华南理工大学
South China University of Technology

《仿真技术在电气工程中领域的应用》
课程作业

(2019-2020　学年第一学期)

作业 0202 稀疏线性方程组的求解

作业 0203 微分方程的求解

作业 0302 多机系统短路故障后时域仿真

学院：电力学院

任课老师：武志刚

组员信息及分工

组员：阮艺源 胡嘉铭 周欣缘 汪笑雨

阮艺源　17 级电气工程卓越班　201730210466

1. 问题 1 部分 Python 程序编写

2. 问题 2 基于欧拉法、改进欧拉法、隐式梯形法的微分方程求解的 Python 程序编写、绘图、中文文档写作

3. 问题 3 通用牛顿拉夫逊法、隐式梯形法接口的 Python 程序编写、绘图、相应部分文档写作

胡嘉铭　17 级电气工程卓越班　201730220489

1. 问题 1 部分 Python 程序编写、中文文档写作

2. 问题 2 讨论并共同设计编程思路

3. 问题 3 输入阻抗、转移阻抗的通用计算（包括 Y.py 和 impedance.py 文件）、通用数据格式接口的设计及程序编写、相应部分中文文档写作

周欣缘　17 级电气工程中澳班　201730241286

1. 问题 1 MATLAB 程序版本

2. 问题 2 MATLAB 程序版本

3. 问题 3 MATLAB 程序版本

4. 英文版本的论文写作

汪笑雨　16 级电气工程五班　201530212585

1. 问题 1 讨论程序编写思路、修改论文

2. 问题 2 讨论程序编写思路、修改论文

3. 问题 3 节点导纳矩阵的程序编写、修改论文

作业 0202 稀疏线性方程组的求解

一、问题重现

利用课堂上介绍的稀疏线性方程组求解方法编写通用程序，求解下述方程组。

$$Ax = b$$

其中：

$$A = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 2 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \end{pmatrix}$$

$$b = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 5 & 5 & 5 & 5 \end{pmatrix}^T$$

二、解题思路

1. 基本作业要求

由于本文需解决的问题是：编写求解稀疏线性方程组的通用程序，因此可采用基于因子表分解的求解线性方程组的方法。

假设需要求解的方程组为：

$$Ax = b$$

又因为每个方阵都可以进行 LDU 分解，因此矩阵 A 可分解为下三角矩阵 L、对角矩阵 D、上三角矩阵的乘积 U：

$$A = LDU$$

则求解方程组的步骤可分解为以下三步——前代、规格化、回代：

$$Lz = \mathrm{b} \quad \text{其中} \quad z = \mathrm{DUx}$$

$$D\mathrm{y}=\mathrm{z} \quad \text{其中} \quad \mathrm{y}=\mathrm{Ux}$$

$$U\mathrm{x}=\mathrm{y}$$

由此可解出未知量 x。基于以上方法能将复杂的线性方程组的求解过程，简化为循环代入过程。

由于矩阵 A 是稀疏矩阵，我们对其进行三角检索存储，可免去对大量零元素进行存储，节约了大量内存。同时，由于我们求解线性方程组时也需要用到 L、D、U 矩阵，对矩阵完成存储后，也得到了 L、D、U 矩阵，此举可谓是一举两得。

2. 加分项

以上分析能够解决 A 矩阵稀疏时的问题。对于实际情况，有时可能我们只需要知道 x 矩阵中某个分量的值，而不需要将整个矩阵 x 求解出来。此时，我们可以利用稀疏向量法实现该想法，并且能够提高运算速度。

假设我们需要求解矩阵 x 中的第 x_index 个分量。运用稀疏向量法时，我们可以在原来的回代步骤中进行化简，将与第 x_index 个分量无关的分量的计算给舍去。经过分析，第 i 个元素和 x_index 无关的充分条件是 U[x_index-1:i-1][i-1]全为 0 或 i < x_index。只需将矩阵 U 中与第 x_index 个分量无关的分量对应的行列删去，再用处理后的矩阵 U 和 x 进行回代计算，即可实现稀疏向量法。

三、实现程序

本文程序是基于 Python。

1. 三角检索存储

先对矩阵 A 进行 LDU 分解，将 L、D、U 三个矩阵结合成一个矩阵 B，该矩阵 B 则为矩阵 A 的因子表。对其因子表进行散居存储，即实现了对原矩阵的三角检索存储。

程序中，函数 LDU(A)即为将矩阵 A 进行 LDU 分解的函数，输入为矩阵 A，输出依次为 L、D、U 三个矩阵。

函数 LDU_store(A)为将矩阵 A 进行三角检索存储的函数，函数中首先调用了函数 LDU，构建矩阵 A 的因子表。该函数输出为散居存储格式，为三个元组:

(L_value,L_row,L_column),(D_value),(U_value,U_row,U_column)

第一个元组有三个 list 元素：L 矩阵的元素、其对应的行号、其对应的列号；

第二个元组有三个 list 元素：D 矩阵的元素；

第三个元组有三个 list 元素：U 矩阵的元素、其对应的行号、其对应的列号。

由于三角检索存储已能够生成 L、D、U 矩阵，下面的步骤可直接使用 LDU_store 函数的输出结果进行计算。

2. 前代计算

由于 L 为下三角矩阵，经过一系列化简，求解 Lz=b 的步骤可简化为 n 次代入，依次求出 z_1、z_2、z_3…z_n ，如下所示：

$$\begin{cases} z_1 = b_1 \\ z_2 = b_2 - l_{21} z_1 \\ \vdots \\ z_i = b_i - \sum_{j=1}^{i-1} l_{ij} z_j \\ \vdots \\ z_n = b_n - \sum_{j=1}^{n-1} l_{nj} z_j \end{cases}$$

