微振动系统的本征振动频率与模式的数值计算探究

王治平

2020 级 物理学二班
320200908151

指导老师：吴枝喜、关剑月

2021 年 11 月 14 日

摘要：微振动系统的本征频率和本征模式问题已有不少人讨论过，本次课题首先复现出他人做过的五自由度的微振动问题，之后进行推广，得到任意自由度的数值计算结果。之后尝试与理论数据比较，进行理论和算法的改进与修正入手，复现出网站的进展，接下来进行理论推广，得到了多自由度的微振动系统的本征问题的数值计算。期间进行了时间复杂度的多次优化和算法改进。本文最后给出了结果较为理想的数值计算思路与数值结果，同时给出对应的解析解，并指出了部分的优化算法思路与实现。

关键字：微振动系统，解析求解，对比，本征问题，多自由度，*python*，近似选取，算法优化

课题程序实现已上传至 $https://github.com/YingD0/level2/tree/main/work2$

目录

1 课题介绍

1.1 课题介绍

- **微振动系统**

振动是指系统对于平衡位形的某种周期性偏离。而微振动是指这种偏移量极小的振动。一般情况下的处理方法列出力学方程之后进行简谐近似来进行处理分析。

- **本征振动频率与模式**

对于一个多自由度的微振动系统，我们可以选取系统自由度的适当线性组合构成简正坐标（又称简正模），这些简正坐标的简谐振动是互相独立的，其相应的，由力学性质与参数决定的频率被称为本征振动频率（又称作本征频率）。

1.2 课题简介

本课题旨在对一种简单的无阻尼五自由度系统的振动进行简要分析[1]的振动能级进行数值计算，期间用到了一些力学基础知识以及线性代数以及数值计算技巧，并对与结果进行了相关讨论。

[1]本项目中提到的五自由度系统来源与 $https://lpsa.swarthmore.edu/MtrxVibe/Anims/VibrationAnimations.html$ 中的 $FifthOrderSystem$

2 理论分析

2.1 一般性原理介绍

2.1.1 分析步骤

本课题的目的是分析系统的振动模式与频率，一般来讲对于一般的微振动我们可以按照如下步骤进行理论分析。[7]

(i) 写出体系的拉格朗日量

(ii) 对体系能量进行简谐近似

(iii) 将近似之后的拉格朗日量代入拉格朗日方程

一般情况下, 由于自由度之间的耦合我们得到的运动方程较为复杂，方便起见一般会对方程进行一些可控变换达到脱耦

(iv) 用傅里叶变换或者二次型对角化方法脱耦

(v) 利用脱耦之后的拉格朗日方程，求出本征频率与简正模

(vi) 结合给出的初始条件，计算出每个简正模式的贡献，得出运动方程

2.1.2 自由度为 s 情况的分析

以上步骤陈列过于泛泛，实际价值不高，下面我们以自由度为 s 的情况进行具体分析：[7][8]

(i) 写出体系的拉格朗日量

按照惯例用 q_i 表示广义坐标,a_{ij} 表示动能系数，对于一般的保守系，系统的势能 U 是且仅是 $q_i(i=1,2,3,4\ldots,s)$ 的函数，故动能可以写作:

$$T=\frac{1}{2}\sum_{i,j}a_{ij}(q)\dot{q}_i\dot{q}_j \tag{1}$$

拉格朗日量 $\mathscr{L}$ 可写作

$$\mathscr{L}=\frac{1}{2}\sum_{i,j}a_{ij}(q)\dot{q}_i\dot{q}_j-U(q) \tag{2}$$

(ii) 对体系能量进行简谐近似

由微振动的定义，U 在 $q_i=q_{i0}$ 处取极小，同时可以将偏离平衡位置的小位移表示为:

$$x_i=q_i-q_{i0} \tag{3}$$

把运动视作简谐振动，同时把展开并保留到二阶项可以得到 (注意到 $q_i=q_{i0}$ 时有

势能最小)

$$U(q) = U(q_0) + \frac{1}{2}\sum_{i,j} k_{ij}x_i x_j \tag{4}$$

拉格朗日量可以写作

$$\mathscr{L} = \frac{1}{2}\sum_{i,j}(m_{ij}\dot{x}_i\dot{x}_j - j_{ij}x_i x_j) \tag{5}$$

(iii) 将近似之后的拉格朗日量代入拉格朗日方程

列出拉格朗日量的全微分

$$d\mathscr{L} = \frac{1}{2}\sum_{ij}(m_{ij}\dot{x}_i d\dot{x}_j + m_{ij}\dot{x}_j d\dot{x}_i - k_{ij}x_i dx_k - k_{ij}x_j dx_i) \tag{6}$$

在具体问题中, 我们总可以认为 m_{ij} 和 k_{ij} 都对下标是对称的即 $m_{ij} = m_{ji} k_{ij} = k_{ji}$（$m$ k 均为实对称矩阵）由此可以得出

$$\frac{\partial \mathscr{L}}{\partial \dot{x}_i} = \sum_j m_{ij}\dot{x}_j$$

$$\frac{\partial \mathscr{L}}{\partial x_i} = -\sum_j k_{ij}x_j$$

拉格朗日方程为

$$\sum_j m_{ij}\ddot{x}_j + \sum_k k_{ij}x_j = 0 \tag{7}$$

显然的，方程具有通解:

$$x_k = \zeta_k e^{-i\omega t} \qquad (\omega^2 = \frac{k}{m}) \tag{8}$$

将其代入方程 (6) 我们可以得到常数 ζ_j 满足的线性方程组

$$\begin{cases} \sum_j (-\omega^2 m_{1j} + k_{1j})\zeta_j = 0 \qquad (1) \\ \sum_j (-\omega^2 m_{2j} + k_{2j})\zeta_j = 0 \qquad (2) \\ \qquad\qquad \cdots\cdots \\ \sum_j (-\omega^2 m_{sj} + k_{sj})\zeta_j = 0 \qquad (s) \end{cases} \tag{9}$$

同时此方程等价于

$$det[\boldsymbol{k} - \omega^2 \boldsymbol{m}] = 0 \tag{10}$$

(iv) 用傅里叶变换或者二次型对角化方法脱耦

一般情况下, 上述方程不好给出一个合适的解, 我们常用里叶变换或者二次型对角化这两种方法进行一些去耦的简化, 本次课题先就二次型对角化进行讨论。同时根据下述定理:

定理 1 任何一个厄米的矩阵总可以通过一个幺正变换将其对角化。而对于一个实对称矩阵，总可以找到一个正交变换将其对角化

因此我们总可以通过一个正交变化将实对称矩阵对角化, 可以证明给定现行变换可以同时将两个二次型动能 (1) 势能 (4) 同时对角化.

设正交变换 P_1, 通过正交变换 $x = P_1 \cdot y$ 使得动能的二次型变为对角的

$$T = \frac{1}{2}\sum_{i=1}^{n} m_i \dot{y_i}^2 \tag{11}$$

由于动能总是正定的, 因此 $m_i > 0$ 是正的实数. 注意, 在这个变换下, 势能 $V = (1/2)y^{\mathrm{T}}K'y$, 其中 $K' = (P_1)^{\mathrm{T}}\, kP_1$ 仍然是实对称的正定矩阵. 我们现在可以令 $z_i = \sqrt{m_i}y_i$, 或者等价地写为 $z = (M_{\mathrm{D}})^{1/2}\, y$, 其中 M_{D} 是对角的矩阵, 其矩阵元就是上述各个 m_i. 这样一来动能部分变为 $T = (1/2)z^{\mathrm{T}}z$, 而势能部分则由一个实对称正定矩阵 K'' 描写:

$$T = \frac{1}{2}z^{\mathrm{T}}z \tag{12}$$

$$V = \frac{1}{2}z^{\mathrm{T}}K''z \tag{13}$$

其中 $K'' = M_{\mathrm{D}}^{-1/2}K'M_{\mathrm{D}}^{-1/2}$. 由于 K'' 仍然是实对称矩阵, 我们可以进一步寻找将 K'' 对角化的正交矩阵 P_2. 也就是说, 我们令

$$z = P_2 \cdot Q$$

并要求它将势能对角化. 由于势能部分也是正定的, 不失一般性我们令其对角元为 $\omega_i^2 > 0$. 由于动能部分已经化为单位矩阵的形式, 因此它在任何正交矩阵变换下是不变的, 仍然保持原形式. 根据上述一系列变换

我们将动能和势能同时对角化了:

$$\mathscr{L} = \sum_{i=1}^{n} \frac{1}{2}\left(\dot{Q}_i^2 - \omega_i^2 Q_i^2\right) \tag{14}$$

其中联系 Q 和 x 的矩阵 $A = P_1 M_{\mathrm{D}}^{-1/2} P_2$ 通常被称为模态矩阵 (modal matrix)

简正坐标与原坐标之间的关系为

$$Q = \left(P_1 M_{\mathrm{D}}^{-1/2} P_2\right)^{-1} \cdot x = A^{-1} \cdot x, \quad A \equiv P_1 M_{\mathrm{D}}^{-1/2} P_2$$

. 这样的一组 Q_i 称为简谐振动系统的简正坐标, (又称为简正模). 因此, 利用简正坐标 Q 来表达, 多自由度的简谐振动问题完全简化为独立的单自由度系统的简谐振动问题.

(v) 利用脱耦之后的拉格朗日方程，求出本征频率与简正模

上面的线性代数证明仅仅是从存在性上, 具体操作层面, 最常用的方法是直接求解本征方程的非零本征矢.

对于方程 (10)

$$det[\boldsymbol{k} - \omega^2 \boldsymbol{m}] = 0$$

ω 显然有 s 个非负实数解 (此处仅讨论无简并的情况, 简并问题后续讨论), 对于每一个 ω_i 我们都可以找到一个非零本征矢 $\boldsymbol{\zeta}_i$

其满足

$$\omega_i{}^2 \boldsymbol{m} \cdot \zeta_i - \boldsymbol{k} \cdot \zeta_i = 0 \tag{15}$$

不难证明, 集合 $\{\boldsymbol{\zeta}_1, \boldsymbol{\zeta}_2, \ldots, \boldsymbol{\zeta}_i, \ldots, \boldsymbol{\zeta}_s\}$ 具有正交性, 即其可以作为 s 维空间的一组正交完备基 (如有需要可以继续进行归一化)

即模态矩阵可以用

$$\mathbf{A}_{ji} = \tilde{\boldsymbol{\zeta}}_j^{(i)} \tag{16}$$

给出, 同时不难证明矩阵 $\mathbf{A}$ 可以恰好将矩阵 $\mathbf{k}$ 对角化, 对角元为 ω_i^2, 即

$$\mathbf{A^T k A} = Diag(\omega_1^2, \omega_2^2, \ldots, \omega_s^2) \tag{17}$$

(vi) 结合给出的初始条件，计算出每个简正模式的贡献，得出运动方程

我们可以由以下事实出发:

$$x = \mathbf{AQ} \tag{18}$$

并具有以下的矩阵关系

$$\mathbf{A^T m A} = \mathbf{I} \tag{19}$$

以及关系 (16)

$$\mathbf{A^T k A} = Diag(\omega_1^2, \omega_2^2, \ldots, \omega_s^2)$$

每一简正模 Q_i 都是以确定的频率 ω_i 进行简谐振动

$$Q_i(t) = \gamma_{1i}\ cos(\omega_i) + \gamma_{2i}\ sin(\omega_i) \tag{20}$$

因此, 最终的 $x(t)$ 的解为:

$$x_j(t) = \sum_{i=0}^{n} \gamma_{1i}\ cos(\omega_i) + \gamma_{2i}\ sin(\omega_i) \tag{21}$$

矩阵表达为

$$\boldsymbol{x}(t) = \boldsymbol{\gamma_1}\ cos(\omega_i t) + \boldsymbol{\gamma_2}\ sin(\omega_i t) \tag{22}$$

2.2 具体理论生成

本课题我们主要讨论一种五自由度的微振动系统，在后面部分会进行 n 自由度的扩展讨论如无特殊说明均为五自由度的情况)。

对于具体实例的介绍分析见下。

2.2.1 实例系统介绍

在本课题中我们称以下系统为“微振动系统”

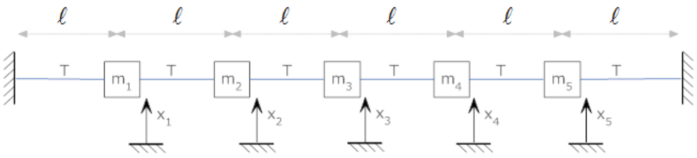


图 1: 微振动系统示意图

如图, 5 个质量均为 m 的物块在均匀张力 T_n 下的弦上物块进行距离 l 的等距分布。每个物体 m_i 均只可以在竖直方向进行微振动, 并且对每个物块都有一个竖直方向的位移 x_i.

