Contents

1 Main function

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Nov  8 14:54:01 2020

@author: nastavirs
"""

import tensorflow as tf
import numpy as np
import time
import scipy.io
np.random.seed(1234)
tf.set_random_seed(1234)
class NSPINN:
    # notational conventions
    # _tf: placeholders for input/output data and points used to regress the equations
    # _pred: output of neural network
    # _data: input-output data
    # _star: preditions
    from init_NN import initialize_NN
    from Xavi_init import xavier_init
    from NN import neural_net
    from Sup_NN import net_NS
    from Unsup_NN import net_f_NS
    from Callback import callback
    from Adam_train import Adam_train
    from BGFS_train import BFGS_train
```

```
    from predict import predict
    def __init__(self, xi, yi, zi, ti, ui, vi, wi, xb, yb, zb, tb,
    ub, vb, wb, x, y, z, t, layers):
        xyzt_i = np.concatenate([xi, yi, zi, ti], 1)
        xyzt_b = np.concatenate([xb, yb, zb, tb], 1)
        xyzt = np.concatenate([x, y, z, t], 1)

        self.lowb = xyzt_b.min(0)
        self.upb = xyzt_b.max(0)

        self.xyzt_i = xyzt_i
        self.xyzt_b = xyzt_b
        self.xyzt = xyzt

        self.xi = xyzt_i[:, 0:1]
        self.yi = xyzt_i[:, 1:2]
        self.zi = xyzt_i[:, 2:3]
        self.ti = xyzt_i[:, 3:4]

        self.xb = xyzt_b[:, 0:1]
        self.yb = xyzt_b[:, 1:2]
        self.zb = xyzt_b[:, 2:3]
        self.tb = xyzt_b[:, 3:4]

        self.x = xyzt[:, 0:1]
        self.y = xyzt[:, 1:2]
        self.z = xyzt[:, 2:3]
        self.t = xyzt[:, 3:4]

        self.ui = ui
        self.vi = vi
        self.wi = wi

        self.ub = ub
        self.vb = vb
        self.wb = wb

        self.layers = layers

        self.weights, self.biases = self.initialize_NN(layers)

        self.learning_rate = tf.placeholder(tf.float32, shape=[])

        self.sess = tf.Session(config=tf.ConfigProto(
    allow_soft_placement=True,

    log_device_placement=True))

        self.x_ini_tf = tf.placeholder(tf.float32, shape=[None,
    self.xi.shape[1]])
        self.y_ini_tf = tf.placeholder(tf.float32, shape=[None,
    self.yi.shape[1]])
        self.z_ini_tf = tf.placeholder(tf.float32, shape=[None,
    self.zi.shape[1]])
        self.t_ini_tf = tf.placeholder(tf.float32, shape=[None,
    self.ti.shape[1]])
        self.u_ini_tf = tf.placeholder(tf.float32, shape=[None,
```

```
self.ui.shape[1]])
        self.v_ini_tf = tf.placeholder(tf.float32, shape=[None,
self.vi.shape[1]])
        self.w_ini_tf = tf.placeholder(tf.float32, shape=[None,
self.wi.shape[1]])

        self.x_boundary_tf = tf.placeholder(tf.float32, shape=[None
, self.xb.shape[1]])
        self.y_boundary_tf = tf.placeholder(tf.float32, shape=[None
, self.yb.shape[1]])
        self.z_boundary_tf = tf.placeholder(tf.float32, shape=[None
, self.zb.shape[1]])
        self.t_boundary_tf = tf.placeholder(tf.float32, shape=[None
, self.tb.shape[1]])
        self.u_boundary_tf = tf.placeholder(tf.float32, shape=[None
, self.ub.shape[1]])
        self.v_boundary_tf = tf.placeholder(tf.float32, shape=[None
, self.vb.shape[1]])
        self.w_boundary_tf = tf.placeholder(tf.float32, shape=[None
, self.wb.shape[1]])

        self.x_tf = tf.placeholder(tf.float32, shape=[None, self.x.
shape[1]])
        self.y_tf = tf.placeholder(tf.float32, shape=[None, self.y.
shape[1]])
        self.z_tf = tf.placeholder(tf.float32, shape=[None, self.z.
shape[1]])
        self.t_tf = tf.placeholder(tf.float32, shape=[None, self.t.
shape[1]])

