🚀 XJTU计算方法课程C++代码库

✨ 作者

动力工程及工程热物理 S5037 杨牧天

GitHub @SheepAndPiggy

Email danyangpinkpiggy@gmail.com

Version 1.0.0

Platform C++ | Vscode | Windows11

📖 简介

XJTU计算方法课程C++代码库 是一个西安交通大学研究生课程——计算方法的算法实现库，为作者练习C++编程以及学习计算方法所写，涉及的算法均在《数值分析》中详细讲解。

这个代码库希望为学习计算方法的同学们提供一个参考，引用需注明来源。

📦 安装

本代码库需要GCC编译环境和VsCode编辑器，使用时需要将 `.vscode` 目录下调试配置文件 `launch.json` 和任务配置文件 `tasks.json` 中的调试器、编译器可执行文件路径更改为本地对应路径。

.vscode\launch.json

```
{
    "miDebuggerPath": "C:\\Program Files\\mingw64\\bin\\gdb.exe",
}
```

.vscode\tasks.json

```
{
    "command": "C:\\Program Files\\mingw64\\bin\\g++.exe",
}
```

在使用时，通过在 `src\main.cpp` 中编写代码并编译运行，查看输出结果，也可以复用 `chapters` 下的工具 `utils.cpp` 和各章节算法 `chapter**.cpp` 。

使用示例

通过查看第二章头文件 `chapter02.hpp` ，可以看到不同样例的简介，其中包含《数值分析》书中课后的计算实习题目和额外的拓展内容。

include\chapter02.hpp

```
// 测试用例
struct Chapter02Demos{
    static void Demo1();  // 计算实习2.1
    static void Demo11(size_t n = 1000);  // 测试LU分解分解正定矩阵和求解的性能
    static void Demo2();  // 计算实习2.2
    static void Demo21(size_t n = 1000);  // 比较LU分解、楚列斯基分解、改进平方根法的计算速度
    static void Demo3();  // 计算实习2.3
    static void Demo4();  // 计算实习2.4
    static void Demo41();  // 使用吉文斯变换进行QR分解，计算计算实习2.4
};
```

在主文件中运行测试样例4

src\main.cpp

```
int main(){
    SetConsoleOutputCP(CP_UTF8);  // 强制控制台使用utf-8编码
    Chapter02Demos::Demo4();
    return 0;
}
```

得到输出结果

```
利用吉文斯变换进行QR分解结果：
Q矩阵：
   0.333  -0.337   0.288  -0.437  -0.098   0.222   0.665
   0.267   0.912  -0.014  -0.246  -0.118   0.071   0.132
   0.467  -0.150   0.196  -0.141  -0.570  -0.445  -0.423
   0.333   0.066   0.569   0.603   0.058   0.424  -0.117
   0.400   0.025  -0.136   0.378   0.381  -0.621   0.383
   0.467  -0.150  -0.729   0.156  -0.155   0.420  -0.055
   0.333  -0.069   0.064  -0.445   0.692   0.078  -0.445

R矩阵：
  15.000  19.533  20.933  19.933  21.600  21.267  19.800
   0.000   7.446   2.700   2.906   3.099   2.362   1.107
  -0.000   0.000   3.242   3.358   1.688  -0.481  -2.304
  -0.000   0.000  -0.000   3.734   0.741  -1.651  -1.114
   0.000  -0.000  -0.000   0.000   3.230   3.205   3.902
   0.000  -0.000  -0.000  -0.000   0.000   1.980  -0.074
   0.000  -0.000  -0.000  -0.000  -0.000   0.000  -0.979

求解结果：
   1.000   1.000   1.000   1.000   1.000   1.000   1.000
```

📂 项目内容

Chapter02 解线性方程组的直接法

> 本章节实现了列主元LU分解法、平方根法、改进平方根法、三对角矩阵追赶法、基于吉文斯变换的QR分解、基于豪斯霍尔德变换的QR分解

1. 列主元LU分解法

LU分解的过程和高斯消去法相同，其优势在于如果需要计算多个方程组 $Ax = B$ ，则LU分解将分解矩阵L和U保存后可以避免重复计算；此外，进行高斯消去法时常常会将列中绝对值最大的行置于主元的位置，这相当于对A左乘一个置换矩阵 P ，因此求解方程 $Ax = b$ 变为

$$\begin{aligned}\text{first step: } PA &= LU \\ \text{second step: } Ax &= P^T LUx = b \\ \text{third step: } Ly &= Pb;\ Ux = y\end{aligned}$$

