Modx

February 19, 2024

Motorcycle dataset analysis, found at http://vincentarelbundock.github.io/Rdatasets/datasets.html under 'mcycle'

[ ]:
```
import os
os.chdir('../')
```

[ ]:
```
import pandas as pd
import GPy
import numpy as np
from hetgp import HetGP
from svgp_multi import SVGPMulti
from sklearn.metrics import (r2_score, mean_squared_error, mean_absolute_error)
data_dir = './modx_data/'

def regression_metrics(y_true: np.array, y_pred: np.array):

    mse = mean_squared_error(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)

    metrics = {
        'Mean Squared Error (MSE)': np.float32(mse),
        'Mean Absolute Error (MAE)': np.float32(mae),
        'R-squared (R2)': np.float32(r2),
    }

    return metrics
```

[ ]:
```
restarts = 10
n_folds = 5
Ms = ['all']#, 100]
#These are just starting values, lengthscales will also be randomized
f_rbf_len = 1 # 0.3
f_rbf_var = 0.5
g_rbf_len = 1 # 0.5
g_rbf_var = 0.5
# This is the log of mean of the posterior of the scale parameter, so we set it
# to be the log of roughly what we expect the scale parameter to be if it were
```

```
# constant
gauss_noise = 0.1
g_mean_val = None  # np.log(gauss_noise)
g_bias_var = None
f_bias_var = None # ['mean']
fixZ = True
preopt_scg_iters = 100
preopt_restarts = 3
scg_iters = 50
max_iters = 250
num_samples = 100000
gtol = 1e-5
ftol = 0
xtol = 0
#gauss_dataset_names = []#['elevators1000']#, 'elevators10000']
#stut_dataset_names = ['motorCorrupt', 'elevators1000']#, 'elevators10000']
starting_df = 4.0
optimize_df = True

fold = 0
seed = 0

name = 'sim1.tsv'
```

[ ]:
```
np.random.seed(seed)
```

[ ]:
```
from sklearn.model_selection import train_test_split

def read_tsv(path: str, name: str):
    return pd.read_csv(os.path.join(path, name), sep='\t')

def std_matrices(*mts):
    std_mts = []
    for mats in mts:
        if isinstance(mats, np.ndarray):  # Single matrix case
            mats = [mats]
        for mat in mats:
            min_vals = np.min(mat, axis=0)
            max_vals = np.max(mat, axis=0)
            std_mat = (mat - min_vals) / (max_vals - min_vals)
            std_mts.append(std_mat)
    return tuple(std_mts)

def holdout_split(
        df,
        test_size: float = 0.1,
        random_state: int = None,
```

```
    ):
    y = df.Y.values
    x = df.values[:, :-1]
    return train_test_split(
            x, y,
            test_size=test_size,
            random_state=random_state
        )
```

[ ]:
```
class Dataset(object):
    def __init__(self, Xtrain, Ytrain, Xtest, Ytest):
        self.Xtrain, self.Ytrain, self.Xtest, self.Ytest = Xtrain, Ytrain,␣
 ↪Xtest, Ytest

def load_modx(seed):

    df = read_tsv(path=data_dir, name=name)
    Y = df.Y.values
    X = df.values[:, :-1]
    Xtrain, Xtest, Ytrain, Ytest = std_matrices(
        holdout_split(
            df=read_tsv(path=data_dir, name=name),
            random_state=seed
            )
        )
    Ytrain = Ytrain[:, None]
    Ytest = Ytest[:, None]
    print("training shape: ", Xtrain.shape)
    print("test shape: ", Xtest.shape)
    print("All: ", X.shape)
    print(Xtrain.shape[0] + Xtest.shape[0] - X.shape[0])
    return Dataset(Xtrain=Xtrain, Ytrain=Ytrain, Xtest=Xtest, Ytest=Ytest), X, Y

dataset, X, Y = load_modx(seed)

Xtrain = dataset.Xtrain
Ytrain = dataset.Ytrain
Xtest = dataset.Xtest
Ytest = dataset.Ytest
```

```
training shape:  (180, 10)
test shape:  (20, 10)
All:  (200, 10)
0
```

