MTF271 Turbulence Modelling
Assignment 1, Part II: Machine Learning

Lars Davidson

Division of Fluid Dynamics, Dept. Mechanics and Maritime Sciences
Chalmers University of Technology, Gothenburg, Sweden.

The Assignment, slides and recorded lecture can be found at

https://www.tfd.chalmers.se/~lada/comp_turb_model/assignment_1/index.html

Machine Learning

- Machine learning (ML) is a method where known data are used for teaching the algorithm to classify another set of data.
  - Photographs where the machine learning algorithm should recognize, e.g., traffic lights [4].

- Machine learning (ML) is a method where known data are used for teaching the algorithm to classify another set of data.
  - Photographs where the machine learning algorithm should recognize, e.g., traffic lights [4].
  - ECG signals where the machine learning algorithm should recognize certain unhealthy conditions of the heart [2].

- Machine learning (ML) is a method where known data are used for teaching the algorithm to classify another set of data.
  - Photographs where the machine learning algorithm should recognize, e.g., traffic lights [4].
  - ECG signals where the machine learning algorithm should recognize certain unhealthy conditions of the heart [2].
  - Detecting fraud for credit card payments [3].

Machine Learning

- Machine learning (ML) is a method where known data are used for teaching the algorithm to classify another set of data.
  - Photographs where the machine learning algorithm should recognize, e.g., traffic lights [4].
  - ECG signals where the machine learning algorithm should recognize certain unhealthy conditions of the heart [2].
  - Detecting fraud for credit card payments [3].
  - Machine learning methods such as Support Vector Machines (SVM) and neural networks are used for solving this type of problems.

- Machine learning (ML) is a method where known data are used for teaching the algorithm to classify another set of data.
  - Photographs where the machine learning algorithm should recognize, e.g., traffic lights [4].
  - ECG signals where the machine learning algorithm should recognize certain unhealthy conditions of the heart [2].
  - Detecting fraud for credit card payments [3].
  - Machine learning methods such as Support Vector Machines (SVM) and neural networks are used for solving this type of problems.
  - Through as much data as possible at ML?

Machine Learning

- Machine learning (ML) is a method where known data are used for teaching the algorithm to classify another set of data.
  - Photographs where the machine learning algorithm should recognize, e.g., traffic lights [4].
  - ECG signals where the machine learning algorithm should recognize certain unhealthy conditions of the heart [2].
  - Detecting fraud for credit card payments [3].
  - Machine learning methods such as Support Vector Machines (SVM) and neural networks are used for solving this type of problems.
  - Through as much data as possible at ML?
- In my case, input and output are numerical values. Regression methods should then be used [2]; I use support vector regression (SVR) methods available in Python.

Training: I need a target database

$$
\begin{aligned}
\frac{\partial v_i}{\partial x_i} &= 0 \\
\frac{\partial v_i}{\partial t} + \frac{\partial}{\partial x_j}(v_i v_j) &= -\frac{\partial p}{\partial x_i} + \frac{\partial^2 v_i}{\partial x_j \partial x_j}
\end{aligned}
$$

- Fully-developed Channel flow
- Database can be found here. Reynolds number is 5 200.
- The DNS data are averaged in time, x_1 and x_3.

SVR (Support Vector Regression)

