**电力系统分析程序2020总结**

# 总体进展

## 程序目标

本程序使用python语言编写，以服务电力系统分析课程教学为目的。程序应支持电力系统潮流计算、短路分析、暂态计算三个基本内容。所用的电网模型基础数据格式按照PSSE软件格式，考虑到程序简化性，会对模型做出相应的简化，具体目标如表 1-1所示。

表 1-1 程序设计目标

|  |  |
| --- | --- |
| 内容 | 要求 |
| 电网模型 | 一条特高压双回交流线，一条常规直流，一处风机、一处光伏，一台水轮机组，电网为典型受端电网 |
| 电网数据 | PSSE格式（.raw，.seq，.dyr），对数据文件进行解析读取 |
| 网络模型 | 节点导纳阵，节点电纳阵；正序、负序、零序导纳阵；暂态导纳阵 |
| 潮流计算 | 牛拉法，导出雅克比矩阵；PQ法；直流潮流法 |
| 短路分析 | 不对称短路计算；计算短路点等值阻抗；计算短路点电压电流；求全网的电压电流分布 |
| 暂态计算 | 支持励磁模型、调速模型，直流模型，风机、光伏模型；改进欧拉法计算状态变量，迭代法求取网络方程；故障为节点发生三相短路 |

## 已完成内容

2020年，程序取得了很大的进展，在经过反复修改调试后，目前程序的版本已经完成原计划目标的60%，总结在表 1-2中。为了程序调试，我们也搭建了IEEE9节点网络模型，作为对比。

表 1-2 程序完成内容

|  |  |
| --- | --- |
| 内容 | 已完成 |
| 电网模型 | 潮流数据和序网数据搭建完毕。IEEE9节点测试模型潮流、序网、动态模型搭建完毕 |
| 电网数据 | 完成了.raw、.seq、.dyr数据导入程序，设计数据的存储格式 |
| 网络模型 | 完成节点导纳阵、节点电纳阵计算程序；完成三序节点导纳阵计算程序；完成暂态导纳阵计算程序 |
| 潮流计算 | 完成交直流系统的潮流计算程序，直流系统采取整流侧定功率逆变侧定电压控制方式 |
| 短路分析 | 完成正序负序零序等值阻抗计算程序；完成电网故障后序电压电流计算程序。 |
| 暂态计算 | 完成IEEE9节点三相短路暂态计算程序，包含了励磁系统和调速系统模型；完成网络方程求解和微分方程求解 |

由于程序内容较多，故整体的框架相对于前期和中期已有了较大调整，该部分会在第二章详细介绍，最终把所有的程序都进行了封装，便于调试和使用。

## 目前存在的问题

程序在设计过程中，遇到了较大的问题，有些已经解决，有些未解决，在此将未解决的问题汇总在表 1-3中。

表 1-3 目前存在的问题

|  |  |
| --- | --- |
| 内容 | 存在的问题 |
| 电网模型 | 搭建直流、风机、光伏动态模型未成功 |
| 电网数据 | 尚不能解析含直流、风机、光伏的动态数据文件 |
| 网络模型 | 负序网络导纳阵有问题，尚未找到 |
| 潮流计算 | 直流控制方式未全加入 |
| 短路分析 | 负序等值阻抗计算有问题 |
| 暂态计算 | 对含直流、风机、光伏的电网尚不支持 |

综上来看，**电力系统分析程序完成了传统交流电网的潮流计算、短路分析、暂态计算，对包含直流和新能源的现代电网只支持潮流计算和短路分析，尚不支持暂态计算**。

**个人想法：由于直流模型和新能源模型过于复杂，个人希望将直流和新能源梦在暂态分析任务中去除，以传统IEEE9节点系统暂态计算作为讲解任务即可。潮流计算和短路分析中可以有直流。**

# 程序架构

原设计中所有的模块都放在了一个目录下，在程序少时维护容易，然而随着暂态分析程序加入，原有的架构已不不适合维护与编写，因此程序的数据存储、模块函数等都进行了重新设计。功能函数中不再直接访问数据库中的数据，而采取API对数据库中的数据进行获取和修改。数据库中的数据存储格式采用了字典、dataFrame等数据结构，充分发挥python的强大灵活性。

本节介绍程序中的API，便于阅读程序。3，4，5，6节主要介绍功能函数，不展示具体代码，具体代码需要查阅程序。

## 程序框架

程序仍放置在src目录下，分为了：数据库，数据导入，数据获取与修改，网络导纳阵，潮流计算，短路分析，暂态计算共7部分，每个部分下对应了各自的模块，总框架如表 2-1所示。

表 2-1 程序总框架

|  |  |  |
| --- | --- | --- |
| 功能包 | 包内包含的模块 | 模块功能 |
| 数据库database | \_\_init\_\_.py | 存储读取的数据 |
| 数据导入  data\_imexporter | parse\_psse\_pf.py | 解析潮流数据 |
| parse\_psse\_sq.py | 解析序网数据 |
| parse\_psse\_dm.py | 解析动态模型数据 |
| 数据获取与修改  apis | apis\_device.py | 潮流设备数据 |
| apis\_basic.py | 基础的数学函数 |
| apis\_system.py | 全局系统参数 |
| apis\_sequence.py | 序网模型数据 |
| apis\_model.py | 动态模型数据 |
| apis\_dynamic.py | 暂态仿真状态变量 |
| 网络导纳阵  network | net\_pf.py | 潮流导纳阵 |
| net\_sq.py | 序网导纳阵 |
| net\_dm.py | 暂态仿真导纳阵 |
| net\_result.py | 保存导纳阵 |
| 潮流计算  powerflow | branch.py | 初始化与更新结果 |
| method\_NR.py | 牛拉法 |
| method\_PQ.py | PQ法 |
| method\_DC.py | 直流潮流法 |
| hvdc.py | 直流相关 |
| powerflow\_result.py | 获取潮流结果 |
| 短路分析fault | fault\_analysis.py | 短路分析 |
| analysis\_result.py | 分析结果 |
| 暂态计算  dynamic | simulation.py | 仿真主干 |
| init\_simulation.py | 初始化状态变量 |
| network\_solution.py | 网络方程求解 |
| disturbance.py | 扰动设置 |
| models | 微分方程求解 |

## 数据存储

数据存储主要使用了pandas的DataFrame类型和字典dict类型，方便获取修改数据。原有设计将每个原件设计为一个类，最大的难处是不方便索引元件对象，这在暂态分析部分非常突出，因此重新设计了数据存储形式。全部的变量名如表 2-2所示，每个变量的下属参数不再介绍，可直接查看database/\_\_init\_\_.py。

