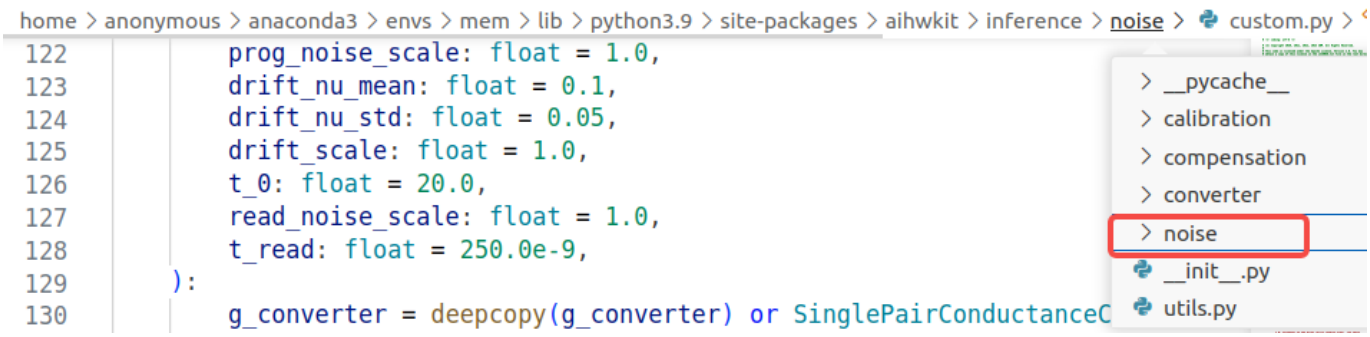
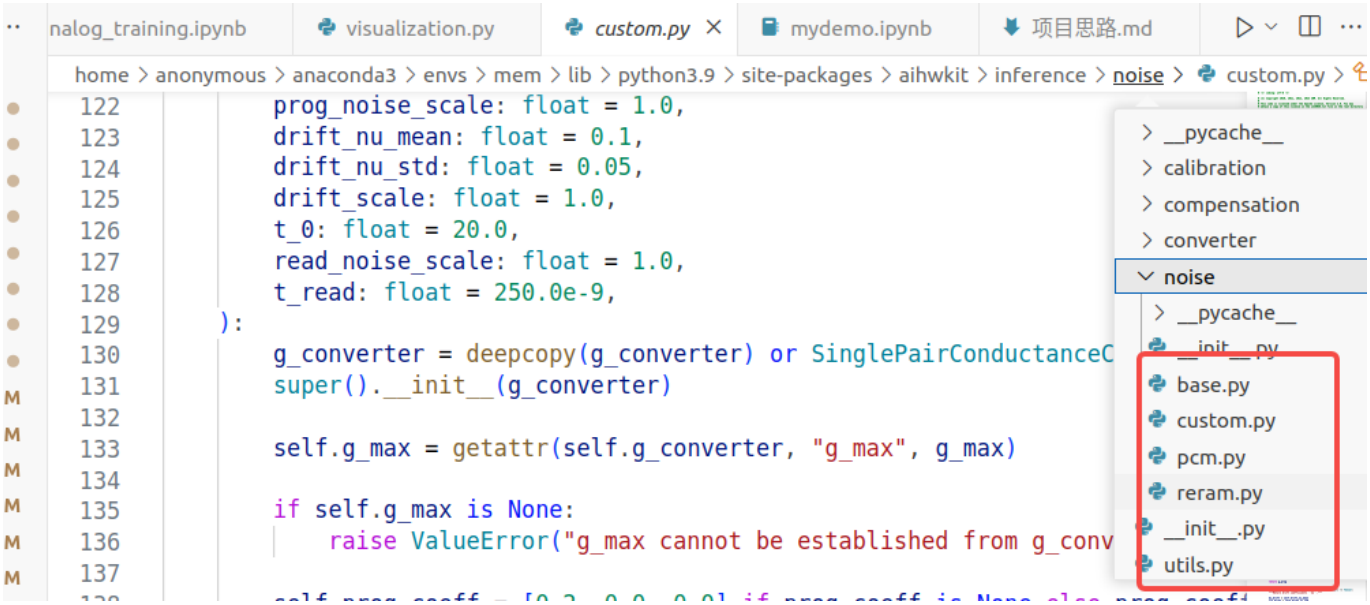
AIHwKit

