https://doi.org/10.1051/jnwpu/20254340659

基于并行计算的轨迹快速优化方法研究

王天一, 赵吉松

(南京航空航天大学 航天学院, 江苏 南京 210016)

摘 要:直接配点法通过对控制变量和状态变量都进行离散将轨迹优化问题转化为非线性规划(nonlinear programming,NLP)进行求解。在求解 NLP 时,需要反复计算 NLP 的一阶/二阶偏导数和动力学系统在各离散点处的值,计算量比较大。针对该问题,提出如下解决策略:引入超对偶数方法准确识别 NLP 二阶偏导数矩阵的稀疏型,确定其中非零元素的位置;采用多核并行方式快速计算 NLP 的一阶/二阶偏导数的非零元素以及动力学系统在各离散点处的值;在 C++环境下采用 OpenMP 方式进行编程计算,从编程语言角度进一步提高计算效率。仿真结果表明,文中方法给出的策略在不影响精度的情况下,均能显著提高轨迹优化效率。

关 键 词:轨迹优化;非线性规划;稀疏;超对偶数;并行计算

中图分类号:V412 **文献标志码**:A **文章编号**:1000-2758(2025)04-0659-09

轨迹优化在航空航天领域应用广泛,具有非常重要的意义和实际应用价值[1]。轨迹优化本质上是一个最优控制问题,目前广泛使用的求解方法有 2 种,即直接法和间接法。间接法基于变分法或极大值原理,而直接法是通过对状态变量和控制变量的离散化构造最优控制问题的近似,从而将最优控制问题转化为非线性规划问题。直接法中的配点法[2]由于不需要推导最优性必要条件,对初值的敏感性较低且易收敛,近年来得到了广泛研究[3-4]。

目前广泛使用的 NLP 求解器 SNOPT[5] 和 IPOPT[6]都需要反复计算 NLP 的一阶/二阶偏导数,因此,提高 NLP 一阶/二阶偏导数的计算效率对于提升轨迹优化的效率至关重要。Agamawi 和 Patterson 等[7-8]进行了深入研究,发现轨迹优化问题离散得到的 NLP 具有高度的稀疏性,即 NLP 的一阶/二阶偏导数矩阵中包含大量零元素。赵吉松[9]研究了局部配点法离散得到的 NLP 一阶偏导数的稀疏特性,建立了一阶偏导数的高效计算方法。此外,为提高偏导数的计算效率,Fike 等[10]提出了基于超对偶数的偏导数计算方法(简称超对偶数方法),相比于目前广泛使用的有限差分法、自动微分法[11]等偏导数计算方法,该方法可以迅速准确地计算一阶和二阶偏导数。Agamawi 等[12]使用该方法为 NLP 求解器提供准确的偏导数,但他们在对偏导数矩阵进行稀疏性分析时是通过一阶偏导数矩阵的稀疏型近似得到二阶偏导数矩阵的稀疏型[13],该方法准确度较低,会错误识别出额外的二阶偏导数,增加偏导数矩阵非零元素的个数,从而增大计算量,最终降低轨迹优化问题求解的速度和精度。因此,本文改进了 NLP 二阶偏导数矩阵的稀疏性分析方法,与 Rao 等人使用超对偶数方法进行实际偏导数计算不同,本文侧重于使用该方法去准确识别 NLP 二阶偏导数矩阵的稀疏型。

此外,通过研究发现,NLP 的一阶/二阶偏导数矩阵中偏导数的计算以及动力学系统在各离散点处的计算都是相互独立的,非常适合进行并行计算。并行计算是相对于串行计算而言,通过同时使用多个计算资源来解决计算问题的过程,是提高计算速度和处理能力的一种有效手段[14]。目前广泛使用的并行计算方法有多种,例如基于 GPU 的 CUDA、基于 CPU 的 MPI、OpenMP,这些方法在轨迹优化领域也有较为广泛的应用。Betts 等[15]提出了一种使用并行处理器来减少这些计算成本的技术,将轨迹分解成多个阶段,可以并行模拟,从而降低单个轨迹的成本。Antony 等[16]基于 CUDA 平台,提出了一种

收稿日期:2024-08-24

作者简介:王天一(2000—),硕士研究生

通信作者:赵吉松(1984—),副教授 e-mail:zhaojisong@nuaa.edu.cn

基于 GPU 加速的间接弹道优化方法,由此产生了快速最优控制框架,能够构建高质量的最优轨迹。钟新玉等[17]基于 MPI 消息传递机制实现了弹道仿真计算模型的并行编程,获得了较高的加速比和计算效率,与单机运行相比,计算机集群可以大大缩短计算时间,提高计算效率。孟振华[18]以组合导航、轨迹规划、图像制导等为背景进行基于共享存储的 OpenMP 并行编程模型的并行设计和优化设计,显著缩短了各类算法的计算时间。与 MPI 和 CUDA 相比,OpenMP 在便于实现的同时兼具较高的计算效率,在处理规模不是特别大的计算任务时比前面 2 种方法更有优势。Agamawi 和 Rao 在文献[12]中使用串行计算偏导数,而并行计算相比于串行计算可以进一步提升计算效率。因此本文选择使用 OpenMP 并行计算 NLP 一阶/二阶偏导数矩阵中的非零元素。具体而言,在得到 NLP 一阶/二阶偏导数矩阵的稀疏型后,通过 OpenMP 将稀疏型中的非零元素分配到不同的线程中进行并行计算,最后再将计算完成的各非零元素放回其原来的位置,组装得到 NLP 的一阶/二阶偏导数矩阵。此外,通过使用 OpenMP 并行计算动力学系统在各离散点处的值,可进一步提升计算效率。

