Qiskit Experiments: Analysis

https://qiskit-extensions.github.io/qiskit-experiments

曾祥东

2024 年 3 月 22 日

T1 实验

```
import numpy as np
from qiskit_experiments.library import T1
from qiskit_ibm_runtime.fake_provider import FakePerth
from qiskit_aer import AerSimulator

backend = AerSimulator.from_backend(FakePerth())

qubit0_t1 = FakePerth().qubit_properties(0).t1
delays = np.arange(1e-6, 3 * qubit0_t1, 3e-5)

exp = T1(physical_qubits=(0,), delays=delays)

exp_data = exp.run(backend=backend)

display(exp_data.figure(0))
```

Rabi 实验

```
import numpy as np
from qiskit import pulse
from qiskit.circuit import Parameter
from qiskit_experiments.test.pulse_backend import SingleTransmonTestBackend
from qiskit_experiments.data_processing import DataProcessor, nodes
from qiskit_experiments.library import Rabi

with pulse.build() as sched:
    pulse.play(pulse.Gaussian(160, Parameter("amp"), sigma=40), pulse.DriveChannel(0))

backend = SingleTransmonTestBackend(seed=100)
exp = Rabi(physical_qubits=(0,), backend=backend, schedule=sched, amplitudes=np.linspace(-0.1, 0.1, 21))

# IQ 数据处理器
data_nodes = [nodes.SVD(), nodes.AverageData(axis=1), nodes.MinMaxNormalize()]
iq_processor = DataProcessor("memory", data_nodes)
exp.analysis.set_options(data_processor=iq_processor)

exp_data = exp.run(meas_level=1, meas_return="single").block_for_results()

display(exp_data.figure(0))
```

数据处理

```
processor = DataProcessor(
    input_key="memory",
    data_actions=[Node1(), Node2(), ...]
)
out_data = processor(in_data)
```

- 将后端返回的数据转换为可分析的格式
- 可以在 `exp.analysis` 中设置数据处理器
- `input_key`：输入数据标签
- `data_actions`：处理步骤，称为节点 (node)

数据级别

- Level 0: 后端返回的原始数据
- Level 1: 核方法提取出的 IQ 数据 (kerneled data)
- Level 2: 分类 / 计数数据 (discriminated data)

```
DataAction
 ├─ AverageData
 ├─ BasisExpectationValue
 ├─ MinMaxNormalize
 ├─ MemoryToCounts
 ├─ TrainableDataAction
 │   └─ SVD
 ├─ CountsAction
 │   ├─ MarginalizeCounts
 │   └─ Probability
 ├─ IQPart
 │   ├─ ToReal
 │   ├─ ToImag
 │   └─ ToAbs
 ├─ RestlessNode
 │   ├─ RestlessToCounts
 │   └─ RestlessToIQ
 └─ DiscriminatorNode

BaseDiscriminator  # scikit-learn 分类器
 ├─ SkLDA
 └─ SkQDA
```

不确定度处理：`uncertainties` 包

- 基本用法：

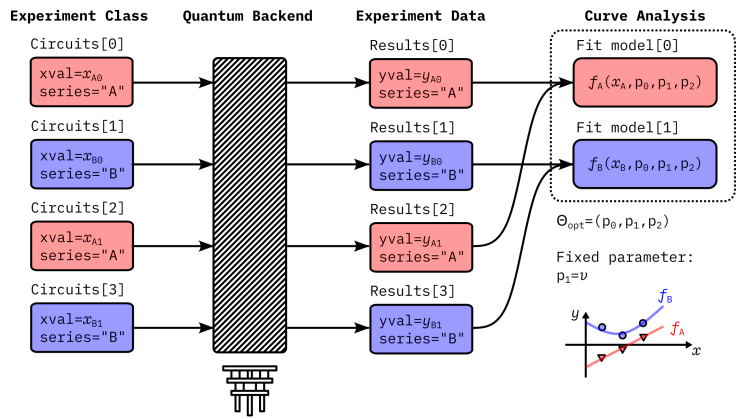
```
>>> from uncertainties import ufloat, umath, unumpy as unp
>>> x = ufloat(1, 0.1)
>>> umath.sin(2 * x)
0.9092974268256817+/-0.08322936730942848
>>> arr = unp.uarray([1, 2], [0.01, 0.002])
>>> arr @ arr
5.0+/-0.021540659228538015
```

