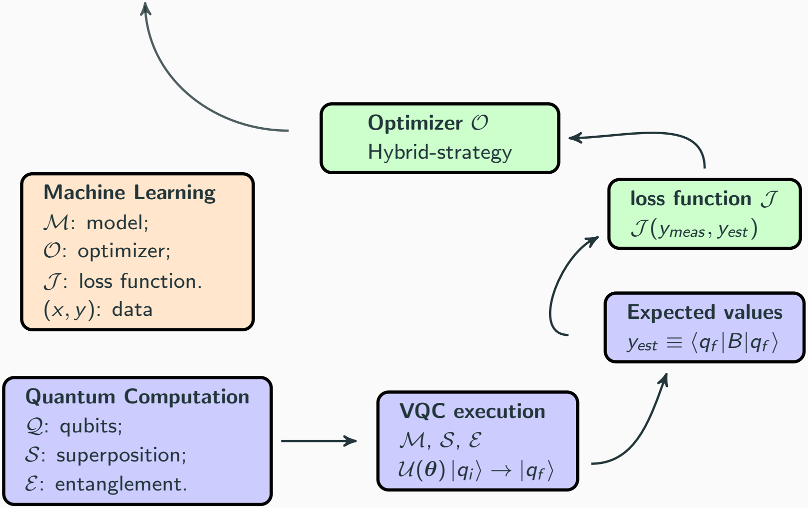
Density estimation via quantum adiabatic computing

Based on arXiv:2303.11346

Matteo Robbiati, Juan Manuel Cruz-Martinez, Stefano Carrazza

20 April 2023

A screenshot of Quantum Machine Learning (QML)



Introduction

Our goal

- QML to tackle **Monte Carlo** Integration (MCI).

Our goal

- QML to tackle **Monte Carlo** Integration (MCI).
- Let's consider the integrand $g(x)$ and a dataset Ω sampled from a distribution $\rho(x)$. We are interested in calculating:

$$E\left[g(x)\right] = \int_\Omega g(x)\rho(x)\mathrm{d}x. \tag{1}$$

Our goal

- QML to tackle **Monte Carlo** Integration (MCI).
- Let's consider the integrand $g(x)$ and a dataset Ω sampled from a distribution $\rho(x)$. We are interested in calculating:

$$E\left[g(x)\right] = \int_{\Omega} g(x)\rho(x)\mathrm{d}x. \tag{1}$$

- Thus we need:

Our goal

- QML to tackle **Monte Carlo** Integration (MCI).
- Let's consider the integrand $g(x)$ and a dataset Ω sampled from a distribution $\rho(x)$. We are interested in calculating:

$$E\left[g(x)\right] = \int_\Omega g(x)\rho(x)\mathrm{d}x. \tag{1}$$

- Thus we need:

a sample of data Ω to be used for evaluating the integral;

Our goal

- QML to tackle **Monte Carlo** Integration (MCI).
- Let's consider the integrand $g(x)$ and a dataset Ω sampled from a distribution $\rho(x)$. We are interested in calculating:

$$E\big[g(x)\big] = \int_{\Omega} g(x)\rho(x)\mathrm{d}x. \tag{1}$$

- Thus we need:
  - a sample of data Ω to be used for evaluating the integral;
  - a way for estimating the Probability Density Function (PDF) value for each given data $\rho(x)$.

Our goal

- QML to tackle **Monte Carlo** Integration (MCI).
- Let's consider the integrand $g(x)$ and a dataset Ω sampled from a distribution $\rho(x)$. We are interested in calculating:

$$E\left[g(x)\right] = \int_{\Omega} g(x)\rho(x)\mathrm{d}x. \tag{1}$$

- Thus we need:
  - a sample of data Ω to be used for evaluating the integral;
  - a way for estimating the Probability Density Function (PDF) value for each given data $\rho(x)$.

In this work

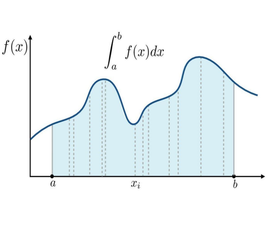
We focus on to find a **density estimation** strategy.

Motivation in HEP

- Several HEP deployments exist:

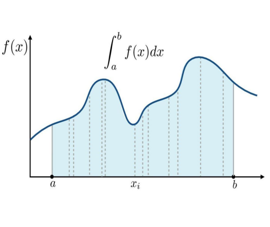
Motivation in HEP

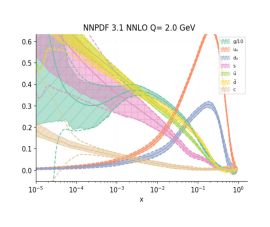
- Several HEP deployments exist:
  - **Monte Carlo** integration requires density estimation techniques;



Motivation in HEP

- Several HEP deployments exist:
  - **Monte Carlo** integration requires density estimation techniques;
  - **Parton density function** estimation (TH already worked on this[1]);

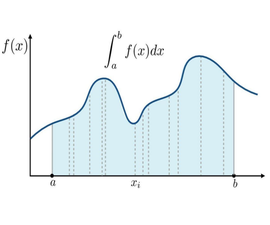


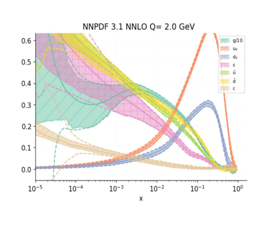


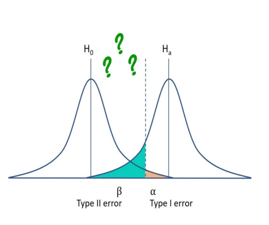
[1] arXiv:2011.13934

Motivation in HEP

- Several HEP deployments exist:
  - **Monte Carlo** integration requires density estimation techniques;
  - **Parton density function** estimation (TH already worked on this[1]);
  - **Anomaly detection**: if a PDF is known and punctually evaluable we can use this for hypotesis testing.







[1] arXiv:2011.13934

Outline

- We are going to follow these steps:

Outline

- We are going to follow these steps:
  - we target the **Cumulative Density Function** (CDF) of a sampled Ω;

Outline

- We are going to follow these steps:
  - we target the **Cumulative Density Function** (CDF) of a sampled Ω;
  - we define a new **Quantum Adiabatic Machine Learning** (QAML) strategy for tackling 1d fitting problems;

Outline

- We are going to follow these steps:
  - we target the **Cumulative Density Function** (CDF) of a sampled Ω;
  - we define a new **Quantum Adiabatic Machine Learning** (QAML) strategy for tackling 1d fitting problems;
  - we fit the CDF via QAML;

Outline

- We are going to follow these steps:
  - we target the **Cumulative Density Function** (CDF) of a sampled Ω;
  - we define a new **Quantum Adiabatic Machine Learning** (QAML) strategy for tackling 1d fitting problems;
  - we fit the CDF via QAML;
  - we use parameter-shift rules for calculating the PDF as derivative of the CDF;

Outline

- We are going to follow these steps:

- we target the **Cumulative Density Function** (CDF) of a sampled Ω;
- we define a new **Quantum Adiabatic Machine Learning** (QAML) strategy for tackling 1d fitting problems;
- we fit the CDF via QAML;
- we use parameter-shift rules for calculating the PDF as derivative of the CDF;
- we validate the procedure on some test cases.

