NN_WM

March 19, 2021

[678]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import tensorflow as tf
from scipy.io import loadmat
import scipy.optimize as sopt

matplotlib.rcdefaults()
font = {'size'   : 18}
matplotlib.rc('font', **font)
```

[252]:
```
# ====================================================
# Borrowing the code to use scipy optimizers
# Taken from HW5
# ====================================================

# use float64 by default
tf.keras.backend.set_floatx("float64")

# Construct a function that can be minimized by scipy optimize
# starting from a tensorflow model
def function_factory(model, x_train, y_train, validation_data=None,
                     iprint=-1):
    """A factory to create a function required by scipy.optimize.
    Args:
        model [in]: an instance of `tf.keras.Model` or its subclasses.
        loss [in]: a function with signature loss_value = loss(pred_y, true_y).
        x_train [in]: input for training data.
        y_train [in]: output for training data.
        validation_data [in]: tuple (x_val,y_val) with validation data.
        iprint [in]: sets the frequency with which the loss info is printed out
    Returns:
        A function that has a signature of:
            loss_value, gradients = f(model_parameters)
    """

    # obtain the shapes of all trainable parameters in the model
```

```
    shapes = tf.shape_n(model.trainable_variables)
    n_tensors = len(shapes)

    # we'll use tf.dynamic_stitch and tf.dynamic_partition later, so we need to
    # prepare required information first
    count = 0
    idx = []  # stitch indices
    part = []  # partition indices

    for i, tensor in enumerate(model.trainable_variables):
        n = np.product(tensor.shape)
        idx.append(tf.reshape(
            tf.range(count, count+n, dtype=tf.int32), tensor.shape))
        part.extend([i]*n)
        count += n

    part = tf.constant(part)

    loss_fun = tf.keras.losses.MeanSquaredError()

    @tf.function
    def assign_new_model_parameters(params_1d):
        """A function updating the model's parameters with a 1D tf.Tensor.
        Args:
            params_1d [in]: a 1D tf.Tensor representing the model's trainable␣
↪parameters.
        """
        params = tf.dynamic_partition(params_1d, part, n_tensors)
        for i, (shape, param) in enumerate(zip(shapes, params)):
            model.trainable_variables[i].assign(tf.reshape(param, shape))

    # function to calculate loss value and gradient
    @tf.function
    def tf_tape_grad(params_1d):

        # update the parameters in the model
        assign_new_model_parameters(params_1d)

        if not (validation_data is None):
          # compute validation loss
          loss_val = model.loss(validation_data[0], validation_data[1])
          # store validation value so we can retrieve later
          tf.py_function(value_and_grad.hist_loss_val.append,
                         inp=[loss_val], Tout=[])

        # use GradientTape so that we can calculate the gradient of loss w.r.t.␣
↪parameters
```

```
        with tf.GradientTape() as g:
            loss = model.loss(x_train,y_train)

        # calculate gradients and convert to 1D tf.Tensor
        grads = g.gradient(loss, model.trainable_variables)
        grads = tf.dynamic_stitch(idx, grads)

        # increment iteration counter
        value_and_grad.iter.assign_add(1)

        # print out iteration & loss
        if (iprint>=1 and value_and_grad.iter%iprint == 0):
            if validation_data is None:
                tf.print("Loss function eval:", value_and_grad.iter, "loss:",␣
↪loss)
            else:
                tf.print("Loss function eval:", value_and_grad.iter, "loss:",␣
↪loss,\
                         "validation loss:", loss_val)
        # store loss value so we can retrieve later
        tf.py_function(value_and_grad.hist_loss.append,
                       inp=[loss], Tout=[])

        return loss, grads

    # create function that will be returned by this factory
    def value_and_grad(params_1d):
        """A function that can be used by optimizer.
        This function is created by function_factory.
        Args:
           params_1d [in]: a 1D tf.Tensor.
        Returns:
            A scalar loss and the gradients w.r.t. the `params_1d`.
        """
        return [vv.numpy().astype(np.float64) for vv in tf_tape_grad(tf.
↪constant(params_1d, dtype=tf.float64))]

    # store this information as members so we can use it outside the scope
    value_and_grad.iter = tf.Variable(0)
    value_and_grad.idx = idx
    value_and_grad.part = part
    value_and_grad.shapes = shapes
    value_and_grad.assign_new_model_parameters = assign_new_model_parameters
    value_and_grad.hist_loss = []
    value_and_grad.hist_loss_val = []

    return value_and_grad
```

```
# Minimize the loss function using a scipy optimizer
def model_fit(model, x_t, y_t, validation_data=None, epochs=1000, iprint=-1,
              figname=None):
    """
    Fit a DNN model using scipy optimizers


    Parameters:
    model: tensorflow DNN model
    x_t: input training data
    y_t: output training data
    validation_data: tuple (x_val,y_val) with validation data
    epochs: maximum number of iterations in optimizer
    iprint: frequency for printing the loss function information.
            Do not print anything if negative. Otherwise, print
            a line every iprint iteration.
    figname [str]: file name to save the figure of the training loss
    """

    value_and_grad = function_factory(model, x_t, y_t, validation_data, iprint)

    # convert initial model parameters to a 1D tf.Tensor
    init_params = tf.dynamic_stitch(value_and_grad.idx, model.
 ↪trainable_variables)

    if (iprint>=1):
        print()

    # train the model
    method = 'L-BFGS-B'
    results = sopt.minimize(fun=value_and_grad, x0=init_params,
                            jac=True, method=method,
                            options={'maxiter': epochs})

    print("\nConvergence information:")
    print('loss:', results.fun)
    # Computing the validation loss
    if not (validation_data is None):
        val_loss = model.loss(validation_data[0], validation_data[1])
        tf.print('validation loss:', val_loss)
        # tf.make_ndarray(tf.make_tensor_proto(val_loss))
    print('number function evaluations:', results.nfev)
    print('number iterations:', results.nit)
    print('success flag:', results.success)
    print('convergence message:', results.message)
```

```
    value_and_grad.assign_new_model_parameters(results.x)

    # Plot history of loss
    plt.figure()
    plt.plot(value_and_grad.hist_loss, label='loss')
    if not (validation_data is None):
        plt.plot(value_and_grad.hist_loss_val, label='validation')
    plt.legend()
    plt.xlabel('epoch')
    plt.yscale('log')
    if validation_data is None:
        plt.title('Training loss')
    else:
        plt.title('Training and validation losses')

    if not (figname is None):
        plt.savefig(figname,dpi=300)

    return results
```

[489]:
```
class NN_WM(tf.keras.models.Model):
    def __init__(self, n_output, n_layers, a_layers, w_loss):
        super(NN_WM, self).__init__()

        self.n_layers = n_layers
        self.hidden_layers = []
        for nl in range(n_layers):
            self.hidden_layers.append(
                tf.keras.layers.Dense(
                    a_layers[nl],
                    activation=tf.keras.activations.tanh,
                    kernel_regularizer=tf.keras.regularizers.L2()))
        self.output_layer = tf.keras.layers.Dense(n_output,
                                activation=tf.keras.activations.linear)

        self.loss_fun = tf.keras.losses.MeanSquaredError()

        self.loss_w1 = w_loss[0]
        self.loss_w2 = w_loss[1]

    def call(self, x_input):
        x1 = x_input
        for nl in range(self.n_layers):
            x2 = self.hidden_layers[nl](x1)
            x1 = x2
        x2 = self.output_layer(x1)
```