表 2-2 数据库变量

|  |  |  |
| --- | --- | --- |
| 变量名 | 类型 | 功能 |
| Base | 字典dict | 存储系统全局参数 |
| YMatrix | 字典dict | 存储计算得到的各类型导纳阵 |
| PowFlowPar | 字典dict | 存储潮流计算参数 |
| DynSimPar | 字典dict | 存储暂态仿真参数 |
| BusData | DataFrame | 存储母线潮流数据 |
| LoadData | DataFrame | 存储负荷潮流数据 |
| ShuntData | DataFrame | 存储无功补偿潮流数据 |
| GenData | DataFrame | 存储发电机潮流数据 |
| LineData | DataFrame | 存储线路潮流数据 |
| TransData | DataFrame | 存储变压器潮流数据 |
| HvdcData | DataFrame | 存储直流线路潮流数据 |
| WtGenData | DataFrame | 存储风机潮流数据 |
| PvUnitData | DataFrame | 存储光伏潮流数据 |
| BusSqData | DataFrame | 存储母线序网数据 |
| GenSqData | DataFrame | 存储发电机序网数据 |
| WtGenSqData | DataFrame | 存储风机序网数据 |
| PvUnitSqData | DataFrame | 存储光伏序网数据 |
| LoadSqData | DataFrame | 存储负荷序网数据 |
| LineSqData | DataFrame | 存储线路序网数据 |
| ShuntSqData | DataFrame | 存储无功补偿序网数据 |
| TransSqData | DataFrame | 存储变压器序网数据 |
| SyncGenMd | 字典dict | 存储发电机模型数据 |
| ExciterMd | 字典dict | 存储励磁系统模型数据 |
| TurGovMd | 字典dict | 存储调速系统模型数据 |
| BusStateVar | 字典dict | 存储暂态过程母线状态变量 |
| GenStateVar | 字典dict | 存储暂态过程发电机状态变量 |
| ExcStateVar | 字典dict | 存储暂态过程励磁系统状态变量 |
| TurStateVar | 字典dict | 存储暂态过程调速系统状态变量 |

在这里需要说明的是，PSS/E软件是将序网数据和潮流数据合并在一起存储的，本程序将他们分开存储了。

## 数据获取与修改

在功能程序编写运行时，通过apis包下的各种API函数从数据库中获取对应的数据或者修改部分数据，用到的API在表 2-3至表 2-8中进行介绍。由于API众多，在此不介绍API函数的参数和返回值，只介绍API的功能。

1. 潮流模型参数操作，位于apis\_device.py中，如表 2-3所示

表 2-3 潮流模型API

|  |  |
| --- | --- |
| API | 功能 |
| add\_device | 在数据库中增加一个设备 |
| get\_all\_devices | 获取所有的设备编号 |
| get\_device\_data | 获取设备参数 |
| set\_device\_data | 修改设备参数 |

1. 序网模型参数操作，位于apis\_sequence.py中，如表 2-4所示。

表 2-4 序网模型API

|  |  |
| --- | --- |
| API | 功能 |
| add\_device\_sequence\_model | 增加一个序网设备模型 |
| get\_device\_sequence\_data | 获取设备序网参数 |
| set\_device\_sequence\_data | 修改设备的序网参数 |

1. 动态模型参数操作，位于apis\_model.py中，如表 2-5所示。

表 2-5 动态模型API

|  |  |
| --- | --- |
| API | 功能 |
| add\_generator\_related\_model | 增加发电机相连的模型 |
| get\_generator\_related\_model\_data | 获取发电机相连的模型 |
| set\_generator\_related\_model\_data | 修改发电机相连的模型 |

1. 动态模型状态变量操作，位于apis\_dynamic.py中，如表 2-6所示。这部分API用于暂态仿真程序，并不对外暴露。

表 2-6 动态模型状态变量API

|  |  |
| --- | --- |
| API | 功能 |
| add\_model\_state\_variable | 增加模型状态变量 |
| get\_generator\_state\_data | 获取发电机模型状态变量 |
| set\_generator\_state\_data | 修改发电机模型状态变量 |
| get\_exciter\_state\_data | 获取励磁系统状态变量 |
| set\_exciter\_state\_data | 修改励磁系统状态变量 |
| get\_turbine\_state\_data | 获取调速系统状态变量 |
| set\_turbine\_state\_data | 修改励磁系统状态变量 |
| get\_bus\_state\_data | 获取母线状态变量 |
| set\_bus\_state\_data | 修改母线状态变量 |

1. 系统变量相关，位于apis\_system.py中，如表 2-7所示。对外的只有get\_simulator\_parameter、set\_simulator\_parameter、prepare\_dynamic\_output\_meter三个API，其余只在内部程序使用。

表 2-7 系统仿真变量API

|  |  |
| --- | --- |
| API | 功能 |
| get\_simulator\_parameter | 获取仿真参数 |
| set\_simulator\_parameter | 修改仿真参数 |
| get\_bus\_num\_after\_renumber | 获取编号后的母线编号 |
| get\_bus\_num\_before\_renumber | 获取编号前的母线编号 |
| get\_system\_bus\_number | 获取系统某类型母线数目 |
| get\_system\_base\_data | 获取系统基准数据 |
| set\_system\_base\_data | 修改系统基准数据 |
| get\_system\_Y\_network\_matrix | 获取节点导纳阵 |
| set\_system\_Y\_network\_matrix | 修改节点导纳阵 |
| prepare\_dynamic\_output\_meter | 设置暂态仿真输出通道 |
| get\_all\_dynamic\_output\_meters | 获取所有的暂态仿真输出通道 |

1. 数学计算相关，位于apis\_basic.py中，如表 2-8所示。这里面API均不对外暴露。

表 2-8 数学计算API

|  |  |
| --- | --- |
| API | 功能 |
| build\_complex\_value | 建立复数向量 |
| convert\_deg\_to\_rad | 角度转弧度 |
| convert\_rad\_to\_deg | 弧度转角度 |
| get\_complex\_phase\_angle | 获取复数向量的相角 |
| convert\_xy\_to\_dq | 将xy坐标系的值转化为dq坐标系的值 |
| composite\_three\_phase\_vector | 由三序分量组合成三相分量 |

## 程序封装

目前将所有的功能程序封装在最外层的PSATool.py中，程序在使用或者测试时直接使用from PSATool import PAST即可。PSATool中的方法查看PASATool.py文件即可，在这不做介绍。一个简单的程序仿真示例如下，包含了数据导入，数据获取，潮流计算、暂态计算、短路分析。