2.2.2 对于研究对象的实际分析

显然的, 本系统符合微振动的定义, 故而我们可以通过之前提到的分析方法进行实际分析.

再次陈列一下基本步骤

(i) 写出体系的拉格朗日量

(ii) 对体系能量进行简谐近似

(iii) 将近似之后的拉格朗日量代入拉格朗日方程

(iv) 用傅里叶变换或者二次型对角化方法脱耦

(v) 利用脱耦之后的拉格朗日方程，求出本征频率与简正模

(vi) 结合给出的初始条件，计算出每个简正模式的贡献，得出运动方程

对于本题具体情况，我们对于基本步骤进行了一些简化, 具体步骤如下

(i) 写出体系的拉格朗日量

首先进行系统力学分析，不妨对于下图这种情况进行分析

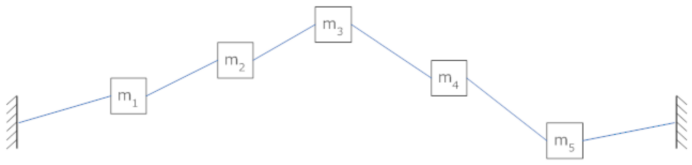


图 2: 微振动系统状态示意图

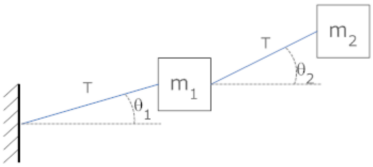


图 3: 微振动系统

由图 2,3 我们可以作出受力分析如下

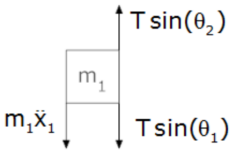


图 4: 受力分析图

由此我们可以得到系统中每个物体的动能和势能

$$T = \sum_{i=0}^{5} \frac{1}{2} m_i \dot{x}_i \tag{23}$$

$$V=\sum_{i=0}^{5}T_n\left(-sin\theta_{l_i}x_i+sin\theta_{r_i}x_i\right) \tag{24}$$

由于微振动，其中

$$sin\theta_{l_i}=\frac{xi}{l},\quad sin\theta_{r_i}=\frac{\Delta x_i}{l} \tag{25}$$

可以写出拉氏量为:

$$\mathscr{L}=\sum_{i=0}^{5}\frac{1}{2}m_i\dot{x}_i+V(i) \tag{26}$$

其中 V 的大小与物块受力有关，其中受力与物块位置有关（受力数目）

(ii) 将近似之后的拉格朗日量代入拉格朗日方程

代入

$$\frac{d}{dt}\frac{\partial\mathscr{L}}{\partial\dot{x}_i}+\frac{\partial\mathscr{L}}{\partial x_i}=0 \tag{27}$$

故拉格朗日方程为

$$\begin{cases}\ddot{x}_1= & -2\dfrac{T_n}{m}x_1 & +\dfrac{T_n}{m}x_2\\ \ddot{x}_2=\dfrac{T_n}{m}x_1 & -2\dfrac{T_n}{m}x_2 & +\dfrac{T_n}{m}x_3\\ \ddot{x}_3=\dfrac{T_n}{m}x_2 & -2\dfrac{T_n}{m}x_3 & +\dfrac{T_n}{m}x_4\\ \ddot{x}_4=\dfrac{T_n}{m}x_3 & -2\dfrac{T_n}{m}x_4 & +\dfrac{T_n}{m}x_5\\ \ddot{x}_5=\dfrac{T_n}{m}x_4 & -2\dfrac{T_n}{m}x_5 & \end{cases} \tag{28}$$

可以写为矩阵形式为

$$\ddot{\boldsymbol{x}}=\boldsymbol{A'}\cdot\boldsymbol{x}=\frac{T_n}{m}\boldsymbol{A}\cdot\boldsymbol{x}=\frac{T_n}{m}\begin{bmatrix}-2&1&0&0&0\\1&-2&1&0&0\\0&1&-2&1&0\\0&0&1&-2&1\\0&0&0&1&-2\end{bmatrix}\boldsymbol{x} \tag{29}$$

(iii) 利用脱耦之后的拉格朗日方程，求出本征频率与简正模

对矩阵 $\boldsymbol{A'}$ 求解本征值与本征矢即对矩阵 $\boldsymbol{A}$ 求本征值与本征矢; 本征值即对应的本

征频率的平方，本征矢即本征频率对应的振动矢量。利用后续介绍的数值计算方法求解本征值与本征矢即可。

(iv) 结合给出的初始条件，计算出每个简正模式的贡献，得出运动方程

上一步我们可以得到 $\{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5\}$ 五个本征频率和 $\{\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}, \boldsymbol{v_4}, \boldsymbol{v_5}\}$ 五个本征矢，不妨令 $\boldsymbol{V} = [\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}, \boldsymbol{v_4}, \boldsymbol{v_5}]$ 不难得到

$$\boldsymbol{V}\boldsymbol{\gamma}_1 = \boldsymbol{x}(0) \tag{30}$$

$$\boldsymbol{V}\ \boldsymbol{Diag}(\omega_1, \omega_2, \omega_3, \omega_4, \omega_5)\boldsymbol{\gamma}_2 = \dot{\boldsymbol{x}}(0) \tag{31}$$

其中，在一般情况下公式 (13) 等价于

$$\boldsymbol{Diag}(\omega_1, \omega_2, \omega_3, \omega_4, \omega_5)\boldsymbol{\gamma}_2 = \boldsymbol{V}^{-1}\dot{\boldsymbol{x}}(0) \tag{32}$$

由此可以得到 $\boldsymbol{\gamma_1}, \boldsymbol{\gamma_2}$ 两个决定简正模式贡献的常数向量，由此结合公式 (22)

$$\boldsymbol{x}(t) = \boldsymbol{\gamma_1}\ cos(\omega_i t) + \boldsymbol{\gamma_2}\ sin(\omega_i t$$

即可得出各个物体的运动方程。

3 数值计算

理论分析过程已经得出了较为清晰的分析思路，并且得到了拉格朗日方程下面来讨论从数值计算角度解决问题。

3.1 计算思路

首先，根据公式 (29)

$$\ddot{\boldsymbol{x}} = \boldsymbol{A'} \cdot \boldsymbol{x} = \frac{T_n}{m}\boldsymbol{A} \cdot \boldsymbol{x} = \frac{T_n}{m}\begin{bmatrix} -2 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix}\boldsymbol{x}$$

以及前置理论我们得知，求解出矩阵 $\boldsymbol{A}$ 的本征值和本征向量稍作变化便可得到本征频率与本征矢，带入初值求解

$$\boldsymbol{V}\boldsymbol{\gamma}_1 = \boldsymbol{x}(0)$$

$$\boldsymbol{V}\ \boldsymbol{Diag}(\omega_1, \omega_2, \omega_3, \omega_4, \omega_5)\boldsymbol{\gamma}_2 = \dot{\boldsymbol{x}}(0)$$

两个方程即可得到各个振动模式的贡献，得出数值解。

因此计算思路如下：

(i) 估算出本征值的范围选取合适的求解方法

(ii) 求出本征值

(iii) 用合适方法求出本征向量

(iv) 用合适方法求出振动模式的贡献

(v) 绘制出振动图像

其中，最重要的是计算过程中的**精度控制**以及**方法选取**，两个重点会互相影响，在后续讨论中会有所说明。

3.2 计算实现

3.2.1 Gershgorin 圆盘定理判断本征频率的范围

首先，我们需要得到本征值的大致范围，（至少精确到本征值数量和数量级）以此选择大概的本征值计算方法（这一步骤是常规解析求解中或许是不必要的，但是我认为

在数值计算中对于算法的选取有影响意义)。其中 *Gershgorin* 圆盘定理是一个可行的方法。

Gershgorin 圆盘定理有下述两个定理组成

定理 2(Gershgorin 圆盘第一定理):

设 $\boldsymbol{A}$ 是 n 阶矩阵, $\boldsymbol{A}=a_{ij_{(n\times n)}}$, 则 $\boldsymbol{A}$ 的本征值在复平面上下列圆盘 (*Gershgorin* 圆盘) 中:

$$|z-a_{ii}|\leq R_i, i=1,2,\cdots,n$$

其中 R_i 为 A 的第 i 行元素去掉 a_{ii} 后的模之和, 即

$$R_i=\sum_{j\neq i}^{n}|a_{ij}|=|a_{i1}|+\cdots+|a_{i,i-1}|+|a_{i,i+1}|+|a_{in}|$$

定理 3(Gerschgorin 圆盘第二定理):

设矩阵 $A=(a_{ij})_{n\times n}$ 的 n 个 *Gershgorin* 圆盘分成若干个连通区域, 若其中一个连通区域含有 k 个 *Gershgorin* 圆盘, 则有且只有 k 个本征值落在这个连通区域内 (若两个 *Gershgorin* 圆盘重合, 需计重数; 又若本征值为重根, 也计重数).

按照**定理 2**, 利用 python 画图功能[2] 我们可以作出如下图片:

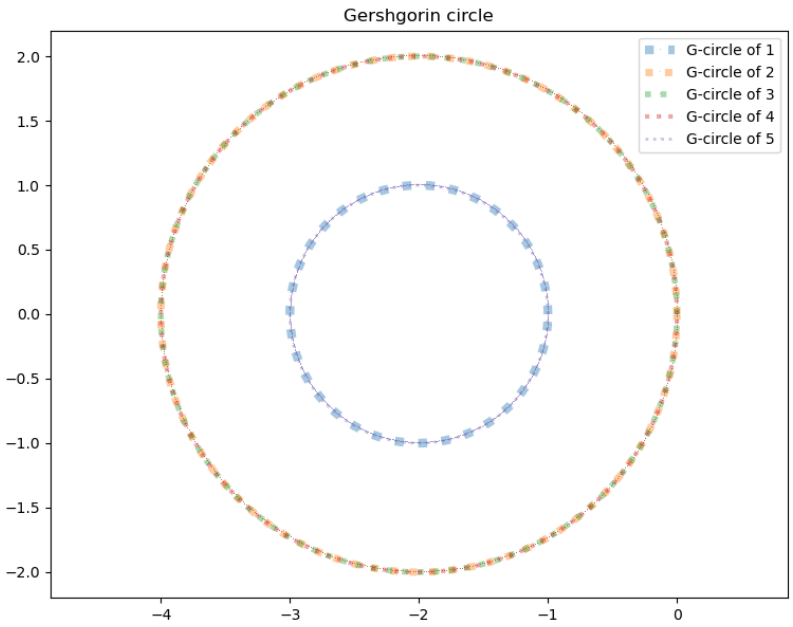


图 5: Gerschgorin 圆盘图

[2]Gershgorin 圆盘图片绘制程序实现见附录

由于上述定理，结合图 5，以及矩阵 $\boldsymbol{A}$ 严格对角占优，我们可以得到以下几点结论：

• 矩阵 $\boldsymbol{A}$ 为可逆矩阵

• 矩阵 $\boldsymbol{A}$ 本征值一定都为负数

• 矩阵 $\boldsymbol{A}$ 所有本征值一定都介于 [-4,0] 之间

由于本题矩阵的对称性极强，可以证明 [1] **以上结论对于自由度更大的时依然成立**。

不难看出本征值的模应该不小于 0.1, 因此所有本征值都与矩阵各非零元数量级大致相同，不用过分担心由于算法造成的误差，故而可以适用绝大多数算法（不过原点平移法或许会由于本征值过于接近而出现问题，故本题并没有尝试）。

（本课题中矩阵过于简单，Gershgorin 圆盘法作用未能完全显现）

3.2.2 *Jacobi* 方法求解本征频率与本征矢

理论分析得到，本征频率的平方为本征值，故而本步骤目的变为本征值与本征向量，对矩阵进行对角化可以同时得到上述预期结果。考虑到矩阵可逆以及为实对称矩阵，本次采用 *Jacobi* 方法, 用 *Givens* 旋转矩阵进行正交对角化。

从结果出发，只要可以成功得到

$$\boldsymbol{A} = \boldsymbol{P}^T \boldsymbol{B} \boldsymbol{P}$$

这个形式，其中 $\boldsymbol{B}$ 为对角矩阵，$\boldsymbol{P}$ 为正交矩阵，我们就可以得到矩阵 $\boldsymbol{A}$ 的本征值与对应本征向量，其中本征值为 $\boldsymbol{B}$ 的对角元，本征向量为 $\boldsymbol{P}$ 矩阵对应的列向量。

因此，我们需要找到合适的矩阵 $\boldsymbol{P}$ 以及对应矩阵 $\boldsymbol{B}$ 使其满足上述条件，成功将 $\boldsymbol{A}$ 对角化为 $\boldsymbol{B}$ 。对于本题来讲，我们可以认为非对角元素之和小于 10^{-15} 即为成功对角化。

