mitiq

Release 0.1.0

Tech Team @ Unitary Fund

Mar 19, 2020

Contents:

1 Change Log **1**

1.1 Version 0.1.0 (Date) 1

2 Users Guide **3**

2.1 Overview of mitiq 3

2.2 Zero Noise Extrapolation 3

3 Indices and tables **7**

Python Module Index **9**

Index **11**

CHAPTER 1

Change Log

1.1 Version 0.1.0 (Date)

- **Initial release.**

CHAPTER 2

Users Guide

2.1 Overview of mitiq

Welcome to *mitiq* Users Guide.

Mitiq is an open source toolkit for implementing error mitigation techniques on most current intermediate-scale quantum computers.

The library allows to postprocess results from quantum circuits with both analog and digital techniques, interfacing with a variety of quantum circuit libraries.

2.2 Zero Noise Extrapolation

2.2.1 Introduction

Zero noise extrapolation (ZNE) was introduced concurrently in Ref. [1] and [2]. With *mitiq.zne* module it is possible to extrapolate what the expected value would be without noise. This is done by first setting up one of the key objects in *mitiq*, which is a `mitiq.Factory` object.

2.2.2 Importing Quantum Circuits

mitiq allows one to flexibly import and export quantum circuits from other libraries. Here is an example:

```
>>> from mitiq import Factory
```

This is the top level module from which functions and classes of Mitiq can be directly imported.

`mitiq.`**`version`**`()`

Returns the Mitiq version number.

Contains all the main classes corresponding to different zero-noise extrapolation methods.

class mitiq.factories.**BatchedFactory**(*scalars: Iterable[float]*)
Abstract class of a non-adaptive Factory.

This is initialized with a given batch of scaling factors ("scalars"). The "self.next" method trivially iterates over the elements of "scalars" in a non-adaptive way. Convergence is achieved when all the correpsonding expectation values have been measured.

Specific (non-adaptive) zero-noise extrapolation algorithms can be derived from this class by overriding the "self.reduce" and (if necessary) the "__init__" method.

is_converged() → bool
Returns True if all needed expectation values have been computed, else False.

next() → float
Returns the next noise level to execute a circuit at.

class mitiq.factories.**ExpFactory**(*scalars: Iterable[float], asymptote: Optional[float] = None*)
Factory object implementing a zero-noise extrapolation algotrithm assuming an exponential ansatz y(x) = a + b * exp(-c * x), with c > 0.

If the asymptotic value (y(x->inf) = a) is known, a linear fit with respect to z(x) := log[sing(b) (y(x) - a)] is used. Otherwise, a non-linear fit of y(x) is perfomed.

reduce() → float
Returns the zero-noise limit, assuming an exponential ansatz: y(x) = a + b * exp(-c * x), with c > 0.

class mitiq.factories.**Factory**
Abstract class designed to adaptively produce a new noise scaling parameter based on a historical stack of previous noise scale parameters ("self.instack") and previously estimated expectation values ("self.outstack").

Specific zero-noise extrapolation algorithms, adaptive or non-adaptive, are derived from this class. A Factory object is not supposed to directly perform any quantum computation, only the classical results of quantum experiments are processed by it.

is_converged() → bool
Returns True if all needed expectation values have been computed, else False.

next() → float
Returns the next noise level to execute a circuit at.

push(*instack_val: float*, *outstack_val: float*) → None
Appends "instack_val" to "self.instack" and "outstack_val" to "self.outstack". Each time a new expectation value is computed this method should be used to update the internal state of the Factory.

reduce() → float
Returns the extrapolation to the zero-noise limit.

class mitiq.factories.**LinearFactory**(*scalars: Iterable[float]*)
Factory object implementing a zero-noise extrapolation algotrithm based on a linear fit.

reduce() → float
Determines, with a least squared method, the line of best fit associated to the data points. The intercept is returned.

class mitiq.factories.**PolyExpFactory**(*scalars: Iterable[float], order: int, asymptote: Optional[float] = None*)
Factory object implementing a zero-noise extrapolation algotrithm assuming an (almost) exponential ansatz with a non linear exponent, i.e.:

y(x) = a + s * exp(z(x)), where z(x) is a polynomial of a given order.