|  |
| --- |
| from PSATool import PAST  raw\_file = '.\data\IEEE9-2阶.raw'  seq\_file = '.\data\IEEE9.seq'  dyr\_file = '.\data\IEEE9-2阶.dyr'  simulator = PAST()  simulator.load\_simulation\_data(raw\_file, 'powerflow')  buses = simulator.get\_all\_devices('BUS')  loads = simulator.get\_all\_devices('LOAD')  generators = simulator.get\_all\_devices('GENERATOR')  simulator.build\_network\_Y\_matrix('basic')  simulator.save\_network\_Y\_matrix('./test\_result/IEEE9\_Y\_mat.csv', 'basic')  simulator.solve\_powerflow('NR')  simulator.save\_powerflow\_result('./test\_result/IEEE9\_powerflow\_result.csv')  simulator.load\_simulation\_data(dyr\_file, 'dynamic')  simulator.build\_network\_Y\_matrix('dynamic')  simulator.save\_network\_Y\_matrix('./test\_result/IEEE9\_Y\_dynamic\_mat.csv', 'dynamic')  for generator in generators:  simulator.prepare\_dynamic\_output\_meter(generator, 'GEN', 'ROTOR ANGLE IN DEG')  simulator.set\_simulator\_parameter('dynamic', 'output\_file', 'IEEE9\_dynamic\_result.csv')  simulator.start\_dynamic\_simulation()  simulator.run\_dynamic\_simulation\_to\_time(0.1)  bus = 7  Yf = -1j\*1e6  simulator.set\_bus\_fault(bus, Yf)  simulator.run\_dynamic\_simulation\_to\_time(0.2)  simulator.clear\_bus\_fault(bus, Yf)  simulator.run\_dynamic\_simulation\_to\_time(2.0)  simulator.load\_simulation\_data(seq\_file, 'sequence')  simulator.build\_network\_Y\_matrix('positive')  simulator.save\_network\_Y\_matrix('./test\_result/IEEE9\_Y\_positive\_mat.csv', 'positive')  simulator.build\_network\_Y\_matrix('negative')  simulator.build\_network\_Y\_matrix('zero')  simulator.solve\_bus\_asymmetry\_fault(4, 'single phase grounding', 0.0)  simulator.save\_fault\_analysis\_result('./test\_result/IEEE9\_fault\_analysis.csv') |

程序 2-1 仿真示例程序

# 网络导纳阵

潮流计算、短路分析、暂态计算均需要建立系统的导纳矩阵，导纳阵有关的程序均位于network包下。build\_network\_Y\_matrix是建立节点导纳阵最上层的函数，位于\_\_init\_\_.py中。

## 潮流计算网络导纳阵

潮流计算需要的导纳阵主要分为牛拉法的常规导纳阵和PQ法的节点电纳阵两种，程序位于net\_pf.py中，包含4个函数。

1. **calculate\_network\_Y\_matrix**

|  |
| --- |
| 功能：计算系统节点导纳矩阵。  参数：① c，bool型，True为考虑支路充电电容和变压器非标准变比。② r，bool型，True为考虑支路电阻。  返回：Y\_mat，array类型，节点导纳阵。 |

1. **build\_line\_equivalent\_circuit**

|  |
| --- |
| 功能：建立线路的等值电路。  参数：① line，tuple型，线路索引，（IBUS，JBUS，CKT）。② c，bool型，True为考虑支路充电电容和变压器非标准变比。③ r，bool型，True为考虑支路电阻。  返回：① Yij，互导纳。② Yi，i侧自导纳。③ Yj，j侧自导纳。 |

1. **build\_transformer2\_equivalent\_circuit**

|  |
| --- |
| 功能：建立双绕组变压器的等值电路。  参数：① transformer，tuple型，变压器索引，（IBUS，JBUS，KBUS）。② c，bool型，True为考虑支路充电电容和变压器非标准变比。③ r，bool型，True为考虑支路电阻。  返回：① Yij，互导纳。② Yi，i侧自导纳。③ Yj，j侧自导纳。 |

1. **build\_transformer3\_equivalent\_circuit**

|  |
| --- |
| 功能：建立三绕组变压器的等值电路。  参数：① transformer，tuple型，变压器索引，（IBUS，JBUS，KBUS）。② c，bool型，True为考虑支路充电电容和变压器非标准变比。③ r，bool型，True为考虑支路电阻。  返回：trans3\_Y\_mat，array，三绕组变压器的节点导纳阵。 |

在计算PQ法需要的节点电纳阵时，B1矩阵忽略之路充电电容和变压器非标准变比，置参数c=False，B2矩阵忽略支路电阻，置参数r=False，即可计算相应的节点电纳阵。

## 短路分析的序网导纳阵

序网到那真程序位于net\_sq.py中。正序网络建立时采用发电机次暂态电抗，发电机星形接地；负荷转化为导纳，负荷也接地；直流转化为负荷；正序和负序网在原导纳阵修改即可，零序导纳阵需要新建立。在计算零序导纳阵时，忽略了线路的零序互阻抗。

1. **calculate\_sequence\_Y\_matrix**

|  |
| --- |
| 功能：计算序网导纳阵  参数：par\_type，str，导纳阵类型，正序、负序、零序。  返回：Y\_mat，array，序网导纳阵。 |

1. **calculate\_positive\_Y\_matrix**

|  |
| --- |
| 功能：计算正序导纳阵  参数：无。  返回：Y\_mat，array，正序导纳阵。  部分示例代码：  loads = apis.get\_all\_devices('LOAD') # add load  for load in loads:  i = apis\_system.get\_bus\_num\_after\_renumber(load)  VM = apis.get\_device\_data(load, 'BUS', 'VM')  PL = apis.get\_device\_data(load, 'LOAD', 'PL')  QL = apis.get\_device\_data(load, 'LOAD', 'QL')  Y\_mat[i, i] = Y\_mat[i, i] + (PL - 1j \* QL) / SBASE / VM\*\*2 |

1. **calculate\_negative\_Y\_matrix**

|  |
| --- |
| 功能：计算负序导纳阵  参数：无  返回：Y\_mat，array，负序导纳阵  部分示例代码：  loads = apis.get\_all\_devices('LOAD')  for load in loads:  i = apis\_system.get\_bus\_num\_after\_renumber(load)  VM = apis.get\_device\_data(load, 'BUS', 'VM')  PNEG = apis.get\_device\_sequence\_data(load, 'LOAD', 'PNEG')  QNEG = apis.get\_device\_sequence\_data(load, 'LOAD', 'QNEG')  Y\_mat[i, i] = Y\_mat[i, i] + (PNEG - 1j \* QNEG) / SBASE / VM\*\*2 |

**注明：在计算短路点负序戴维南阻抗时，计算结果有较大的出入，经过本人的分析，目前定位是负序导纳阵计算有问题，但是该问题尚未找到。**

1. **calculate\_zero\_Y\_matrix**

|  |
| --- |
| 功能：计算零序导纳阵  参数：无  返回：Y\_mat，array，零序导纳阵。  部分示例代码：  SBASE = apis\_system.get\_system\_base\_data('SBASE')  generators = apis.get\_all\_devices('GENERATOR')  for generator in generators:  i = apis\_system.get\_bus\_num\_after\_renumber(generator)  MBASE = apis.get\_device\_data(generator, 'GENERATOR', 'MBASE')  ZR0 = apis.get\_device\_sequence\_data(generator, 'GENERATOR', 'ZR0')  ZX0DV = apis.get\_device\_sequence\_data(generator, 'GENERATOR', 'ZX0DV')  ZRG = apis.get\_device\_sequence\_data(generator, 'GENERATOR', 'ZRG')  ZXG = apis.get\_device\_sequence\_data(generator, 'GENERATOR', 'ZXG')  gen\_z = (ZR0 + 1j \* ZX0DV + 3 \* (ZRG + 1j \* ZXG)) \* SBASE / MBASE  Y\_mat[i, i] = Y\_mat[i, i] + 1.0 / gen\_z |