•*Jacobi* 方法

我们之前说到我们需要把 $\boldsymbol{A}$ 对角化，他的目的是通过一次次的合适的正交相似变换进行形如

$$\boldsymbol{A}_{i+1} = \boldsymbol{Q}^T \boldsymbol{A}_i \boldsymbol{Q}$$

的迭代，将 A 转化为

$$\boldsymbol{Q}_n^T \dots \boldsymbol{Q}_2^T \boldsymbol{Q}_1^T \boldsymbol{A}_n \boldsymbol{Q}_1 \boldsymbol{Q}_2 \dots \boldsymbol{Q}_n$$

的形式，显然的，如果矩阵 $\boldsymbol{Q}$ 选取合适可以达到对角化的作用。一般 *Jacobi* 都和误差控制结合使用，达到需要精度之后跳出迭代即可。

•*Givens* 旋转矩阵

Givens 旋转矩阵表示为如下形式的矩阵

$$\boldsymbol{G}(i,j,\theta)=\begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

这里的 $c=\cos(\theta)$ 和 $s=\sin(\theta)$ 出现在第 i 行和第 j 行与第 i 列和第 j 列的交叉点上。就是说, *Givens* 旋转矩阵的所有非零元定义如下: :

$$\begin{cases} g_{kk}=1 \quad (for \quad k\neq i,j) \\ g_{ii}=c \\ g_{jj}=c \\ g_{ij}=s \\ g_{ji}=-s \end{cases}$$

$\boldsymbol{G}(i,j,\theta)\boldsymbol{x}$ 表示向量 $\boldsymbol{x}$ 在 (i,j) 平面中的逆时针旋转 θ 弧度。Givens 旋转在数值线性代数中主要的用途是在向量或矩阵中介入零。这个本步的目的不谋而合。同时他还有便于并行计算和对于稀疏矩阵支持较好两个优点, 对于本题的高自由度情况有较大用处。

结合上述介绍，我们不难想到用 *Givens* 矩阵来进行 *Jacobi* 方法进行对角化，显然的旋转矩阵的正交性完美满足要求，接下来的问题就只需要生成矩阵并通过迭代达成

$$\boldsymbol{A}_n=\boldsymbol{G}_n^T\ldots\boldsymbol{G}_2^T\boldsymbol{G}_1^T\boldsymbol{A}\boldsymbol{G}_1\boldsymbol{G}_2\ldots\boldsymbol{G}_n$$

从而得到正交化之后的 $\boldsymbol{A}_n$. 而这一步通过对于 A 矩阵进行不断的左乘右乘即可实现。

同时，由于 $Givens$ 矩阵十分稀疏，我们不妨将其展开得到如下公式

$$\begin{cases} b_{ip} = b_{pi} = a_{pi}\cos\theta - a_{qi}\sin\theta, & i \neq p,q \\ b_{iq} = b_{qi} = a_{pi}\sin\theta + a_{qi}\cos\theta, & i \neq p,q \\ b_{pp} = a_{pp}\cos^2\theta + a_{qq}\sin^2\theta - a_{pq}\sin 2\theta \\ b_{qq} = a_{pp}\sin^2\theta + a_{qq}\cos^2\theta + a_{pq}\sin 2\theta \\ b_{pq} = b_{qp} = a_{pq}\cos 2\theta + \frac{a_{pp}-a_{qq}}{2}\sin 2\theta \end{cases}$$

通过这项变换，进行一次循环即可得到

$$\boldsymbol{A}_{(i)} = \boldsymbol{G}_{(i)}^T \boldsymbol{A}\boldsymbol{G}_{(i)}$$

的结果。不难注意到时间复杂度从 $O(n^3)$ 变成了 $O(n)$，在矩阵维数较大的时候可以极大加快运算。

上式中的第二个等式以及模态矩阵和本征向量的关系可以得出一个对角矩阵，本步骤用到了 $Jacobi$，矩阵乘法，矩阵转置三个函数[3]，上述操作可以在 $Jacobi$ 函数中把两次矩阵乘法加上一次矩阵转置化为一体，达到加速效果。

最终得到的对角矩阵的对角元便是本征频率的绝对值，右侧矩阵的列向量便是本征矢。

运行环境

$Ubuntu9.3.0-17\ Ubuntu1\ 20.04LTS$

$Python3.8.8\quad Anaconda4.10.3\quad AMD-4800u$

```
import numpy as np
```

```
def Jacobi(A,eps,Eigv,detail):
    for ii in range(1,100):
        simga = 0
        apq = 0
        tan2 = 0
        for i in range(len(A)):
            for j in range(len(A)):
                if (i!=j):
                    simga +=A[i][j]              #step1
                    if (abs(A[i][j])>abs(apq)):
```

[3]完整程序实现见附录

```
                apq = A[i][j]            #step2
                (p,q) = (i,j)
if ((A[q][q]-A[p][p])!=0):
    tan2 = 2*A[p][q]/(A[q][q]-A[p][p])
    theta = np.math.atan(tan2) / 2
else:
    theta = np.math.pi/4
cos = np.math.cos(theta)
sin = np.math.sin(theta)
tan = np.math.tan(theta)
Q=np.eye(len(A))
Q[p][p]=cos
Q[q][q]=cos
Q[p][q]=sin
Q[q][p]=-sin
app = float(A[p][p])
aqq = float(A[q][q])
apq = float(A[p][q])
for j in range(len(A)):
    if (j!=q)and(j!=p):
        apj = float(A[p][j])
        aqj = float(A[q][j])
        A[p][j] = cos*apj - sin*aqj
        A[q][j] = sin * apj + cos * aqj
        A[j][p] = A[p][j]
        A[j][q] = A[q][j]
E = Product(Eigv,Q)
for i in range(len(A)):
    for j in range(len(A)):
        Eigv[i][j] = E[i][j]
#print(np.round(A,10))
if (detail):
    print("times={}\n{}".format(ii,np.round(A,4)))
    print(np.round(Eigv,4))
sum = 0
for i in range(len(A)):
    for j in range(len(A)):
        if (i != j):
            sum += abs(A[i][j])
if (sum < eps):
    break
```

```
    for i in range(len(A)):
        value[0][i]=(A[i][i])
    for i in range(len(A)):
        for j in range(len(A)):
            B[i][j] = A[i][j]
    #print(np.round(Eigv,3))

    return (value)
```

```
def Product(A, B):
    C = np.zeros([len(A),len(B[0])])
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                C[i][j] += A[i][k]*B[k][j]
    return C
```

```
def Trans(A):
    C=np.zeros((len(A[0]),len(A)))
    for i in range (len(A[0])):
        for k in range(len(A)):
            C[i][k]=A[k][i]
    return C
```

3.2.3 初值讨论

经过上述处理后，我们可以得到

$$\boldsymbol{A}_{(n)} = \boldsymbol{G}_n^T \dots \boldsymbol{G}_2^T \boldsymbol{G}_1^T \boldsymbol{A} \boldsymbol{G}_1 \boldsymbol{G}_2 \dots \boldsymbol{G}_n$$

的对角化形式。其中 $\boldsymbol{A}_n$ 为对角矩阵，对角值为本征值。左右都为本征向量组成的矩阵。

代入之前得到的解

$$\boldsymbol{x}(t) = \boldsymbol{\gamma_1}\ cos(\omega_i t) + \boldsymbol{\gamma_2}\ sin(\omega_i t)$$

带入之前的拉格朗日方程

$$\sum_j m_{ij}\ddot{x}_j + \sum_k k_{ij}x_j = 0$$

并且结合

$$\boldsymbol{V}\boldsymbol{\gamma}_1 = \boldsymbol{x}(0)$$

$$\boldsymbol{V}\ \boldsymbol{Diag}(\omega_1,\omega_2,\omega_3,\omega_4,\omega_5)\boldsymbol{\gamma_2}=\dot{\boldsymbol{x}}(0)$$

给出 $2s=10$ 个初值，我们就可以得到 $\boldsymbol{\gamma_1},\boldsymbol{\gamma_2}$ 的解，从而得到振动方程。求解过程在本部分将采用直接用 $\boldsymbol{V}^T=\boldsymbol{V}^{-}1$ 来求解 γ_1,γ_2. [4]

3.2.4 运动状态与图像给出

带入初值得到最终的运动方程，我们便可以进行运动图像的绘制，一般的，我们可以分别得到五个不同的运动图像，绘制在同一张图像我们可以得到并比较其运动状态。注意选取合适的且确定的横纵坐标范围，以得到较为稳定的图像便于比较 [5]。

3.3 计算结果

根据上述方法计算矩阵的本征值与本征向量我们可以得到如下结果

表 1: 5 自由度系统本征值

本征值				
-3	-1	-3.73205081	-0.26794919	-2

表 2: 5 自由度系统本征矢

本征向量				
0.5	0.5	0.28867513	0.28867513	0.57735027
-0.5	0.5	-0.5	0.5	0.
-0.	0.	0.57735027	0.57735027	-0.57735027
0.5	-0.5	-0.5	0.5	0.
-0.5	-0.5	0.28867513	0.28867513	0.57735027

不妨取初值 $\boldsymbol{x_0}=[1,0,0,0,0]^T,\dot{\boldsymbol{x}}_{\boldsymbol{0}}=[0,0,0,0,0]^T$（与微振动系统来源的默认值一样），我们可以得到 $\boldsymbol{\gamma}_1$，$\omega\boldsymbol{\gamma}_2$

表 3: 5 自由度系统在初值 1 下 $\gamma_1,\omega\gamma_2$

γ_1	0.5	0.5	0.28867513	0.28867513	0.57735027
$\omega\gamma_2$	0	0	0	0	0

代入 $\boldsymbol{x}(t)=\boldsymbol{\gamma_1}\ cos(\omega_i t)+\boldsymbol{\gamma_2}\ sin(\omega_i t)$ 可以作出如下图像。

[4]程序实现见附录

[5]程序实现见附录

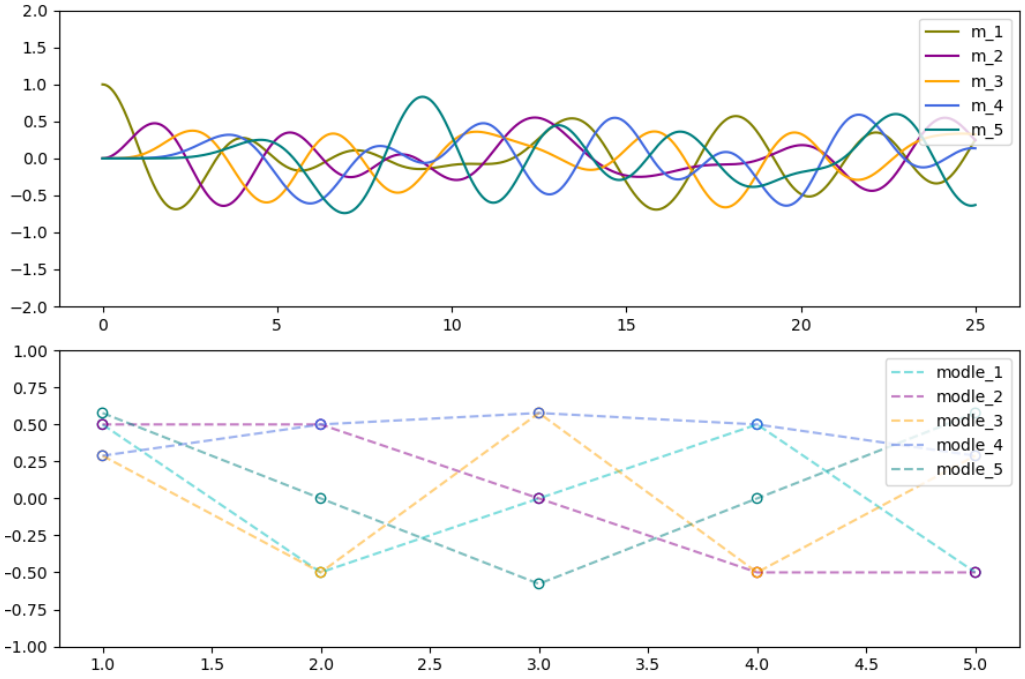


图 6: 在初值 1 状态下的振动图以及本征向量图

至此，我们已经完美完成了网站上项目的复现，并且进行了速度初值的计算以及参与结果。

3.4 结果与讨论

根据以上计算，我们可以得到五自由度的微振动系统的数值运动解。

3.4.1 结果陈列

初值 2

- $\boldsymbol{x_0} = [1,0,0,0,1]^T, \dot{\boldsymbol{x}}_{\boldsymbol{0}} = [1,0,1,0,1]^T$