在实际计算中，只需要求解两个上下对角阵的方程即可，通过回代很容易求解。LU分解类如下

```
class DoolittleSolver{  // LU分解亦称杜利尔特分解
    public:
    // lu分解（列主元），返回包含L和U的矩阵和列主元的行索引矩阵
    static luResult luDecompose(const Matrix& mat, bool pivot = false);

    // 根据上或下三角矩阵进行回代，求解方程组的解，返回方程组的解
    static Matrix solveByTri(const Matrix& tri, const Matrix& b, std::string tri_type);
};
```

题目：计算实习2.1

利用LU分解法求解线性方程组，已知线性方程组

$$A\mathbf{x} = \mathbf{b}$$

其中

$$A = \begin{bmatrix} 1.1348 & 3.8326 & 1.1651 & 3.4017 \\ 0.5301 & 1.7875 & 2.5330 & 1.5435 \\ 3.4129 & 4.9317 & 8.7643 & 1.3142 \\ 1.2371 & 4.9998 & 10.6721 & 0.0147 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 9.5342 \\ 6.3941 \\ 18.4231 \\ 16.9237 \end{bmatrix}.$$

调用程序(`Demo1()`)

```
luResult result = DoolittleSolver::luDecompose(A, true);  // 列主元LU分解

Matrix L = result.LU_mat.lower_tri();
Matrix U = result.LU_mat.upper_tri();
for (int i = 0; i < L.rows; ++i)
    L(i, i) = 1;  // 将L矩阵的对角线元素置为1

Matrix y = DoolittleSolver::solveByTri(L, result.LU_P * b, "low");  // 求解Ly = Pb
Matrix x = DoolittleSolver::solveByTri(U, y, "up");  // 求解Ux = y
```

得到结果

```
LU分解矩阵：
   3.413   4.932   8.764   1.314
   0.362   3.212   7.495  -0.462
   0.333   0.683  -6.866   3.280
   0.155   0.318   0.177   0.907


置换矩阵P：
   0.000   0.000   1.000   0.000
   0.000   0.000   0.000   1.000
   1.000   0.000   0.000   0.000
   0.000   1.000   0.000   0.000

求解结果：
   1.000   1.000   1.000   1.000
```

测试：列主元LU分解法（列主元高斯消去法）的计算效率

调用程序(`Demo11`)，得到结果

```
矩阵维度：1000
LU分解用时（s）：
1.32033
LU分解法方程组求解用时（s）：
0.0097692
```

将程序 `Demo11` 中矩阵维度设置为2000，得到结果

```
矩阵维度：2000
LU分解用时（s）：
11.2337
LU分解法方程组求解用时（s）：
0.0238054
```

可以看到，相比于高斯分解法，当方程组系数不变时，LU分解可以节省大量的系数矩阵消元时间；同时当矩阵维数增大时，由于LU分解时间复杂度为 $O(n^3)$ ，因此耗时大约增大为8倍，**对于求解大型矩阵，LU分解法并不适用**。

> 注：在实际使用过程中，专业计算库会对矩阵的求解过程进行深度优化并使用并行计算，可大大提升效率（用时约为本项目代码的十分之一）。

2. 平方根法和改进平方根法

对于对称正定矩阵，有楚列斯基分解

$$A = GG^T$$

其中 G 为下三角矩阵，对于G的非对角元 g_{ij} ，可以通过 a_{ij} 、 g_{ik} 、 g_{jk} 、 g_{jj} 确定($0 \leq k \leq j-1$)；对于G的对角元 g_{jj} 则需要根据 a_{jj} 、 g_{jk} 进行开方运算得到；矩阵整体通过从上至下，从左至右的迭代求解即可得到。

在使用楚列斯基分解求解对称正定矩阵时（亦称平方根法），过程如下

$$\begin{aligned}\text{first step: } & A = GG^T \\ \text{second step: } & Ax = GG^Tx = b \\ \text{third step: } & Gy = b;\ G^Tx = y\end{aligned}$$

由于楚列斯基分解需要计算n次开方运算，计算机时间成本较大，因此使用改进平方根法，改进平方根法通过LU分解中 u_{ij} 的计算公式，利用 l_{ik} 、 u_{kj} 计算得到($0 \leq k \leq i-1$)，并根据 LDL^T 这一特殊的分解形式得到 l_{ji} 的计算表达式，其本质上是特殊情况的LU分解，由于对称正定带来的特殊性其比标准LU分解具有更少的计算步骤。

在使用 LDL^T 分解求解对称正定矩阵时（亦称改进平方根法），过程如下

$$\begin{aligned}\text{first step: } & A = LDL^T \\ \text{second step: } & Ax = LDL^Tx = b \\ \text{third step: } & Lz = b;\ Dy = z;\ L^Tx = y\end{aligned}$$