本文结合超对偶数方法和 OpenMP 并行计算的综合策略不仅提高了计算效率,还保留了计算的精确性,为解决较为复杂的轨迹优化问题提供了一种高效且可靠的途径。

1 轨迹优化问题的数学描述

轨迹优化问题本质上属于最优控制问题,以 Bolza 型最优控制问题为例,可描述为:求解控制变量 $\boldsymbol{u}(t) \in \mathbf{R}^m$ 使得(1)式所示目标函数最小化。

$$J = M(\boldsymbol{x}(t_0), t_0, \boldsymbol{x}(t_{\mathrm{f}}), t_{\mathrm{f}}) + \int_{t_0}^{t_{\mathrm{f}}} L(\boldsymbol{x}(t), \boldsymbol{u}(t), t)\,\mathrm{d}t \tag{1}$$

式中,$M: \mathbf{R}^n \times \mathbf{R} \times \mathbf{R}^n \times \mathbf{R} \to \mathbf{R}$,$L: \mathbf{R}^n \times \mathbf{R}^m \times \mathbf{R} \to \mathbf{R}$,$\boldsymbol{x} \in \mathbf{R}^n$,$\boldsymbol{u} \in \mathbf{R}^m$,$t \in [t_0, t_{\mathrm{f}}]$。

状态方程为

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}(t), \boldsymbol{u}(t), t), t \in [t_0, t_{\mathrm{f}}] \tag{2}$$

端点条件为

$$\boldsymbol{E}(\boldsymbol{x}(t_0), t_0, \boldsymbol{x}(t_{\mathrm{f}}), t_{\mathrm{f}}) = 0 \tag{3}$$

路径约束为

$$\boldsymbol{C}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \leqslant 0, t \in [t_0, t_{\mathrm{f}}] \tag{4}$$

式中 $\boldsymbol{f}: \mathbf{R}^n \times \mathbf{R}^m \times \mathbf{R} \to \mathbf{R}^n$,$\boldsymbol{E}: \mathbf{R}^n \times \mathbf{R} \times \mathbf{R}^m \times \mathbf{R} \to \mathbf{R}^e$,$\boldsymbol{C}: \mathbf{R}^n \times \mathbf{R}^m \times \mathbf{R} \to \mathbf{R}^c$。(1) ~ (4) 式所描述的问题称为连续 Bolza 型最优控制问题。

2 基于超对偶数的稀疏性分析方法

2.1 NLP 二阶偏导数矩阵

为构造 NLP 二阶偏导数矩阵,定义函数 F,F 包括状态方程 $\boldsymbol{f}$、端点条件 $\boldsymbol{E}$ 和路径约束 $\boldsymbol{C}$ 三部分。定义总二阶偏导数矩阵如(5) 式所示。

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{H}_1 & & \\ \boldsymbol{H}_2 & \boldsymbol{H}_3 & \\ \boldsymbol{H}_4 & \boldsymbol{H}_5 & \boldsymbol{H}_6 \end{bmatrix}$$

$$\boldsymbol{H}_1 = \begin{bmatrix} \dfrac{\partial^2 F}{\partial x_{1i}^2} & & & \\ \dfrac{\partial^2 F}{\partial x_{1i}\partial x_{2i}} & \dfrac{\partial^2 F}{\partial x_{2i}^2} & & \\ \vdots & \vdots & & \\ \dfrac{\partial^2 F}{\partial x_{1i}\partial x_{ni}} & \dfrac{\partial^2 F}{\partial x_{2i}\partial x_{ni}} & \cdots & \dfrac{\partial^2 F}{\partial x_{ni}^2} \end{bmatrix}$$

$$\boldsymbol{H}_2 = \begin{bmatrix} \dfrac{\partial^2 F}{\partial x_{1i}\partial u_{1i}} & \dfrac{\partial^2 F}{\partial x_{2i}\partial u_{1i}} & \cdots & \dfrac{\partial^2 F}{\partial x_{ni}\partial u_{1i}} \\ \dfrac{\partial^2 F}{\partial x_{1i}\partial u_{2i}} & \dfrac{\partial^2 F}{\partial x_{2i}\partial u_{2i}} & \cdots & \dfrac{\partial^2 F}{\partial x_{ni}\partial u_{2i}} \\ \vdots & \vdots & & \vdots \\ \dfrac{\partial^2 F}{\partial x_{1i}\partial u_{mi}} & \dfrac{\partial^2 F}{\partial x_{2i}\partial u_{mi}} & \cdots & \dfrac{\partial^2 F}{\partial x_{ni}\partial u_{mi}} \end{bmatrix}$$

$$\boldsymbol{H}_3 = \begin{bmatrix} \dfrac{\partial^2 F}{\partial u_{1i}^2} & & & \\ \dfrac{\partial^2 F}{\partial u_{1i}\partial u_{2i}} & \dfrac{\partial^2 F}{\partial u_{2i}^2} & & \\ \vdots & \vdots & & \\ \dfrac{\partial^2 F}{\partial u_{1i}\partial u_{mi}} & \dfrac{\partial^2 F}{\partial u_{2i}\partial u_{mi}} & \cdots & \dfrac{\partial^2 F}{\partial u_{mi}^2} \end{bmatrix}$$

$$\boldsymbol{H}_4 = \begin{bmatrix} \dfrac{\partial^2 F}{\partial x_{1i}\partial t_{0i}} & \dfrac{\partial^2 F}{\partial x_{2i}\partial t_{0i}} & \cdots & \dfrac{\partial^2 F}{\partial x_{ni}\partial t_{0i}} \\ \dfrac{\partial^2 F}{\partial x_{1i}\partial t_{fi}} & \dfrac{\partial^2 F}{\partial x_{2i}\partial t_{fi}} & \cdots & \dfrac{\partial^2 F}{\partial x_{ni}\partial t_{fi}} \end{bmatrix}$$

$$\boldsymbol{H}_5 = \begin{bmatrix} \dfrac{\partial^2 F}{\partial u_{1i}\partial t_{0i}} & \dfrac{\partial^2 F}{\partial u_{2i}\partial t_{0i}} & \cdots & \dfrac{\partial^2 F}{\partial u_{mi}\partial t_{0i}} \\ \dfrac{\partial^2 F}{\partial u_{1i}\partial t_{fi}} & \dfrac{\partial^2 F}{\partial u_{2i}\partial t_{fi}} & \cdots & \dfrac{\partial^2 F}{\partial u_{mi}\partial t_{fi}} \end{bmatrix}$$

$$\boldsymbol{H}_6 = \begin{bmatrix} \dfrac{\partial^2 F}{\partial t_{0i}^2} & \\ \dfrac{\partial^2 F}{\partial t_{0i}\partial t_{fi}} & \dfrac{\partial^2 F}{\partial t_{fi}^2} \end{bmatrix} \tag{5}$$