表 4: 5 自由度系统在初值 2 下 $\gamma_1, \omega\gamma_2$

γ_1	0	0	0.57735	0.57735	1.1547
$\omega\gamma_2$	0	0	1.1547	1.1547	0.57735

初值 3

- $\boldsymbol{x_0} = [-1,1,0,-1,1]^T, \dot{\boldsymbol{x}}_{\boldsymbol{0}} = [1,-1,0,1,-1]^T$

表 5: 5 自由度系统在初值 3 下 $\gamma_1, \omega\gamma_2$

γ_1	2	0	0	0	0
$\omega\gamma_2$	2	0	0	0	0

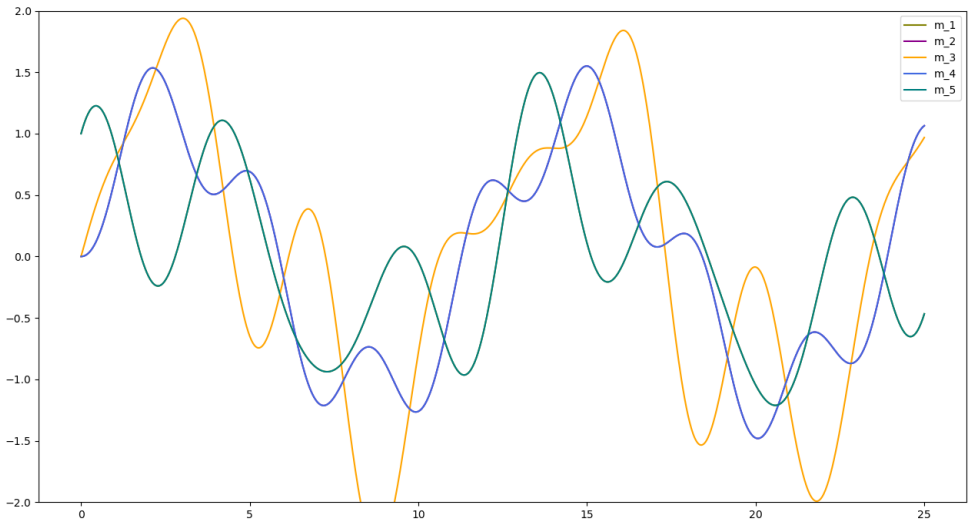


图 7: 在初值 2 状态下的振动图

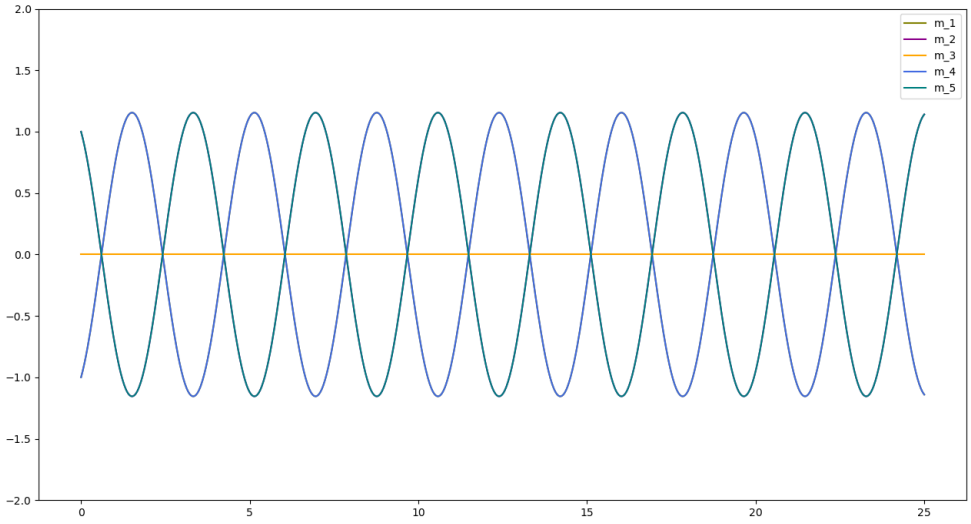


图 8: 在初值 3 状态下的振动图

3.4.2 对称性讨论

之前的初值选取是有技巧的，我们可以看出系统振动有极高的对称性，这是因为我们选取的初值与模式形状的对称性匹配。例如，初值 $3x_0 = [-1, 1, 0, -1, 1]^T, \dot{\boldsymbol{x}}_{\mathbf{0}} = [1, -1, 0, 1, -1]^T$ 的对称性与且仅与 $modle_1$ 完美匹配，故其有且仅有唯一激发的 $\omega =$

1.73205081 . 类似细节不多赘述，由此我们得到激发特定频率的方式。

3.4.3 question：如何激活特定频率

由上，我们知道了对称性的初值可以激发有相同对称性的频率，一般的，如果初值的对称性是振动 $modle$ 对称性的耦合我们也可以得到具有耦合关系的频率。

因此我们有了激活特定频率的一般性思想：可以通过初值的选取来激活特定频率。

3.5 多自由度分析

3.5.1 多自由度情况矩阵生成

显然的，根据上述理论得到该矩阵是解题的首要关键。

矩阵形如

$$\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}$$

用双重循环即可得到该矩阵，具体计算方式不再赘述。

3.5.2 多自由度本征求解

函数计算思路大体不变，不过之前时间复杂度的考量有了存在意义。

不妨考虑 $s = 1000$ 的情况，即矩阵为 1000 维。我们可以得到 1000 个本征值。按照自由度展开做图可得

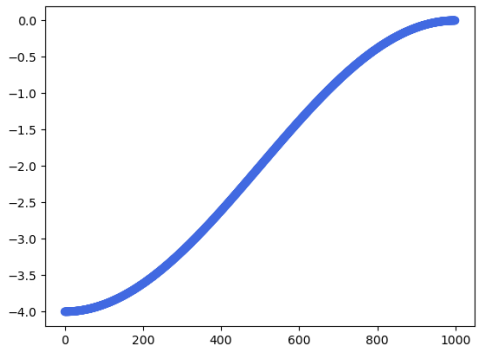


图 9: 本征值-自由度图像

不难想到，在自由度足够的情况下，本征值将会遍历 *Gerschgorin Circle* 的实数区，即所有可能的本征值位置。

按照同样的方法，我们可以得到本征解并且绘出振动图像节约时间，取 $s = 100$, 初值为初值 1，我们可以的到如下图像

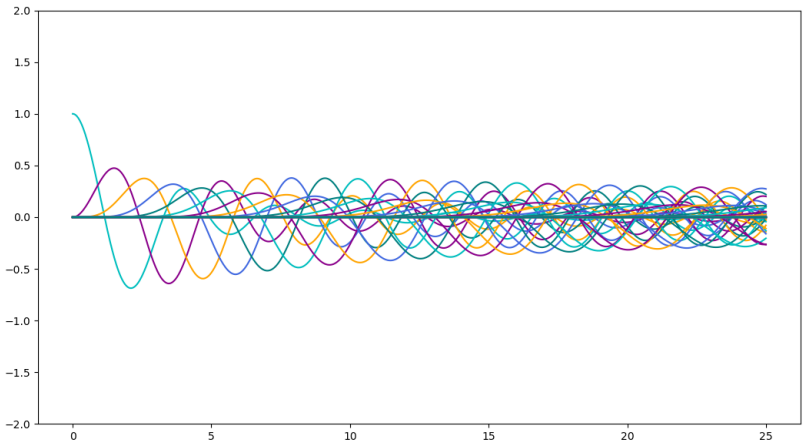


图 10: 振动-时间图像

或许这张图过于杂乱不清晰，通过下面这张图可以看出可以看出波传播的趋势——我们得到了标准的驻波图像（美观起见 s 取为 2000，算法优化不够，调用了 *numpy* 的函数）.

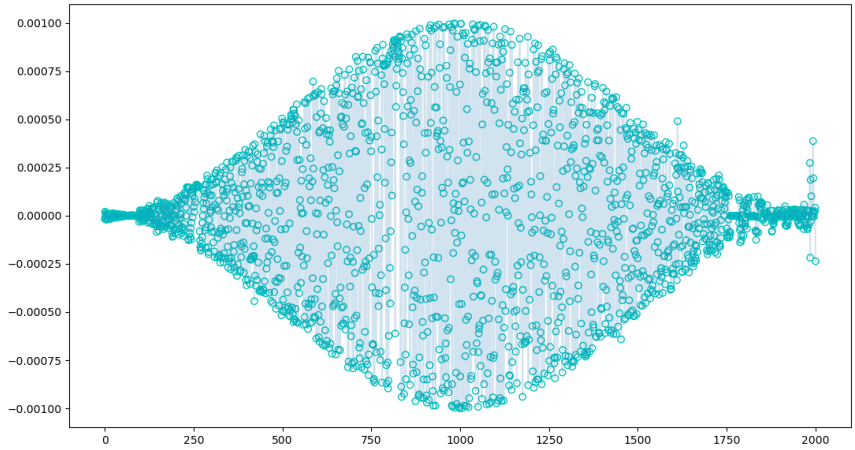


图 11: t=2000 时的波形图

其物理意义将于后续部分再行探究。

4 问题分析与探讨

在进行多自由度微振动的本正频率问题的数值计算时，我们主要遇到了如下问题

· 本征向量的求出

· 精度问题

· 初值计算

· 时间复杂度过高

· 物理意义不明确

下面对各个问题的进行讨论。

4.1 问题概述与解决

4.1.1 本征向量的求出

在求出本征向量的过程中，最显而易见的便是直接用 $Givens$ 矩阵的连乘，不过我们在讨论过程中肯定会有疑问：用迭代法求解或者矩阵分解求解可以吗?

矩阵分解法显然是可以的，不过普遍的分解法时间复杂度过高，而对称 QR 分解虽然时间复杂度降低到了 $O_{(n)}$[1]，可是用 QR 求解方程需要循环次数过高，故而也不予以考虑。

至于迭代方法，我们常用的雅可比法，高斯-赛德尔迭代，以及松弛迭代法都有一个通用的收敛条件：谱半径大于 1, 目前暂且没有查到或者自己想到合适的优化算法，因此没有考虑。

故而，目前采用的还是连乘 $Givens$ 矩阵，时间还在可接受范围内。同时由于 Givens 矩阵十分稀疏，因此如果展开用元素写出还有更大优化空间。最终可以得到一个时间复杂度为 $O_{(n)}$ 的算法，不过在 python 上的实现仍距有距 $numpy$ 有较大距离。

4.1.2 精度问题

本次课题主要用的是 $Jacobi$ 迭代求解本征向量与本征值，因此我们需要考虑何时退出迭代。

同时在求 $sin\theta$ 等三角函数时，我使用了 python 中 numpy 库里面的三角函数包，经尝试可以有效增加精度。

显然的，迭代次数会随着矩阵维数增加而增加，我们不妨设定非对角元绝对值之和 $< 10^{-15}$ 即为成功对角化，最终误差期望应该在 10^{-15} 左右。

4.1.3 初值计算

初值计算等价与解线性方程组，线性方程组的解法多样回归到了类似之 4.1.1 的问题。同样的，控制谱半和优化算法都是较难的工作同时在 $s < 100$ 时，整体程序运行时间均不超过 20s，因此没有作出计算方法的创新。

4.1.4 时间复杂度过高

时间复杂度过高主要是由于循环过多。例如，矩阵乘法的三重循环，寻找最佳旋转角的双重循环，矩阵赋值的双重循环等 ..... （只考虑考虑双重以上循环）

对于矩阵乘法，我们应该在可以简化的地方（例如 $Givens$ 矩阵的矩阵运算展开，对元素求解。同样的寻找最佳旋转角也应该利用的对称性进行对角线搜索，矩阵赋值多用深度复制 (copy.deepcopy()) ，等，对于时间复杂度已有较好优化，当然还有更多想法由于能力不足未能实现。

4.1.5 物理意义不明确

最开始我不是很了解弹力大小不变的微振动物理意义，直到接触到弦震动方程想到了增加自由度来逼近。

此外，查阅文献发现更符合的情况是边界条件为质量无穷大原子的一维原子链，下一部分将会进行解析求解对比。

4.2 解析求解对比

固定边界条件下的一维原子链的集体振动和本题描述的是十分相似，故而进行解析讨论。

一纬原子链是固体物理中的常用模型，其考虑了原子之间的作用里考虑在有初值条件下的运动情况，本题的实质是固定边界条件下的一维单原子链，下面进行讨论。

在简谐近似下，一维原子链可简化为如课题探究的模型，细绳的倔强系数为 k（根据题目，k=1），振子的质量为 m，体系的总振子数目为 N（自由度）。易见，[6] 体系的拉氏量可写成：

$$\mathscr{L} = \sum_i \frac{1}{2} m \dot{x}_l - \frac{1}{2} k \left[x_1^2 + (x_2 - x_1)^2 + \cdots + (x_N - x_{N-1})^2 + x_N^2 \right] \tag{33}$$