At this site you find the figure below

SVR data predictions

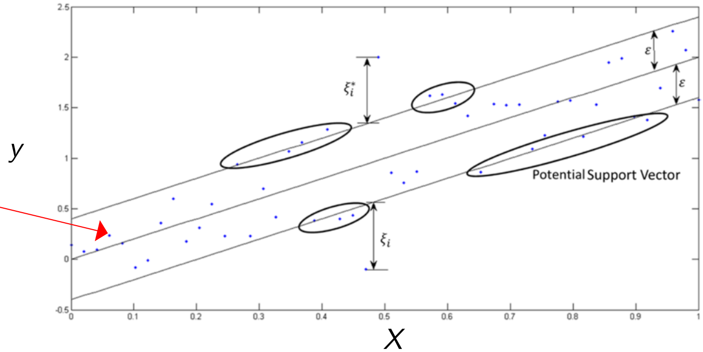


Figure: Hyperplane, ε tube and slack, ξ_i. $\cdot$: predicted (SVR) data point

At this site you find the figure below

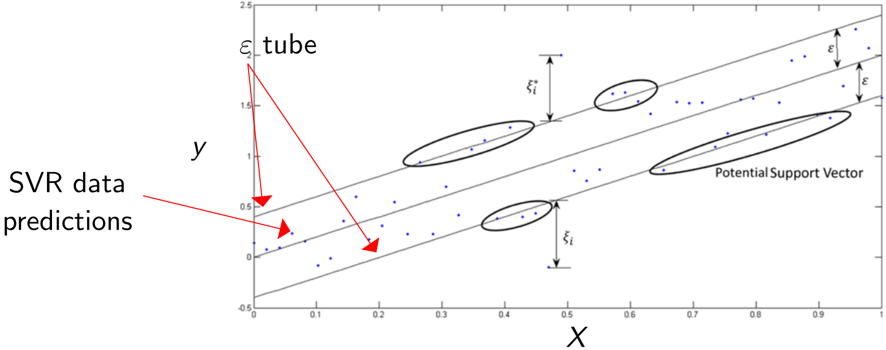


Figure: Hyperplane, ε tube and slack, ξ_i. $\cdot$: predicted (SVR) data point

SVR (Support Vector Regression)

At this site you find the figure below

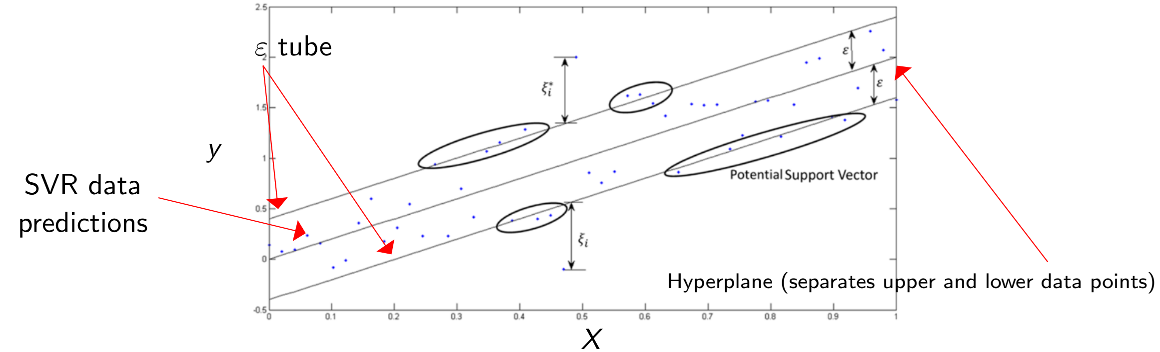


Figure: Hyperplane, ε tube and slack, ξ_i. •: predicted (SVR) data point

SVR (Support Vector Regression)