The parameter "s" is a sign variable which can be either 1 or -1, corresponding to decreasing and increasing exponentials, respectively. The parameter "s" is automatically deduced from the data.

If the asymptotic value (y(x->inf) = a) is known, a linear fit with respect to z(x) := log[s(y(x) - a)] is used. Otherwise, a non-linear fit of y(x) is perfomed.

reduce() → float

Returns the zero-noise limit, assuming an exponential ansatz: y(x) = a + s * exp(z(x)), where z(x) is a polynomial of a given order. The parameter "s" is a sign variable which can be either 1 or -1, corresponding to decreasing and increasing exponentials, respectively. The parameter "s" is automatically deduced from the data. It is also assumed that z(x-->inf)=-inf, such that y(x-->inf)-->a.

static static_reduce(*instack: List[float], outstack: List[float], asymptote: Optional[float], order: int, eps: float = 1e-09*) → float

Determines the zero-noise limit, assuming an exponential ansatz: y(x) = a + s * exp(z(x)), where z(x) is a polynomial of a given order.

The parameter "s" is a sign variable which can be either 1 or -1, corresponding to decreasing and increasing exponentials, respectively. The parameter "s" is automatically deduced from the data.

It is also assumed that z(x-->inf)=-inf, such that y(x-->inf)-->a.

If asymptote is None, the ansatz y(x) is fitted with a non-linear optimization. Otherwise, a linear fit with respect to z(x) := log(sign * (y(x) - asymptote)) is performed.

This static method is equivalent to the "self.reduce" method of PolyExpFactory, but can be called also by other factories which are particular cases of PolyExpFactory, e.g., ExpFactory.

Parameters

- **instack** -- x data values.
- **outstack** -- y data values.
- **asymptote** -- y(x->inf).
- **order** -- extrapolation order.
- **eps** -- epsilon to regularize log(sign (instack - asymptote)) when the argument is to close to zero or negative.

class mitiq.factories.**PolyFactory**(*scalars: Iterable[float], order: int*)

Factory object implementing a zero-noise extrapolation algotrithm based on a polynomial fit. Note: RichardsonFactory and LinearFactory are special cases of PolyFactory.

reduce() → float

Determines with a least squared method, the polynomial of degree equal to "self.order" which optimally fits the input data. The zero-noise limit is returned.

static static_reduce(*instack: List[float], outstack: List[float], order: int*) → float

Determines with a least squared method, the polynomial of degree equal to 'order' which optimally fits the input data. The zero-noise limit is returned.

This static method is equivalent to the "self.reduce" method of PolyFactory, but can be called also by other factories which are particular cases of PolyFactory, e.g., LinearFactory and RichardsonFactory.

class mitiq.factories.**RichardsonFactory**(*scalars: Iterable[float]*)

Factory object implementing Richardson's extrapolation.

reduce() → float

Returns the Richardson's extrapolation to the zero-noise limit.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

mitiq, 3
mitiq.factories, 3

Index

B

BatchedFactory (*class in mitiq.factories*), 3

E

ExpFactory (*class in mitiq.factories*), 4

F

Factory (*class in mitiq.factories*), 4

I

is_converged() (*mitiq.factories.BatchedFactory method*), 4
is_converged() (*mitiq.factories.Factory method*), 4

L

LinearFactory (*class in mitiq.factories*), 4

M

mitiq (*module*), 3
mitiq.factories (*module*), 3

N

next() (*mitiq.factories.BatchedFactory method*), 4
next() (*mitiq.factories.Factory method*), 4

P

PolyExpFactory (*class in mitiq.factories*), 4
PolyFactory (*class in mitiq.factories*), 5
push() (*mitiq.factories.Factory method*), 4

R

reduce() (*mitiq.factories.ExpFactory method*), 4
reduce() (*mitiq.factories.Factory method*), 4
reduce() (*mitiq.factories.LinearFactory method*), 4
reduce() (*mitiq.factories.PolyExpFactory method*), 5
reduce() (*mitiq.factories.PolyFactory method*), 5
reduce() (*mitiq.factories.RichardsonFactory method*), 5
RichardsonFactory (*class in mitiq.factories*), 5

S

static_reduce() (*mitiq.factories.PolyExpFactory static method*), 5
static_reduce() (*mitiq.factories.PolyFactory static method*), 5

V

version() (*in module mitiq*), 3