噪声施加





在 `/home/anonymous/anaconda3/envs/mem/lib/python3.9/site-packages/aihwkit/inference/noise/` 下有4个文件：

- `base.py` 提供的是用于推理的现象学噪声模型的基础类（BaseNoiseModel）。
  - BaseNoiseModel 定义了现象学噪声模型的基本框架和接口。它包括了如何将噪声应用到权重上的一般方法，例如 apply_noise、apply_programming_noise 和 apply_drift_noise。
  - 这个基类也定义了一些方法的框架，如 apply_programming_noise_to_conductance 和 apply_drift_noise_to_conductance，但没有具体实现这些方法（标记为 raise NotImplementedError），意味着这些方法需要在子类中被覆写和具体实现。
- `custom.py` 基类的一个具体实现。
  - StateIndependentNoiseModel。它继承了基础类并提供了具体的噪声模型实现。

    - 在这个实现中，apply_programming_noise_to_conductance 和 apply_drift_noise_to_conductance 等方法被具体定义，包括了如何生成和应用编程噪声和漂移噪声。
    - StateIndependentNoiseModel 还引入了一些额外的属性和方法，用于处理特定于该模型的行为，例如处理编程噪声和漂移噪声的特定参数。

- `pcm.py` 该代码中的 PCMLikeNoiseModel 是基于 BaseNoiseModel （在 base.py 中定义）的另一个具体实现。这个模型特别针对相变记忆（PCM, Phase-Change Memory）设备的噪声特性进行了建模。

    - PCMLikeNoiseModel 继承自 BaseNoiseModel，这意味着它使用了 BaseNoiseModel 中定义的通用框架和方法。
    - PCMLikeNoiseModel 对一些方法进行了重写，以便更精确地模拟PCM设备的特定噪声行为。这包括对编程噪声和漂移噪声的处理方式进行了特定的调整和实现。 与 StateIndependentNoiseModel 相比，PCMLikeNoiseModel 提供了一个更专门化的噪声模型，这个模型是基于真实PCM设备的测量数据来构建的。 这意味着其内部的参数和噪声生成方法是针对PCM设备的特性进行优化的，例如，其漂移系数和程序噪声的计算方法是基于对PCM设备的实际测量数据的分析。

- `reram.py` 这里的 ReRamWan2022NoiseModel 是针对ReRAM设备的噪声模型， 它继承自基础噪声模型类 BaseNoiseModel 并根据ReRAM的特定特性进行了定制化的实现。这个模型基于Wan等在Nature上发表的2022年的研究（详参：https://www.nature.com/articles/s41586-022-04992-8 ），该研究提供了ReRAM设备的噪声和漂移特性的实验数据。

    - 针对ReRAM设备的专门化噪声模型：ReRamWan2022NoiseModel 是一个为了模拟 ReRAM设备的噪声特性而设计的模型。与PCM设备不同，ReRAM设备具有其独特的物理和电子特性，因此需要一个专门的噪声模型来精确模拟这些特性。

    - 和 pcm.py 中的 PCMLikeNoiseModel 一样，ReRamWan2022NoiseModel 也继承自 base.py 中的 BaseNoiseModel。这意味着它继承了基础噪声模型的一些通用方法和属性，但根据ReRAM设备的特性进行了特定的实现和扩展。

    - 噪声模型的实现和特性：

        - ReRamWan2022NoiseModel 实现了针对ReRAM设备的编程噪声和漂移噪声。与PCM模型不同，ReRAM模型可能使用了不同的数学方法和公式来模拟这些噪声，这些方法是基于对ReRAM设备特定行为的实验观察。
        - 模型中使用了四阶多项式来逼近目标电导的偏差，并根据不同的时间点（如1秒、1天、2天）调整噪声特性。