平方根法和改进平方根法求解类如下

```
class SqrtMethodSolver{
    public:
    // 楚列斯基分解，返回分解后的矩阵G
    static Matrix choleskyDecompose(const Matrix& mat);

    // 改进平方根法，返回分解后的矩阵LU
    static Matrix improvedSqrtDecompose(const Matrix& mat);
};
```

题目：计算实习2.2

构造一个 20×20 的对称正定矩阵 A，元素由

$$A_{ij} = \min(i,j), \quad i,j = 1,\ldots,20$$

给出，并进行楚列斯基分解和 LDL^T 分解。

调用程序(Demo2)

```
Matrix G = SqrtMethodSolver::choleskyDecompose(A);
Matrix LU = SqrtMethodSolver::improvedSqrtDecompose(A);
```

得到结果

（G为元素为1的单位下三角矩阵）
（L为元素为1的单位下三角矩阵，D为单位矩阵）
（由于矩阵较大，此处使用文字描述，不做展示）

测试：LU分解法、平方根法、改进平方根法计算效率对比

调用程序 Demo21 ，得到结果

```
矩阵维度：1000
LU分解用时（s）：
1.41472
楚列斯基分解用时（s）：
0.673267
改进平方根法分解用时（s）：
0.699274
```

可以看到楚列斯基分解和 LDL^T 分解用时约为LU分解的一半，与书中结论相符，但楚列斯基分解用时小于 LDL^T 分解，这可能是由于**本项目的矩阵为作者手动实现的数据类型，其数据按行存储在连续内存块中，因此楚列斯基分解中涉及的按行取值相比于 LDL^T 分解的大量按列取值内存命中率更高**，且作者实现的 LDL^T 分解将中间变量 u_{ij} 存储在矩阵的上三角位置，增加了部分内存写入时间，不过经过资料查询，作者找到有力的证据证明楚列斯基分解更优秀的性能。

1. 专业代码库Eigen对 GG^T 分解和 LDL^T 分解进行了基准测试，用时如下表

solver/size	8x8	100x100	1000x1000	4000x4000	10000x8	10000x100	10000x1000	10000x4000
LLT	0.05	0.42	5.83	374.55	6.79 *	30.15 *	236.34 *	3847.17 *
LDLT	0.07 (x1.3)	0.65 (x1.5)	26.86 (x4.6)	2361.18 (x6.3)	6.81 (x1) *	31.91 (x1.1) *	252.61 (x1.1) *	5807.66 (x1.5) *

（表中的 LL^T 即为楚列斯基分解）

2. 论文PERFORMANCE OPTIMIZATION OF SYMMETRIC FACTORIZATION ALGORITHMS提到，聚焦在 SPD（对称正定） 情形下，楚列斯基分解比 LDL^T 分解更高效。原因主要为 Cholesky 对 SPD 矩阵不需要进行主元选择，而 LDL^T 为保证数值稳定性往往实现时需要进行主元选择，这将引发 LDL^T 分解的访存更离散、缓存命中率更低、依赖链更长等诸多问题。

3. 三对角追赶法

三对角追赶法计算过程较为简单，可以参考《数值计算》P35。

题目：计算实习2.3

给定一个严格对角占优的三对角对称矩阵，已知n=20，

$$A=\begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & -1 & 4 & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 4 \end{bmatrix}$$

$$\mathbf{b}=(3,2,\ldots,2,3)^{\top}.$$

求解方程组 $Ax = b$ 。

调用(Demo3)，输出结果为

```
三对角矩阵追赶法求解结果：
1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000
```

4. 基于吉文斯变换和豪斯霍尔德变换的QR分解

吉文斯变换通过构造矩阵 P_{1j} 不断消去矩阵每列的第j个元素，将矩阵化为上阶梯矩阵，其中构造的参数 s 和 c 要求

$$c = \frac{x_1}{\sqrt{x_1^2 + x_j^2}} \quad s = \frac{x_j}{\sqrt{x_1^2 + x_j^2}}$$

通过将每列主元之后的元素通过如上吉文斯变换转换为0，即可得到上阶梯矩阵

豪斯霍尔德变换通过构造变换矩阵 $H = I - 2uu^T$ ，可证明存在 $Hx = \sigma v$ ，其中 v 为单位向量，只需满足

$$\sigma = \pm \|x\|_2 \quad u = \frac{x - \sigma v}{\|x - \sigma v\|_2}$$

题目：计算实习2.4

利用豪斯霍尔德变换求解方程组

$$A = \begin{bmatrix} 5 & 4 & 7 & 5 & 6 & 7 & 5 \\ 4 & 12 & 8 & 7 & 8 & 8 & 6 \\ 7 & 8 & 10 & 9 & 8 & 7 & 7 \\ 5 & 7 & 9 & 11 & 9 & 7 & 5 \\ 6 & 8 & 8 & 9 & 10 & 8 & 9 \\ 7 & 8 & 7 & 7 & 8 & 10 & 10 \\ 5 & 6 & 7 & 5 & 9 & 10 & 10 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 39 \\ 53 \\ 56 \\ 53 \\ 58 \\ 57 \\ 52 \end{bmatrix}.$$