式中:$x_{1i}, x_{2i}, \cdots, x_{ni}$ 为第 i 个节点处的 n 个状态变量;$u_{1i}, u_{2i}, \cdots, u_{mi}$ 为第 i 个节点处的 m 个控制变量;t_{0i} 为第 i 个节点处的初始时刻;t_{fi} 为第 i 个节点处的终端时刻。以 x_{1i} 为例,1 表示第 1 个状态变量 x_1,i 表示第 i 个节点,其余变量同理。上述二阶偏导数矩阵在各个节点处的形式都是相同的。

提供准确的 NLP 二阶偏导数矩阵可以显著提高 NLP 求解器的计算效率,然而得到精确的二阶偏导数矩阵却非常困难。考虑到 NLP 二阶偏导数矩阵具有高度的稀疏性,如果准确识别其稀疏型,可以有效提升 NLP 问题求解的效率和准确性。目前流行的通用轨迹优化软件 GPOPS-Ⅱ已经可以准确地识别 NLP 一阶偏导数矩阵稀疏型,但在识别 NLP 二阶偏导数矩阵的稀疏型时还不够准确。

以函数 $z = x + y$ 为例,GPOPS-Ⅱ计算一阶导数

$$\frac{\partial z}{\partial x} = 1, \quad \frac{\partial z}{\partial y} = 1 \tag{6}$$

由于 GPOPS-Ⅱ对各变量存在一阶导数,即使 z 对变量 x 和 y 的二阶偏导数都为 0,GPOPS-Ⅱ也会推导二阶偏导数存在且不为 0,这就导致 GPOPS-Ⅱ在识别二阶偏导数矩阵的稀疏型时过于保守,将零元素识别成非零元素。虽然 GPOPS-Ⅱ的稀疏型识别方法会识别出额外的非零元素,但在进行非零元素的计算时,该元素的计算结果还是为 0,所以并不会影响最终的寻优结果,只会影响寻优速度。

GPOPS-Ⅱ通过一阶偏导数矩阵的稀疏型近似得到二阶偏导数矩阵的稀疏型,该方法准确度较低,会识别出额外的二阶偏导数,增加偏导数矩阵非零元素的个数,从而增大计算量,最终降低轨迹优化问题求解的速度和精度。

2.2　基于超对偶数的偏导数计算方法

对于多变量函数,基于超对偶数的二阶偏导数计算如(7)式所示。

$$\frac{\partial^2 \boldsymbol{f}(\boldsymbol{x})}{\partial x_i \partial x_j} = \frac{\varepsilon_1\varepsilon_2 \mathrm{part}[\boldsymbol{f}(\boldsymbol{x} + h_1\varepsilon_1\boldsymbol{e}_i + h_2\varepsilon_2\boldsymbol{e}_j + 0\varepsilon_1\varepsilon_2)]}{h_1 h_2} \tag{7}$$

式中:$\boldsymbol{f}(\boldsymbol{x})$ 为问题的函数,$\boldsymbol{f}: \mathbf{R}^n \to \mathbf{R}^m$,$\boldsymbol{x}$ 为 n 维向量;h_1,h_2 为步长,本文设置为 1;$\boldsymbol{e}_i$,$\boldsymbol{e}_j$ 为 2 个单位向量。$\varepsilon_1\varepsilon_2\mathrm{part}[\cdot]$ 表示提取函数中 $\varepsilon_1\varepsilon_2$ 的系数,即 $\boldsymbol{f}(\boldsymbol{x})$ 对 $x_i x_j$ 的 m 维偏导数向量。

下面演示偏导数的计算过程。

给定一个 n 维自变量 $\boldsymbol{x}$ 和 2 个 n 维单位向量 $\boldsymbol{e}_i$,$\boldsymbol{e}_j$,如(8)式所示。

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \ \boldsymbol{e}_i = \begin{bmatrix} \mathbf{0}_{(i-1)\times 1} \\ 1 \\ \mathbf{0}_{(n-i)\times 1} \end{bmatrix}, \ \boldsymbol{e}_j = \begin{bmatrix} \mathbf{0}_{(j-1)\times 1} \\ 1 \\ \mathbf{0}_{(n-j)\times 1} \end{bmatrix} \tag{8}$$

式中,单位向量 $\boldsymbol{e}_i$ 的第 i 个分量为 1,其余分量均为 0;单位向量 $\boldsymbol{e}_j$ 的第 j 个分量为 1,其余分量均为 0。

利用上述向量与零向量构造的超对偶数向量为新的函数自变量向量,包括 4 个部分:①n 维自变量 $\boldsymbol{x}$;②单位向量 $\boldsymbol{e}_i$ 乘以复数单位 ε_1;③单位向量 $\boldsymbol{e}_j$ 乘以复数单位 ε_2;④零向量乘以复数单位 $\varepsilon_1\varepsilon_2$,如(9)式所示。

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_j \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{(i-1)\times 1} \\ 1 \\ \mathbf{0}_{(n-i)\times 1} \end{bmatrix}\varepsilon_1 + \begin{bmatrix} \mathbf{0}_{(j-1)\times 1} \\ 1 \\ \mathbf{0}_{(n-j)\times 1} \end{bmatrix}\varepsilon_2 + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix}\varepsilon_1\varepsilon_2 \tag{9}$$