custom.py 噪声详解

主要包括了三种噪声类型：编程噪声（Programming Noise）、漂移噪声（Drift Noise）和读取噪声（Read Noise）。

- 编程噪声（Programming Noise）：在编程阶段，这种噪声被加到目标电导值上。它被建模为具有依赖于电导水平的方差的高斯噪声，并通过多项式进行缩放。可以调整多项式系数，prog_noise_scale 控制噪声的整体水平。
    - 施加方式：在apply_programming_noise_to_conductance方法中实现。编程噪声是通过向目标电导值（g_target）添加高斯噪声来施加的。噪声的强度是根据目标电导值和多项式系数（self.prog_coeff）来确定的。
    - 采样分布：使用高斯分布来采样，其中均值为目标电导值，标准差（sig_prog）由上述多项式计算得到。
    - 代码在第178行：

      ```
      g_prog = g_target + self.prog_noise_scale * sig_prog * randn_like(g_target)
      ```

- 漂移噪声（Drift Noise）：随着时间的推移，电导值会发生漂移，这种漂移在这里被建模。漂移的特点是时间的幂律依赖性，其中的系数（drift_nu_mean, drift_nu_std）决定了漂移的程度。
    - 施加方式：在generate_drift_coefficients和apply_drift_noise_to_conductance方法中实现。漂移噪声通过改变编程后的电导值（g_prog）来模拟电导值随时间的漂移效果。漂移系数（nu_drift）是根据漂移的平均值（drift_nu_mean）和标准差（drift_nu_std）生成的，再乘以漂移比例因子（drift_scale）。
    - 采样分布：漂移系数的生成采用高斯分布进行采样，然后取其绝对值并排除负值。
    - 代码在第188行：

      ```
      nu_drift = torch_abs(mu_drift + sig_drift * randn_like(g_target)).clamp(min=0.0)
      ```

- 读取噪声（Read Noise）：在读取电导值时，会加入另一种噪声。这种噪声是基于时间的函数，并且与电导值成比例，由read_noise_scale 控制。
    - 施加方式：在apply_drift_noise_to_conductance方法中实现。在读取漂移后的电导值（g_drift）时添加读取噪声。这种噪声的强度依赖于漂移后的电导值和时间，通过对数函数计算得出。
    - 采样分布：使用高斯分布进行采样，标准差是基于对数函数（涉及t_read和t）计算得到的。

- 代码在第208行：

```
g_final = g_drift + torch_abs(
g_drift / self.g_max
) * self.read_noise_scale * sig_noise * randn_like(g_drift)
```

pcm.py 噪声详解

在 PCMLikeNoiseModel 中，噪声的类型和 `custom.py` 中的一样，不同在于，噪声是基于真实硬件（这里是相变存储器，PCM）设备的实际测量数据来定义的。噪声的形式仍然被设定为高斯分布。

也就是说，PCMLikeNoiseModel 中的噪声模型是基于实际测量的PCM设备数据来构建的。它通过高斯分布来生成噪声，但是这些噪声的特性（例如标准差）是根据电导值、时间以及其他参数动态计算的，从而模拟PCM设备的物理特性和行为。只是在高斯分布参数的给定上结合了真实忆阻器的特性。具体地：

- 编程噪声（Programming Noise）：
  - 这种噪声是在目标电导值 (g_target) 上添加的，其强度由一个二次多项式决定，该多项式考虑了电导值相对于最大电导值 (g_max) 的比例。这在 apply_programming_noise_to_conductance 方法中实现。
  - 分布：编程噪声采用高斯（正态）分布进行采样，其均值为目标电导值，标准差（sig_prog）由上述多项式确定。
  - 参数：
    - 标准差：通过一个二次多项式计算得到，这个多项式考虑了电导值相对于最大电导值 (g_max) 的比例。多项式的系数 (prog_coeff) 是根据实际的PCM设备数据来确定的。
    - 均值：通常设为目标电导值（g_target），这意味着噪声是在目标电导值周围对称分布的。
  - 代码实现：在 apply_programming_noise_to_conductance 方法中，使用 randn_like(g_target) 生成高斯噪声，然后乘以计算出的标准差和比例因子 prog_noise_scale。
- 漂移噪声（Drift Noise）：
  - 这种噪声是对编程后电导值 (g_prog) 的修改，以模拟电导值随时间的漂移。漂移系数是基于对PCM设备的实际测量数据进行计算的。这在 generate_drift_coefficients 和 apply_drift_noise_to_conductance 方法中实现。
  - 分布：漂移系数的生成采用了高斯分布进行采样，并进行了特定的计算以模拟PCM设备的漂移特性。
  - 参数：

