次微分集。

几何意义 在可微点, 次微分只有一个元素, 即梯度本身; 在不可微点 (如 $|x|$ 在 $x = 0$ 处), 次微分是所有支持该点的切线斜率的集合。

最优化条件 对于凸函数, x^* 是极小点当且仅当

$$0 \in \partial f(x^*)$$

即 0 属于 x^* 处的次微分集。

例子 15.13 (绝对值函数的次微分). 设 $f(x) = |x|$, 则

$$\partial f(x) = \begin{cases} \{1\}, & x > 0 \\ \{-1\}, & x < 0 \\ [-1, 1], & x = 0 \end{cases}$$

即在 $x = 0$ 处, 所有斜率在 $[-1, 1]$ 区间内的直线都是 $|x|$ 的支持切线。

例子 15.14 (最大值函数的次微分). 设 $f(x) = \max\{x_1, x_2, \dots, x_n\}$, 则在 x 处的次微分为

$$\partial f(x) = \text{conv}\{e_i \mid x_i = f(x)\}$$

其中 e_i 是第 i 个标准基向量, conv 表示取凸包。即所有达到最大值的分量对应的基向量的凸组合。

沿着sub-differential的方向前进, 函数值不会下降。次微分的概念在凸优化中非常重要, 尤其是处理不可微凸函数时。

对于非光滑函数, 最速下降的方向是次微分集中的膜长最小的元素的反方向。

15.5.6 凸函数的单调性与一阶条件

(次) 梯度的单调性 (Monotonicity) 任何凸函数的(次)梯度是单调的, 即对于任意 x, y , 有

$$\langle y - x, \nabla f(y) - \nabla f(x) \rangle \geq 0$$

, 这里是内积。更一般地, 对于不可微的凸函数, 若 $g_x \in \partial f(x)$, $g_y \in \partial f(y)$, 则

$$\langle y - x, g_y - g_x \rangle \geq 0$$

推导 该性质可以通过将凸函数的切线性质在 x 和 y 处分别写出并相加得到:

$$\begin{aligned} f(y) &\geq f(x) + \nabla f(x)^T(y - x) \\ f(x) &\geq f(y) + \nabla f(y)^T(x - y) \end{aligned}$$

两式相加, 得

$$0 \geq (\nabla f(x) - \nabla f(y))^T(y - x)$$

即

$$\langle y - x, \nabla f(y) - \nabla f(x) \rangle \geq 0$$

意义 该单调性是凸优化理论的基础之一, 保证了梯度型方法的收敛性, 也可用于证明极值点的唯一性等性质。

15.6 无约束的非凸优化

15.6.1 最速下降 steepest gradient descent

$$x^{k+1} = x^k - \eta \nabla f(x^k) \quad (266)$$

15.6.2 步长选择与线搜索 (Step Size and Line Search)

最速下降法的关键在于步长 (step size, 记作 η) 的选择。常见的步长选择策略包括:

- 常数步长 (Constant step size): $\eta = c$, 其中 c 为常数。
- 递减步长 (Diminishing step size): $\eta = c/k$, k 为迭代次数。
- 精确线搜索 (Exact line search): $\eta = \arg \min_{\alpha} f(x^k + \alpha d)$, 在当前方向 d 上精确最小化目标函数, 即在当前状态下使系统下降最多的步长。
- 非精确线搜索 (Inexact line search): η 满足

$$\eta \in \{\alpha \mid f(x^k) - f(x^k + \alpha d) \geq -c \cdot \alpha d^\top \nabla f(x^k)\}$$

其中 $c \in (0, 1)$ 是常数, 保证每步都有足够的下降。

不同的步长策略影响算法的收敛速度和稳定性。精确线搜索通常收敛较快, 但计算代价高; 常数步长实现简单, 但需手动调参; 非精确线搜索 (如Armijo规则) 在实际中应用广泛, 兼顾效率与收敛性。