```
        return x2

    def loss(self, X, Y):
        loss_mse = 0
        loss_cross = 0

        # accuracy to training data
        u = self(X)
        loss_mse += self.loss_fun(Y,u)

        # penalty when mean varies at different filter levels
        #u = self(X)
        #umean = tf.math.reduce_mean(u,1)
        #ustd = tf.math.reduce_std(u,0)
        #loss_cross = tf.math.reduce_sum(u)

        loss_tot = self.loss_w1*loss_mse + self.loss_w2*loss_cross
        return loss_tot
```

[551]:
```
def load_feature_data(case):
    input_fname = 'data/input'+case+'.mat'
    output_fname = 'data/output'+case+'.mat'

    input_data = loadmat(input_fname)
    output_data = loadmat(output_fname)
    X = input_data['input_features']
    Y = output_data['output_features']

    return X, Y
```

0.0.1 Neural Network 1

[552]:
```
model = NN_WM(1,3,[4,3,3], [1000,100])
model.build((None,2))

model.summary()
```

```
Model: "nn_wm_58"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_224 (Dense)            multiple                  12
_________________________________________________________________
dense_225 (Dense)            multiple                  15
_________________________________________________________________
dense_226 (Dense)            multiple                  12
_________________________________________________________________
```

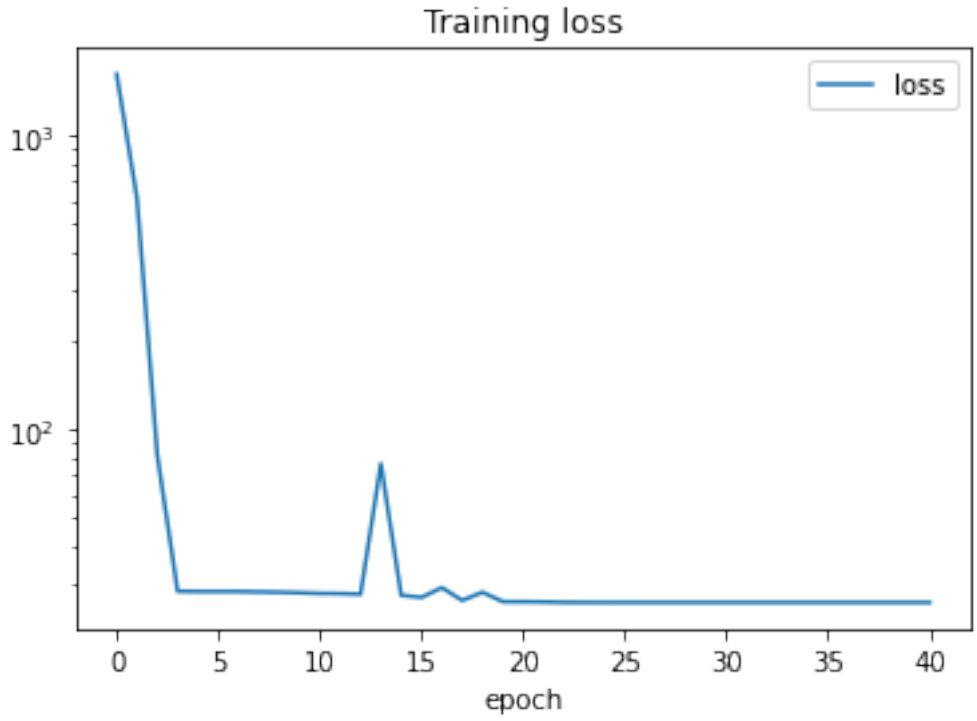
```
dense_227 (Dense)            multiple                  4
=================================================================
Total params: 43
Trainable params: 43
Non-trainable params: 0
_________________________________________________________________
```

[553]:
```
X_1, Y_1 = load_feature_data('NN1_h50box16')
X_2, Y_2 = load_feature_data('NN1_h50box32')
X_3, Y_3 = load_feature_data('NN1_h30box16')
X_4, Y_4 = load_feature_data('NN1_h30box32')

X_train = np.ndarray((4,*np.shape(X_1)))
Y_train = np.ndarray((4,*np.shape(Y_1)))

X_train[0,:,:] = X_1
X_train[1,:,:] = X_2
X_train[2,:,:] = X_3
X_train[3,:,:] = X_4
Y_train[0,:,:] = Y_1
Y_train[1,:,:] = Y_2
Y_train[2,:,:] = Y_3
Y_train[3,:,:] = Y_4


model_fit(model, X_train, Y_train)
```

```
Convergence information:
loss: 25.709999725222588
number function evaluations: 41
number iterations: 22
success flag: True
convergence message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
WARNING:tensorflow:8 out of the last 8 calls to <function
function_factory.<locals>.assign_new_model_parameters at 0x000002A35E7049D8>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

[553]:
```
      fun: 25.709999725222588
 hess_inv: <43x43 LbfgsInvHessProduct with dtype=float64>
```

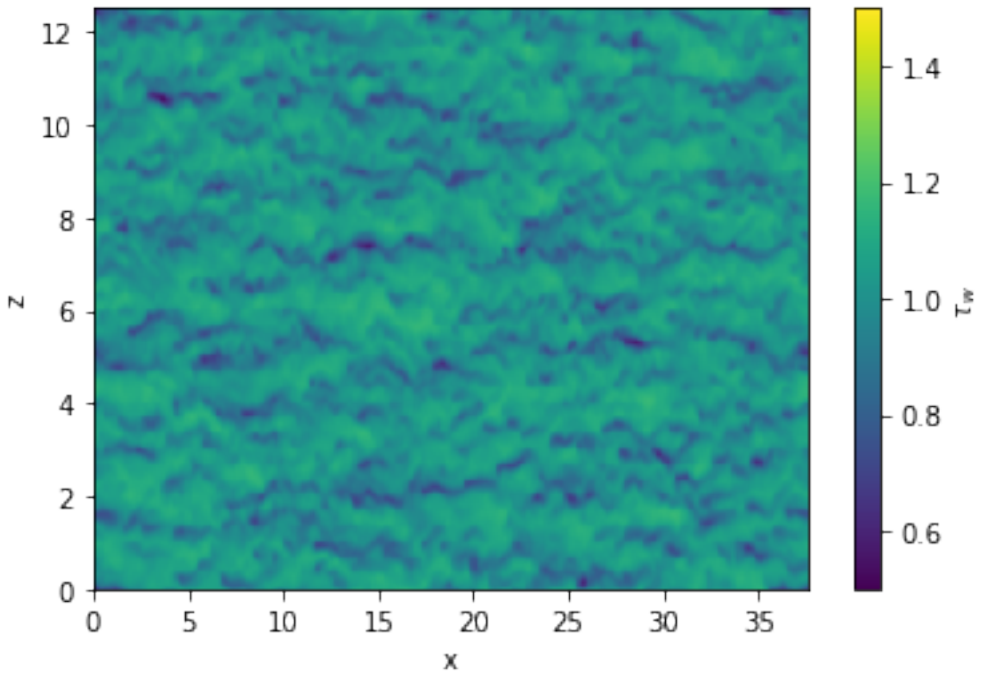
```
      jac: array([-0.00356027, -0.00220671, -0.00533938, -0.00261968,
-0.00702716,
       -0.00268283, -0.00333666, -0.00256897, -0.00655692, -0.0048081 ,
       -0.0079851 , -0.00431233, -0.00132148, -0.00034628, -0.00187852,
        0.00067009, -0.00450099,  0.00033596,  0.0011763 , -0.00645121,
        0.00058692, -0.00205102,  0.00507865, -0.00266871,  0.00566164,
       -0.01369502,  0.00698009, -0.00030813,  0.00188181, -0.00059062,
       -0.0042148 , -0.00258101,  0.00382964, -0.00391214, -0.00820146,
        0.00788438,  0.01482263,  0.02596943, -0.02869111,  0.00926047,
        0.01697882, -0.00952856,  0.0372105 ])
  message: 'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'
     nfev: 41
      nit: 22
     njev: 41
   status: 0
  success: True
        x: array([-0.24627609, -0.1672224 ,  0.33058687, -0.50717338,
0.73253712,
        0.72071729,  1.03580385,  0.09754034, -0.36830443, -0.03967958,
       -0.28189415, -0.17188031, -0.15297516,  0.82625272,  0.78679991,
        0.57680864,  0.45132172, -0.52141924,  0.65143714,  1.04952003,
       -0.76882981,  0.6292229 ,  0.30605215, -0.55668141,  0.03604405,
       -0.29309027, -0.13659081,  0.52880681,  0.37120794,  0.40618865,
       -0.30988693, -0.62069661,  0.35654525, -0.14161222, -0.1670831 ,
       -0.50185454,  0.14086997,  0.31492114, -0.32556178,  0.42324451,
        0.92532738, -1.09152278,  0.25189803])
```