        self.u_ini_pred, self.v_ini_pred, self.w_ini_pred, self.
p_ini_pred = \
            self.net_NS(self.x_ini_tf, self.y_ini_tf, self.z_ini_tf
, self.t_ini_tf)
        self.u_boundary_pred, self.v_boundary_pred, self.
w_boundary_pred, self.p_boundary_pred = \
            self.net_NS(self.x_boundary_tf, self.y_boundary_tf,
self.z_boundary_tf, self.t_boundary_tf)
        self.u_pred, self.v_pred, self.w_pred, self.p_pred, self.
f_u_pred, self.f_v_pred, self.f_w_pred, self.f_e_pred = \
            self.net_f_NS(self.x_tf, self.y_tf, self.z_tf, self.
t_tf)

        alpha = 100
        beta = 100

        self.loss = alpha * tf.reduce_mean(tf.square(self.u_ini_tf
- self.u_ini_pred)) + \
                    alpha * tf.reduce_mean(tf.square(self.v_ini_tf
- self.v_ini_pred)) + \
                    alpha * tf.reduce_mean(tf.square(self.w_ini_tf
- self.w_ini_pred)) + \
                    beta * tf.reduce_mean(tf.square(self.
u_boundary_tf - self.u_boundary_pred)) + \
                    beta * tf.reduce_mean(tf.square(self.
v_boundary_tf - self.v_boundary_pred)) + \
                    beta * tf.reduce_mean(tf.square(self.
```

```
    w_boundary_tf - self.w_boundary_pred)) + \
                    tf.reduce_mean(tf.square(self.f_u_pred)) + \
                    tf.reduce_mean(tf.square(self.f_v_pred)) + \
                    tf.reduce_mean(tf.square(self.f_w_pred)) + \
                    tf.reduce_mean(tf.square(self.f_e_pred))

        self.optimizer = tf.contrib.opt.ScipyOptimizerInterface(
    self.loss,
    method='L-BFGS-B',
    options={'maxiter': 50000,
          'maxfun': 50000,
          'maxcor': 50,
          'maxls': 50,
          'ftol': 1.0 * np.finfo(float).eps})

        self.optimizer_Adam = tf.train.AdamOptimizer(self.
    learning_rate)
        self.train_op_Adam = self.optimizer_Adam.minimize(self.loss
    )

        init = tf.global_variables_initializer()
        self.sess.run(init)



if __name__ == "__main__":

    N_train = 10000

    layers = [4, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 4]

    def data_generate(x, y, z, t):

        a, d = 1, 1
        u = - a * (np.exp(a * x) * np.sin(a * y + d * z) + np.exp(a
     * z) * np.cos(a * x + d * y)) * np.exp(- d * d * t)
        v = - a * (np.exp(a * y) * np.sin(a * z + d * x) + np.exp(a
     * x) * np.cos(a * y + d * z)) * np.exp(- d * d * t)
        w = - a * (np.exp(a * z) * np.sin(a * x + d * y) + np.exp(a
     * y) * np.cos(a * z + d * x)) * np.exp(- d * d * t)
        p = - 0.5 * a * a * (np.exp(2 * a * x) + np.exp(2 * a * y)
    + np.exp(2 * a * z) +
                             2 * np.sin(a * x + d * y) * np.cos(a *
     z + d * x) * np.exp(a * (y + z)) +
                             2 * np.sin(a * y + d * z) * np.cos(a *
     x + d * y) * np.exp(a * (z + x)) +
                             2 * np.sin(a * z + d * x) * np.cos(a *
     y + d * z) * np.exp(a * (x + y))) * np.exp(
            -2 * d * d * t)

        return u, v, w, p
```

```

    xdata = np.linspace(-1, 1, 31)
    ydata = np.linspace(-1, 1, 31)
    zdata = np.linspace(-1, 1, 31)
    tdata = np.linspace(0, 1, 11)
    b0 = np.array([-1] * 900)
    b1 = np.array([1] * 900)

   #boundary values
    xr1 = np.tile(xdata[0:30], 30)
    yr1 = np.tile(ydata[0:30], 30)
    zr1 = np.tile(zdata[0:30], 30)
    xr2 = np.tile(xdata[1:31], 30)
    yr2 = np.tile(ydata[1:31], 30)
    zc2 = np.tile(zdata[1:31], 30)

    xc1 = xdata[0:30].repeat(30)
    yc1 = ydata[0:30].repeat(30)
    zc1 = zdata[0:30].repeat(30)
    xc2 = xdata[1:31].repeat(30)
    yc2 = ydata[1:31].repeat(30)
    zr1 = zdata[1:31].repeat(30)

    trainx = np.concatenate([b1, b0, xr2, xr1, xr2, xr1], 0).repeat
    (tdata.shape[0])
    trainy = np.concatenate([yr1, yr2, b1, b0, yc2, yc1], 0).repeat
    (tdata.shape[0])
    trainz = np.concatenate([zc1, zr1, zc1, zr1, b1, b0], 0).repeat
    (tdata.shape[0])
    traint = np.tile(tdata, 5400)