15.6.3 Armijo条件（充分下降条件）

在非精确线搜索中，常用的步长选择准则之一是**Armijo条件** (sufficient decrease condition)，其数学表达为：

$$\eta \in \{\alpha \mid f(x^k) - f(x^k + \alpha d) \geq -c \cdot \alpha d^\top \nabla f(x^k)\}, \quad c \in (0, 1)$$

即步长 α 需要使目标函数的下降量至少达到梯度下降的一个比例。

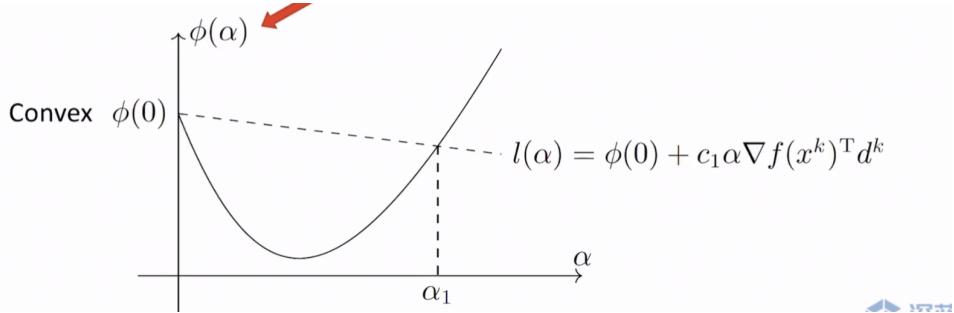


图 21: Armijo条件的几何解释： $\phi(\alpha)$ 为目标函数在方向 d 上的截面， $l(\alpha)$ 为线性下界。

如上图所示， $\phi(\alpha)$ 表示在当前点沿方向 d 的目标函数截面， $l(\alpha) = \phi(0) + c_1 \alpha \nabla f(x^k)^\top d^k$ 为线性下界。Armijo条件要求实际下降量不小于该线性下界。

意义

15.6.4 Backtracking/Armijo线搜索算法

Algorithm 12: Backtracking/Armijo线搜索

Input: 当前点 x^k , 目标函数 $f(x)$, 梯度 $\nabla f(x^k)$, 初始步长 $\eta > 0$, 常数 $c \in (0, 1)$

Output: 步长 η , 新点 x^{k+1}

选择搜索方向 $d = -\nabla f(x^k)$;

while $f(x^k + \eta d) > f(x^k) + c \cdot \eta d^\top \nabla f(x^k)$ **do**

$\eta \leftarrow \eta/2$;

更新: $x^{k+1} = x^k + \eta d$;

重复上述过程，直到梯度足够小或次微分包含零为止。我们在最优控制中也用到了这个下降方法。

结论：在迭代精度要求更高的时候性能是相似的。最速梯度下降因为只是用了一阶信息，所以收敛速度较慢，特别当条件数较大时，收敛速度会很慢。

15.7 modified damped newton method (修正阻尼牛顿法)

需要函数连续，一阶和二阶导数连续。

15.7.1 Newton法与二阶泰勒展开

二阶泰勒展开 对光滑函数 $f(x)$ ，在点 x_k 处的二阶泰勒展开为

$$f(x) \approx \hat{f}(x) \triangleq f(x_k) + \nabla f(x_k)^\top (x - x_k) + \frac{1}{2}(x - x_k)^\top \nabla^2 f(x_k)(x - x_k)$$

最小化二次近似 对二次近似 $\hat{f}(x)$ 求极小值，令梯度为零：

$$\nabla \hat{f}(x) = \nabla^2 f(x_k)(x - x_k) + \nabla f(x_k) = 0$$

解得

$$x = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k)$$

前提是 $\nabla^2 f(x_k) \succ 0$ (正定)。

Newton步 因此，Newton法的迭代公式为

$$x_{k+1} = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k)$$

如果 f 是二次函数，则Newton法一步即可到达最优解。

注释 15.15. Newton法利用了二阶信息 (Hessian)，在Hessian正定时具有二次收敛速度，但每步需解线性方程组，计算代价较高。实际中常结合阻尼或修正策略以保证收敛性。