[ ]:
```
def build_kernf(D, dataset, bias, f_rbf_len, f_rbf_var, seed, fold):
    kernf = GPy.kern.RBF(D, variance=f_rbf_var,
```

```
                        lengthscale=np.ones(D)*f_rbf_len, ARD=True,
                        name='kernf_rbf')
    kernf += GPy.kern.White(1, variance=0.001, name='f_white')
    if bias is not None:
        if bias == 'mean':
            f_bias_var = dataset.Ytrain.mean()
        else:
            f_bias_var = bias
            kernf += GPy.kern.Bias(1, variance=f_bias_var, name='f_bias')
    kernf.f_white.fix()
    kernf.name = 'kernf'
    return kernf

def build_kerng(D, g_bias, g_rbf_len, seed, fold):
    # Needs white or variance doesn't checkgrad!
    kerng = GPy.kern.RBF(D, variance=g_rbf_var,
                        lengthscale=np.ones(D)*g_rbf_len, ARD=True,
                        name='kerng_rbf')
    kerng += GPy.kern.White(1, variance=0.001, name='g_white')
    if g_bias is not None:
        kerng += GPy.kern.Bias(1, variance=g_bias, name='g_bias')
    kerng.g_white.fix()
    kerng.name = 'kerng'
    return kerng
from scipy.cluster.vq import kmeans as scipy_kmeans

def kmeans(dataset, k, seed):
    Z, _ = scipy_kmeans(dataset.Xtrain, k)
    return Z

def build_gauss_model(dataset, Z, fixZ, bias, f_rbf_len, f_rbf_var, seed, fold):
    D = dataset.Xtrain.shape[1]
    kernf = build_kernf(D, dataset, bias, f_rbf_len, f_rbf_var, seed, fold)
    m_gauss = GPy.models.SparseGPRegression(dataset.Xtrain.copy(), dataset.
 ↪Ytrain.copy(), Z=Z.copy(), kernel=kernf)
    m_gauss.name='gauss_single'
    m_gauss.likelihood.variance[:] = gauss_noise
    return m_gauss

def preopt_gauss_scheme(m):
    m.kernf.constrain_positive()
    m.likelihood.variance.constrain_positive()
    if hasattr(m, 'Z'):
        m.Z.fix()
    m.optimize('bfgs', max_iters=preopt_scg_iters, gtol=gtol, messages=1,␣
 ↪xtol=xtol, ftol=ftol)
    if hasattr(m, 'Z'):
```

```
        m.Z.unfix()
    m.optimize('bfgs', max_iters=5*preopt_scg_iters, gtol=gtol, messages=1,␣
 ↪xtol=xtol, ftol=ftol)
    return m

def preopt_gauss(dataset, Z, fixZ, bias, f_rbf_len, f_rbf_var, seed, fold):
    m = build_gauss_model(dataset, Z, fixZ, bias, f_rbf_len, f_rbf_var, seed,␣
 ↪fold)
    print("PreOptimizing gauss ", m)
    print(m.kernf.kernf_rbf.lengthscale)
    best_m = preopt_gauss_scheme(m)
    print("Found best gauss model")
    print(best_m)
    return best_m[:]

m1_opt = preopt_gauss(dataset, Xtrain.copy(), fixZ, f_bias_var, f_rbf_len,␣
 ↪f_rbf_var, seed, fold)
m1 = build_gauss_model(dataset, Xtrain.copy(), fixZ, f_bias_var, f_rbf_len,␣
 ↪f_rbf_var, seed, fold)
m1[:] = m1_opt

print(m1)
print(m1.kernf.kernf_rbf.lengthscale)
m1.Z.unfix()
m1.optimize('bfgs', max_iters=3000, messages=1)
print(m1)
mu_gauss, var_gauss = m1.predict(Xtest)
print(mu_gauss)
metrics = regression_metrics(y_true=Ytest, y_pred=mu_gauss)
print(f'train: {metrics}')
```

```
reconstraining parameters gauss_single.kernf
```

```
PreOptimizing gauss
Name : gauss_single
Objective : 45.25088315164309
Number of Parameters : 1813
Number of Optimization Parameters : 1812
Updates : True
Parameters:
  gauss_single.                |      value  |  constraints  |  priors
  inducing_inputs              |  (180, 10)  |               |
  kernf.kernf_rbf.variance     |        0.5  |      +ve      |
  kernf.kernf_rbf.lengthscale  |      (10,)  |      +ve      |
  kernf.f_white.variance       |     0.001  |   +ve fixed   |
  Gaussian_noise.variance      |       0.1  |      +ve      |
  index  |  gauss_single.kernf.kernf_rbf.lengthscale  |  constraints
|  priors
```

```
  [0]    |                                  1.00000000  |      +ve
|
  [1]    |                                  1.00000000  |      +ve
|
  [2]    |                                  1.00000000  |      +ve
|
  [3]    |                                  1.00000000  |      +ve
|
  [4]    |                                  1.00000000  |      +ve
|
  [5]    |                                  1.00000000  |      +ve
|
  [6]    |                                  1.00000000  |      +ve
|
  [7]    |                                  1.00000000  |      +ve
|
  [8]    |                                  1.00000000  |      +ve
|
  [9]    |                                  1.00000000  |      +ve
|