Outline

- We are going to follow these steps:

- we target the **Cumulative Density Function** (CDF) of a sampled Ω;
- we define a new **Quantum Adiabatic Machine Learning** (QAML) strategy for tackling 1d fitting problems;
- we fit the CDF via QAML;
- we use parameter-shift rules for calculating the PDF as derivative of the CDF;
- we validate the procedure on some test cases.

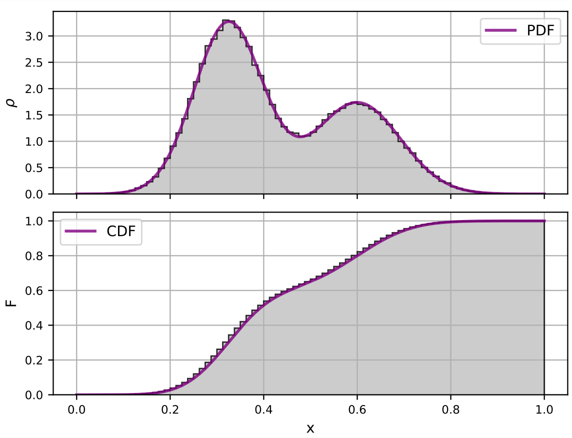
```
import qibo

# in some boxes like this
# we will show how to implement the QAML strategy
```

Code here: qiboteam/adiabatic-fit

METHOD: CDF fit with a VQC

Simple case: 1d data in $\mathcal{D} = [0, 1]$.

➜ Each $x \in \Omega$ can be labeled with its empirical CDF[2] value $F(x)$, which is related to the PDF value via $\rho(x) = \frac{dF(x)}{dx}$.



[2] **Cumulative Density Function:** after sorting the data, $F(x)$ is calculated by counting how many elements are smaller then the target one.

Variational Quantum Circuits (VQC) as regression model

- It is great to use a VQC as model for estimating F:

Variational Quantum Circuits (VQC) as regression model

- It is great to use a VQC as model for estimating F:

$$\hat{F}(x;\boldsymbol{\theta}) \equiv \langle\psi_i|\mathcal{C}^\dagger(x;\boldsymbol{\theta})\,\hat{\mathcal{O}}\,\mathcal{C}(x;\boldsymbol{\theta})|\psi_i\rangle\,, \tag{2}$$

where $\mathcal{C}(x;\boldsymbol{\theta})$, $\mathcal{O}$ and ψ_i are the respectively the VQC, a target observable and the initial state on which we apply $\mathcal{C}$.

Variational Quantum Circuits (VQC) as regression model

- It is great to use a VQC as model for estimating F:

$$\hat{F}(x;\boldsymbol{\theta}) \equiv \langle\psi_i|\mathcal{C}^\dagger(x;\boldsymbol{\theta})\,\hat{\mathcal{O}}\,\mathcal{C}(x;\boldsymbol{\theta})|\psi_i\rangle\,, \tag{2}$$

where $\mathcal{C}(x;\boldsymbol{\theta})$, $\mathcal{O}$ and ψ_i are the respectively the VQC, a target observable and the initial state on which we apply $\mathcal{C}$.

- We know how to derivate circuits, *e.g.* using the Parameter Shift Rule (PSR)[3], thanks to which we can calculate:

$$\partial_\mu \hat{F} = r\big[\hat{F}(\mu^+) - \hat{F}(\mu^-)\big]. \tag{3}$$

[3] arXiv:1811.11184

Variational Quantum Circuits (VQC) as regression model

- It is great to use a VQC as model for estimating F:

$$\hat{F}(x;\boldsymbol{\theta}) \equiv \langle\psi_i|\mathcal{C}^\dagger(x;\boldsymbol{\theta})\,\hat{\mathcal{O}}\,\mathcal{C}(x;\boldsymbol{\theta})|\psi_i\rangle\,, \tag{2}$$

where $\mathcal{C}(x;\boldsymbol{\theta})$, $\mathcal{O}$ and ψ_i are the respectively the VQC, a target observable and the initial state on which we apply $\mathcal{C}$.

- We know how to derivate circuits, *e.g.* using the Parameter Shift Rule (PSR)[3], thanks to which we can calculate:

$$\partial_\mu \hat{F} = r\big[\hat{F}(\mu^+) - \hat{F}(\mu^-)\big]. \tag{3}$$

- In case of rotational gates[4] $\exp\left\{-i\mu\hat{\sigma}\right\}$ we have $r = 0.5$, $\mu^\pm = \mu \pm s$ and $s = \pi/2$.

[3] arXiv:1811.11184

[4] arXiv:1803.00745

PSR

- We can upload x into a rotation angle and calculate $\partial_x \hat{F}$.

```python
from qibo import models, gates, hamiltonians, derivative

# here you define a parametric circuit c as explained during the tutorials
# in which you upload x into the p-th rotation angle, as theta = x*PAR
# then you define an observable
h = hamiltonians.Z(nqubits=1)

# derivative with respect to x of < h >
derivative = derivative.parameter_shift(
    circuit = c,                       # parametric circuit
    hamiltonian = h,                   # target observable
    parameter_index = p,               # parameter index
    initial_state = initial_state,     # initial state (before applying c)
    scale_factor = scale_factor)       # in this case PAR
```

PSR

- We can upload x into a rotation angle and calculate $\partial_x \hat{F}$.

```python
from qibo import models, gates, hamiltonians, derivative

# here you define a parametric circuit c as explained during the tutorials
# in which you upload x into the p-th rotation angle, as theta = x*PAR
# then you define an observable
h = hamiltonians.Z(nqubits=1)

# derivative with respect to x of < h >
derivative = derivative.parameter_shift(
    circuit = c,                        # parametric circuit
    hamiltonian = h,                    # target observable
    parameter_index = p,                # parameter index
    initial_state = initial_state,      # initial state (before applying c)
    scale_factor = scale_factor)        # in this case PAR
```

In a nutshell

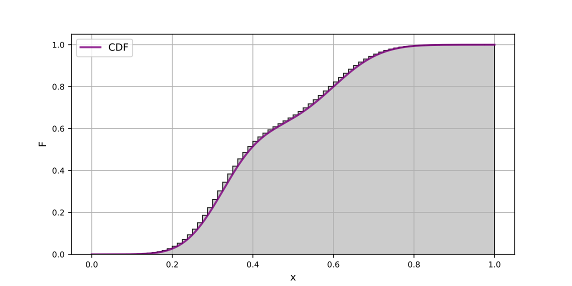
We estimate the CDF using a VQC and we derivate it with the PSR for calculating the PDF.

