ANDES能够轻松建立电力系统动态仿真中基于微分代数方程的模型，并保证较高的数值准确性。

它含有丰富的商业级动态模型，并实现了很多细节比如limiters, saturation, zeroing out time constants;

它能够生成很多仿真结果；

它能够在脚本环境下建模：基于DAE设备的建模（比如Controller Model and Equation）过程，可以像写数学表达式一样简单。之后，数值代码将会自动被生成，用于快速仿真。

它可以自动生成建模文档；

其他特点：

有大量的传递函数、用于model prototying和系统分析的非连续组件（包括limiters, deadbands, saturation functions）；

工业级二代可持续模型（solar PV, type3 和type4风电）、分布式光伏和储能模型

有很多方法，保括用于潮流计算的牛顿法、trapezoidal method for time-domain simulation、全特征值分析；

可以在一般的台式电脑上几秒钟内完成2000节点系统的20秒暂态仿真；

开箱即用：一旦开发出来一个模型，那么dyr数据可以直接输入。

andes -v num // 控制输出信息：报错、警告

# 自查

Andes selftest //检查安装上的andes的功能是否完整

上述命令等价于 Andes st

如果没有修改模型，那么可以使用自查的快速版，命令为 andes st -q

# 准备

目标：基于andes中定义的符号模型(symbolic model)，生成数值代码（numerical code），为后续仿真做准备。详细可以查看英文手册Development中code generation节。代码的生成利用了Python的符号工具包SymPy。

使用andes run运行一个算例，它的数值代码能够自动生成。另外，andes selftest也能够自动生成代码。

生成的代码文件会保存在目录 .andes\pycode下，用于以后重复利用并加速。

为保持一致性，只有在修改模型（比如加元器件、更改方程的字符串）时才需要生成新的代码。Andes能够自动检测到变化，可以使用命令 andes prepare -i 手动触发。

Andes prepare// 手动生成代码文件

上述命令等价于Andes prep

Andes prepare -i

相当于Andes prepare - -incremental

用于加速数值代码的生成——只有检测到模型有改动时才会生成新代码

如果模型和代码不匹配，那么仿真结果可能无法反映出模型的变化，甚至可能报错。

详情查阅andes.system.System.prepare() ，这个函数根据符号定义的模型生成数值函数，不依赖于测试算例。

生成的lambda函数会被序列化到文件中，但是在这一过程中产生的symPy 对象只存在于调用prepare()的System实例中。

Andes.system.System.undill()用于从文件夹/.andes/pycode下重新加载已经生成过的函数。

如果没有修改过模型，那么可以使用andes.system.System.undill()函数代替.prepare()函数，用于加速。

# 运行

Andes run -h // 查看run这一上层命令下的子命令

### Input path:算例的输入路径

Andes run case\_name.xlsx // case\_name.xlsx 表示算例文件的相对路径或绝对路径

使用-p 或者 - -input -path，制定一个路径去找算例文件

Andes run kundru\_full.xlsx -p /home/hacui/cases

(andesSecond) C:\Windows\System32>andes run D:\Anaconda2\envs\andesSecond\Lib\site-packages\andes\cases\kundur\kundur\_full.xlsx

\_ \_ | Version 1.8.10

/\_\ \_ \_ \_\_| |\_\_\_ \_\_\_ | Python 3.11.7 on Windows, 01/18/2024 09:31:25 PM

/ \_ \| ' \/ \_` / -\_|\_-< |

/\_/ \\_\\_||\_\\_\_,\_\\_\_\_/\_\_/ | This program comes with ABSOLUTELY NO WARRANTY.

Working directory: "C:\Windows\System32"

> Loaded config from file "C:\Users\senu\.andes\andes.rc"

> Loaded generated Python code in "C:\Users\senu\.andes\pycode".

Parsing input file "D:\Anaconda2\envs\andesSecond\Lib\site-packages\andes\cases\kundur\kundur\_full.xlsx"...

Input file parsed in 0.5549 seconds.

System internal structure set up in 0.0270 seconds.

-> System connectivity check results:

No islanded bus detected.

System is interconnected.

Each island has a slack bus correctly defined and enabled.

-> Power flow calculation

Numba: Off

Sparse solver: KLU

Solution method: NR method

Power flow initialized in 0.0045 seconds.

0: |F(x)| = 14.9282832

1: |F(x)| = 3.608627841

2: |F(x)| = 0.1701107882

3: |F(x)| = 0.002038626956

4: |F(x)| = 3.745104006e-07

Converged in 5 iterations in 0.0041 seconds.

Report saved to "kundur\_full\_out.txt" in 0.0015 seconds.

-> Single process finished in 0.9159 seconds.

### Multiprocessing:多线程处理

当输入多个算例文件时，计算潮流（run power flow）

(andesSecond) C:\Windows\System32>andes run kundur\_full.xlsx kundur\_motor.xlsx -p D:\Anaconda2\envs\andesSecond\Lib\site-packages\andes\cases\kundur

\_ \_ | Version 1.8.10

/\_\ \_ \_ \_\_| |\_\_\_ \_\_\_ | Python 3.11.7 on Windows, 01/18/2024 09:40:27 PM

/ \_ \| ' \/ \_` / -\_|\_-< |

/\_/ \\_\\_||\_\\_\_,\_\\_\_\_/\_\_/ | This program comes with ABSOLUTELY NO WARRANTY.

Working directory: "C:\Windows\System32"

> Loaded generated Python code in "C:\Users\senu\.andes\pycode".

-> Processing 2 jobs on 4 CPUs.

Process 0 for "D:\Anaconda2\envs\andesSecond\Lib\site-packages\andes\cases\kundur\kundur\_full.xlsx" started.

Process 1 for "D:\Anaconda2\envs\andesSecond\Lib\site-packages\andes\cases\kundur\kundur\_motor.xlsx" started.

Log saved to "C:\Users\senu\AppData\Local\Temp\andes-2\_\_p59lk\andes.log".

-> Multiprocessing finished in 2.1644 seconds.

### Routine: 潮流计算、时域仿真、特征值分析

指定routine选项，具体有如下三种选择：

Andes run case\_file\_path -r pflow // 潮流计算

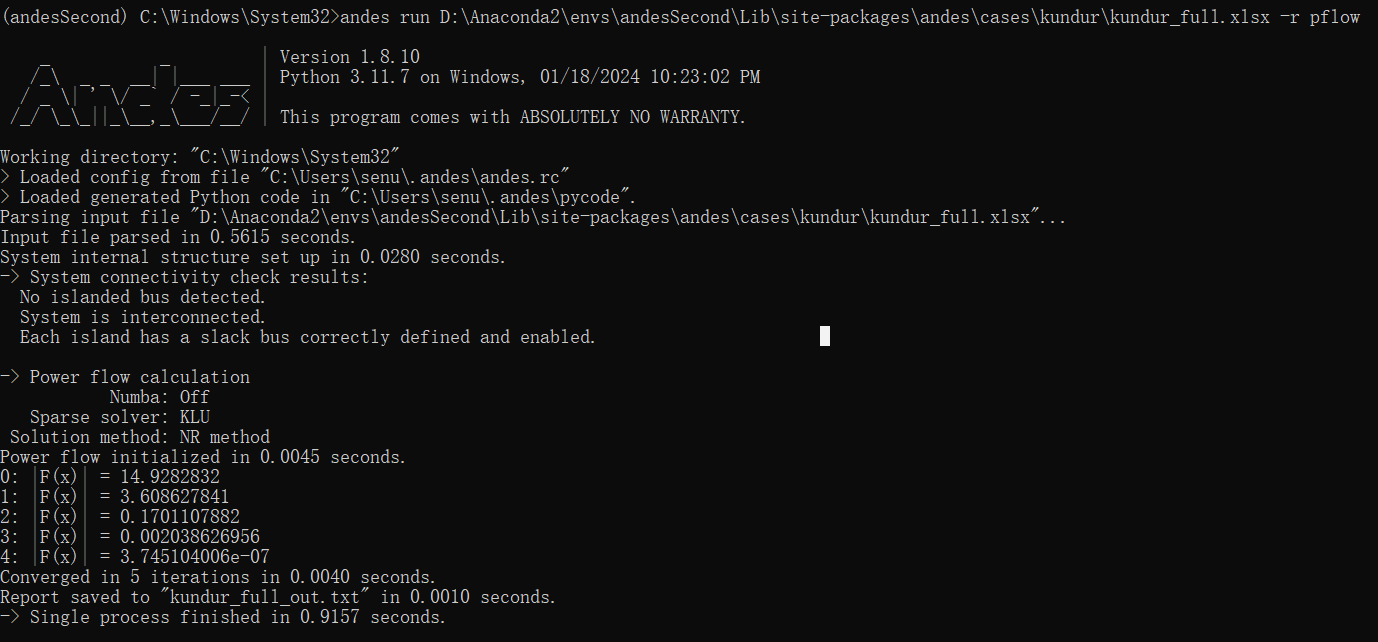
Andes run case\_file\_path -r tds // 时域仿真

Andes run case\_file\_path -r eig // 特征值分析

如果-r选项不写，那么默认是潮流计算。

每个routine都有配置选项（config）去控制具体操作。注意，config选项不同于命令行选项，因为有些config选项在命令行中看不到，详细参考

https://docs.andes.app/en/latest/getting\_started/config/index.html#configuration



#### routine1: 潮流计算

这里使用cundur\_full.xlsx测试算例；

在命令行中执行 andes run kundur\_full.xlsx

得到潮流报告，其中包含如下4个部分

1. 系统分析
2. Ac bus and dc node data
3. Ac line data
4. 其他代数变量、状态变量的结果

潮流计算的默认方法是牛顿拉夫逊方法，默认不检查无功功率极限（reactive power limit）。如果默认配置不能满足你的需要，可以在PFlow，PV等文件中编辑配置文件。

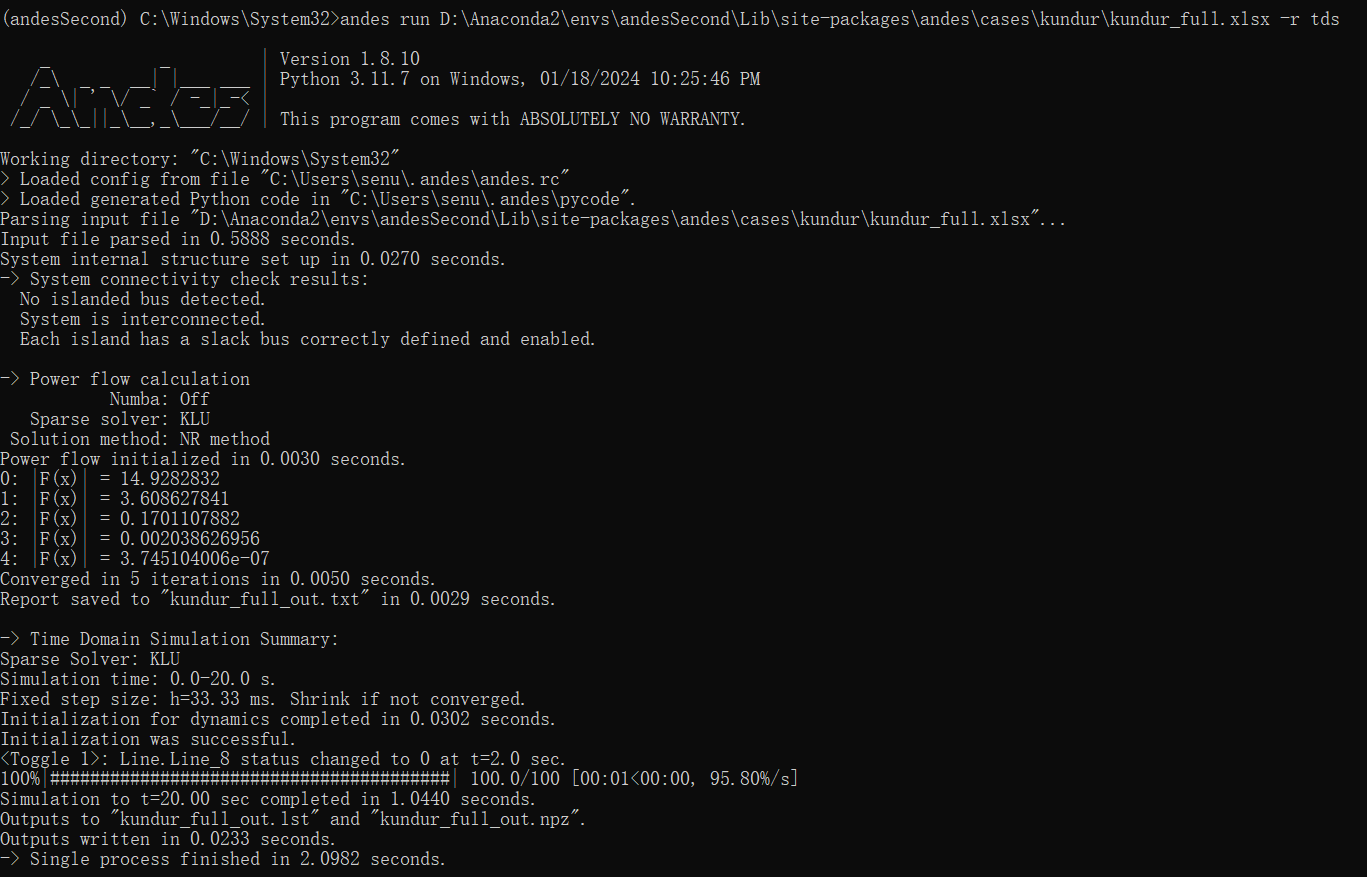
D:\Anaconda2\envs\andesSecond\Lib\site-packages\andes\pflow.py

比如，为了节省时间，andes本身在进行潮流计算时不初始化动态模型。而在开发动态模型时，你可以在配置文件中指定 init\_tds=1。

#### Routine2: 时域仿真

Andes run kundru\_full.xlsx -r tds

输出求解器名称、步长等仿真关键信息；打印出扰动信息，比如下图中线路8在第2秒时的状态变化。



需要注意的是，

1. 在动态仿真之前潮流计算要成功，否则仿真将不能从好起点开始。
2. 确保没有可疑的初始化错误。否则，系统即便是在扰动之前也无法达到稳态。

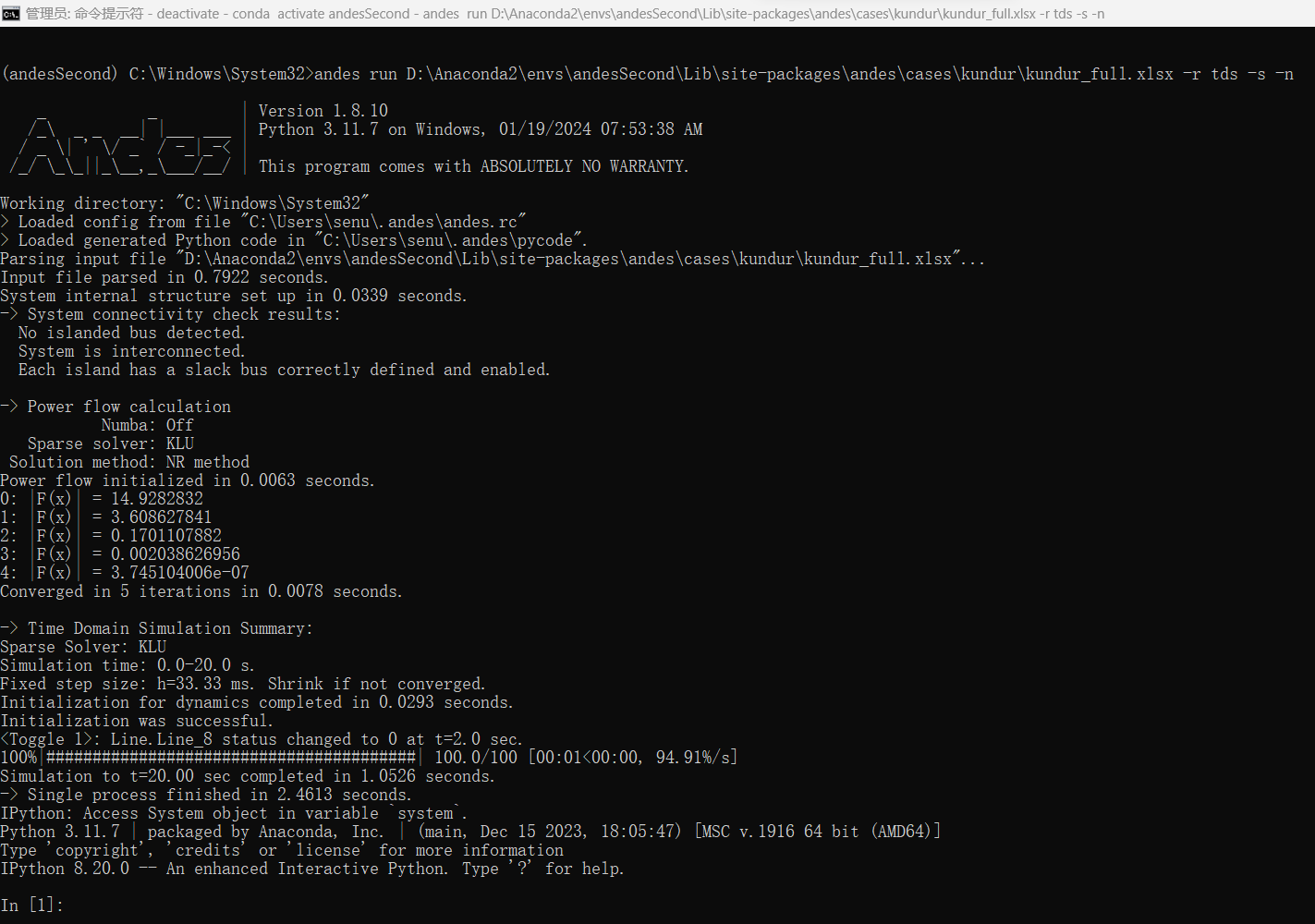
这条命令会有两个输出文件，分别是变量列表文件kundur\_full\_out.lst和kundur\_full\_out.npz，前者的列包括变量索引、变量文本名称、LaTex格式的变量名称。

