Quantum Parameter Estimation with Neural Networks

Yakov Solomons & Ariel Smooha

15.08.2021

Greplova Eliska, et al. *arXiv preprint arXiv:1711.05238* (2017)

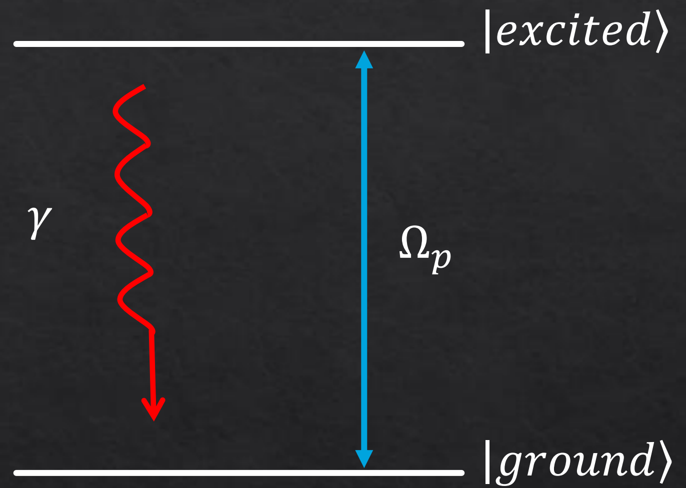
Contents

- Background
- Our goal
- Data
- Models & Methods
- Results

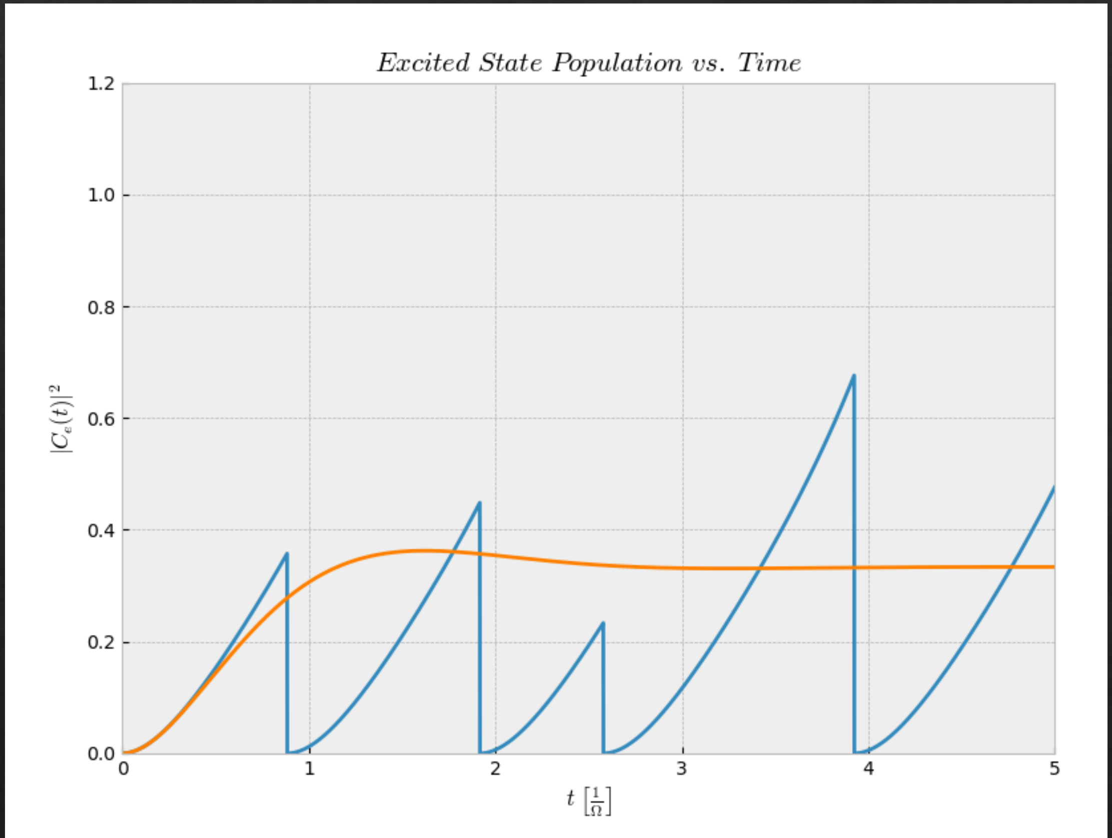
Open quantum systems

2 level system

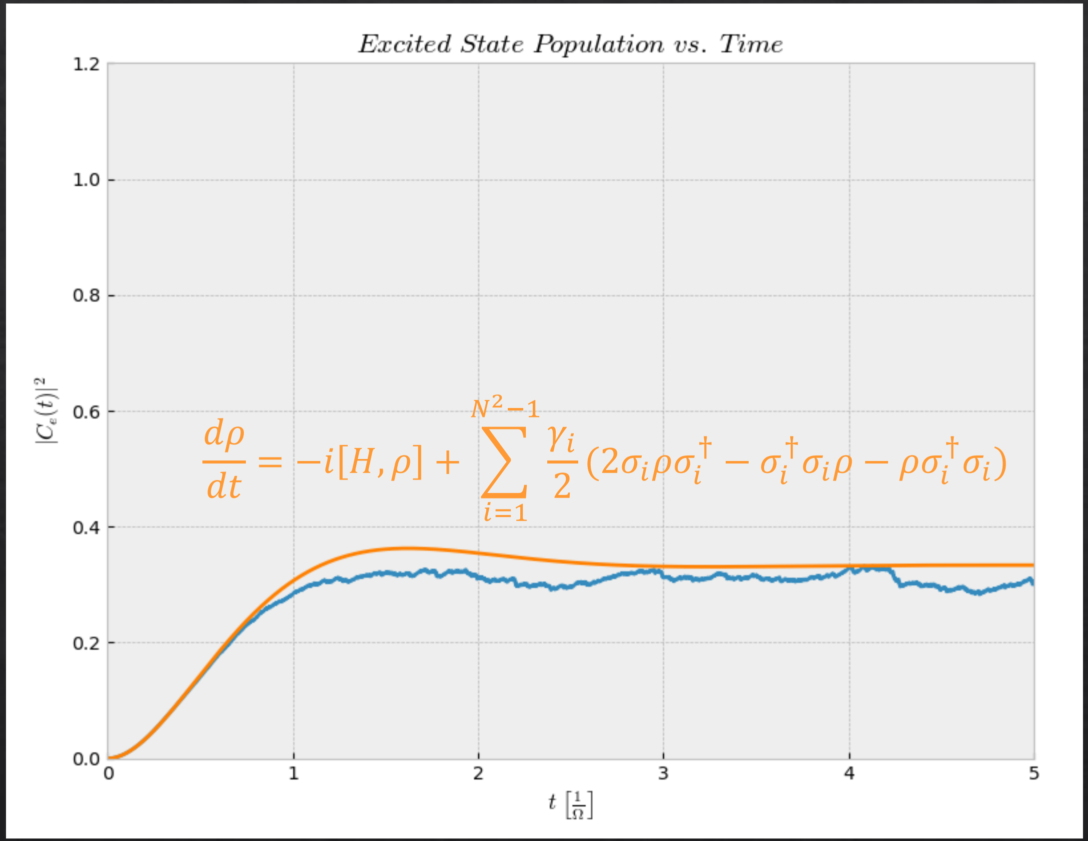
Noise and dissipation



Measurements are noisy!



1 repetition



400 repetitions

Mølmer Klaus. *JOSA B* 10.3 (1993): 524-538

What is our goal?

Find the physical parameters from a single shot measurement!

Ω – Rabi frequency (driving field)

$\gamma - decay\ rate$

```python
import numpy as np
import random
import time
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm
from matplotlib import style

""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""

for r in range(repetition):

  for w in omega_vec:
    omega = w

    for g in gamma_vec:
      gamma = g

      for d in delta_vec:
        delta = d

        for jj in range(leng):

          c = np.zeros((2, int(T / dT)), dtype=np.complex128)     # |C(2,:)|^2 is the excited state population
          c[0, 0] = 1

          for n in range(int(T / dT)-1):

            rand = random.uniform(0,1)

            if rand < (2 * gamma * dT * (abs(c[1, n])) * (abs(c[1, n]))):
              c[0, n + 1] = 1
              c[1, n + 1] = 0
            else:
              c[0, n + 1] = 1j * dT * (omega * c[1, n]) + c[0, n]
              c[1, n + 1] = 1j * dT * (omega * c[0, n]) + c[1, n]-c[1, n] * gamma * dT + 1j * dT * (delta * c[1, n]) + 1j * delta * dT * c[1, n]

              c[0, n + 1] = c[0, n+1] / np.sqrt(abs(c[0, n + 1])*abs(c[0, n + 1]) + abs(c[1, n + 1])*abs(c[1, n + 1]))
              c[1, n + 1] = c[1, n + 1] / np.sqrt(abs(c[0, n + 1]) *abs(c[0, n + 1]) + abs(c[1, n + 1]) * abs(c[1, n + 1]))

          c_mean[jj] = (abs(c[1])) * (abs(c[1]))


          np.save(r'C:\Users\ariel\Dropbox (Weizmann Institute)\Deep Learning\project\Dataset\validation\repetition={} omega={}  gamma={} delta={}'.format(r + 1, w, gamma, delta(w-1)*10+gamma-1, c_mean[jj])
```

Generating the Data
Monte Carlo simulation

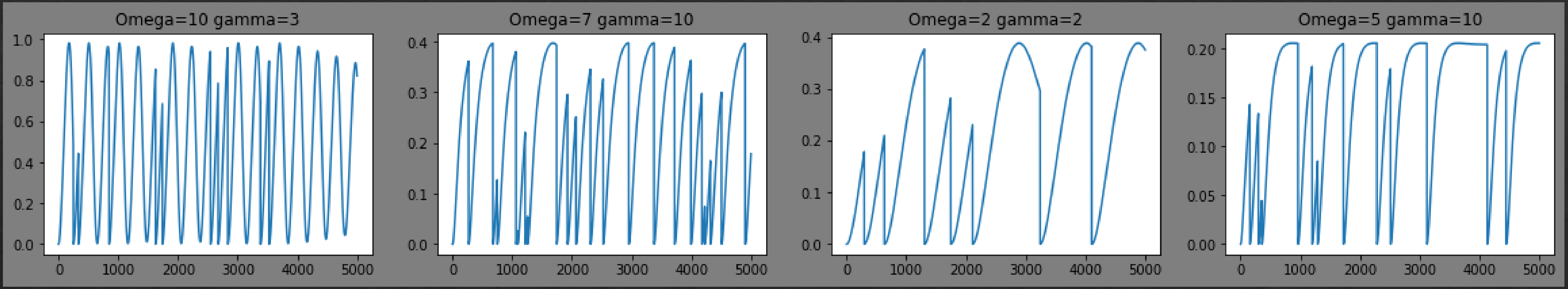
Training set : 10,000 files (about 30 min to generate) Corresponding to 100 repetitions per each combination of omega and gamma, 400 MB

Evaluation set : 5,000 files (about 15 min to generate), 200MB

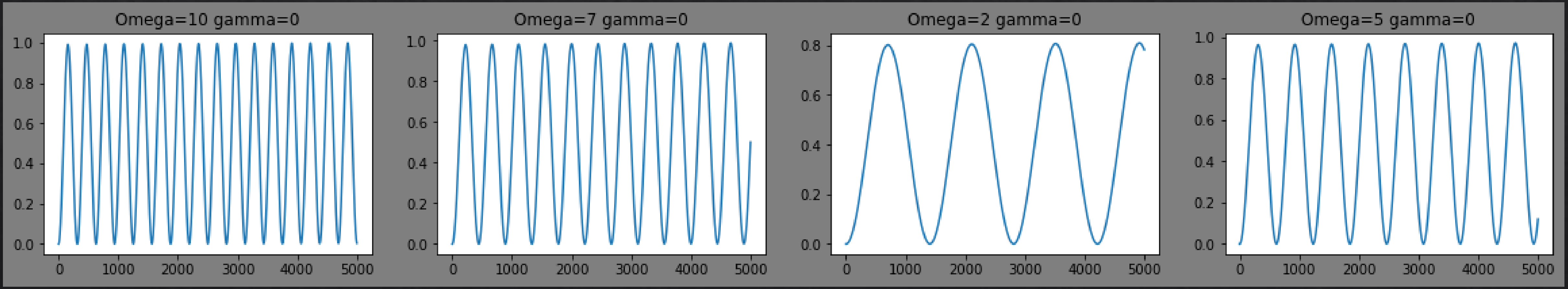
10 values for omega × 10 values for gamma = 100 combinations

× 100 repetitions = 10,000 files

Generating the Data - examples



Comparison: Data without noise



```python
import torch
import numpy as np
import torch.nn as nn
from torch.nn import ReLU

class Net(nn.Module):

    def __init__(self):
        super().__init__()

        self.features1 = nn.Sequential(
            ReLU()
            ,nn.Linear(5000,100)
            ,ReLU()
            ,nn.Linear(100,100)
            ,ReLU()
            ,nn.Linear(100,100)
            ,ReLU()
            ,nn.Linear(100,10)
            ,ReLU()
            ,nn.Linear(10,10,bias = False)
            ,nn.BatchNorm1d(10)
           )

        self.features2 = nn.Sequential(
            nn.ReLU()
            ,nn.Linear(5000,100)
            ,ReLU()
            ,nn.Linear(100,100)
            ,ReLU()
            ,nn.Linear(100,100)
            ,ReLU()
            ,nn.Linear(100,10)
            ,ReLU()
            ,nn.Linear(10,10,bias = False)
            ,nn.BatchNorm1d(10)
       )


    def forward(self, x):

        out1 = self.features1(x)
        out2 = self.features2(x)

        return out1, out2
```

Models

Method 1

Two NN for classification, one for Ω and one for γ

FCNN

CNN

Method 2

One net for classification of two parameters simultaneously Ω & γ

FCNN

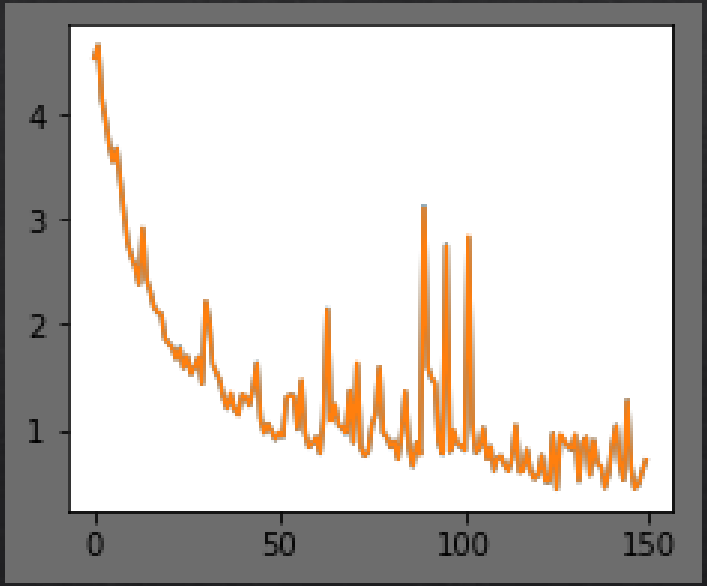
CNN

Results – method 1
FCNN

Reaching a low and limited accuracy

Loss vs. epochs

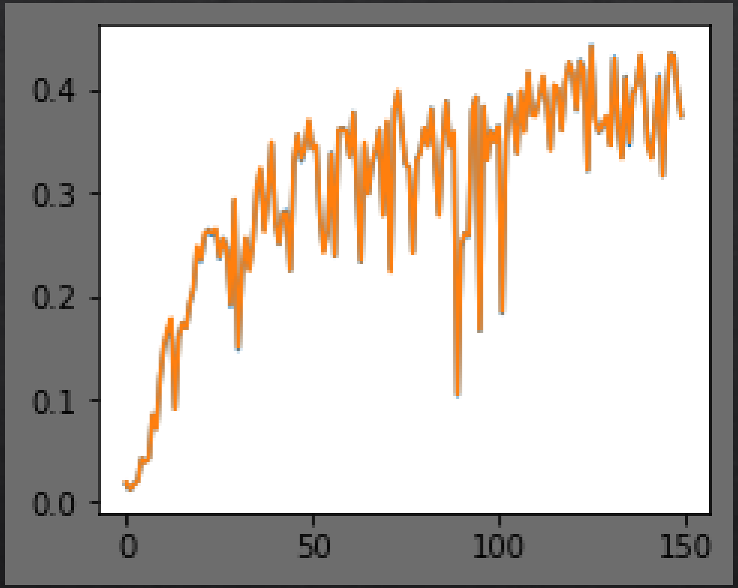
Loss



Epochs

Accuracy vs. epochs

Accuracy



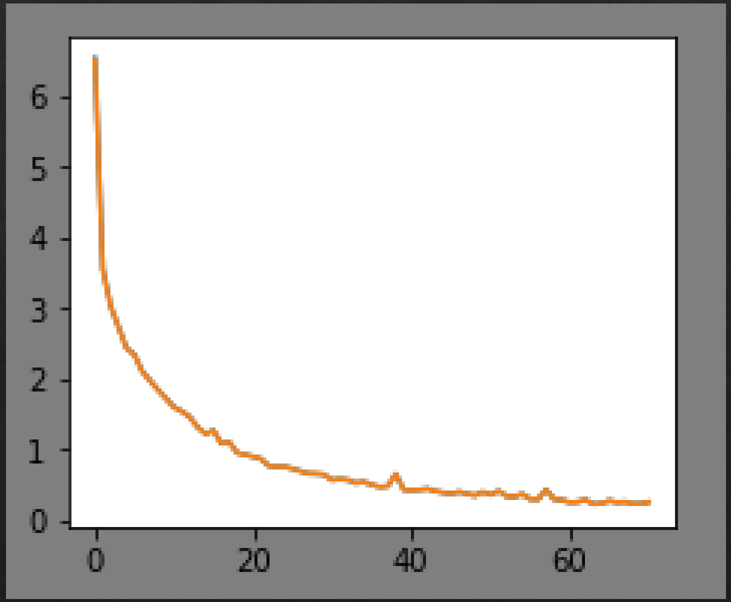
Epochs

val acc:0.39950, train acc:0.39885: 100% 150/150 [29:34<00:00, 11.83s/it]

Results – method 1
CNN

Reaching similar accuracy but much stable process!
Simple CNN in this case

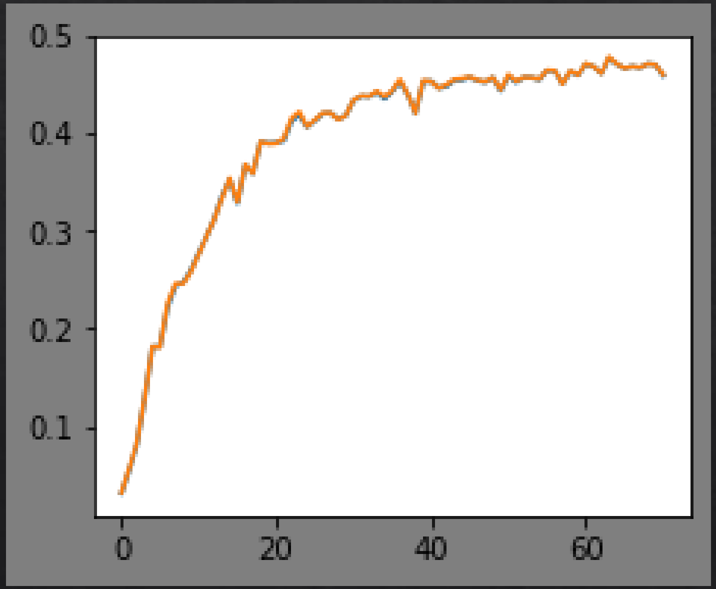
Loss vs. epochs



Loss

Epochs

Accuracy vs. epochs



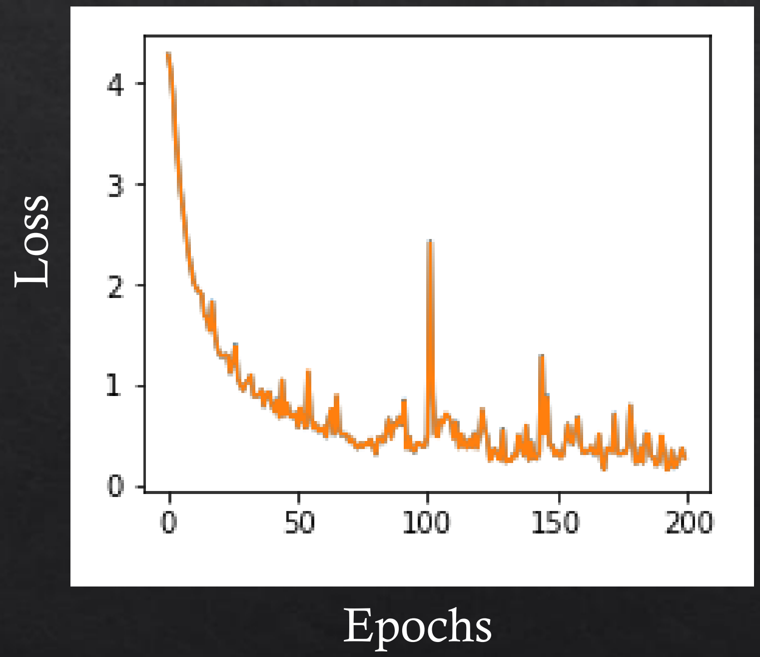
Accuracy

Epochs

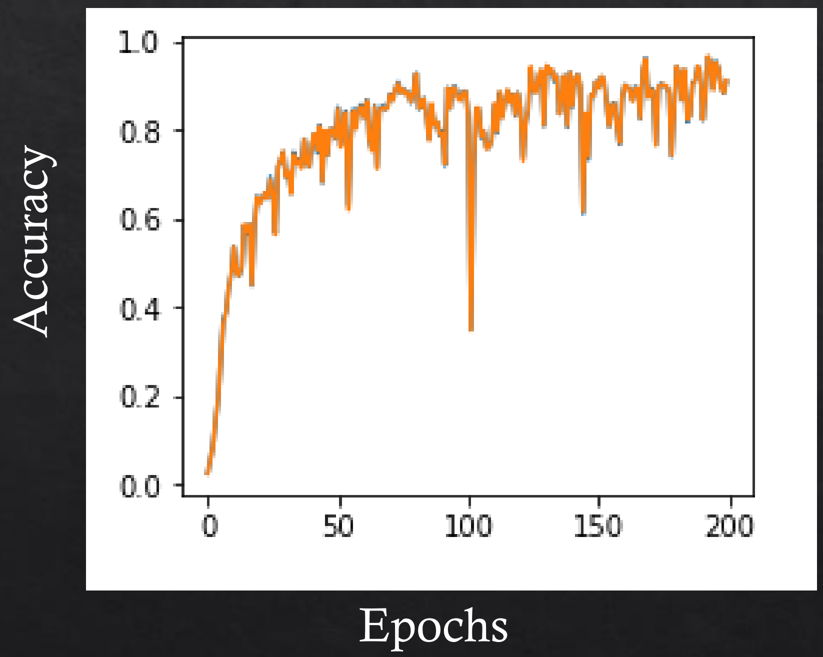
Results - method 2
FCNN

Reaching better accuracy in spite of the jumps

Loss vs. epochs



Accuracy vs. epochs

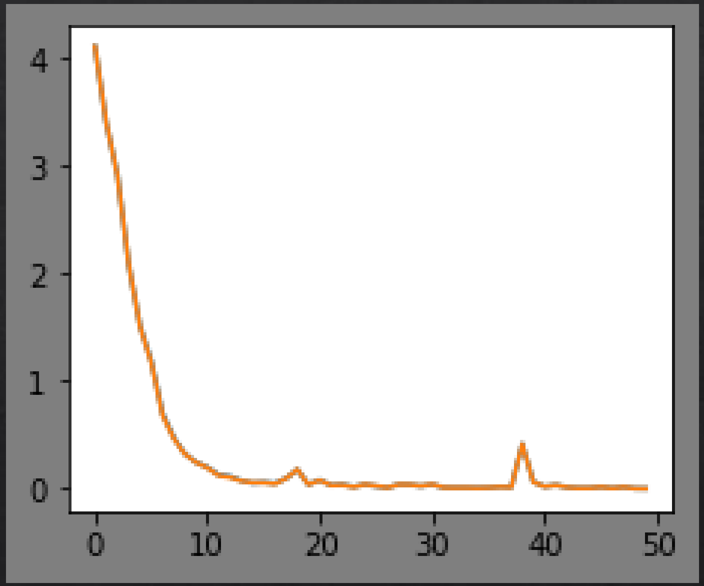


Results – method 2
CNN

Reaching great accuracy without jumps!

Loss vs. epochs

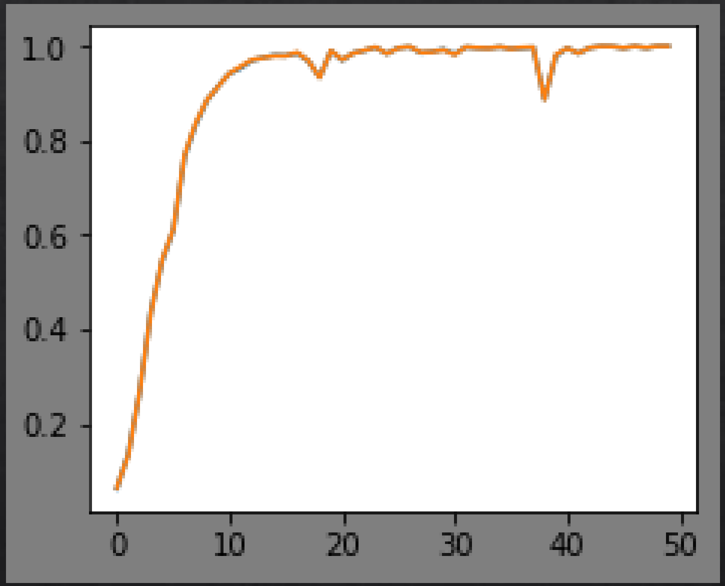
Loss



Epochs

Accuracy vs. epochs

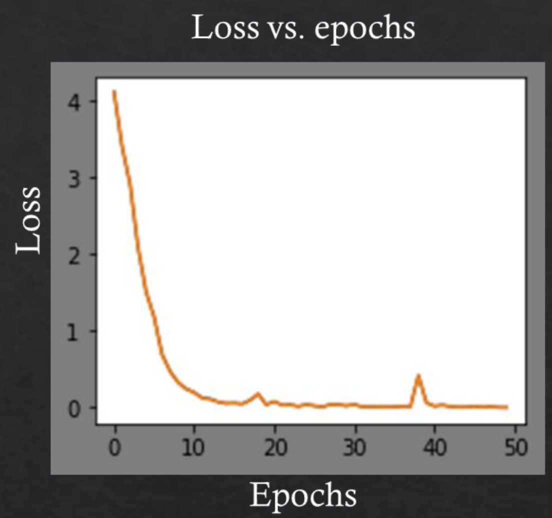
Accuracy

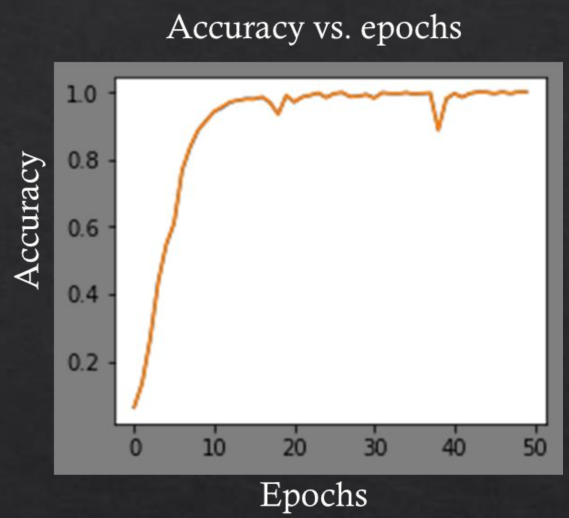


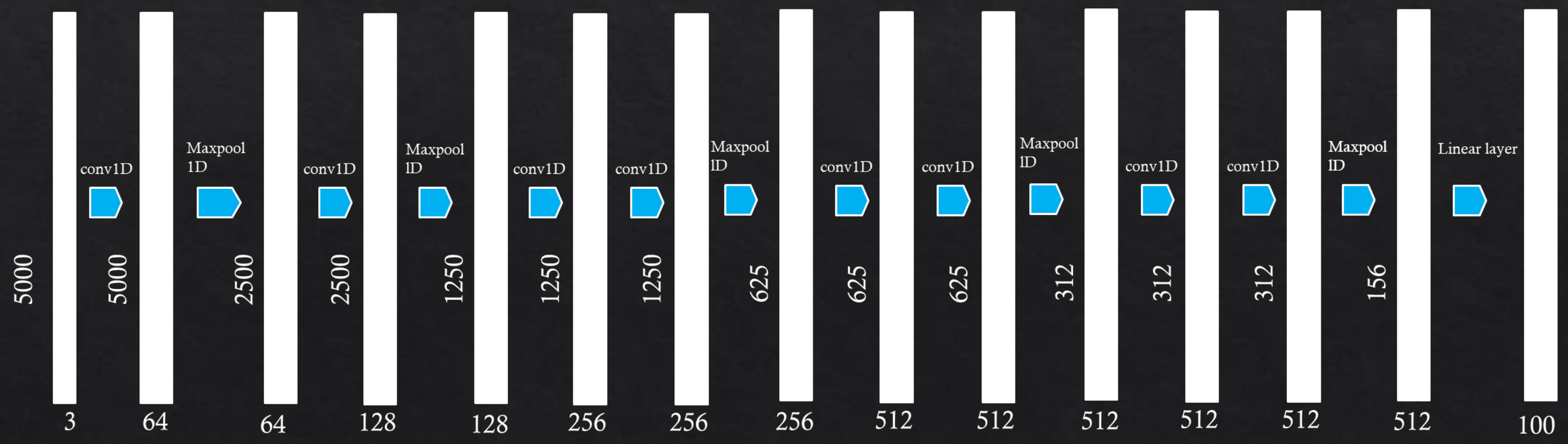
Epochs

val acc:1.00000, train acc:1.00000: 33% 50/150 [45:52<1:30:59, 54.60s/it]

Results – method 2: CNN – 1D





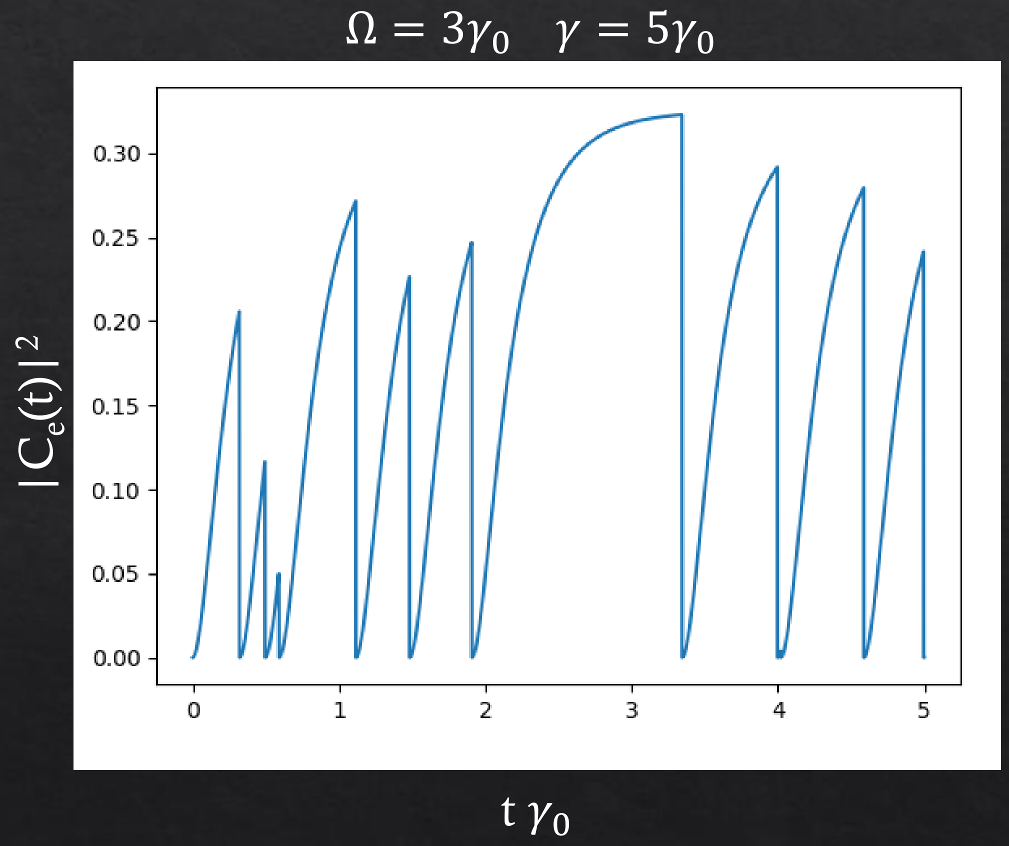


Is it possible to solve it “classically”?

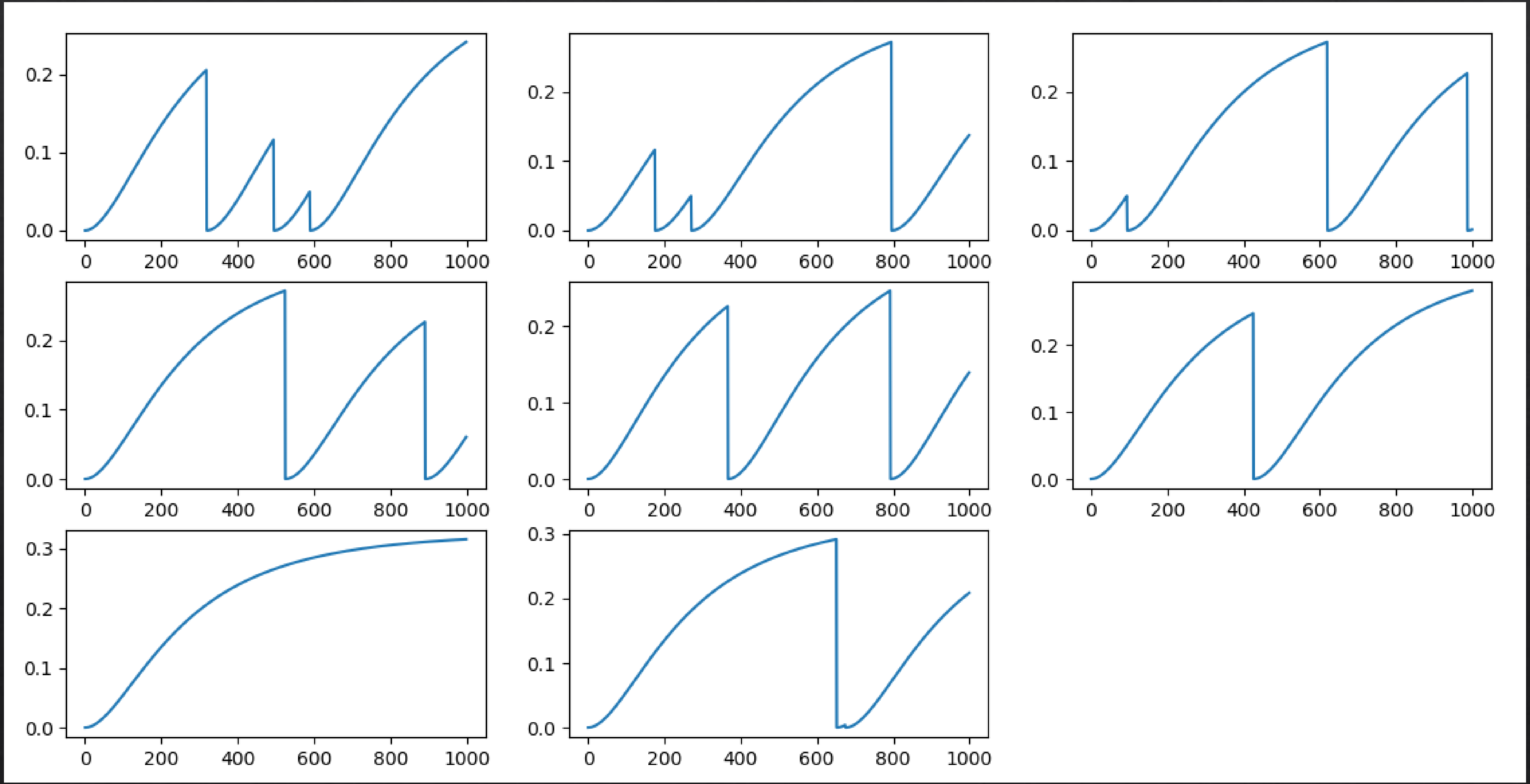
Only if there is an analytical solution for the master equation

Possible solution:

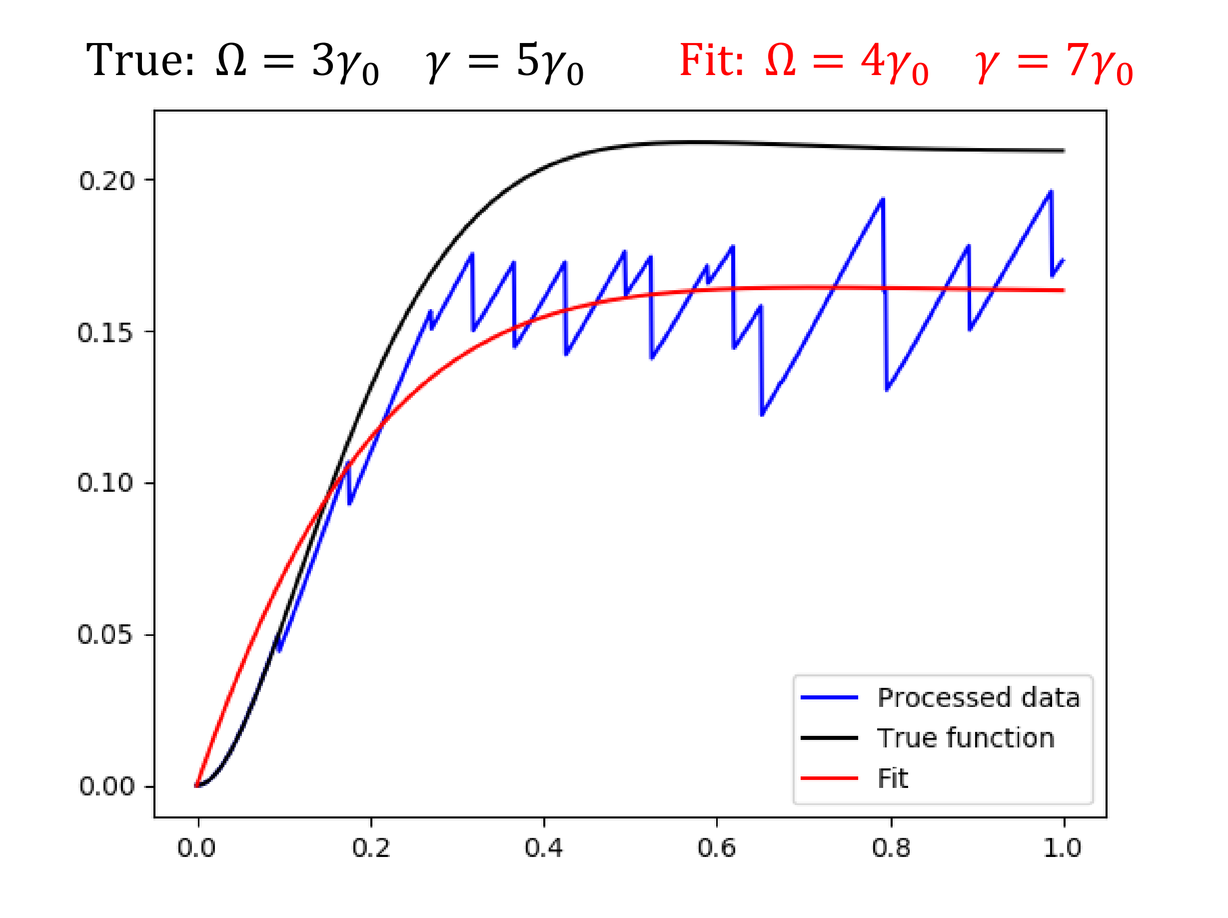
1. Cut the measurement to pieces
2. Sum them together
3. Fit to the analytical solution



Step 1 – Cut the function to pieces



Steps 2 & 3 – Summing and Fitting



Prospects

- Regression task
- Dealing with more complex problems