调用程序(Demo4)，得到结果

利用豪斯霍尔德变换进行QR分解结果：

Q矩阵：

```
  -0.333   0.337  -0.288   0.437   0.098   0.222   0.665
  -0.267  -0.912   0.014   0.246   0.118   0.071   0.132
  -0.467   0.150  -0.196   0.141   0.570  -0.445  -0.423
  -0.333  -0.066  -0.569  -0.603  -0.058   0.424  -0.117
  -0.400  -0.025   0.136  -0.378  -0.381  -0.621   0.383
  -0.467   0.150   0.729  -0.156   0.155   0.420  -0.055
  -0.333   0.069  -0.064   0.445  -0.692   0.078  -0.445
```

R矩阵：

```
 -15.000 -19.533 -20.933 -19.933 -21.600 -21.267 -19.800
   0.000  -7.446  -2.700  -2.906  -3.099  -2.362  -1.107
   0.000   0.000  -3.242  -3.358  -1.688   0.481   2.304
  -0.000  -0.000   0.000  -3.734  -0.741   1.651   1.114
   0.000  -0.000  -0.000   0.000  -3.230  -3.205  -3.902
   0.000  -0.000   0.000  -0.000   0.000   1.980  -0.074
  -0.000  -0.000   0.000  -0.000  -0.000   0.000  -0.979
```

求解结果：

```
   1.000   1.000   1.000   1.000   1.000   1.000   1.000
```

测试：利用吉文斯变换求解计算实习2.4

调用程序(Demo41)，得到结果

利用吉文斯变换进行QR分解结果：

Q矩阵：

```
   0.333  -0.337   0.288  -0.437  -0.098   0.222   0.665
   0.267   0.912  -0.014  -0.246  -0.118   0.071   0.132
   0.467  -0.150   0.196  -0.141  -0.570  -0.445  -0.423
   0.333   0.066   0.569   0.603   0.058   0.424  -0.117
   0.400   0.025  -0.136   0.378   0.381  -0.621   0.383
   0.467  -0.150  -0.729   0.156  -0.155   0.420  -0.055
   0.333  -0.069   0.064  -0.445   0.692   0.078  -0.445
```

R矩阵：

```
  15.000  19.533  20.933  19.933  21.600  21.267  19.800
   0.000   7.446   2.700   2.906   3.099   2.362   1.107
  -0.000   0.000   3.242   3.358   1.688  -0.481  -2.304
  -0.000   0.000  -0.000   3.734   0.741  -1.651  -1.114
   0.000  -0.000  -0.000   0.000   3.230   3.205   3.902
   0.000  -0.000  -0.000  -0.000   0.000   1.980  -0.074
   0.000  -0.000  -0.000  -0.000  -0.000   0.000  -0.979
```

求解结果：

```
   1.000   1.000   1.000   1.000   1.000   1.000   1.000
```

Chapter03 解线性方程组的迭代法

1. 雅可比迭代、高斯赛德尔迭代和超松弛迭代

三种迭代方法可写为通用迭代格式

$$x_{k+1} = Bx_k + g$$

对于雅可比迭代，有：

$$B = D^{-1}(E + F), \quad g = D^{-1}b$$

对于超松弛迭代，有：

$$B = (D - \omega E)^{-1}[(1 - \omega)D + \omega F], \quad g = \omega(D - \omega E)^{-1}b$$

对于高斯赛德尔迭代，为超松弛迭代 $\omega = 1$ 时的特例

当迭代收敛，意味着相邻两步步长差距不大，采取变化量和上一步 x 比值的模长可以得到收敛判据

$$\left\| \frac{x_{k+1} - x_k}{x_k} \right\| \leq \epsilon$$

该判据由函数 `convergenceError` 实现

```
double IterationSolver::convergenceError(const Matrix& x_old, const Matrix& x_new){
    double error = norm((x_new - x_old) / x_old);  // 相对误差
    return error;
}
```

当求解成功，意味着方程组残差 $Ax_k - b$ 足够小，因此可以通过判断残差的模长与 x 模长的比值大小判断是否求解成功

$$\frac{\|Ax_k - b\|}{\|x_k\|} \leq \epsilon$$

该判据由函数 `residualNormError` 实现

```
double IterationSolver::residualNormError(const Matrix& x){
    double error = norm(A * x - b);
    return error;
}
```