[611]:
```
X_1, Y_1 = load_feature_data('NN1_h50box8')
X_train = np.ndarray((1,*np.shape(X_1)))
Y_train = np.ndarray((1,*np.shape(Y_1)))
X_train[0] = X_1
Y_train[0] = Y_1

print(model.loss(X_train,Y_train))

Y_pred = model(X_train)

N0 = np.size(Y_pred,1)
Y_pred = np.reshape(Y_pred[0,0:int(N0/2),0],[int(512/2), int(768/2)])
x = np.linspace(0,12*np.pi,int(768/2)+1)[0:int(768/2)]
z = np.linspace(0,4*np.pi,int(512/2)+1)[0:int(512/2)]
plt.pcolor(x,z,Y_pred)
cbar = plt.colorbar()
cbar.set_label(r'$\tau_w$')
plt.clim(np.min(Y_pred),np.max(Y_pred))
plt.ylabel('z')
plt.xlabel('x')
plt.clim(0.5,1.5)
#plt.gca().set_aspect('equal', adjustable='box')
plt.show()

plt.savefig('NN1_tau_pred_h50box8.eps')
```

```
tf.Tensor(82.76614546775818, shape=(), dtype=float64)
```

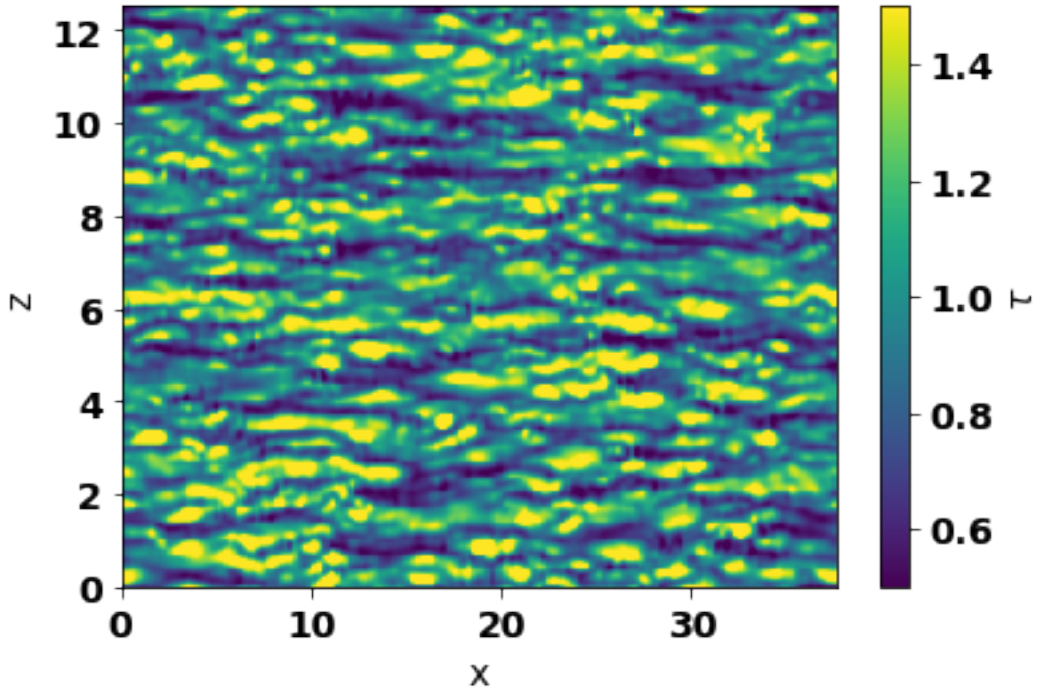
```
C:\Users\Michael\Anaconda3\envs\me343\lib\site-
packages\ipykernel_launcher.py:15: MatplotlibDeprecationWarning: shading='flat'
when X and Y have the same dimensions as C is deprecated since 3.3.  Either
specify the corners of the quadrilaterals with X and Y, or pass shading='auto',
'nearest' or 'gouraud', or set rcParams['pcolor.shading'].  This will become an
error two minor releases later.
  from ipykernel import kernelapp as app
```




```
<Figure size 432x288 with 0 Axes>
```

[669]:
```
N0 = np.size(Y_train,1)
Y_true = np.reshape(Y_train[0,0:int(N0/2),0],[int(512/2), int(768/2)])
x = np.linspace(0,12*np.pi,int(768/2)+1)[0:int(768/2)]
z = np.linspace(0,4*np.pi,int(512/2)+1)[0:int(512/2)]
plt.pcolor(x,z,Y_true)
cbar = plt.colorbar()
cbar.set_label(r'$\tau$')
plt.clim(0.5,1.5)
plt.ylabel('z')
plt.xlabel('x')
#plt.gca().set_aspect('equal', adjustable='box')
plt.show()

plt.savefig('NN1_tau_true_h50box8.eps')
```

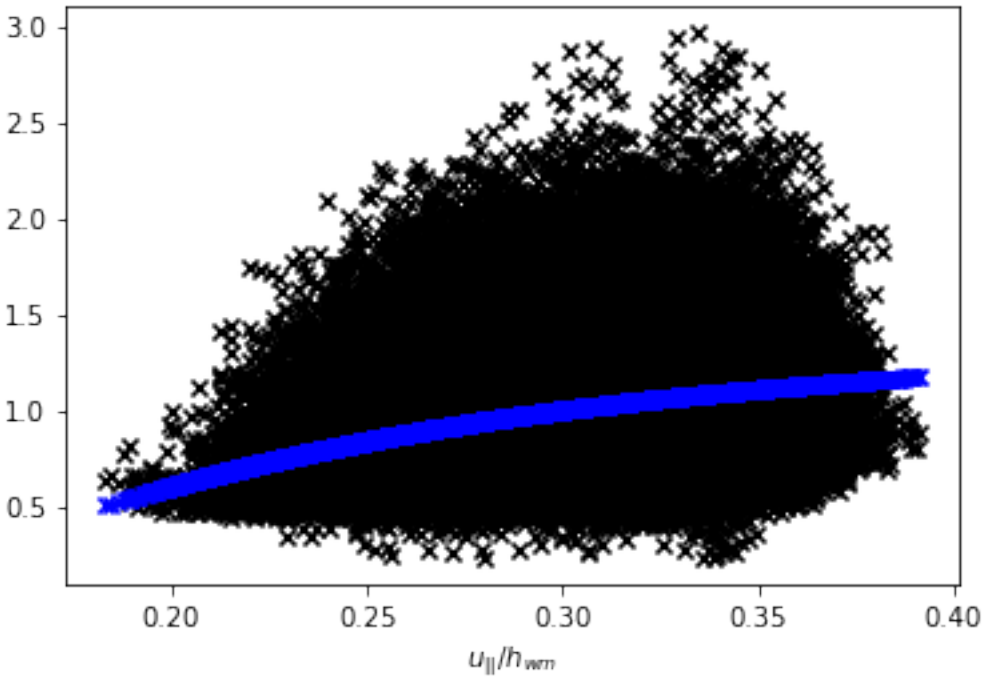
```
C:\Users\Michael\Anaconda3\envs\me343\lib\site-packages\ipykernel_launcher.py:5:
MatplotlibDeprecationWarning: shading='flat' when X and Y have the same
dimensions as C is deprecated since 3.3.  Either specify the corners of the
quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or
set rcParams['pcolor.shading'].  This will become an error two minor releases
later.
  """
```