按照之前的思路经过数学推导[6]，我们可以得到

$$\left[\sin\frac{n\pi}{N+1},\sin\frac{2n\pi}{N+1},\sin\frac{3n\pi}{N+1},\ldots\ldots,\sin\frac{Nn\pi}{N+1}\right]^T$$

为体系的简正模, 其所对应的简正频率为

$$w_n^2=\frac{2k}{m}\left(1-\cos\theta_n\right)\left(\theta_n=\frac{n}{N+1}\pi,n=1,2,3\ldots N\right)$$

. (本课题中 k 取 $k=1$)

代入 $N=s=5$ 的情况我们可以得到

表 6: 5 自由度系统解析本征值

ω_1^2	ω_2^2	ω_3^2	ω_4^2	ω_5^2
$2(1-2cos\frac{1}{6}\pi)$	$2(1-2cos\frac{2}{6}\pi)$	$2(1-2cos\frac{3}{6}\pi)$	$2(1-2cos\frac{4}{6}\pi)$	$2(1-2cos\frac{5}{6}\pi)$
0.26795	1	2	3	3.73205

与表 1　五自由度系统本征值十分匹配

本征值				
-3	-1	-3.73205081	-0.26794919	-2

对于 n 自由度，绘制出本正频率-自由度图像, 即 $2\left(1-\cos\frac{n}{N+1}\pi\right)$–$n$ 图像

不妨 n 取 1000. 我们可以得到：

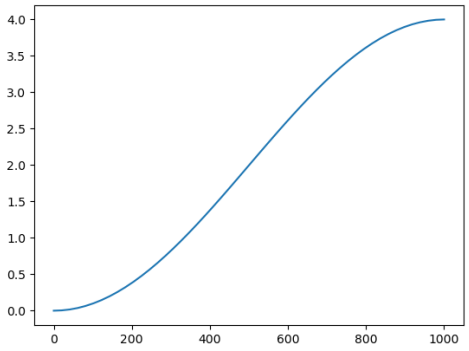
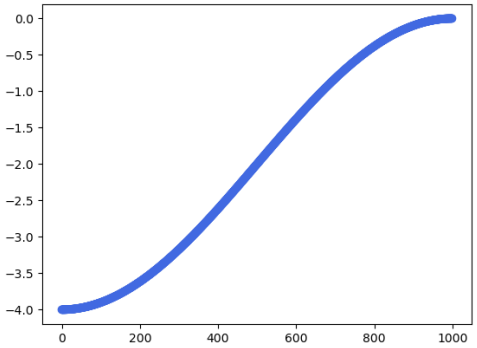


图 12: 本征值-自由度解析图像

[6]具体过程见附录

与图 9



完美匹配。

至此，我们基本可以认为本次课题计算结果正确。

5 总结

本次课题首先以五自由度的微振动系统的问题的数值探究入手，复现出网站的进展，接下来进行理论推广，得到了多自由度的微振动系统的本征问题的数值计算。期间进行了时间复杂度的多次优化和算法改进，最终得到的计算结果和理论值匹配，并认为本课题程序计算结果较为准确，可以进行固定边界条件下一维原子链的集体振动行为数值模拟。

参考文献

[1] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery. *Numerical recipes the art of scientific computing* [M]

[2] Roberto Tamassia , Michael H. Goldwasser , Michael T. Goodrich *Data Structures and Algorithms in Python*[M].

[3] 彭芳麟. 计算物理基础 [M]. 高等教育出版社

[4] Koonin. *Computational Physics*[M]

[5] Wikipedia.*Iterative method*

[6] zhihu. 固定边界条件与周期边界条件下一维原子链的集体振动行为

[7] 刘川. 理论力学 [M]. 北京大学出版社

[8] 朗道. 力学 [M].

6 附录

.1 全部程序实现

.1.1 *Gerschgorin* 圆盘

运行环境

*Ubuntu*9.3.0 − 17 *Ubuntu*1 20.04*LTS*

*Python*3.8.8 *Anaconda*4.10.3 *AMD* − 4800*u*

```
import numpy as np
import matplotlib.pyplot as plt

A=np.array([[   -2  ,   1  ,   0    ,   0    ,    0],
            [    1  ,  -2  ,   1    ,   0    ,    0],
            [    0  ,   1  ,  -2    ,   1    ,    0],
            [    0  ,   0  ,   1    ,  -2    ,    1],
            [    0  ,   0  ,   0    ,   1    ,   -2]])

def getr(ii,A):
    r = 0
    for i in range(len(A)):
        if (ii != i):
            r += abs(A[ii][i])
    return r

def p_circle(r,ii,fig):
    a = A[ii][ii]
    b = 0
    theta = np.arange(0, 2*np.pi, 0.01)
    x = a + r * np.cos(theta)
    y = b + r * np.sin(theta)
    plt.plot(x, y,alpha = 0.4,linestyle=':',label = '{}'.format(i),linewidth
                                          = '{}'.format(len(A)-i+1),marker='
                                          ,')

if __name__ == "__main__":

    Gershgorin = plt.figure()
    for i in range(len(A)):
        r = getr(i,A)
        p_circle(r,i,Gershgorin)
    plt.axis('equal')
```

```
    plt.legend(['G-circle of 1',"G-circle of 2","G-circle of 3","G-circle of
                                        4","G-circle of 5"] ,loc='upper
                                        right' )
    plt.title('Gershgorin circle')
    plt.show()
```

.1.2 5 自由度 + 画图 + 模式图

运行环境

$Ubuntu9.3.0-17\ Ubuntu1\ 20.04LTS$

$Python3.8.8\quad Anaconda4.10.3\quad AMD-4800u$

```
import numpy as np
import matplotlib.pyplot as plt

col = ['olive','darkmagenta','orangE','royalblue','teal']

A=np.array([[   -2  ,   1   ,   0    ,   0    ,    0],
            [    1  ,  -2   ,   1    ,   0    ,    0],
            [    0  ,   1   ,  -2    ,   1    ,    0],
            [    0  ,   0   ,   1    ,  -2    ,    1],
            [    0  ,   0   ,   0    ,   1    ,   -2]])

def Jacobi(A,eps,Eigv,detail):
    for ii in range(1,100):
        simga = 0
        apq = 0
        tan2 = 0
        for i in range(len(A)):
            for j in range(len(A)):
                if (i!=j):
                    simga +=A[i][j]              #step1
                    if (abs(A[i][j])>abs(apq)):
                        apq = A[i][j]            #step2
                        (p,q) = (i,j)
        if ((A[q][q]-A[p][p])!=0):
            tan2 = 2*A[p][q]/(A[q][q]-A[p][p])
            theta = np.math.atan(tan2) / 2
        else:
            theta = np.math.pi/4
```

```
        cos = np.math.cos(theta)
        sin = np.math.sin(theta)
        tan = np.math.tan(theta)
        Q=np.eye(len(A))
        Q[p][p]=cos
        Q[q][q]=cos
        Q[p][q]=sin
        Q[q][p]=-sin
        app = float(A[p][p])
        aqq = float(A[q][q])
        apq = float(A[p][q])
        for j in range(len(A)):
            if (j!=q)and(j!=p):
                apj = float(A[p][j])
                aqj = float(A[q][j])
                A[p][j] = cos*apj - sin*aqj
                A[q][j] = sin * apj + cos * aqj
                A[j][p] = A[p][j]
                A[j][q] = A[q][j]
        E = Product(Eigv,Q)
        for i in range(len(A)):
            for j in range(len(A)):
                Eigv[i][j] = E[i][j]
        #print(np.round(A,10))
        if (detail):
            print("times={}\n{}".format(ii,np.round(A,4)))
            print(np.round(Eigv,4))
        sum = 0
        for i in range(len(A)):
            for j in range(len(A)):
                if (i != j):
                    sum += abs(A[i][j])
        if (sum < eps):
            break
    for i in range(len(A)):
        value[0][i]=(A[i][i])
    for i in range(len(A)):
        for j in range(len(A)):
            B[i][j] = A[i][j]
    #print(np.round(Eigv,3))
```

```
    return (value)

def Product(A, B):
    C = np.zeros([len(A),len(B[0])])
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                C[i][j] += A[i][k]*B[k][j]
    return C

def Trans(A):
    C=np.zeros((len(A[0]),len(A)))
    for i in range (len(A[0])):
        for k in range(len(A)):
            C[i][k]=A[k][i]
    return C

if __name__ == "__main__":
    Eigv=np.eye(5)
    B=np.eye(len(A))
    G=np.eye(len(A))
    yy = np.zeros(5)
    xx = np.zeros(5)
    value = np.zeros([1,5])
    Jacobi(A.copy(),1.e-15,Eigv,False)
    print("\t\t\t\t\t\tEigenvalue")
    print(np.round(value,10))
    print("\t\t\t\t\t\tEigenvector")
    print(np.round(Eigv,10))
    omega = np.sqrt(-value)[0].copy()

    print("\n\n\n")

    dotx0 = np.array([[1,-1, 0, 1,-1]])
    x0 = np.array([[-1, 1, 0, -1, 1]])
    x = Product(x0,Eigv.copy())
    print("gamma1",x)
    gamma1 = x[0]
    x = Product(dotx0,Eigv.copy())
    print("wgamma2",x)
    wgamma2 = (x[0])
```

```
plt.figure(figsize=(15, 15),dpi=100)
plt.subplot(2,  1,  1)
x = np.linspace(0,25, 1000)
for i in range(len(Eigv)):
    y = 0

    for j in range(len(Eigv)):
        y += (Eigv[i][j]*gamma1[j]*np.cos(omega[j]*x)+
        Eigv[i][j]*wgamma2[j]/omega[j]*np.sin(omega[j]*x))

    plt.plot(x, y,color = col[i%5],label = 'm_{}'.format(i))
    # plt.scatter(x,y,color='r',marker='x')

plt.legend(['m_1','m_2','m_3','m_4','m_5'] ,loc='upper right' )
plt.ylim(-2,2)
plt.subplot(2,  1,  2)
xxx=[]
yyy=[]
col = ['c','darkmagenta','orangE','royalblue','teal']
for j in range(len(A)):
    xxx.clear()
    yyy.clear()
    for i in range(len(A)):
        xxx.append(i+1)
        yyy.append(Eigv[i][j])
    plt.plot(xxx, yyy,color = col[j%5],alpha=0.5,linestyle='--',label = '
                                        modle_{}'.format(j))
    plt.scatter(xxx, yyy,marker='o',color="",edgecolors = col[j%5])
    plt.ylim(-1,1)
    plt.legend(['modle_1','modle_2','modle_3','modle_4','modle_5'] ,loc='
                                        upper right' )
plt.savefig('plot1.jpg')
plt.show()
print(omega)
```

.1.3 N 自由度

运行环境

$Ubuntu9.3.0-17\ Ubuntu1\ 20.04LTS$

$Python3.8.8\quad Anaconda4.10.3\quad AMD-4800u$

```
import numpy as np
import matplotlib.pyplot as plt

col = ['c','darkmagenta','orangE','royalblue','teal']

A=np.eye(8)
for i in range(len(A)):
    A[i][i]=-2
    if (0<i):
        A[i][i-1]=1
    if (4>i):
        A[i][i+1]=1


def Jacobi(A,eps,Eigv,detail):
    for ii in range(1,100):
        simga = 0
        apq = 0
        tan2 = 0
        for i in range(len(A)):
            for j in range(len(A)):
                if (i!=j):
                    simga +=A[i][j]              #step1
                    if (abs(A[i][j])>abs(apq)):
                        apq = A[i][j]            #step2
                        (p,q) = (i,j)
        if ((A[q][q]-A[p][p])!=0):
            tan2 = 2*A[p][q]/(A[q][q]-A[p][p])
            theta = np.math.atan(tan2) / 2
        else:
            theta = np.math.pi/4
        cos = np.math.cos(theta)
        sin = np.math.sin(theta)
        tan = np.math.tan(theta)
        Q=np.eye(len(A))
        Q[p][p]=cos
        Q[q][q]=cos
        Q[p][q]=sin
        Q[q][p]=-sin
        A=Product(Product(Trans(Q),A),Q)
```

```
        # for i in range(len(A)):
        #     if ((i !=p)and(i != q)):
        #         A[i][p] = A[p][i]*cos - A[q][i]*sin
        #         A[i][q] = A[p][i]*sin + A[q][i]*cos
        #         A[p][i] = A[i][p]
        #         A[q][i] = A[i][q]
        # app = A[p][p]
        # aqq = A[q][q]
        # A[p][p] = sin*sin*app + cos*cos*aqq + 2*sin*cos*app
        # A[p][p] = sin*sin*aqq + cos*cos*app - 2*sin*cos*app
        # A[q][q] = A[q][q] + tan*A[p][q]
        E = Product(Eigv,Q)
        for i in range(len(A)):
            for j in range(len(A)):
                Eigv[i][j] = E[i][j]
        #print(np.round(A,10))
        if (detail):
            print("times={}\n{}".format(ii,np.round(A,4)))
            print(np.round(Eigv,4))
        sum = 0
        for i in range(len(A)):
            for j in range(len(A)):
                if (i != j):
                    sum += abs(A[i][j])
        if (sum < eps):
            break
    for i in range(len(A)):
        value[0][i]=(A[i][i])
    for i in range(len(A)):
        for j in range(len(A)):
            B[i][j] = A[i][j]
    #print(np.round(Eigv,3))

    return (value)

def Product(A, B):
    C = np.zeros([len(A),len(B[0])])
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                C[i][j] += A[i][k]*B[k][j]
```

```
    return C

def Trans(A):
    C=np.zeros((len(A[0]),len(A)))
    for i in range (len(A[0])):
        for k in range(len(A)):
            C[i][k]=A[k][i]
    return C

if __name__ == "__main__":
    Eigv=np.eye(len(A))
    B=np.eye(len(A))
    G=np.eye(len(A))
    yy = np.zeros(5)
    xx = np.zeros(5)
    value = np.zeros([1,len(A)])
    Jacobi(A,1.e-10,Eigv,False)
    print("\t\t\t\t\t\t\tEigenvalue")
    print(np.round(value,10))
    print("\t\t\t\t\t\t\tEigenvector")
    print(np.round(Eigv,6))
    omega = np.sqrt(-value)[0].copy()

    print("\n\n\n")

    dotx0 = np.zeros([1,len(A)])
    x0 = np.zeros([1,len(A)])
    x0[0][0] = 1
    x = Product(Trans(Eigv),x0)
    x = (Trans(x))[0].copy()
    print("gamma1",x)
    gamma1 = x
    x = Product(Trans(Eigv),dotx0)
    x = (Trans(x))[0]
    print("wgamma2",x)
    wgamma2 = (x)
```