        - 标准差：基于对PCM设备的实际测量数据分析确定。漂移系数 (nu_drift) 是通过对电导值的对数进行线性拟合得出的，然后通过高斯分布采样来确定。
        - 均值：通常为0，表示噪声是关于漂移后电导值的中心对称分布。
    - 代码实现：在 generate_drift_coefficients 和 apply_drift_noise_to_conductance 方法中。
- 读取噪声（Read Noise）：
    - 这种噪声是在读取漂移后的电导值 (g_drift) 时添加的。其强度取决于漂移后的电导值和时间，通过一个对数函数来计算。这在 apply_drift_noise_to_conductance 方法中实现。
    - 分布：读取噪声采用高斯分布进行采样，其标准差是基于特定对数函数计算得出的。
    - 参数：
        - 标准差：通过特定的对数函数计算得出，该函数考虑了时间参数和电导值。这个对数函数反映了1/f噪声的特性。
        - 均值：通常为0，意味着噪声是关于读取值的中心对称分布。
    - 代码实现：在 apply_drift_noise_to_conductance 方法中，使用 randn_like(g_prog) 生成高斯噪声，然后乘以计算出的标准差和比例因子 read_noise_scale。

reram.py 噪声详解

在 `reram.py` 中的 ReRamWan2022NoiseModel 定义了针对ReRAM（电阻式随机存取存储器）设备的噪声。这个模型的噪声是基于对ReRAM设备的实验数据进行分析和逼近得到的。具体地，该模型如何定义噪声和采取的分布如下：

- 噪声定义：
    - 噪声是基于电导值的偏差，并且与目标电导值（g_target）有关。这种偏差是通过四阶多项式来逼近的，多项式的系数（coeff_dic）是从实验数据中得出的。
    - 不同于前面所有其他的噪声脚本，reram.py的噪声只有两类：
        - 编程噪声（Programming Noise）：通过一个多项式函数来模拟编程噪声，并添加到目标电导率上。
        - 累积噪声（Accumulated Noise）：根据时间推断出的ReRAM设备的长期漂移行为，以多项式形式表示，然后添加到目标电导率上。 两种噪声作用的阶段有所不同：编程噪声是在初始阶段应用的，而漂移噪声则模拟了随时间累积的噪声。
    - 关于多项式方法：
        - 多项式用于建立电导值（g_target）与噪声强度之间的关系。具体来说，这个关系是通过电导值相对于其最大值（g_max）的比例来定义的。

- 通常，这种多项式是一个四阶多项式，形式为

$$\sum_{i=0}^{4} c_i \left(\frac{g_t}{g_{\max}} \right)^i$$

其中，c_i 是多项式的系数，g_t 是目标电导值，$g_{\max}$ 是电导的最大值。

这种形式允许噪声强度随着电导值的变化而非线性地变化，更准确地反映真实设备的物理特性。

多项式系数的决定：通常是从实验数据中拟合得出的。在 ReRamWan2022NoiseModel 的例子中，这些系数是根据ReRAM设备的实验测量数据来确定的。这些系数可以代表在特定时间点（例如1秒、1天、2天）的噪声特性，不同时间点的噪声特性可能由不同的系数集合来描述。

- 噪声分布：
  - 噪声采用高斯（正态）分布进行采样。具体地，噪声是在目标电导值周围通过加性高斯噪声实现的，其标准差由上述的四阶多项式决定。
  - 在 apply_programming_noise_to_conductance 和 apply_drift_noise_to_conductance 方法中，使用 randn_like(g_target) 生成高斯噪声，然后乘以由多项式计算出的标准差和噪声比例因子（noise_scale）。
- 噪声施加：
  - 编程噪声：在编程阶段应用，模拟初始电导值的变化。
  - 漂移噪声：模拟随时间累积的噪声，根据推断时间（t_inference）使用不同的多项式系数来计算噪声。