程序中，函数 front(L,b)则为实现前代过程的函数，其输入为 LDU_store( ) 函数输出的第一个元组及 b 矩阵，输出为 z 矩阵。

3. 规格化计算

由于 D 为对角矩阵，因此求解 Dy=z 的步骤可简化为 n 次除法，依次求出 y_1、y_2、y_3…y_n ，如下所示：

$$y_i = \frac{z_i}{d_{ii}} \qquad (i = 1, \cdots, n)$$

程序中，函数 mid(D,z)则为实现前代过程的函数，其输入为 LDU_store( )函数输出的第二个元组及 z 矩阵，输出为 y 矩阵。

4. 回代计算

由于 U 为下三角矩阵，经过一系列化简，求解 Ux=y 的步骤可简化为 n 次代入，依次求出 x_n、x_{n-1}、x_{n-2}…x_1 ，如下所示：

$$\begin{cases} x_n = y_n \\ x_{n-1} = y_{n-1} - u_{n-1,n}x_n \\ \vdots \\ x_i = y_i - \sum_{j=i+1}^{n} u_{ij}x_j \\ \vdots \\ x_1 = y_1 - \sum_{j=2}^{n} u_{1j}x_j \end{cases}$$

程序中，函数 back(U,y)则为实现前代过程的函数，其输入为 LDU_store( )函数输出的第三个元组及 y 矩阵，输出为 x 矩阵，即线性方程组的解。

本程序将前代、规格化、回代三个步骤包装在函数 solve(A,b)中，其输入为系数矩阵 A 和常量矩阵 b，输出为需求解的未知量矩阵 x。

5．稀疏向量法

在第二节已阐述了稀疏向量法的目的、意义及实现方法，下面具体介绍实际函数。本程序通过函数 solve_Sparse_vector(A,b,x_index)实现稀疏向量法，其输入为系数矩阵 A、常量矩阵 b 和未知量矩阵 x 中需求解分量的序号 x_index，输出为需求解分量的值。

四、结果展示

本程序将原始数据通过参数传入函数 solve( )中，解得未知量矩阵 x，并且通过 solve_Sparse_vector( )函数利用稀疏向量法求解出矩阵 x 中的第三个分量 x_3。

```
A=np.array([[2,0,0,0,0,0,1,0,0,0,0,1],
            [0,2,0,0,1,0,0,0,1,0,0,0],
            [0,0,2.,0,0,1,0,0,1,0,0,0],
            [0,0,0,2,0,0,1,0,0,1,0,0],
            [0,1,0,0,2,0,0,0,0,0,0,1],
            [0,0,1,0,0,2,0,0,0,1,0,0],
            [1,0,0,1,0,0,2,0,0,0,0,0],
            [0,0,0,0,0,0,0,2,1,0,1,0],
            [0,1,1,0,0,0,0,1,2,0,0,0],
            [0,0,0,1,0,1,0,0,0,2,1,0],
            [0,0,0,0,0,0,0,1,0,1,2,1],
            [1,0,0,0,1,0,0,0,0,0,1,2]])
b=np.array([[4],[4],[4],[4],[4],[4],[4],[4],[5],[5],[5],[5]])

print("解得的X为：")
print(solve(A,b))
print("需求的特定变量的值为：")

print(solve_Sparse_vector(A,b,3))
```

```
解得的X为：
[[1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]]
需要求解的特定变量的值为：
[1.]
```

作业 0203 微分方程的求解

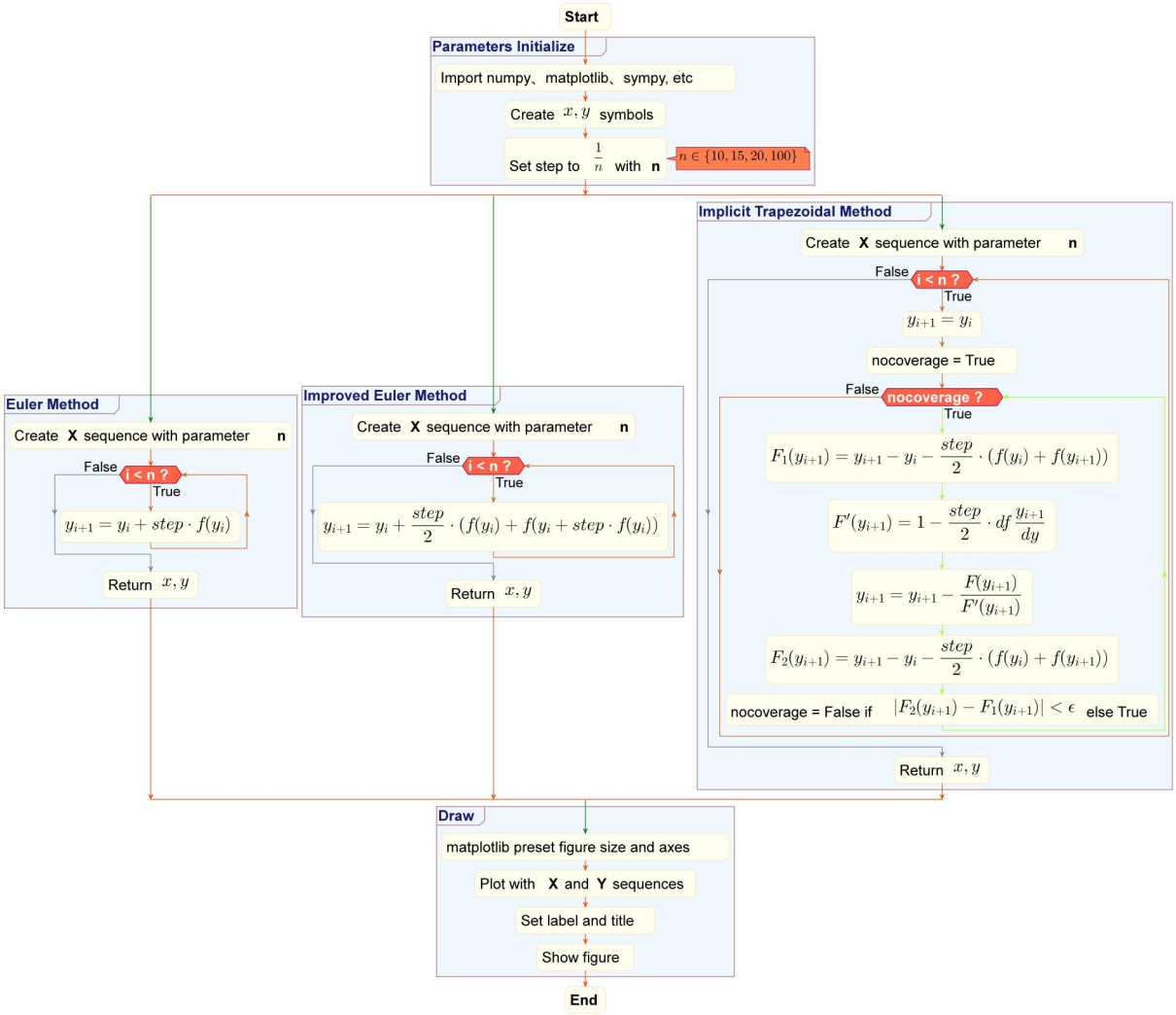
一、问题重述

求解具有如下初值的微分方程

$$\begin{cases} \dfrac{dy}{dx} = f(y) & x \in [a,b] \\ y(x_0) = y_0 \end{cases}$$

方程中 f(y)不显含 x，求解的结果为 x 与 y 在定义域[a,b]上的数值序列

二、程序涉及的微分方程求解的流程图

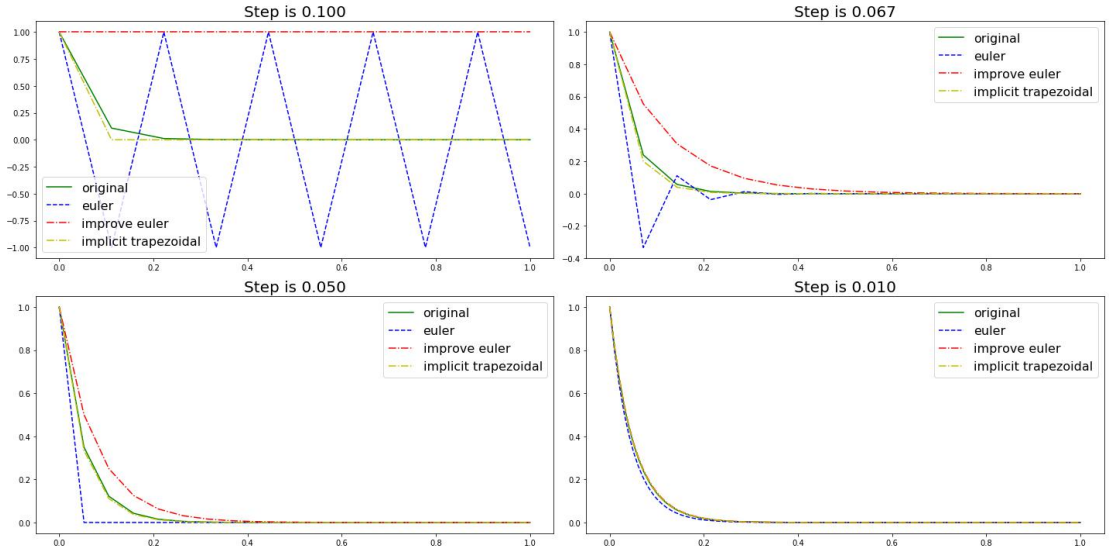


三、程序运行后的结果对比

求解的具体微分方程为：

$$\begin{cases} \dfrac{dy}{dx} = -20x & x \in [0,1] \\ y(0) = 1 \end{cases}$$

当步长 $\Delta x = \dfrac{1}{n}$ 中 n 分别取 10，15，20，100 对应的欧拉法、改进欧拉法、隐式梯形法的图像分别如下：



1.当步长为 **0.1** 时(n=10)

欧拉法求得的图像上下波动，原因是 f(y)=-20y 与 y 负相关，过大的步长产生较大的误差，甚至使与其叠加后的 y_1 为负值，于是 $f(y_1)$变为正值，叠加后导致 y_2 又变为正值，如此往复，甚至形成一个无法收敛的波动。

改进欧拉法为一条与 x 轴平行的直线，其原因在于 y_0=1，$f(y_0)+f(y_0+0.1\cdot f(y_0))$=0,则叠加后 $y_1=y_0$=1,于是根据数学归纳法，$y_n=y_{n-1}=...=y_0$=1，即改进欧拉法的图像为一条直线。

隐式梯形法由于每一轮循环都要迭代至该轮收敛，因而在 step=0.1 时就能很快收敛至原函数。

2. 当步长为 **0.067** 时(n=15)

欧拉法的图像为衰减地收敛到与真实曲线重合，而改进欧拉法较真实解偏大，隐式梯形法的曲线依然与真实曲线高度吻合。

3 当步长为 **0.05** 时(n=20)

欧拉法图像在 y_1 后就保持为 0 值，原因是 $y_1=y_0+0.05*f(y_0)=0$, 则 $y_2=y_1+0.05*f(y_1)=0$,由数学归纳法 $y_n=y_{n-1}=...=y_1=0$

改进欧拉法依然较真实解偏大。

4 当步长为 **0.01** 时(n=50)

三种方法的解皆与真实解贴近，且三者的精确度为 ε排序为 $\varepsilon_{隐}>\varepsilon_{改}>\varepsilon_{欧}$。

四、微分方程求解方法的数值评价

由于方程为非线性方程，不能选取拟合优度 $\mathbf{R^2}$ 作为评价指标，这里直接选取误差平方和 **ESS** 作为评价指标，**ESS** 指标如下：

$$ESS=\sum_{i=1}^{n}(\hat{y}_i-y_i)^2$$

Method / N	Euler	Improve Euler	Implicit Trapezoidal
10	9.20742838	8.76880526	0.01188318
15	0.333880795	0.200184481	0.001912445
20	0.138710433	0.049243171	0.000398136
100	0.011638651	0.000396699	5.57732E-05

注：蓝色为低精度，粉红色为高精度

由色阶图，欧拉法和改进欧拉法在步长很大时的精度远低于隐式梯形法，随着步长减小，欧拉法和隐式梯形法的精度均有明显提升，而在步长变化过程中隐式梯形法始终能保持很高的精度。

作业 0302 多机系统短路故障后时域仿真

一、问题重述

给定初始状态稳定和元件参数已知的多机电力系统，对其短路故障和故障切除后的状态进行时域仿真。

二、问题建模

1. 主要模型

研究电力系统运行的稳定性，主要是确定各同步电机是否处于同步运行状态，即各转子间有无相对摇摆。功角δ随时间变化描述了各发电机转子间的相对运动，恰好反映了各发电机之间是否同步。

根据算例所给的条件特点，考虑使用同步电机转子二阶模型，即

$$\begin{cases} \frac{d\Delta\omega}{dt} = (P_m - P_e)/T_J \\ \frac{d\delta}{dt} = \Delta\omega\omega_B \end{cases}$$

由隐式梯形法得，t_n 至 t_{n-1} 时步的差分方程为

$$\begin{cases} q_\omega = \Delta\omega_{n+1} - \Delta\omega_n - \frac{h}{2T_J}\left[(P_{m,n+1} - P_{e,n+1}) + (P_{m,n} - P_{e,n})\right] = 0 \\ q_\delta = \delta_{n+1} - \delta_n - \frac{\omega_B h}{2}(\Delta\omega_{n+1} + \Delta\omega_n) = 0 \end{cases}$$

对于三机系统，任意一台发电机的电磁功率如下

$$P_{ei} = \frac{E_i^2}{|Z_{ii}|}\sin\alpha_{ii} + \frac{E_i E_j}{|Z_{ij}|}\sin(\delta_{ij} - \alpha_{ij}) + \frac{E_i E_k}{|Z_{ik}|}\sin(\delta_{ik} - \alpha_{ik})$$

其中

$$\begin{aligned} \delta_{ij} &= \delta_i - \delta_j \\ \delta_{ik} &= \delta_i - \delta_k \\ \delta_{jk} &= \delta_j - \delta_k \end{aligned}$$

2. 方程组中未知参数的求解

本小节主要针对电磁功率 P_e 表达式中的未知参数进行计算。其中，未知参数主要为各台发电机的输入阻抗和之间的转移阻抗，P_e 表达式中需要用到上述阻抗的模值及阻抗角。

计算输入阻抗和转移阻抗的关键是生成节点阻抗矩阵，根据计算公式即能算得所需阻抗值。为了正确生成故障发生时和故障切除后的节点阻抗矩阵，需要按

以下步骤进行操作：

（1） 定义通用的数据格式

（2） 生成正负零序的节点导纳矩阵

（3） 计算附加阻抗

（4） 生成节点阻抗矩阵，计算发电机节点的输入阻抗和之间的转移阻抗

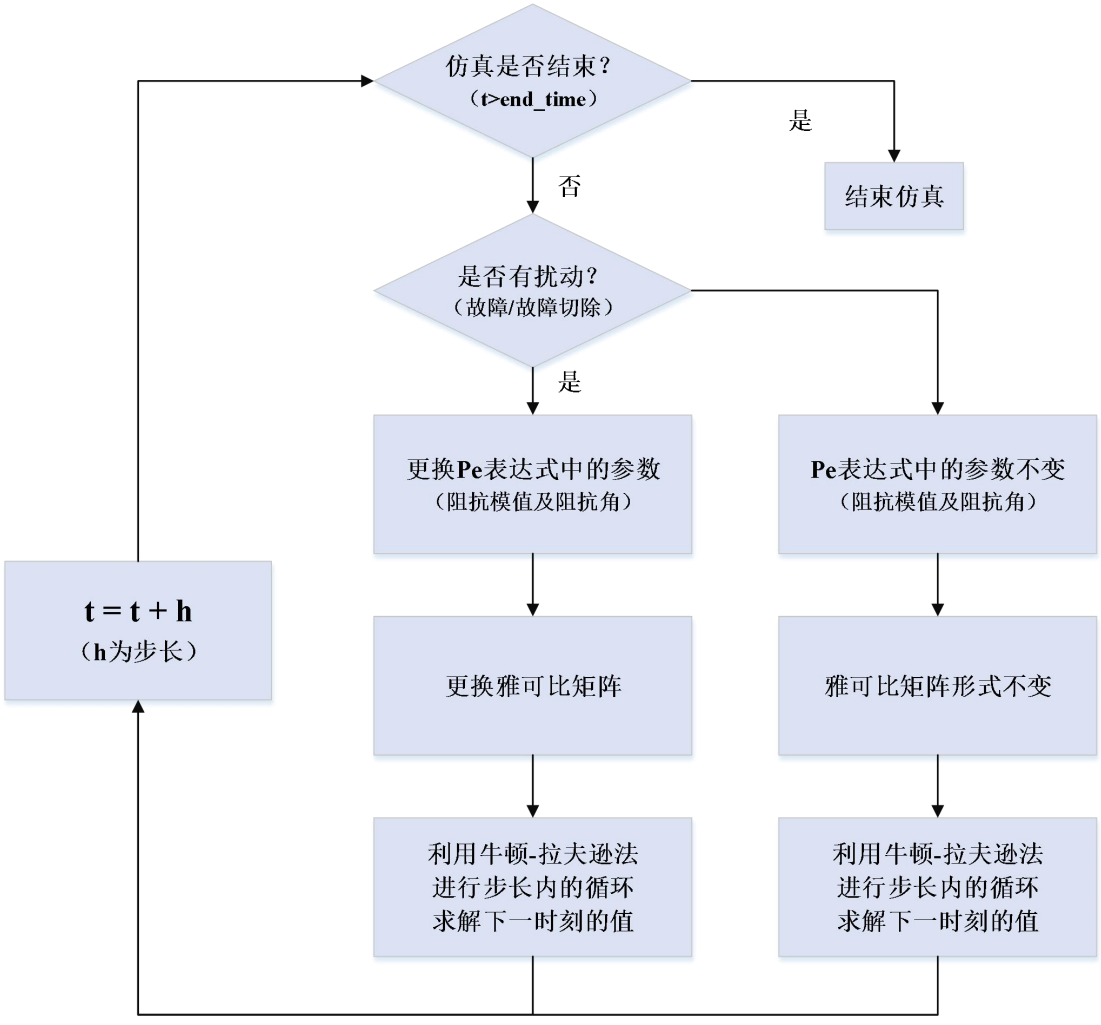
具体程序及实现方法将在第三节进行详细说明。

三、程序流程

本程序的实现流程及具体实现方法将在本节详细介绍。

1. 主程序

主程序，即利用牛顿-拉夫逊法及隐式梯形法求解模型的流程图，如下图所示：



该部分由 power_system_stability.py 文件内程序实现。

（1） 接口函数

- newton_laphson(Functions, init_values, e=0.00001)

牛顿拉夫逊法的接口函数，其参数如下：

·Functions 需要通过牛拉法迭代的形如 f(x,y,...,z)=0 的一组代数方程

·init_values 初始值的字典

·e 前后两次迭代的差值，所有方程组一次迭代前后差值最大者 ＜e

该函数将传入的方程组迭代收敛至给定误差后，返回迭代后值的字典。

- hiding_trapezium(derivative, n=50, limit=(0,1), e=0.0001)

隐式梯形法的接口函数，其参数如下：

·derivative 含所有方程（包括微分方程和代数方程）及相应初值的字典

·n 绘制点个数

·limit 绘图定义域

·e 传入牛拉法中的迭代精度

该函数将微分方程先变形为代数方程，并与代数方程一同传入牛拉法函数迭代。

- Pe(E=None, Delta=None, Z_value=None, Alpha=None)

·E 算得得三台同步电机电势向量

·Delta 三台电机功角 sympy 符号

·Z_value 每一次系统状态改变后，相应的发电机组输入、转移阻抗模值矩阵。

·Alpha 每一次系统状态改变后，相应的弧度制阻抗角矩阵

返回值为三台电机的电磁功率向量。

（2） 程序编写思路

A. 隐式梯形法

a. 生成定义域内对应步长的 numpy 序列；

b. 将所有方程中属于微分方程的定义转换为隐式梯形法对应的代数方程；

c. 利用给定初始值生成每一时间因变量的 numpy 序列；

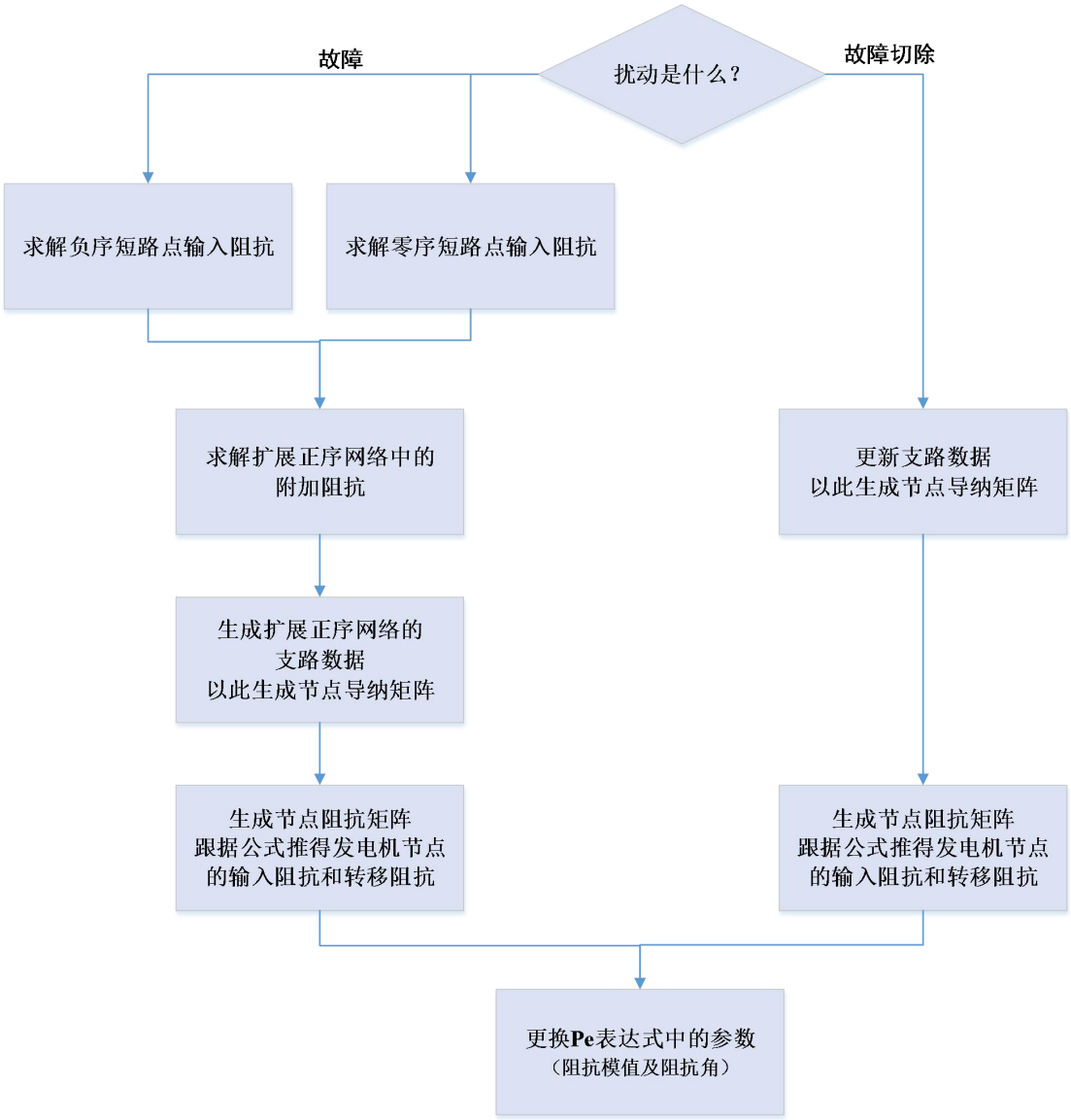
d. 将每一次牛拉法迭代后的初值和代数方程一同传入牛拉法迭代，产生下一时间点数据，修改因变量序列对应时间索引的值。

B. 牛顿拉夫逊法

为了发挥 numpy、sympy 数值计算的最大优势，使用 sympy 自动求导功能，并且成功尝试使用 sy.diff 函数，让代数 n 个方程组成的向量对由 n 个自变量组成的向量求导，从而能一步产生 n*n 的求导后矩阵，大大简化了程序编写，具体过程如下：

a. 初始化雅可比矩阵及各参数；
b. 进入迭代过程；
c. 将每一次迭代初值代入代数方程；
d. 将每一次迭代初值代入雅可比矩阵，求解代数方程组计算因变量变化量Δ；
e. 利用Δ修改每一次迭代值，重复步骤 3；
f. 判断两次的代数方程差是否小于给定误差，若小于则返回最终迭代值，否则回到过程 3。

2. 计算 P_e 表达式参数流程：



该部分由 Y.py 和 impedance.py 文件中的程序实现。具体程序实现如下所示。

（1） 定义通用的数据格式

本程序定义了通用的数据格式，使得对于任何满足格式的案例，不需要修改程序代码就能得到计算结果。

本程序定义了三个数据文件：描述节点参数的 Node_data.npy、描述支路参数的 branch_data.npy、描述变压器参数的 trans_data.npy。它们定义的皆为 numpy 的 array 格式，例题的数据保存在附件中的 data 文件夹中。具体定义格式分别如下所示：

Node_data.npy 的数据格式：

列数	含义
1	节点序号

2	节点类型（短路节点记为 0,负荷节点或联络节点记为 1,电源节点记为 2）
3	电源节点电动势 E'
4	电源节点的 X_d'（正序）
5	电源节点 X_2（负序）
6	电源节点 X_0（零序）

附：

（1）列 3-6：不是电源节点的节点标为 0

（2）本题中，节点 0：地；节点 1：A；节点 2：B；节点 3：C；节点 4：D；节点 5：E1；节点 6：E2；节点 7：E3。

（3）电源节点定义是在 X_d'外。

branch_data.npy 的数据格式：

列数	含义
1	起始节点编号
2	终止节点编号
3	切除故障前支路上等价的电阻
4	切除故障前支路上等价的电抗
5	切除故障后支路上等价的电阻
6	切除故障后支路上等价的电抗
7	零序电路中的电抗

附：

（1）若终止节点为地（负荷），则将其置零。

（2）将短路节点定为节点 1。

（3）原始数据中不需要加入 X_d'。

trans_data.npy 的数据格式：

列数	含义
1	变压器两端节点中靠近短路点的节点序号

2	变压器两端节点中远离短路点的节点序号
3	变压器在列 1 对应一端的绕组接法
4	变压器在列 2 对应一端的绕组接法 （YN：0,　Y:1,　D:2）

（2） 生成节点导纳矩阵

利用支路追加法，根据支路参数的数据可生成节点导纳矩阵。

该部分程序在 Y.py 文件中。文件中定义了两个函数，Y_generation( )和 Y_generation_ZS( )，前者可以生成正负序的节点导纳矩阵，后者可生成零序的节点导纳矩阵。由于正负序电路中除 X_d’外的参数是相同的，而零序电路中线路电抗与正负序电路不同，故设计两个函数分别计算。

（3） 计算附加阻抗

根据正序等效定则，不对称故障分析时，可将负序、零序电路的效果等效为附加阻抗叠加在正序电路上，使得分析不对称故障时只需分析扩展正序电路。

I. 负序、零序短路点处的输入阻抗

向 branch_data 中增加发电机节点的 X_{d2}’及改变负荷等效阻抗值为负序电抗值，即能生成新的 temp_branch。将 temp_branch 传入节点导纳矩阵生成函数 Y_generation( )中，能够生成负序电路的节点导纳矩阵 Y_2。将 Y_2 求逆即可得到节点阻抗矩阵 Z_2，从中便能读出负序电路中短路点处的输入阻抗 Z_{2ff} 。

零序电路的求解方法类似，不同之处为：零序电路需要考虑零序电流的流通情况，此时则需要变压器参数的数据 trans_data。对于两侧不都是星型接地的变压器，零序电流不能流经其下游电路。并且，零序电路中不考虑负荷。

程序中，通过 NSI( )和 ZSI( )分别计算得到负序、零序短路点处的输入阻抗。

II. 附加阻抗的生成

根据正序等效定则，对于不同的不对称故障类型，附加阻抗的计算方式不同。本程序由函数 X_star( )生成附加阻抗 $X_{\triangle}$，其第四个输入参数为短路类型，输入不同的数值则能计算得到不同的。其定义如下所示：

短路类型	对应的第四个输入参数	附加阻抗的计算
三相短路	1	0
两相接地短路	2	X_{2ff} // X_{0ff}
两相短路	3	X_{2ff}
单相短路	4	X_{2ff} + X_{0ff}

（4） 计算发电机节点的输入阻抗和之间的转移阻抗

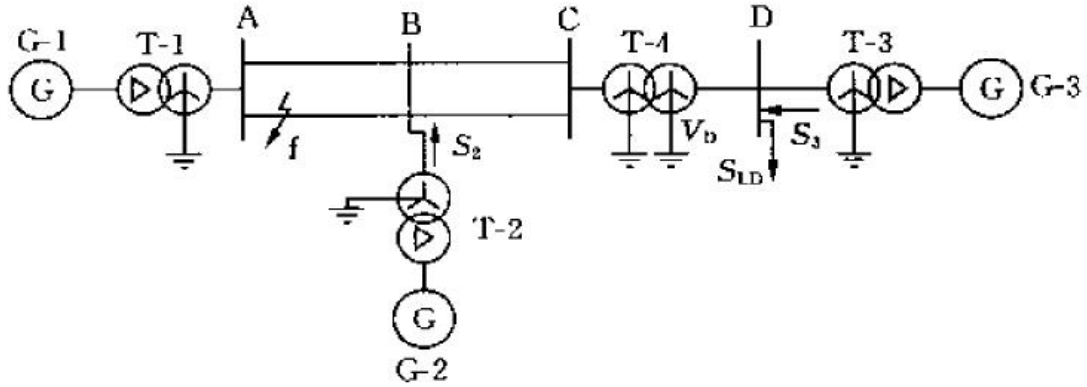
此步需要计算出节点阻抗矩阵，并由此得到输入阻抗和转移阻抗。生成节点阻抗矩阵前，首先必须得到对应的正确的支路数据。

本程序通过函数 branch_data_for_imp_SC( )，向 branch_data 中增加发电机节点的 X_{d2}'及短路节点处的 $X_{\triangle}$，生成故障发生时的新支路数据。利用函数 Impedance( )调用新支路数据，生成发电机节点的输入阻抗和之间的转移阻抗。输出结果为 m×m 矩阵 Z，对角元为输入阻抗，非对角元为转移阻抗（m 为发电机节点的个数）。

通过函数 branch_data_for_imp_afterCut( )，向 branch_data 中增加发电机节点的 X_{d2}'并更新切除故障处附近的支路数据，生成故障切除后的新支路数据。类似上段做法，生成故障切除后的发电机节点输入阻抗和转移阻抗组成的 m×m 矩阵。

通过函数 Z_value( )和 alpha( )，可以生成上述矩阵 Z 各元素的模值和阻抗角，将其代入模型的 P_e 表达式中。

四、程序求解的算例



$G-1:x'_d=0.1,\ x_2=0.1,T_J=10s$
$G-2:x'_d=0.15,x_2=0.15,T_J=7s$
$G-3:x'_d=0.06,x_2=0.06,T_J=15s$

变压器电抗：$x_{T1}=0.08$，$x_{T2}=0.1$，$x_{T3}=0.04$，$x_{T4}=0.05$；

线路电抗：AB 段双回$x_{L1}=0.2$，$x_{L0}=3.5x_{L1}$；BC 段双回$x_{L1}=0.1$，$x_{L0}=3.5x_{L1}$。

系统的初始状态：$V_{D0}=1.0$，$S_{LD0}=5.5+j1.25$，$S_{20}=1.0+j0.5$，$S_{30}=3.0+j0.8$。

扰动事件描述：在线路 AB 段首端 f 点发生两相短路接地，经 0.1s 切除故障线路。

五、程序运行结果及分析

1. 故障各阶段仿真时间的分配

仿真过程中 0-0.05s 为稳定状态，0.05sf 点发生两相接地短路，故障持续至 0.15s，0.15s 后故障切除。

2. 仿真 0.5s 后各状态量，及同步电机相对功角变化曲线图

由曲线图 1，可得出以下结论：

（1） 由系统中各同步电机相对功角变化，可知该系统在电机第一摆能保持稳定，而未发散。

（2） 三台机组中，仅有电机一在故障切除，即 0.15s 左右速度达到第一摆动最大，电机二、电机三分别在 0.44s 及 0.5s 后才至第一摆动最大值。

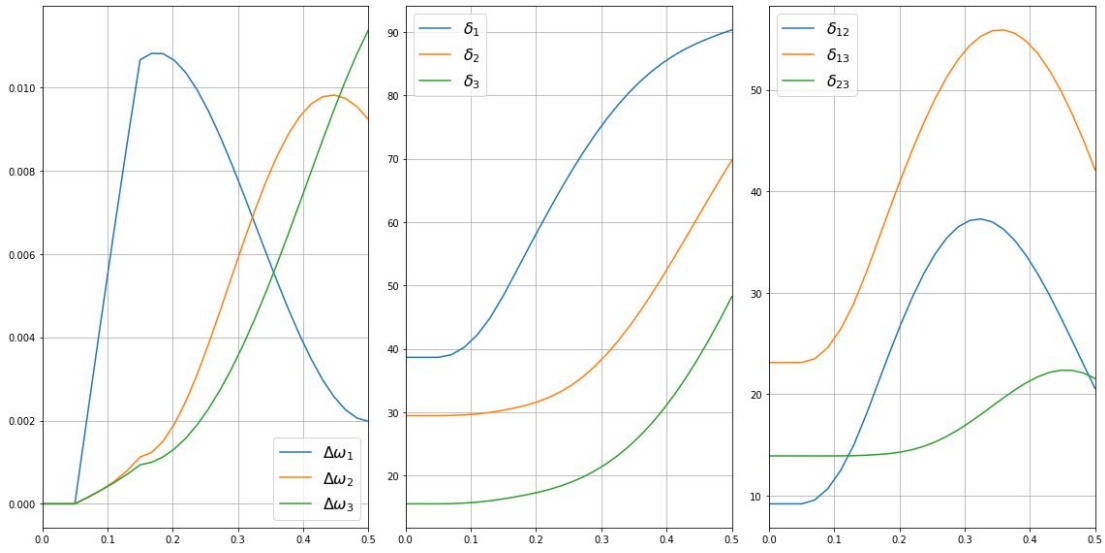


图 1 仿真 0.5s 同步电机各状态量及相对功角变化

3. 仿真 3s 后各状态量，及同步电机相对功角变化曲线图

为了探究各台机组在故障切除后最终能否达到相对稳定，讲仿真时间延长至 3s，从图 2 有如下新的发现：

（1） 三台同步电机在故障切除后都无法达到相对稳定状态，而是在上下反复波动。原因是模型中没有考虑阻尼项导致摆动无法衰减。

（2） 三台机组的转速是波动上升的，说明转速变化同时具有波动分量和升高分量。

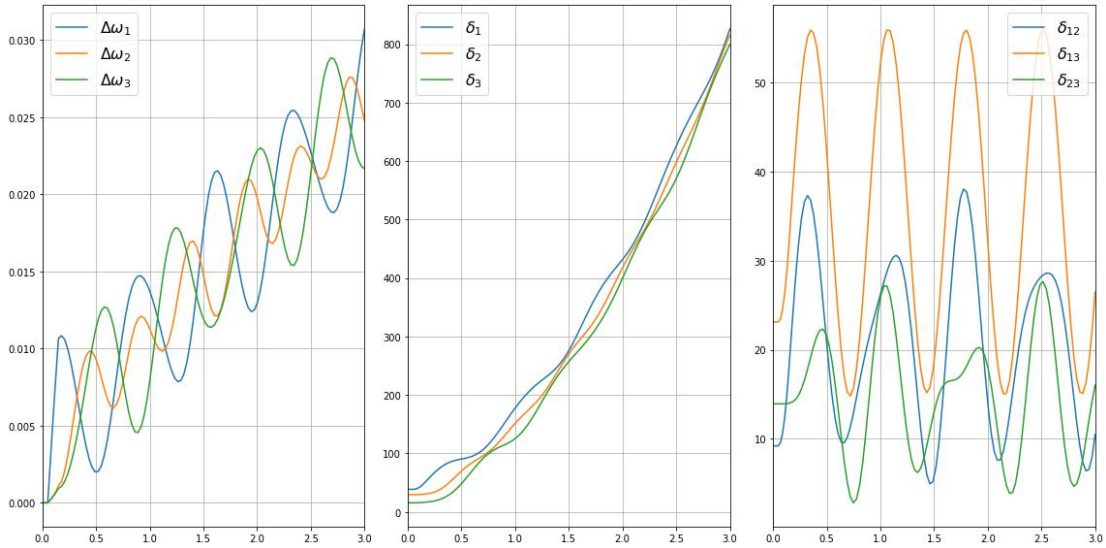


图 2 仿真 3s 同步电机各状态量及相对功角变化

六、程序创新点

1. 本程序定义了通用的数据格式，使得对于任何满足格式的案例，不需要修改程序代码就能得到计算结果。

2. 本程序利用了 sympy 符号库的自动求导功能与优化的 latex 输出，免去了求解雅可比矩阵时繁重的求导任务。

3. 本程序编写了许多通用的函数，如：牛顿-拉夫逊法、生成节点导纳矩阵、求解输入阻抗及转移阻抗等，日后针对其他问题均能调用这些函数进行求解。