.1.4 N 自由度 + 画图 (用 numpy 本征值函数)

运行环境

$Ubuntu9.3.0-17\ Ubuntu1\ 20.04LTS$

$Python3.8.8\quad Anaconda4.10.3\quad AMD-4800u$

```
import numpy as np
import matplotlib.pyplot as plt
N = 10
col = ['c','darkmagenta','orangE','royalblue','teal','r']

A=np.eye(N)

for i in range(N):
    A[i][i]=-2
    if (0<i):
        A[i][i-1]=1
    if (N-1>i):
        A[i][i+1]=1

print(A)

def Jacobi(A,eps,Eigv,detail):
    for ii in range(1,1000):
        simga = 0
        apq = 0
        tan2 = 0
        N=len(A)
        for i in range(N):
            for j in range(i+1,N):
                    simga +=A[i][j]              #step1
                    if (abs(A[i][j])>abs(apq)):
                        apq = A[i][j]            #step2
                        (p,q) = (i,j)
        if ((A[q][q]-A[p][p])!=0):
            tan2 = 2*A[p][q]/(A[q][q]-A[p][p])
            theta = np.math.atan(tan2) / 2
        else:
            theta = np.math.pi/4
        cos = np.math.cos(theta)
        sin = np.math.sin(theta)
        cos2 = np.math.cos(2*theta)
        sin2 = np.math.sin(2*theta)
        Q=np.eye(N)
        Q[p][p]=cos
        Q[q][q]=cos
        Q[p][q]=sin
```

```
        Q[q][p]=-sin
        # A=Product(Product(Trans(Q),A),Q)
        app = float(A[p][p])
        aqq = float(A[q][q])
        apq = float(A[p][q])
        for j in range(len(A)):
            if (j!=q)and(j!=p):
                apj = float(A[p][j])
                aqj = float(A[q][j])
                A[p][j] = cos*apj - sin*aqj
                A[q][j] = sin * apj + cos * aqj
                A[j][p] = A[p][j]
                A[j][q] = A[q][j]

        A[p][p] = aqq*sin*sin + app*cos*cos -apq *sin2
        A[q][q] = app*sin*sin + aqq*cos*cos +apq *sin2
        A[p][q] = float(apq)*cos2 + (app-aqq)/2*(sin2)
        A[q][p] = A[p][q]

        #E = copy.copy(Product(Eigv,Q))
        for i in range(N):
            aip = Eigv[i][p]
            aiq = Eigv[i][q]
            Eigv[i][p] = cos*aip-sin*aiq
            Eigv[i][q]= sin*aip+cos*aiq
        #E=Eigv@Q
        # for i in range(N):
        #     for j in range(N):
        #         Eigv[i][j] = E[i][j]
        #print(np.round(A,10))
        if (detail):
            print("times={}\n{}".format(ii,np.round(A,4)))
            print(np.round(Eigv,4))
        sum = 0
        for i in range(N):
            for j in range(N):
                if (i != j):
                    sum += abs(A[i][j])
        if (sum < eps):
            break
    for i in range(N):
```

```
            value[0][i]=(A[i][i])
        for i in range(N):
            for j in range(N):
                B[i][j] = A[i][j]
        #print(np.round(Eigv,3))

        return (value)

def Product(A, B):
        C = np.zeros([len(A),len(B[0])])
        for i in range(len(A)):
            for j in range(len(B[0])):
                for k in range(len(B)):
                    C[i][j] += A[i][k]*B[k][j]
        return C

def Trans(A):
        C=np.zeros((len(A[0]),len(A)))
        for i in range (len(A[0])):
            for k in range(len(A)):
                C[i][k]=A[k][i]
        return C

if __name__ == "__main__":
        Eigv=np.eye(N)
        B=np.eye(len(A))
        G=np.eye(len(A))
        yy = np.zeros(N)
        xx = np.zeros(N)
        value = np.zeros([1,len(A)])
        time1=time.time()
        Jacobi(A.copy(),1.e-10,Eigv,False)
        time2=time.time()
        print("\t\t\t\t\t\t\tEigenvalue")
        print(np.round(value,10))
        print("\t\t\t\t\t\t\tEigenvector")
        print(np.round(Eigv,3))
        omega = np.sqrt(-value.copy())[0].copy()

        print("\n\n\n")
```

```
    dotx0 = np.zeros([1,N])
    x0 = np.zeros([1,N])
    x0[0][0] = 1
    x = Product(Trans(Eigv),x0)
    x = (Trans(x))[0].copy()
    print("gamma1",x)
    gamma1 = x
    x = Product(Trans(Eigv),dotx0)
    x = (Trans(x))[0]
    print("wgamma2",x)
    wgamma2 = (x)
    print("time=",time2-time1,"s")



    print(np.round(np.sort(value[0]),2))
    time3 = time.time()
    a,b=np.linalg.eig(A)
    time4= time.time()
    print("time=",time4-time3,"s")
    print(np.round(np.sort(a),2))
    plt.figure(figsize=(15, 5),dpi=100)
    x = np.linspace(0,25, 1000)
    for i in range(len(Eigv)):
        y = 0
        for j in range(len(Eigv)):
            y += Eigv[i][j]*(gamma1[j]*np.cos(omega[j]*x)+wgamma2[j]/omega[j]
                                                    *np.sin(omega[j]*x))
        plt.plot(x, y,color = col[i%5],label = 'm_{}'.format(i))
    plt.legend(['m_1','m_2','m_3','m_4','m_5'] ,loc='upper right' )
    plt.ylim(-2,2)
    plt.show()
```

.1.5 N 自由度 + 画图 (未用 numpy 本征值函数)

运行环境

$Ubuntu9.3.0-17\ Ubuntu1\ 20.04LTS$

$Python3.8.8\quad Anaconda4.10.3\quad AMD-4800u$

```
import numpy as np
import matplotlib.pyplot as plt
import copy
N = 10
```

```
col = ['c','darkmagenta','orangE','royalblue','teal','r']

A=np.eye(N)

for i in range(N):
    A[i][i]=-2
    if (0<i):
        A[i][i-1]=1
    if (N-1>i):
        A[i][i+1]=1

print(A)

def Jacobi(A,eps,Eigv,detail):
    for ii in range(1,1000*len(A)):
        simga = 0
        apq = 0
        tan2 = 0
        N=len(A)
        for i in range(N):
            for j in range(i+1,N):
                    simga +=A[i][j]              #step1
                    if (abs(A[i][j])>abs(apq)):
                        apq = A[i][j]            #step2
                        (p,q) = (i,j)
        if ((A[q][q]-A[p][p])!=0):
            tan2 = 2*A[p][q]/(A[q][q]-A[p][p])
            theta = np.math.atan(tan2) / 2
        else:
            theta = np.math.pi/4
        cos = np.math.cos(theta)
        sin = np.math.sin(theta)
        cos2 = np.math.cos(2*theta)
        sin2 = np.math.sin(2*theta)
        Q=np.eye(N)
        Q[p][p]=cos
        Q[q][q]=cos
        Q[p][q]=sin
        Q[q][p]=-sin
        # A=Product(Product(Trans(Q),A),Q)
        app = float(A[p][p])
```

```
        aqq = float(A[q][q])
        apq = float(A[p][q])
        for j in range(len(A)):
            if (j!=q)and(j!=p):
                apj = float(A[p][j])
                aqj = float(A[q][j])
                A[p][j] = cos*apj - sin*aqj
                A[q][j] = sin * apj + cos * aqj
                A[j][p] = A[p][j]
                A[j][q] = A[q][j]

        A[p][p] = aqq*sin*sin + app*cos*cos -apq *sin2
        A[q][q] = app*sin*sin + aqq*cos*cos +apq *sin2
        A[p][q] = float(apq)*cos2 + (app-aqq)/2*(sin2)
        A[q][p] = A[p][q]

        #E = copy.copy(Product(Eigv,Q))
        for i in range(N):
            aip = Eigv[i][p]
            aiq = Eigv[i][q]
            Eigv[i][p] = cos*aip-sin*aiq
            Eigv[i][q]= sin*aip+cos*aiq
        #E=Eigv@Q
        # for i in range(N):
        #     for j in range(N):
        #         Eigv[i][j] = E[i][j]
        #print(np.round(A,10))
        if (detail):
            print("times={}\n{}".format(ii,np.round(A,4)))
            print(np.round(Eigv,4))
        sum = 0
        for i in range(N):
            for j in range(N):
                if (i != j):
                    sum += abs(A[i][j])
        if (sum < eps):
            break
    for i in range(N):
        value[0][i]=(A[i][i])
    for i in range(N):
        for j in range(N):
```

```
            B[i][j] = A[i][j]
    #print(np.round(Eigv,3))

    return (value)

def Product(A, B):
    C = np.zeros([len(A),len(B[0])])
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                C[i][j] += A[i][k]*B[k][j]
    return C

def Trans(A):
    C=np.zeros((len(A[0]),len(A)))
    for i in range (len(A[0])):
        for k in range(len(A)):
            C[i][k]=A[k][i]
    return C

if __name__ == "__main__":
    Eigv=np.eye(N)
    B=np.eye(len(A))
    G=np.eye(len(A))
    yy = np.zeros(N)
    xx = np.zeros(N)
    value = np.zeros([1,len(A)])
    Jacobi(A,1.e-10,Eigv,False)
    print("\t\t\t\t\t\t\tEigenvalue")
    print(np.round(value,10))
    print("\t\t\t\t\t\t\tEigenvector")
    print(np.round(Eigv,6))
    omega = np.sqrt(-value)[0].copy()

    print("\n\n\n")

    dotx0 = np.zeros([1,N])
    x0 = np.zeros([1,N])
    x0[0][0] = 1
    x = Product(Trans(Eigv),x0)
    x = (Trans(x))[0].copy()
```

```
    print("gamma1",x)
    gamma1 = x
    x = Product(Trans(Eigv),dotx0)
    x = (Trans(x))[0]
    print("wgamma2",x)
    wgamma2 = (x)
    plt.figure(figsize=(15, 8),dpi=100)
    x = np.linspace(0,25, 1000)
    for i in range(len(Eigv)):
            y = 0
            for j in range(len(Eigv)):
                y += Eigv[i][j]*(gamma1[j]*np.cos(omega[j]*x)+wgamma2[j]/
                                                    omega[j]*np.sin(omega[
                                                    j]*x))
            plt.plot(x, y,color = col[i%5],label = 'm_{}'.format(i))
        #plt.legend(['m_1','m_2','m_3','m_4','m_5'] ,loc='upper right' )
    plt.ylim(-2,2)
    plt.show()
```