- `DataAction` 中的使用：

```
class AverageData(DataAction):
    def _process(self, data: np.ndarray) -> np.ndarray:
        reduced_array = np.mean(data, axis=self._axis)
        nominals = unp.nominal_values(reduced_array)
        errors = unp.std_devs(reduced_array)
        return unp.uarray(nominals, errors)
```

- 技术细节：
  - 不确定度：由概率分布定义（`Variable`、`AffineScalarFunc` 类）
  - 误差传播：自动微分

曲线分析




```python
class CurveAnalysis:
    def _run_analysis(self, experiment_data):
        self._initialize(experiment_data)
        table = self._run_data_processing(experiment_da
        table = self._format_data(table)
        formatted_subset = table.filter(category=self.c
        fit_data = self._run_curve_fit(formatted_subset
        ...
```

- 曲线分析类

```
BaseCurveAnalysis
 ├─ CurveAnalysis
 │   ├─ DecayAnalysis
 │   │   ├─ T1Analysis
 │   │   ├─ T1KerneledAnalysis
 │   │   └─ T2HahnAnalysis
 │   └─ DampedOscillationAnalysis
 │       └─ T2RamseyAnalysis
 └─ CompositeCurveAnalysis
     └─ CrossResonanceHamiltonianAnalysis
```

- 数据容器：`ScatterTable`
  - 基于 `pandas.DataFrame`

曲线拟合：定义模型

```
import lmfit
import numpy as np

def exp_decay(x, amp, alpha, base):
    return amp * np.exp(-alpha * x) + base

models = [
    lmfit.Model(func=exp_decay),
    lmfit.models.ExpressionModel(
        expr="amp * cos(2 * pi * freq * x + phi) + base",
        name="my_experiment",
    ),
]
```

```
class MyAnalysis(CurveAnalysis):

    @classmethod
    def _default_options(cls) -> Options:
        options = super()._default_options()
        options.fixed_parameters = { "amp": 3.0 }  # 固定参数
        return options
```

曲线拟合：猜测初始参数

```
class GaussianAnalysis(curve.CurveAnalysis):

    def _generate_fit_guesses(self, user_opt, curve_data):
        max_abs_y, _ = curve.guess.max_height(curve_data.y, absolute=True)
        user_opt.bounds.set_if_empty(
            a=(-2 * max_abs_y, 2 * max_abs_y),
            sigma=(0, np.ptp(curve_data.x)),
            freq=(min(curve_data.x), max(curve_data.x)),
            b=(-max_abs_y, max_abs_y),
        )
        user_opt.p0.set_if_empty(b=curve.guess.constant_spectral_offset(curve_data.y))
        y_ = curve_data.y - user_opt.p0["b"]
        _, peak_idx = curve.guess.max_height(y_, absolute=True)
        fwhm = curve.guess.full_width_half_max(curve_data.x, y_, peak_idx)
        user_opt.p0.set_if_empty(
            a=curve_data.y[peak_idx] - user_opt.p0["b"],
            freq=curve_data.x[peak_idx],
            sigma=fwhm / np.sqrt(8 * np.log(2)),
        )

        return user_opt
```

曲线拟合：评估结果

```
class GaussianAnalysis(curve.CurveAnalysis):

    def _evaluate_quality(self, fit_data):
        freq_increment = np.mean(np.diff(fit_data.x_data))

        fit_a = fit_data.ufloat_params["a"]
        fit_b = fit_data.ufloat_params["b"]
        fit_freq = fit_data.ufloat_params["freq"]
        fit_sigma = fit_data.ufloat_params["sigma"]

        snr = abs(fit_a.n) / np.sqrt(abs(np.median(fit_data.y_data) - fit_b.n))
        fit_width_ratio = fit_sigma.n / np.ptp(fit_data.x_data)

        criteria = [
            fit_data.x_range[0] <= fit_freq.n <= fit_data.x_range[1],
            1.5 * freq_increment < fit_sigma.n,
            fit_width_ratio < 0.25,
            0 < fit_data.reduced_chisq < 3,
            curve.utils.is_error_not_significant(fit_sigma),
            snr > 2,
        ]

        return "good" if all(criteria) else "bad"
```

可视化

- Drawer：`MplDrawer`
  - 基于 `matplotlib`
- Plotter：`CurvePlotter`、`IQPlotter`