At this site you find the figure below

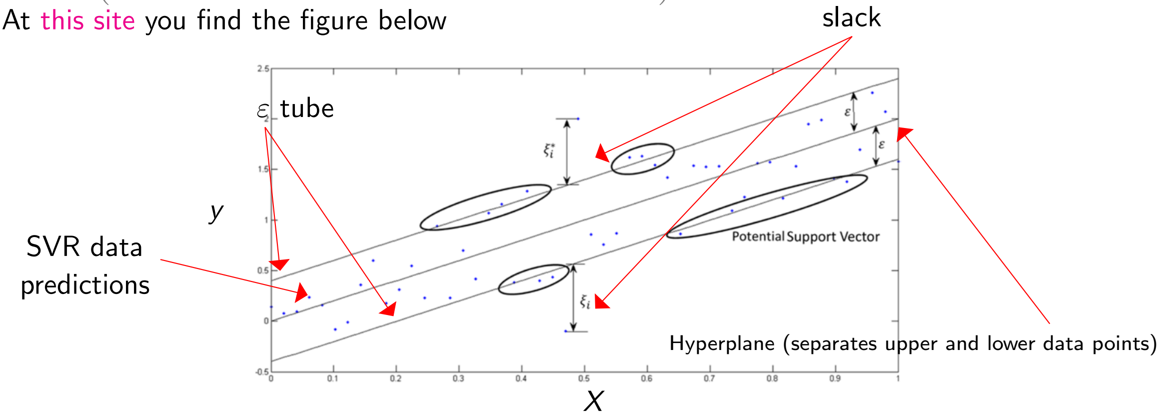


Figure: Hyperplane, ε tube and slack, ξ_i. $\cdot$: predicted (SVR) data point

SVR (Support Vector Regression)

At this site you find the figure below

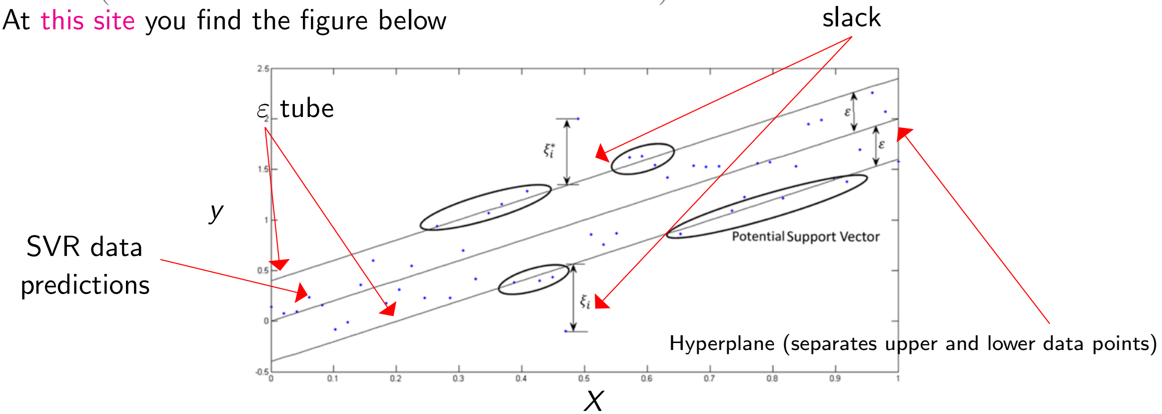


Figure: Hyperplane, ε tube and slack, ξ_i. $\cdot$: predicted (SVR) data point

- How close to the Hyperplane should the predicted data points be?

SVR (Support Vector Regression)

At this site you find the figure below

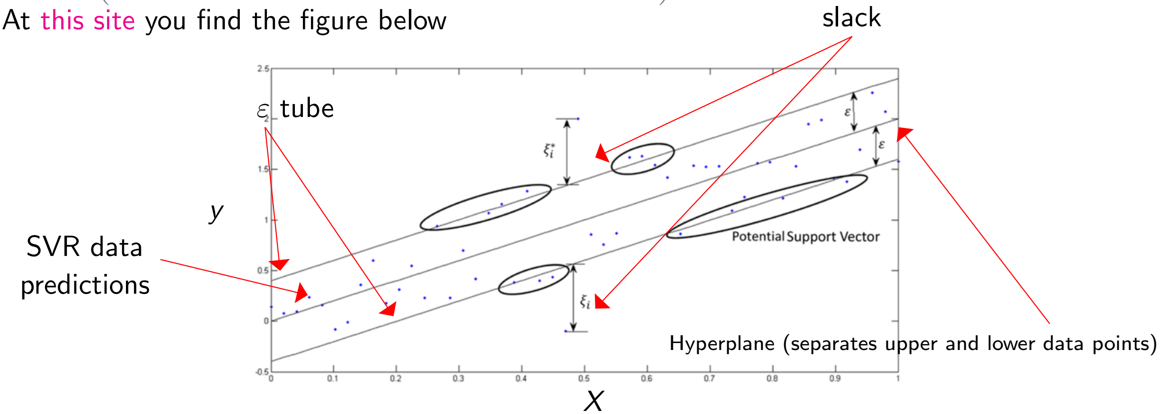


Figure: Hyperplane, ε tube and slack, ξ_i. $\cdot$: predicted (SVR) data point

- How close to the Hyperplane should the predicted data points be? Answer: ε

SVR (Support Vector Regression)