reconstraining parameters gauss_single.Gaussian_noise.variance

Running L-BFGS-B (Scipy implementation) Code:
  runtime   i     f              |g|
WARNING: l-bfgs-b doesn't have an xtol arg, so I'm going to ignore it
WARNING: l-bfgs-b doesn't have an ftol arg, so I'm going to ignore it
    01s15  008  -2.584729e+02   4.456042e+03
    05s44  040  -6.671977e+02   1.046573e+02
    14s31  106  -7.345604e+02   3.262095e-01
Runtime:     14s31
Optimization status: Maximum number of f evaluations reached

Running L-BFGS-B (Scipy implementation) Code:
  runtime   i     f              |g|
WARNING: l-bfgs-b doesn't have an xtol arg, so I'm going to ignore it
WARNING: l-bfgs-b doesn't have an ftol arg, so I'm going to ignore it
    00s13  001  -7.345604e+02   6.587885e+04

 /home/rbarbano/projects/dev/ChainedGP/myenv/lib/python3.6/site-
packages/paramz/transformations.py:111: RuntimeWarning:overflow encountered in
expm1

    02s21  015  -7.345604e+02   1.483596e+05
    03s28  022  -7.345604e+02   6.587885e+04
Runtime:     03s28
Optimization status: Errorb'ABNORMAL_TERMINATION_IN_LNSRCH'

Found best gauss model
```

```
Name : gauss_single
Objective : -734.5604224896524
Number of Parameters : 1813
Number of Optimization Parameters : 1813
Updates : True
Parameters:
  gauss_single.                |                    value  |
constraints  |  priors
  inducing_inputs              |                (180, 10)  |
|
  kernf.kernf_rbf.variance     |       27.10048066891114  |      +ve
|
  kernf.kernf_rbf.lengthscale  |                   (10,)  |      +ve
|
  kernf.f_white.variance       |  5.562684646268137e-309  |      +ve
|
  Gaussian_noise.variance      |    4.70992791572058e-06  |      +ve
|

Name : gauss_single
Objective : -734.5604224896524
Number of Parameters : 1813
Number of Optimization Parameters : 1812
Updates : True
Parameters:
  gauss_single.                |                    value  |
constraints  |  priors
  inducing_inputs              |                (180, 10)  |
|
  kernf.kernf_rbf.variance     |       27.10048066891114  |      +ve
|
  kernf.kernf_rbf.lengthscale  |                   (10,)  |      +ve
|
  kernf.f_white.variance       |  5.562684646268137e-309  |   +ve
fixed   |
  Gaussian_noise.variance      |    4.70992791572058e-06  |      +ve
|
  index  |  gauss_single.kernf.kernf_rbf.lengthscale  |  constraints
|  priors
  [0]    |                            21087.03659660  |      +ve
|
  [1]    |                            21085.39646182  |      +ve
|
  [2]    |                            25418.93502051  |      +ve
|
  [3]    |                                0.27710782  |      +ve
|
  [4]    |                               12.15871401  |      +ve
```

```
|
  [5]    |                            17657.10949428  |      +ve
|
  [6]    |                            20191.03222636  |      +ve
|
  [7]    |                            24812.15887600  |      +ve
|
  [8]    |                            24601.51142006  |      +ve
|
  [9]    |                            21149.06386446  |      +ve
|
Running L-BFGS-B (Scipy implementation) Code:
  runtime   i      f              |g|
    01s23  0009  -7.345604e+02   8.264703e+04
    03s00  0022  -7.345604e+02   6.587885e+04
Runtime:     03s00
Optimization status: Errorb'ABNORMAL_TERMINATION_IN_LNSRCH'


Name : gauss_single
Objective : -734.5604224896524
Number of Parameters : 1813
Number of Optimization Parameters : 1812
Updates : True
Parameters:
  gauss_single.                |                   value  |
constraints  |  priors
  inducing_inputs              |               (180, 10)  |
|
  kernf.kernf_rbf.variance     |       27.10048066891114  |      +ve
|
  kernf.kernf_rbf.lengthscale  |                   (10,)  |      +ve
|
  kernf.f_white.variance       |  5.562684646268137e-309  |   +ve
fixed   |
  Gaussian_noise.variance      |    4.70992791572058e-06  |      +ve
|
[[0.01684967]
 [0.03034525]
 [0.02361767]
 [0.15330109]
 [0.01016346]
 [0.92311137]
 [0.00533783]
 [0.01223599]
 [0.00641353]
 [0.80164076]
 [0.00657335]
```

```
 [0.01073035]
 [0.01252087]
 [0.01275975]
 [0.0136043 ]
 [0.97913087]
 [0.00707842]
 [0.00720325]
 [0.01546857]
 [0.14795963]]
train: {'Mean Squared Error (MSE)': 0.00022804616, 'Mean Absolute Error (MAE)':
0.013531367, 'R-squared (R2)': 0.99764705}
```