    trainub, trainvb, trainwb, trainpb = data_generate(trainx,
    trainy, trainz, traint)

    xb_train = trainx.reshape(trainx.shape[0], 1)
    yb_train = trainx.reshape(trainy.shape[0], 1)
    zb_train = trainx.reshape(trainz.shape[0], 1)
    tb_train = trainx.reshape(traint.shape[0], 1)
    ub_train = trainx.reshape(trainub.shape[0], 1)
    vb_train = trainx.reshape(trainvb.shape[0], 1)
    wb_train = trainx.reshape(trainwb.shape[0], 1)
    pb_train = trainx.reshape(trainpb.shape[0], 1)

    # inital values
    x_0 = np.tile(xdata, 31 * 31)
    y_0 = np.tile(ydata.repeat(31), 31)
    z_0 = zdata.repeat(31 * 31)
    t_0 = np.array([0] * x_0.shape[0])

    u_0, v_0, w_0, p_0 = data_generate(x_0, y_0, z_0, t_0)

    ui_train = u_0.reshape(u_0.shape[0], 1)
    vi_train = v_0.reshape(v_0.shape[0], 1)
    wi_train = w_0.reshape(w_0.shape[0], 1)
    p0_train = p_0.reshape(p_0.shape[0], 1)
    xi_train = x_0.reshape(x_0.shape[0], 1)
    yi_train = y_0.reshape(y_0.shape[0], 1)
```

```
    zi_train = z_0.reshape(z_0.shape[0], 1)
    ti_train = t_0.reshape(t_0.shape[0], 1)

    # xyzt data
    xx = np.random.randint(31, size=10000) / 15 - 1
    yy = np.random.randint(31, size=10000) / 15 - 1
    zz = np.random.randint(31, size=10000) / 15 - 1
    tt = np.random.randint(11, size=10000) / 10

    uu, vv, ww, pp = data_generate(xx, yy, zz, tt)

    x_train = xx.reshape(xx.shape[0], 1)
    y_train = yy.reshape(yy.shape[0], 1)
    z_train = zz.reshape(zz.shape[0], 1)
    t_train = tt.reshape(tt.shape[0], 1)

    model = NSPINN(xi_train, yi_train, zi_train, ti_train,
                   ui_train, vi_train, wi_train,
                   xb_train, yb_train, zb_train, tb_train,
                   ub_train, vb_train, wb_train,
                   x_train, y_train, z_train, t_train, layers)

    model.Adam_train(5000, 1e-3)
    model.Adam_train(5000, 1e-4)
    model.Adam_train(50000, 1e-5)
    model.Adam_train(50000, 1e-6)
    model.BFGS_train()

    x_star = (np.random.rand(1000, 1) - 1 / 2) * 2
    y_star = (np.random.rand(1000, 1) - 1 / 2) * 2
    z_star = (np.random.rand(1000, 1) - 1 / 2) * 2
    t_star = np.random.randint(11, size=(100, 1)) / 10

    u_star, v_star, w_star, p_star = data_generate(x_star, y_star, z_star, t_star)


    u_pred, v_pred, w_pred, p_pred = model.predict(x_star, y_star, z_star, t_star)

    # Error
    error_u = np.linalg.norm(u_star - u_pred, 2) / np.linalg.norm(u_star, 2)
    error_v = np.linalg.norm(v_star - v_pred, 2) / np.linalg.norm(v_star, 2)
    error_w = np.linalg.norm(w_star - w_pred, 2) / np.linalg.norm(w_star, 2)
    error_p = np.linalg.norm(p_star - p_pred, 2) / np.linalg.norm(p_star, 2)

    print('Error u: %e' % error_u)
    print('Error v: %e' % error_v)
    print('Error v: %e' % error_w)
    print('Error p: %e' % error_p)