1. **build\_transformer2\_zero\_equivalent\_circuit**

|  |
| --- |
| 功能：建立双绕组变压器零序等值电路  参数：transformer，tuple，双绕组变压器，（IBUS，JBUS，KBUS）  返回：① Yij，i侧到j侧互导纳。② Yi，i侧自导纳。③ Yj，j侧自导纳  部分示例代码：  if CC == 1: # YNyn  Zt = WINDV2 \*\* 2 \* (R01 + 1j \* X01)  k = WINDV1 / WINDV2  Yij = 1 / (k \* Zt)  Yj = (k - 1) / (k \* Zt)  Yi = (1 - k) / (k \*\* 2 \* Zt)  elif CC == 2: # YNd  Zt = R01 + 1j \* X01 + 3 \* (RG1 + 1j \* XG1)  k = WINDV1  Yi = 1 / (k \* Zt)  Yij = 0.0  Yj = 0.0 |

双绕组变压器目前支持的接线方式有YNyn，YNd，Dyn，Yy，Yd，Dy，Dd，Yyn，YNy。

1. **build\_transformer3\_zero\_equivalent\_circuit**

|  |
| --- |
| 功能：建立三绕组变压器零序等值电路  参数：transformer，tuple，三绕组变压器，（IBUS，JBUS，KBUS）。  返回：trans3\_mat，array，三绕组变压器零序节点导纳阵。  部分示例代码：  if CC == 1: # YNynyn  Z10 = R01 + 1j \* X01  Z20 = R02 + 1j \* X02  Z30 = R03 + 1j \* X03  Zg0 = 3 \* (RG1 + 1j \* XG1)  Yeq1, Yeq2, Yeq3 = 1 / (WINDV1 \* Z10), 1 / (WINDV2 \* Z20), 1 / (WINDV3 \* Z30)  temp = Yeq1 + Yeq2 + Yeq3 + 1 / Zg0  Y11 = Yeq1 - Yeq1 \*\* 2 / temp  Y12 = - Yeq1 \* Yeq2 / temp  Y13 = - Yeq1 \* Yeq3 / temp  Y21 = - Yeq1 \* Yeq2 / temp  Y22 = Yeq2 - Yeq2 \*\* 2 / temp  Y23 = - Yeq2 \* Yeq3 / temp  Y31 = - Yeq3 \* Yeq1 / temp  Y32 = - Yeq3 \* Yeq2 / temp  Y33 = Yeq3 - Yeq3 \*\* 2 / temp |

三绕组变压器目前支持的接线方式有YNynyn，YNynd，Dynd，Ddd, Ddy, Dyd, Dyy, Ydd, Ydy, Yyd，Yyy。

## 暂态计算导纳阵

暂态计算导纳阵程序位于net\_dm.py中，在原潮流计算导纳阵基础上，负荷转换为导纳，加入发电机内导纳。

1. **calculate\_dynamic\_Y\_matrix**

|  |
| --- |
| 功能：计算暂态导纳阵  参数：无  返回：Y\_mat，array，暂态计算导纳阵。 |

1. **calculate\_generator\_internal\_admittance**

|  |
| --- |
| 功能：计算发电机内导纳  参数：generator，int，发电机连接母线号。  返回：YGp，发电机内导纳  部分示例代码：  GMN = apis.get\_generator\_related\_model\_data(generator, 'GEN', 'GMN')  SBASE = apis\_system.get\_system\_base\_data('SBASE')  MBASE = apis.get\_device\_data(generator, 'GENERATOR', 'MBASE')  if GMN == 'GENCLS':  Xd = apis.get\_device\_data(generator, 'GENERATOR', 'ZX')  Xd = Xd \* SBASE / MBASE  YGp = 1.0 / (1j \* Xd) |

# 潮流计算

潮流计算的所有程序都位于powerflow包中，在潮流初始化时采用平启动计算。\_\_init\_\_.py中solve\_powerflow函数是最上层的求解函数。

|  |
| --- |
| def solve\_powerflow(method):  network.build\_network\_Y\_matrix('basic')  S, Um, Ua = init\_powerflow\_solution()  if method == 'NR':  S, Um, Ua = solve\_powerflow\_with\_NR\_method(S, Um, Ua)  update\_powerflow(S, Um, Ua)  elif method == 'PQ':  network.build\_network\_Y\_matrix('B1')  network.build\_network\_Y\_matrix('B2')  S, Um, Ua = solve\_powerflow\_with\_PQ\_method(S, Um, Ua)  update\_powerflow(S, Um, Ua)  elif method == 'DC':  network.build\_network\_Y\_matrix('B1')  network.build\_network\_Y\_matrix('B2')  solve\_powerflow\_with\_DC\_method(S, Um, Ua)  else:  pass  show\_powerflow\_result()  return |

init\_powerflow\_solution是平启动函数，用于返回系统节点注入复功率、节点电压幅值、电压相角。update\_powerflow是更新系统潮流，将求解后的潮流情况写入内存中。

在powerflow包下，branch.py中包含了平启动函数init\_powerflow\_solution、update\_powerflow、show\_powerflow\_result三个函数，powerflow\_result.py中包括了获取潮流结果相关的函数，如get\_bus\_result、get\_line\_result等，这些函数不在介绍，请直接查阅程序即可。

## 牛顿法

牛顿法的程序在method\_NR.py中，包含了求解主干函数、功率不平衡量计算函数、雅克比矩阵建立、节点电压和相角修正共4个函数。

1. **solve\_powerflow\_with\_NR\_method**

|  |
| --- |
| 功能：牛拉法求解潮流主函数。  参数：① S，array，求解前节点注入复功率。② Um，array，求解前节点电压幅值标幺值。③ Ua，array，求解前节点电压相角弧度值。  返回：① S，array，求解后节点注入复功率。② Um，array，求解后节点电压幅值标幺值。③ Ua，array，求解后节点电压相角弧度值。 |

1. **calculate\_power\_imbalance**

|  |
| --- |
| 功能：计算功率不平衡量。  参数：① S，array，节点注入复功率。② Um，array，节点电压幅值标幺值。③ Ua，array，节点电压相角弧度值。  返回：power\_err，array，节点功率不平衡量。 |

1. **build\_jacobian\_matrix**

|  |
| --- |
| 功能：计算雅克比矩阵。  参数：① Um，array，节点电压幅值标幺值。② Ua，array，节点电压相角弧度值。  返回：J\_mat，array，雅克比矩阵。 |

1. **correct\_node\_voltage\_and\_angle**

|  |
| --- |
| 功能：求解牛拉法的修正方程，修正电压和相角。  参数：① power\_err，array，节点功率不平衡量。② Um，array，节点电压幅值标幺值。③Ua，array，节点电压相角弧度值。④ J\_mat，array，雅克比矩阵  返回：① Um，array，节点电压幅值标幺值。② Ua，array，节点电压相角弧度值。 |

## PQ法

PQ法的程序在method\_PQ.py中，包含了求解主干函数、有功不平衡量计算函数、无功不平衡量计算函数，相角修正函数，电压修正函数共5个函数。

1. **solve\_powerflow\_with\_PQ\_method**

|  |
| --- |
| 功能：牛拉法求解潮流主函数。  参数：① S，array，求解前节点注入复功率。② Um，array，求解前节点电压幅值标幺值。③ Ua，array，求解前节点电压相角弧度值。  返回：① S，array，求解后节点注入复功率。② Um，array，求解后节点电压幅值标幺值。③ Ua，array，求解后节点电压相角弧度值。 |