还有一系列可选参数。。。

如何控制不输出（--no-output, -n）、或者将输出结果放到内存(--shell)中。

##### Ipython shell

如果需要在Python中写脚本快速处理内存中的仿真结果（比如画图），那么可以使用’--shell’,’-s’选项进入到Ipython shell中。



使用‘exit’+enter键，退出Ipython shell界面

##### Format converter: 格式转换

Andes使用excel格式的文件来存储电力系统数据。其他输入数据的文件可以被识别并被转换成xlsx格式。这样转换的好处是你可以使用excel编辑数据了；坏处是无法将xlsx数据转换成原格式。

允许输入的文件格式有: .m, .raw, .dyr, .xlsx, .json

可选的输出文件格式有：.json, .xlsx

##### PSS/E 输入： .raw, .dyr

用andes run kundru.raw 计算潮流

用- - addfile加入dyr文件

使用andes run kundru.raw - -addfile kundru\_full.dyr -r tds ，做无干扰的时域仿真分析

加扰动的方法有两种，

1. 方法一：把PSS/E数据文件转成andes所接受的xlsx文件，修改这个xlsx文件，然后运行；
2. 方法二：<https://docs.andes.app/en/latest/getting_started/formats/psse.html#creating-disturbances>

##### Profiling: 简要介绍

用于分析计算时间、代码效率。

选项 ‘--profile’可以生成andes执行的简要介绍，包含两个文件，第一个文件的文件名以`\_prof.txt`, 第二个文件是····`\_prof.prof`。

.prof文件可以用第三方包`snakeviz`可视化出来。（安装这个第三包： pip install snakeviz）

# 画图

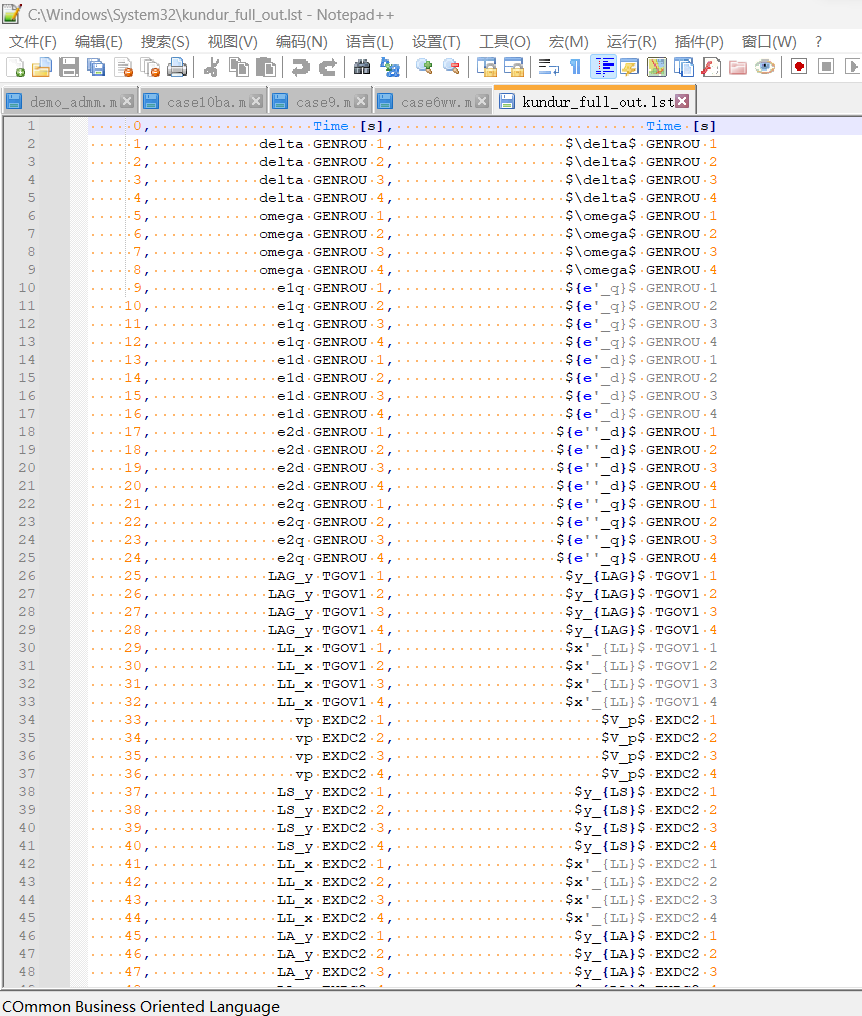
支持时域仿真数据。使用命令`andes plot `时，必须要有三个位置参数，

参数1：以`.out`结尾的仿真输出文件名；

参数2：x轴变量索引，一般地，0表示时间

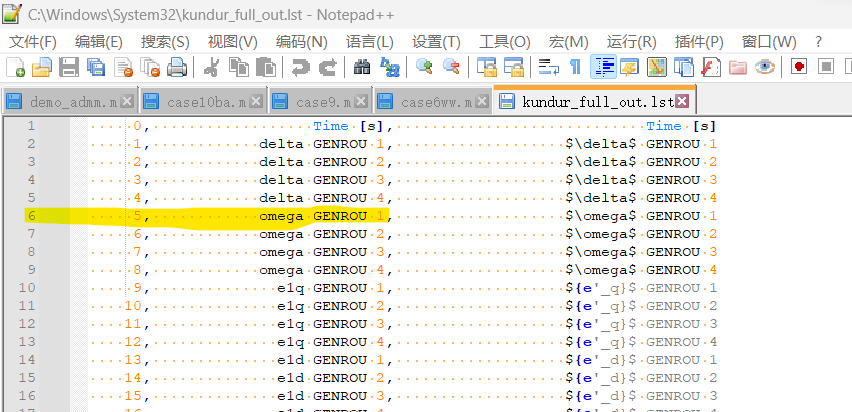
参数3：y轴变量索引，当需要画多个y轴变量时，可以使用空格分隔的索引，也可以使用冒号分隔的索引（下面有示例）。

可以使用命令`andes plot -h`，查阅其他可选参数；

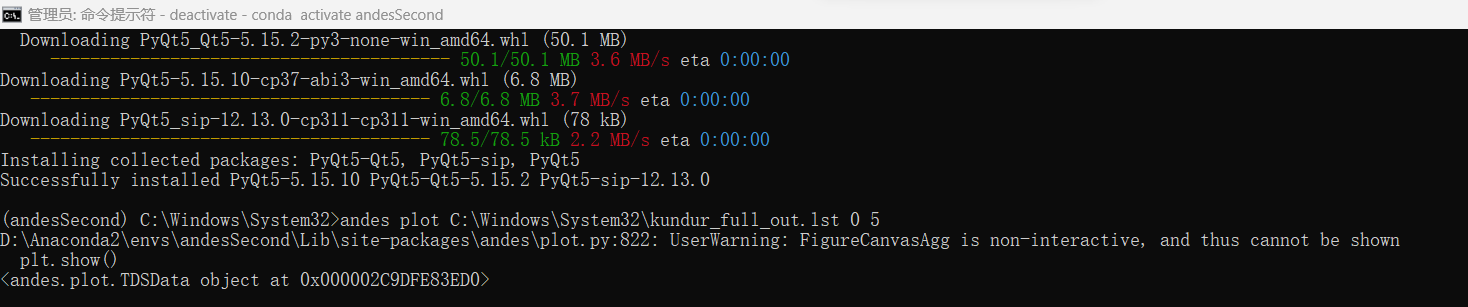


比如，画“GENROU1的发电机速度omega&时间”图，步骤如下：

先在.lst输出文件中找到变量omega GENROU1所对应的变量索引；



输入命令 andes plot kundru\_full\_out.lst 0 5 即可完成画图。（怎么在交互式页面完成画图？？）



注意：

1. 命令 andes plot kundru\_full\_out.lst 0 5 等价于 命令andes plot kundru\_full\_out.npz 0 5。这是因为andes会自动提取文件名。
2. 在指定y轴变量索引时，可以使用Python range，比如：命令 andes plot kundru\_full\_out.npz 0 2:21:6 所绘制的x轴表示时间，y轴分别是索引2、8、14、20所表示的变量。
3. 如果你不想在.ls文件里查找变量索引，你也可以使用`--xargs`或者`-a`搭配变量名画图。详细请参考“Using CLI from Notebook”
4. 默认画图将会有Latex公式，这虽然很好看但是也比较耗时间，你可以使用`-d`把它关掉。

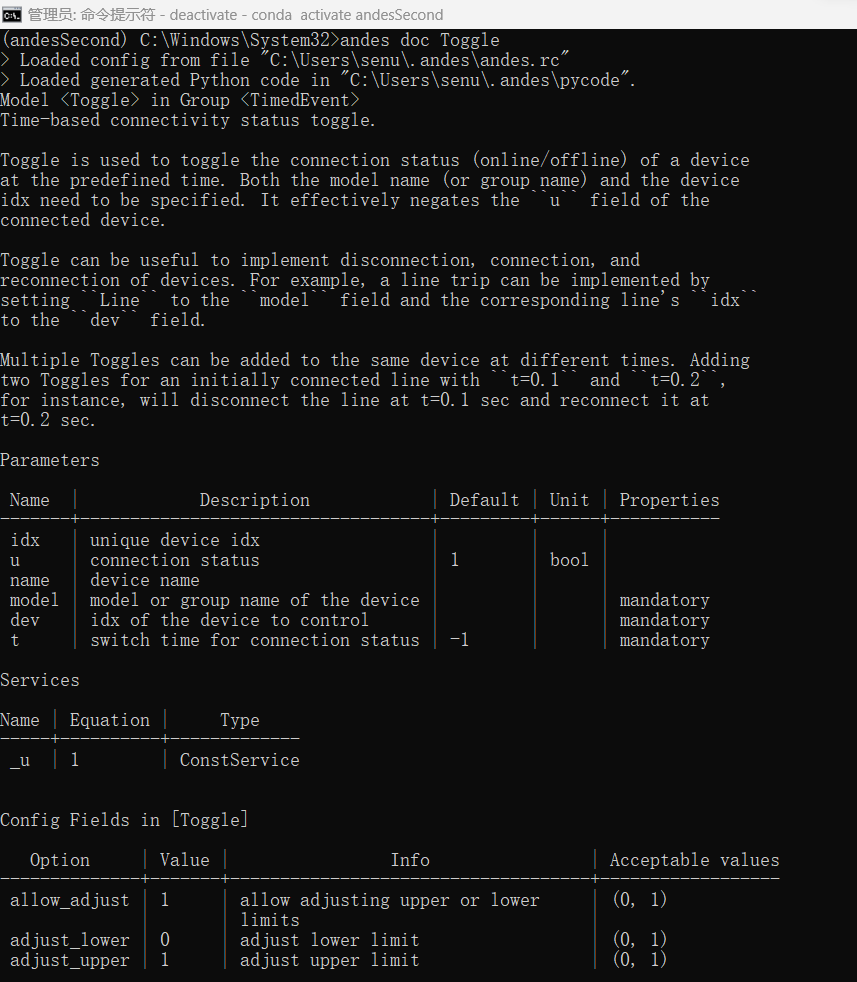
# 写文档

查阅模型、操作（routine）、配置文档。

模型文档包括参数、变量和配置说明，可以参考

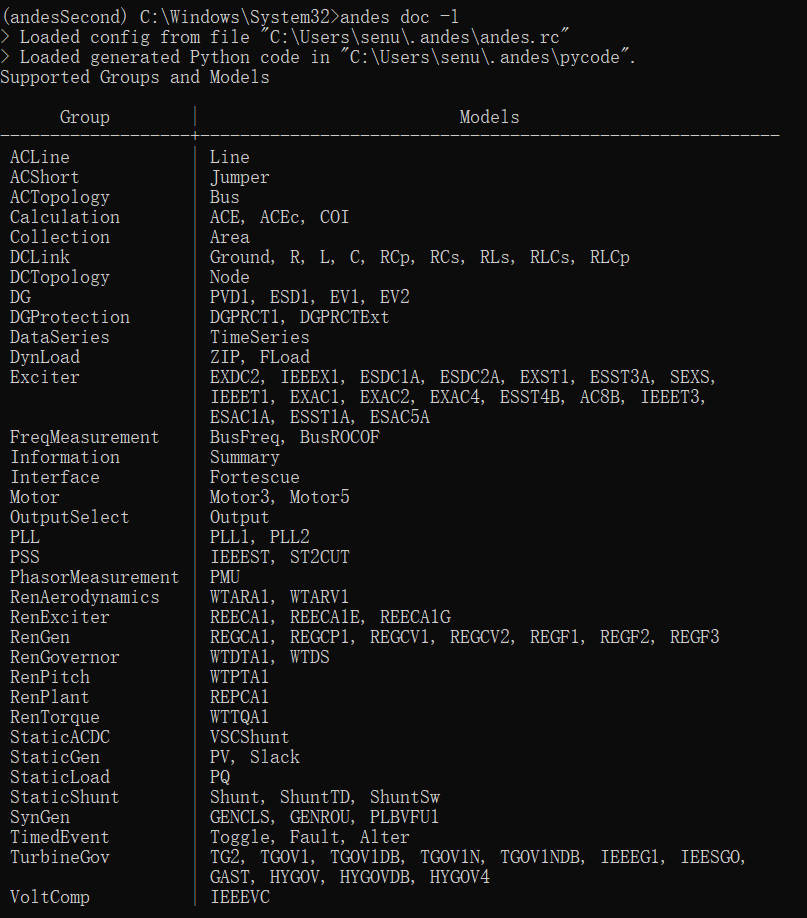
<https://docs.andes.app/en/latest/modelref.html#modelref>

比如，使用命令`andes doc Toggle`查阅模型Toggle。

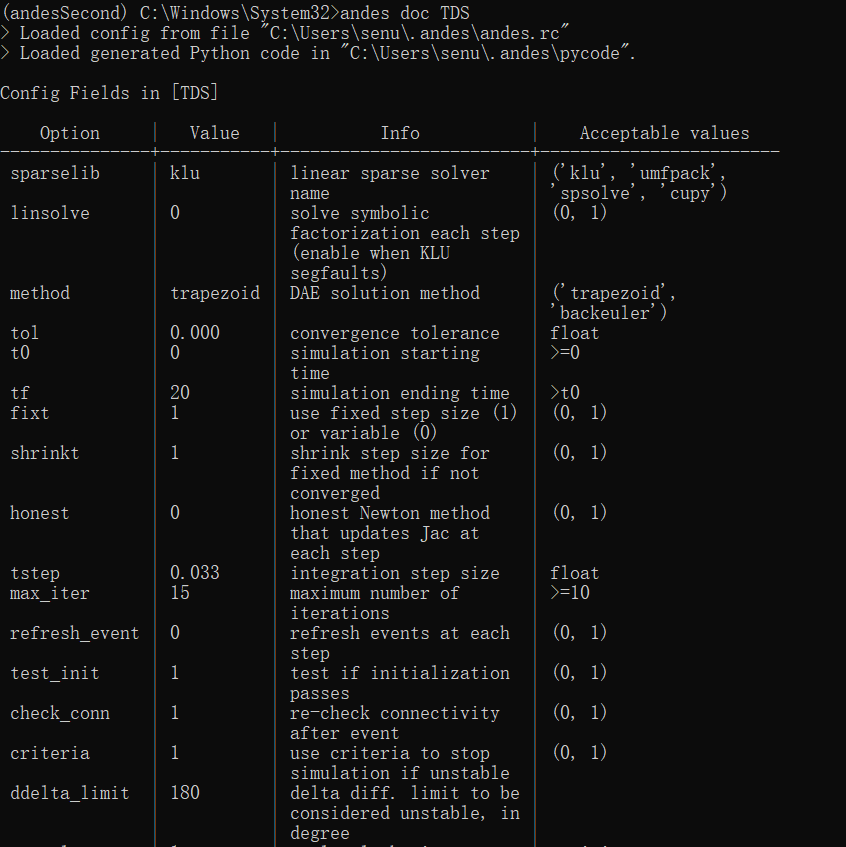


对于一个routine, 使用命令andes doc 将输出配置文件的域。

使用命令`andes doc -l`列出所有模型



使用命令andes doc TDS, 查阅routine时域仿真的配置文件.



# Misc：其他分散的函数功能

Andes misc 包含一些分散的函数，比如如何清空配置、输出文件。

## Configuration: 配置

Andes使用配置文件为指定的routine设置运行时配置参数。

命令Andes misc –save-config 会将所有配置保存到一个文件。默认保存到 $HOME/.andes/andes.conf  文件中。

使用选项`--edit`设置编辑者名字，比如`--edit vim`.

## Cleanup:清空

命令 andes misc -C, --clean 会清除掉所有生成的文件，比如

如果指定删除后缀为`\_out.txt`的文件，那么删除掉的是潮流报告；

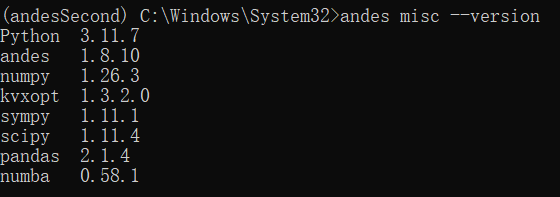
如果指定删除后缀为`\_out.npy`的文件，那么删除掉的是时域仿真数据；

如果指定删除后缀为`\_out.lst`的文件，那么删除掉的是时域仿真变量列表；

如果指定删除后缀为`\_eig.txt`的文件，那么删除掉的是特征值报告；

## Version:检查andes版本

命令 andes misc - -version



# 开发

## 系统

### 模型与微分代数方程的值

建立、求解非线性方程的最大的计算资源开销是建立雅可比矩阵。当我们使用间接的集成方法线性化微分方程时，这变得很重要。考虑到电力系统模型独有的数据结构，雅可比稀疏矩阵也可以逐步地（incrementally, model after model）构建出来。

这里有两种方法逐步地构建稀疏矩阵，

第一种：直接加在稀疏矩阵上，比如

self.fx+=spmatrix(v,i,j,(n,n),’d’)

尽管操作看着很简单，但实际上这一过程需要在右端项创建和丢弃临时对象，甚至有可能修改到雅可比稀疏矩阵self.fx的稀疏模式。

第二种：把行、列、值信息存在数组状的对象中，并最终构建出雅可比矩阵。这种方法很有效，但需要注意的是：在建造稀疏矩阵时不能获取它。

Andes使用预分配的方法，向已知的稀疏模式中填值，从而避免稀疏模式的改变。System收集每个雅可比矩阵的行列索引。

在使用原地做加法之前，andes创建了一个临时用0填充的稀疏矩阵，后续计算出的雅可比矩阵的值会写在这个稀疏矩阵的。

因为这种原地相加操作（in-place addition）修改了已经存在的值，所以它不会改变稀疏模式并且避免了内存拷贝。另外，可以使用方法一中的代码更新稀疏矩阵。

这种方法依然会有创建、丢弃临时对象的问题。

Andes.system.System.store\_sparse\_pattern() 收集并存储雅可比矩阵的稀疏模式。

对于gy矩阵（函数g对变量y的雅可比矩阵），对角元被保存，以防建模用户忘记了方程中的对角元。

### 调用模型方法Calling model methods

System可以调用不同模型的共享方法，比如初始化、更新方程、更新雅可比矩阵、更新discrete flags。

* Andes.system.System.init(self,models,routine)

下面方法的参数是`models`(类型：有序字典)，`routine`（类型：字符串）

为每一个指定的model，初始化变量。

对每一个model，初始化步骤如下：给所有ExtService赋值；调用该model的init()方法，初始化内部变量；拷贝变量到DAE中。

* Andes.system.e\_clear(self,models,routine)

清理在DAE中的方程数组、模型变量

这个函数必须在调用f\_update,g\_update之前完成，以清理（flush）现存值。

* andes.system.System.l\_update\_var(self, models: OrderedDict, niter=0, err=None)

更新基于变量的限制器离散状态

这个函数必须在方程评估之前调用。

* andes.system.System.f\_update(self, models: OrderedDict)

依次调用models的微分方程更新方法。

* andes.system.System.l\_update\_eq(self, models: OrderedDict, init=False, niter=0)

更新与方程有关的限制器离散元件。在评估方程之后，预测方程。

在微分方程更新后才能调用这个函数。

* andes.system.System.g\_update(self, models: OrderedDict)

依次调用models的代数方程更新方法。

* andes.system.System.j\_update(self, models: OrderedDict, info=None)

依次调用models的雅可比更新方法

更新后的雅可比矩阵可以在DAE中的稀疏矩阵（fx, fy, gx, gy）中看到。

### 配置 configuration

System, model, routines都有成员属性config.