将新的函数自变量向量代入函数 $\boldsymbol{f}$ 中得到

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ \vdots \\ f_m(\boldsymbol{x}) \end{bmatrix} \tag{10}$$

式中,$\boldsymbol{f}(\boldsymbol{x})$ 为得到的 m 维函数向量,该向量由 m 个超对偶数向量分量组成。

提取函数输出中 $\varepsilon_1\varepsilon_2$ 的系数,便可得到函数 $\boldsymbol{f}$ 对 $x_i x_j$ 的二阶偏导数,如(11)式所示。

$$\frac{\partial^2 \boldsymbol{f}(\boldsymbol{x})}{\partial x_i \partial x_j} = \begin{bmatrix} \dfrac{\partial^2 f_1(\boldsymbol{x})}{\partial x_i \partial x_j} \\ \dfrac{\partial^2 f_2(\boldsymbol{x})}{\partial x_i \partial x_j} \\ \vdots \\ \dfrac{\partial^2 f_m(\boldsymbol{x})}{\partial x_i \partial x_j} \end{bmatrix} = \varepsilon_1\varepsilon_2 \mathrm{part} \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ \vdots \\ f_m(\boldsymbol{x}) \end{bmatrix} \tag{11}$$

2.3 稀疏型对比

为了描述 $\boldsymbol{H}$ 的稀疏型，定义 struct 函数[9]

$$\text{struct}(x)=\begin{cases}1 & x\neq 0\\ 0 & x=0\end{cases} \tag{12}$$

记作

$$\boldsymbol{S}=\text{struct}(\boldsymbol{H}) \tag{13}$$

式中，struct($\boldsymbol{H}$) 表示对 $\boldsymbol{H}$ 的每个元素进行 struct 运算。$\boldsymbol{S}$ 表示 $\boldsymbol{H}$ 的稀疏型。为了得到 $\boldsymbol{S}$，不需要计算 $\boldsymbol{H}$ 的每个元素具体值，只需要判断是否为 0。本文使用的超对偶数方法是通过随机选取多个实部值进行偏导数计算来判断某个元素是否为 0，进而识别出偏导数矩阵的稀疏型，并非计算元素的准确值，在进行非零元素计算时依然基于中心差分法。

下面以轨道转移问题为例，将 GPOPS-Ⅱ方法与超对偶数方法识别出的稀疏型进行对比，轨道转移问题的动力学方程在 4.1 节中给出。表 1 为使用不同方法识别出的二阶偏导数矩阵稀疏型的对比。

显然，使用超对偶数方法，能够更为精确地识别二阶偏导数矩阵的稀疏型，与手工解析方法得到的稀疏型完全相同。从表 1 中可以看出，使用 GPOPS-Ⅱ方法得到的稀疏型中非零元素个数为 30，而使用超对偶数方法得到的稀疏型中非零元素个数仅为 14，与 GPOPS-Ⅱ方法相比减少了 16 个非零元素。综上所述，使用超对偶数方法减少了冗余非零元素的计算，提高了轨迹优化的效率。

表 1 轨道转移问题 NLP 二阶偏导数矩阵稀疏型比较

对比项	GPOPS-Ⅱ方法	超对偶数方法
稀疏型	$\begin{bmatrix}3&0&0&0&0&0\\1&2&0&0&0&0\\3&1&3&0&0&0\\1&0&1&1&0&0\\1&1&1&0&1&0\\2&1&2&1&1&2\end{bmatrix}$	$\begin{bmatrix}3&0&0&0&0&0\\1&0&0&0&0&0\\3&1&1&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&1&1&2\end{bmatrix}$
非零元素个数	30	14