1. **calculate\_act\_power\_imbalance**

|  |
| --- |
| 功能：计算有功功率不平衡量。  参数：① P，array，节点注入有功功率。② Um，array，节点电压幅值标幺值。③ Ua，array，节点电压相角弧度值。  返回：P\_err，array，节点有功功率不平衡量。 |

1. **correct\_node\_voltage\_angle**

|  |
| --- |
| 功能：修正节点电压相角  参数：① P\_err，array，有功不平衡量。② Um，array，节点电压幅值标幺值。③ Ua，array，节点电压相角弧度值。  返回：Ua，array，节点电压相角弧度值。 |

1. **calculate\_ract\_power\_imbalance**

|  |
| --- |
| 功能：计算无功功率不平衡量。  参数：① Q，array，节点注入无功功率。② Um，array，节点电压幅值标幺值。③ Ua，array，节点电压相角弧度值。  返回：Q\_err，array，节点无功功率不平衡量。 |

1. **correct\_node\_voltage**

|  |
| --- |
| 功能：修正节点电压  参数：① Q\_err，array，无功不平衡量。② Um，array，节点电压幅值标幺值。③ Ua，array，节点电压相角弧度值。  返回：Um，array，节点电压标幺值。 |

## 直流潮流法

直流潮流程序位于method\_DC.py中，相比牛拉法和PQ法，直流潮流不需要迭代，因此很简单，在此不进行阐述。

## 基础算例分析

以IEEE9节点系统为例，进行潮流计算分析。

牛拉法和PQ法的迭代结果如图 4-1和图 4-2所示，两者结果相等，潮流计算结果正确。

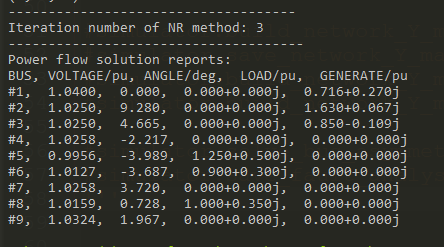


图 4-1 牛拉法计算结果

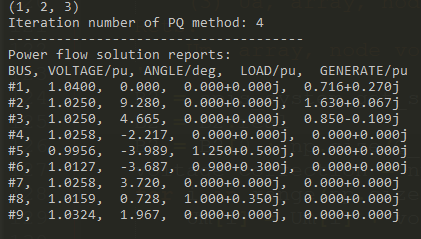


图 4-2 PQ法计算结果

## 交直流系统潮流计算说明

直流计算程序位于hvdc.py中，目前支持的直流控制方式为整流侧定功率逆变侧定电压控制。控制变量包括了换流变压器分接头和换向角，输入参数为两侧交流电压，相关程序查看hvdc.py即可。具体计算方法如下：

1. 由整流侧功率和逆变侧直流电压计算直流电流。
2. 换向角置为最小，计算换流变压器合理的分接头。
3. 由分接头反推计算换向角
4. 由换向角计算功率因数，进而计算吸收无功功率

# 不对称短路分析

短路分析的程序位于fault包下，fault\_analysis.py是计算的主干程序，analysis\_result.py处理计算结果

## 短路点戴维南等值阻抗计算

短路分析需要计算短路点的三序戴维南等值阻抗，理论上等值阻抗需要建立节点阻抗矩阵，然后取对应对角元素即可，但是对于电力系统来说，建立节点阻抗矩阵成本很“昂贵”，因此本程序获取等值阻抗是从导纳阵计算得到。

阻抗阵的对角元素等于在对应i节点注入单位电流，其余节点注入电流为0时节点i的电压，i节点和j节点的互阻抗则是j节点的电压，因此计算对应阻抗时，求I=YU方程计算电压即可。具体程序见，参数row为阻抗对应的行，column为阻抗对应的列，par\_type为导纳阵类型。

|  |
| --- |
| def get\_network\_Z\_matrix\_element(row, column, par\_type):  Y\_mat = apis\_system.get\_system\_Y\_network\_matrix(par\_type)  buses = apis.get\_all\_devices('BUS')  I = np.zeros(len(buses))  I[row] = 1.0  U = np.linalg.solve(Y\_mat, I)  value = U[column]  return value |

程序 5-1 等值阻抗计算程序

## 故障点和非故障点电压计算

首先计算短路点的三序电流，再由三序电流计算短路点的序电压和非短路点序电压，非短路点序电压计算时需要获取对应的互阻抗。计算程序如下。参数中，node是短路母线号，If为短路点三序电流元组。

|  |
| --- |
| def update\_bus\_sequence\_voltage(node, If):  ibus = apis\_system.get\_bus\_num\_after\_renumber(node)  (If1, If2, If0) = If  buses = apis.get\_all\_devices('BUS')  for bus in buses:  VM = apis.get\_device\_data(bus, 'BUS', 'VM')  VA = apis.get\_device\_data(bus, 'BUS', 'VA')  U0 = apis\_basic.build\_complex\_value(VM, VA)  i = apis\_system.get\_bus\_num\_after\_renumber(bus)  Z1 = get\_network\_Z\_matrix\_element(i, ibus, 'positive')  Z2 = get\_network\_Z\_matrix\_element(i, ibus, 'negative')  Z0 = get\_network\_Z\_matrix\_element(i, ibus, 'zero')  VP = U0 - Z1 \* If1  VN = - Z2 \* If2  VZ = - Z0 \* If0  apis.set\_device\_sequence\_data(bus, 'BUS', 'VP', VP)  apis.set\_device\_sequence\_data(bus, 'BUS', 'VN', VN)  apis.set\_device\_sequence\_data(bus, 'BUS', 'VZ', VZ)  return |

程序 5-2 序电压计算程序

计算出序电压后，三相电压可由序电压合成，支路的序电流也可由序电压和支路阻抗计算。对于三相电流，线路的已经支持计算，变压器涉及到具体接线方式，目前暂不支持。

## 算例分析

以IEEE9节点为例，计算母线4发生单相接地故障时各个节点三序电压，如图 5-1所示。**在这里指出，由于负序戴维南等值阻抗一直计算和PSSE结果不太一致，导致网络序电压计算有问题。目前我与昊良两人程序结果一致。**

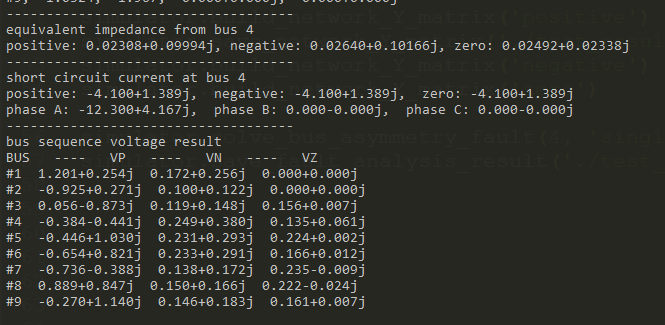


图 5-1 不对称短路计算结果

# 暂态计算

总则中表明，暂态计算精度要有六位小数的精度。首先要保证程序在没有扰动时稳定运行，各状态变量不能有大幅度变化，其次再让程序在电网发生扰动后运行，查看功角曲线是否合理。代数方程和微分方程是交替求解，代数方程采用迭代法求解，微分方程采用改进欧拉法求解。所有与暂态计算相关的程序均位于dynamic包中。