Two problems

➜ We tried to fit CDFs using a VQC as QML model, but we had two problems:

- by encoding x into the rotation angles, our results often did not retain a **strictly increasing monotony**.
- we need to fix $\hat{F}(0) = 0$ and $\hat{F}(1) = 1$, so we need to manipulate $\hat{F}$ in order to follow these constraints.

➜ These conditions are needed if we deal with CDFs.



METHOD: Quantum Adiabatic ML

Adiabatic Evolution (AE) as QML model

➔ Let's use an Adiabatic Evolution (AE) as model:

$$H_{ad} = H_0\big[1 - s(\tau;\boldsymbol{\theta})\big] + s(\tau;\boldsymbol{\theta})H_1, \tag{4}$$

➔ following a scheduling s, depending on the evolution time $\tau \in [0,1]$ and on some variational parameters $\boldsymbol{\theta}$.

Adiabatic Evolution (AE) as QML model

➔ Let's use an Adiabatic Evolution (AE) as model:

$$H_{ad} = H_0\big[1 - s(\tau;\boldsymbol{\theta})\big] + s(\tau;\boldsymbol{\theta})H_1, \tag{4}$$

➔ following a scheduling s, depending on the evolution time $\tau \in [0,1]$ and on some variational parameters $\boldsymbol{\theta}$.

```python
# with qibo we can implement an Adiabatic Evolution via trotter formula
from qibo import models, hamiltonians, callbacks

# problem's parameters
nqubits = 1
h0 = hamiltonians.X(nqubits)
h1 = hamiltonians.Z(nqubits)
target_observable = h1

# we track the energy of h1 on the evolved ground state
energies = callbacks.Energy(target_observable)
evolution = models.AdiabaticEvolution(
    h0=h0, h1=h1, s = lambda t : t, dt=0.1, callbacks = [energies])

# calculate the evolved final state at time t=final_time
evolved_state = evolution(final_time = final_time)
```

Adiabatic Evolution (AE) as QML model

➔ We encode our problem into an AE by mapping (x, F) into (τ, E): evolution time τ and target observable's energy E.

Adiabatic Evolution (AE) as QML model

- We encode our problem into an AE by mapping (x, F) into (τ, E): evolution time τ and target observable's energy E.
- An AE **naturally** solves the two problems:

Adiabatic Evolution (AE) as QML model

- We encode our problem into an AE by mapping (x, F) into (τ, E): evolution time τ and target observable's energy E.
- An AE **naturally** solves the two problems:

the evolution is naturally monotonic.

Adiabatic Evolution (AE) as QML model

- We encode our problem into an AE by mapping (x, F) into (τ, E): evolution time τ and target observable's energy E.
- An AE **naturally** solves the two problems:

- the evolution is naturally monotonic.
- we can fix $\hat{F}(0) = 0$ and $\hat{F}(1) = 1$ by chosing H_0 and H_1 such that their ground-state energies are zero and one.

Adiabatic Evolution (AE) as QML model

- We encode our problem into an AE by mapping (x, F) into (τ, E): evolution time τ and target observable's energy E.
- An AE **naturally** solves the two problems:

  - the evolution is naturally monotonic.
  - we can fix $\hat{F}(0) = 0$ and $\hat{F}(1) = 1$ by chosing H_0 and H_1 such that their ground-state energies are zero and one.

- We want to push the Energy of $\mathcal{O}$ to approximate the target F value when τ corresponds to the target variable x.

Adiabatic Evolution (AE) as QML model

- We encode our problem into an AE by mapping (x, F) into (τ, E): evolution time τ and target observable's energy E.
- An AE **naturally** solves the two problems:

  - the evolution is naturally monotonic.
  - we can fix $\hat{F}(0) = 0$ and $\hat{F}(1) = 1$ by chosing H_0 and H_1 such that their ground-state energies are zero and one.

- We want to push the Energy of $\mathcal{O}$ to approximate the target F value when τ corresponds to the target variable x.

Optimizing the AE

The task becomes to optimize the scheduling parameters in order to let the AE pass through the training points.

`METHOD`: Optimizing the AE

Optimizing the AE - ansatz and optimizer

- We use a **CMA-ES**[5] genetic algorithm and the following loss function:

$$J_{\mathrm{MSE}} = \frac{1}{N_{\mathrm{train}}} \sum_{k=1}^{N_{\mathrm{train}}} \left[F(x_k) - E_k(\theta) \right]^2. \qquad (5)$$

[5] arXiv:1604.00772

Optimizing the AE - ansatz and optimizer

- We use a **CMA-ES**[5] genetic algorithm and the following loss function:

$$J_{\mathrm{MSE}} = \frac{1}{N_{\mathrm{train}}} \sum_{k=1}^{N_{\mathrm{train}}} \left[F(x_k) - E_k(\boldsymbol{\theta})\right]^2. \tag{5}$$

- Thus we evolve the system using a **polynomial scheduling** function[6]:

$$s(t;\theta) = \frac{1}{\eta} \sum_{i=1}^{p} \theta_i x^i, \qquad \text{with} \qquad \eta = \sum_{i=1}^{p} \theta_i, \tag{6}$$

[5] arXiv:1604.00772

[6] One is free to set a custom anstatz, *e.g.* a neural network. With `qibo` an s function is required such as $s(0) = 0$ and $s(1) = 1$.

Optimizing the AE - ansatz and optimizer

- We use a **CMA-ES**[5] genetic algorithm and the following loss function:

$$J_{\mathrm{MSE}} = \frac{1}{N_{\mathrm{train}}} \sum_{k=1}^{N_{\mathrm{train}}} \big[F(x_k) - E_k(\boldsymbol{\theta})\big]^2. \tag{5}$$

- Thus we evolve the system using a **polynomial scheduling** function[6]:

$$s(t;\theta) = \frac{1}{\eta} \sum_{i=1}^{p} \theta_i x^i, \qquad \text{with} \qquad \eta = \sum_{i=1}^{p} \theta_i, \tag{6}$$

In each optimization step

We execute the evolution collecting $\{E_k\}$, thanks to which we evaluate J_{MSE}. Then, we update $\boldsymbol{\theta}$ according to the chosen technique.

[5] arXiv:1604.00772

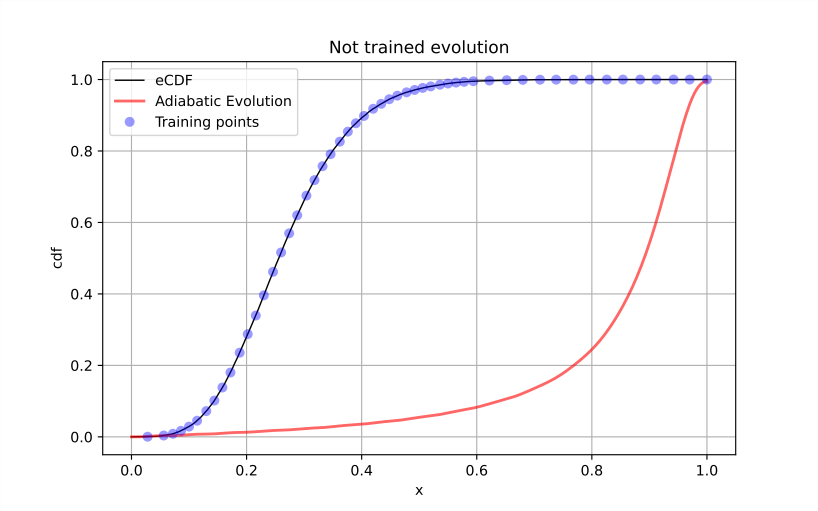
[6] One is free to set a custom anstatz, *e.g.* a neural network. With `qibo` an s function is required such as $s(0) = 0$ and $s(1) = 1$.