.1.6 驻波画图

运行环境

*Ubuntu*9.3.0 − 17 *Ubuntu*1 20.04*LTS*

*Python*3.8.8 *Anaconda*4.10.3 *AMD* − 4800*u*

```
import numpy as np
import matplotlib.pyplot as plt
import time
import copy
N = 2000
col = ['c','darkmagenta','orangE','royalblue','teal','r']

A=np.eye(N)

for i in range(N):
    A[i][i]=-2
    if (0<i):
        A[i][i-1]=1
    if (N-1>i):
        A[i][i+1]=1

print(A)
```

```
def Jacobi(A,eps,Eigv,detail):
    for ii in range(1,1000*len(A)):
        simga = 0
        apq = 0
        tan2 = 0
        N=len(A)
        for i in range(N):
            for j in range(i+1,N):
                    simga +=A[i][j]              #step1
                    if (abs(A[i][j])>abs(apq)):
                        apq = A[i][j]            #step2
                        (p,q) = (i,j)
        if ((A[q][q]-A[p][p])!=0):
            tan2 = 2*A[p][q]/(A[q][q]-A[p][p])
            theta = np.math.atan(tan2) / 2
        else:
            theta = np.math.pi/4
        cos = np.math.cos(theta)
        sin = np.math.sin(theta)
        cos2 = np.math.cos(2*theta)
        sin2 = np.math.sin(2*theta)
        Q=np.eye(N)
        Q[p][p]=cos
        Q[q][q]=cos
        Q[p][q]=sin
        Q[q][p]=-sin
        # A=Product(Product(Trans(Q),A),Q)
        app = float(A[p][p])
        aqq = float(A[q][q])
        apq = float(A[p][q])
        for j in range(len(A)):
            if (j!=q)and(j!=p):
                apj = float(A[p][j])
                aqj = float(A[q][j])
                A[p][j] = cos*apj - sin*aqj
                A[q][j] = sin * apj + cos * aqj
                A[j][p] = A[p][j]
                A[j][q] = A[q][j]

        A[p][p] = aqq*sin*sin + app*cos*cos -apq *sin2
```

```
        A[q][q] = app*sin*sin + aqq*cos*cos +apq *sin2
        A[p][q] = float(apq)*cos2 + (app-aqq)/2*(sin2)
        A[q][p] = A[p][q]

        #E = copy.copy(Product(Eigv,Q))
        for i in range(N):
            aip = Eigv[i][p]
            aiq = Eigv[i][q]
            Eigv[i][p] = cos*aip-sin*aiq
            Eigv[i][q]= sin*aip+cos*aiq
        #E=Eigv@Q
        # for i in range(N):
        #     for j in range(N):
        #         Eigv[i][j] = E[i][j]
        #print(np.round(A,10))
        if (detail):
            print("times={}\n{}".format(ii,np.round(A,4)))
            print(np.round(Eigv,4))
        sum = 0
        for i in range(N):
            for j in range(N):
                if (i != j):
                    sum += abs(A[i][j])
        if (sum < eps):
            break
    for i in range(N):
        value[0][i]=(A[i][i])
    for i in range(N):
        for j in range(N):
            B[i][j] = A[i][j]
    #print(np.round(Eigv,3))

    return (value)

def Product(A, B):
    C = np.zeros([len(A),len(B[0])])
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                C[i][j] += A[i][k]*B[k][j]
    return C
```

```
def Trans(A):
    C=np.zeros((len(A[0]),len(A)))
    for i in range (len(A[0])):
        for k in range(len(A)):
            C[i][k]=A[k][i]
    return C

if __name__ == "__main__":
    Eigv=np.eye(N)
    B=np.eye(len(A))
    G=np.eye(len(A))
    yy = np.zeros(N)
    xx = np.zeros(N)
    value = np.zeros([1,N])
    time1 = time.time()
    #Jacobi(A.copy(),1.e-10,Eigv,False)
    time2 = time.time()

    time3 = time.time()
    (value,Eigv)=np.linalg.eig(A)
    time4 = time.time()


    print("\t\t\t\t\t\t\tEigenvalue")
    print(np.round(value,10))
    print("\t\t\t\t\t\t\tEigenvector")
    print(np.round(Eigv,4))

    # value = np.sort(value[0])

    # print(a-value)


    print("\n\n\n")
    dotx0 = np.zeros([1,N])
    x0 = np.zeros([1,N])
    x0[0][0] = 1
    x = Product(Trans(Eigv),x0)
    x = (Trans(x))[0].copy()
    print("gamma1",x)
```

```
gamma1 = x
x = Product(Trans(Eigv),dotx0)
x = (Trans(x))[0]
print("wgamma2",x)
wgamma2 = (x)


dotx0 = np.zeros([1,N])
x0 = np.zeros([1,N])
x0[0][0] = 1
x = ((Eigv.T)@x0[0])
x = x
print("gamma1",x)
gamma1 = x
x = ((Eigv.T)@dotx0[0])
x = x
print("wgamma2",x)
wgamma2 = (x)
omega = np.sqrt(-value).copy()
print("omega=",omega)
plt.figure(figsize=(15, 8),dpi=100)
# x = np.linspace(0,25, 1000)
# for i in range(len(Eigv)):
#         y = 0
#         for j in range(len(Eigv)):
#             y += Eigv[i][j]*(gamma1[j]*np.cos(omega[j]*x)+wgamma2[j]/
                                    omega[j]*np.sin(omega[j]*x))
#         plt.plot(x, y,color = col[i%5],label = 'm_{}'.format(i))
#     #plt.legend(['m_1','m_2','m_3','m_4','m_5'] ,loc='upper right' )
# plt.ylim(-2,2)

xx = np.linspace(0,N, N)
yy = []
x=10000
y = 0
yy.clear()
for j in range(len(Eigv)):
    y += Eigv[i][j]*(gamma1[j]*np.cos(omega[j]*x)+wgamma2[j]/omega[j]*np.
                                            sin(omega[j]*x))
    yy.append(y)
    y = 0
```

```
    plt.plot(xx, yy,alpha=0.2)
    plt.scatter(xx, yy,color="",edgecolor=col[0])
        #plt.legend(['m_1','m_2','m_3','m_4','m_5'] ,loc='upper right' )
    plt.show()
```

.1.7 本征值-自由度解析画图

运行环境

$Ubuntu9.3.0-17\ Ubuntu1\ 20.04LTS$

$Python3.8.8\quad Anaconda4.10.3\quad AMD-4800u$

```
import numpy as np
import matplotlib.pyplot as plt
import copy

n=np.linspace(0,1000)
y = 2*(1-np.cos(n/1001*np.math.pi))

plt.plot(n,y)
plt.show()
```

.2 固定边界条件下的一维原子链的集体振动模式的推导

在简谐近似下，一维原子链可简化为如课题探究的模型，细绳的倔强系数为 k（根据题目，k=1），振子的质量为 m，体系的总振子数目为 N（自由度）。易见，体系的拉氏量可写成：

$$\mathscr{L}=\sum_i \frac{1}{2}m\dot{x}_l-\frac{1}{2}k\left[x_1^2+(x_2-x_1)^2+\cdots+(x_N-x_{N-1})^2+x_N^2\right] \tag{34}$$

那么下面我们来求解一下这个体系的简正模与简正频率:

$$
\begin{aligned}
L &= \sum_{i=1}^{N}\frac{1}{2}m\dot{x}_i - \sum_{i=2}^{N}\frac{1}{2}k\left(x_i - x_{i-1}\right)^2 + \frac{1}{2}k\left(x_N^2 - x_1^2\right) \\
&= \sum_{i=1}^{N}\frac{1}{2}m\dot{x}_l - \sum_{i=2}^{N}\frac{1}{2}k\left(x_i^2 - 2x_ix_{i-1} + x_{i-1}^2\right) + \frac{1}{2}k\left(x_N^2 - x_1^2\right) \\
U &= \sum_{i=2}^{N}\frac{1}{2}k\left(x_i^2 + x_{i-1}^2\right) - \sum_{i=2}^{N}kx_ix_{i-1} + \frac{1}{2}k\left(x_N^2 - x_1^2\right) \\
dU &= \sum_{i=2}^{N}k\left(x_idx_i + x_{i-1}dx_{i-1}\right) - \sum_{i=2}^{N}k\left(dx_ix_{i-1} + x_idx_{i-1}\right) + k\left(x_Ndx_N - x_1dx_1\right) \\
\frac{\partial U}{\partial x_\alpha} &= \sum_{i=2}^{N}k\left(x_i\frac{\partial x_i}{\partial x_\alpha} + x_{i-1}\frac{\partial x_{i-1}}{\partial x_\alpha}\right) - \sum_{i=2}^{N}k\left(\frac{\partial x_i}{\partial x_\alpha}x_{i-1} + x_i\frac{\partial x_{i-1}}{\partial x_\alpha}\right) + k\left(x_N\frac{\partial x_N}{\partial x_\alpha} - x_1\frac{\partial x_1}{\partial x_\alpha}\right) \\
&= \sum_{i=2}^{N}k\left(x_i\delta_\alpha^i + x_{i-1}\delta_\alpha^{i-1}\right) - \sum_{i=2}^{N}k\left(x_{i-1}\delta_\alpha^i + x_i\delta_\alpha^{i-1}\right) + k\left(x_N\delta_\alpha^N - x_1\delta_{ij}^1\right]
\end{aligned}
$$

由上式可知:

(1) 当 $\alpha = 1$ 时

$$\frac{\partial U}{\partial x_\alpha} = kx_1 - kx_2 - kx_1 = -kx_2$$

(2) 当 $\alpha = N$ 时

$$\frac{\partial U}{\partial x_\alpha} = kx_N - kx_{N-1} - kx_N = 2kx_N - kx_{N-1}$$

(3) 当 $\alpha = 2, 3, 4, \ldots, N-1$ 时

$$\frac{\partial U}{\partial x_\alpha} = kx_\alpha + kx_\alpha - kx_{\alpha-1} - kx_{\alpha+1} = k\left(2x_\alpha - x_{\alpha-1} - x_{\alpha+1}\right)$$

因此对于 $\frac{\alpha^2 U}{\alpha x_\alpha \alpha x_\beta}$ 有: (1) 当 $\alpha = 1, \beta = 2$ 时

$$\frac{\partial^2 U}{\partial x_\alpha \alpha x_\beta} = -k$$

(2) 当 $\alpha = 1, \beta \neq 2$ 时

$$\frac{\partial^2 U}{\partial x_\alpha \partial x_\beta} = 0$$

(3) 当 $\alpha = N, \beta = N$

$$\frac{\partial^2 U}{\partial x_\alpha \partial x_\beta} = 2k$$

(4) 当 $\alpha = N, \beta = N-1$ 时

$$\frac{\partial^2 U}{\partial x_\alpha \partial x_\beta} = -k$$

(5) $\alpha = N, \beta \neq N$ 或 $N-1$ 时

$$\frac{\partial^2 U}{\partial x_\alpha \partial x_\beta} = 0$$

(6) 当 $\alpha = 2, 3, 4, \ldots, N-1$ 时

$$\frac{\partial^2 U}{\partial x_\alpha \partial x_\beta} = k\left(2\delta_\beta^\alpha - \delta_\beta^{\alpha-1} - \delta_\beta^{\alpha+1}\right)$$

综上所述, $\hat{U} = \left[\frac{\partial^2 U}{\partial x_\alpha \partial x_\beta}\right]_{\alpha\beta}$ 矩阵可写为

$$\hat{U} = k\begin{bmatrix} 2 & -1 & 0 & \cdots & \cdots & \\ -1 & 2 & -1 & \ddots & & \\ 0 & -1 & 0 & & \ddots & \\ 0 & & \ddots & \ddots & & \\ \vdots & & & \ddots & \ddots & -1 \end{bmatrix}_{NXN}$$

而又因 $T = \frac{1}{2}\sum_i m_{ik}\dot{x}_l\dot{x}_k = \sum_{i=1}^N \frac{1}{2}m\dot{x}_l \therefore \hat{M} = [m_{ik}]_{ik}$ 矩阵可写为

$$\hat{M} = m\begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$$

于是体系的久期方程为 $\det\left(-w^2\hat{M} + \hat{U}\right) = 0$, 即

$$\begin{bmatrix} 2k-w^2m & -k & \cdots & \\ -k & 2k-w^2m & -k & \cdots \\ 0 & -k & 2k-w^2m & \\ \vdots & \vdots & & \end{bmatrix}_{N\times N} = 0 \text{ (在后面的推导中, 我们将该行列记为} D_N \text{)}$$

易发现久期方程为一个三对角矩阵的行列式, 将其按第一行展开有:

$$D_N = \left(2k-w^2m\right) D_{N-1} - k^2 D_{N-2} \cdots\cdots$$

其中 $D_1 = 2k-w^2m, D_2 = \begin{vmatrix} 2k-w^2m & -k \\ -k & 2k-w^2m \end{vmatrix} = \left(2k-w^2m\right) - k^2$ 引入待定参数 α 和 β, 满足:

$$\alpha+\beta = 2k-w^2m, \alpha\beta = k^2 \left(\alpha+\beta = D_1, \alpha\beta = D_1^2 - D_2\right)$$