题目：计算实习3.1

用雅可比迭代法和高斯赛德尔迭代法求解方程

$$A = \begin{bmatrix} 2.52 & 0.95 & 1.25 & -0.85 \\ 0.39 & 1.69 & -0.45 & 0.49 \\ 0.55 & -1.25 & 1.96 & -0.98 \\ 0.23 & -1.15 & -0.45 & 2.31 \end{bmatrix}, \quad b = \begin{bmatrix} 1.38 \\ -0.34 \\ 0.67 \\ 1.52 \end{bmatrix}$$

使误差小于 10^{-3} ，并比较迭代次数

调用程序(`Demo1`)，得到结果

```
...   ...   ...   ...
第21步残差||Ax-b||/||x||:
0.000133206
*****求解收敛！*****
*****求解成功！*****
   0.884  -0.514  -0.086   0.297

...   ...   ...   ...
第15步残差||Ax-b||/||x||:
0.000
*****求解收敛！*****
*****求解成功！*****
   0.884  -0.514  -0.085   0.297
```

可以看到，高斯赛德尔迭代法比雅可比迭代法收敛更快（15 < 21）。

2. 共轭梯度法

共轭梯度法通过将解线性方程的问题转化为极值问题，可以使维度为n的对称正定矩阵在n步内收敛。

共轭梯度法的求解过程如下

$$
\begin{cases}
d^{(0)} = r^{(0)} = b - Ax^{(0)}, \\
\alpha_k = \dfrac{(r^{(k)})^\top d^{(k)}}{(d^{(k)})^\top A d^{(k)}}, \\
x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)}, \\
r^{(k+1)} = b - Ax^{(k+1)}, \\
\beta_k = -\dfrac{(r^{(k+1)})^\top A d^{(k)}}{(d^{(k)})^\top A d^{(k)}}, \\
d^{(k+1)} = r^{(k+1)} + \beta_k d^{(k)}.
\end{cases}
$$

题目：计算实习3.2

用共轭梯度法求解线性方程组 (Ax=b)，其中矩阵 (A) 与向量 (b) 定义为：

$$
A = \begin{pmatrix}
-2 & 1 & & & \\
1 & -2 & 1 & & \\
 & \ddots & \ddots & \ddots & \\
 & & 1 & -2 & 1 \\
 & & & 1 & -2
\end{pmatrix}_{n\times n}, \qquad
b = \begin{pmatrix}
-1 \\ 0 \\ \vdots \\ 0 \\ -1
\end{pmatrix}.
$$

其中 (A) 为三对角 Toeplitz 矩阵，主对角元为 (-2)，上下对角元为 (1)，其余元素为 (0)；向量 (b) 的首尾分量为 (-1)，中间分量为 (0)。

当矩阵阶数 (n) 分别取 (100)、(200)、(400) 时，用共轭梯度法计算并指出结果是否可靠。

调用程序(`Demo2`)，得到结果

```
维度为100的共轭梯度法过程共用89步，残差||Ax-b||/||x||为0.000950882
维度为200的共轭梯度法过程共用151步，残差||Ax-b||/||x||为0.000988049
维度为400的共轭梯度法过程共用142步，残差||Ax-b||/||x||为0.000993269
```

可以看到，残差均收敛达到标准（$< 10^{-3}$），且步数均小于矩阵维度n。

3. 基于伽辽金原理的迭代法

1. 伽辽金原理的数学含义

设 $A \in R^{n\times n}$ ， $V = \{v_1, v_2, \cdots, v_m\}$ 为 R^n 中的m维子空间，由于

$$\nabla(\frac{1}{2}x^TAx - b^Tx) = Ax - b$$

因此求解线性方程组的问题等价为求解 $F(x) = \frac{1}{2}x^TAx - b^Tx$ 的极小值，对于真解 x^* 和近似解 x ，可以近似认为

$$x^* = \arg\min_{x\in R^n} F(x)$$
$$e = x^* - x$$

因此寻找最优的 x 等价为使得误差 e 最小，而越小的误差e意味着 $F(x)$ 和 $F(x^*)$ 的差值越小，因此问题等价为

$$\arg\min_{x\in R^n} F(x) - F(x^*)$$
$$F(x) - F(x^*) = \frac{1}{2}x^TAx - b^Tx - \frac{1}{2}x^{*\top}Ax^* + b^Tx^*$$
$$\text{since}\ \ b = Ax^*$$
$$F(x) - F(x^*) = \frac{1}{2}e^TAe \propto \|e\|_A$$