    scipy.io.savemat('../NS3D_beltrami_%s.mat' %(time.strftime('%d_%m_%Y')),
```

```
253                          {'U_pred':u_pred, 'V_pred':v_pred, 'W_pred':
     w_pred, 'P_pred':p_pred})
```

Listing 1: nain

Line 8-11 : Import tensorflow,numpy,time,scipy.io to load,calculate and save data.

Line : Initialize seed for pseudo random number generator.

Line 14 : Define class NSPINN to initialize variables.

Line 20-28 : Import necessary functions into the class for solver calculations.

Line 29 : Define function _init_ to initialize tensor flow variables with initial,boundary and dataset values.

Line 30-32 : Concatenate values to create a single array for data manipulation.

Line 34-35 : Assign boundary values.

Line 37-64 : Assign input dataset values to local class variables.

Line 66 : Initialize weight and biases according to the no of layers.

Line 68 : Initialize tensor variable for learning rate.

Line 70 : Initialize session variable.

Line 73-92 : Initialize tensor variables for class variables.

Line 94-100 : Calculate pred values from the data set using neuralnet.

Line 104 : Define loss function.

Line 115-124 : Define Optimizer wrapper and input loss parameter.

Line 126-127 : Initialize session for calculation.

Line 131 : Define main function.

Line 133 : No of training loops.

Line 135 : No of layers.

Line 137 : Define function data generate to create dataset for beltrami flow.

Line 151-156 : input Dataset.

Line 159-178 : Data manipulation to generate more points.

Line 180-187 : Assign boundary values to variables.

Line 190-195 : Data manipulation for initial values.

Line 197-204 : Assign initial values to variables.

Line 207-212 : Data manipulation for dataset values.

Line 214-217 : Assign dataset values to variables.

Line 219 : Pass initial,boundary,and dataset values to model to predict values.

Line 225-229 : Train the model for the given iterations and learning rate.

Line 231-234 : Predict values for x,y,z,t.

Line 236 : Generate dataset with the predicted values.

Line 239 : Generated values from neural network.

Line 242-245 : Calculate error value.

Line 247-250 : Print error values.

Line 252: Save data as an .mat file for output.

2 Neuralnet initialize

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Nov  8 14:54:44 2020
4 
5 @author: nastavirs
6 """
7 import tensorflow as tf
8 import numpy as np
9 np.random.seed(1234)
10 tf.set_random_seed(1234)
11 def initialize_NN(self, layers):
12         weights = []
13         biases = []
14         num_layers = len(layers)
15         for l in range(0, num_layers - 1):
16             W = self.xavier_init(size=[layers[l], layers[l + 1]])
17             b = tf.Variable(tf.zeros([1, layers[l + 1]], dtype=tf.
    float32), dtype=tf.float32)
18             weights.append(W)
19             biases.append(b)
20         return weights, biases
```

Listing 2: Neural net

```
Line 7-8: import numpy and tensorflow for calculations

Line 9 : initialize_NN function is used to initialize and return weights and biases of a NN
         with layers size and solver type as arguments

Line 10-11 : initialize empty list for weights and biases

Line 12 : assign the size of the layers variable to num_layers by using the len function

Line 13 : loop around from to layers-1 to calculate weights and biases for each layer.

Line 14 : xavier initialize the weights value by passing size as a list layer l,layer 1+l

Line 15 : assign a tensorflow variable with zero matrix of size l,layers 1+i to the variable b

Line 16-17 : append the values created during each loop to the list weights and biases

Line 18 : return weights and biases
```

3 Xavier Initialization

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Nov  8 14:54:56 2020
4 
5 @author: nastavirs
6 """
7 import tensorflow as tf
8 import numpy as np
9 np.random.seed(1234)
10 tf.set_random_seed(1234)
11 def xavier_init(self, size):
12         in_dim = size[0]
13         out_dim = size[1]
14         xavier_stddev = np.sqrt(2 / (in_dim + out_dim))
15         return tf.Variable(tf.truncated_normal([in_dim, out_dim],
    stddev=xavier_stddev), dtype=tf.float32)
```

Listing 3: Xavier init

Line 11 : initialize the xavier_init function definition with arguments class and size

Line 12 : in_dim is the number of input nodes into each output and is assigned 1st value from arg size

Line 13 : out_dim is the number of output nodes for each input and is assigned 2nd value from arg size

Line 14 : calculate the standard deviation.

Line 15 : return a tensorflow variable with from a truncated normal distribution

4 Neuralnet code

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Nov  8 14:55:07 2020
4 
5 @author: nastavirs
6 """
7 import tensorflow as tf
8 import numpy as np
9 np.random.seed(1234)
10 tf.set_random_seed(1234)
11 def neural_net(self, X, weights, biases):
12         num_layers = len(weights) + 1
13 
14         H = 2.0 * (X - self.lowb) / (self.upb - self.lowb) - 1.0
15         for l in range(0, num_layers - 2):
16             W = weights[l]
17             b = biases[l]
18             H = tf.tanh(tf.add(tf.matmul(H, W), b))
19         W = weights[-1]
20         b = biases[-1]
21         Y = tf.add(tf.matmul(H, W), b)
22         return Y
```

Listing 4: NN

Line 7-8: import numpy and tensorflow for calculations

Line 9 : define function neural net with arguments, class, data set X, weights and biases calculated earlier.