3 基于 OpenMP 的并行计算

如图 1 所示，NLP 偏导数矩阵中包含大量元素。

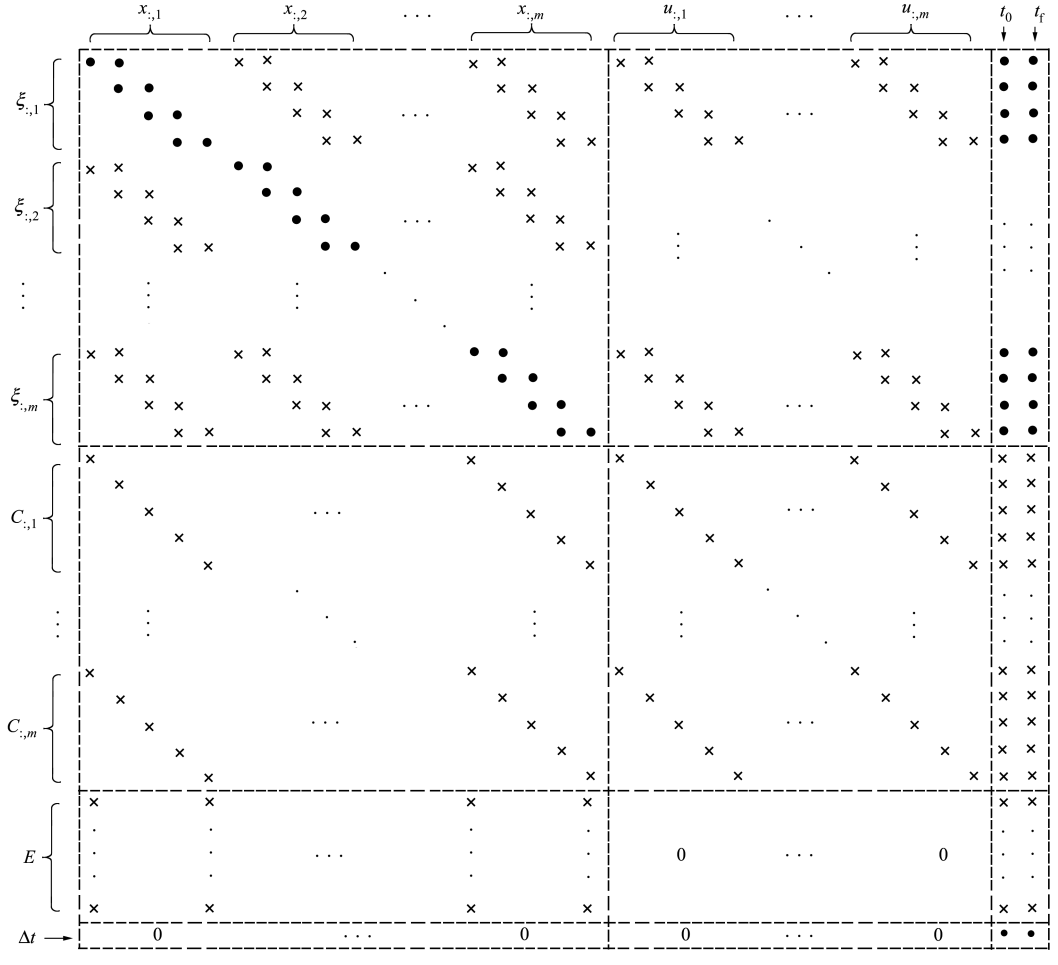


图 1 NLP 一阶偏导数矩阵示意图

其中,空白元素表示恒为零,“ • ”表示非零元素,“×”表示可能不为零的元素。

本文使用 OpenMP 作为并行计算方法。OpenMP 是一种基于共享内存的多线程并行编程模型,专为多核处理器设计,通过使用 OpenMP,程序能够同时对独立的计算子块进行并行处理,从而显著提高计算速度和程序的整体运行效率。

具体来说,本文首先准确识别 NLP 一阶/二阶偏导数矩阵的稀疏型,然后使用 OpenMP 将其中的非零元素分配到多个线程中进行计算。如图 2 所示,假设需要计算的非零元素的个数为 m,通过 OpenMP 将这 m 个非零元素分配到 k 个线程中并行计算,在每个线程计算完成后收集结果,组装得到 NLP 一阶/二阶偏导数矩阵。

此外,除了 NLP 的一阶/二阶偏导数,动力学系统在各离散点处的值也可以进行并行计算。同理,将各离散点处的值也分配到这 k 个线程中进行计算,可以进一步提升计算效率。

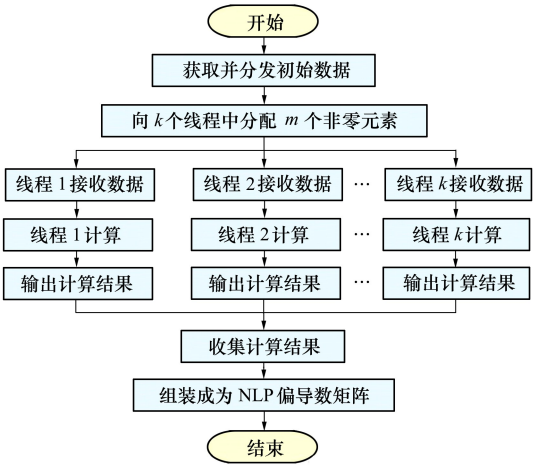


图 2　非零元素并行计算流程示意图

4　仿真与分析

4.1　轨道转移问题

轨道转移问题是航空航天领域的经典问题,需要在满足物理约束的前提下找到最优的转移策略。

该轨迹优化问题的状态变量分别为矢径 r、轨道角度 θ、径向速度 v_r、切向速度 v_θ,控制变量为 u_r 和 u_θ。该轨迹优化问题的描述如下:求解 1 个最优控制 $\boldsymbol{u}^*(t)=(u_r(t),u_\theta(t))^{\mathrm{T}}$,使系统在末端时刻矢径最大化。因此,目标函数为

$$J=\min(-r_{t_\mathrm{f}}) \tag{14}$$

满足动力学方程

$$\begin{cases}\dot r=v_r\\ \dot\theta=\dfrac{v_\theta}{r}\\ \dot v_r=\dfrac{v_\theta^2}{r}-\dfrac{\mu}{r^2}+\dfrac{T}{m}\mu_r\\ \dot v_\theta=\dfrac{v_r v_\theta}{r}+\dfrac{T}{m}\mu_\theta\end{cases} \tag{15}$$

式中:μ 为太阳引力常数;T 为发动机推力大小;m 为飞行器质量,它会因为燃料和其他部件的消耗而随时间逐渐变化。m 可以表示为

$$m=m(t_0)-m_\mathrm{s}\cdot t \tag{16}$$

式中:t_0 为初始时间;m_s 为飞行器质量变化速度,为简化研究,设其为常值。

边界条件为

$$\boldsymbol{x}(t_0)=x_0,\ \boldsymbol{x}(t_\mathrm{f})=x_\mathrm{f} \tag{17}$$

在轨道转移的过程中,状态变量和控制变量需要满足约束

$$\boldsymbol{x}_{\min}\leqslant\boldsymbol{x}(t)\leqslant\boldsymbol{x}_{\max} \tag{18}$$

$$\boldsymbol{u}_{\min}\leqslant\boldsymbol{u}(t)\leqslant\boldsymbol{u}_{\max} \tag{19}$$

同时,由于推力受限,在轨道转移中需要满足路径约束

$$\|\boldsymbol{u}\|_2=\frac{T}{m(t_0)} \tag{20}$$

初始条件是 $t_0=0$,$m(t_0)=1\ 000$ kg,$r(t_0)=1$ AU,$\theta(t_0)=0$,$v_r(t_0)=0$,$v_\theta(t_0)=1$ AU/TU,$T=0.140\ 5\ m_0V_0{}^2/r_0$,$\mu=1$ AU^3/TU^3。终端约束是 $t_\mathrm{f}=3.32$ TU,$v_r(t_\mathrm{f})=0$。其中 AU 为天文单位,指地球到太阳的平均距离,在这里取 $1.495\ 978\ 7\times10^{11}$ m。在本算例中,TU(time unit) 为时间单位,其取值为一年,即 365.25 d。