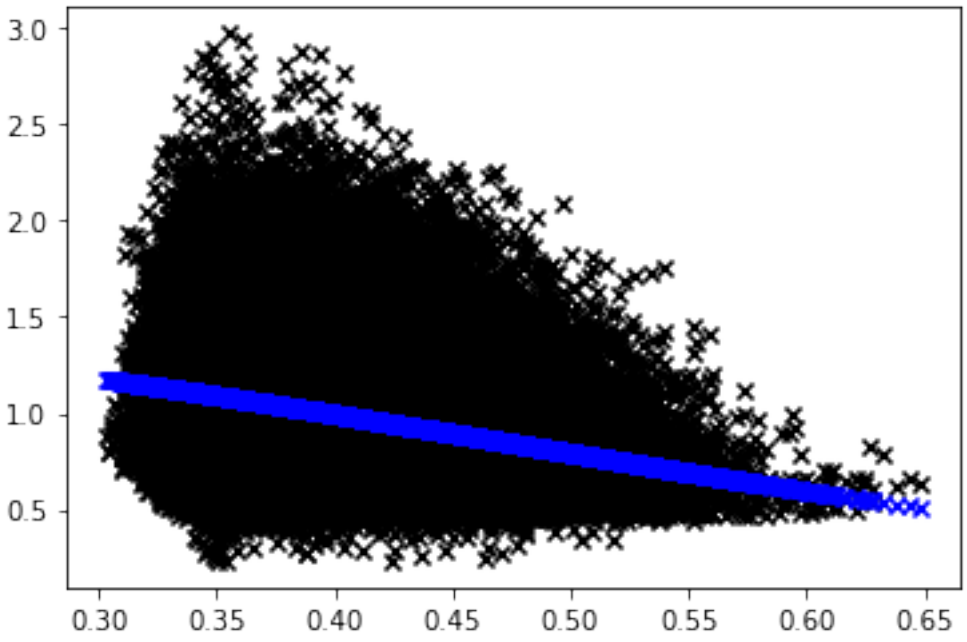



```
<Figure size 432x288 with 0 Axes>
```

[563]:
```
Y_pred = model(X_train)

print(Y_pred[0,:,0])

plt.scatter(X_train[0,:,0], Y_train[0,:,0], marker='x', color='k')
plt.scatter(X_train[0,:,0], Y_pred[0,:,0], marker='x', color='b')
plt.xlabel(r'$u_{||}/h_{wm}$')
plt.show()

plt.scatter(X_train[0,:,1], Y_train[0,:,0], marker='x', color='k')
plt.scatter(X_train[0,:,1], Y_pred[0,:,0], marker='x', color='b')
plt.show()
```

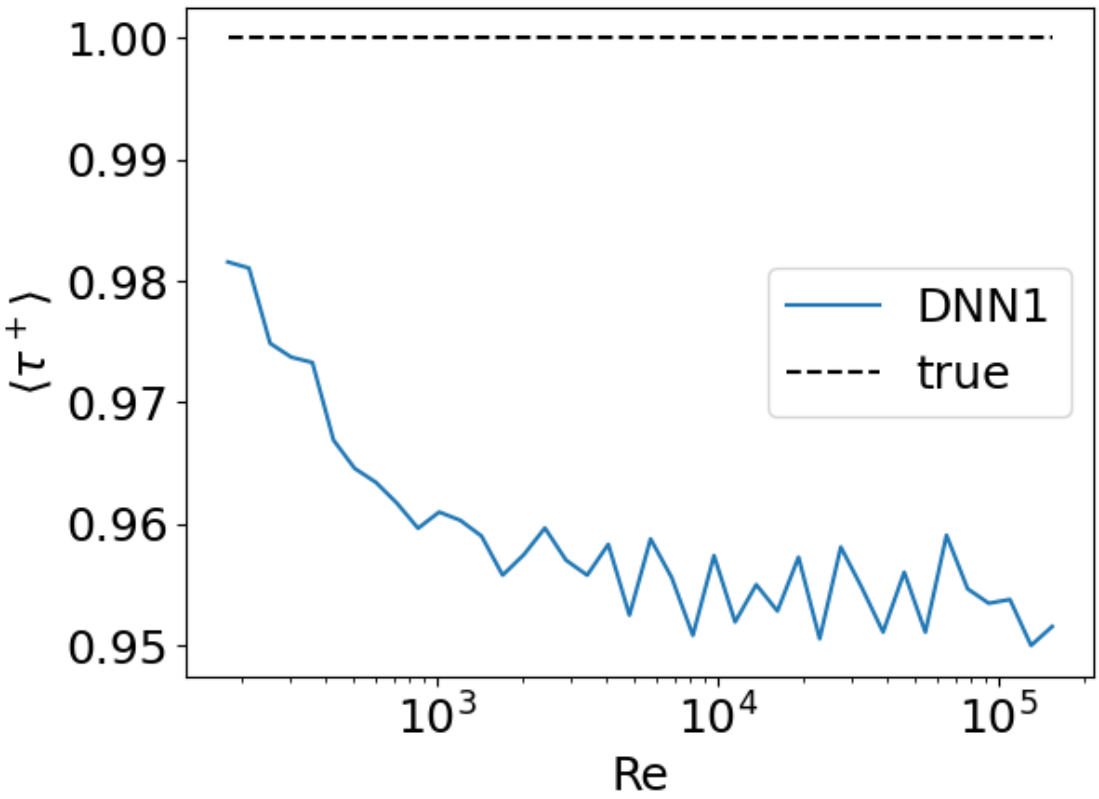
```
tf.Tensor([0.66907978 0.67486136 0.71029359 … 1.04988526 1.04707904 1.0492511
], shape=(196608,), dtype=float64)
```

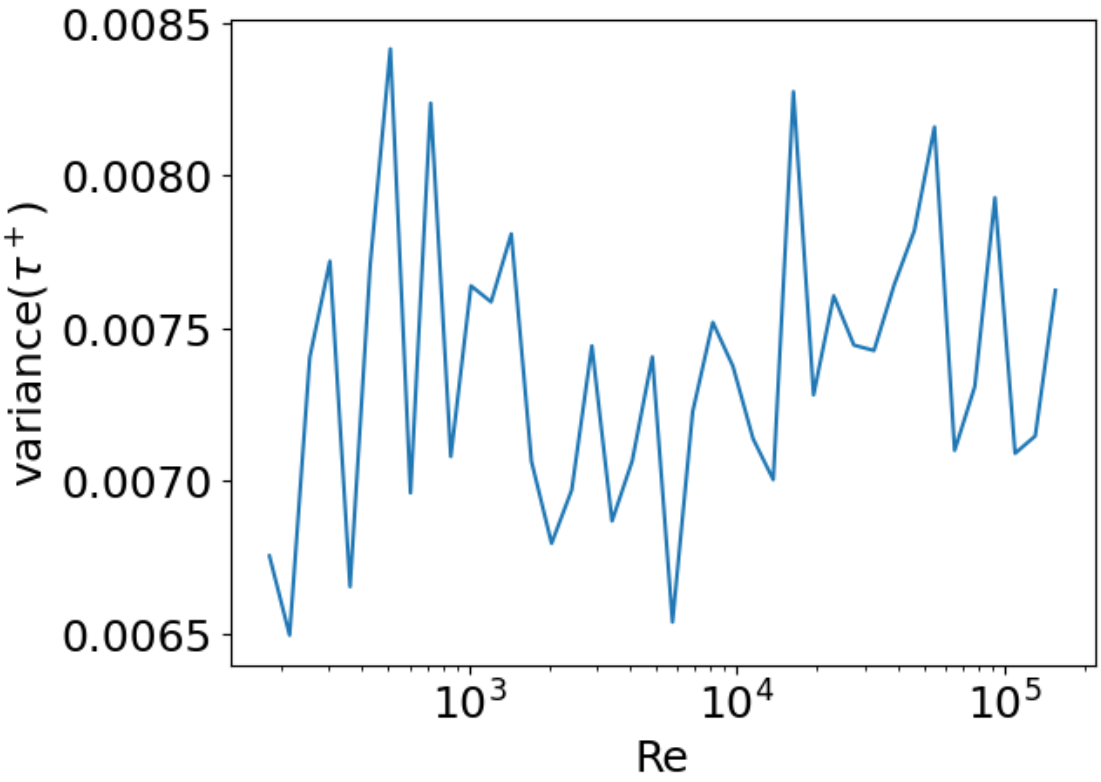




[686]:
```
# Generate pseudo-velocity data at Re=4200
Re_tau = 180*2**np.arange(0,10,0.25)
tau = np.zeros(np.size(Re_tau))
```

```
tau_var = np.zeros(np.size(Re_tau))
for jj,Re in enumerate(Re_tau):
    nu = 1/Re
    utau = 1
    kappa = 0.41
    B = 5.0
    y0 = nu/utau*np.exp(-kappa*B)

    hwm = 0.10 # 10% of boundary layer thickness
    var = 0.1
    upar = 1/kappa*np.log(hwm/y0)*(1+var*np.random.randn(1000))

    xf1 = (upar/utau)/(hwm*utau/nu)
    xf2 = np.log(hwm/y0)/(upar/utau)

    X_pseudo = np.zeros((2,np.size(xf1)))
    X_pseudo[0,:] = xf1
    X_pseudo[1,:] = xf2

    X_in = np.ndarray((1,*np.shape(X_pseudo.T)))
    X_in[0] = X_pseudo.T

    Y_pseudo = model(X_in)
    tau[jj] = np.mean(Y_pseudo[0,:,0])
    tau_var[jj] = np.var(Y_pseudo[0,:,0])

plt.plot(Re_tau, tau)
plt.plot(Re_tau, 0*Re_tau+1,'--k')
plt.xscale('log')
plt.xlabel('Re')
plt.ylabel(r'$\langle\tau^+\rangle$')
plt.legend(['DNN1','true'])
plt.tight_layout()
plt.draw()

plt.savefig('NN1_tau_v_Re.png')

plt.show()

plt.plot(Re_tau, tau_var)
plt.xscale('log')
plt.xlabel('Re')
plt.ylabel(r'variance($\tau^+$)')
plt.show()
```