Line 10 : assign the length of weights + 1 to the variable num_layers

Line 12 : the right hand size of the equation represents the assumed hypothesis function/representation

Line 13 : for loop to loop around layers from 0 to n-2 layers

Line 14-15 : initialize and assign Weights and biases of layer l to W and b

Line 16 : we use forward propagation to calculate the H, where we first matrix multiply hypothesis function and weights and add then add the biases. finally pass this to the tanh activation fuction tanh(H*W+b)

Line 17-19: finally we do the same step for the final output layer

Line 20 : return Y

5 Supervised neuralnet

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Nov  8 14:55:15 2020
4
5  @author: nastavirs
6  """
7  import tensorflow as tf
8  import numpy as np
9  np.random.seed(1234)
10 tf.set_random_seed(1234)
11 def net_NS(self, x, y, z, t):
12
13         u_v_w_p = self.neural_net(tf.concat([x, y, z, t], 1), self.
    weights, self.biases)
14         u = u_v_w_p[:, 0:1]
15         v = u_v_w_p[:, 1:2]
16         w = u_v_w_p[:, 2:3]
17         p = u_v_w_p[:, 3:4]
18         return u, v, w, p
```

Listing 5: supervised NN

`Line 11 : Define function net_NS with class,x,y,z,t as variables.`

`Line 13 : Concat x,y,z,t and pass it to the neural net as an argument.`

`Line 14-17 : assign values from the results of the neural net to variables.`

`Line 18 : Return u,v,w,p`

6 Unsupervised neuralnet

```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov  8 14:55:30 2020

@author: nastavirs
"""
import tensorflow as tf
import numpy as np
np.random.seed(1234)
tf.set_random_seed(1234)
def net_f_NS(self, x, y, z, t):

        Re = 1

        u_v_w_p = self.neural_net(tf.concat([x, y, z, t], 1), self.
    weights, self.biases)
        u = u_v_w_p[:, 0:1]
        v = u_v_w_p[:, 1:2]
        w = u_v_w_p[:, 2:3]
        p = u_v_w_p[:, 3:4]

        u_t = tf.gradients(u, t)[0]
        u_x = tf.gradients(u, x)[0]
        u_y = tf.gradients(u, y)[0]
        u_z = tf.gradients(u, z)[0]
        u_xx = tf.gradients(u_x, x)[0]
        u_yy = tf.gradients(u_y, y)[0]
        u_zz = tf.gradients(u_z, z)[0]

        v_t = tf.gradients(v, t)[0]
        v_x = tf.gradients(v, x)[0]
        v_y = tf.gradients(v, y)[0]
        v_z = tf.gradients(v, z)[0]
        v_xx = tf.gradients(v_x, x)[0]
        v_yy = tf.gradients(v_y, y)[0]
        v_zz = tf.gradients(v_z, z)[0]

        w_t = tf.gradients(w, t)[0]
        w_x = tf.gradients(w, x)[0]
        w_y = tf.gradients(w, y)[0]
        w_z = tf.gradients(w, z)[0]
        w_xx = tf.gradients(w_x, x)[0]
        w_yy = tf.gradients(w_y, y)[0]
        w_zz = tf.gradients(w_z, z)[0]

        p_x = tf.gradients(p, x)[0]
        p_y = tf.gradients(p, y)[0]
        p_z = tf.gradients(p, z)[0]

        f_u = u_t + (u * u_x + v * u_y + w * u_z) + p_x - 1/Re * (
    u_xx + u_yy + u_zz)
        f_v = v_t + (u * v_x + v * v_y + w * v_z) + p_y - 1/Re * (
    v_xx + v_yy + v_zz)
        f_w = w_t + (u * w_x + v * w_y + w * w_z) + p_z - 1/Re * (
    w_xx + w_yy + w_zz)
```