* andes.system.System.save\_config(self, ffle\_path=None, overwrite=False)

保存所有的system, model, configurations到.rc格式的文件中；

一旦System实例创建成功，那么保存好的配置文件就被会加载回来。

值得注意的是，

在实例化system时，配置文件中的所有配置会被传到model的构造器中。如果你想要在某次运行中修改配置，那么一定要在实例化System之前完成，或者在在命令行中运行andes之前完成。直接修改Model.config可能不会起作用，而且对当前执行会有负面影响。

## 组 Group

由相似功能的model构成一个组，这些model有共同的变量和参数。

当开发新模型时，要强制加上共同变量和参数，因为他们一般是连接其他组的model的接口。

比如，组RenGen中共同变量Pe,Qe分别表示有功功率输出和无功功率输出。这两个公共变量也是其他组，如RenExciter中的model的公共变量。

如此一来，实现了相同的变量接口，也就是说，同一个组里的model可以执行相似的函数。

组的基本类： GroupBase()

GroupBase的详细内容见英文手册。

## 元件 Models

在文档中，model表示一种元件的数学表示（比如，发电机），而device表示一个model的实例（比如，某个具体的发电机）。

在ANDES中，要用ModelData和Model两个类来定义一种model。类ModelData的功能：根据输入文件定义参数；从devices中添加新数据并且管理数据。类Model的功能：定义其他非输入参数、service变量、DAE变量；转化符号方程、保存雅可比模式、更新方程。

与models相关的类：

|  |  |
| --- | --- |
| 类名 | 功能 |
| ModelData | 保存模型的参数数据 |
| Model | 电力系统微分代数方程模型（DAE models）的基类 |
| ModelCache | 缓存callback函数的返回值 |
| ModelCall | 存储已经生成的函数调用、雅可比调用，和参数 |

### Andes.core.model.ModelData

class andes.core.model.ModelData(\*args, three\_params=True, \*\*kwargs)

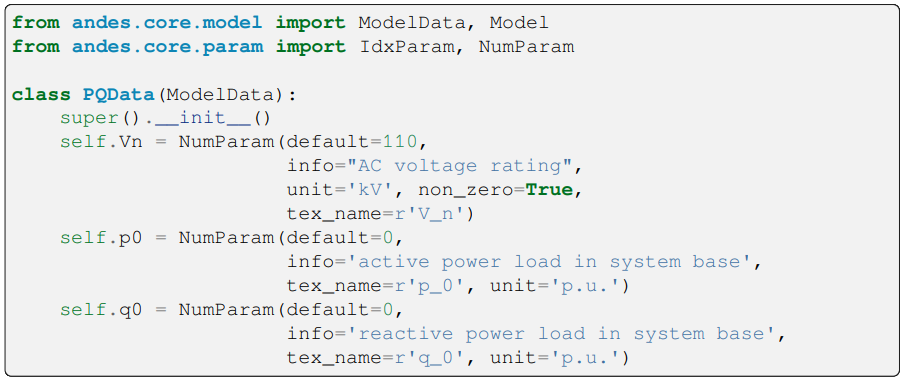
用于保存model的参数

模型要继承这个类来定义输入文件中的参数。推荐将ModelData类的子类的名字定义为元件名称+Data。比如，将GENROU的数据定义为GENROUData。子类自己的参数要定义在子类的\_\_init\_\_函数中。此外，子类将会继承父类ModelData的三个参数，分别是

|  |  |  |
| --- | --- | --- |
| 参数名称 | 含义 | 类型 |
| idx | 唯一的设备索引 | andes.core.param.DataParam |
| u | 连接状态 | andes.core.param.NumParam |
| name | 设备名称 | andes.core.param.DataParam |

* 如何写类ModelData的子类？

示例：定义子类PQData，该子类有三个参数Vn, p0, q0;



类ModelData的方法

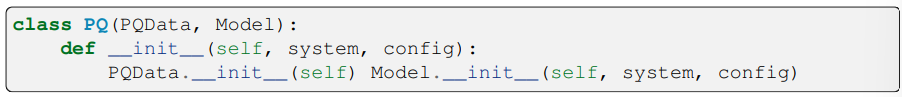
|  |  |
| --- | --- |
| 方法名称 | 含义 |
| .add | 向当前model中添加一个实例 |
| .as\_df | 以pandas.DataFrame格式输出所有参数 |
| .as\_df\_local | 以pandas.DataFrame格式输出局部变量值和service |
| .as\_dict | 以字典格式输出所有参数 |
| .find\_idx | 根据给定模式，返回所有匹配设备的索引 |
| .find\_param(prop) | 根据给定性质，以有序字典的形式返回匹配的参数 |
| .update\_from\_df(df) | 根据给定的DataFrame,更新参数值 |

### andes.core.model.Model

电力系统DAE model的基类。

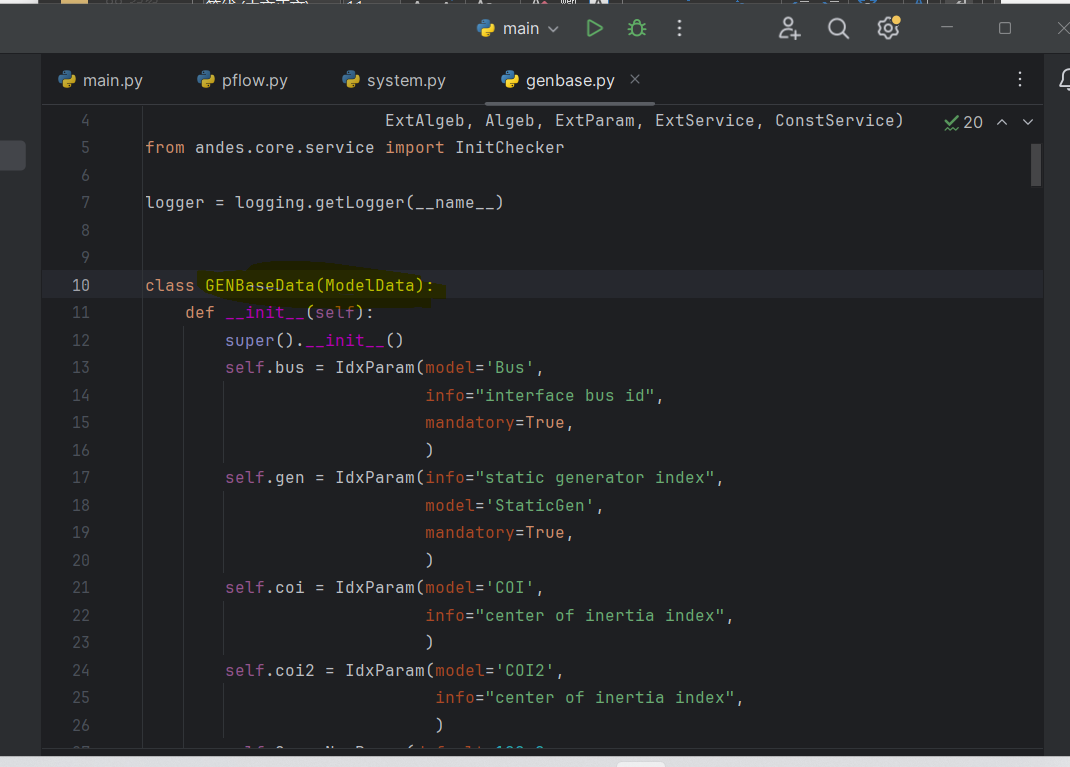
类Model在构造函数.\_\_init\_\_中，定义了DAE变量，service，其他类型的参数。

下面定义了类PQData和类Model的子类PQ



从上图可以看到，类PQ的构造函数依次调用了父类PQData、父类Model的构造函数。调用父类的构造函数意味着当前这个类PQ不能再有自己的子类。

注意，如果继承Model的子类将来会有自己的子类，那么这个继承Model的子类将只能继承Model这一个类，并类的命名要以Base结尾。比如，andes.models.synchronous.genbase.GENBase。



在PQ.\_\_init\_\_中，设置flag来标识这个model要用哪个routine,

比如，self.flags.update({‘pflow’:True})

当前，andes支持pflow和tds的flag。默认pflow和tds的flag都是False，这意味着既不使用潮流计算，也不使用时域仿真。千万不要忘记设置这个flag。

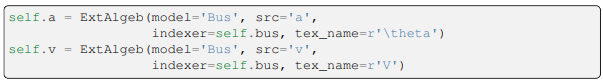
接下来，设置这个model所属组的名称，self.group=’StaticLoad’。这表示当前的这个model属于组StaticLoad，也要有这个组所需要的公共变量和参数。另外，组的名字必须在andes.model.groupz中有定义。如果不提供组的名字，那么这么model将会被放进未定义组中。

接着，添加其他配置flags(configuration flags)。配置flags指的是加载时间变量(load-time variable)，用于指定一个model的行为。这些配置会被输出到一个andes.rc文件中，并且随着System的创建而被自动加载。只要配置flags类型是数值，可以被用在方程字符串中。这里推荐使用有序字典，而不是字典。虽然这样做，语法会显得冗长。示例，self.config.add(OrderedDict(((‘pq2z’,1),)))。

注意到布尔值应该用整数01表示，因为在.rf配置文件中的True/False被加载过来时是字符串，这会造成错误。

下面是变量和方程。

PQ类没有自己内部的变量。它使用bus参数来获取对应的相角变量a和电压变量v。方程上，该类有有功负荷方程、无功负荷方程。



最简单的PQ模型采用常数P,Q,代码为



这里e\_str表示方程字符串属性；u表示连接状态。在方程字符串中也可以使用参数、配置、service、变量。

方程会用到的其他三个量，

-dae\_t 表示当前仿真时间，当model有时域仿真的flag时，这个变量会被用到；

-sys\_f 表示系统频率，参考 system.config.freg

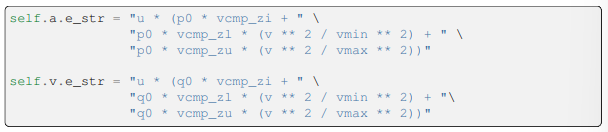
-sys\_mva 表示系统基准值，参考system.config.mva

上述例子是简化版。现在考虑为PQ model加一个功能——当电压不在范围(vmin,vmax)中时，自调为常数阻抗（constant impedance）。为实现这一功能，引入一个叫限制器（Limiter）离散部件（discrete component），这个限制器会产生三个数组，每个数组都是由01的flag构成。这三个数组zi,zl,zu分别表示是否在范围内(in-range)、是否低于下限(below lower-limit)、是否高于上限（above upper-limit）。

示例，创建一个属性vcmp，该属性是Limiter的一个实例



其中，self.config.pq2z 是一个flag，表示是否要开启这个功能。创建了这个属性之后，我们在其他方程字符串中使用vcmp\_zi, vcmp\_zl, vcmp\_zu，示例如下



注意，PQ.a.e\_str可以在尚未定义PQ.vcmp的时候直接用vcmp所产生的三个变量，只要保证确实有PQ.vcmp的定义就可以了，因为v在.e\_str中vcmp\_zi只是一个字符串。

最后，为了能够让ANDES注意到新定义的model，你需要将这个model的名字添加到models/\_\_init\_\_.py中。可以参考有序字典的例子，字典中每一项的键是文件名，其值是类名。

方法列表

|  |  |
| --- | --- |
| 方法名称 | 功能 |
| a\_reset() | 将地址充值为空，重置flags.address为False |
| alter(src,idx, value) | 更改输入参数、constant service的值 |
| doc([max\_width, export]) | 以字符串格式输出模型文件 |
| e\_clear() | 清除所有与内部变量有关的方程值数组 |
| externalize() | 把内部数据外部化成简要介绍(snapshot) |
| f\_numeric | 自定义的fcall函数 |
| f\_update | 评估微分方程 |
| g\_numeric | 自定义gcall函数 |
| g\_update | 评估代数方程 |
| get(src, idx[, attr, allow\_none, default]) | 获取模型某属性的值 |
| get\_init\_order() | 获取变量的初始化顺序并且送到logger.info |
| get\_inputs([refresh]) | 获取输入到数值函数的有序字典 |
| get\_md5 | 返回合并后方程字符串的md5哈希 |
| get\_times | 从TimeParam中获取事件切换时间（event switch times） |
| idx2uid | 把索引转换成（0-indexed）唯一索引 |
| init(routine) | 数值初始化一个model |
| internalize() | 内部化snapshot data |
| j\_numeric() | 自定义数值型更新函数 |
| j\_update() | 更新雅可比矩阵中的元素 |
| l\_check\_eq | 调用离散部件（discrete component）中的check\_eq方法，从而更新与方程有关的flags |
| l\_update\_var | 调用离散部件（discrete component）中的check\_var方法，从而更新内部状态的flags |
| list2array | 把所有值属性转化成numpy数组 |
| mock\_refresh\_inputs | 使用模拟数据填充输入 |
| numba\_jitify | 把方程残差调用、雅可比调用、变量service调用转化成JIT编译的函数 |
| post\_init\_check | 事后（Post）初始化检查 |
| precompile | 对当前model做numba编译 |
| prepare | 符号处理（Symbolic processing）和代码生成 |
| refresh\_inputs | 刷新输入数据的帮助函数 |
| refresh\_inputs\_arg | 用各自的参数列表刷新每个函数的输入 |
| register\_debug\_equation | 帮助函数，注册变量以便于调试初始化 |
| s\_numeric | 自定义service值函数 |
| s\_numeric\_var | 自定义变动的(variable)service值函数 |
| s\_update | 更新service方程值 |
| s\_update\_post | 更新事后初始化service |
| s\_update\_var | 更新andes.core.service.VarService的值 |
| set() | 设置model property的属性值 |
| set\_backref | 帮助函数，设置idx-es到BackRef |
| set\_in\_use | 设置in\_use属性 |
| solve\_iter | 求解迭代初始化 |
| solve\_iter\_single | 对给定device，求解迭代初始化 |
| store\_sparse\_pattern | 保存雅可比矩阵的非零元的行号、列号 |
| switch\_action | 调用切换动作 |
| v\_numeric | 自定义变量初始化函数 |

### andes.core.model.ModelCache

缓存callback函数的返回值

方法列表

|  |  |
| --- | --- |
| 方法名称 | 功能 |
| add\_callback | 添加缓存属性和更新该缓存属性的callback函数 |
| refresh | 刷新缓存值 |

### andes.core.model.ModelCall

存储已生成函数的调用、雅可比调用和参数。

方法列表

|  |  |
| --- | --- |
| 方法名称 | 功能 |
| append\_ijv |  |
| clear\_ijv |  |
| zip\_ijv | 返回已打包的迭代器，指向给定矩阵的行、列、值 |

### 缓存 Cache

ModelData类使用轻量级的类andes.core.model.ModelCache，将数据缓存为字典、pandas的DataFrame。在ModelData中定义了4个属性，分别是

|  |  |
| --- | --- |
| 属性名称 | 描述 |
| dict | 把所有数据存储在字典中，参数名是key，数组作为值 |
| dict\_in | 把所有数据存储在字典中，除了原输入的值（v\_in, original input） |
| df | 把所有数据存储在pandas的DataFrame中 |
| df\_in | 把所有数据存储在pandas的DataFrame中，除了原输入的值（v\_in, original input） |

用’cache.add\_callback’，你可以添加其他属性。

### 定义电压额定值 Define Voltage Ratings

如果一个元件（model）与交流母线（AC bus）和直流节点（DC node）相连，而bus, bus1, node, node1为参数，那么必然会有额定电压所对应的参数Vn, Vn1,Vdcn, Vdcn1。

那些不与Bus、Node相连的控制器会忽略额定电压，因此 Vb=Vn=1，除非你使用andes.core.param.ExtParam获取bus/node值。

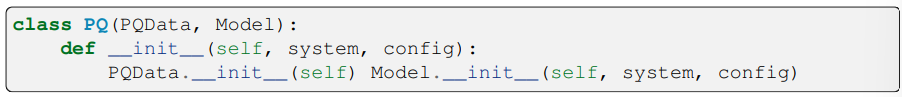
一般来说，那些不与Bus、Node直接相连的控制器会使用电压和当前参数的系统基准值（ststem-base per unit）。控制器，比如水轮调速器turbine governor, 会继承受控models的额定功率，所以功率参数也能完成一致性转换。