- The optimization step is performed using qibo:

```python
import qibo

# before we define a loss_evaluation function as J_MSE
def loss_evaluation(parameters): {...}

# then we use the cma optimizer provided by qibo
def optimize(force_positive=False, method="cma"):
    """Use qibo to optimize the parameters of the schedule function"""

    options = {
        "ftarget": target,            # Target loss function
        "maxiter": max_iterations,    # Maximum number of iterations
        "maxfeval": max_evals,        # Maximum number of function evaluations
    }

    # forcing the parameters to be positive; unused in this case.
    if force_positive:
        options["bounds"] = [0, 1e5]

    result = qibo.optimizers.optimize(loss_evaluation, parameters, method=method,
      options=options)
    return result
```

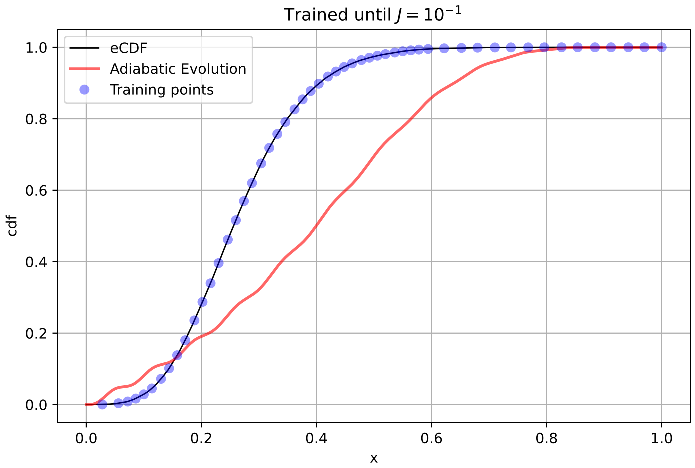
A toy example with `nqubits=1` - starting point

- `nparams=20`, `dt=0.1`, `final_time=50` , `target_loss=None`



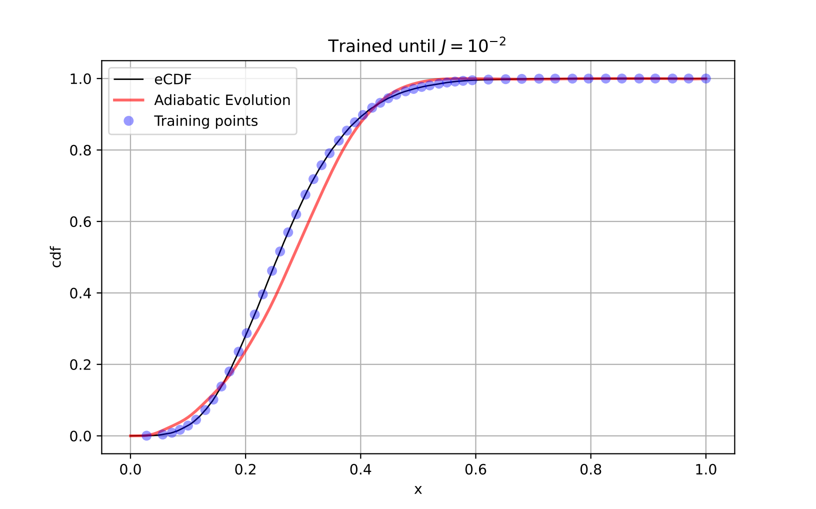
A toy example - until $J_{\mathrm{MSE}} = 10^{-1}$

- `nparams=20`, `dt=0.1`, `final_time=50` , `target_loss=1e-1`



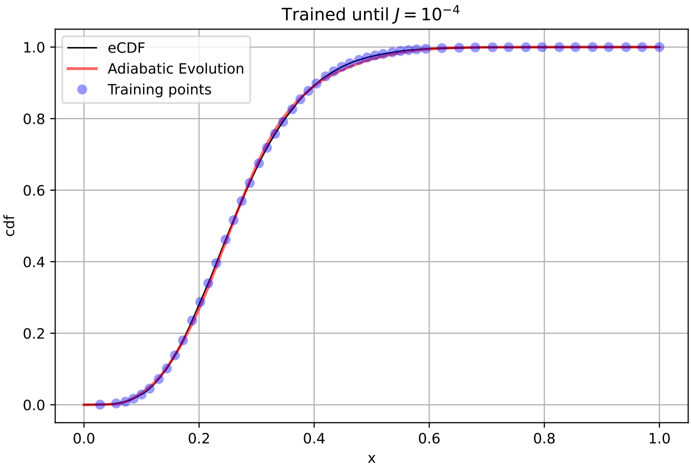
A toy example - until $J_{\mathrm{MSE}} = 10^{-2}$

- `nparams=20`, `dt=0.1`, `final_time=50` , `target_loss=1e-2`



A toy example - ending at $J_{\mathrm{MSE}} = 10^{-4}$

- `nparams=20`, `dt=0.1`, `final_time=50` , `target_loss=1e-4`



`DERIVATION`: from $\{H_j\}$ to a circuit

Translating the AE into circuits

- As previously said, we want to apply some **derivation rules**.

Translating the AE into circuits

- As previously said, we want to apply some **derivation rules**.
- During an `nsteps`-AE, we get `nsteps` **“local time”** adiabatic hamiltonians $\{H_j\}$.

Translating the AE into circuits

- As previously said, we want to apply some **derivation rules**.
- During an `nsteps`-AE, we get `nsteps` **“local time”** adiabatic hamiltonians $\{H_j\}$.
- Each of these can be associated with an **instantaneous** time evolution operator U_j, which at $\tau_j = j\,\mathrm{d}t$ should be applied on to the state obtained in τ_{j-1}.

- As previously said, we want to apply some **derivation rules**.
- During an `nsteps`-AE, we get `nsteps` **“local time”** adiabatic hamiltonians $\{H_j\}$.
- Each of these can be associated with an **instantaneous** time evolution operator U_j, which at $\tau_j = j\,\mathrm{d}t$ should be applied on to the state obtained in τ_{j-1}.
- The time-evolved state τ_n is obtained as follows:

$$|\psi(\tau_n)\rangle = \prod_{j=0}^{n} U(\tau_j)\,|\psi(\tau_0)\rangle\,, \tag{7}$$

where the initial state is the ground state of H_0 by construction.

From a sequence to a single U

➔ To calculate the derivative of (7) is not trivial, is better to have a **single unitary** $\mathcal{C}(\tau)$ which sums up all the sequence needed to get the evolved state at τ.

From a sequence to a single U

- To calculate the derivative of (7) is not trivial, is better to have a **single unitary** $\mathcal{C}(\tau)$ which sums up all the sequence needed to get the evolved state at τ.
- For doing this:

From a sequence to a single U

- To calculate the derivative of (7) is not trivial, is better to have a **single unitary** $\mathcal{C}(\tau)$ which sums up all the sequence needed to get the evolved state at τ.
- For doing this:

1. we noticed that each $U_j = \exp\{-iH_j \mathrm{d}\tau\}$ can be diagonalized;

From a sequence to a single U

- To calculate the derivative of (7) is not trivial, is better to have a **single unitary** $\mathcal{C}(\tau)$ which sums up all the sequence needed to get the evolved state at τ.
- For doing this:

1. we noticed that each $U_j = \exp\{-iH_j \mathrm{d}\tau\}$ can be diagonalized;
2. once diagonalized $\forall j$, we consider the limit such that $\mathrm{d}\tau \to 0$;

From a sequence to a single U

➜ To calculate the derivative of (7) is not trivial, is better to have a **single unitary** $\mathcal{C}(\tau)$ which sums up all the sequence needed to get the evolved state at τ.

➜ For doing this:

1. we noticed that each $U_j = \exp\{-iH_j \mathrm{d}\tau\}$ can be diagonalized;
2. once diagonalized $\forall j$, we consider the limit such that $\mathrm{d}\tau \to 0$;
3. thanks to which the diagonalizing element $P_j P_{j-1}^{-1} \to I$ and we get:

$$\mathcal{C}(\tau) = \Lambda_{t0} P_t \exp\left\{ - it \int_0^t \hat{D}(t) \mathrm{d}t \right\} P_0^{-1}, \tag{8}$$

Where the Λ factor is is used to make the determinant of P be one and $\hat{D}$ is the diagonalized hamiltonian.

➔ To calculate the derivative of (7) is not trivial, is better to have a **single unitary** $\mathcal{C}(\tau)$ which sums up all the sequence needed to get the evolved state at τ.

➔ For doing this:

1. we noticed that each $U_j = \exp\{-iH_j \mathrm{d}\tau\}$ can be diagonalized;
2. once diagonalized $\forall j$, we consider the limit such that $\mathrm{d}\tau \to 0$;
3. thanks to which the diagonalizing element $P_j P_{j-1}^{-1} \to I$ and we get:

$$\mathcal{C}(\tau) = \Lambda_{t0} P_t \exp\left\{ - it \int_0^t \hat{D}(t)\mathrm{d}t \right\} P_0^{-1}, \tag{8}$$

   Where the Λ factor is is used to make the determinant of P be one and $\hat{D}$ is the diagonalized hamiltonian.
4. Now we have a circuit $\mathcal{C}$ after which evaluate $\hat{Z}$.

From $\mathcal{C}$ to a 3-rotations circuit $\mathcal{C}_R$

- We have a unitary $\mathcal{C}$, of which we can calculate the elements c_{ij}.

From $\mathcal{C}$ to a 3-rotations circuit $\mathcal{C}_R$

- We have a unitary $\mathcal{C}$, of which we can calculate the elements c_{ij}.
- The final step is to re-write this unitary in a form on which we can use the **parameter shift rule**.

- We have a unitary $\mathcal{C}$, of which we can calculate the elements c_{ij}.
- The final step is to re-write this unitary in a form on which we can use the **parameter shift rule**.
- Each unitary $\mathcal{C} \in SU(2)$ can be written as sequence of **three rotations** using the Euler angles:

$$\mathcal{C} \equiv R_z(\phi)R_x(\theta)R_z(\psi), \tag{9}$$

- We have a unitary $\mathcal{C}$, of which we can calculate the elements c_{ij}.
- The final step is to re-write this unitary in a form on which we can use the **parameter shift rule**.
- Each unitary $\mathcal{C} \in SU(2)$ can be written as sequence of **three rotations** using the Euler angles:

$$\mathcal{C} \equiv R_z(\phi) R_x(\theta) R_z(\psi), \tag{9}$$

in which the relationships between the angles and the elements of $\mathcal{C}$ are:

$$\begin{cases} \phi = \pi/2 - \arg(c_{01}) - \arg(c_{00}) \\ \theta = -2 \arccos(|c_{00}|) \\ \psi = \arg(c_{01}) - \pi/2 - \arg(c_{00}). \end{cases} \tag{10}$$

- We have a unitary $\mathcal{C}$, of which we can calculate the elements c_{ij}.
- The final step is to re-write this unitary in a form on which we can use the **parameter shift rule**.
- Each unitary $\mathcal{C} \in SU(2)$ can be written as sequence of **three rotations** using the Euler angles:

$$\mathcal{C} \equiv R_z(\phi)R_x(\theta)R_z(\psi), \tag{9}$$

in which the relationships between the angles and the elements of $\mathcal{C}$ are:

$$\begin{cases} \phi = \pi/2 - \arg(c_{01}) - \arg(c_{00}) \\ \theta = -2\arccos(|c_{00}|) \\ \psi = \arg(c_{01}) - \pi/2 - \arg(c_{00}). \end{cases} \tag{10}$$

- Now we can derivate with respect to the rotation angles!

`DERIVATION`: derivating $\mathcal{C}_R$ via PSR

Derivating $\mathcal{C}_R$ to get the PDF

- I we call θ one of the three angles, we get $\partial_\tau \hat{F}$ by calculating:

$$\partial_\tau \hat{F}(\tau; \boldsymbol{\theta}) = \sum_{i=1}^{3} \frac{\partial \hat{F}}{\partial \theta_i} \frac{\partial \theta_i}{\partial s} \frac{\partial s}{\partial \tau}, \tag{11}$$

Derivating $\mathcal{C}_R$ to get the PDF

- I we call θ one of the three angles, we get $\partial_\tau \hat{F}$ by calculating:

$$\partial_\tau \hat{F}(\tau; \boldsymbol{\theta}) = \sum_{i=1}^{3} \frac{\partial \hat{F}}{\partial \theta_i} \frac{\partial \theta_i}{\partial s} \frac{\partial s}{\partial \tau}, \tag{11}$$

- where the first term is calculated via parameter shift rule, and the remaining twos can be obtained **analytically** if we use an analytical ansatz for s.

Derivating $\mathcal{C}_R$ to get the PDF

- I we call θ one of the three angles, we get $\partial_\tau \hat{F}$ by calculating:

$$\partial_\tau \hat{F}(\tau;\boldsymbol{\theta}) = \sum_{i=1}^{3} \frac{\partial \hat{F}}{\partial \theta_i} \frac{\partial \theta_i}{\partial s} \frac{\partial s}{\partial \tau}, \tag{11}$$

- where the first term is calculated via parameter shift rule, and the remaining twos can be obtained **analytically** if we use an analytical ansatz for s.

Summary

- We fit the eCDF via QAML;
- we translate $\{H_j\}$ into $\mathcal{C}_R$;
- we derivate the result via chain rule.

VALIDATION: **test cases**

Test results

- We performed the QAML strategy to some test cases.

Test results

- We performed the QAML strategy to some test cases.
- We did it by simulating circuits both in ideal[7] and shot-noisy[8] way.

[7] Exact state vector simulation

[8] Sampling the frequencies from the simulated state

Test results

- We performed the QAML strategy to some test cases.
- We did it by simulating circuits both in ideal[7] and shot-noisy[8] way.
- In the following table we collect realistic results in which we used $N_{\text{shots}} = 2 \cdot 10^5$ shots for evaluating $\hat{F}$.

Fit function	N_{sample}	p	J_f	N_{ratio}	χ^2
Gamma	$5 \cdot 10^4$	25	$2.9 \cdot 10^{-6}$	31	$2.2 \cdot 10^{-4}$
Gaussian mix	$2 \cdot 10^5$	30	$2.75 \cdot 10^{-5}$	31	$4.39 \cdot 10^{-3}$
t	$5 \cdot 10^4$	20	$2.1 \cdot 10^{-6}$	34	$3.4 \cdot 10^{-4}$
s	$5 \cdot 10^4$	20	$7.9 \cdot 10^{-6}$	34	$1.20 \cdot 10^{-3}$
y	$5 \cdot 10^4$	8	$3.7 \cdot 10^{-6}$	34	$1.45 \cdot 10^{-3}$

Table 1: Summary. $N_{\text{shots}} = 5 \cdot 10^4$.

[7] Exact state vector simulation

[8] Sampling the frequencies from the simulated state

Test results

- We performed the QAML strategy to some test cases.
- We did it by simulating circuits both in ideal[7] and shot-noisy[8] way.
- In the following table we collect realistic results in which we used $N_{\text{shots}} = 2 \cdot 10^5$ shots for evaluating $\hat{F}$.

Fit function	N_{sample}	p	J_f	N_{ratio}	χ^2
Gamma	$5 \cdot 10^4$	25	$2.9 \cdot 10^{-6}$	31	$2.2 \cdot 10^{-4}$
Gaussian mix	$2 \cdot 10^5$	30	$2.75 \cdot 10^{-5}$	31	$4.39 \cdot 10^{-3}$
t	$5 \cdot 10^4$	20	$2.1 \cdot 10^{-6}$	34	$3.4 \cdot 10^{-4}$
s	$5 \cdot 10^4$	20	$7.9 \cdot 10^{-6}$	34	$1.20 \cdot 10^{-3}$
y	$5 \cdot 10^4$	8	$3.7 \cdot 10^{-6}$	34	$1.45 \cdot 10^{-3}$

Table 1: Summary. $N_{\text{shots}} = 5 \cdot 10^4$.

- The last three elements in the table refers to a $pp \to t\bar{t}$ production.

[7] Exact state vector simulation

[8] Sampling the frequencies from the simulated state

Test 1: Gamma CDF

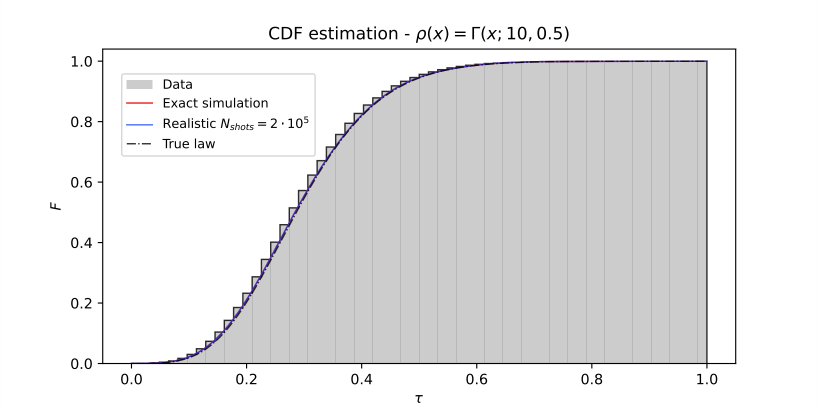


Figure 1: We show the sample (grey hist), ideal (red line) and realistic (blue line) simulation, theoretical CDF of the gamma distribution (dashed black line).

Test 1: Gamma PDF

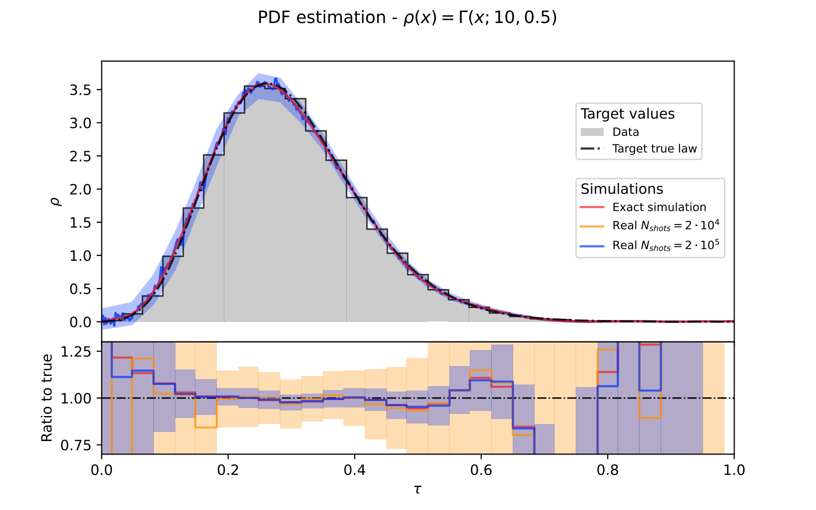


Figure 2: Above: data (grey hist), ideal (red) and realistic (blue) simulation, theoretical PDF law (dashed black line). Below: same quantities but normalized with respect to the true law with the addition of a second realistic simulation (yellow) line.

Test 2: gaussian mixture CDF

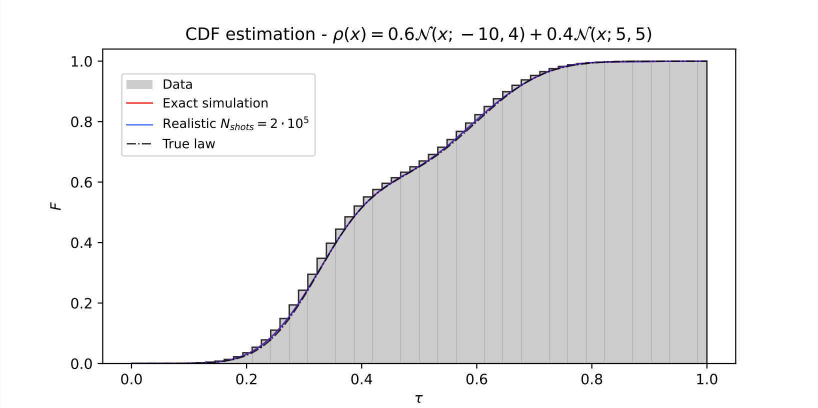


Figure 3: We show the sample (grey hist), ideal (red line) and realistic (blue line) simulation, theoretical CDF of the gamma distribution (dashed black line).