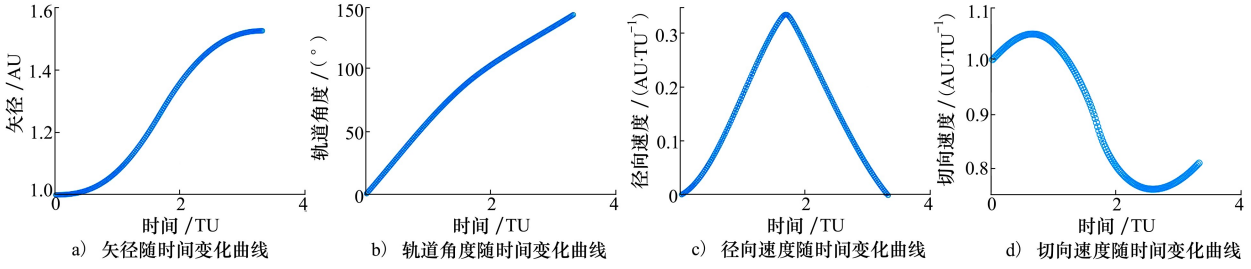
项目组前期开发的轨迹优化软件[19]使用的优化算法为梯形格式的配点法,使用 GPOPS-Ⅱ中的偏导数矩阵稀疏型识别方法,并且不使用 OpenMP 并行计算(简称对比方法)。本文方法使用的优化算法为梯形格式的配点法,使用超对偶数方法识别偏导数矩阵的稀疏型,并使用 OpenMP 并行计算一阶/二阶偏导数矩阵中的非零元素以及动力学系统在各离散点处的值。

本文所使用的计算平台的配置为 Intel Core i3-10100 3.60 GHz 处理器,8 GB RAM 内存,Windows

10 家庭版 64 位操作系统。本算例使用的优化方法均取 101 个节点进行求解,OpenMP 采用 4 线程并行。为确保结果的可靠性和准确性,最终的优化耗时结果是基于 10 次优化的平均耗时。仿真结果如表 2 所示,算例的状态变量随时间变化曲线如图 3 所示,最优控制方案输出结果如图 4 所示。

表 2 不同优化方法的优化结果

优化方法	编程语言	目标函数/AU	优化耗时/ms
GPOPS-Ⅱ	Matlab	1.525 22	5 054
对比方法	C++	1.525 22	1 112
本文方法	C++	1.525 22	715



a) 矢径随时间变化曲线 b) 轨道角度随时间变化曲线 c) 径向速度随时间变化曲线 d) 切向速度随时间变化曲线

图 3 状态变量随时间变化曲线

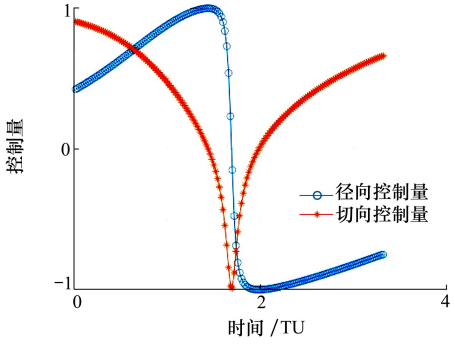


图 4 控制量随时间变化曲线

由优化结果可以看出,每种优化方法的目标函数都是相同的,均为 1.525 22 AU,这表明 3 种方法在优化结果的精度上是等效的。但值得注意的是,本文方法的优化耗时明显短于 GPOPS-Ⅱ和对比方法,求解时间仅为 715 ms,相比于 GPOPS-Ⅱ大幅减少了约 85.8%,相比于对比方法减少了 35.7%。

4.2 超声速飞行器再入问题

超声速飞行器再入问题的状态变量包括飞行器到地心的距离 r,速度 v,航迹倾角 γ,航迹偏角 ψ,经度 θ,纬度 ϕ;控制变量包括攻角 α 和侧倾角 σ。

超声速飞行器再入问题的目标函数为

$$J = \min(\phi_f) \tag{21}$$

满足动力学方程

$$\begin{cases} \dfrac{\mathrm{d}r}{\mathrm{d}t} = v\sin\gamma \\ \dfrac{\mathrm{d}v}{\mathrm{d}t} = -\dfrac{D}{m} - g\sin\gamma \\ \dfrac{\mathrm{d}\gamma}{\mathrm{d}t} = \dfrac{L\cos\sigma}{mv} - \dfrac{g\cos\gamma}{v} + \dfrac{v\cos\gamma}{r} \\ \dfrac{\mathrm{d}\psi}{\mathrm{d}t} = \dfrac{L\sin\sigma}{mv\cos\gamma} - \dfrac{v\cos\gamma\cos\psi\tan\phi}{r} \\ \dfrac{\mathrm{d}\theta}{\mathrm{d}t} = \dfrac{v\cos\gamma\cos\psi}{r\cos\phi} \\ \dfrac{\mathrm{d}\phi}{\mathrm{d}t} = \dfrac{v\cos\gamma\sin\psi}{r} \end{cases} \tag{22}$$

式中:m 为质量;L 和 D 分别为升力和阻力;g 为重力加速度;$r = h + R_e$,h 为高度;R_e 为地球半径。

升力和阻力的计算公式为

$$\begin{cases} L = \dfrac{1}{2}\rho v^2 S C_L \\ D = \dfrac{1}{2}\rho v^2 S C_D \end{cases} \tag{23}$$

式中:C_L 为升力系数;C_D 为阻力系数;ρ 为大气密度;S 为参考面积。

升力系数和阻力系数计算公式如(24)式所示。

$$\begin{cases} C_L = C_{L0} + C_{L1} \cdot \alpha \\ C_D = C_{D0} + C_{D1} \cdot \alpha + C_{D2} \cdot \alpha^2 \end{cases} \tag{24}$$