1.00
0.99
0.98
0.97
0.96
0.95
$\langle \tau^+ \rangle$
10^3
10^4
10^5
Re
DNN1
true



[640]:
```
X_1, Y_1 = load_feature_data('NN1_h50box8')
X_train = np.ndarray((1,*np.shape(X_1)))
Y_train = np.ndarray((1,*np.shape(Y_1)))
X_train[0] = X_1
Y_train[0] = Y_1

Y_pred = model(X_train)

print(np.mean(abs(Y_pred-Y_train)/Y_train)*100)
```

```
23.73071530495981
```

0.0.2 Neural Network 2

[572]:
```
model2 = NN_WM(1,4,[6,4,3,3], [1000,10])
model2.build((None,3))

model2.summary()
```

```
Model: "nn_wm_61"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
dense_238 (Dense)            multiple                  24
_________________________________________________________________
dense_239 (Dense)            multiple                  28
_________________________________________________________________
dense_240 (Dense)            multiple                  15
_________________________________________________________________
dense_241 (Dense)            multiple                  12
_________________________________________________________________
dense_242 (Dense)            multiple                  4
=================================================================
Total params: 83
Trainable params: 83
Non-trainable params: 0
_________________________________________________________________
```

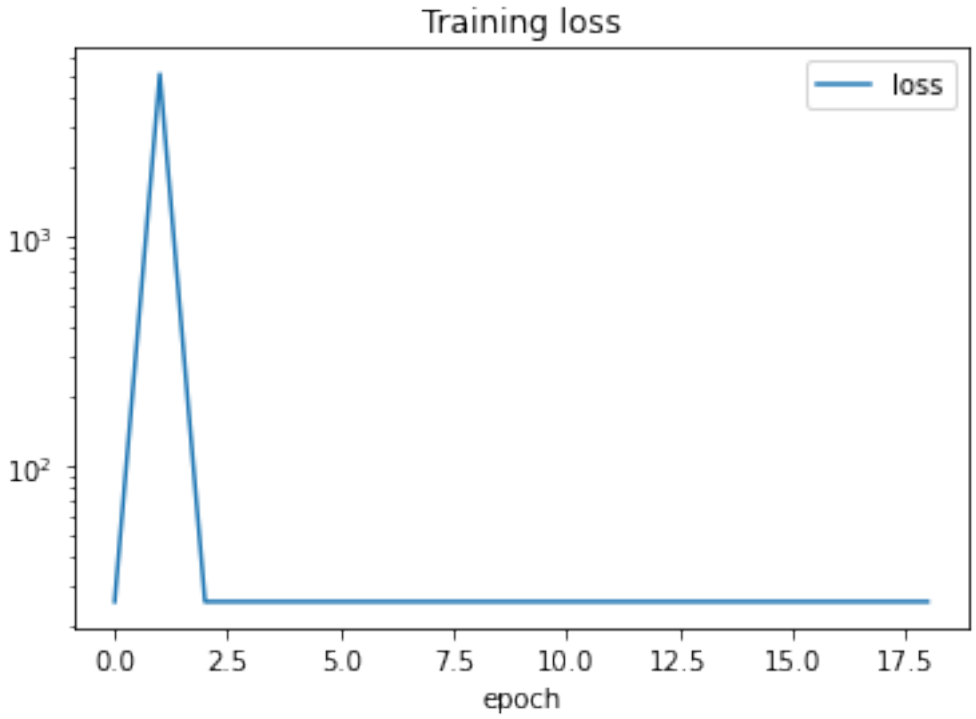
[577]:
```
X_1, Y_1 = load_feature_data('NN2_h50box16')
X_2, Y_2 = load_feature_data('NN2_h50box32')
X_3, Y_3 = load_feature_data('NN2_h30box16')
X_4, Y_4 = load_feature_data('NN2_h30box32')

X_train = np.ndarray((4,*np.shape(X_1)))
Y_train = np.ndarray((4,*np.shape(Y_1)))

X_train[0,:,:] = X_1
X_train[1,:,:] = X_2
X_train[2,:,:] = X_3
X_train[3,:,:] = X_4
Y_train[0,:,:] = Y_1
Y_train[1,:,:] = Y_2
Y_train[2,:,:] = Y_3
Y_train[3,:,:] = Y_4


model_fit(model2, X_train, Y_train)
```

```
Convergence information:
loss: 25.70275403559208
number function evaluations: 19
number iterations: 1
success flag: True
convergence message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
WARNING:tensorflow:11 out of the last 11 calls to <function
function_factory.<locals>.assign_new_model_parameters at 0x000002A3598E5798>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
```

```
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

[577]:
```
      fun: 25.70275403559208
 hess_inv: <83x83 LbfgsInvHessProduct with dtype=float64>
      jac: array([ 1.75791249e-04,  2.93408095e-03,  8.44084664e-04,
3.67374591e-03,
       -7.85015288e-03,  1.87360340e-02, -4.95787353e-04, -2.88080060e-04,
       -3.79485911e-04, -1.72871278e-03, -1.51685443e-02,  3.30698132e-02,
       -9.36505125e-04,  1.14782121e-02,  2.18290517e-03,  8.74942310e-03,
       -9.89813909e-02,  2.17410121e-01, -3.76204358e-04,  4.25165433e-03,
        8.38487607e-04,  3.42990450e-03, -2.33232359e-02,  4.75240476e-02,
        2.17993011e-02,  9.03638272e-03,  2.67174715e-03, -4.62764086e-02,
        1.96319579e-02,  8.62044019e-03,  2.47757723e-03, -4.29812679e-02,
        2.18911498e-02,  8.73526047e-03,  2.47510177e-03, -4.74077666e-02,
       -1.61401205e-02, -6.88308278e-03, -1.49747675e-03,  4.08039030e-02,
        1.52790496e-02,  5.96729122e-03,  2.05534287e-03, -2.79625840e-02,
       -1.46532382e-03,  1.24099765e-03,  6.16607537e-04,  8.23853205e-03,
        2.22104549e-02,  9.40847622e-03,  2.90150552e-03, -4.59390136e-02,
        1.26939962e-03, -7.62249935e-03,  9.81858137e-03, -4.33948633e-03,
        1.01219445e-02, -1.74653604e-02,  6.76518218e-03, -9.20281675e-03,
        2.20367524e-02,  1.72556196e-03,  1.06525485e-03,  1.26560147e-03,
       -9.97880478e-03,  1.51888602e-02, -3.46280715e-02,  1.24949207e-03,
        5.81737106e-04,  2.65226218e-04, -2.82882377e-03,  4.14419923e-03,
        5.17755700e-03, -3.37705344e-03, -3.97443094e-04, -5.40800123e-04,
       -1.87265034e-02,  8.89574673e-03,  1.07217892e-02, -1.86941386e-02,
        9.67275541e-03,  9.41190537e-03,  3.47161426e-02])
  message: 'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'
     nfev: 19
      nit: 1
     njev: 19
   status: 0
  success: True
        x: array([ 4.67584018e-01, -4.09556838e-01,  7.96295059e-02,
6.09498470e-02,
        8.71015586e-01,  3.97291133e-01,  7.35619259e-02,  5.13421924e-01,
       -4.01532321e-01, -1.17390104e-01,  8.47157777e-01, -3.27835111e-01,
        7.19059504e-01,  7.70645521e-01,  6.30247213e-01, -4.93955816e-01,
       -3.89153076e-02, -1.69362700e-02,  8.69712825e-03,  5.99649179e-03,
        1.68753122e-01,  2.38205769e-02,  6.27067831e-02, -4.88093362e-02,
       -1.58518246e-01, -1.32957549e-01,  1.51126221e-02, -2.41432881e-01,
       -2.24448945e-03,  8.23814410e-01, -5.92782195e-01, -6.02909415e-01,
        4.89301029e-01, -3.48139765e-01,  5.02129256e-01,  1.94578377e-01,
        4.65587595e-01, -5.76923254e-01,  3.59306839e-01, -3.87507586e-04,
```