15.7.2 Hessian修正与阻尼牛顿法 (Modified/Damped Newton Method)

Hessian矩阵不一定一直可逆，当其不可逆的时候，Newton法就不适用了。此时需要修正Hessian矩阵，使其可逆，我们已经在最优控制中学习过一遍了，这里稍加复习：

Hessian修正 当Hessian矩阵 $\nabla^2 f(x_k)$ 不可逆或不正定时，常用的修正方法有：

- **加正则项 (Levenberg-Marquardt修正):** 在Hessian上加上一个正定矩阵（通常是 λI ），即

$$H_{\text{mod}} = \nabla^2 f(x_k) + \lambda I$$

其中 $\lambda > 0$ ，保证 H_{mod} 正定且可逆。

- **截断负特征值:** 将Hessian的负特征值调整为正值，确保正定性。
- **BFGS等拟牛顿方法:** 用正定近似矩阵代替Hessian。

阻尼牛顿法 为防止步长过大导致发散，采用阻尼 (damping) 策略，即

$$x_{k+1} = x_k - \eta [H_{\text{mod}}]^{-1} \nabla f(x_k)$$

其中 $\eta \in (0, 1]$ 为阻尼系数，通常结合线搜索 (如Armijo条件) 自适应调整。

算法流程

1. 计算梯度 $\nabla f(x_k)$ 和Hessian $\nabla^2 f(x_k)$ 。
2. 若Hessian不正定或不可逆，则加正则项 λI 修正。
3. 求解修正后的牛顿步 $d_k = -[H_{\text{mod}}]^{-1} \nabla f(x_k)$ 。
4. 用线搜索确定步长 η ，更新 $x_{k+1} = x_k + \eta d_k$ 。

5. 重复直到收敛。

注释 15.16. 修正和阻尼策略保证了牛顿法在非凸或病态问题下的收敛性和稳定性，是实际优化中常用的改进方法。

15.8 Practical Newton's Method 实用牛顿法

基本思想 Newton法的核心是解线性方程组

$$\nabla^2 f(x)d = -\nabla f(x)$$

但实际上Hessian矩阵可能是半正定（PSD）或不定（indefinite），导致数值不稳定或不可逆。

凸函数情形 若 f 是凸函数，Hessian必为半正定。此时可采用如下修正：

$$M = \nabla^2 f(x) + \epsilon I, \quad \epsilon = \min(1, \|\nabla f(x)\|_\infty) / 10$$

其中 ϵ 为小正数， I 为单位阵，确保 M 正定。

Cholesky分解 由于 M 正定，可用Cholesky分解 $M = LL^\top$ ， L 为下三角阵。搜索方向 d 由

$$Md = -\nabla f(x)$$

求解。

非凸函数情形 若 f 非凸，Hessian可能不定。此时可用Bunch-Kaufman分解：

$$M = LBL^\top$$

其中 B 为块对角阵（块大小 1×1 或 2×2 ），所有 2×2 块包含非正特征值，可方便地修正。

小结

- 对于凸问题，Hessian加正则项后用Cholesky分解，保证数值稳定。
- 对于非凸问题，采用Bunch-Kaufman分解，动态修正Hessian的非正定性。
- 这两种方法都能有效提升Newton法在实际优化中的鲁棒性和收敛性。

16 无约束优化理论

- 回顾 Newton 方法：二阶泰勒展开、Hessian 性质、优缺点分析。
- 拟牛顿法（如 BFGS、L-BFGS）：Hessian 近似、割线条件、实际应用。
- 非凸与非光滑优化：下降方向保证、Wolfe 条件、Lewis & Overton 策略。
- Hessian-Free 优化（Newton-CG）：Hessian-向量乘法、共轭梯度法、适用场景。
- 应用案例：最小能量样条曲线在机器人路径平滑中的优化建模与求解。