#### 定义DAE model

电力系统DAE model的基类。

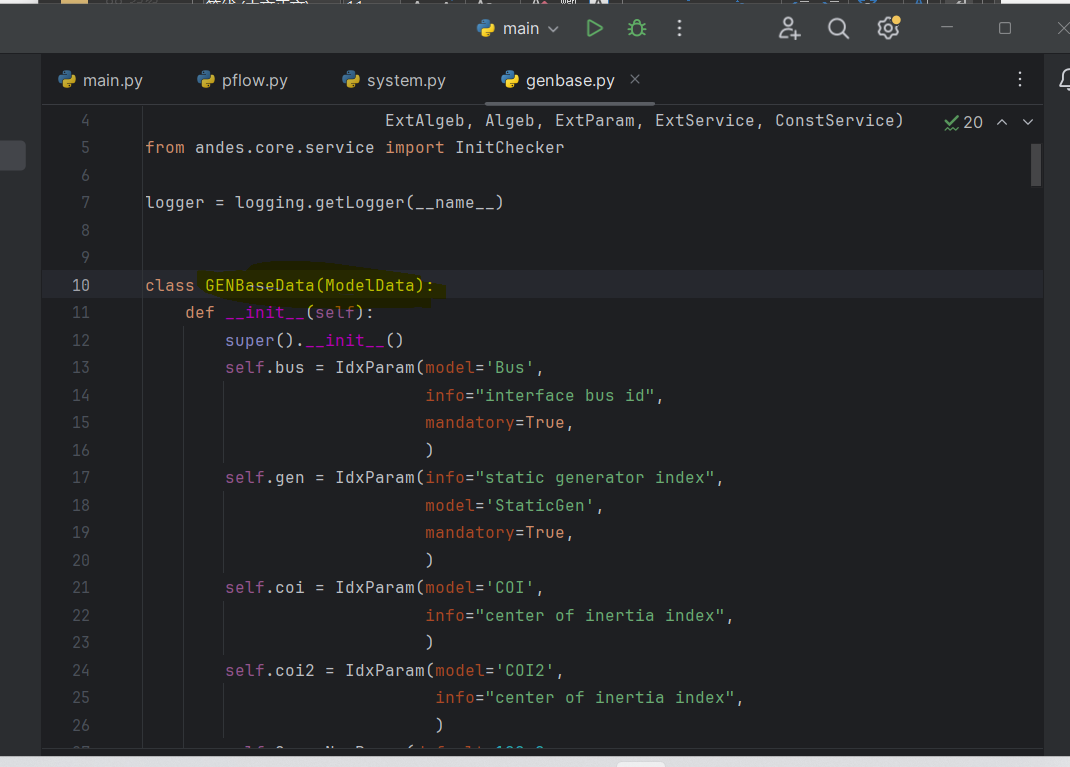
类Model在构造函数.\_\_init\_\_中，定义了DAE变量，service，其他类型的参数。

下面定义了类PQData和类Model的子类PQ



从上图可以看到，类PQ的构造函数依次调用了父类PQData、父类Model的构造函数。调用父类的构造函数意味着当前这个类PQ不能再有自己的子类。

注意，如果继承Model的子类将来会有自己的子类，那么这个继承Model的子类将只能继承Model这一个类，并类的命名要以Base结尾。比如，andes.models.synchronous.genbase.GENBase。



在PQ.\_\_init\_\_中，设置flag来标识这个model要用哪个routine,

比如，self.flags.update({‘pflow’:True})

当前，andes支持pflow和tds的flag。默认pflow和tds的flag都是False，这意味着既不使用潮流计算，也不使用时域仿真。千万不要忘记设置这个flag。

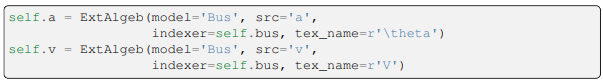
接下来，设置这个model所属组的名称，self.group=’StaticLoad’。这表示当前的这个model属于组StaticLoad，也要有这个组所需要的公共变量和参数。另外，组的名字必须在andes.model.groupz中有定义。如果不提供组的名字，那么这么model将会被放进未定义组中。

接着，添加其他配置flags(configuration flags)。配置flags指的是加载时间变量(load-time variable)，用于指定一个model的行为。这些配置会被输出到一个andes.rc文件中，并且随着System的创建而被自动加载。只要配置flags类型是数值，可以被用在方程字符串中。这里推荐使用有序字典，而不是字典。虽然这样做，语法会显得冗长。示例，self.config.add(OrderedDict(((‘pq2z’,1),)))。

注意到布尔值应该用整数01表示，因为在.rf配置文件中的True/False被加载过来时是字符串，这会造成错误。

下面是变量和方程。

PQ类没有自己内部的变量。它使用bus参数来获取对应的相角变量a和电压变量v。方程上，该类有有功负荷方程、无功负荷方程。



最简单的PQ模型采用常数P,Q,代码为



这里e\_str表示方程字符串属性；u表示连接状态。在方程字符串中也可以使用参数、配置、service、变量。

方程会用到的其他三个量，

-dae\_t 表示当前仿真时间，当model有时域仿真的flag时，这个变量会被用到；

-sys\_f 表示系统频率，参考 system.config.freg

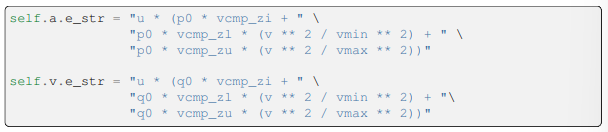
-sys\_mva 表示系统基准值，参考system.config.mva

上述例子是简化版。现在考虑为PQ model加一个功能——当电压不在范围(vmin,vmax)中时，自调为常数阻抗（constant impedance）。为实现这一功能，引入一个叫限制器（Limiter）离散部件（discrete component），这个限制器会产生三个数组，每个数组都是由01的flag构成。这三个数组zi,zl,zu分别表示是否在范围内(in-range)、是否低于下限(below lower-limit)、是否高于上限（above upper-limit）。

示例，创建一个属性vcmp，该属性是Limiter的一个实例



其中，self.config.pq2z 是一个flag，表示是否要开启这个功能。创建了这个属性之后，我们在其他方程字符串中使用vcmp\_zi, vcmp\_zl, vcmp\_zu，示例如下



注意，PQ.a.e\_str可以在尚未定义PQ.vcmp的时候直接用vcmp所产生的三个变量，只要保证确实有PQ.vcmp的定义就可以了，因为v在.e\_str中vcmp\_zi只是一个字符串。

最后，为了能够让ANDES注意到新定义的model，你需要将这个model的名字添加到models/\_\_init\_\_.py中。可以参考有序字典的例子，字典中每一项的键是文件名，其值是类名。

#### 后台的动态性

变量自动生成的奥秘均在andes.core.model.Model.\_\_setattr\_\_()中，而且代码很简单。这个方法设置了名称、文本名称、当前属性实例所属的model，更重要的是起了记账（book keeping）的作用。特别地，当属性是andes.core.block.Block子类时，\_\_setattr\_\_能够递归地获取导出的实例（exported instances），并且预置block名称到导出实例之前。这些方便都要归功于Python的动态特性。

在代码生成过程中，通过检查book keeping属性，比如states,algebs和Model.cache中的属性，创建symbols。

在数值评估阶段，类Model提供了一种方法andes.core.model.get\_inputs()，把可变的值数组收集到字典中，然后作为参数传给数值函数。

### Models中的通用属性

下面列举的属性会被用于调试。如果属性是有序字典，那么该有序字典的值是字符串格式的属性名称，值是实例。

params, params\_ext 分别是内部参数、外部参数的有序字典；

num\_params 是数值型参数，包括内部参数和外部参数。

states, algebs分别是状态变量、代数变量的有序字典。

states\_ext, algebs\_ext分别是外部状态变量、代数变量的有序字典。

discrete是离散部件（discrete component）的有序字典；

blocks是blocks的有序字典；

service是带v\_str的services的有序字典；

service是外部获得的有序字典；

### Model.cache中的属性

Model.cache中的属性是变量、参数、service的记账结构。

|  |  |
| --- | --- |
| 属性名称 | 功能 |
| all\_vars | 所有变量 |
| all\_vars\_name | 由所有变量名字构成的列表 |
| all\_params | 所有参数 |
| all\_params\_name | 由所有参数名字构成的列表 |
| algebs\_and\_ext | 内、外部代数变量的有序字典 |
| states\_and\_ext | 内、外部微分变量的有序字典 |
| services\_and\_ext | 内、外部service变量的有序字典 |
| vars\_int | 内部变量的有序表 |
| vars\_ext | 外部变量的有序表 |

### 生成方程

Model.syms是SymProcessor的一个实例，根据符号型方程生成数值型方程。方程的生成过程分为符号准备、方程生成、雅可比生成、初始化器生成和生成SymPy格式的方程。

xy表示SymPy格式的变量，其构成依次为State,ExtState,Algeb,ExtAlgeb;

f表示微分方程；g表示代数方程；

df表示 df/d(xy);

dg表示 dg/d(xy);

类SymProcessor的方法

|  |  |
| --- | --- |
| 方法名称 | 功能 |
| generate\_equation | 生成方程 |
| generate\_init | 生成舒适化·初始化方程 |
| generate\_jacobians | 生成雅可比矩阵，并以稀疏矩阵的形式存储，比如dg/dy是稀疏矩阵，元素为(row,col,val)，其中row和col分别是方程部索引、变量索引，val是对应的数值lambda函数 |
| generate\_symbols | 在符号方程生成过程中，生成symbol |

函数generate\_equation 把每个微分方程组转化成数值方程calls,并存储在Model.callls中；

微分方程组和代数方程组的属性分别为f和g;

不同的是，service变量是逐个生成，并存储在Model.calls.s的有序字典中。

### 雅可比存储 Jacobian Storage

#### 抽象雅可比存储

在sympy.Matrix上用.jacobian方法，你可以很轻松得到符号版本的（symbolic）雅可比。难点在于雅可比矩阵元素的存储。注意到，在system加载之前，雅可比方程的生成已经开始了，因此在变量数组中只有变量索引是可得到的。对于每个雅可比矩阵的非零元，ANDES存储了方程索引、变量索引、雅可比值（一个常数，或者一个返回数组的可调用函数(callable functions)）。

雅可比矩阵的非零元要么是一个常数，要么是一个表达式。就效率而言，常数、lambda函数被分开存储。所以，在给定系统的情况下，常数会被加载到稀疏矩阵中。

警告：雅可比存储的数据结构已经变了。详情见andes.core.common.JacTriplet类。

Triplets（三元组，分别指方程索引、变量索引、常数值或可调用函数(callable functions)）被存储在Model属性中，命名为\_{i,j,v}{Jacobian Name}{c or None}，其中{I,j,v}分别指方程索引、变量索引、常数值或可调用函数(callable functions)；{Jacobian Name}是两个字符的Jacobian name, 比如fx,fy,gx,gy；{c or None}表示要么是一个字符‘c’表示“是一个常数”，要么是无字符“不是常数”。

比如，对于雅可比矩阵gy(代数方程对变量y的微分)中的常数项，被存储在\_igyc,\_jgyc和\_vgyc中。

对于雅可比矩阵中非常数项，可调用函数(callable functions)被存储在对应的\_v{Jacobian Name}数组中。注意到，\_vgy和\_vgyc之间的差异在于，\_vgy是可调用函数(callable functions)的列表，\_vgyc是常数的列表。

#### 具体雅可比存储

一旦system被加载并且地址也被赋值给变量，那么抽象雅可比的三元组会被地址的数组所取代。新地址和值被存储在Model属性中，命名为{i,j,v}{Jacobian Name}{c or None}。注意，这里没有前置下划线。

比如，model PV有变量{p,q,a,v}。与p相关的方程为-u\*p0；与q相关的方程为u\*(v0-v)。因此，方程u\*(v0-v)对v偏导得到-u；注意到，在生成中u是未知数，所以这个雅可比不是常数，应该存到vgy中。

在\_igy,\_jgy,\_vgy中分别是1，3，lambda函数-u。

比如，当5节点系统被加载时，q和v的**地址？？是[11,13,15, and [5,7,9].** PV.igy和PV.jgy查询PV.\_igy和PV.\_jgy对应的地址列表，存储**[11,13,15, and [5,7,9]（我怀疑是笔误，应该是[11,13,15], [5,7,9]）。**

##### 初始化

提供值的一方，比如service和DAE变量，需要被初始化。service的初始化要在DAE变量之前。Service和DAE变量的初始化要按照其声明顺序依次进行。

每个service，除了用于符号（symbolic）初始化的v\_str, 用v\_numeric指定自定义函数做初始化。对DAE变量，自定义初始化函数为Model.v\_numeric。

ANDES有基于迭代初始化的Newton-Krylov方法。所有带有v\_iter的DAE变量使用这个迭代方法初始化。

##### 其他数值方程 Additional Numerical Equation

其他数值方程可以去构成“符号-数值混合式”框架。当DAE model不标准或者很难被通用化时，可以使用数值函数。因为除了数值仿真还有“从符号到数值”的通用化，所以它和用户所提供的数值函数相同。

ANDES在每个Model子类中提供了如下hook函数 作为自定义数值函数

v\_numeric 自定义初始化函数

s\_numeric 自定义service值函数

g\_numeric 自定义代数方程，更新对应变量的e；

f\_numeric 自定义定义微分方程，更新对应变量的e；

j\_numeric 自定义定义雅可比方程，这函数应该追加到\_i,\_j,\_v中。

对于大多数模型而言，数值函数是不一定要有的，也不被推荐，因为它增加了代码的复杂性。不过，当数据结构和DAE很难在符号框架（symbolic framework）下被泛化（generalize）时，可以使用数值方程。

感兴趣的读者可以看看COI的符号实现（symbolic implementation）计算发动机的惯性速度。COI也可以用for循环执行，而不使用NumReduce,NumRepeat和外部变量。

## 原子类型

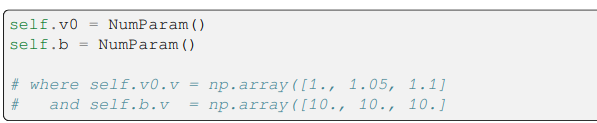
在构建DAE model时，ANDES有三种原子类型，分别是参数、变量和service。

### 值提供者（value provider）

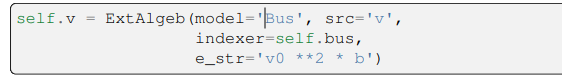
简称v-provider,指的是那些含有v这个成员属性的类，而成员属性v指的是列表或者一维数组，它们均由值构成。比如，所有的参数类都是值提供者，因为参数类要给参数提供值。

注意：所有原子类型都是值提供者，也就是说，原子类的实例必须含有值。

实例所带参数v的值将会用在后续的计算中，示例：如果某model有参数self.v0,self.b



之后这些参数值会被代入到下列方程中，计算接口self.v



共享接口v实现了参数、变量和service之间的交互操作（interoperability）。

在上述例子中，你也可以把self.v0定义为ConstService实例，具体如：



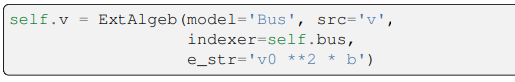
上述的计算照常进行。

### 方程提供者(equation provider)

简称e-provider，指的是那些含有成员属性e的类，其中e为一维数组，数组中的元素是值。在e数组中的值是方程的结果，按照地址属性a汇总到数值型DAE中。

注意：目前，只有变量（variable）是方程提供者，参数、service均不是。

如果某model有与Bus.v(节点电压)相联系的外部变量，比如



对应地，电压变量的地址将会存在self.v.a，方程计算结果（equation evaluation results）将会存在self.v.e。

## 参数

### 背景

参数是一种为DAE model的原子类型。大多数参数可以直接从输入文件中读取到，然后传入到方程中。还有些参数可以使用现存参数计算得到。

在ANDES中，参数的基类是BaseParam，定义了添加值的接口并且检查了值的数量。BaseParam类把值存储在成员属性v(本质是列表)中。BaseParam类的子类NumParam把值存在Numpy数组中。

下面简单列出目前支持的参数

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| BaseParam | 基参数类（base parameter class） |
| DataParam | BaseParam类的别名 |
| IdxParam | BaseParam加上所在model的名称 |
| NumParam | 计算数值型参数 |
| ExtParam | 从外部model或组中获取的参数 |
| TimerParam | 参数值表示在仿真过程中事件发生的时间 |

#### andes.core.param.BaseParam

这个类提供了所有参数类型的基本数据结构和接口。通常，参数来源于输入文件并且一旦被初始化就是常数。

子类应该重载（overload）方法n()，用于计算数组中值的数量。

BaseParam类所含有的参数

|  |  |
| --- | --- |
| 参数名称 | 含义 |
| default | 这个参数的默认值 |
| name | 参数名称。如果不提供，那么默认使用所属元件(model)的属性名称。 |
| tex\_name | Latex格式的参数名称。如果不提供，那么默认使用上面的name |
| info | 这个参数的描述信息 |
| mandatory | 参数是否强制的 |
| export | 参数是否要和其他数据一起写入文件中 |
| iconvert | 当添加设备时，该转化器(类型：callable)被用于输入 |
| oconvert | 输出时用在内部数据的转换器(类型：callable) |

警告：BaseParam,DataParam,IdxParam的明显特征是值被存在列表中，不用转化成数组。BaseParam,DataParam,IdxParam不能在方程中（not allowed in equations）。

BaseParam类所含有的属性

|  |  |
| --- | --- |
| 属性名称 | 含义 |
| v | 存储所有值的列表，类BaseParam无法将属性v转化成numpy数组 |
| property | 含有model properties真实值的字典 |

方法

|  |  |
| --- | --- |
| 方法名称 | 说明 |
| add | 在元件(model)中添加新设备(device)时，把新参数值添加到列表v中 |
| get\_names | 返回列表格式的self.name |
| get\_property | 检查给定性质的布尔值 |
| set | 设置给定位置上的BaseParam类属性取新值 |
| set\_all | 设置所有位置上的BaseParam类属性取新值 |

性质

|  |  |
| --- | --- |
| Property | 说明 |
| class\_name | 返回类的名称 |
| n | 返回数组中数值的个数 |

#### andes.core.param.DataParam

BaseParam类的别名，用于字符串型参数、非计算数值参数（non-computational numerical parameters）。此类不包含.to\_array方法。所有输入数据被存在列表v中。

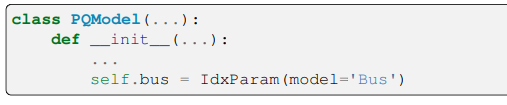
其余信息参考BaseParam类。

#### andes.core.param.IdxParam

与BaseParam类不同的是，存储了该参数所属model的名称。这可以用于将来的数据一致性检查。

示例：

与Bus model相连的PQ model的构造函数如下



可以看到，self.bus作为PQModel类的IdxParam参数，并声明了self.bus所属的model是Bus.

