sim3

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 = 'sim3.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:  (90, 10)
test shape:  (10, 10)
All:  (100, 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 : 27.600127790319092
Number of Parameters : 913
Number of Optimization Parameters : 912
Updates : True
Parameters:
  gauss_single.                |     value  |  constraints  |  priors
  inducing_inputs              |  (90, 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
    01s21  011  -2.307728e+01   2.079922e-01
    02s27  020  -2.394675e+01   1.205119e-01
    05s41  047  -2.411326e+01   3.335780e-03
    08s71  076  -2.411518e+01   2.925295e-03
    11s81  102  -2.411544e+01   3.384313e-05
Runtime:     11s81
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
    00s08  001  -2.411544e+01   3.384416e-05
    02s18  018  -2.411554e+01   3.330773e-05
    05s25  043  -2.411559e+01   1.402632e-07
    06s33  053  -2.411560e+01   2.869662e-07
Runtime:     06s33
Optimization status: Converged

Found best gauss model

Name : gauss_single
```

```
Objective : -24.1155960447727
Number of Parameters : 913
Number of Optimization Parameters : 913
Updates : True
Parameters:
  gauss_single.                |                  value  |
constraints  |  priors
  inducing_inputs              |              (90, 10)  |
|
  kernf.kernf_rbf.variance     |     0.5501847805472116  |      +ve
|
  kernf.kernf_rbf.lengthscale  |                 (10,)  |      +ve
|
  kernf.f_white.variance       |  3.755771246071219e-10  |      +ve
|
  Gaussian_noise.variance      |   0.027182919127092208  |      +ve
|

Name : gauss_single
Objective : -24.1155960447727
Number of Parameters : 913
Number of Optimization Parameters : 912
Updates : True
Parameters:
  gauss_single.                |                  value  |
constraints  |  priors
  inducing_inputs              |              (90, 10)  |
|
  kernf.kernf_rbf.variance     |     0.5501847805472116  |      +ve
|
  kernf.kernf_rbf.lengthscale  |                 (10,)  |      +ve
|
  kernf.f_white.variance       |  3.755771246071219e-10  |   +ve fixed
|
  Gaussian_noise.variance      |   0.027182919127092208  |      +ve
|
  index  |  gauss_single.kernf.kernf_rbf.lengthscale  |  constraints
|  priors
  [0]    |                                8.83314930  |      +ve
|
  [1]    |                                8.04645510  |      +ve
|
  [2]    |                                5.96912527  |      +ve
|
  [3]    |                                5.22305494  |      +ve
|
  [4]    |                                6.38462090  |      +ve
|
```

```
  [5]    |                                14.13460097  |      +ve
|
  [6]    |                                11.79109088  |      +ve
|
  [7]    |                                 6.10872534  |      +ve
|
  [8]    |                                 4.08833581  |      +ve
|
  [9]    |                                11.00249428  |      +ve
|
Running L-BFGS-B (Scipy implementation) Code:
  runtime   i      f              |g|
    00s08  0001  -2.411560e+01   2.869648e-07
    00s48  0004  -2.411560e+01   7.636115e-08
Runtime:     00s48
Optimization status: Converged


Name : gauss_single
Objective : -24.115596049570627
Number of Parameters : 913
Number of Optimization Parameters : 912
Updates : True
Parameters:
  gauss_single.               |                  value  |
constraints  |  priors
  inducing_inputs             |              (90, 10)  |
|
  kernf.kernf_rbf.variance    |     0.5501849406950954  |      +ve
|
  kernf.kernf_rbf.lengthscale |                  (10,)  |      +ve
|
  kernf.f_white.variance      |  3.755771246071219e-10  |   +ve fixed
|
  Gaussian_noise.variance     |    0.02718253191382625  |      +ve
|
[[0.50352948]
 [0.42292685]
 [0.32296903]
 [0.56375191]
 [0.53098842]
 [0.40857516]
 [0.68371398]
 [0.25909172]
 [0.40824183]
 [0.49348227]]
train: {'Mean Squared Error (MSE)': 0.04835533, 'Mean Absolute Error (MAE)':
0.18078122, 'R-squared (R2)': 0.45183206}
```

[ ]:
```
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
```

[ ]:
```
# 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 : 2438.3208829856667
Number of Parameters : 9294
Number of Optimization Parameters : 8370
Updates : True
Parameters:
  multi_gauss.                |      value  |  constraints  |  priors
  inducing_inputs             |   (90, 10)  |     fixed     |
  q_u_means                   |    (90, 2)  |               |
  qf_u_chols                  |  (4095, 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|
    00s11  008   3.337792e+02   4.033348e+04
    00s19  015   2.471028e+02   7.633304e+03
    01s20  075   7.642674e+01   2.192320e+02
    03s21  192   4.194508e+01   1.465574e+01
    05s05  302   4.098008e+01   4.850772e-01
reconstraining parameters multi_gauss.kernf
reconstraining parameters multi_gauss.kerng
```

```
Runtime:     05s05
Optimization status: maxiter exceeded

Running Scaled Conjugate Gradients Code:
  runtime   i     f              |g|
    00s16  004   3.318364e+01   4.671793e+02
    01s17  029   1.651501e+01   5.392064e+02
    05s29  132  -2.269755e+00   1.707019e+02
    09s34  233  -1.134619e+01   1.537391e+02
    12s13  302  -1.371387e+01   7.810738e+01
Runtime:     12s13
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
    00s16  003  -1.375764e+01   2.797429e+01
    02s22  050  -1.573935e+01   9.808751e+01
    06s27  148  -1.849268e+01   2.954760e+01
    19s58  475  -1.994491e+01   2.028666e+00
    20s69  502  -2.002190e+01   3.789635e+00
Runtime:     20s69
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 : -20.02189706280341
Number of Parameters : 9294
Number of Optimization Parameters : 8392
Updates : True
Parameters:
  multi_gauss.                |                value  |  constraints
```