```
-3.82648502e-01, -3.58627424e-01, -4.60415575e-01,  5.63535264e-01,
 1.22232967e+00, -1.06525142e+00, -4.25125094e-01, -9.01746502e-01,
 4.99543014e-02,  7.08848695e-02, -2.01927657e-01,  9.83333772e-02,
 3.38402023e-01,  7.83666067e-01, -6.90544657e-01, -4.38970141e-01,
-1.85608867e-01, -2.90972500e-01, -5.46859636e-01, -4.43403381e-01,
-2.08036778e-01,  4.10245676e-01, -1.04133555e+00,  7.68255028e-01,
-2.85315299e-02,  4.07700681e-01, -4.40098872e-02, -1.59725005e-02,
-2.27969612e-01, -7.60960145e-01, -6.28160160e-01,  4.12216698e-01,
 5.12478538e-01,  1.30331978e+00, -9.76688320e-01, -5.50377130e-01,
-3.81000372e-01,  1.52039542e-02,  1.17320901e-04, -1.03083735e+00,
 3.38945930e-01,  3.48326279e-01,  2.48396881e-01])
```

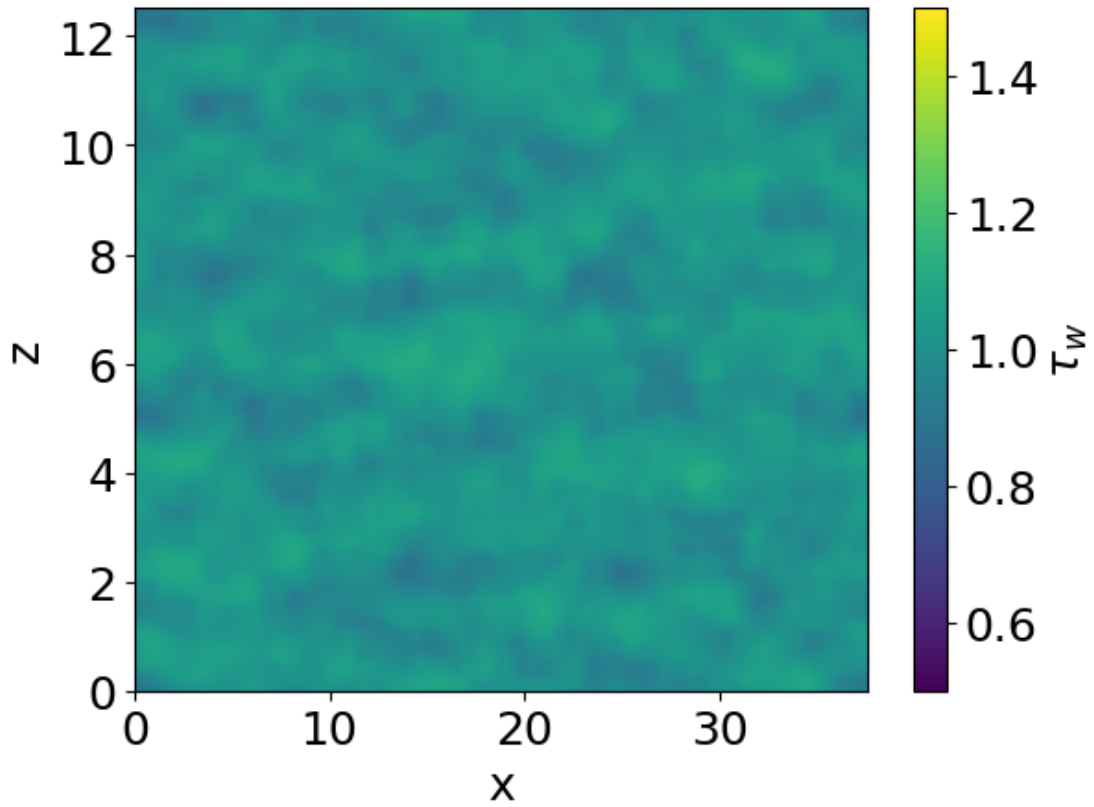


[681]:
```
X_1, Y_1 = load_feature_data('NN2_h50box32')
X_train = np.ndarray((1,*np.shape(X_1)))
Y_train = np.ndarray((1,*np.shape(Y_1)))
X_train[0] = X_1
Y_train[0] = Y_1

print(model2.loss(X_train,Y_train))

Y_pred = model2(X_train)

N0 = np.size(Y_pred,1)
```

```
Y_pred = np.reshape(Y_pred[0,0:int(N0/2),0],[int(512/2), int(768/2)])
x = np.linspace(0,12*np.pi,int(768/2)+1)[0:int(768/2)]
z = np.linspace(0,4*np.pi,int(512/2)+1)[0:int(512/2)]
plt.pcolor(x,z,Y_pred)
cbar = plt.colorbar()
cbar.set_label(r'$\tau_w$')
plt.clim(0.5,1.5)
plt.ylabel('z')
plt.xlabel('x')
#plt.gca().set_aspect('equal', adjustable='box')
plt.draw()

plt.savefig('NN2_tau_pred_h50box32.png')
```

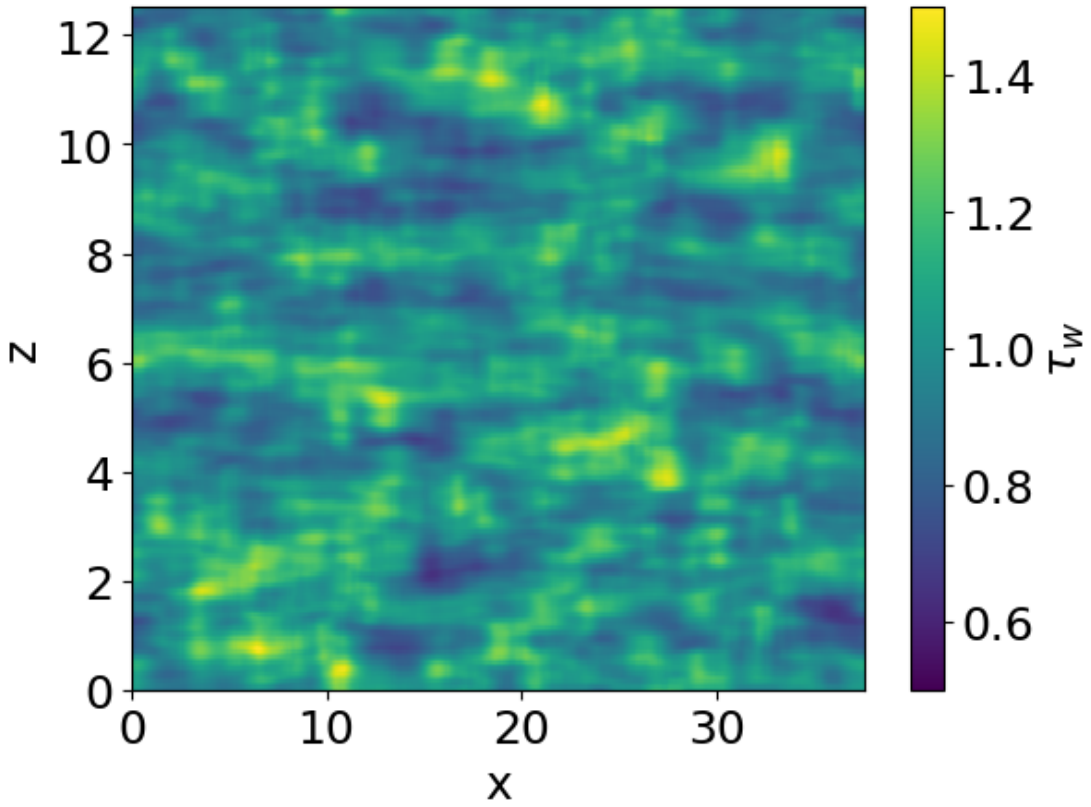
```
tf.Tensor(13.487759046256542, shape=(), dtype=float64)
```