如果近似解 $x \in V$ ，则问题可以进一步简化为使残差 $r = A(x^* - x)$ 正交于 V 的所有基

$$\text{def}\ \ \phi(x) = \|x^* - x\|_A = (x^* - x)^TA(x^* - x)$$
$$\text{since}\ \ \frac{d\phi(x + tv)}{dt} = -(x^* - x)^TAv = -r^Tv. \quad \forall v \in V$$
$$\text{that}\ \ \arg\min_{x\in V}\phi(x) \iff r^Tv_i = 0. \quad \forall v_i \in B, \quad V = spanB$$

2. 思考：共轭梯度法与伽辽金原理的联系

本质上，共轭梯度法也是基于伽辽金原理的原理型方法，不同于一般基于伽辽金原理的算法——例如阿诺尔迪过程需要求解上海森伯格矩阵组成的线性方程组(维度为 $m \times m$)，共轭梯度法通过构造一组共轭向量 $d_i, d_i \in V_k$ ，通过三项递推公式使得 $r_k \perp V_k$ ，通过不断迭代更新 V_k 求解，其中 $r_k, d_k \in V_{k+1}$ 可以通过数学归纳法证明

$$d_0 = r_0 \in V_1 = \{r_0\}$$
$$\text{if}\ \ d_k, r_k \in V_{k+1}$$
$$\text{then}\ \ r_{k+1} = b - Ax_{k+1} = r_k - A\alpha_kd_k \in V_{k+2}$$
$$d_{k+1} = r_{k+1} + \beta_kd_k \in V_{k+2}.$$

因此本质上，共轭梯度法为基于伽辽金原理的方法，二者具有共性。

题目：计算实习3.3

给定线性方程组 (A x = d)，其中矩阵 (A) 为分块三对角矩阵

$$A = \begin{pmatrix} B & -I & & \\ -I & B & -I & \\ & \ddots & \ddots & \ddots \\ & & -I & B \end{pmatrix}_{200\times 200},$$

这里每个块均为 10×10 阶矩阵：$B \in \mathbb{R}^{10\times 10}$，(I) 为 10×10 单位矩阵。于是 (A) 由 20×20 个块组成，总阶数为 (200)。

分块对角元 (B) 为三对角矩阵

$$B = \begin{pmatrix} 4 & a & & & \\ b & 4 & a & & \\ & \ddots & \ddots & \ddots & \\ & & b & 4 & a \\ & & & b & 4 \end{pmatrix}_{10\times 10},$$

其中参数

$$a = -1 + \delta, \qquad b = -1 - \delta, \qquad \delta = 0.01.$$

右端向量由

$$d = Af, \qquad f = (1, 1, \ldots, 1)^{\mathsf{T}}$$

给出。显然该方程的精确解为

$$x = f.$$

可任意选取初始向量 $x^{(0)}$。

调用程序(Demo3)

```
Chapter03Demos::Demo3(10);
Chapter03Demos::Demo3(20);
```

得到结果：

```
循环型阿诺尔迪算法开始！矩阵维度为: 100
初始残差为421.412
第1步残差为1.17651
第2步残差为0.0632076
第3步残差为0.00157015
第4步残差为9.94872e-05
循环型阿诺尔迪算法开始！矩阵维度为: 200
初始残差为845.811
第1步残差为1.71807
第2步残差为0.182956
第3步残差为0.025624
第4步残差为0.00399607
第5步残差为0.000811107
```

可以看到，对于题目中特定结构的矩阵，循环型阿诺尔迪算法收敛速度较快，100维矩阵和200维矩阵收敛步长仅相差一步。

测试：循环型广义极小残余算法

作为同样基于伽辽金原理的广义极小残余算法，其收敛性在一定条件下具有严格的数学证明，因此被广泛应用于大型稀疏线性方程求解，作者实现了广义极小残余算法，具体可参考 `chapter03` 源代码。

调用程序(`Demo31`)，得到结果

```
对基于伽辽金原理的大型数组迭代解法进行测试，A为随机对称正定矩阵，b为[-10,10]随机矩阵：
维度为1000的共轭梯度法过程共用13步，残差||Ax-b||/||x||为7.97012e-05
循环型阿诺尔迪算法开始！矩阵维度为: 1000
初始残差为1.00335e+11
第1步残差为8.23106e-05
循环型广义极小残余(GMRES)算法开始！矩阵维度为: 1000
初始残差为1.00335e+11
第1步残差为9.45652e-05
```