[ ]:
```python
def random_multi_lengthscales(X_):
    normed_X = (X_.max(0) - X_.min(0))/X_.std(0)
    print(normed_X)
    f_lengthscales = np.random.uniform(size=X_.shape[1])*(0.4/normed_X) + 0.001
    g_lengthscales = np.random.uniform(size=X_.shape[1])*(0.4/normed_X) + 0.001
    print(f_lengthscales)
    return f_lengthscales, g_lengthscales
```

[ ]:
```python
# Make the kernels :
D = dataset.Xtrain.shape[1]
kernf = build_kernf(D, dataset, f_bias_var, f_rbf_len, f_rbf_var, seed, fold)
kerng = build_kerng(D, g_bias_var, g_rbf_len, seed, fold)
kern = [kernf, kerng]

# Multiple latent process model :
if g_mean_val is not None:
    g_mean = GPy.mappings.Constant(input_dim=13, output_dim=1, value=g_mean_val)
    print(g_mean)
    mean_functions = [None, g_mean]

mean_functions = [None, None]
likelihood = HetGP()

Z = dataset.Xtrain.copy()

# Make the model :
m2 = SVGPMulti(dataset.Xtrain.copy(), dataset.Ytrain.copy(), Z.copy(),␣
↪kern_list=kern,
                    likelihood=likelihood, mean_functions=mean_functions,␣
↪name='multi_gauss')


def pretrain_multi(m, randomize=False):
    if randomize:
        f_lens, g_lens = random_multi_lengthscales(m.X.values)
```

```
        m.kernf.kernf_rbf.lengthscale[:] = f_lens
        m.kernf.kernf_rbf.variance[:] = f_rbf_var
        m.kerng.kerng_rbf.lengthscale[:] = g_lens
        m.kerng.kerng_rbf.variance[:] = g_rbf_var
    m.kernf.fix()
    m.kerng.fix()
    if hasattr(m, 'Z'):
        m.Z.fix()
    if hasattr(m, 'constmap'):
        m.constmap.fix()
    print(m)

    # Optimize model with fixed parameters to get latent functions in place
    m.optimize('scg', max_iters=1*preopt_scg_iters, gtol=gtol, messages=1,␣
↪xtol=xtol, ftol=ftol)
    # Constrain all kernel parameters positive and reoptimize
    m.kernf.constrain_positive()
    m.kerng.constrain_positive()
    m.kernf.f_white.fix()
    m.kerng.g_white.fix()
    if hasattr(m, 'constmap'):
        m.constmap.unfix()

    # Continue with optimization with everything released
    m.optimize('scg', max_iters=1*preopt_scg_iters, gtol=gtol, messages=1,␣
↪xtol=xtol, ftol=ftol)
    m.optimize('bfgs', max_iters=5*preopt_scg_iters, gtol=gtol, messages=1,␣
↪xtol=xtol, ftol=ftol)

    return m

m2 = pretrain_multi(m2)
```

```
Name : multi_gauss
Objective : 10461.21433919741
Number of Parameters : 34764
Number of Optimization Parameters : 32940
Updates : True
Parameters:
  multi_gauss.                 |       value  |  constraints  |
priors
  inducing_inputs              |   (180, 10)  |     fixed     |
  q_u_means                    |    (180, 2)  |               |
  qf_u_chols                   |  (16290, 2)  |               |
  kernf.kernf_rbf.variance     |         0.5  |   +ve fixed   |
  kernf.kernf_rbf.lengthscale  |       (10,)  |   +ve fixed   |
```

```
  kernf.f_white.variance       |       0.001  |   +ve fixed   |
  kerng.kerng_rbf.variance     |         0.5  |   +ve fixed   |
  kerng.kerng_rbf.lengthscale  |       (10,)  |   +ve fixed   |
  kerng.g_white.variance       |       0.001  |   +ve fixed   |
Running Scaled Conjugate Gradients Code:
  runtime   i     f              |g|
    00s18  006   2.880737e+03   5.203363e+05
    03s21  097   1.260989e+02   1.070677e+03
    06s24  190   5.118501e+01   1.317155e+02
    09s92  302   3.933839e+01   2.266342e+01

reconstraining parameters multi_gauss.kernf
reconstraining parameters multi_gauss.kerng


Runtime:     09s92
Optimization status: maxiter exceeded

Running Scaled Conjugate Gradients Code:
  runtime   i     f              |g|
    00s15  002   3.933839e+01   1.779183e+03
    03s27  042  -2.795209e+01   2.433270e+03
    10s58  135  -8.567871e+01   2.311578e+03
    23s18  302  -2.073071e+02   1.355440e+03
Runtime:     23s18
Optimization status: maxiter exceeded

Running L-BFGS-B (Scipy implementation) Code:
  runtime   i     f              |g|
WARNING: l-bfgs-b doesn't have an xtol arg, so I'm going to ignore it
WARNING: l-bfgs-b doesn't have an ftol arg, so I'm going to ignore it
    00s07  000  -2.073071e+02   1.217967e+03
    02s12  024  -2.152608e+02   6.938998e+02
    06s16  077  -2.451016e+02   1.001766e+02
    20s48  244  -2.735892e+02   3.087948e+02
    31s75  376  -2.912170e+02   1.077876e+01
    41s66  502  -2.929021e+02   5.410048e+00
Runtime:     41s66
Optimization status: Maximum number of f evaluations reached
```