IdxParam类中的方法请参考BaseParam类。

#### andes.core.param.NumParam

用于计算的数值型参数（computational numerical parameter）。

使用这个类定义出的参数将会把属性v（原类型是列表）转化成numpy数组。

原输入值会被拷贝到vin中，系统基准值转化系数将会被存储在pu\_coeff。

NumParam类的参数如下

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| default | 字符串型或者浮点型，如果没有提供值，这个参数可以取的默认值 |
| name | 字符串型，参数名称。如果没有提供，那么将使用所属model的属性名 |
| tex\_name | 字符串型，Latex格式的参数名称。如果不提供，那么默认使用上面的name |
| info | 字符串型，这个参数的描述信息 |
| mandatory | 布尔型，参数是否强制的 |
| unit | 字符串型，参数的单位 |
| vrange | 列表或数组，取值范围 |
| vtype | type型，v的类型，默认是float |
| Sn | 字符串型，设备基准功率参数的名字 |
| Vn | 字符串型，设备基准电压参数的名字 |
| non-zero | 布尔型，参数是否一定非零 |
| non-positive | 布尔型，参数是否一定非正 |
| non-negative | 布尔型，参数是否一定非负 |
| mandatory | 布尔型，参数是否不能为None |
| power | 布尔型，参数是否power per-unit quantity |
| iconvert | callable，将输入数据从excel或其他格式转化成内部v字段的格式 |
| oconvert | callable，将输入数据转化成内部型转化成可序列化型 |
| ipower | 布尔型，参数是否inverse-power per-unit quantity |
| voltage | 布尔型，参数是否是voltage pu quantity |
| current | 布尔型，参数是否是current pu quantity |
| z | 布尔型，参数是否是AC impedance pu quantity // 交流阻抗 |
| y | 布尔型，参数是否是AC admittance pu quantity // 交流导纳 |
| r | 布尔型，参数是否是DC resistance pu quantity // 直流电阻 |
| g | 布尔型，参数是否是DC conductance pu quantity //直流电导 |
| dc\_current | 布尔型，参数是否是DC current pu quantity //直流电流 |
| dc\_voltage | 布尔型，参数是否是DC voltage pu quantity //直流电压 |

方法列表

|  |  |
| --- | --- |
| 方法名称 | 说明 |
| add | 把新参数值添加到列表v中 |
| get\_names | 返回列表格式的self.name |
| get\_property | 检查给定性质的布尔值 |
| restore | 通过拷贝self.vin到self.v中，将参数恢复成原输入值 |
| set | 设置给定位置上的BaseParam类属性取新值 |
| set\_all | 设置所有位置上的BaseParam类属性取新值 |
| set\_pu\_coeff | 把基准值转化系数存储到self.pu\_coeff中，计算系统基准值 |
| to\_array | 将字段v转化成numpy格式 |

注：系统基准值（the system-base per unit）的计算方式为self.v=self.vin\*self.pu\_coeff.

#### andes.core.param.ExtParam

指的是那些参数值需要从外部model,group中获取的参数。

ExtParam类的参数

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| model | 字符串型，提供原参数的model或group的名称 |
| src | 字符串型，原参数名称 |
| indexer | BaseParam类型，indexer.v应该包含到model.src.v的索引。如果indexer是None，那么原参数值会被完全拷贝。如果第一个参数model是组名，那么indexer不能为None。 |
| vtype | 每个元素的类型。如果这个参数是IdxParam，那么vtype是字符串类型。 |
| parent\_model | 提供原参数的父model |

方法列表

|  |  |
| --- | --- |
| 方法名称 | 说明 |
| add | 函数体为空 |
| get\_names | 返回一个只含有参数名称的列表 |
| get\_property | 检查给定性质的布尔值 |
| link\_external | 更新由外部model所提供的参数值 |
| restore | 函数体为空 |
| set | 设置指定位置上的BaseParam类属性取新值 |
| set\_all | 设置所有位置上的BaseParam类属性取新值 |
| set\_pu\_coeff | 把基准值转化系数存储到self.pu\_coeff中，计算系统基准值 |
| to\_array | 将字段v转化成numpy格式 |

性质

|  |  |
| --- | --- |
| Property | 说明 |
| class\_name | 返回类的名称 |
| n | 返回数组中数值的个数 |

#### andes.core.param.TimerParam

这类参数的参数值表示在仿真过程中事件的发生时间。

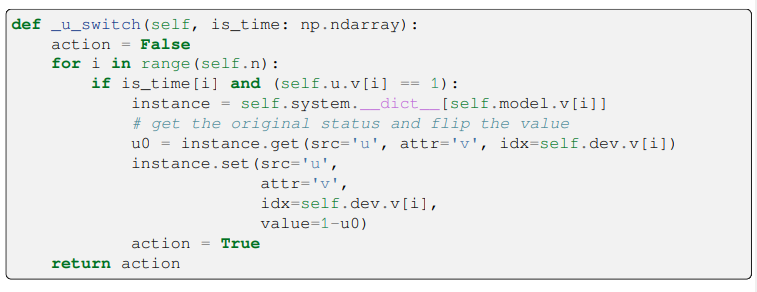
它的构造函数中需要一个Callable型参数self.callback表示事件的动作。这种参数默认值为-1，表示‘不激活‘。

示例：

连接状态类Toggle的参数t表示切断时刻。在Toggle.\_\_init\_\_中，关于参数t的定义如下：



Toggle类中使用\_u\_switch方法切断连接状态



在Toggle.\_\_init\_\_的最后，把上述函数设置为Toggle类的参数t的callback函数，即



方法列表

|  |  |
| --- | --- |
| 方法名称 | 说明 |
| add | 添加一个值到参数值列表中 |
| get\_names | 返回一个只含有参数名称的列表 |
| get\_property(property\_name) | 检查给定性质的布尔值 |
| is\_time(dae\_t) | 逐个元素地检查DAE时间是否与给定参数dae\_t相同 |
| restore | 通过拷贝self.vin到self.v，把参数恢复到原输入值 |
| set | 设置指定位置上的BaseParam类属性取新值 |
| set\_all | 设置所有位置上的BaseParam类属性取新值 |
| set\_pu\_coeff | 把基准值转化系数存储到self.pu\_coeff中，计算系统基准值 |
| to\_array | 将字段v转化成numpy格式 |

## 变量 Variable

DAE变量，简称变量，是未知数，需要使用分析法、数值法求解出来。一个变量会把值(values)、方程值（equation values）、地址（address）存在DAE数组中。变量的基类是BaseVar。

|  |  |
| --- | --- |
| 变量 | 说明 |
| BaseVar | 基变量类 |
| ExtVar | 与外部model相连的代数变量 |
| State | 微分变量类，BaseVar的别名 |
| algeb | 代数变量类，andes.core.var.BaseVar的别名 |
| ExtState | 外部状态变量 |
| ExtAlgeb | 外部代数变量 |
| AliasState | 别名状态变量（Alias state variable） |
| AliasAlgeb | 别名代数变量（Alias algebraic variable） |

### andes.core.var.BaseVar

基变量类，它的继承类State,Algeb才会被用于构建model的变量。

BaseVar的参数列表如下

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| info | 字符串型，描述信息 |
| unit | 字符串型，单位 |
| tex\_name | 字符串型，Latex格式变量符号 |
| discrete | Discrete型，这个变量所依赖的离散部件（discrete component）。ANDES会先运行离散部件的check\_var()，然后初始化变量。 |
| name | 字符串型，变量名称，将会被作为符号名字（symbol name）用在表达式中 |

属性列表如下

|  |  |
| --- | --- |
| 属性名称 | 说明 |
| a | 数组状(array-like)，变量地址 |
| v | 数组状，变量值的局部存储 |
| e | 数组状，对应方程值的局部存储 |
| e\_str | 字符串，方程的字符串/符号化表示 |
| v\_str | 字符串，显式初始化方程 |
| v\_str\_add | 布尔型，如果v\_str的值被加入到变量中，那么返回True。其用处体现在其他model能够拿到这个变量并且设置部分初始值。 |
| v\_iter | 字符串，隐式迭代方程，格式为0=v\_iter |

方法列表

|  |  |
| --- | --- |
| 方法 | 含义 |
| get\_names |  |
| reset | 重置内部的numpy数组和flags |
| set\_address | 设置内部变量的地址 |
| set\_arrays | 设置方程和值数组 |

性质

BaseVar.class\_name

### andes.core.var.ExtVar

与外部model相连的代数变量。这个类能够得到外部model所定义变量的地址。针对这个ExtVar，你可以定义一个方程，方程的估值会存在ExtVar.e中，然后被带入到这个外部变量所对应的方程中。

参数列表

|  |  |
| --- | --- |
| 参数 | 说明 |
| model | 字符串型，原model的名称 |
| src | 字符串型，原变量名称 |
| indexer | [BaseParam,BaseService] 当作到原model和变量的索引。如果为None,那么将会直接复制原变量的地址。 |
| allow\_none | 布尔型，是否允许在indexer中有None |
| e\_str | 字符串型，方程字符串，对应方程的估值将被加入到原残差方程中 |

属性

|  |  |
| --- | --- |
| 属性名称 | 说明 |
| parent\_model | Model类型，提供原参数的父model |
| uid | 数组状，含有到父实例值的索引 |
| e\_code | 字符串型，方程代码字符串 |
| v\_code | 字符串型，变量代码字符串 |

方法列表

|  |  |
| --- | --- |
| 方法名称 | 说明 |
| get\_names |  |
| link\_external | 更新由外部model所提供的变量地址 |
| reset | 重置内部numpy数组和flags |
| set\_address() | 为方程组的右端项指定地址 |
| set\_arrays | 当e\_str存在时，为外部变量的右端项得到dae.h或dae.i |

警告：link\_external不检查ExtVar类型是否与原变量一样，以减少性能代价。如果模型使用外部代数变量ExtAlgeb来获取State，将会有一个无声错误（”维度太小了”,这个报错来自于dae.build\_pattern）。

性质

ExtVar.class\_name

### andes.core.var.State

微分变量类，BaseVar类的别名

参数列表

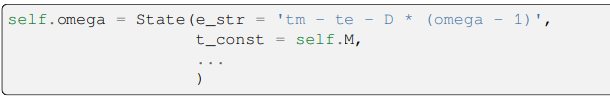
|  |  |
| --- | --- |
| 参数名称 | 说明 |
| t\_const | [BaseParam,DummyValue]微分方程的左端项时间常数。存在dae.Tf中。在估计e\_str中的右端项时，不使用时间常数；但是时间常数会被用到左端项。 |
| check\_init | 布尔值，如果方程右端项初始化为0，那么返回True。当初始化输入不为0时，那就不执行检查 |

示例：

如何实现swing equation，



在对应的model类中的构造函数中，声明如下：



注意到，惯性参数self.M是借助t\_const体现出来，而不是在e\_str中。

方法列表

|  |  |
| --- | --- |
| 方法名称 | 说明 |
| get\_names |  |
| reset | 重置内部numpy数组和flags |
| set\_address() | 指定内部变量的地址 |
| set\_arrays | 指定方程和值数组 |

性质

State.class\_name

State.e\_code=’f’

State.v\_code=’x’

### andes.core.var.Algeb

代数变量类，是andes.core.var.BaseVar的别名。

隐式地表达对应代数变量的残差方程。

示例：

如果想要定义代数变量y和方程y=x+z，那么定义e\_str如下



表示 方程 x+z-y=0。

注意，不要使用e\_str=’x+z’，这将会造成雅可比矩阵奇异，因为d(x+z)/dy为0.

此时，它的e\_code为g，表示代数方程； v\_code为y，表示代数变量y。

方法列表

|  |  |
| --- | --- |
| 方法名称 | 说明 |
| get\_names |  |
| reset | 重置内部numpy数组和flags |
| set\_address() | 指定内部变量的地址 |
| set\_arrays | 指定方程和值数组 |

### andes.core.var.ExtState

外部状态变量类型

警告: ExtState不能设置t\_const，因为它可能会与原State变量相冲突。

只有在少数情况下，你可以为ExtState设置e\_str。原状态变量的t\_const可以使用。

方法列表

|  |  |
| --- | --- |
| 方法名称 | 说明 |
| get\_names |  |
| link\_external | 更新由外部model所提供的变量地址 |
| reset | 重置内部numpy数组和flags |
| set\_address() | 指定方程右端项的地址 |
| set\_arrays | 当e\_str存在时，为外部变量的右端项获取dae.h或dae.i |

属性

ExtState.class\_name

ExtState.e\_code=’f’ 表示微分方程f

ExtState.r\_code=’h’

ExtState.t\_const=None

ExtState.v\_code=’x’ 表示变量x

### andes.core.var.ExtAlgeb

外部状态变量类型

方法列表

|  |  |
| --- | --- |
| 方法名称 | 说明 |
| get\_names |  |
| link\_external | 更新由外部model所提供的变量地址 |
| reset | 重置内部numpy数组和flags |
| set\_address() | 指定方程右端项的地址 |
| set\_arrays | 当e\_str存在时，为外部变量的右端项获取dae.h或dae.i |

属性

ExtAlgeb.class\_name

ExtAlgeb.e\_code=’g’ 表示代数方程

ExtAlgeb.r\_code=’i’

ExtAlgeb.v\_code=’y’ 表示变量y

### andes.core.var.AliasState

方法列表

|  |  |
| --- | --- |
| 方法名称 | 说明 |
| get\_names |  |
| link\_external | 更新由外部model所提供的变量地址 |
| reset | 重置内部numpy数组和flags |
| set\_address() | 指定方程右端项的地址 |
| set\_arrays | 当e\_str存在时，为外部变量的右端项获取dae.h或dae.i |

属性

AliasState.class\_name

AliasState.e\_code=’f’ 表示微分方程

AliasState.r\_code=’h’

AliasState.v\_code=’x’ 表示变量x

### andes.core.var.AliasAlgeb

用处体现在当model的最终输出是来自于block时，model必须提供最终的输出结果，使用预定义的名字。使用AliasAlgeb，model可以避免加入额外的变量和虚拟的方程（dummy equation）。

像ExtVar一样，AliasAlgeb的标签不会保存在最终的输出结果里。当根据文件画图时，你需要查找原变量的名字。

方法列表

|  |  |
| --- | --- |
| 方法名称 | 说明 |
| get\_names |  |
| link\_external | 更新由外部model所提供的变量地址 |
| reset | 重置内部numpy数组和flags |
| set\_address() | 指定方程右端项的地址 |
| set\_arrays | 当e\_str存在时，为外部变量的右端项获取dae.h或dae.i |

属性

AliasAlgeb.class\_name

AliasAlgeb.e\_code=’g’ 表示代数方程

AliasAlgeb.r\_code=’i’

AliasAlgeb.v\_code=’y’ 表示变量y

注意到，与状态变量有段的方程有如下形式

其中，x表示微分变量，y表示代数变量，M是质量矩阵（mass matrix），f是右端项的微分函数。与代数变量相关的方程有如下形式

其中g表示右端项。

BaseVar有两种类型，分别是微分变量类型State、代数变量类型Algeb。微分方程描述状态变量，代数方程描述代数变量。状态变量只能够连续地变化，而代数变量可能是不连续的。

基于model, 你可以定义变量，变量可能是内部的、外部的。大多数变量是内部的，只出现在方程中。有些model会定义一些公共变量，这些变量可以由其他model所获得。比如，Bus定义变量v表示电压量级。每个与母线相关的设备需要获得值，并且实行无功功率注入（impose the reactive power injection）。这一点可以用ExtAlgeb或者ExtState来实现，这两个变量与其他model或group的现存的变量相联系。

### 变量、方程和地址

BaseVar的子类是值提供者、方程提供者。每个BaseVar有成员属性v和e，分别表示变量值、方程值。v的初始值由初始化routine设置，e被初始化为0。在潮流计算或时域仿真过程中，v不是由models直接做修改，而是在解完非线性方程之后更新。e是由models来完成更新，并且在求解方程之前被汇总起来。

比如，每个BaseVar也将这个变量的地址存储在成员属性a中。地址是到数值DAE数组（指的是f和g）的索引。

比如，Bus的电压相角变量self.a=Algeb()。对于5节点系统，Bus.a.a存储了5个Bus设备的变量a的地址。一般来说，Bus.a.a会被赋值成np.array([0,1,2,3,4])。

### 值和方程字符串

符号框架的主要特点是使用字符串定义方程。对于一个变量而言，它有三种字符串，分别如下

v\_str表示方程字符串，以v=v\_str(x,y)的格式做显式初始化

v\_iter表示方程字符串，以v\_iter(x,y)=0的格式做隐式初始化。

e\_str表示微分、代数方程的方程字符串。

下面说明v\_str与v\_iter之间的差别。v\_str直接估成初始值，而所有的v\_iter方程时使用Newton-Krylov迭代方法求数值解。

### 在DAE与Models之间的值

ANDES采用去中心化的结构，在对方程做估值（equation evalution）之前，为每个model拷贝一份变量值。这种结构使得对方程做估值这一过程可以并行展开（在实际操作中，你要能绕过Python的GIL）。然而，这种结构也需要固定的流程，去更新DAE数组和BaseVar数组。具体地，需要定义VarBase的变量值和方程值如何被汇总起来。

这种流程会体现在model定义BaseVar的子类，这种子类可以视为“公有的”(public)。其他model借助ExtAlgeb和ExtState来共享这个变量。

默认，在同一地址的所有v和e要汇总起来。举一个最常见的例子，一个Bus要连接多个设备，那么这些设备的功率注入应该被加起来。

另外，BaseVar提供了两个flags，v\_setter和e\_setter，其用处体现在当一个VarBase需要重写变量、方程值。

### 重写值的flags

BaseVar有专门的flags去处理值初始化和方程值，但这仅用于公共变量、外部变量（public or external variables）。v\_setter表示BaseVar实例是否设置初始值；e\_setter表示与BaseVar相关的方程是否设置方程值。

当从models中收集数据构成数值DAE数组时，程序会检查v\_setter。如果v\_setter是False，那么相同地址的变量值将会被添加。如果某变量的v\_setter是True,那么它会设置DAE数组的值为它的值。在一个地址处，只有一个BaseVar能够设置v\_setter为True.r

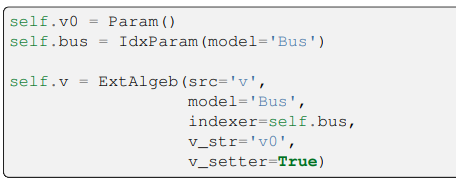
### 关于v\_setter的例子

Bus的默认初始电压等级为1，默认电压相角为0。如果PV设备连接到一个Bus设备，那么PV可以用电压set point(voltage set point)重载电压初始值。

在Bus的初始化函数\_\_init\_\_()中，

self.v=Algeb(v\_str=’1’)

在PV的初始化函数\_\_init\_\_()中，



这个初始化函数中ExtAlgeb借助self.bus作为indexer，获得Bus.v。v\_str所在行设置初值为v0。也就是说，在PV的变量初始化阶段，设置PV.v.v为0.