可以看到，循环型阿诺尔迪算法和循环型广义极小残余算法对于大型对称正定矩阵的求解比共轭梯度法步数更短。

Chapter04 插值法

1. 拉格朗日插值法和牛顿插值法及其分段实现

拉格朗日插值法与牛顿插值法形式等价，相比于拉格朗日插值法，牛顿插值法更适合修改和多次计算，其基本可以替代一般的拉格朗日插值法，不过为了理解算法原理，首先学习表达式更美观的拉格朗日插值法是具有一定价值的。此外，工业界常用的是 `barycentric Lagrange` ，并不是课本最原始那种写法。

由于多项式等距插值当次数较高时存在**龙格现象**，即插值函数在区间两侧会发生数值振荡，因此一般的解决办法为分段低次插值或者采用不等距节点分布例如Chebyshev节点，本节实现分段低次插值的逻辑为先构造任意区间段的插值函数，再将区间分段，每段对应一个插值函数，当实际插值时，通过判断插值点落在哪个区间决定调用哪个插值函数，具体实现如下所示

```
class InterpolateTool{
    public:
    InterpolateTool(Matrix x, Matrix y);

    InterpolateResult lagrangeInterpolation(int seg_num = 1);  // 拉格朗日插值函数，返回一个插值函数指针和一个计算截断误差
    InterpolateResult newtonInterpolation(int seg_num = 1);  // 牛顿插值函数，返回值同上

    private:
    double n;
    Matrix x;
    Matrix y;

    static double _langrangeFunction(Matrix xs, Matrix ys, double x); // 拉格朗日插值分段计算函数生成器
    static double _newtonFunction(Matrix xs, Matrix ys, double x);  // 牛顿插值分段计算函数生成器

    std::vector<SepInterpolateResult> _sep_function(int seg_num, double (*function)(Matrix, Matrix, double));  // 一个通
    InterpolateResult _function(int seg_num, double (*function)(Matrix, Matrix, double));  // 一个通用的插值函数生成器，
    double _error(Matrix xs, Matrix ys, double x, double (*function)(Matrix, Matrix, double));  // 一个通用的截断误差函数
};
```

通过这种方式可以实现分段低次插值的基本调度，当 `seg_num` 取值为1时意味着区间只有一段即不分段插值。

> 注：由于埃米尔特插值等价于重复节点的牛顿插值，因此不在此赘述

2. 三次样条插值

1. 三次样条的来源

书中提到，早期的绘图员用压铁把一条称为样条的富有弹性的细木条或金属条固定在相邻的若干点上，然后沿着样条画下的即为二阶导数连续的三次曲线，作者看到这里是有一点疑问，为什么这样做画下的曲线就是二阶导数连续且是三次的呢？为此作者重新翻看了材料力学书籍，总结出如下证明过程

- 为什么是三次的?

可以将曲线理解为随位置变化的挠度函数 $\omega(x)$ ，假设杆上存在一个分布力 $q(x)$ ，因此对杆上任意一个微小单元列力平衡方程和弯矩平衡方程可以得到

$$
\begin{aligned}
V(x)-(V(x)+dV)-q(x)=0 \\
M(x)-(M(x)+dM)+V(x)dx+q(x)dx\cdot\frac{dx}{2}=0
\end{aligned}
$$

得到

$$
\begin{aligned}
V'(x)=-q(x) \\
M'(x)=V(x)
\end{aligned}
$$

因此根据弯矩-曲率方程和曲率-挠度在小挠度下的关系可以得到

$$
\begin{aligned}
M(x)=EI\kappa(x)=EI\omega''(x) \\
EI\omega''''(x)=-q(x)
\end{aligned}
$$

由于压铁仅在有限点上对细杆进行固定，因此可以认为相邻两个固定点之间不存在分布载荷 $q(x)$ ，即 $q(x)=0$ ，因此可以得出

$$
\omega''''(x)=0,\qquad x\in(x_i,x_{i+1})
$$

因此通过多次积分可以得到挠度分布为三次多项式，即

$$
\omega(x)=ax^3+bx^2+cx+d,\qquad x\in(x_i,x_{i+1})
$$

- 为什么二阶导数连续?

根据材料力学的原理，集中力 P 不会导致弯矩突变，这是由于

$$
0<M(x_i^+)-M(x_i^-)=\int_{x_i^-}^{x_i^+}V(x)dx<P\epsilon
$$

其中 ϵ 为一个极小值，由于 P 为常量，因此该值无限趋近于0，根据夹逼定理证得弯矩的连续性。事实上，只有集中弯矩会导致弯矩的突变。

由于弯矩连续，根据弯矩和挠度的关系，挠度的二阶导数因此也必须连续。

题目：计算实习4.1

观察高次插值多项式的龙格现象

调用程序(`Demo1`)，得到结果

****
拉格朗日插值（不分段）在x=0.01处的插值结果：
截断误差为：-2.63942e-05
插值结果为：0.997513
真实结果为：0.997506
相对误差为：0.000653256%

拉格朗日插值（不分段）在x=0.13处的插值结果：
截断误差为：1.3215e-05
插值结果为：0.702961
真实结果为：0.702988
相对误差为：0.00373654%