[ ]:
```
print(m2)
print(m2.kernf.kernf_rbf.lengthscale)
print(m2.kerng.kerng_rbf.lengthscale)

m2.Z.unfix()
m2.optimize('bfgs', max_iters=100, messages=1)
print(m2)
```

```
mu_multi_gauss, _ = m2._raw_predict(Xtest, 0)
var_multi_gauss, _ = m2._raw_predict(Xtest, 1)
print(mu_multi_gauss)
metrics = regression_metrics(y_true=Ytest, y_pred=mu_multi_gauss)
print(f'train: {metrics}')
```

```
Name : multi_gauss
Objective : -292.9021089576384
Number of Parameters : 34764
Number of Optimization Parameters : 32962
Updates : True
Parameters:
  multi_gauss.                  |                value  |  constraints
|  priors
  inducing_inputs               |            (180, 10)  |     fixed
|
  q_u_means                     |             (180, 2)  |
|
  qf_u_chols                    |           (16290, 2)  |
|
  kernf.kernf_rbf.variance      |  0.40964320498198514  |      +ve
|
  kernf.kernf_rbf.lengthscale   |                (10,)  |      +ve
|
  kernf.f_white.variance        |                0.001  |   +ve fixed
|
  kerng.kerng_rbf.variance      |   19.438170932308232  |      +ve
|
  kerng.kerng_rbf.lengthscale   |                (10,)  |      +ve
|
  kerng.g_white.variance        |                0.001  |   +ve fixed
|
  index  |  multi_gauss.kernf.kernf_rbf.lengthscale  |  constraints  |
priors
  [0]    |                              55.71511535  |      +ve      |
  [1]    |                              57.27557082  |      +ve      |
  [2]    |                              58.29194648  |      +ve      |
  [3]    |                               0.15101946  |      +ve      |
  [4]    |                              50.70320208  |      +ve      |
  [5]    |                              56.81047999  |      +ve      |
  [6]    |                              57.48156752  |      +ve      |
  [7]    |                              59.18003259  |      +ve      |
  [8]    |                              57.05838860  |      +ve      |
  [9]    |                              58.28034595  |      +ve      |
  index  |  multi_gauss.kerng.kerng_rbf.lengthscale  |  constraints  |
priors
```

```
  [0]    |                              26.57237259  |      +ve      |
  [1]    |                              28.66996081  |      +ve      |
  [2]    |                              27.65806290  |      +ve      |
  [3]    |                              13.57784299  |      +ve      |
  [4]    |                              20.67961642  |      +ve      |
  [5]    |                              27.69144574  |      +ve      |
  [6]    |                              27.21175572  |      +ve      |
  [7]    |                              29.37792282  |      +ve      |
  [8]    |                              26.52774611  |      +ve      |
  [9]    |                              27.81884634  |      +ve      |
Running L-BFGS-B (Scipy implementation) Code:
  runtime   i     f              |g|
    02s22  011  -2.929377e+02   4.043953e+02
    09s03  044  -2.934837e+02   7.479588e+02
    20s68  102  -2.938297e+02   1.276985e+02
Runtime:     20s68
Optimization status: Maximum number of f evaluations reached


Name : multi_gauss
Objective : -293.82973246347353
Number of Parameters : 34764
Number of Optimization Parameters : 34762
Updates : True
Parameters:
  multi_gauss.                 |               value  |  constraints
|  priors
  inducing_inputs              |           (180, 10)  |
|
  q_u_means                    |            (180, 2)  |
|
  qf_u_chols                   |          (16290, 2)  |
|
  kernf.kernf_rbf.variance     |   0.404691995158725  |      +ve
|
  kernf.kernf_rbf.lengthscale  |              (10,)  |      +ve
|
  kernf.f_white.variance       |               0.001  |   +ve fixed
|
  kerng.kerng_rbf.variance     |  19.434818463972896  |      +ve
|
  kerng.kerng_rbf.lengthscale  |              (10,)  |      +ve
|
  kerng.g_white.variance       |               0.001  |   +ve fixed
|
[[0.00919811]
 [0.03616447]
 [0.01377627]
```

```
 [0.15506812]
 [0.00688926]
 [0.89801689]
 [0.00689866]
 [0.00596226]
 [0.01464663]
 [0.79729051]
 [0.00665584]
 [0.00703708]
 [0.00728649]
 [0.00703954]
 [0.00914823]
 [0.94973018]
 [0.01389281]
 [0.00708392]
 [0.00736825]
 [0.14763896]]
train: {'Mean Squared Error (MSE)': 0.00044020778, 'Mean Absolute Error (MAE)':
0.017868891, 'R-squared (R2)': 0.995458}
```

