Using BAYESGAN

### What exactly do you want BayesGAN to generate? I want both New synthetic rows similar to your dataset? (like data augmentation) and forecasting future values My dataset is all continuous features (like FX returns, equity prices etc) The dataset is 680 rows by 13 columns I want 1000 synthetic samples

**Based on your goal:**

* You want **correlations**.
* You want it to be **arbitrage-free**.
* You want realistic **multivariate** behavior.

👉 **You should definitely train a SINGLE GAN** across all 12 columns **together**.  
Otherwise, if you train 12 separate GANs, each feature would be independently sampled — and you could easily have unrealistic combinations (like FX-EQ spreads misbehaving, or outright arbitrage opportunities).

**Quick sketch of model flow if you want it *perfect*:**

1. Take your 12 columns (FX and Equity levels).
2. Convert them to **returns**: rt=log⁡(pt/pt−1)r\_t = \log(p\_t/p\_{t-1})rt​=log(pt​/pt−1​).
3. Train a **single BayesGAN** on the 12-dimensional returns.
4. After generation, **reconstruct prices** if needed by cumulative sums.
5. Validate that the generated synthetic dataset preserves cross-asset correlations (you can easily plot real vs synthetic correlation matrices).
6. Confirm no arbitrage-like behavior visually and statistically.

# Rerun the Model

Clone Repo:  
git clone <https://github.com/AIHawking187/bayesgan.git>  
**Find the bayesgan folder** inside it — it should have subfolders like models/, etc.

**Move** that bayesgan folder **into the same directory** where your Jupyter Notebook or script is.

cd "C:\Users\dullz\OneDrive\Documents\DropBox - OneDrive\bayesgan"

conda create -n bgan35 python=3.5

conda activate bgan35

pip install tensorflow==1.0.0

**Minorly adjust the repo code** to fix Python 2 print statements (print → print())

Run this simple command inside your repo folder (Windows PowerShell or cmd):

2to3 -w .

Because Conda **downloads pre-built binaries** (already compiled .dll files for Windows)

and Pip is **trying to compile** from C and Fortran — which fails without manual BLAS/LAPACK.

conda install scipy=0.19.1

**Change the division into integer division** (//) inside bgan\_util.py.

**Go to** this line inside bgan\_util.py (around line 103):

X = np.dot(np.random.randn(self.N / num\_clusters, self.true\_z\_dim) + cluster\_mean,

and change it to:

X = np.dot(np.random.randn(self.N // num\_clusters, self.true\_z\_dim) + cluster\_mean,

Then can run:  
 python bgan\_synth.py --x\_dim 100 --z\_dim 10 --numz 10 --out ./Results/

**Training begins:**

* The **Bayesian GAN (BGAN)** model **starts training**.
* It prints after every few iterations:
  + **Discriminator (Disc) loss**: How well the discriminator is doing.
  + **Generator (Gen) loss**: How well the generator is fooling the discriminator.

You can also see small numbers decreasing (0.009, 0.008, etc.), probably **learning rate decay** or **loss decay monitoring**.

**Periodic evaluation:**

* At different stages, your script **runs GMM (Gaussian Mixture Model)** on the synthetic samples.
* It prints:

*Fake number of clusters (AIC estimate): 9*

*JS div: 0.180*

* + **Fake number of clusters**: How many clusters were found by GMM (unsupervised check).
  + **JS divergence (JS div)**: A measure of how "different" the synthetic distribution is from the real one (lower = better).
* You trained a Bayesian GAN to generate 100-dimensional synthetic data, and although TensorFlow warned you about some missing operations (that you didn't actually use), the model trained normally, measured the quality of generated samples over time, and improved them.

# Here’s your custom dataset class for FX + EQ:

Save this as a new Python file inside your bayesgan folder, name it for example:  
**fx\_eq\_dataset.py**:

import numpy as np

class FXEQDataset:

def \_\_init\_\_(self, path, train\_split=0.65):

# Load data

self.data = np.loadtxt(path, delimiter=',', skiprows=1)

# Dataset size

self.dataset\_size = self.data.shape[0]

# Chronological split

split\_idx = int(self.dataset\_size \* train\_split)

self.train\_imgs = self.data[:split\_idx]

self.test\_imgs = self.data[split\_idx:]

# Reshape into (batch, 3, 4, 1)

self.train\_imgs = self.train\_imgs.reshape((-1, 3, 4, 1))

self.test\_imgs = self.test\_imgs.reshape((-1, 3, 4, 1))

# Define x\_dim for BGAN

self.x\_dim = (3, 4, 1)

# Dummy labels

self.train\_labels = np.zeros((self.train\_imgs.shape[0], 1))

self.test\_labels = np.zeros((self.test\_imgs.shape[0], 1))

def get\_train\_data(self):

return self.train\_imgs, self.train\_labels

def get\_test\_data(self):

return self.test\_imgs, self.test\_labels

def next\_batch(self, batch\_size, class\_id=None):

"""

Returns a batch of images from the training set.

"""

idx = np.random.randint(0, self.train\_imgs.shape[0], batch\_size)

return self.train\_imgs[idx], np.zeros((batch\_size, 1)) # returning dummy labels

Inside your **run\_bgan.py** script:

1. Add this import at the top:

from fx\_eq\_dataset import FXEQDataset

Find where the code chooses the dataset (maybe something like if args.dataset == "mnist": ...).

elif args.dataset == "fx\_eq":

dataset = FXEQDataset(path='../input/raw (FX + EQ).csv')

Then Train:

python run\_bgan.py --data\_path ./ --dataset fx\_eq --numz 10 --num\_mcmc 2 --out\_dir ./Results/ --train\_iter 75000 --save\_samples --n\_save 100

**12 columns** total — exactly what we need.

So **reshaping into (3, 4, 1)** is perfect (3 rows, 4 columns, 1 channel).  
That way, the GAN will **see each 12-feature vector** as a small "image."

The problem is simply **Python list vs tuple** mismatch in bgan.py when TensorFlow tries to define the placeholder shapes.

Inside your bgan.py, **find** this line (around line 235):

[self.batch\_size] + self.x\_dim

and **change it** to:

[self.batch\_size] + list(self.x\_dim)

Then Update line 99 and 100 inside run\_bgan.py to:

if isinstance(d\_losses, (list, tuple)):

                print("Disc losses = %s" % (", ".join(["%.2f" % float(dl) for dl in d\_losses])))

            else:

                print("Disc losses =", float(d\_losses))

 if isinstance(g\_losses, (list, tuple)):

                print("Gen losses = %s" % (", ".join(["%.2f" % float(gl) for gl in g\_losses])))

            else:

                print("Gen losses =", float(g\_losses))

Final:

**python run\_bgan.py --data\_path ./ --dataset fx\_eq --num\_gen 10 --num\_mcmc 2 --out\_dir ./Results/ --train\_iter 75000 --save\_samples --n\_save 100**