```
52          f_e = u_x + v_y + w_z
53
54          return u, v, w, p, f_u, f_v, f_w, f_e
```

Listing 6: unsupervised NN

Line 7-8: import numpy and tensorflow for calculations

Line 9 : define net_f_NS function with arguments class x,y,z,t

Line 11 : Initialize Reynolds number.

Line 13 : define variable u_v_w_p as the return of the function neuralnet with tensorflow variable which is the result of concatenation of x,y and t and 1 implies along column axis, weights, biases which are passed as arguments for the class

Line 14-17 : slice u_v_w_p columns wise and assign it to u,v,w,p.

Line 19 : calculating the gradient of u wrt t and assign it to u_t

Line 20 : calculating the gradient of u wrt x and assign it to u_x

Line 21 : calculating the gradient of u wrt y and assign it to u_y

Line 22 : calculating the gradient of u wrt z and assign it to u_z

Line 23 : calculating the gradient of u_x wrt x and assign it to u_xx

Line 24 : calculating the gradient of u_y wrt y and assign it to u_yy

Line 25 : calculating the gradient of u_y wrt y and assign it to u_yy

Line 27 : calculating the gradient of v wrt t and assign it to v_t

Line 28 : calculating the gradient of v wrt x and assign it to v_x

Line 29 : calculating the gradient of v wrt y and assign it to v_y

Line 30 : calculating the gradient of v wrt z and assign it to v_z

Line 31 : calculating the gradient of v_x wrt x and assign it to v_xx

Line 32 : calculating the gradient of v_y wrt y and assign it to v_yy

Line 33 : calculating the gradient of v_z wrt z and assign it to v_zz

Line 35 : calculating the gradient of w wrt t and assign it to w_t

Line 36 : calculating the gradient of w wrt x and assign it to w_x

Line 37 : calculating the gradient of w wrt y and assign it to w_y

Line 38 : calculating the gradient of w wrt z and assign it to w_z

Line 39 : calculating the gradient of w_x wrt x and assign it to w_xx

Line 40 : calculating the gradient of w_y wrt y and assign it to w_yy

Line 41 : calculating the gradient of w_z wrt z and assign it to w_zz

Line 43 : calculating the gradient of p wrt x and assign it to p_x

Line 44 : calculating the gradient of p wrt y and assign it to p_y

Line 45 : calculating the gradient of p wrt z and assign it to p_z

Line 47 : calculate the nonlinear partial differentiation equation (N-S) f as u_t+(u*u_x+v*u_y+w*u_z)+p_x-1/Re*(u_xx+u_yy+u_zz) and assign it to variable f_u

```
Line 48 : calculate the nonlinear partial differentiation equation (N-S) f as
           v_t+(u*v_x+v*v_y+w*v_z)+p_y-1/Re*(v_xx+v_yy+v_zz) and assign it to variable f_v

Line 49 : calculate the nonlinear partial differentiation equation (N-S) f as
           w_t+(u*w_x+v*w_y+w*w_z)+p_w-1/Re*(w_xx+w_yy+w_zz) and assign it to variable f_w

Line 50 : calculate the nonlinear partial differentiation equation (N-S) f as
          (u_x+v_y+w_z) and assign it ti variable f_e

Line 52 : returns u,v,w,p,f_u,f_v,f_w.f_e as the result of function net_NS
```

7 Callback

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Nov  8 14:55:53 2020
4  
5  @author: nastavirs
6  """
7  import tensorflow as tf
8  import numpy as np
9  np.random.seed(1234)
10 tf.set_random_seed(1234)
11 def callback(self, loss):
12         print('Loss: %.3e' % loss)
```

Listing 7: callback

Line 7-8: import numpy for calculations

Line 9 : define function callback with parameter class,loss

Line 10 : print loss.