```
C:\Users\Michael\Anaconda3\envs\me343\lib\site-
packages\ipykernel_launcher.py:15: MatplotlibDeprecationWarning: shading='flat'
when X and Y have the same dimensions as C is deprecated since 3.3.  Either
specify the corners of the quadrilaterals with X and Y, or pass shading='auto',
'nearest' or 'gouraud', or set rcParams['pcolor.shading'].  This will become an
error two minor releases later.
  from ipykernel import kernelapp as app
```

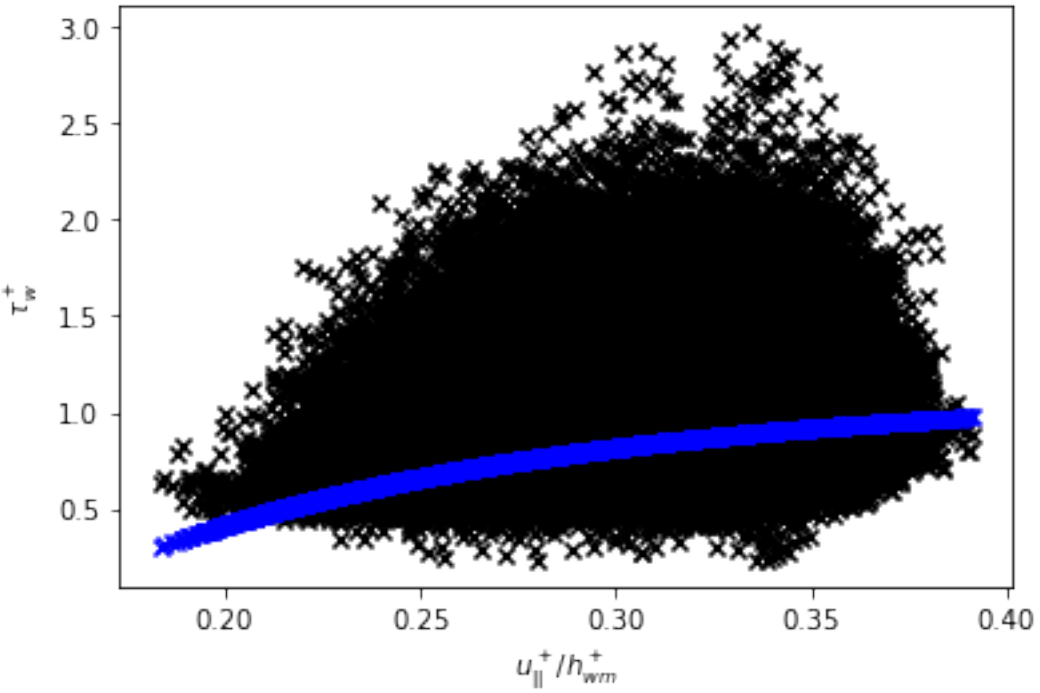


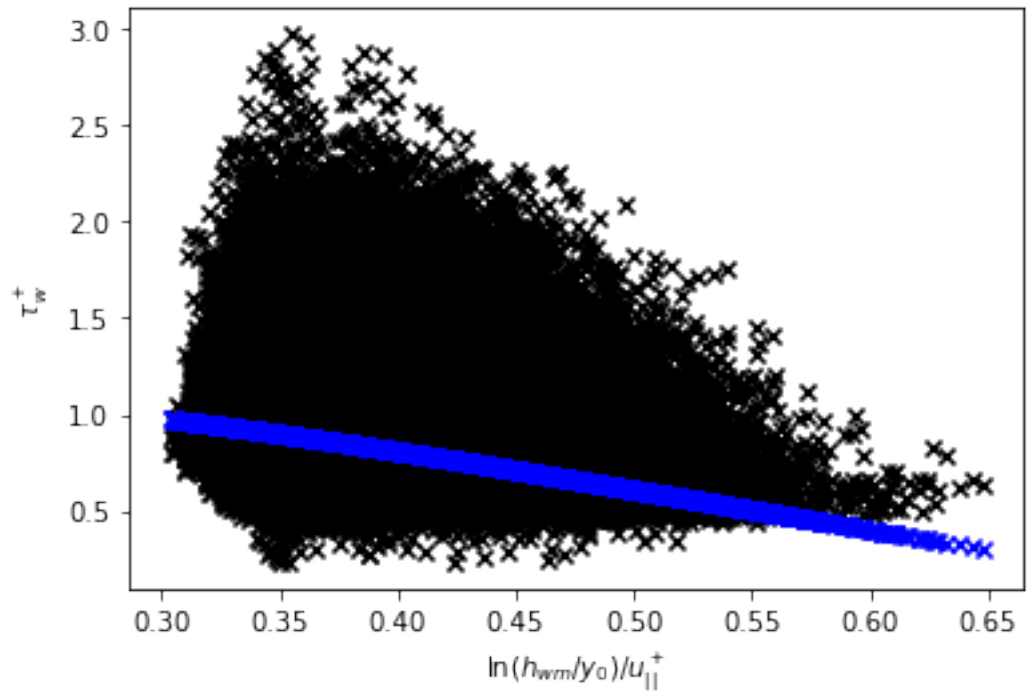
[682]:
```
N0 = np.size(Y_train,1)
Y_true = np.reshape(Y_train[0,0:int(N0/2),0],[int(512/2), int(768/2)])
x = np.linspace(0,12*np.pi,int(768/2)+1)[0:int(768/2)]
z = np.linspace(0,4*np.pi,int(512/2)+1)[0:int(512/2)]
plt.pcolor(x,z,Y_true)
cbar = plt.colorbar()
cbar.set_label(r'$\tau_w$')
plt.clim(0.5,1.5)
plt.ylabel('z')
plt.xlabel('x')
#plt.gca().set_aspect('equal', adjustable='box')
plt.draw()

plt.savefig('NN2_tau_true_h50box32.png')
```

```
C:\Users\Michael\Anaconda3\envs\me343\lib\site-packages\ipykernel_launcher.py:5:
MatplotlibDeprecationWarning: shading='flat' when X and Y have the same
dimensions as C is deprecated since 3.3.  Either specify the corners of the
quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or
set rcParams['pcolor.shading'].  This will become an error two minor releases
later.
  """
```



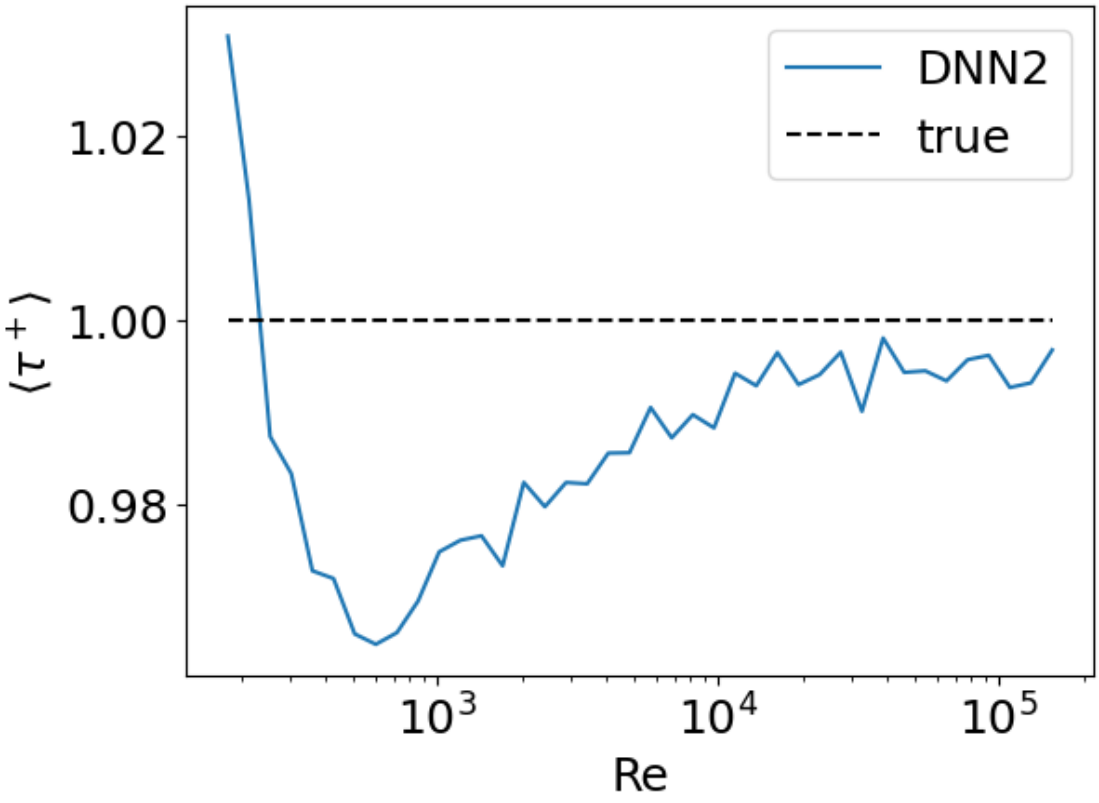
[599]:
```
Y_pred = model2(X_train)