在借助System类为DAE.y做值收集，PV.v将会重写PV.bus中索引对应的bus的电压等级（voltage magnitude）。

## Service

Services是除了DAE变量列表之外的辅助变量（helper variables）。Service是值提供者（value providers），这意味着每个service都有属性v，用于存储service值。services的基类是BaseService，目前支持的service类如下

|  |  |
| --- | --- |
| 类名 | 说明 |
| BaseService | Service的基类 |
| OperationService | 有具体操作的Service的基类 |

### andes.core.service.BaseService

Service是保留了内部、临时值的值提供者（v-provider）。子类需要实现成员属性v，或者使用性质装饰器（property decorator）。

参数：name, 字符串型，表示这个Service实例名称；

属性：owner，Model型，表示这个实例所属的model的实例

方法列表

|  |  |
| --- | --- |
| 方法名称 | 说明 |
| assign\_memory | 为self.v分配内存并且设置数组为0 |
| get\_name | 以列表格式，返回service的名称 |

性质

|  |  |
| --- | --- |
| property | 说明 |
| class\_name | 类名 |
| n | self.v中的数值个数 |

### andes.core.service.OperationService

执行具体操作的Service的基类。OperationService可能不使用BaseService的assign\_memory方法，因为它可以有不同大小。

|  |  |
| --- | --- |
| 方法名称 | 说明 |
| assign\_memory | 为self.v分配内存并且设置数组为0 |
| get\_name | 以列表格式，返回service的名称 |

性质

|  |  |
| --- | --- |
| property | 说明 |
| class\_name | 类名 |
| n | self.v中的数值个数 |
| v | self.v中的数值 |

不同的子类对sekf.v有不同的实现。

下面是Service子类的简单介绍

|  |  |
| --- | --- |
| 类名 | 说明 |
| ConstService | 常数值的内部Service |
| VarService | variable service,指的是在迭代时先于方程更新的可变服务（variable service） |
| ExtService | 外部服务，用于从值提供者获取值 |
| PostInitService | 在时域仿真初始化之后估出的常服务。 |
| NumReduce | 用于将线性二维数组转化为一维数组 |
| NumRepeat | 重复一维数组成线性二维数组 |
| IdxRepeat | 重复一维列表成线性二维列表 |
| EventFlag | 把输入的变化表示成一个事件（event） |
| VarHold | 当hold signal处于激活状态时，保存输入值（hold input value） |
| ExtendedEvent | 在一段时间内，拓展一个事件信号 |
| DataSelect | 如果有，那么选择字符串数据；或者使用fallback |
| NumSelect | 如果有，那么选择数值数据 |
| DeviceFinder | 查找或创建与给定设备相关的设备 |
| BackRef | 收集反向引用（backward reference）的索引 |
| RefFlatten | 把BackRef列表转化成一维列表 |
| InitChecker | 检查初始值、一般值（typical values） |
| Flag Value | 与给定值相等的Flag |
| Replace | 对于给定的lambda函数，取代那些返回为True的值 |

### 内部常数（Internal constant）

最常用的服务是ConstService，用于存储常数数组，其中常数值是从所提供的符号字符串计算出来的（evaluate）。它们只在model初始化阶段被计算一次，然后初始化变量。

如果你想从参数计算出中间常数（intermediate constants），那么ConstService就能派上用场了。

比如，水轮机调速器（turbine governor）用numParam型参数R表示下垂（the droop）。ConstService可以计算the droop的倒数，得到the gain，并把这一结果用在方程中。示例，下面是水轮机调速器的构造函数，如下



其中，u是在线状态参数。之后，水轮机调速器可以在后续的变量、方程字符串中用G.

ConstService通常是根据参数计算出来的常数。它们只在初始化阶段被估值一次，然后初始化变量。因此，未初始化的变量不能用在v\_str中。ConstService的估值发生在获取外部变量和参数之后、初始化内部变量之前。

参数

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| name | 字符串型，这个ConstService的名称 |
| v\_str | 字符串型，计算变量值的方程字符串 |
| v\_numeric | Callable，返回这个ConstService值的回调函数 |
| v\_type | 值数组中元素的类型 |

属性

v是ConstSerive值

VarService

可变的service（variable service）在每一次迭代都要更新，在计算残差方程之前。最终，从第k步迭代出的变量值被用于计算VarService，而得到的VarService将会被用于计算第k+1步的残差。

这个类的用处体现在不可微分的代数方程上，因为这些方程使用了abs(), re(),im()。你可以把这些方程放在VarService中，而不是创建Algeb变量、放在Algeb变量中，而VarService将会在求解代数方程之前被更新。

参数sequential，布尔型，如果这个VarService依赖之前定义的VarService并且按序被估值，那么为True; 如果这个VarService仅仅使用已知变量。

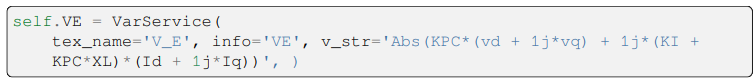
警告：VarService不能使用其他代数方程求解，这意味着在代数变量和VarService之间存在着一步延迟。只要有可能，那么使用一个代数变量。

示例，

在ESST3A model中，电压 ，电流 ，使用如下方程估计VE



注意到这个方程中有绝对值，属于不可微分的代数方程。所以，你可以用VarService实现这个方程



PostInitService

在初始化之后立即被保存的不变服务（constant service ）。

示例，在ESST3A model中，变量要被初始化，并之后还有很多其他变量。你可以把初始的存到中，使得方程成立。

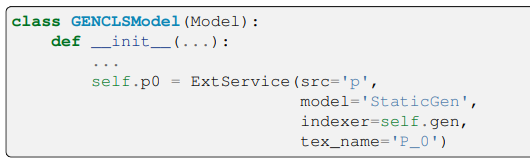


因为所有ConstService在方程估值（equation evaluation）之前做估值，如果不使用PostInitService，你需要创建大量的ConstService去存储在初始化中的值，从而能够正确地初始化.

### external constants （ExtService）

service constants的值来自外部model或group。ExtService的用法与外部变量的用法相似。ExtService的值在初始化阶段只会被获取一次，发生在ConstService估值之前。

示例，同步发电机要用静态发电机的p和q值做初始化。ExtService可以实现这一目标。在同步发电机model的构造函数\_\_init\_\_中，你可以定义参数p0如下，



ExtService类的参数

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| src | 字符串，原model或group的变量或参数名 |
| model | 字符串型，model名或group名 |
| indexer | IdxParam类型或BaseParam类型，Indexer实例的字段v含有model或group中设备的idx |

### shape manipulators (shape manipulator services)

本小节面向高级建模开发者。

所有生成的方程均运行在一维数组上，只能够做代数计算。但在有些情况下，model可能会使用BackRef来获取二维索引，并且使用这个二维索引来获得变量地址（variable address）。由于引用model的内容不同，获取到的变量地址内部长短不一，所以不能直接用来计算。

NumReduce是辅助Service类型，能够将线性存储的2维ExtParam降维后存入到1维Service.

NumRepeat是辅助Service类型，基于BackRef返回值的形状，将一维数组重复成二维数组。

andes.core.service.BackRef类

引用收集器，用于收集设备的索引，这些设备所属的model引用了这个BackRef所属的model，也就是父model。字段v是内部元素为列表的列表，每个元素列表包含其他model的索引，这些model引用了父model的设备。

BackRef可以作为参数或变量的indexer、形状（各个维度的大小），传入到NumReduce和NumRepeat中。

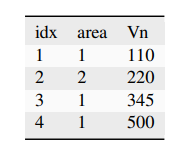
andes.core.service.NumReduce

下面举一个用BackRef类构建COI模型的例子

Bus设备会有area的IdxParam，具体地，保存bus设备所属area的idx。在它的构造函数Bus.\_\_init\_\_中，



Bus的数据如下



Area model想收集它所拥有的母线节点的索引。在Area.\_\_init\_\_中，有



其中，成员属性名称Bus要与这个Area想要收集索引的model的名称完全匹配。同样地，你可以定义self.ACTopology=BackRef()收集在ACTopology组里哪些设备引用了当前Area。

andes.system.System.\_collect\_ref\_param()会收集idx.值得注意的是，某个Area必须存在，才能去收集那些引用它的model的索引。实例，如果Area有如下数据，



那么，只有Bus 1,3,4会被收集到self.Bus.v中，也就是，self.Bus.v=[[1,3,4]]。

如果Area有如下数据，

，

那么，self.Bus.v=[[1,3,4],[2]]。

andes.core.service.NumReduce

这一个辅助Service（helper service）会把线性存储的二维ExtParam降维成一维Service.

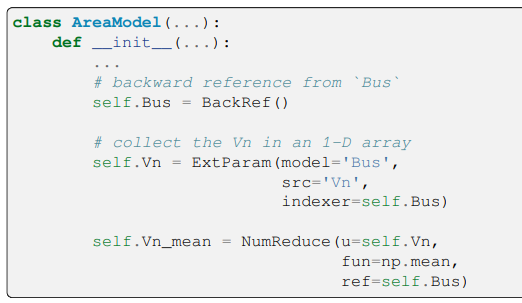
NumReduce作用ExtParam上，它的字段v是列表的列表。要选择一个降维函数（reduce function）作为NumReduce的参数，这个降维函数取数组作为输入并输出一个数值。NumReduce会在字段v的每个子列表上调用这个降维函数，并返回所有的数值到一个数组中。

参数

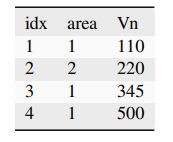
|  |  |
| --- | --- |
| 参数名称 | 说明 |
| u | ExtParam类型，输入ExParam的字段v是线性存储的二维数组 |
| ref | BackRef类型，它的二维形状被用于索引查找（indexing） |
| fun | Callable类型，将一维数组转化为一个数值 |

示例，

如果你想计算每个Area的母线节点Vn的均值，那么你可以在Area model中做如下定义



沿用上述例子，有两个Area,它们的节点数据如下



此时，self.Bus.v=[[1,3,4],[2]]，self.Vn.v=[110,345,500,220]。基于self.Bus的形状，numpy.mean()函数将会分别作用在列表[110,345,500]和[220]上。因此，self.Vn\_mean.v=[318.33,220]。

andes.core.service.NumRepeat()

基于BackRef反馈来的形状，重复值提供者的值（v-provider’s value）

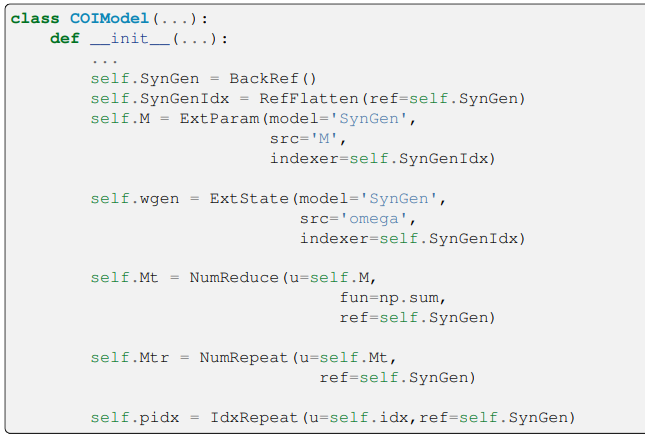
示例，

起初设计NumRepeat的目的是计算转速（rotor speed, center of inertia speed, 惯性中心速度）的加权均值。COI速度的计算公式如下

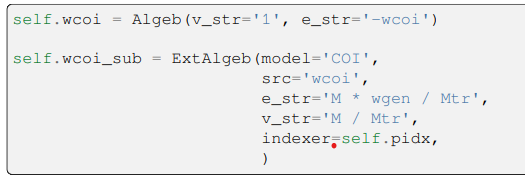


上式的分子可以用BackRef,ExtParam,ExtState计算出来；而分母可以用NumRepeat和NumReduce计算出来，即用NumReduce计算加和，使用NumRepeat为每个设备重复这个加和结果。

在COI类中，



惯性中心速度定义为如下



值得注意的是，上述实现的设计很巧妙——把加权和分离成n个子方程，每个子方程的计算如下。所有变量都存在于子方程中，这保证了求导数不会有差错。

andes.core.service.IdxRepeat

重复IdxParam的辅助类，它与andes.core.service.NumRepeat类有相同功能，但是只能作用在IdxParam，DataParam或NumParam。

andes.core.service.RefFlatten

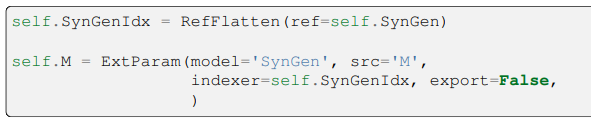
把andes.core.service.BackRef打平成一维列表。当你想把BackRef的结果作为索引值传入时，RefFlatten就能派上用场了。

示例，andes.models.coi.COI收集被引用的同步发电机的索引，代码如下



在收集了BackRefs之后，self.SynGen.v是有索引构成的双层列表，其中第一层表示COI,第二层表示COI所对应的发电机。

下面把self.SynGen转化成一维的self.SynGenIdx，而后self.SynGenIdx将作为获取其他参数、变量的索引。



### 值操作（value manipulation）

andes.core.service.Replace

如果函数值返回真，那么用新值替换参数。

andes.core.service.ParamCalc

计算参数，用于利用已有参数计算新参数

### 索引和引用( Idx and references)

andes.core.service.DeviceFinder

查找与给定设备相连的设备的索引。

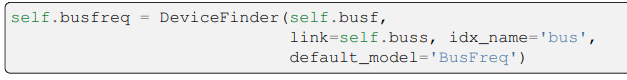
参数auto\_find控制是否自动查找设备索引。参数auto\_add控制这个设备是否要被自动添加。这两个参数不排外（not exclusive）,你可以跳过查找设备，而自动添加它。

如果auto\_find=True并且idx为None，那么DeviceFinder将会查找此设备。如果没找到且auto\_add=True，那么DeviceFinder将会添加此设备。那些被找到、添加的设备的索引会被存储到这个DeviceFinder实例中，因此DeviceFinder的用法与IdxParam的用法很像。

在andes.system.System.setup()中，一开始就是添加新设备。

示例，

在IEEEST稳定器有一个IdxParam类型的可选参数busf，指定相连的母线频率测量设备。为了避免在IEEEST内部重复执行BusFreq，你可以做如下定义



其中，self.busf是母线频率估计设备(bus frequency estimation device)的索引；self.buss是self.busf所测量的bus设备的索引；idx\_name是BusFreq参数的名称，借助这个BusFreq参数被测量母线的索引也是给定的。

对于self.busf中的None和无效值，BusFreq设备将会被创建，并且用self.buss设置bus的对应值。也就是说，BusFreq.[idx\_name].v=[link]。

最后，这个DeviceFinder实例将会包含那些与self.bus相连的BusFreq的列表。

在self.busf中有效值，也就是现存的BusFreq设备，DeviceFinder会返回这个值，并不检查这个BusFreq设备将会测量由self.bus所指定的母线。也可以在不同位置使用这个测量，但用户必须要做数据一致性检查。

### 事件(Events)

andes.core.service.EventFlag

当输入值发生变化时，标识事件的服务。一般的输入是二进制数值的值提供者。

在最近的一次迭代中，EventFlag.v存储着输入变量的值。在评估self.check()之后，更新self.v。

andes.core.service.ExtendedEvent

表示一个事件要在消失后持续上一段时间。（Service for indicating an event for an extended, predeffned period of time following the event disappearance.）

参数

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| u | v-provider类型，触发信号，值为1或0 |
| trig | 字符串型，取值为‘rise’或’fall’，指定事件开始时的触发边缘 |
| enable | 布尔型或v-provider型，如果禁用了，那么输出将是v\_disabled |
| extend\_only | 布尔型，只在扩展时间输出，而不在事件时段输出。 |

警告：这个类的性能需要被优化。

andes.core.service.VarHold

如果冻结标识（hold signal）开了的话，冻结（hold）输入。

参数hold：冻结标识数组，其长度与输入长度相等。如果元素值为1，那么对应的输入将会被冻结直到这个冻结标识恢复成0.