Test 2: gaussian mixture PDF

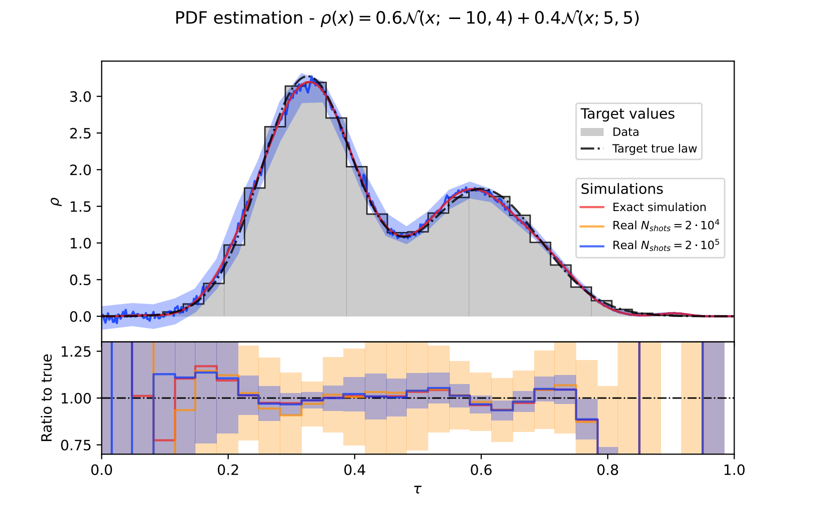
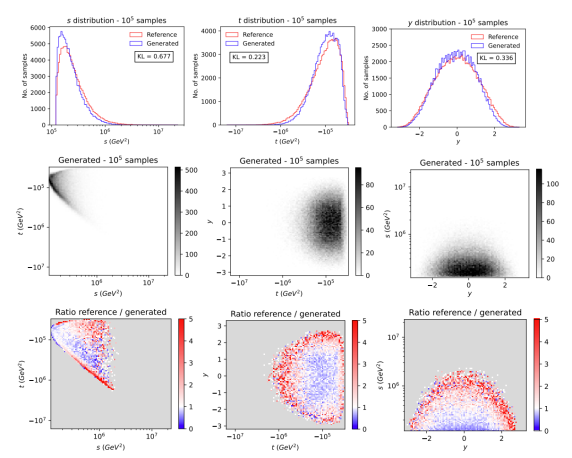


Figure 4: Above: data (grey hist), ideal (red) and realistic (blue) simulation, theoretical PDF law (dashed black line). Below: same quantities but normalized with respect to the true law with the addition of a second realistic simulation (yellow) line.

Test 3: quantum generation of $pp \to t\bar{t}$

- In a previous TH work[9] a sample of 10^5 events for $pp \to t\bar{t}$ at a center of mass $\sqrt{s} = 13$ TeV for LHC configuration was generated[10].



---

[9] arXiv:2110.06933

[10] s and t Mandelstam variables, y rapidity.

HEP 1: y CDF

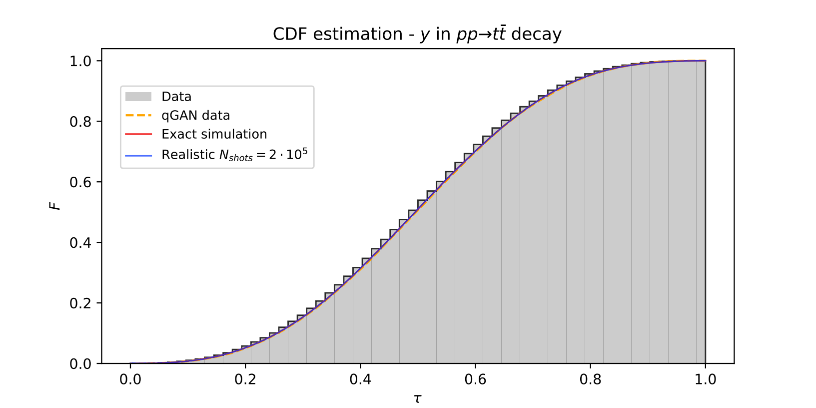


Figure 5: We show the sample (grey hist), ideal (red line) and realistic (blue line) simulation, quantum GAN eCDF (yellow).

HEP 1: y PDF

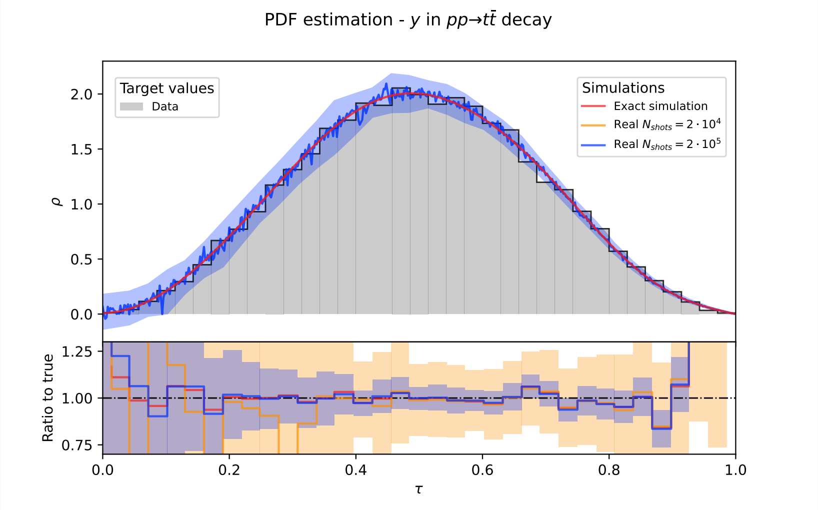


Figure 6: Above: data (grey hist), ideal (red) and realistic (blue) simulation. Below: same quantities but normalized with respect to the true law with the addition of a second realistic simulation (yellow) line.

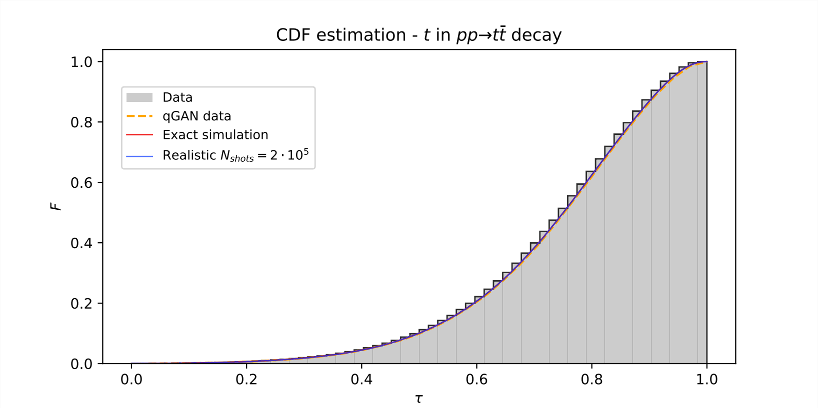


Figure 7: We show the sample (grey hist), ideal (red line) and realistic (blue line) simulation, quantum GAN eCDF (yellow).

