

并行单纯形法

张博阳 211840196

June 2024

目录

- 1 单纯形法
 - 线性规划标准型
 - 单纯形法执行步骤

- 2 并行化及其代码实现
 - 并行单纯形表迭代
 - 类的设计及代码框架

目录

- ① 单纯形法
 - 线性规划标准型
 - 单纯形法执行步骤

- ② 并行化及其代码实现
 - 并行单纯形表迭代
 - 类的设计及代码框架

线性规划标准型

单纯形法适用于求解下述线性规划标准型：

$$\begin{aligned} \min \quad & f = c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

其中 $A \in M_{m \times n}(\mathbb{R})$, $b \geq 0 \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, $x \in \mathbb{R}^n$.

线性规划标准型

问题的解有三种情形：

- 存在有界解
- 存在无界解
- 无解

线性规划标准型

问题的解有三种情形：

- 存在有界解
- 存在无界解
- 无解

目录

- 1 单纯形法
 - 线性规划标准型
 - 单纯形法执行步骤

- 2 并行化及其代码实现
 - 并行单纯形表迭代
 - 类的设计及代码框架

单纯形法执行步骤

- 数据初始化，开辟存储判别数的 n 维数组，指定初始基变量
- 对于每一基变量($O(m)$)
 - 对基变量执行单位化($O(n)$)
 - 对单纯形表执行列消去运算($O(mn)$)
- 计算各变量判别数 $\gamma_j = c_j - \sum_{i \in base} c_i t_{ij} \quad (O(mn))$
- 检查各判别数($O(n)$)
 - 若判别数均不为负，表明最优解已求得，退出迭代
- 若存在负判别数，选择负判别数中绝对值最大者为进基变量，选择该变量所在列中 $\frac{t_{i,n+1}}{t_{ij^*}}$ 为正的最小行对应的基变量为出基变量($O(m)$)，更新基变量，转步骤2
 - 若各行均不为正，表明存在无界解，退出迭代

目录

- 1 单纯形法
 - 线性规划标准型
 - 单纯形法执行步骤

- 2 并行化及其代码实现
 - 并行单纯形表迭代
 - 类的设计及代码框架

并行单纯形表迭代

- 由于单纯形法是高度顺序化的，因此难以完全并行实现
 - 判别数的计算依赖于换基运算后得到的单纯形表
 - 进出基的选择依赖于判别数的计算
 - Barriers调用时机
- 考虑在每次迭代内部使用并行技术加速计算
 - 仅在列消去($O(m^2n)$)和判别数($O(mn)$)计算使用并行技术
 - 基变量确定的情况下，各行的消去是彼此独立的
 - 判别数计算是各变量间独立的
 - 其他步骤仍顺序执行

目录

- 1 单纯形法
 - 线性规划标准型
 - 单纯形法执行步骤

- 2 并行化及其代码实现
 - 并行单纯形表迭代
 - 类的设计及代码框架

类的设计及代码框架

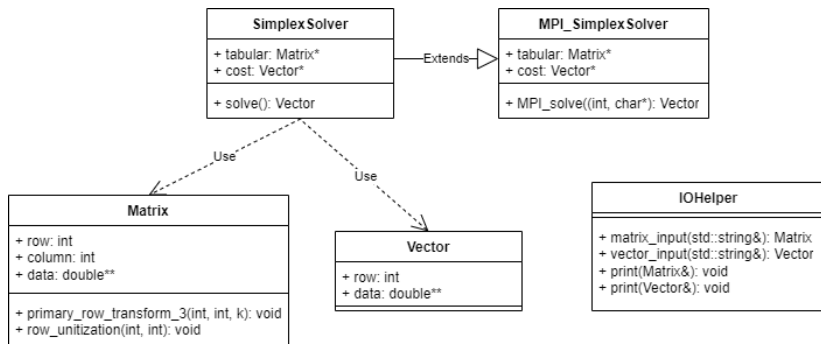
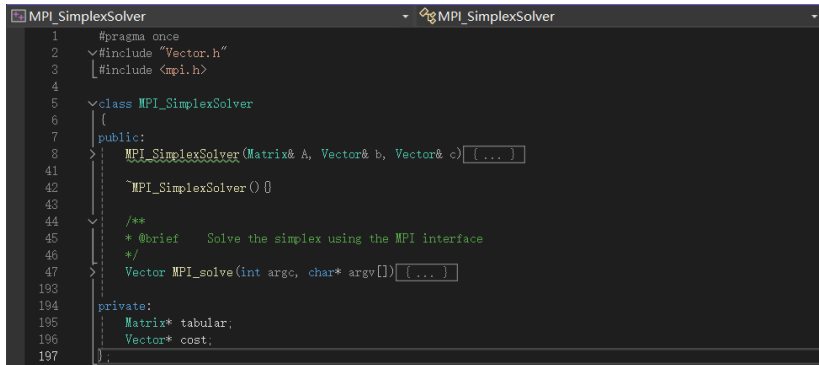


图: MPI_SimplexSolver类图

类的设计及代码框架



```
1  #pragma once
2  #include "Vector.h"
3  #include <mpi.h>
4
5  class MPI_SimplexSolver
6  {
7  public:
8      MPI_SimplexSolver(Matrix& A, Vector& b, Vector& c) { ... }
41
42      ~MPI_SimplexSolver() {}
43
44      /**
45       * @brief    Solve the simplex using the MPI interface
46       */
47      Vector MPI_solve(int argc, char* argv[]) { ... }
193
194  private:
195      Matrix* tabular;
196      Vector* cost;
197  };
```

图: 核心类MPI_SimplexSolver结构

类的设计及代码框架

```
MPI_SimplexSolver Matrix
108  /**
109   * @brief      Multiply row i by a constant k
110   * @param index The index of the row
111   * @param k      The constant
112   *
113   * @return      void
114   */
115  void primary_row_transform_2(int index, double k)
116  {
117      for (int i = 0; i < this->column; i++)
118      {
119          this->data[index][i] *= k;
120      }
121  }
122
123  /**
124   * @brief      Add row i to row j (column) k times
125   * @param augend_index The index of the augend row
126   * @param addend_index The index of the addend row
127   * @param k      The multiplier
128   *
129   * @return      void
130   */
131  void primary_row_transform_3(int augend_index, int addend_index, double k)
132  {
133      for (int i = 0; i < this->column; i++)
134      {
135          this->data[augend_index][i] += (k * this->data[addend_index][i]);
136      }
137  }
```

图: 类Matrix结构

类的设计及代码框架

```
MPI_SimplexSolver
Matrix

138
139 /**
140  * @brief      Unitize row i
141  * @param row_index  The index of the unitized row
142  * @param column_index  The index of the unitizator in the row
143  *
144  * @return      void
145  */
146 void row_unitization(int row_index, int column_index)
147 {
148     double divisor = 1 / data[row_index][column_index];
149     this->primary_row_transform_2(row_index, divisor);
150 }
151
152 /**
153  * @brief      Eliminate column j using row i (here Matrix[i, j] = 1)
154  * @param row_index  The index of the eliminator row
155  * @param column_index  The index of the eliminator in the row
156  *
157  * @return      void
158  */
159 void column_elimination(int row_index, int column_index)
160 {
161     for (int i = 0; i < this->row; i++)
162     {
163         if (i == row_index)
164         {
165             continue;
166         }
167         else
168         {
169             this->primary_row_transform_3(i, row_index, data[i][column_index]);
170         }
171     }
172 }
```