plt.scatter(X_train[0,:,0], Y_train[0,:,0], marker='x', color='k')
plt.scatter(X_train[0,:,0], Y_pred[0,:,0], marker='x', color='b')
plt.xlabel(r'$u^+_{||}/h^+_{wm}$')
plt.ylabel(r'$\tau^+_w$')
plt.show()

plt.scatter(X_train[0,:,1], Y_train[0,:,0], marker='x', color='k')
plt.scatter(X_train[0,:,1], Y_pred[0,:,0], marker='x', color='b')
plt.xlabel(r'ln$(h_{wm}/y_0)/u^+_{||}$')
plt.ylabel(r'$\tau^+_w$')
plt.show()
```

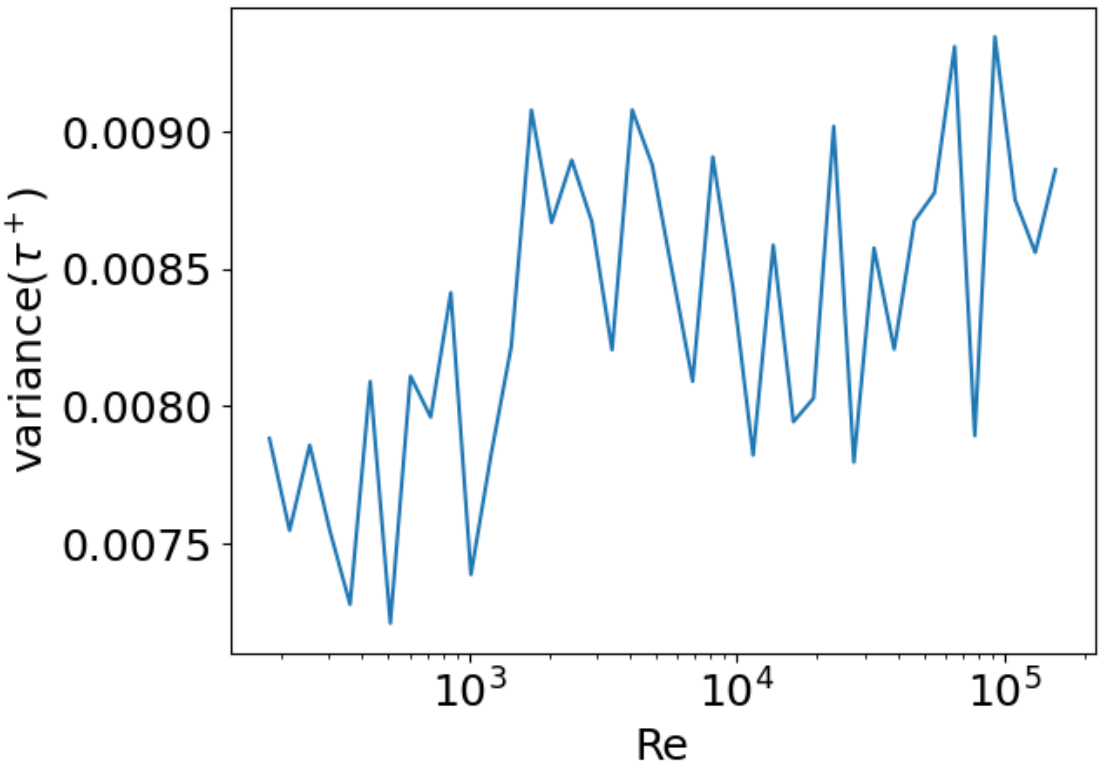

3.0
2.5
2.0
1.5
1.0
0.5
τ_w^+
0.20
0.25
0.30
0.35
0.40
$u_{\|}^+ / h_{wm}^+$


3.0
2.5
2.0
1.5
1.0
0.5
τ_w^+
0.30
0.35
0.40
0.45
0.50
0.55
0.60
0.65
$\ln(h_{wm}/y_0)/u_{||}^+$

[687]:
```
# Generate pseudo-velocity data at Re=4200
Re_tau = 180*2**np.arange(0,10,0.25)
tau = np.zeros(np.size(Re_tau))
tau_var = np.zeros(np.size(Re_tau))
for jj,Re in enumerate(Re_tau):
    nu = 1/Re
    utau = 1
    kappa = 0.41
    B = 5.0
    y0 = nu/utau*np.exp(-kappa*B)

    hwm = 0.10 # 10% of boundary layer thickness
    var = 0.1
    upar = 1/kappa*np.log(hwm/y0)*(1+var*np.random.randn(1000))
    deltaR = 4.0

    xf1 = (upar/utau)/(hwm*utau/nu)
    xf2 = np.log(hwm/y0)/(upar/utau)
    xf3 = deltaR*np.ones(np.shape(xf2))

    X_pseudo = np.zeros((3,np.size(xf1)))
    X_pseudo[0,:] = xf1
    X_pseudo[1,:] = xf2
    X_pseudo[2,:] = xf3

    X_in = np.ndarray((1,*np.shape(X_pseudo.T)))
    X_in[0] = X_pseudo.T

    Y_pseudo = model2(X_in)
    tau[jj] = np.mean(Y_pseudo[0,:,0])
    tau_var[jj] = np.var(Y_pseudo[0,:,0])

plt.plot(Re_tau, tau)
plt.plot(Re_tau, 0*Re_tau+1,'--k')
plt.xscale('log')
plt.xlabel('Re')
plt.ylabel(r'$\langle\tau^+\rangle$')
plt.legend(['DNN2','true'])
plt.tight_layout()
plt.draw()

plt.savefig('NN2_tau_v_Re.png')

plt.show()

plt.plot(Re_tau, tau_var)
plt.xscale('log')
```

```
plt.xlabel('Re')
plt.ylabel(r'variance($\tau^+$)')
plt.show()
```





[654]:
```
for h in [30,50]:
    for box in [8,16,32,64]:
        X_1, Y_1 = load_feature_data("NN2_h{0:d}box{1:d}".format(h,box))
        X_train = np.ndarray((1,*np.shape(X_1)))
        Y_train = np.ndarray((1,*np.shape(Y_1)))
        X_train[0] = X_1
        Y_train[0] = Y_1

        Y_pred = model2(X_train)

        print("{0:d}, {1:d}, {2:.1f}".format(h,box,np.mean(abs(Y_pred-Y_train)/
 ↪Y_train)*100))
```

```
30, 8, 21.5
30, 16, 15.3
30, 32, 9.0
30, 64, 10.6
50, 8, 22.7
50, 16, 16.2
50, 32, 9.4
50, 64, 6.9
```

[ ]: