$3D$-Gaussian, studying the covariance

(Dated: August 15, 2021)

We want to see whether our quantum GAN has truly learnt the distribution it is supposed to. This includes whether it correctly captures the covariance matrix. In this set of tests we attempt to quantify how well the covariance matrix is learnt compared to the exactly specified one in the $3D$ Gaussian setup.

I. MOTIVATION

The ultimate goal is to arrive at a quantum GAN that can be used as surrogate model to the generative Monte-Carlo algorithm used traditionally. To this extent we want to be able to sample, i.e. generate, data that are realistic, ideally with quantifiable deviations from the traditional ones and that can be used to boost statistics beyond the current level.

To achieve in particular the last goal it is important that the correlations between different data entries (x, y, z) are captured correctly. In some sense this is the physics: if it is learnt correctly then in principle an infinite amount of samples can be drawn from the surrogate model, if it is an approximation it will be able to generate samples up to a certain level until errors are so much accumulated that the result distribution visibly does not match the target, and finally if it is not learnt at all then the surrogate model can only be used to produce mock-images of the data but using it as a generative model could be difficult.

Note, we need to keep in mind a distinction here as the full underlying physics cannot be learned from a finite representative dataset, although we might get very, very close. It is this last aspect of finiteness of data that entail we are doing "approximate physics", if one will. Our GAN can hope to learn this approximate physics through training.

There are as a result two sources of deviation or approximation, that from the GAN approximating the physics encapsulated in the finite dataset and then that from the finite dataset only approximating the full physics underneath. These statements are true only to the level of precision one is interested in, if both approximation levels can be controlled then we can also hope to understand to what precision we can trust the surrogate results. Quantifying this is partly the motivation of this note. Also it is important to step away from the theoretical standpoint in which the underlying theory is known and can be written down exactly.

A. Re-training with 2,5,7 and 10 layers

To study the quantum GAN's behaviour we focus on the 3D Gaussian test in which the input covariance matrix is exactly known. The aim is to develop and deploy measures that effectively quantify how well this covariance matrix has been learned in the surrogate model.

Note, the original target covariance matrix input during our development phase is not positive semi-definite. According to the `python` docs this can lead to undefined behaviour in sampling using the multivariate normal. Since we want to study a known cov.mat., we change it to:

```
cov = [[0.5, 0.1, 0.25], [0.1, 0.5, 0.1], [0.25, 0.1, 0.5]]
mean = [0, 0, 0]
x = np.random.multivariate_normal(mean, covm, 10000).T/4
```

- as side remark, the outcome cov.mat.s will always be an exact factor 16 different from the defined one due to the $1/4$ normalisation in sampling $\mathbf{x}$.

After adapting `quantum-classical_3Dgaussian.py` by inputting this new cov.mat. Stefano has re-run the program and uploaded the results to the git. Choosing to scan with $n_{layers} = 2, 5, 7, 10$ the latent dimension was varied $d_{latent} = 1, 2, 3, 4, 5, 6$. Plots showing the losses and their convergence in training can be generated via the `plots_3dgaussian.py` script provided in the git. All runs look very well converged but some still exhibit fluctuations also late in training.

The provided `PARAMS` files are used to generate $n_{samp} = 10000$ samples $N(real)$ using the known and $N(fake)$ by executing the quantum circuit with the learned cov.mat.s.

II. APPROACHING THE COVARIANCES

A. 3D histograms

We can attempt a visualisation of the sets of samples $N(real)$ and $N(fake)$ by first turning them into 3D histograms:

```
# there is an n_bin dependence here
nbin=40 (default)

# data looks like
datareal=(x,y,z)
datafake=(x,y,z)

D_real=np.histogramdd(datareal.T,bins=nbin,range=[[-1,1],[-1,1],[-1,1]])
D_fake=np.histogramdd(datafake.T,bins=nbin,range=[[-1,1],[-1,1],[-1,1]])
```

These define effective densities $D(real)$ and $D(fake)$ that we can use to estimate how close or different they are. These densities and any derived results are dependent on the number of bins set for the 3D histograms. Currently $n_{bin} = 40$, in all directions this entails a resolution of a total of $64k$ bins.

In Fig. 1(top row) the density for the real data, i.e. the target, derived in this way is shown. For the visualisation the data is projected from $(x, y, z) \to (y, z)$. The choice of (y, z) stems from the new cov.mat. having a visible by eye correlation here.

In the second, third and fourth row of the same figure the differences of the target densities $D(real)$ and the learned $D(fake)$ with different n_{layers} and d_{latent} are shown. Hereby the first two rows show the dependence in d_{latent} with $n_{layers} = 5$ fixed. The fourth row shows the dependence with $d_{latent} = 3$ fixed instead. If the densities matched perfectly we would see a fully flat, white grid in these figures.

In the first figure, second row left, where the number of latent dimensions is 1, we see that the two densities differ significantly at the edges. This in turn leads to the whitened "well" in the middle of the figure, this is an artefact of the visualisation. Nevertheless, as we increase the latent dimension this well closes and we are left with local fluctuations. The last row comparing the dependence in the number of layers does not show a systematic trend, the cases 5 and 7 at fixed latent dimension exhibit larger fluctuations than their 3 and 10 layered counterparts. This could perhaps be an indication of undertraining(?).

In all cases we do not observe a perfect match, however, also the target density is estimated from a finite dataset and only approximates the true density. Redoing this test with more samples could improve this situation. Additionally there is the dependence on the bins, which plays into the same issue, as the bins need to be sufficiently well populated while retaining a resolution that permits observing the interesting features.

B. Towards single number measures

With this in mind we can define a single-number measure based on the difference of the densities as:

```
# define measure as summed difference between the real and fake densities
meas = sum( D(real) - D(fake) )

# there are deviations due to the sampling in D(real),
these can be used to normalise and offset the result

# redraw n
datareal_n=(x,y,z)

# calculate standard deviation and mean of datareal and form the difference
D(baseline) = D(real) - D(realmean)

# correct measure to get relative shift from the baseline
meas = sum[ D(real) - D(fake) ) /  D(baseline) ]
```

The results are shown in Fig. 2(a). Note, this only works when both $D(real)$ and $D(fake)$ are normalised in the same way. Here this was achieved by fixing the number of bins and samples to be the same in both cases. Since the corrected measure shows the relative deviation from the baseline, i.e. the internal deviation due to the finite sampling, the situation where the target and learned distributions are the same occurs when the shown value is equal to 1. For some of the runs this is achieved at $d_{latent} = 3$.

C. 3D histogram projections

The two estimated densities $D(real)$ and $D(fake)$ in themselves provide approximations to the 3D distributions that are being sampled. This means they can be used to compute also Kullbeck-Leibler divergences. In this case we need to compare $D(real, w)$ with $D(fake, w)$ where $w = x, y, z$. As such we compute the KL-divergence

$$KL_{sum} = \sum_{w=x,y,z} KL[D(real, w), D(fake, w)]$$

by projecting the $real/fake$ combinations to 2D in e.g. x_{real}, x_{fake}. If the distributions match perfectly KL_{sum} should be 0. The results are shown in Fig.2(b) and we observe a minimum $KL_{sum} \simeq 1e^{-2}$ reached.

D. Covariance matrix

Perhaps more interesting than comparing approximate densities is to look at the cov.mat.s defined by the real and fake data directly. This allows for the comparison between the exact input and the real as well as the fake datasets. To achieve this our synthetic data can be converted into cov.mat.s and further analysed in the following way:

```
# data looks like
datareal=(x,y,z)
datafake=(x,y,z)

1.)
# can define covariance matrix like
covmatt= cov(datareal)
covmat= cov(datafake)

2.)
# determine the eigenvalues and -vectors via
wt, vt = eig(covmatt)
w, v = eig(covmat)

3.)
# to generate samples from the covariance matrices:
# draw from Gaussian centered at 0 and a standard deviation of 0.5
x = norm.rvs(size=(3, 10000), loc=0, scale=0.5)

# convert the data to correlated random variables.
yt = np.dot(np.linalg.cholesky(covmatt), x)
y = np.dot(np.linalg.cholesky(covmat), x)

4.)
# to compute the KL divergence of the generated data note that
# KL can only compare 2 distributions, so need to limit to e.g. x_real with x_fake:
offset=1 # to ensure there's no 1/0.
kl_div = 0.5* (entropy(yt[0]+offset, y[0]+offset) + entropy(y[0]+offset, y[0]+offset))
```

- in this pseudo-code the first step defines the cov.mat.s of the real and fake data, the second determines the Eigenvalues and -vectors of these matrices, the third then draws samples using these cov.mat.s and the fourth determines the KL_{sum}, akin to that of the previous section.

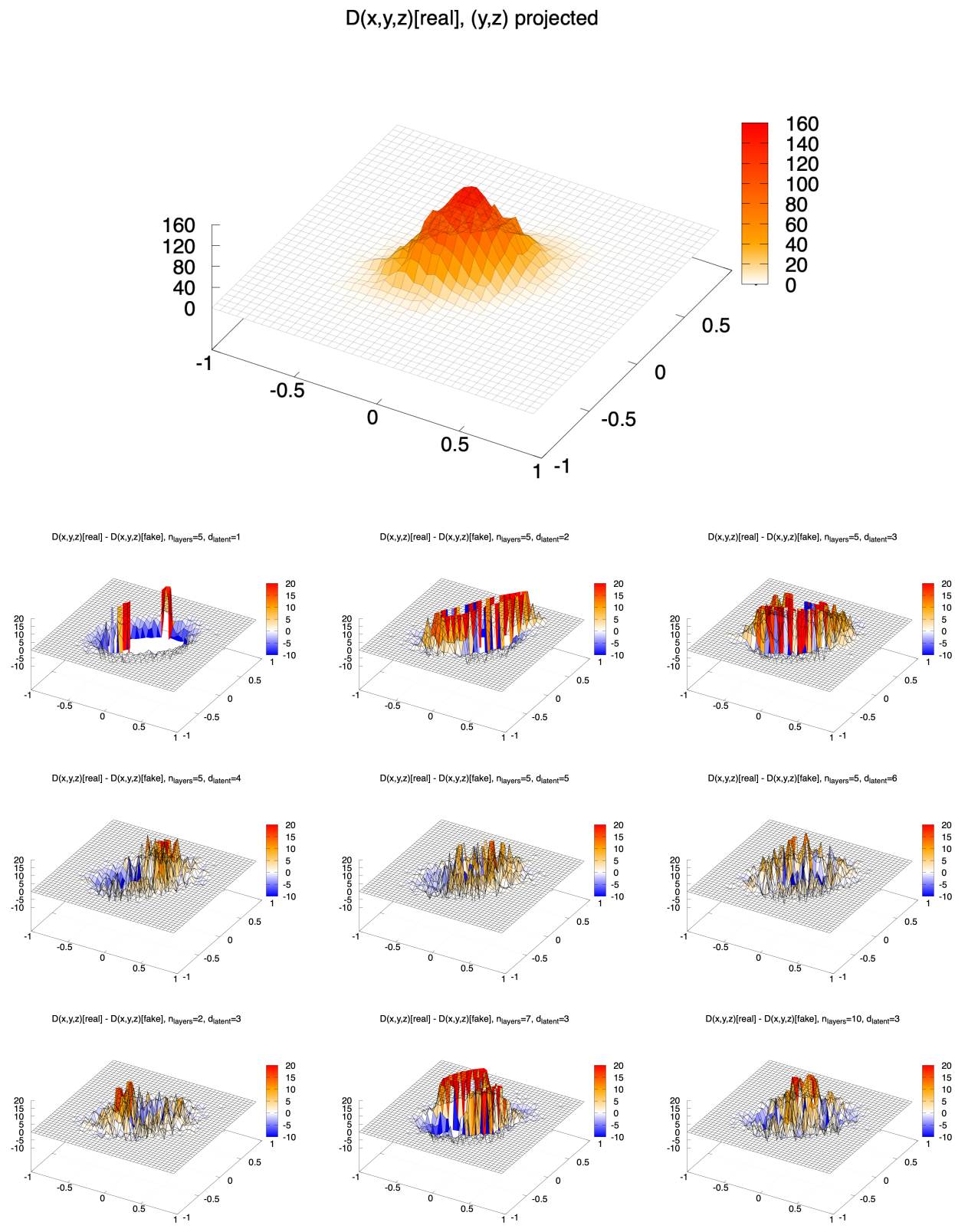
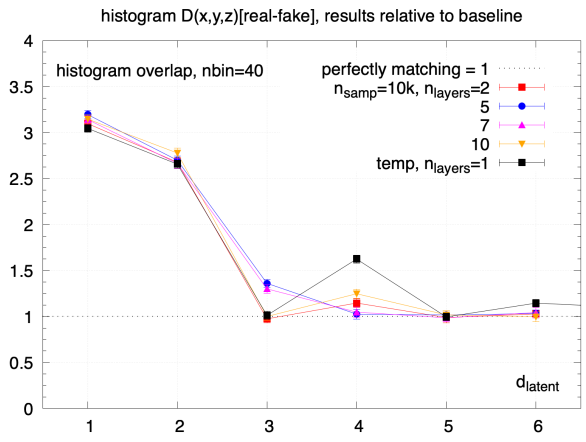
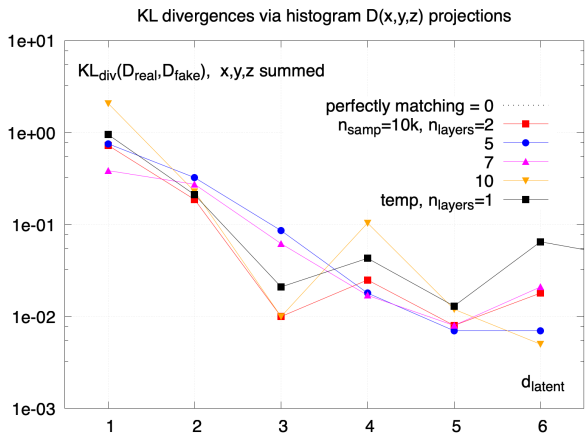


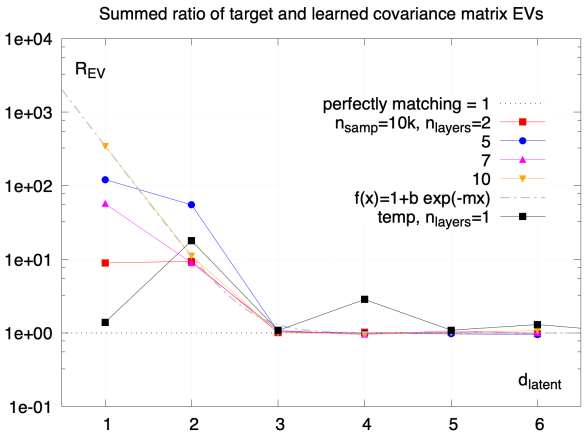
FIG. 1: Estimated densities, please see text for details.



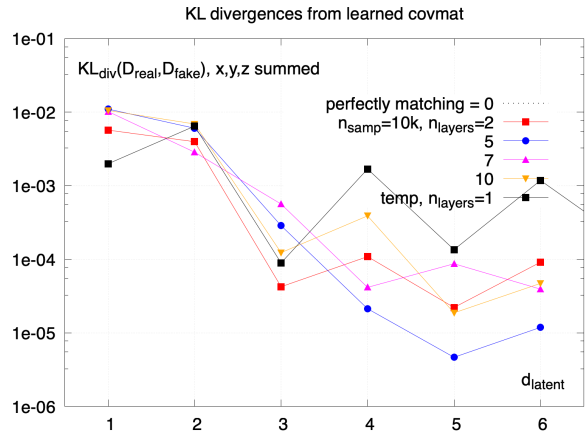
(a) Density measure



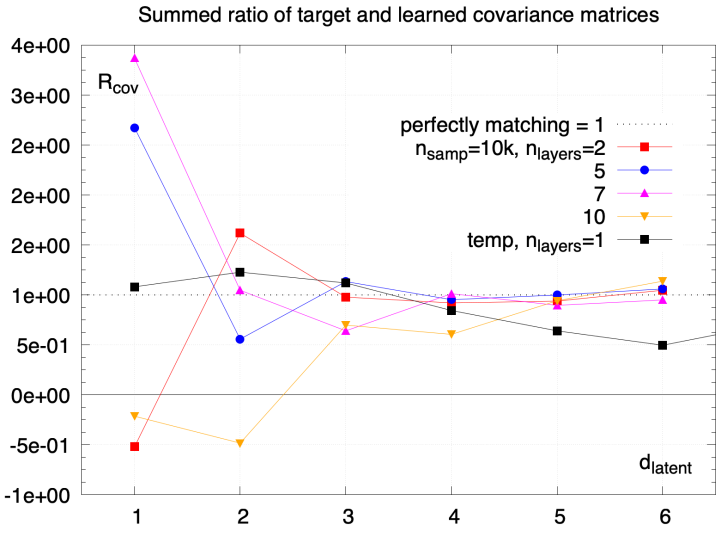
(b) Summed KL divergence of the projected densities



(c) Summed ratio of target and learned EVs



(d) Summed KL divergence from sampling the learned cov.mat.



(e) Summed ratio of target and learned cov.mat.s

FIG. 2: Single number measure results, please see text for details.

These ingredients enable us to study a few different indicators of how well the real cov.mat. has been learned.

Firstly, shown in Fig. 2(c) we can study the behaviour of the Eigenvalues. One way to visualise whether they are the same between the target and the learned cases is to look at their summed ratios:

$$R_{EV} = \frac{\lambda_x(real)}{\lambda_x(fake)} + \frac{\lambda_y(real)}{\lambda_y(fake)} + \frac{\lambda_z(real)}{\lambda_z(fake)}$$

- the idea is that this gives the overall relative deviation from the real case. A result close or equal to 1 would then indicate the EV's have been well learned. We observe this is the case for all runs with $d_{latent} \geq 3$.

Based on the learned cov.mat. we can also generate new samples that can be converted into KL divergences along the same lines of the previous section. The results are shown in Fig. 2(d). We observe a minimum value achieved of $KL_{sum} \simeq 1e^{-5}$.

Finally, we can also look at the cov.mat.s directly. For example we can look at the sum of their elementwise ratios, i.e. the relative deviation of each entry of the cov.mat. visualised in a single number. This is shown in Fig. 2(e). The results are mixed, however, the ideal result of 1 is approached around $d_{latent} = 3$. That some of the higher layered results do not reach the ideal value at this point could be an indication that they are not sufficiently converged yet. This merits further investigation.

III. OVERVIEW

To summarise, it seems the quantum GAN is indeed capable of learning to be a good approximation of the underlying physics and does capture the important correlations, when setup and trained well.

Things to do:

- It will be interesting to extend this to the LHC data case and to see how these observations carry over.
- In a separate, yet unfinished, run it also seemed that $n_{layer} = 1$ could provide an adequate surrogate model (shown in black in the figures), this could be interesting to verify as it would reduce the quantum circuit depth required.
- If it works with $n_{layers} = 1$ this would also be further indication that the number of latent dimensions is the relevant hyperparameter and that layers cannot take over the role the latent dimensions play.
- Further study of the convergence of the GAN setups, in particular dependent on the number of layers could be interesting. It would we good to verify that deviations from the ideal or perfectly matching scenarios are indeed due to undertraining or undersampling.
- Going further it could be useful to estimate the size of the accumulated errors through not having a perfect model, this could give a bound on how many samples can be generated safely.
- The current $d_{latent} = 3$ as good value could indicate a rule of thumb of $d_{latent} = D$, as we conjectured before as well. This could be verified by decreasing/increasing the dimensionality of our problem and performing the same analysis as here.

Appendix A: Example results for $n_{layers} = 5$

```
# ------------------------------------------------------------------------------------------
Checking Nqubits=3, latent_dim=1, layers=5
  Difference between exact covmat and after sampling, sum= 0.01198679694463975 and matrix:
[[ 0.00168647 -0.00051401 -0.00023555]
 [-0.00051401  0.00645194 -0.00010834]
 [-0.00023555 -0.00010834  0.00556418]]
  Difference of target and learned covmat, sum= -0.09452419263881154 and matrix:
[[ 0.0237237  -0.01506462  0.00298025]
 [-0.01506462 -0.0365453  -0.03280291]
 [ 0.00298025 -0.03280291  0.00807198]]
  Relative difference target and learned covmat, dmeas= 2.6735873805432178 and matrix:
[[0.23827218 3.39801367 0.8094436 ]
 [3.39801367 2.18473744 6.24278623]
 [0.8094436  6.24278623 0.7387898 ]]
  Eigenvalues of target and learned covmat:
  [0.01538321 0.02687432 0.05063606]
  [4.78756719e-05 6.37709881e-04 9.69576223e-02]
 Averaged ratio of Eigenvalues: 121.32662615137433
  KL divergences in (x,x), (y,y) and (z,z): 0.0010486517232 0.0069318471856 0.0030532662923

# ------------------------------------------------------------------------------------------
Checking Nqubits=3, latent_dim=2, layers=5
  Difference between exact covmat and after sampling, sum= -0.03936119511012656 and matrix:
[[-0.00214601 -0.00966366  0.00136619]
 [-0.00966366 -0.00562811 -0.00318301]
 [ 0.00136619 -0.00318301 -0.00862613]]
  Difference of target and learned covmat, sum= 0.027026833626518774 and matrix:
[[ 0.01113906  0.01201717 -0.01201197]
 [ 0.01201717  0.02118219  0.00467749]
 [-0.01201197  0.00467749 -0.01465979]]
  Relative difference target and learned covmat, dmeas= 0.5582975062949531 and matrix:
[[ 0.64507328 -0.75331235  1.77299044]
 [-0.75331235  0.32971493  0.27468799]
 [ 1.77299044  0.27468799  1.46115718]]
  Eigenvalues of target and learned covmat:
  [0.01603619 0.02717897 0.05155985]
  [9.89432851e-05 1.31294584e-02 6.38851502e-02]
 Averaged ratio of Eigenvalues: 54.98391700171371
  KL divergences in (x,x), (y,y) and (z,z): 0.0001539141289 0.0014901378694 0.0044090953997

# ------------------------------------------------------------------------------------------
Checking Nqubits=3, latent_dim=3, layers=5
  Difference between exact covmat and after sampling, sum= -0.05928154435683117 and matrix:
[[-0.0072412   0.00124053 -0.00880884]
 [ 0.00124053 -0.02005435 -0.0058771 ]
 [-0.00880884 -0.0058771  -0.00509518]]
  Difference of target and learned covmat, sum= -0.013266407000566579 and matrix:
[[-0.0134416  -0.00377314 -0.00319119]
 [-0.00377314  0.00583374  0.00164912]
 [-0.00319119  0.00164912  0.00497186]]
  Relative difference target and learned covmat, dmeas= 1.1339699475719878 and matrix:
[[1.42399064 1.61128485 1.19728471]
 [1.61128485 0.82051896 0.75078769]
 [1.19728471 0.75078769 0.84250543]]
  Eigenvalues of target and learned covmat:
  [0.01545217 0.02831707 0.05200518]
```

```
  [0.01427184 0.02321926 0.06091932]
 Averaged ratio of Eigenvalues: 1.0519759662133357
  KL divergences in (x,x), (y,y) and (z,z): 0.0001520321011 7.19484496e-05 6.31842921e-05

# ------------------------------------------------------------------------------------
Checking Nqubits=3, latent_dim=4, layers=5
  Difference between exact covmat and after sampling, sum= 0.021022324592579966 and matrix:
[[ 0.00628769  0.00203881  0.01262325]
 [ 0.00203881 -0.01052536 -0.0032216 ]
 [ 0.01262325 -0.0032216   0.00237907]]
  Difference of target and learned covmat, sum= 0.004463140948470955 and matrix:
[[ 0.00164411  0.0008757   0.00150448]
 [ 0.0008757   0.00023399 -0.00021929]
 [ 0.00150448 -0.00021929 -0.00173673]]
  Relative difference target and learned covmat, dmeas= 0.9527042452145663 and matrix:
[[0.94671845 0.85697169 0.89859325]
 [0.85697169 0.9926668  1.03399094]
 [0.89859325 1.03399094 1.05584118]]
  Eigenvalues of target and learned covmat:
  [0.01613969 0.02757623 0.05015025]
  [0.01755469 0.02746975 0.04870036]
 Averaged ratio of Eigenvalues: 0.9843474817523593
  KL divergences in (x,x), (y,y) and (z,z): 2.8396429e-06 2.1445399e-06 1.63587725e-05

# ------------------------------------------------------------------------------------
Checking Nqubits=3, latent_dim=5, layers=5
  Difference between exact covmat and after sampling, sum= -0.03429947567655989 and matrix:
[[-0.00520919 -0.00016507 -0.00763329]
 [-0.00016507 -0.0056034  -0.00228954]
 [-0.00763329 -0.00228954 -0.00331107]]
  Difference of target and learned covmat, sum= -0.0007359360966052719 and matrix:
[[-4.30517396e-04  3.41746375e-05  5.59618948e-04]
 [ 3.41746375e-05 -8.73167457e-04 -6.43764188e-05]
 [ 5.59618948e-04 -6.43764188e-05 -4.91085576e-04]]
  Relative difference target and learned covmat, dmeas= 0.9996211283440396 and matrix:
[[1.01363451 0.99454107 0.96524555]
 [0.99454107 1.0276317  1.01006968]
 [0.96524555 1.01006968 1.01561136]]
  Eigenvalues of target and learned covmat:
  [0.01541318 0.02760196 0.05161759]
  [0.01643237 0.02829249 0.05170264]
 Averaged ratio of Eigenvalues: 0.9706415690539442
  KL divergences in (x,x), (y,y) and (z,z): 1.846945e-07 8.88932e-07 3.5944667e-06

# ------------------------------------------------------------------------------------
Checking Nqubits=3, latent_dim=6, layers=5
  Difference between exact covmat and after sampling, sum= 0.0013553976179461746 and matrix:
[[-0.00243867  0.00280478  0.00261115]
 [ 0.00280478 -0.00692321 -0.00440816]
 [ 0.00261115 -0.00440816  0.00870175]]
  Difference of target and learned covmat, sum= -0.006665748062326426 and matrix:
[[-0.00172035 -0.00022736 -0.00013135]
 [-0.00022736 -0.00035814 -0.00108532]
 [-0.00013135 -0.00108532 -0.0016992 ]]
  Relative difference target and learned covmat, dmeas= 1.0606566188562583 and matrix:
[[1.05478405 1.03742742 1.00849507]
 [1.03742742 1.01130387 1.16631961]
 [1.00849507 1.16631961 1.05533745]]
```

```
 Eigenvalues of target and learned covmat:
 [0.01557559 0.02752268 0.05069298]
 [0.01707598 0.02751335 0.05297962]
Averaged ratio of Eigenvalues: 0.9564377432184054
 KL divergences in (x,x), (y,y) and (z,z): 2.9562424e-06 1.310023e-07 8.8315599e-06
```