使 (1) 式变为:

$$D_N - \alpha D_{N-1} = \beta \left(D_{N-1} - \alpha D_{N-2}\right) \cdots\cdots$$

为求参数 α 和 β, 令:

$$\lambda^2 - \left(2k-w^2m\right)\lambda + k^2 = 0$$

上式即为本征方程, 而 α β 为期的两根计算本征方程的判别式, 有:

$$\Delta = b^2 - 4ac = \left(2k-w^2m\right)^2 - 4k^2 = \left(2k-w^2m\right)^2 - (2k)^2 < 0$$

因此, 本征方程有两个互为共轭的复根:

$$\alpha = \frac{1}{2}\left(2k-w^2m\right) + i\frac{1}{2}\sqrt{-\Delta}$$
$$\beta = \frac{1}{2}\left(2k-w^2m\right) - i\frac{1}{2}\sqrt{-\Delta}$$

注意到 α 和 β 实际上地位是价位的, 因此 (2) 式可写为

$$\begin{cases} D_N - \alpha D_{N-1} = \beta \left(D_{N-1} - \alpha D_{N-2} \right) \cdots\cdots \\ D_N - \beta D_{N-1} = \alpha \left(D_{N-1} - \beta D_{N-2} \right) \cdots\cdots \end{cases}$$

(3) 式告诉我们, $\{D_N - \alpha D_{N-1}\}$ 是以 β 为公比的等比数列, 因此有:

$$D_N - \alpha D_{N-1} = \beta^{N-2} \left(D_2 - \alpha D_1 \right) \cdot\cdots\cdots$$

同样, (4) 式告诉我们, $\{D_N - \beta D_{N-1}\}$ 是以 α 为公比的等比数列, 因此有:

$$D_N - \beta D_{N-1} = \alpha^{N-2} \left(D_2 - \beta D_1 \right) \cdot\cdots\cdots$$

联立 (5)、(6) 式, 消去 D_{N-1} 得:

$$\begin{gathered} D_N - \beta D_{N-1} = \alpha^{N-2} \left(D_2 - \beta D_1 \right) \\ D_N = \frac{1}{\alpha - \beta} \left[\alpha^{N-1} \left(D_2 - \beta D_1 \right) - \beta^{N-1} \left(D_2 - \alpha D_1 \right) \right] \\ = \frac{1}{\alpha - \beta} \left[\alpha^{N-1} D_2 - \alpha^{N-1} \beta D_1 - \beta^{N-1} D_2 + \beta^{N-1} \alpha D_1 \right] \\ = \frac{1}{\alpha - \beta} \left[D_2 \left(\alpha^{N-1} - \beta^{N-1} \right) + D_1 \left(\alpha \beta^{N-1} - \beta \alpha^{N-1} \right) \right] \\ \because \alpha + \beta = D_1, \alpha + \beta = D_1^2 - D_2 \\ \therefore D_2 = D_1^2 - (\alpha + \beta) = (\alpha + \beta)^2 - (\alpha + \beta) = \alpha^2 + \beta^2 + \alpha\beta \\ \therefore D_N = \frac{1}{\alpha - \beta} \left[\left(\alpha^2 + \beta^2 + \alpha\beta \right) \left(\alpha^{N-1} - \beta^{N-1} \right) + (\alpha + \beta) \left(\alpha \beta^{N-1} - \beta \alpha^{N-1} \right) \right] \end{gathered}$$

化简后有

$$D_N = \frac{1}{\alpha - \beta} \left[D_2 \left(\alpha^{N-1} - \beta^{N-1} \right) + D_1 \left(\alpha \beta^{N-1} - \beta \alpha^{N-1} \right) \right]$$

同时

$$\because \alpha+\beta=D_1, \alpha+\beta=D_1^2-D_2$$

$$\therefore D_2=D_1^2-(\alpha+\beta)=(\alpha+\beta)^2-(\alpha+\beta)=\alpha^2+\beta^2+\alpha\beta$$

$$\begin{aligned}\therefore D_N &= \frac{1}{\alpha-\beta}\left[\left(\alpha^2+\beta^2+\alpha\beta\right)\left(\alpha^{N-1}-\beta^{N-1}\right)+(\alpha+\beta)\left(\alpha\beta^{N-1}-\beta\alpha^{N-1}\right)\right] \\ &= \frac{1}{\alpha+\beta}\left[\left(\alpha^{N+1}-\alpha^2\beta^{N-1}+\beta^2\alpha^{N-1}-\beta^{N+1}+\alpha^N\beta-\alpha\beta^N\right)+\alpha^2\beta^{N-1}-\beta\alpha+\alpha\beta^N\right. \\ &\quad \left.-\beta^2\alpha^{N-1}\right] \\ &= \frac{1}{\alpha+\beta}\left(\alpha^{N+1}-\beta^{N-1}\right)\end{aligned}$$

$$令\alpha=re^{i\theta}\ \beta=re^{-i\theta}$$

则久期方程可化为:

$$\begin{aligned}&\therefore e^{i(2N+2)\theta}=1 \\ &\therefore \theta_n=\frac{n}{N+1}\pi(n=1,2,3\ldots N) \\ &\because \alpha=\frac{1}{2}\left(2k-w^2m\right)+i\frac{1}{2}\sqrt{4k^2-\left(2k-w^2m\right)^2} \\ &\therefore \tan\theta_n=\frac{\sqrt{4k^2-\left(2k^2-w_n^2m\right)^2}}{2k-w_n^2m} \\ &\therefore \left(2k^2-w_n^2m\right)^2\tan^2\theta_n=4k^2-\left(2k^2-w_n^2m\right)^2 \\ &\therefore \left(2k^2-w_n^2m\right)^2\left(\tan^2\theta_n+1\right)=4k^2 \\ &\therefore \left(2k^2-w_n^2m\right)^2=\frac{4k^2}{\tan^2\theta_n+1} \\ &\therefore 2k^2-w_n^2m=\pm 2k\cdot\left(\tan^2\theta_n+1\right)^{-\frac{1}{2}} \\ &\therefore w_n^2=\frac{2k}{m}\left[1\pm\left(\tan^2\theta_n+1\right)^{-\frac{1}{2}}\right]\left(\theta_n=\frac{n}{N+1}\pi, n=1,2,3\ldots N\right) \\ &\because \tan^2\theta_n+1=\frac{1}{\cos^2\theta_n}\end{aligned}$$

$\therefore w_n^2=\frac{2k}{m}\left[1-\cos\theta_n\right]$ (此处将士取作 "-" 的原因为 $\theta_n\in(0,\pi)$, 因此 "-" 包含了 "+" 所有情况)

$$\because \hat{M}=m\cdot\hat{I}$$

$\therefore$ 对于体系的简正模, 我们只需考虑 $\hat{U}$ 的本征向量即可猜测与 w_n 相应的简正模的形式为

$$\left[\sin\frac{n\pi}{N+1},\sin\frac{2n\pi}{N+1},\sin\frac{3n\pi}{N+1},\ldots\ldots,\sin\frac{Nn\pi}{N+1}\right]^T$$

此简正模的形式相当于驻波（y = y1 + y2 = 2 Acos 2π(x/λ) cos 2π(t/T)), 所以在固定边界条件下猜此解是合理的。我们不妨代入验证:

$$k\begin{bmatrix} 2 & -1 & 0 & \cdots & \cdots \\ -1 & 2 & -1 & 0 & \cdots \\ 0 & -1 & 2 & -1 & \\ \vdots & \vdots & \vdots & \ddots & \\ & & & \ddots & \end{bmatrix}_{NXN}\begin{bmatrix} \sin\frac{n\pi}{N+1} \\ \sin\frac{2n\pi}{N+1} \\ \sin\frac{3n\pi}{N+1} \\ \vdots \\ \sin\frac{N}{N+1}n\pi \end{bmatrix} = k\begin{bmatrix} 2\sin\frac{n\pi}{N+1} - \sin\frac{2n\pi}{N+1} \\ -\sin\frac{n\pi}{N+1} + 2\sin\frac{2n\pi}{N+1} - \sin\frac{3n\pi}{N+1} \\ \frac{2n\pi}{N+1} + 2\sin\frac{3n\pi}{N+1} + \sin\frac{4n\pi}{N+1} \\ \vdots \\ -\sin\frac{(N-1)}{(N+1)}n\pi + 2\sin\frac{N}{N+1}n\pi \end{bmatrix}$$

$$2\sin\frac{n\pi}{N+1} - 2\sin\frac{n\pi}{N+1}\cos\frac{n\pi}{N+1} = 2\sin\frac{n\pi}{N+1}\left(1-\cos\frac{n\pi}{N+1}\right)$$

$$\begin{aligned}
&-\sin\frac{n\pi}{N+1} + 2\sin\frac{2n\pi}{N+1} - \sin\frac{3n\pi}{N+1} \\
&= -\sin\frac{n\pi}{N+1} + 2\sin\frac{2n\pi}{N+1} - \left(\sin\frac{2n\pi}{N+1}\cos\frac{n\pi}{N+1} + \sin\frac{n\pi}{N+1}\cos\frac{2n\pi}{N+1}\right) \\
&= -\sin\frac{n\pi}{N+1} - \sin\frac{n\pi}{N+1}\cos\frac{2n\pi}{N+1} + 2\sin\frac{2n\pi}{N+1} - \sin\frac{2n\pi}{N+1}\cos\frac{n\pi}{N+1} \\
&= \left(-\sin\frac{n\pi}{N+1} - \sin\frac{n\pi}{N+1}\cos\frac{2n\pi}{N+1} + \sin\frac{2n\pi}{N+1}\cos\frac{n\pi}{N+1}\right) + 2\sin\frac{2n\pi}{N+1}\left(1-\cos\frac{n\pi}{N+1}\right) \\
&= \left[-\sin\frac{n\pi}{N+1} - \sin\frac{n\pi}{N+1}\left(2\cos^2\frac{2n\pi}{N+1} - 1\right) + 2\sin\frac{n\pi}{N+1}\cos^2\frac{n\pi}{N+1}\right] + 2\sin\frac{2n\pi}{N+1}(1 \\
&-\cos\frac{n\pi}{N+1}\Big) \\
&= 2\sin\frac{2n\pi}{N+1}\left(1-\cos\frac{n\pi}{N+1}\right) \\
&-\sin\frac{N-1}{N+1}n\pi + 2\sin\frac{N}{N+1}n\pi \\
&= -\left[\sin\frac{N}{N+1}n\pi\cos\frac{1}{N+1}n\pi - \sin\frac{1}{N+1}n\pi\cos\frac{N}{N+1}n\pi\right] + 2\sin\frac{N}{N+1}n\pi \\
&= -\left[\sin\frac{N}{N+1}n\pi\cos\frac{1}{N+1}n\pi - \sin\frac{1}{N+1}n\pi\cos\frac{N}{N+1}n\pi - 2\sin\frac{N}{N+1}n\pi\cos\frac{1}{N+1}n\pi\right] \\
&+ 2\sin\frac{N}{N+1}n\pi - 2\sin\frac{N}{N+1}n\pi\cos\frac{1}{N+1}n\pi \\
&= \left[-\sin\frac{1}{N+1}n\pi\cos\frac{N}{N+1}n\pi - \sin\frac{N}{N+1}n\pi\cos\frac{1}{N+1}n\pi\right] + 2\sin\frac{N}{N+1}n\pi(1 \\
&-2\cos\frac{n}{N+1}\pi\Big) \\
&= -\sin\frac{N+1}{N+2}n\pi + 2\sin\frac{N}{N+1}n\pi\left(1-2\cos\frac{n}{N+1}\pi\right)
\end{aligned}$$

$$\therefore k\begin{bmatrix} 2 & -1 & \cdots \\ -1 & 2 & \cdots \\ \vdots & & \ddots \end{bmatrix}_{NXN}\begin{bmatrix} \sin\frac{n\pi}{N+1} \\ \sin\frac{2n\pi}{N+1} \\ \vdots \\ \sin\frac{N}{N+1}n\pi \end{bmatrix} = 2k\left(1-\cos\frac{n}{N+1}\pi\right)\begin{bmatrix} \sin\frac{n\pi}{N+1} \\ \sin\frac{2n\pi}{N+1} \\ \vdots \\ \sin\frac{N}{N+1}n\pi \end{bmatrix}$$

即, 向量

$$\left[\sin\frac{n\pi}{N+1}, \sin\frac{2n\pi}{N+1}, \sin\frac{3n\pi}{N+1}, \ldots\ldots, \sin\frac{Nn\pi}{N+1}\right]^T$$

确实为体系的简正模, 其所对应的简正频率为

$$w_n^2 = \frac{2k}{m}\left(1-\cos\theta_n\right)\left(\theta_n = \frac{n}{N+1}\pi, n = 1, 2, 3 \ldots N\right)$$

.