```
|  priors
  inducing_inputs               |            (90, 10)  |     fixed
|
  q_u_means                     |             (90, 2)  |
|
  qf_u_chols                    |           (4095, 2)  |
|
  kernf.kernf_rbf.variance      |  0.2916027403740681  |      +ve
|
  kernf.kernf_rbf.lengthscale   |               (10,)  |      +ve
|
  kernf.f_white.variance        |              0.001   |   +ve fixed
|
  kerng.kerng_rbf.variance      |   5.622388298263775  |      +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]    |                              13.57230035  |      +ve      |
  [1]    |                               7.94232437  |      +ve      |
  [2]    |                              10.19188783  |      +ve      |
  [3]    |                               3.68308892  |      +ve      |
  [4]    |                               9.51018733  |      +ve      |
  [5]    |                              12.15174001  |      +ve      |
  [6]    |                              10.71813528  |      +ve      |
  [7]    |                               4.98546459  |      +ve      |
  [8]    |                               2.83820830  |      +ve      |
  [9]    |                               4.83737225  |      +ve      |
  index  |  multi_gauss.kerng.kerng_rbf.lengthscale  |  constraints  |
priors
  [0]    |                               3.28251966  |      +ve      |
  [1]    |                               9.84848567  |      +ve      |
  [2]    |                               8.30963984  |      +ve      |
  [3]    |                              10.07874610  |      +ve      |
  [4]    |                               8.35076963  |      +ve      |
  [5]    |                               0.85780015  |      +ve      |
  [6]    |                               8.46479646  |      +ve      |
  [7]    |                               8.50942853  |      +ve      |
  [8]    |                              10.58917482  |      +ve      |
  [9]    |                               2.95263037  |      +ve      |
Running L-BFGS-B (Scipy implementation) Code:
  runtime   i     f              |g|
    00s15  001  -2.002190e+01   4.781652e+00
    03s36  022  -2.003915e+01   2.037872e+00
    12s04  079  -2.006985e+01   3.111215e+01
```

```
    15s75  102  -2.007916e+01   6.017257e+00
Runtime:     15s75
Optimization status: Maximum number of f evaluations reached


Name : multi_gauss
Objective : -20.07915972077836
Number of Parameters : 9294
Number of Optimization Parameters : 9292
Updates : True
Parameters:
  multi_gauss.                 |               value  |  constraints
|  priors
  inducing_inputs              |            (90, 10)  |
|
  q_u_means                    |             (90, 2)  |
|
  qf_u_chols                   |           (4095, 2)  |
|
  kernf.kernf_rbf.variance     |  0.2816189589135138  |      +ve
|
  kernf.kernf_rbf.lengthscale  |              (10,)  |      +ve
|
  kernf.f_white.variance       |               0.001  |   +ve fixed
|
  kerng.kerng_rbf.variance     |   5.637655134228079  |      +ve
|
  kerng.kerng_rbf.lengthscale  |              (10,)  |      +ve
|
  kerng.g_white.variance       |               0.001  |   +ve fixed
|
[[0.41869876]
 [0.38854038]
 [0.37592674]
 [0.51870381]
 [0.4870561 ]
 [0.3682724 ]
 [0.66577931]
 [0.27501269]
 [0.41246048]
 [0.46120484]]
train: {'Mean Squared Error (MSE)': 0.04921519, 'Mean Absolute Error (MAE)':
0.18096557, 'R-squared (R2)': 0.44208446}
```

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

```
  index  |  multi_gauss.kernf.kernf_rbf.lengthscale  |  constraints  |
```

```
priors
  [0]    |                                13.59403419  |      +ve      |
  [1]    |                                 7.94011704  |      +ve      |
  [2]    |                                10.20101807  |      +ve      |
  [3]    |                                 3.67450479  |      +ve      |
  [4]    |                                 9.51506189  |      +ve      |
  [5]    |                                12.16611876  |      +ve      |
  [6]    |                                10.72969814  |      +ve      |
  [7]    |                                 5.00580850  |      +ve      |
  [8]    |                                 2.85670260  |      +ve      |
  [9]    |                                 4.82792921  |      +ve      |
  index  |  multi_gauss.kerng.kerng_rbf.lengthscale   |  constraints  |
priors
  [0]    |                                 3.25465204  |      +ve      |
  [1]    |                                 9.85998332  |      +ve      |
  [2]    |                                 8.32164707  |      +ve      |
  [3]    |                                10.09187642  |      +ve      |
  [4]    |                                 8.36315053  |      +ve      |
  [5]    |                                 0.85559144  |      +ve      |
  [6]    |                                 8.47708501  |      +ve      |
  [7]    |                                 8.51724039  |      +ve      |
  [8]    |                                10.60363555  |      +ve      |
  [9]    |                                 2.96353077  |      +ve      |
```

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

[ ]:
```
array([[0.05670662],
       [0.03116187],
       [0.01534846],
       [0.04768099],
       [0.02711849],
       [0.02197655],
       [0.01808694],
       [0.06077603],
       [0.01381278],
       [0.02740834]])
```