HEP 2: t PDF

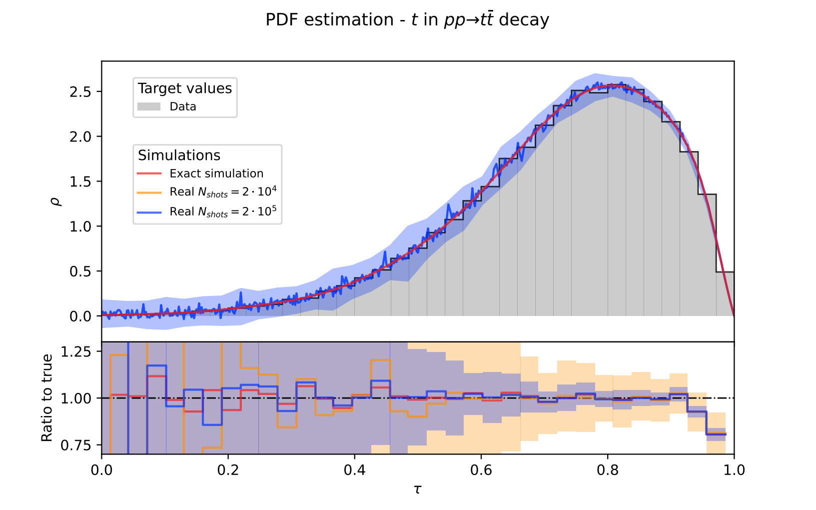


Figure 8: Above: data (grey hist), ideal (red) and realistic (blue) simulation. Below: same quantities but normalized with respect to the true law with the addition of a second realistic simulation (yellow) line.

HEP 3: s CDF

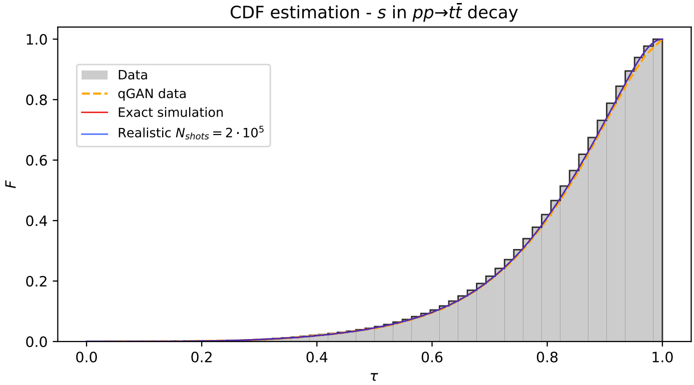


Figure 9: We show the sample (grey hist), ideal (red line) and realistic (blue line) simulation, quantum GAN eCDF (yellow).

HEP 3: s PDF

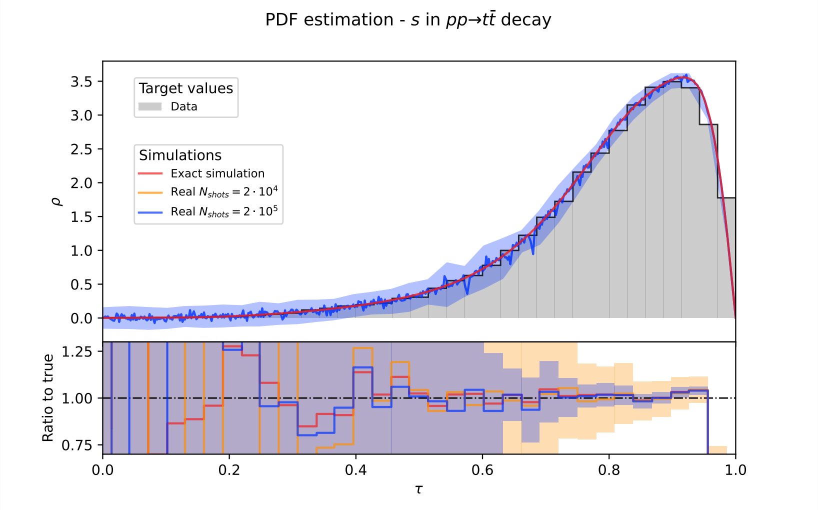


Figure 10: Above: data (grey hist), ideal (red) and realistic (blue) simulation. Below: same quantities but normalized with respect to the true law with the addition of a second realistic simulation (yellow) line.

Outlook and references

Outlook

- Open roads:

Outlook

- Open roads:

  - improve QAML validation: *try more general scheduling functions (e.g. ANNs), benchmarking with other density estimation techniques;*

[11] In case of *iid* random variables this model can be used individually for each dimension.

Outlook

- Open roads:
  - improve QAML validation: *try more general scheduling functions (e.g. ANNs), benchmarking with other density estimation techniques;*
  - hardware deployment: *can a NISQ device be used for this? what is the speed-up advantage when running the QAML on annealers?*
  - multi-dimensional[11] PDFs : *how to preserve correlations? how to exploit multi-qubits hamiltonians?*

[11] In case of *iid* random variables this model can be used individually for each dimension.

Outlook

- Open roads:
  - improve QAML validation: *try more general scheduling functions (e.g. ANNs), benchmarking with other density estimation techniques;*
  - hardware deployment: *can a NISQ device be used for this? what is the speed-up advantage when running the QAML on annealers?*
  - multi-dimensional[11] PDFs : *how to preserve correlations? how to exploit multi-qubits hamiltonians?*
  - use the QAML strategy for integrating, *e.g.* deploying of a new NNPDF feature;

[11] In case of *iid* random variables this model can be used individually for each dimension.

Outlook

- Open roads:

  - improve QAML validation: *try more general scheduling functions (e.g. ANNs), benchmarking with other density estimation techniques;*
  - hardware deployment: *can a NISQ device be used for this? what is the speed-up advantage when running the QAML on annealers?*
  - multi-dimensional[11] PDFs : *how to preserve correlations? how to exploit multi-qubits hamiltonians?*
  - use the QAML strategy for integrating, *e.g.* deploying of a new NNPDF feature;
  - anomaly detection applications of the QAML algorithm.

[11] In case of *iid* random variables this model can be used individually for each dimension.

Some references

We leave some references and links thanks to which you can use our codes and read more about the project:

- open-access codes for personal coding or contributing:
  - the qibo package;
  - the qibolab package;
  - the qibocal package;
- our official webpage, with the following documentations:
  - the qibo docs;
  - the qibolab docs;
  - the qibocal docs;

The code is open-source and available here!