拉格朗日插值（不分段）在x=0.43处的插值结果：
截断误差为：-0.00158343
插值结果为：0.179582
真实结果为：0.177857
相对误差为：0.970245%

拉格朗日插值（不分段）在x=0.73处的插值结果：
截断误差为：-0.172563
插值结果为：0.129189
真实结果为：0.0698202
相对误差为：85.0304%

拉格朗日插值（不分段）在x=0.99处的插值结果：
截断误差为：-226249
插值结果为：2195.77
真实结果为：0.0392118
相对误差为：5.59967e+06%

****

****
牛顿插值（不分段）在x=0.01处的插值结果：
截断误差为：-2.63942e-05
插值结果为：0.997513
真实结果为：0.997506
相对误差为：0.000653256%

牛顿插值（不分段）在x=0.13处的插值结果：
截断误差为：1.3215e-05
插值结果为：0.702961
真实结果为：0.702988
相对误差为：0.00373654%

牛顿插值（不分段）在x=0.43处的插值结果：
截断误差为：-0.00158343
插值结果为：0.179582
真实结果为：0.177857
相对误差为：0.970245%

牛顿插值（不分段）在x=0.73处的插值结果：

截断误差为：-0.172563
插值结果为：0.129189
真实结果为：0.0698202
相对误差为：85.0304%

牛顿插值（不分段）在x=0.99处的插值结果：
截断误差为：-226249
插值结果为：2195.77
真实结果为：0.0392118
相对误差为：5.59967e+06%

****

****
拉格朗日插值（分10段）在x=0.01处的插值结果：
截断误差为：0.00483846
插值结果为：0.996704
真实结果为：0.997506
相对误差为：0.0804394%

拉格朗日插值（分10段）在x=0.13处的插值结果：
截断误差为：-0.0019
插值结果为：0.70335
真实结果为：0.702988
相对误差为：0.0515375%

拉格朗日插值（分10段）在x=0.43处的插值结果：
截断误差为：-0.000277858
插值结果为：0.177948
真实结果为：0.177857
相对误差为：0.0512919%

拉格朗日插值（分10段）在x=0.73处的插值结果：
截断误差为：1.79439e-05
插值结果为：0.0698184
真实结果为：0.0698202
相对误差为：0.00259289%

拉格朗日插值（分10段）在x=0.99处的插值结果：
截断误差为：-0.00028676
插值结果为：0.0392151
真实结果为：0.0392118
相对误差为：0.00828196%

****

****
牛顿插值（分10段）在x=0.01处的插值结果：
截断误差为：0.00483846
插值结果为：0.996704
真实结果为：0.997506
相对误差为：0.0804394%

牛顿插值（分10段）在x=0.13处的插值结果：

截断误差为：-0.0019
插值结果为：0.70335
真实结果为：0.702988
相对误差为：0.0515375%

牛顿插值（分10段）在x=0.43处的插值结果：
截断误差为：-0.000277858
插值结果为：0.177948
真实结果为：0.177857
相对误差为：0.0512919%

牛顿插值（分10段）在x=0.73处的插值结果：
截断误差为：1.79439e-05
插值结果为：0.0698184
真实结果为：0.0698202
相对误差为：0.00259289%

牛顿插值（分10段）在x=0.99处的插值结果：
截断误差为：-0.00028676
插值结果为：0.0392151
真实结果为：0.0392118
相对误差为：0.00828196%

****

****
三次样条插值在x=0.01处的插值结果：
插值结果为：0.997316
真实结果为：0.997506
相对误差为：0.019119%

三次样条插值在x=0.13处的插值结果：
插值结果为：0.703047
真实结果为：0.702988
相对误差为：0.00841251%

三次样条插值在x=0.43处的插值结果：
插值结果为：0.177849
真实结果为：0.177857
相对误差为：0.00444366%

三次样条插值在x=0.73处的插值结果：
插值结果为：0.0698202
真实结果为：0.0698202
相对误差为：2.05837e-05%

三次样条插值在x=0.99处的插值结果：
插值结果为：0.0392422
真实结果为：0.0392118
相对误差为：0.0775073%

****

可以看到不分段的拉格朗日插值和牛顿插值法在插值区间边界处精度很差，采用分段低次插值可以大大改善这一结果，此外，三次样条插值也可以很好的改善龙格现象。

Chapter05 函数最优逼近