## 模型选择与简化

暂态计算的发电机模型选择PSSE模型中的GENROU模型、GENSAL模型、GENTRA模型、GENCLS模型，分别对应了教材中的六阶、五阶、三阶、二阶模型。励磁系统选择SEXS模型。调速系统选择IEEEG1和IEEEG3模型，分别对应汽轮调速系统和水轮调速系统。负荷模型直接采用恒导纳模型。模型具体参数请查阅PSSE手册。

目前发电机GENCLS模型和GENTRA模型已经调试完毕，GENSAL模型和GENROU模型还在调试中。

考虑模型的复杂性，对模型进行相应的简化，简化内容如表 6-1所示

表 6-1 模型简化内容

|  |  |  |
| --- | --- | --- |
| 模型 | 简化内容 | 简化参数 |
| GENROU、GENSAL、GENTRA、GENCLS | 忽略磁路饱和；忽略转子阻尼；忽略定子电阻 | D=0; S(1.0)=S(1.2)=0; Xl=0 |
| SEXS | 忽略超前滞后环节；忽略PSS稳定器影响 | TA=TB=0; VS=0 |
| IEEEG1（汽轮机） | 单输出功率；忽略超前滞后环节；减少后期的惯性环节 | T1=T2=0; T6=T7=0; K2=K4=K5=K6=K7=K8=0 |
| IEEEG3（水轮机） | 此模型无法简化 |  |

由简化后的模型，列出每个模型的微分方程，接下来以二阶模型为例，介绍程序编写。二阶模型只考虑发电机转子运动方程，网络方程不需要迭代，编程实现最简单。三阶及以上模型的程序都是在二阶模型程序框架上增加程序来实现功能的。

## 模型状态变量初始化

### 初始化原理

在模型状态变量初始化前，需要求解潮流，获取稳态时发电机端口电压和端口电流，初始化相关的程序位于init\_simulation.py中。

对于发电机GENCLS二阶模型来说，需要初始化的状态变量有发电机转子角、转速、发电机内电势、电磁功率和机械功率，下面这段程序是初始化程序。首先获取发电机端电压Vt，发电机出力PG、QG，发电机内电抗Xdp，并将他们转化为系统基准下的标幺值。其次，计算发电机注入电流It，并利用It计算出EQ，转子角即为EQ的相角。最后，初始化转速、功率等。

|  |
| --- |
| VM = apis.get\_device\_data(generator, 'BUS', 'VM')  VA = apis.get\_device\_data(generator, 'BUS', 'VA')  VA = apis\_basic.convert\_deg\_to\_rad(VA)  Vt = apis\_basic.build\_complex\_value(VM, VA)  PG = apis.get\_device\_data(generator, 'GENERATOR', 'PG')  QG = apis.get\_device\_data(generator, 'GENERATOR', 'QG')  SBASE = apis\_system.get\_system\_base\_data('SBASE')  It = (PG - 1j \* QG) / SBASE / Vt.conjugate()  Xq = apis.get\_device\_data(generator, 'GENERATOR', 'ZX') # Xq = Xdp  MBASE = apis.get\_device\_data(generator, 'GENERATOR', 'MBASE')  Xq = Xq \* SBASE / MBASE  EQ = Vt + 1j \* Xq \* It  delta = apis\_basic.get\_complex\_phase\_angle(EQ)  omega = 1.0  Pm = PG / SBASE  Pe = PG / SBASE |

程序 6-1 状态变量初始化

对于励磁模型或者调速模型，初始化时应将传递函数中的微分算子置0，由最终的输出回推初始状态变量和参考值，以SEXS励磁模型为例，如图。

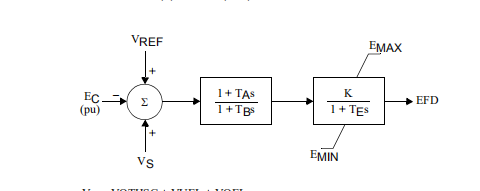


图 6-1 励磁模型SEXS传递函数框图

|  |
| --- |
| Efq = apis\_dynamic.get\_generator\_state\_data(generator, True, 'Efq')  EC = abs(apis\_dynamic.get\_bus\_state\_data(generator, True, 'Vt'))  K = apis.get\_generator\_related\_model\_data(generator, 'AVR', 'K')  Vref = Efq / K + EC  apis.set\_generator\_related\_model\_data(generator, 'AVR', 'Vref', Vref) |

程序 6-2 SEXS模型初始化程序

### 模块中的函数。

1. **init\_system\_state\_parameter**

|  |
| --- |
| 功能：初始化系统状态变量  参数：无  返回：无 |

1. **init\_generators\_model\_state\_parameter**

|  |
| --- |
| 功能：初始化发电机模型状态变量  参数：无  返回：无 |

1. **init\_exciters\_model\_state\_parameter**

|  |
| --- |
| 功能：初始化励磁模型状态变量  参数：无  返回：无 |

1. **init\_turbine\_governor\_model\_state\_parameter**

|  |
| --- |
| 功能：初始化调速模型状态变量  参数：无  返回：无 |

1. **init\_generator\_GENCLS\_model\_state\_parameter**

|  |
| --- |
| 功能：初始化发电机GENCLS二阶模型状态变量  参数：generator，int，发电机模型连接母线号  返回：value，dict，状态变量 |

1. **init\_generator\_GENTRA\_model\_state\_parameter**

|  |
| --- |
| 功能：初始化发电机GENTRA三阶模型状态变量  参数：generator，int，发电机模型连接母线号  返回：value，dict，状态变量 |

1. **init\_exciter\_SEXS\_model\_state\_parameter**

|  |
| --- |
| 功能：初始化励磁模型SEXS模型状态变量  参数：generator，int，发电机模型连接母线号  返回：value，dict，状态变量 |

1. **init\_turbine\_IEEEG1\_model\_state\_paremeter**

|  |
| --- |
| 功能：初始化调速模型IEEEG1模型状态变量  参数：generator，int，发电机模型连接母线号  返回：value，dict，状态变量 |

1. **init\_turbine\_IEEEG3\_model\_state\_paremeter**

|  |
| --- |
| 功能：初始化调速模型IEEEG3模型状态变量  参数：generator，int，发电机模型连接母线号  返回：value，dict，状态变量 |

## 网络方程求解

### 网络方程求解原理

暂态计算时对网络方程I=YU求解主要有直接法和迭代法，本文采取迭代法求解。与直接法相比，迭代法不需要扩展导纳矩阵，稳定性强，且迭代2-3次就能获得比较好的结果，详细推导过程可以参考文献【动态电力系统分析】

二阶模型不需要迭代，网络方程求解过程如下。U0是网络初始电压，I1是发电机注入电流I1向量，只和发电机内电势和转子角有关。I2是发电机注入电流I2向量，与发电机端电压有关，对于二阶模型，I2等于0。par\_type是改进欧拉法计算预估值和实际值的标识符。

|  |
| --- |
| I1 = calculate\_generators\_injection\_current\_I1(par\_type)  for i in range(max\_net\_iter):  I2 = calculate\_generators\_injection\_current\_I2(U0, par\_type)  I = I1 + I2  U1 = np.linalg.solve(Y\_mat, I)  if np.max(np.abs(U1-U0)) < 1e-12: # Convergence error of network equation  break  else:  U0 = U1.copy() |

### 网络方程求解程序

网络方程求解程序位于network\_solution.py中，主要任务是求解每一步的网络节点电压。

1. **solve\_dynamic\_bus\_voltage**

|  |
| --- |
| 功能：求解暂态母线电压  参数：par\_type，bool型，True代表求解实际值，False代表求解估计值  返回：无  部分示例程序：  I1 = calculate\_generators\_injection\_current\_I1(par\_type)  buses = apis.get\_all\_devices('BUS')  U0 = np.zeros(len(buses), dtype=complex)  for bus in buses:  i = apis\_system.get\_bus\_num\_after\_renumber(bus)  U0[i] = apis\_dynamic.get\_bus\_state\_data(bus, par\_type, 'Vt')  for i in range(max\_net\_iter):  I2 = calculate\_generators\_injection\_current\_I2(U0, par\_type)  I = I1 + I2  U1 = np.linalg.solve(Y\_mat, I)  if np.max(np.abs(U1-U0)) < 1e-12: # Convergence error of network equation  break  else:  U0 = U1.copy() |

1. **update\_generators\_electromagnetic\_power**

|  |
| --- |
| 功能：更新发电机的电磁功率  参数：par\_type，bool型，True代表求解实际值，False代表求解估计值  返回：无 |

1. **calculate\_generators\_injection\_current\_I1**

|  |
| --- |
| 功能：计算发电机的注入电流I1向量。  参数：par\_type，bool型，True代表求解实际值，False代表求解估计值。  返回：gens\_I1，array，发电机注入电流I1向量。 |

1. **calculate\_generators\_injection\_current\_I2**

|  |
| --- |
| 功能：计算发电机的注入电流I2向量。  参数：① bus\_v，array，母线暂态电压。② par\_type，bool型，True代表求解实际值，False代表求解估计值。  返回：gens\_I2，array，发电机注入电流I2向量。  部分程序示例：  buses = apis.get\_all\_devices('BUS')  gens\_I1 = np.zeros(len(buses), dtype=complex)  generators = apis.get\_all\_devices('GENERATOR')  for generator in generators:  Ed, Eq = get\_generator\_internal\_ralated\_potential(generator, par\_type)  delta = apis\_dynamic.get\_generator\_state\_data(generator, par\_type, 'delta')  Ex = Ed \* sin(delta) + Eq \* cos(delta)  Ey = -Ed \* cos(delta) + Eq \* sin(delta)  E = Ex + 1j \* Ey  i = apis\_system.get\_bus\_num\_after\_renumber(generator)  YGp = calculate\_generator\_internal\_admittance(generator)  gens\_I1[i] = YGp \* E |

1. **get\_generator\_internal\_ralated\_potential**

|  |
| --- |
| 功能：获取发电机相关的内电势。  参数：① generator，int，发电机相连母线号。② par\_type，bool型，True为求解实际值，False为求解估计值。  返回：① Ed，d轴内电势。② Eq，q轴内电势。  部分代码示例：  GMN = apis.get\_generator\_related\_model\_data(generator, 'GEN', 'GMN')  if GMN == 'GENTRA' or GMN == 'GENCLS':  Ed = 0.0  Eq = apis\_dynamic.get\_generator\_state\_data(generator, par\_type, 'Eqp')  elif GMN == 'GENROU' or GMN == 'GENSAL':  Ed = apis\_dynamic.get\_generator\_state\_data(generator, par\_type, 'Edpp')  Eq = apis\_dynamic.get\_generator\_state\_data(generator, par\_type, 'Eqpp')  else:  Ed = 0.0  Eq = 0.0 |

注意：不同模型内电势取值不一样，有些为暂态电势，有些为次暂态电势

1. **calculate\_generator\_internal\_admittance**

|  |
| --- |
| 功能：计算发电机的内导纳。  参数：generator，int，发电机相连母线号。  返回：YGp，发电机内导纳。 |

计算内导纳程序与暂态节点导纳阵计算时一样

## 微分方程求解

模型微分方程使用改进欧拉法求取下一步迭代的状态变量，改进欧拉法的原理在此不再叙述。求解程序位于model文件夹下，sync\_generator.py中是发电机模型的求解，exciter.py中是励磁模型的求解，turbine\_governor.py中是调速模型的求解。每种模型的求解程序包括求预估值和实际值两部分，以GENCLS二阶模型为例。

1. **solve\_GENCLS\_model\_state\_estimated\_value**

|  |
| --- |
| 功能：求解GENCLS模型状态变量估计值  参数：generator，int，发电机所连母线号。  返回：无  部分程序示例：  ddelta = omegas \* (omega - 1)  domega = 1 / Tj \* (Pm - Pe)  omega0 = omega + domega \* time\_step  delta0 = delta + ddelta \* time\_step |

1. **solve\_GENCLS\_model\_state\_actual\_value**

|  |
| --- |
| 功能：求解GENCLS模型状态变量实际值  参数：generator，int，发电机所连母线号。  返回：无  部分程序示例：  ddelta = omegas \* (omega - 1)  domega = 1 / Tj \* (Pm - Pe)  ddelta0 = omegas \* (omega0 - 1)  domega0 = 1 / Tj \* (Pm0 - Pe0)  delta = delta + (ddelta + ddelta0) \* 0.5 \* time\_step  omega = omega + (domega + domega0) \* 0.5 \* time\_step |

## 计算流程

暂态仿真主程序框架位于simulation.py中，若网络未发生扰动，在初始化状态变量后，程序一步的执行过程如下：

1. 求取实际网络暂态节点电压和发电机电磁功率。
2. 计算模型状态遍历估计值，如转速估计值和转子角估计值
3. 利用预估的状态变量，计算预估网络节点电压及发电机预估功率。
4. 由预估值和实际值的状态变量微分值求下一步状态变量的实际值。

设计到的主干程序有

1. **start\_dynamic\_simulation**

|  |
| --- |
| 功能：初始化仿真程序，初始化各个模型的状态变量  参数：无  返回：无 |

1. **run\_dynamic\_simulation\_to\_time**

|  |
| --- |
| 功能：运行仿真至某一个时间  参数：stop\_time，停止时间。  返回：无  程序示例：  while True:  current\_time = apis.get\_simulator\_parameter('dynamic', 'current\_time')  if current\_time > stop\_time:  break  else:  show\_transient\_process\_data() # Show computing information  add\_transient\_process\_data\_to\_file()  run\_dynamic\_simulation\_to\_a\_step() # Run one step simulation  current\_time = current\_time + time\_step  apis.set\_simulator\_parameter('dynamic', 'current\_time', round(current\_time, 4)) |

1. **run\_dynamic\_simulation\_to\_a\_step**

|  |
| --- |
| 功能：仿真运行一步  参数：无  返回：无  程序示例：  solve\_dynamic\_bus\_voltage(True) # Solving the actual transient voltage of system bus  for generator in generators:  solve\_model\_state\_variables(generator, 'GEN', False)  solve\_model\_state\_variables(generator, 'AVR', False)  solve\_model\_state\_variables(generator, 'GOV', False)    solve\_dynamic\_bus\_voltage(False) # Solving the estimated transient voltage of system bus  for generator in generators:  solve\_model\_state\_variables(generator, 'GEN', True)  solve\_model\_state\_variables(generator, 'AVR', True)  solve\_model\_state\_variables(generator, 'GOV', True) |

## 扰动设置

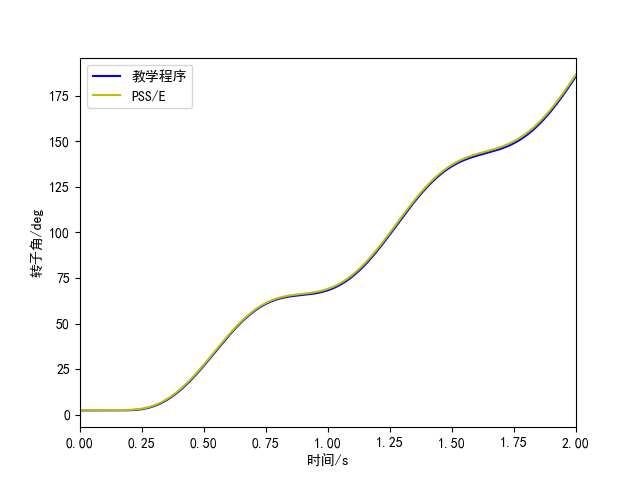
目前电网的扰动一共设置了三种，分别为设置母线三相短路接地故障，清除母线三相短路故障，切除线路，扰动相关程序位于disturbance.py中，以设置母线三相短路为例。

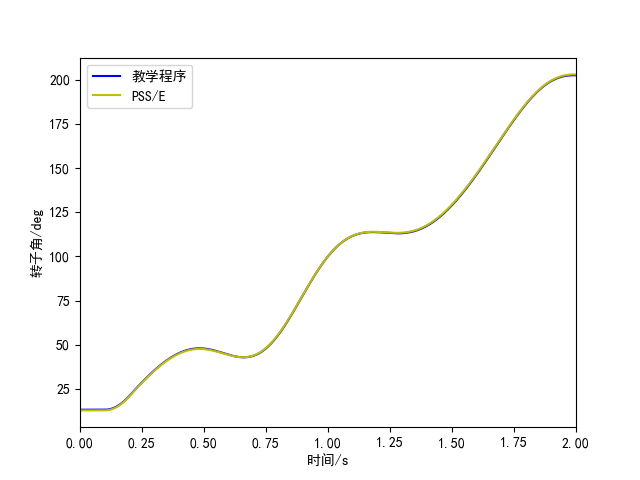
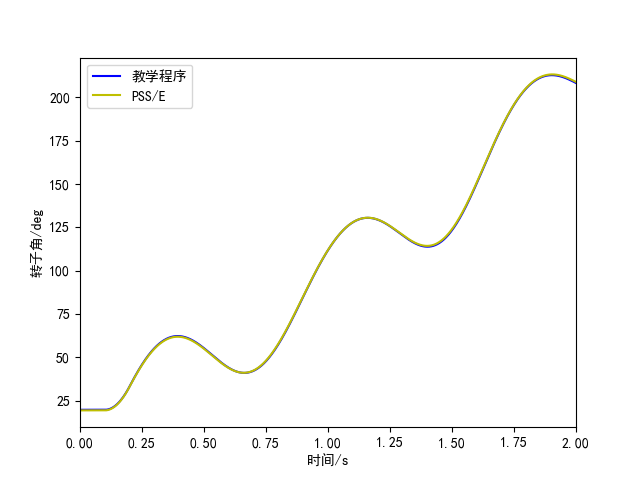
**set\_bus\_fault**

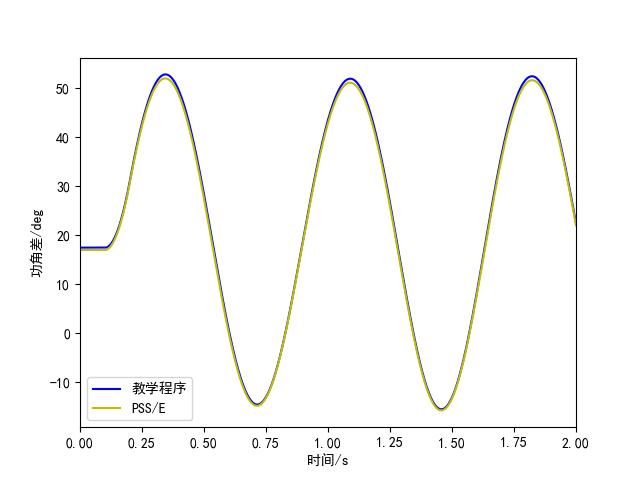
|  |
| --- |
| 功能：设置母线三相短路故障  参数：① bus，int，母线号；② Yf，complex，接地导纳。  返回：无  程序示例：  i = apis\_system.get\_bus\_num\_after\_renumber(bus)  Y\_mat = apis\_system.get\_system\_Y\_network\_matrix('dynamic')  Y\_mat[i, i] = Y\_mat[i, i] + Yf  apis\_system.set\_system\_Y\_network\_matrix('dynamic', Y\_mat) |

## 算例分析

以IEEE9节点算例，验证暂态计算程序。仿真步长0.001s，设置母线7处在0.1s发生三相短路故障，在0.11s清除故障，1.0s结束仿真，输出程序的转子角曲线。使用PSSE软件对算例进行仿真对比。绘制功角曲线，如图所示。







通过对比功角响应曲线，教学程序的功角响应和PSSE软件基本一致。事实上，查看两个软件在2.0s时刻最后的功角数值，差别也不大。考虑到二者在求解方法上的差异，计算软件的精度以及功角初始的差异，这些差异可以接受。

表 6-2 仿真结束功角对比

|  |  |  |  |
| --- | --- | --- | --- |
| 仿真结束功角/deg | 发电机1 | 发电机2 | 发电机3 |
| 教学软件 | 185.8143712 | 208.232306 | 202.570029 |
| PSSE | 186.8269671 | 208.8367044 | 203.0833022 |

在这里需要指出。两个软件初始值存在差异，但原因尚未找到，本教学软件初始值结果与王锡凡老师《现代电力系统》算例中更为接近。

表 6-3 初始功角对比

|  |  |  |  |
| --- | --- | --- | --- |
| 初始功角/deg | 发电机1 | 发电机2 | 发电机3 |
| 教学程序 | 2.27155628 | 19.73132397 | 13.16626627 |
| PSS/E | 2.399978747 | 19.37615676 | 12.77871269 |
| 现代电力系统分析 | 2.27165 | 19.73159 | 13.16641 |