重力加速度满足

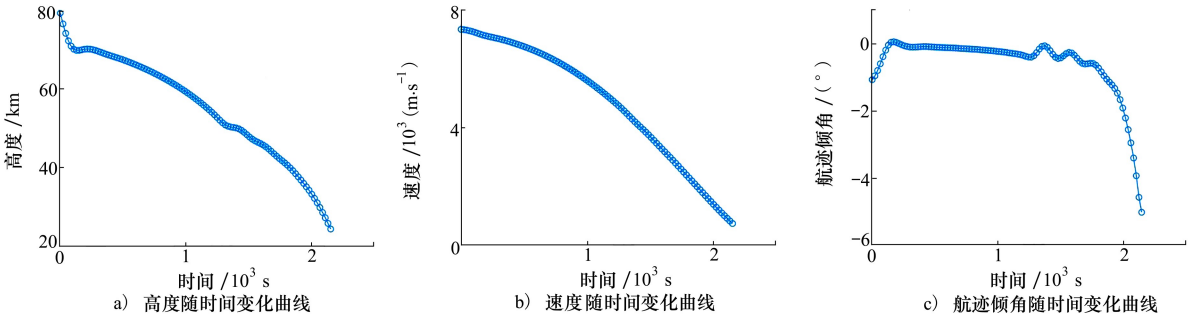
$$g = \frac{\mu}{r^2} \tag{25}$$

式中,μ 为地球引力常数。

初始条件 h_0 = 79 248 m,v_0 = 7 333.792 8 m/s,γ_0 = − 0.018 57 rad,ψ_0 = 0 rad,θ_0 = 0 rad,ϕ_0 = 0 rad。终端条件为 h_f = 24 384 m,v_f = 762 m/s,γ_f = − 0.087 27 rad。优化方法均取 101 个节点对本算例进行求解,OpenMP 采用 4 线程并行,为确保结果的可靠性和准确性,最终的优化耗时结果是基于 10 次优化的平均耗时。仿真结果如表 3 所示,状态变量随时间变化曲线如图 5 所示,控制变量随时间变化曲线如图 6 所示。

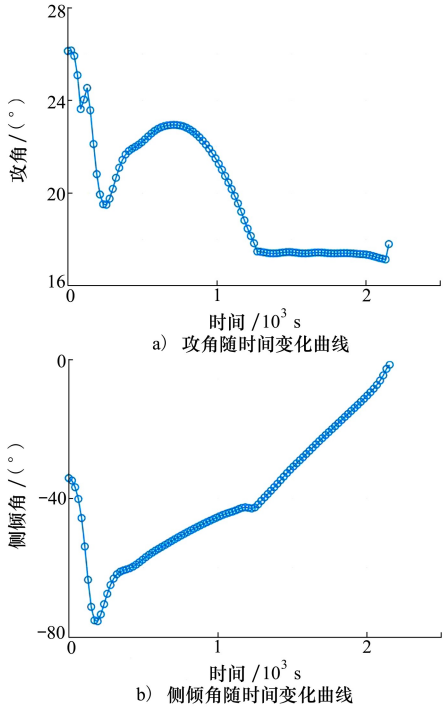
表 3　不同优化方法得到的优化结果

优化方法	编程语言	目标函数/rad	优化耗时/ms
对比方法	C++	−0.501 63	2 258
本文方法	C++	−0.501 63	1 562



a) 高度随时间变化曲线　b) 速度随时间变化曲线　c) 航迹倾角随时间变化曲线

图 5　状态变量随时间变化曲线



a) 攻角随时间变化曲线　b) 侧倾角随时间变化曲线

图 6　控制变量随时间变化曲线

从优化结果可以看出,2 种方法的目标函数都是相同的,均为−0.501 63 rad,但本文方法的优化耗时明显少于对比方法,求解时间为 1 562 ms,相比于对比方法减少了 30.8%。

通过以上 2 个算例可以看出,结合超对偶数方法和 OpenMP 的综合计算方法,可以在保证计算精度的同时,较大程度地提升轨迹优化问题的求解效率。超对偶数方法和 OpenMP 并行计算方法的引入对轨迹优化问题的求解效率有着显著影响,结合 2 种方法能够较大限度地发挥计算资源的优势,缩短计算时间,提高计算效率。

5　结　论

本文深入分析了飞行器轨迹优化问题,结合超对偶数方法与 OpenMP 并行计算方法,显著提高了轨迹优化效率。具体而言,本文通过引入超对偶数方法准确识别了 NLP 二阶偏导数矩阵的稀疏型,确定其中非零元素的位置,有效地解决了二阶偏导数的冗余计算问题,而引入 OpenMP 则充分利用了其多核处理器的优势,通过并行计算 NLP 的一阶/二阶偏导数的非零元素以及动力学系统在各离散点处的值,大幅缩短了计算时间。仿真结果表明,采用这种综合方法不仅保证了计算精度,而且提升了计算效率。对于轨道转移问题,优化时间相比于 GPOPS-Ⅱ大幅减少了约 85.8%,相比于对比方法减少了 35.7%。对于超声速飞行器再入问题,优化时间与对比方法相比减少了 30.8%。相比于对比方

法,本文方法在处理不同轨迹优化问题时均有显著的效率提升。

未来的研究可以探索更高效的并行计算策略,进一步提升计算速度。同时,还可以将这一方法扩展到其他类型的优化问题中,以验证其通用性和适应性。

参考文献:

[1] 张灿, 胡冬冬, 叶蕾, 等. 2017 年国外高超声速飞行器技术发展综述[J]. 战术导弹技术, 2018(1): 47-50
ZHANG Can, HU Dongdong, YE Lei, et al. Overview of foreign hypersonic vehicle technology development in 2017[J]. Tactical Missile Technology, 2018(1): 47-50 (in Chinese)

[2] BETTS J T. Practical methods for optimal control and estimation using nonlinear programming[M]. Philadelphia: Society for Industrial and Applied Mathematics, 2010