[ ]:
```
print(m2.kernf.kernf_rbf.lengthscale)
print(m2.kerng.kerng_rbf.lengthscale)
```

```
  index  |  multi_gauss.kernf.kernf_rbf.lengthscale  |  constraints  |
priors
  [0]    |                              55.71956982  |      +ve      |
  [1]    |                              57.28003813  |      +ve      |
  [2]    |                              58.29654779  |      +ve      |
  [3]    |                               0.15278995  |      +ve      |
  [4]    |                              50.70665552  |      +ve      |
  [5]    |                              56.81488811  |      +ve      |
  [6]    |                              57.48616366  |      +ve      |
  [7]    |                              59.18433911  |      +ve      |
  [8]    |                              57.06301059  |      +ve      |
  [9]    |                              58.28468622  |      +ve      |
  index  |  multi_gauss.kerng.kerng_rbf.lengthscale  |  constraints  |
priors
  [0]    |                              26.57399249  |      +ve      |
  [1]    |                              28.67053263  |      +ve      |
  [2]    |                              27.65944747  |      +ve      |
  [3]    |                              13.58434484  |      +ve      |
  [4]    |                              20.68191456  |      +ve      |
  [5]    |                              27.69241620  |      +ve      |
  [6]    |                              27.21344969  |      +ve      |
  [7]    |                              29.37865580  |      +ve      |
  [8]    |                              26.52955190  |      +ve      |
  [9]    |                              27.81962121  |      +ve      |
```

[ ]:
```
np.exp(var_multi_gauss)
```

[ ]:
```
array([[0.00137611],
       [0.00141226],
       [0.00137682],
       [0.00143073],
       [0.00138889],
       [0.00142263],
       [0.00139716],
       [0.00137794],
       [0.00141646],
       [0.00139194],
       [0.00139698],
       [0.00141782],
       [0.00136795],
       [0.00143066],
       [0.0013674 ],
       [0.00141695],
       [0.00139434],
       [0.00140966],
       [0.00141985],
       [0.00139894]])
```