### Flags

andes.core,service.FlagCondition

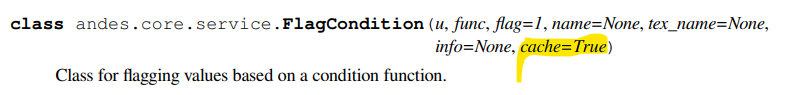
基于条件函数（condition function），标识数值

默认，那些条件函数输出值为1或True的值将被标识为1。否则，标识为0。

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| u | 输入参数 |
| func | 返回True或False的条件函数 |
| flag | 取值0或1，默认为1。那些条件输出为True的值的标识。 |

警告：这个类没有准备好。

FlagCondition只能把cache设为True，应用在BaseParam上。如果不把cache设置为False，那么无法应用到Service上。cache指的是下面标黄处



andes.core.service.FlagGreaterThan

标识那些＞或≥给定值的参数，如果满足＞或≥，那么参数被标识为flag(flag默认为1)。

andes,core.service.FlagLessThan

标识那些<或给定值的参数，如果满足<或，那么参数被标识为flag(flag默认为1)。

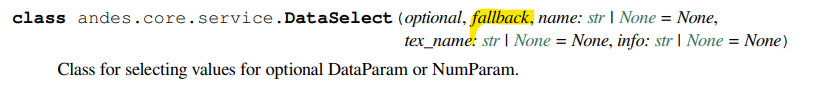
andes,core.service.FlagValue

标识那些=给定值的参数，如果满足=，那么参数被标识为flag(flag默认为0)。

### 数据选择（data select）

andes.core.service.DataSelect

当提供DataParam时，这个service是一个值提供者；否则，使用备用值(fallback value)。



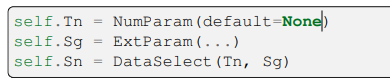
DataParam检查是不是None，NumParam检查是不是np.isnan()。

DataSelect的应用之一：删除节点。通过

self.buss=DataSelect(option=self.busr,fallback=self.bus)

然后，把self.buss，而不是self.bus，作为indexer，去获取电压。

DataSelect的应用之二：设置可选的turbine rating。如下



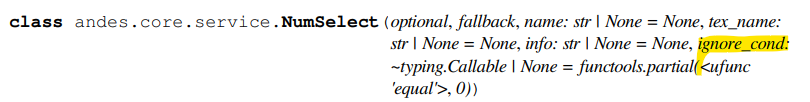
andes.core.service.NumSelect

对可选的NumParam，选择值。

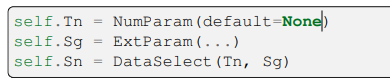
适用于内部、外部参数。

忽略那些值为np.nan的值。如果你想根据额外条件选择性地忽略值，那么通过ignore\_cond传入。比如，如果你想忽略0值，那么你可以用

ignore\_cond=partial(np.equal,0)



示例，设置可选的turbine rating。如下



### Miscellaneous 其他函数和方法

andes.core.service.InitChecker

根据已知典型值，检查初始值。

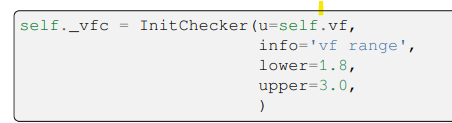
实例被存储在Model.services\_post和Model.services\_icheck之中。在初始化之后，Model.post\_init\_check()内会执行这项检查。

参数

|  |  |
| --- | --- |
| 参数名称 | 说米娜给 |
| u | 待检查的值提供者 |
| lower | 下界 |
| upper | 上界 |
| equal | v\_str中的值应该取的值 |
| not\_equal | v\_str中的值不应该取的值 |
| enable | 布尔型，True表示能够检查 |

示例，

比如，发电机励磁电压取值范围是1.6~3.0标幺值（per unit），那么你可以在GENBase类中添加如下实例



注：lower和upper也可以取v-providers,而不是浮点型数值。

你也可以从Config中传入浮点值，使得它可调，就像GENBase.\_vfc中的实现。

andes.core.service.CurrentSign

用于计算流过一系列设备的电流的符号。

给定一电线，端点分别是bus1和bus2。你可以计算它的电流，也就是从bus1的流出流量，计算公式如下 (v1\*exp(1j\*a1)-v2\*exp(1j\*a2))/(r+jx)。

CurrentSign can be used to compute the sign to be multiplied depending on the observing bus. 对于bus上的值，如果在bus1上，那么符号是+；如果在bus2上，那么符号是-。

andes.core.service.RandomService

用于生成随机数

参数

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| name | 这个RandomService的名字 |
| func | 生成随机数的回调函数（callable） |

## Discrete

### 背景

离散部件（discrete component）库含有一个特殊的block，用于建模电力系统设备的非连续型。这种连续性可能是设备层面的物理约束、施加在控制器上的algorithmic limits。

离散部件的基类是 andes.core.discrete.Discrete

离散类输出标识数组（flag arrays）。

方法列表

|  |  |
| --- | --- |
| 方法名称 | 说明 |
| check\_eq | 在更新方程之后，l\_check\_eq调用这个函数 |
| check\_iter\_err | 检查是否最小迭代和最大误差是否到了启用这个discrete block的程度  Check if the minimum iteration or maximum error is reached so that this discrete block should be enabled. |
| check\_var | 在评估方程之前，l\_update\_var调用这个函数 |
| get\_names | 这个类的可用symbols |
| get\_tex\_name | 输出标识的tex\_name |
| get\_values |  |
| list2array(n) | 为那些在self.export\_flags中指明 的discrete flags分配内存 |
| warn\_init\_limit | 如果相关变量被初始化到了极限，那么发出警告 |

discrete component有输入值、标准（critia），输出一些有特定含义的标识（flags）。输出的标识（flags）可以被用在代数或微分方程中，以建立分段方程（piece-wise equations）。

示例，

Limiter取一个v-provider作为输入，另外两个v-providers分别作为上界、下界。Limiter输出三个flags,分别是zi,zl,zu，分别表示在界内，低于下界，高于上界。可以参考代码models/pv.py。基于电压的PQ到Z的转换。

注意：标识被更新的时间。离散子类（discrete subclasses）使用三个方法检查、更新值和方程。在这些方法中，check\_var会在方程估值（equation evaluation）之前被调用，但check\_eq和set\_eq在方程更新后被调用。在当前的实现中，对于基于变量的离散部件（discrete component），比如Limiter，用check\_var更新标识；对于含有方程的离散部件，比如AntiWindup，AntiWindup只用set\_var存储固定的状态（pegged states）.

### Limiters

Limiter基类

用于比较数值、设置极限值，输出的标识有zi, zl和zu。

参数

|  |  |
| --- | --- |
| 参数名称 | 描述 |
| u | BaseVar型，输入变量实例 |
| lower | BaseParam型，下限的参数实例 |
| upper | BaseParam型，上限的参数实例 |
| no\_lower | 布尔型，如果取True，那么只使用上限 |
| no\_upper | 布尔型，如果取True，那么只使用上限 |
| sign\_lower | 取值1或-1，用于与下限相乘 |
| sign\_upper | 布尔型，用于与上限相乘 |
| equal | 布尔型，在比较中包含“=”，如<=或>= |
| no\_warm | 布尔型，是否禁用初始限值的警告 |
| zu | 布尔型，Default value for zu if not enabled |
| zl | 布尔型，Default value for zl if not enabled |
| zi | 布尔型，Default value for zi if not enabled |

注：

如果不启用，那么默认标识（default flags）zu=zl=0,zi=1。

zl, 数组状（array-like），标识元素是否违反下限；

zi, 数组状（array-like），标识元素是否在界限内；

zu, 数组状（array-like），标识元素是否违反上限；

andes.core.discrete.SortedLimiter

基于元素违反上下限的绝对量、相对量，对输入做排序的限制器。

参数

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| n\_select | 整数型，能够标识为越限的最大元素数。如果n\_select为1，那么最多一个越限的输入会被标识。 |
| abs\_violation | 布尔型，如果取值为True，那么使用绝对越限量（absolute violation）；否则，使用相对越限量。因为大部分变量是标幺值，所以推荐使用绝对越限值。 |

andes.core.discrete.HardLimiter

代数或者微分变量的硬限制器(hard limiter)。它是限制器(Limiter)的别名。

andes.core.discrete.RateLimiter

微分变量的rate limiter。它不会输出任何变量，直接修改了微分方程值。

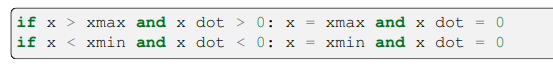
警告：RateLimiter不能用于经历过AntiWindup限制器的状态变量。对于既要速率限制（rate-limited）又要抗积分饱和（anti-windup）的限制器，使用AntiWindRate。

注：

RateLimiter继承了Discrete类，避免了与Limiter的内部命名冲突。

andes.core.discrete.AntiWindup

抗积分饱和限制器。它阻止了微分变量的wind-up effect。如果越限后继续增加，那么重置微分变量的导数。在返回导数时，限制器将失效。



这个类需要不止一个可选参数指定方程。

参数state, 其类型是State类、ExtState类，该状态变量的方程值将会被检查，并且当条件满足时，抗积分饱和限制器会重置它。

andes.core.discrete.AntiWindupRate

带有速率限制的抗积分饱和限制器。

### 值比较（Compares）

andes.core.discrete.LessThan(u,bound,……)

<比较函数，用于检查u<bound是否成立。输出两个标识符，分别是z1和z0。对于满足小于条件的元素，对应的z1=1,而z0是z1的相反数（element-wise negation of z1）。

如果默认的z0，z1没有被启用，那么可以通过构造器设置z0和z1。默认，这个model将会调整这个限制。

andes.core.discrete.Selector

使用提供的降维函数(reduce function)，在两个变量之间做选择。

示例，降维函数可以为np.maximum.reduce用于选择最大值。

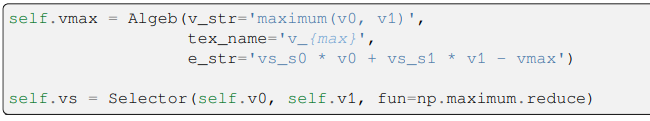
警告：当提供两个以上的输入并且不同输入的值相等时，会报错。你只能有两个输入。

numpy.ufunc.reduce 是Numpy的降维函数。

注意：在Selector的输出标识中，索引从0开始。在Selector构造器中，s0表示第一个变量。

示例：选择v0与v1中的最大值并将其放到vmax中。

在定义完v0和v1之后，定义代数变量vmax为最大值，定义筛选器vs如下：



vmax的初始值由maximum(v0,v1)计算，其中maximum()是SymPy库中找数值每个元素的最大值的函数，然后演化成np.maximum(v0,v1)。vmax的方程基于vs\_v0和vs\_v1选值。

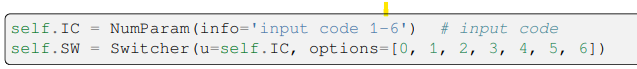
andes.core.discrete.Switcher

基于输入参数的切换器。Switch类取v-provider作为输入，比较输入与可选列表中的每个元素值之间的大小关系，并输出每个选项的标识数组（索引从0开始）。

输出标识的命名分别是\_s0,\_s1,……，其长度为len(options)。

注意：切换器与筛选器不同。切换器基于输入参数，生成选择方案的标识（flags indicating option selection）；而筛选器基于变量值和筛选函数（selection function），在运行时生成标识。

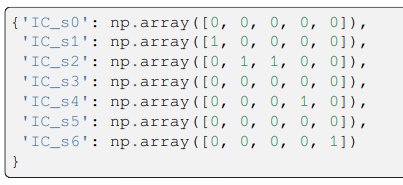
示例: IEEEST取一个输入选择信号。



如果从数据文件得到的IC值如下，



那么，输出的标识数组为



其中，IC\_s0用于填充，使得之后的标识与选项对齐。

### Deadband

andes.core.discrete.DeadBand

deadband基类

参数

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| u | NumParam类型，pre-deadband输入变量 |
| center | NumParam类型，输出的中性值(neutral value) |
| lower | NumParam类型，下界 |
| upper | NumParam类型，上界 |
| enable | 布尔型，是否启用 |

注：

在deadband范围内输入的变化不会造成输出的变化。这个离散部件会计算并输出三个标识flags。

从当前输入计算的标识如下：

zl: 如果输入值低于下界，那么取值为True；

zi: 如果输入值在deadband之间，那么取值为True;

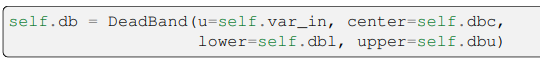
zu: 如果输入值高于上界，那么取值为True。

初始状态：

三个flags都被初始化为0。如果deadband处在启用状态（enabled），check\_var时更新所有标识。如果deadband没有被启用，那么所有标识都设置为0。

示例

与post-deadband变量相对应的代数方程会用到输出的deadband标识。假设pre-deadband输入变量是var\_in，post-deadband变量是var\_out。首先，定义deadband实例db如下



要想实现no-memory deadband——输入介于上下界之间（within the band）输出中心值（center），var的方程可以写成如下形式



andes.core.discrete.DeadBandRT

带返回方向标识的deadband. (deadband with flags for directions of return)。

参数

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| u | NumParam类型，pre-deadband输入变量 |
| center | NumParam类型，输出的中性值(neutral value) |
| lower | NumParam类型，下界 |
| upper | NumParam类型，上界 |
| enable | 布尔型，是否启用 |

注：

在deadband范围内输入的变化不会造成输出的变化。这个离散部件会计算并输出五个标识flags。

除了DeadBand基类中的三个标识之外，其他的两个标识分别是：

zur: 如果输入值在deadband之间并且从上界返回；

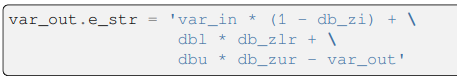
zlr: 如果输入值在deadband之间并且从下界返回;

初始状态：

五个flags都被初始化为0。如果deadband处在启用状态（enabled），check\_var时更新所有标识。如果deadband没有被启用，那么所有标识都设置为0。

示例

要想实现输出固定在deadband边界附近，var的方程如下



### 其他

andes.core.discrete.Average

计算一段时间或若干仿真步骤中BaseVar的均值。

Average基于Delay类中实现的内存。

这个Average类的输出名称为<INSTANCE\_NAME>，其中<INSTANCE\_NAME>是Average的实例名。

andes.core.discrete.Delay

对输入变量，施加预定的延迟（若干步steps或者若干秒seconds）。

延迟量一定是数值，在定义实例化Delay时给定这个值，

Delay使用内部内存，去存储已有变量值。

默认的延迟模式是步数step，但有可能被设置为time。In the time mode, the value at the current time - delay will be interpolated based on the two nearest times and values.

Delay可以用在状态变量、代数变量。输出变量命名为<INSTANCE\_NAME>\_v，其中<INSTANCE\_NAME>是Delay实例的名字。

andes.core.discrete.Derivative

使用数值微分计算变量的导数。

导数利用了Delay类中实现的内存。延迟量被设置为1步，则当前步和前一步都会被使用。

一阶导数的计算公式如下，

其中，tstep是当前步长。

考虑到代数变量的不连续性，Derivative专用在代数变量上。它也可以应用在状态变量上，但你要把该状态变量的右端项写成代数方程，才能得到准确的导数。

It can be applied to a state variable, but one should instead implement the right-hand side equation of the state variable in an algebraic equation to obtain the accurate derivative.

另外，高通滤波器（washout filter, andes.core.block.Washout）可以求数值稳定的导数。

Derivative类的输出的名字是<INSTANCE\_NAME>\_v，正如Delay。

andes.core.discrete.Sampling

采样并冻结。

定期采集一个输入变量，并冻结该值直到下次抽样。

比如，这可以用这个类实现AGC信号的4秒钟抽样。

Sample类的输出的名字是<INSTANCE\_NAME>\_v，其中<INSTANCE\_NAME>是抽样实例的名字。

andes.core.discrete.ShuntAdjust

用于调节可切换的分流器。

参数

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| v | BaseVar型，电压测量值 |
| lower | BaseParam类型，电压下界 |
| upper | BaseParam类型，电压上界 |
| bsw | SwBlock类型，电纳的SwBlock实例 |
| gsw | SwBlock类型，电导的SwBlock实例 |
| dt | NumParam类型，延迟时间 |
| u | NumParam类型，连接状态 |
| min\_iter | 整数型，切换分流器的最小迭代次数 |
| err\_tol | float型，启用切换的最小迭代容忍限度（tolerance） |

## Block

### 背景

block库里包含着公共block，比如传递函数、非线性函数。写DAEmodel的过程很像搭乐高积木，变量、方程会被预定义到block中。block的基类是andes.core.block.Block。

支持的block包括Lag, LeadLag, Washout, LeadLagLimit, PIController。另外，还提供了分段非线性函数的基类Piecewise。Piecewise用于实现二次饱和函数MagneticQuadSat、指数饱和函数MagneticExpSat。

在构造函数中，block的所有变量都要被定义成属性(attributes)，就像model中的变量。不同之处在于，变量从block导出到capturing model。在block构造器的结尾处，所有输出变量被放在self.vars中。

andes.core.block.Block

用于控制blocks。blocks会被实例化成Model属性，提供预定义的方程组。子类必须重载\_\_init\_\_方法，从而接收自定义输入。Block的子类必须重载define方法，从而提供初始化和方程字符串。在构造器末尾处，导出的变量、services、blocks必须定义到字典self.vars中。

block可以嵌套使用。block中含有block，但它自身可以作为属性(attributes)，因此复用了方程。当block有子块（sub-block）时，外部块的构造必须有‘name’。

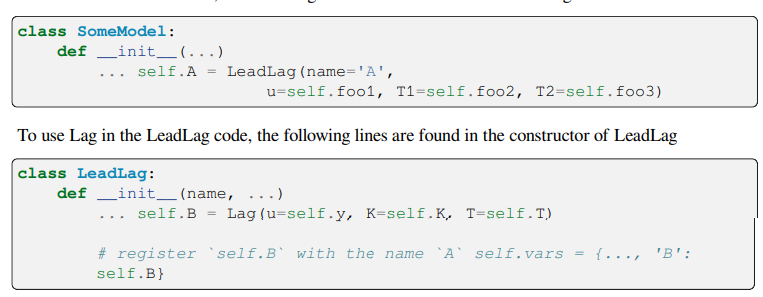
嵌套式block：父block修改子block的属性name，具体地，在构造阶段，把父block的名字附加到子block的名字上。父block把所有子block当作一个整体导出。当父Model类使用这个block时，循环地引入block中的变量和子block。

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| name | 字符串型，block的名字 |
| tex\_name | 字符串型，block的Latex名称 |
| info | 字符串型，block的描述 |
| namespace | 字符串型，取值为’local’或’parent’。如果取值为’local’，那么block的名字将会被附加上父block的名字。如果取值为’parent’，那么在导出时会使用原名字。 |

警告：尽量避免嵌套使用。

示例：两层的block

SomeModel的实例M中含有LeadLeg block的实例A，而这个LeadLeg block的实例A含有Lag实例B。A和B输出x和y两个变量。



当实例化block时，\_\_setattr\_\_函数为导出的变量和block指定名字。对于LeadLag实例，它自己的名字是A，它成员属性名称是B，也就是说，B.name=’A\_B’。

当SomeModel.\_\_setattr\_\_识别到A时，伴随着A的导出，B也将会被识别到。按照递归顺序，B的变量被导出。回顾B.name是A\_B，根据命名规则（父block的名字+变量的名字），B的内部变量为A\_B\_x和A\_B\_y。

在SomeModel的构造函数中实例化中，LeadLag实例名字，A.name，必须给定，从而确保正确的名字能够被传下去。如果有多层，那么所有父block的名字必须在初始化时就全部提供。

因为命名规则相同，所以B的define()不需要做修改。比如，尽管被嵌套着的B有了新名字‘A\_B’，B的内部变量y是{self.name}\_y。

define()

用于内部变量的初始化和方程字符串的设置。这个方法必须由子类来实现。

方程的书写应该使用最终变量的名字。比如，block实例被命名为blk（保存在这个block的self.name中），定义内部变量v。那么，在父model中，这个内部变量被记作blk\_v。因此，所有方程应该使用{self.name}\_v表示变量v，其中{self.name}是运行时这个block的名字。

另一方面，由外部提供的参数、变量的名字可以直接从name属性中获得。比如，如果self.T是block构造器所提供的参数，那么在方程中可以用self.T.name。

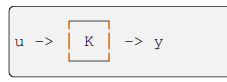
你也可以参考 PIController.define, 里面是PT controller block的方程。

### 传递函数（Transfer Function）

下面是一些已经实现的传递函数的blocks，它们可以被用来去构建新model.