[3] 赵吉松, 张建宏, 李爽. 高超声速滑翔飞行器再入轨迹快速、高精度优化[J]. 宇航学报, 2019, 40(9): 1034-1043
ZHAO Jisong, ZHANG Jianhong, LI Shuang. Rapid and high accuracy approach for hypersonic glide vehicle reentry trajectory optimization[J]. Journal of Astronautics, 2019, 40(9): 1034-1043 (in Chinese)

[4] 任鹏飞, 王洪波, 周国峰. 基于自适应伪谱法的高超声速飞行器再入轨迹优化[J]. 北京航空航天大学学报, 2019, 45(11): 2257-2265
REN Pengfei, WANG Hongbo, ZHOU Guofeng. Reentry trajectory optimization of hypersonic vehicle based on adaptive pseudospectrum method[J]. Journal of Beijing University of Aeronautics and Astronautics, 2019, 45(11): 2257-2265 (in Chinese)

[5] GILL P E, MURRAY W, SAUNDERS M A. SNOPT: an SQP algorithm for large-scale constrained optimization[J]. SIAM Journal on Optimization, 2002, 12(4): 979-1006

[6] BIEGLER L T, ZAVALA V M. Large-scale nonlinear programming using IPOPT: an integrating framework for enterprise-wide dynamic optimization[J]. Computers & Chemical Engineering, 2009, 33(3): 575-582

[7] AGAMAWI Y M, RAO A V. Comparison of derivative estimation methods in optimal control using direct collocation[J]. AIAA Journal, 2020, 58(1): 341-354

[8] PATTERSON M A, RAO A V. Exploiting sparsity in direct collocation pseudospectral methods for solving optimal control problems[J]. Journal of Spacecraft and Rockets, 2012, 49(2): 364-377

[9] 赵吉松. 求解轨迹优化问题的局部配点法的稀疏性研究[J]. 宇航学报, 2017, 38(12): 1263-1272
ZHAO Jisong. Exploiting sparsity in local collocation methods for solving trajectory optimization problems[J]. Journal of Astronautics, 2017, 38(12): 1263-1272 (in Chinese)

[10] FIKE J, ALONSO J. The development of hyper-dual numbers for exact second-derivative calculations[C] // 49th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, 2011: 886

[11] CSENDES T. Developments in Reliable Computing[M]. Dordrecht: Kluwer Academic Publishers, 1999: 77-104

[12] AGAMAWI Y M, RAO A V. CGPOPS: a C++software for solving multiple-phase optimal control problems using adaptive Gaussian quadrature collocation and sparse nonlinear programming[J]. ACM Transactions on Mathematical Software, 2020, 46(3): 1-38

[13] PATTERSON M A, RAO A V. GPOPS-Ⅱ: a Matlab software for solving multiple-phase optimal control problems using hp-adaptive Gaussian quadrature collocation methods and sparse nonlinear programming[J]. ACM Trans on Mathematical Software, 2014, 41(1): 1-37

[14] 都志辉. 高性能计算之并行编程技术: MPI 并行程序设计[M]. 北京: 清华大学出版社, 2001
DU Zhihui. Parallel programming techniques in high-performance computing: MPI parallel programming[M]. Beijing: Tsinghua University Press, 2001 (in Chinese)

[15] BETTS J T, HUFFMAN W P. Trajectory optimization on a parallel processor[J]. Journal of Guidance, Control, and Dynamics, 1991, 14(2): 431-439

[16] ANTONY T, GRANT M J. Rapid indirect trajectory optimization on highly parallel computing architectures[J]. Journal of Spacecraft and Rockets, 2017, 54(5): 1081-1091

[17] 钟新玉, 王恂, 张学军. 基于 MPI 的弹道仿真的并行计算研究[J]. 航天控制, 2015, 33(3), 63-67

ZHONG Xinyu, WANG Xun, ZHANG Xuejun. The parallel computing research of trajectory simulation based on MPI[J]. Aerospace Control, 2015, 33(3): 63-67 (in Chinese)

[18] 孟振华. 基于 FPGA+DSP 的高性能并行计算架构研究[D]. 北京: 中国航天科技集团公司第一研究院, 2018
MENG Zhenhua. Research on high performance parallel computing architecture based on FPGA+DSP[D]. Beijing: The First Academy of China Aerospace Science and Technology Corporation, 2018 (in Chinese)

[19] 朱博灵. 基于配点法的通用轨迹优化方法研究与程序开发[D]. 南京: 南京航空航天大学, 2023
ZHU Boling. Research and program development of general trajectory optimization method based on collocation method[D]. Nanjing: Nanjing University of Aeronautics and Astronautics, 2023 (in Chinese)

A rapid trajectory optimization method based on parallel computing

WANG Tianyi, ZHAO Jisong

(School of Astronautics, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

Abstract: The direct collocation method transforms a trajectory optimization problem into a nonlinear programming (NLP) problem by discretizing both control and state variables. During the NLP solution process, repeated calculations of the first and second derivatives of the NLP and the values of the dynamic system at each discrete point are required, leading to great computational complexities. Therefore, this paper proposes the following method: First, the hyper-dual number method is introduced to accurately identify the sparsity of the second-derivative matrix of the NLP and to determine the locations of the non-zero elements. Then, a multi-core parallel approach is used to rapidly compute the non-zero elements of the first and second derivatives of the NLP as well as the values of the dynamic system at each discrete point. Finally, OpenMP is employed for programming calculation in the C++ environment to further enhance computational efficiency from the perspective of programming language. Simulation results demonstrate that the proposed method effectively improves the efficiency of trajectory optimization and its computational efficiency without compromising accuracy.

Keywords: trajectory optimization; nonlinear programming; sparsity; hyper-dual numbers; parallel computing

引用格式: *王天一, 赵吉松. 基于并行计算的轨迹快速优化方法研究[J]. 西北工业大学学报, 2025, 43(4): 659-667*
WANG Tianyi, ZHAO Jisong. A rapid trajectory optimization method based on parallel computing[J]. Journal of Northwestern Polytechnical University, 2025, 43(4): 659-667 (in Chinese)