8 Adam optimizer

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Nov  8 14:56:19 2020
4
5 @author: nastavirs
6 """
7 import tensorflow as tf
8 import numpy as np
9 import time
10 np.random.seed(1234)
11 tf.set_random_seed(1234)
12 def Adam_train(self, nIter=5000, learning_rate=1e-3):
13
14         tf_dict = {self.x_ini_tf: self.xi, self.y_ini_tf: self.yi,
    self.z_ini_tf: self.zi, self.t_ini_tf: self.ti,
15                    self.u_ini_tf: self.ui, self.v_ini_tf: self.vi,
    self.w_ini_tf: self.wi,
16                    self.x_boundary_tf: self.xb, self.y_boundary_tf:
     self.yb, self.z_boundary_tf: self.zb,
17                    self.t_boundary_tf: self.tb, self.u_boundary_tf:
     self.ub, self.v_boundary_tf: self.vb,
18                    self.w_boundary_tf: self.wb, self.x_tf: self.x,
    self.y_tf: self.y, self.z_tf: self.z,
19                    self.t_tf: self.t, self.learning_rate:
    learning_rate}
20
21         start_time = time.time()
22         for it in range(nIter):
23             self.sess.run(self.train_op_Adam, tf_dict)
24
25             # Print
26             if it % 10 == 0:
27                 elapsed = time.time() - start_time
28                 loss_value = self.sess.run(self.loss, tf_dict)
29                 print('It: %d, Loss: %.3e, Time: %.2f' %
30                       (it, loss_value, elapsed))
31                 start_time = time.time()
32
33         self.optimizer.minimize(self.sess,
34                                 feed_dict=tf_dict,
35                                 fetches=[self.loss],
36                                 loss_callback=self.callback)
```

Listing 8: adam

Line 7,8,9: Import tensorflow, numpy and time for run time.

Line 10-11: Set seed for pseudo random number generator.

Line 12: Define function adam train with input variables of class, no of Iterations and learning rate

Line 14: Initiate tf.dict with all the required variables

Line 21: Initiate variable start.time to start time counter.

Line 22: Start for loop to loop for nIter.

Line 23: Run session to initialize tensor variables

`Line 26-31: Print Loss and time for every ten iterations`

`Line 33: Use Wrapper to minimize the loss function in tf.dict`

9 BGFS Optimizer

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Nov  8 14:56:32 2020

@author: nastavirs
"""
import tensorflow as tf
import numpy as np
import time
np.random.seed(1234)
tf.set_random_seed(1234)
def BFGS_train(self):

        tf_dict = {self.x_ini_tf: self.xi, self.y_ini_tf: self.yi, self.z_ini_tf: self.zi, self.t_ini_tf: self.ti,
                   self.u_ini_tf: self.ui, self.v_ini_tf: self.vi, self.w_ini_tf: self.wi,
                   self.x_boundary_tf: self.xb, self.y_boundary_tf: self.yb, self.z_boundary_tf: self.zb,
                   self.t_boundary_tf: self.tb, self.u_boundary_tf: self.ub, self.v_boundary_tf: self.vb,
                   self.w_boundary_tf: self.wb, self.x_tf: self.x, self.y_tf: self.y, self.z_tf: self.z,
                   self.t_tf: self.t}

        self.optimizer.minimize(self.sess,
                                feed_dict=tf_dict,
                                fetches=[self.loss],
                                loss_callback=self.callback)

    # mini-batch to be implemented
    # def train(self, epoch=10, nIter=150, learning_rate=1e-3):
    #
    #     for ep in range(epoch):
    #
    #         batch_size1 = len(self.x0) // nIter
    #         batch_size2 = len(self.xb) // nIter
    #         batch_size3 = len(self.x) // nIter
    #
    #         arr1 = np.arange(batch_size1 * nIter)
    #         arr2 = np.arange(batch_size2 * nIter)
    #         arr3 = np.arange(batch_size3 * nIter)
    #
    #         permu1 = np.random.permutation(arr1).reshape((nIter, batch_size1))
    #         permu2 = np.random.permutation(arr2).reshape((nIter, batch_size2))
    #         permu3 = np.random.permutation(arr3).reshape((nIter, batch_size3))
    #
    #         start_time = time.time()
    #         for it in range(nIter):
    #             tf_dict = {self.x_ini_tf: self.x0[permu1[it, :], :],
    #                        self.y_ini_tf: self.y0[permu1[it, :],
```

```
   :],
47 #                         self.z_ini_tf: self.z0[permu1[it, :], :],
48 #                         self.t_ini_tf: self.t0[permu1[it, :], :],
49 #                         self.u_ini_tf: self.u0[permu1[it, :], :],
50 #                         self.v_ini_tf: self.v0[permu1[it, :], :],
51 #                         self.w_ini_tf: self.w0[permu1[it, :], :],
52 #                         self.x_boundary_tf: self.xb[permu2[it, :], :],
53 #                         self.y_boundary_tf: self.yb[permu2[it, :], :],
54 #                         self.z_boundary_tf: self.zb[permu2[it, :], :],
55 #                         self.t_boundary_tf: self.tb[permu2[it, :], :],
56 #                         self.u_boundary_tf: self.ub[permu2[it, :], :],
57 #                         self.v_boundary_tf: self.vb[permu2[it, :], :],
58 #                         self.w_boundary_tf: self.wb[permu2[it, :], :],
59 #                         self.x_tf: self.x[permu3[it, :], :],
60 #                         self.y_tf: self.y[permu3[it, :], :],
61 #                         self.z_tf: self.z[permu3[it, :], :],
62 #                         self.t_tf: self.t[permu3[it, :], :],
63 #                         self.learning_rate: learning_rate}
64 #
65 #              self.sess.run(self.train_op_Adam, tf_dict)
66 #
67 #              # Print
68 #              if it % 10 == 0:
69 #                  elapsed = time.time() - start_time
70 #                  loss_value = self.sess.run(self.loss, tf_dict)
71 #                  print('epoch: %d, It: %d, Loss: %.3e, Time: %.2f' %
72 #                        (ep, it, loss_value, elapsed))
73 #                  start_time = time.time()
74 #
75 #      self.optimizer.minimize(self.sess,
76 #                              feed_dict=tf_dict,
77 #                              fetches=[self.loss],
78 #                              loss_callback=self.callback)
```