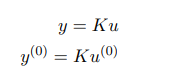
#### 线性 Linear

andes.core.block.Gain



导出代数输出y，

define()实现的方程和初始化条件是



andes.core.block.GainLimiter

输出代数变量y。

这个block有一个列表points，列表中含有N个点，分别是[，定义了n+1段，分别是。它的另外一个列表funs含有N+1个函数字符串，分别是。

检查输入落在了哪个区间段，那么使用对应的函数。比如，第一段(-inf,x0)对应的函数是fun\_0。

如果没有条件得到满足，那么函数返回0.

define()为分段方程建立方程字符串。

andes.core.block.HVGate

High Value Gate.输出两个输入的最大值。

andes.core.block.LVGate

Low Value Gate. 输出两个输入的最小值。

andes.core.block.DeadBand1

第一种Deadband：线性，非阶梯。

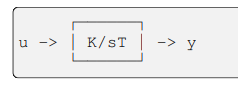
参数

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| center | 当取值落在deadband内时，返回的默认值。如果输入是错误信号，那么center性感被设置为0. |
| gain | 用于乘上DeadBand discrete block的输出  Gain multiplied to DeadBand discrete block's output. |

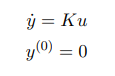
#### 一阶, 二阶，PI控制器

andes.core.block.Integrator

积分block，输出微分变量y。初始化输出需要通过y0指定。

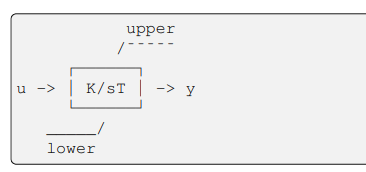


define()实现的方程和初始化条件如下：

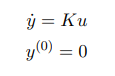


andes.core.block.IntegratorAntiWindup

带抗积分饱和限制器的积分block，导出微分变量y和抗积分饱和lim。初始化输出需要通过y0指定。



define()实现的方程和初始化条件如下：



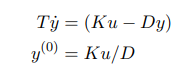
andes.core.block.Lag

Lag（low pass filter）transfer function, 低通滤波的传递函数，输出状态变量y。

参数

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| K | Gain |
| T | 时间常数 |
| D | 常数 |
| u | 输入变量 |

方程和初始值如下



其他类

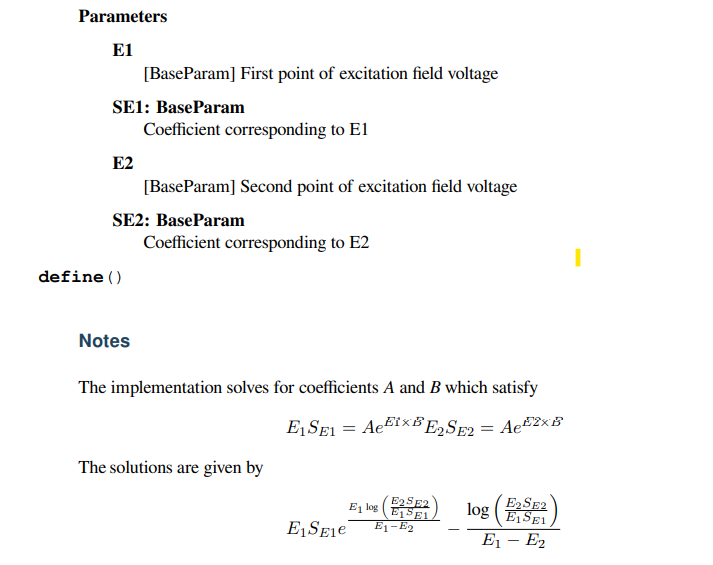
|  |  |
| --- | --- |
| 类名 | 说明 |
| andes.core.block.LagAntiWindup | 带抗积分饱和限制器的低通滤波的传递函数 |
| andes.core.block.LagFreeze | Lag with an input to freeze the state. |
| andes.core.block.LagAWFreeze | Lag with anti-windup limiter and state freeze |
| andes.core.block.LagRate | 带速率限制器的低通滤波的传递函数 |
| andes.core.block.LagAntiWindupRate | 带抗积分饱和限制器、速率限制器的低通滤波的传递函数 |
| andes.core.block.Washout | 高通滤波block |
| andes.core.block.WashoutOrLag | Washout with the capability to convert to Lag when K = 0 |
| andes.core.block.LeadLag | 超前-滞后传递函数 |
| andes.core.block.LeadLagLimit | 带硬限制器的超前-滞后传递函数 |
| andes.core.block.Lag2ndOrd | 二阶滞后传递函数 |
| andes.core.block.LeadLag2ndOrd | 二阶超前-滞后传递函数 |
| andes.core.block.PIController | 比例积分控制器 |
| andes.core.block.PIAWHardLimit | 带抗积分饱和限制器和硬限制器的PI控制器 |
| andes.core.block.PITrackAW | PI with tracking anti-windup limiter |
| andes.core.block.PIFreeze | 状态冻结的PI控制器 |
| andes.core.block.PITrackAWFreeze | PI controller with tracking anti-windup limiter and state freeze. |
| andes.core.block.PIDController | Proportional Integral Derivative Controller |

### 饱和 Saturation

andes.models.exciter.ExcExpSat (E1, SE1, E2, SE2, name=None, tex\_name=None,。。。)

指数励磁机饱和block，根据E1,SE1,E2和SE2计算A和B。输入参数会被校正、用户也会被警告。为了禁止饱和，设置E1或者E2为0.

参数解释如下



### 命名规则 Name convention

当用block建模时，采用如下命名规则。建模block的实例用两个字母的缩略词表示，随后加上数字或者短但是有意义的变量名。注意，缩略词和变量名都不能含有下划线，这是因为block的输出含有下划线‘\_y’。

比如，两个高通滤波器(washout filter)的名字可以是WO1和WO2。再比如，用于电压感知（voltage sensing）的一阶滞后函数的名字是LG；进一步地，如果model中只有一个Lag实例，那么名字为LG。

## 示例

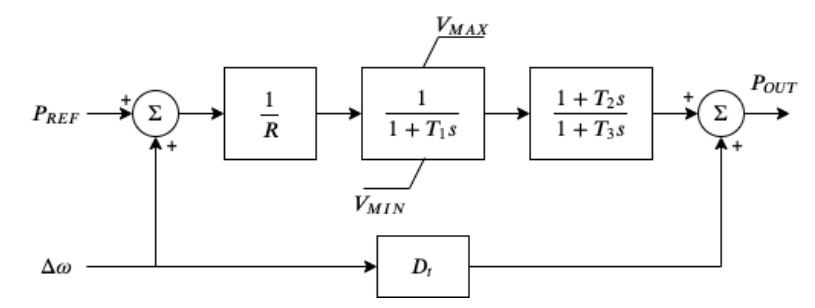
下面举两个例子说明根据方程建模、根据控制模块图(control block diagrams)建模。

示例1：TGOV1展示了基于方程建模、基于block建模的一部分代码；

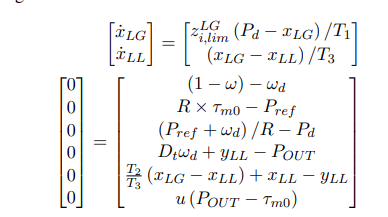
示例2：IEEEST展示了源代码，并解释了完整的准备工作（setup），内容包括可选参数、输入的选择和人为标幺值转换。

### TGOV1

水轮机调速器



这个model内部包含了超前-滞后传递函数和带抗积分饱和限制器的一阶滞后传递函数，整体来说是一个相当复杂的示例。对应的微分方程、代数方程如下：



其中，LG和LL分别表示滞后block、超前-滞后block；

是内部状态

是超前-滞后输出

是发动机under-speed

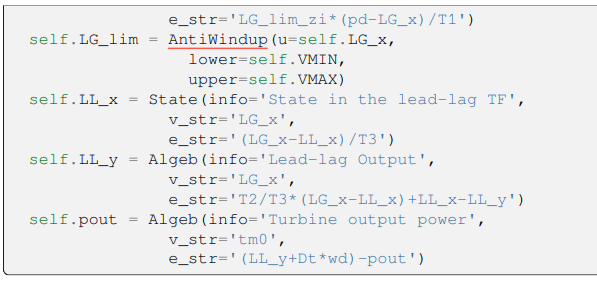
是droop output

是初始状态torque输入；

是turbine输出。

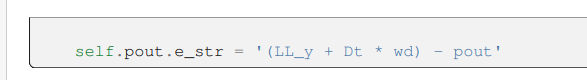
关于上述方程的代码展示如下，完整的代码可以在andes/models/governor.py文件中TGOV1ModelAlt中找到





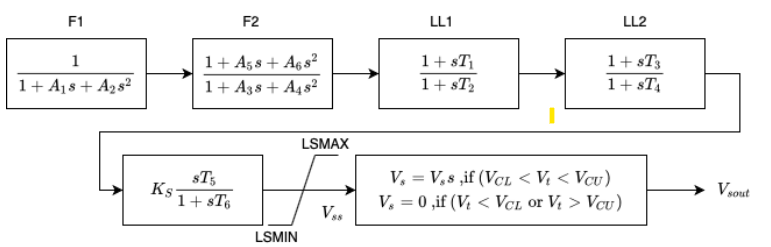
TGOV1的另外一种实现充分利用了block，使得整个代码可读性更高，展示部分如下，完整的代码可以在andes/models/governor.py文件中TGOV1Model中找到





### IEEEST

IEEEST的块图（block diagram）如下，更推荐阅读源码andes/models/pss.py



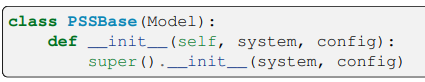
首先，引入建模部件(modeling component)；

接着，PSSBaseData被用来定义那些被所有PSS共享的参数。PSSBaseData继承ModelData并且调用基类构造函数。只有字段avr定义被连接的励磁机的索引。

然后，IEEESTData定义IEEEST的输入参数。使用IdxParam作为字段，用于存储IEEEST所连设备的索引。对于数值型参数，使用NumParam。

#### PSSBase

定义由所有PSS所共享的公共外部参数、服务和变量。PSSBase类和构造器的签名如下



PSSBase继承Model，调用基构造器。调用Model构造器接收两个位置参数，system和config，分别是System和ModelConfig型。

接下来，指定所属group，并且设置model flags。



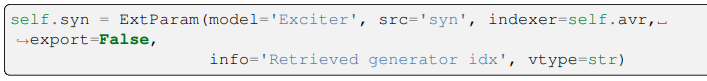
注：切记一定要设置TDS flag以注册这个model。如果不设置，虽然这个model的数据会被成功加载，但是这个model的变量不接收任何地址，并且也会跳过model方程。相似的标识，self.flag.pflow，表示这个model是否要进行潮流计算。然而，大多数动态model是在潮流计算之后被初始化的。

接下来，Replace()用新值替换满足lambda函数的输入参数。

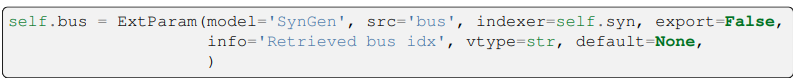


当VCUr和VCLr第一次被获取时会发生值替换。在service更新的末尾处、model初始化阶段，Replace()被执行。

然后，获取相连的发电机、母线、母线频测（bus frequency measurements）。同步发电机的索引可通过如下方式获取



借助self.syn，获取与发动机相连的母线



PSS model通过参数busr支持可选的远程母线(optional remote bus)。当busr为None时，与发动机相连的母线将会被使用。接下来，使用DataSelect优先选择busr；如果busr为None，会选择bus。



每个PSS都有一个总线频测设备(bus frequency device)。如果输入数据不能指定一个或者指定的不存在，那么DeviceFinder找正确的测量设备测量母线的频测。

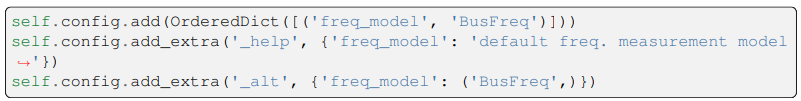


其中，busf是可选的频测设备索引。buss是频测设备所对应的母线索引。

接下来，获取外部参数、变量和service。PSS输出vsout被预分配，但是方程字符串被留在指定模型中。

#### IEEESTModel

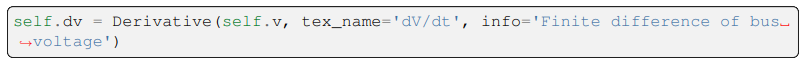
IEEESTModel继承PSSBase类，并在此基础上添加具体的其他部件。在调用完PSSBase构造器之后，IEEESTModel添加配置，以指定频测的model，因为未来将会有很多种类的频测设备model。



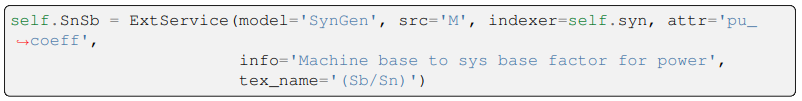
把选中的频测设备model赋给busf，从而DeviceFinder在需要创建新设备的情况下，可以知道要使用哪个model。



接下来，因为母线电压是代数变量，我们使用Derivative去计算有限差分，近似这个导数。

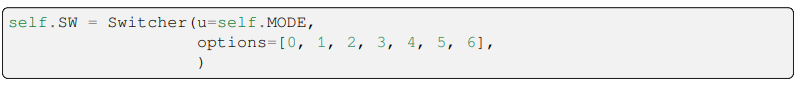


然后，使用ConstService，获取系数，从而把功率从机器基准(machine base)转化为系统基准(system base)。



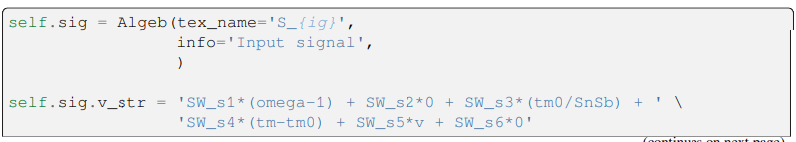
注意到，ExtService获取同步发电机变量M的字段pu\_coeff。因为M是基于机器的功率量，M.pu\_coeff是将机器基准值转化为系统基准值的乘子系数，也就是Sb/Sn。

使用Switcher，将输入模式分解成布尔型标识



其中，输入u是MODE参数，options是接受值的列表。Switcher布尔型数组，s0,s1,……,sN,其中N=len(options) – 1。把0补充到列表options的目的是填充，这样一来，SW\_s1对应的就是MODE 1。这将使得代码更具可读性。

输入信号(input signal) sig是代数变量，





v\_str和e\_str从构造函数中分离了出去，以提高代码可读性。创建分段函数，基于mode，选择正确的初始值和方程。v\_str中的变量必须定义在sig之前，这样就能在sig之前被初始化。显然，omega,tm,v被定义在PSSBase中，因此它们也在sig之前。

下面是最有用的一部分，使用传递函数bloack建模。

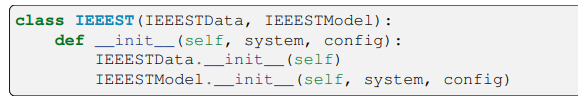
利用几个block描述流程图中的模型。注意：block的输出是block的名字加上\_y。比如，F1的输出，F1\_y，是F2的输入。



注意到末尾处，方程输出被赋值到vsout.e\_str中。这就完成了IEEEST模型的方程定义。

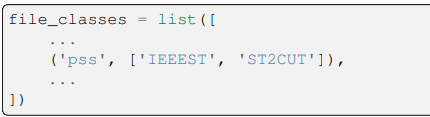
#### 收尾

下面把IEEESTData和IEEESTModel集成到IEEEST中



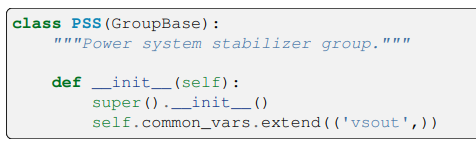
定位并编辑文件andes/models/\_\_init\_\_.py。变量file\_classes是由元组作为元素所组成的列表，每个元组存储了字符串格式的源文件和类名。在每个元组中，第一个元素是models文件夹下.py文件的文件名，第二个元素是由该文件所含有的类的类名组成的列表。

比如，pss所在行，把IEEEST加入到对应的model名称列表中，具体如下



只要 from andes.models.pss import IEEEST是有效的，那么上述的那行语句就是有效的。如果源文件名不存在于file\_classes中的任何一行，你可以在预备（prerequisite）model之后加入。比如，上述的pss那行应该加在励磁机、发电机之后。

最后，找到andes/models/group.py，检查类PSS是否存在。它要与IEEEST的组名匹配上。如果不匹配，那么创建GroupBase的继承类，如下



其中，把‘vsout’加入到common\_vars列表中。PSS组中所有model必须有一个名为vsout的变量，这个变量定义在PSSBase中。

这样就完成了IEEEST model。在开发新model时，使用andes prepare生成数值代码（numerical code）、调试。

# 一些名词

R，C，L: 电阻，电容，电感

a constant impedance 恒定阻抗

DC current rating 支流额定电流

flat start, 平启动，指进行潮流计算时，将节点电压的幅值初始值设置为1，相角初始值设置为0.

power ratio 功率比

Frequency tripping response points 频率跳闸响应点

voltage tripping response points 电压跳闸响应点

torque 扭矩，力矩

turbine 涡轮机

coupling reactance 耦合电抗

Inverter 换流器

Sn表示功率比

储能一次调频响应又称调速下垂控制(Droop Response)或频率响应储备 (Frequency Response Reserve)

caveats 警告

Exciter 励磁机

Frequency measurements 测频

Bus frequency measurement 母线频测

PPS 电力系统稳定器；

urbine governors 水轮机调节器；

jumper 电力线路两段之间的不承受张力的电气连接用短导线

susceptance 电纳

shunt conductance 分流电导

Power rating 额定功率值

AC voltage rating 交流额定电压

rated frequency 额定频率

line reactance 线路电抗

transformer branch tap ratio 电压器变比

Area Control Error model 区域控制误差模型

Center of inertia 惯性中心

delta, （发动机的）转子角度