Listing 9: BGFS

`Line 7,8,9: Import tensorflow, numpy and time for run time.`

`Line 10-11: Set seed for pseudo random number generator.`

`Line 12: Define function BGFS train with input variables of class, no of Iterations and learning rate`

`Line 14: Initiate tf.dict with all the required variables`

`Line 21: Use Wrapper to minimize the loss function in tf.dict`

10 Predict

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Nov  8 14:56:40 2020
4 
5 @author: nastavirs
6 """
7 import numpy as np
8 import tensorflow as tf
9 def predictNS(self, x_star, y_star, z_star, t_star):
10 
11         tf_dict = {self.x_tf: x_star, self.y_tf: y_star, self.z_tf:
      z_star, self.t_tf: t_star}
12 
13         u_star = self.sess.run(self.u_pred, tf_dict)
14         v_star = self.sess.run(self.v_pred, tf_dict)
15         w_star = self.sess.run(self.w_pred, tf_dict)
16         p_star = self.sess.run(self.p_pred, tf_dict)
17 
18         return u_star, v_star, w_star, p_star
```

Listing 10: Solver

```
Line 7-8 : import numpy and tensorflow for calculations

Line 9 : define function predict with arguments class and x_star,y_star,z_star,t_star predict values for training

Line 11 : define tensorflow dict with x_tf,y_tf,z_tf,t_tf with corresponding input value.

Line 13-16 : sess.run evaluates and returns values after a session for input parameters u.pred,v_pred,w_pred,p_pred
            and train_dict is also an input because pred values depends on it, assign it to u_star,v_star,w_star,p_star.

Line 16 : return u_star, v_star,w_star, and p_star for function predict
```

Matlab Code

```matlab
clear
close all

set(0,'defaulttextinterpreter','latex')
%load Cylinder3D.mat
load NS3D_beltrami.mat
fig = figure();
set(fig,'units','normalized','outerposition',[0 0 1 1])

for num = 100:100 %size(t_star,1)
    disp(num)

    clf

    subplot(2,2,1)
    plot_isosurface_griddata(x_star, y_star, z_star, U_star(:,num), '$x$','$y$','$z$','Regressed $u(t,x,y,z)$')
    drawnow()

    subplot(2,2,2)
    plot_isosurface_griddata(x_star, y_star, z_star, V_star(:,num), '$x$','$y$','$z$','Regressed $v(t,x,y,z)$')
    drawnow()

    subplot(2,2,3)
    plot_isosurface_griddata(x_star, y_star, z_star, W_star(:,num), '$x$','$y$','$z$','Regressed $w(t,x,y,z)$')
    drawnow()

    subplot(2,2,4)
    plot_isosurface_griddata(x_star, y_star, z_star, P_star(:,num), '$x$','$y$','$z$','Regressed $p(t,x,y,z)$')
    drawnow()

    %%%

end

% addpath ~/export_fig
% export_fig ./Cylinder_3D_results.png -r300
```

Listing 11: plotting

```matlab

function plot_isosurface_griddata(x_star, y_star, z_star, u_star, xlab, ylab, zlab, tit)

x_l = min(x_star);
x_r = max(x_star);

y_l = min(y_star);
y_r = max(y_star);

z_l = min(z_star);
z_r = max(z_star);
```

```
nn = 100;
x = linspace(x_l, x_r, nn)';
y = linspace(y_l, y_r, nn)';
z = linspace(z_l, z_r, nn)';
[Xplot, Yplot, Zplot] = meshgrid(x,y,z);


Uplot = griddata(x_star,y_star,z_star, u_star, Xplot,Yplot,Zplot);

idx = linspace(min(u_star),max(u_star),5);

isosurface(Xplot, Yplot, Zplot, Uplot, idx(2));
hold all
isosurface(Xplot, Yplot, Zplot, Uplot, idx(3));
hold all
isosurface(Xplot, Yplot, Zplot, Uplot, idx(4));
zlim([idx(1), idx(5)])
view(3)
xlabel(xlab);
ylabel(ylab);
zlabel(zlab);
title(tit);

axis tight
axis equal
colormap jet
colorbar
alpha(0.7)
set(gca,'FontSize',20);
set(gcf, 'Color', 'w');
```

Listing 12: calculate Isosurface