At this site you find the figure below

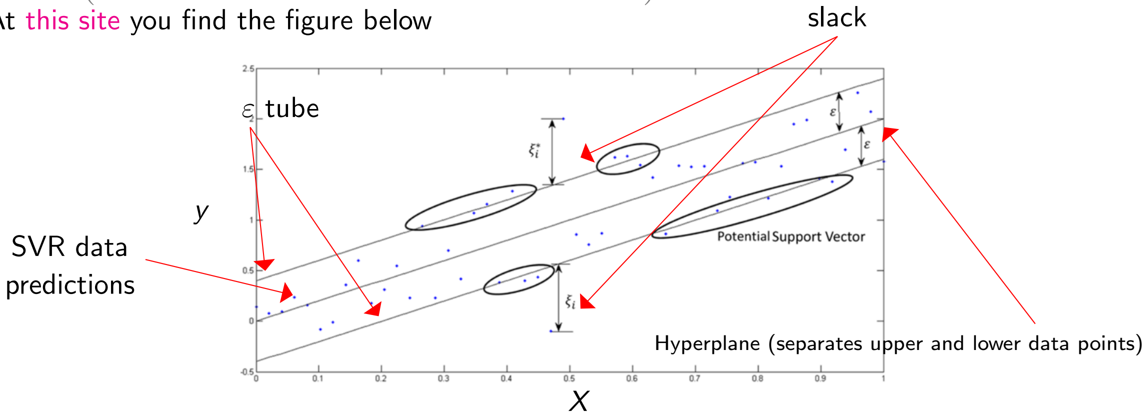


Figure: Hyperplane, ε tube and slack, ξ_i. $\cdot$: predicted (SVR) data point

- How close to the Hyperplane should the predicted data points be? Answer: ε
- How far outside of the ε tube can a predicted point be?

At this site you find the figure below

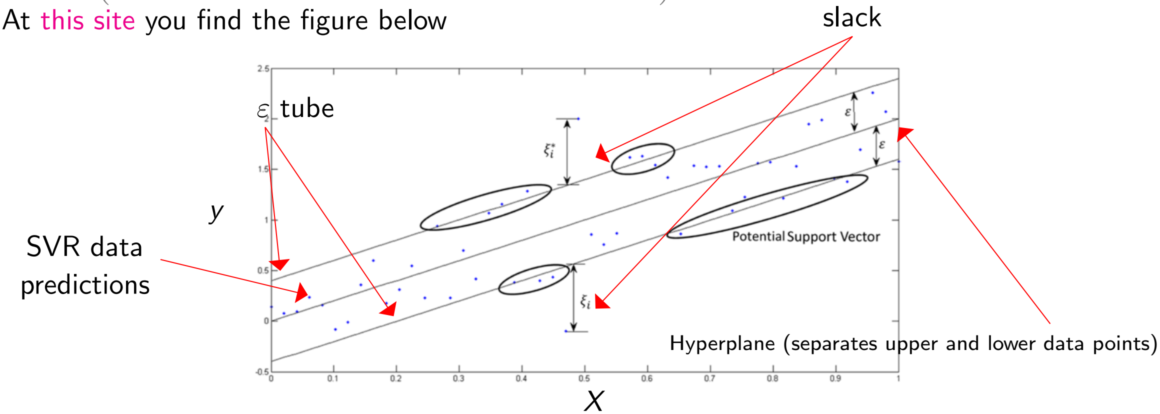
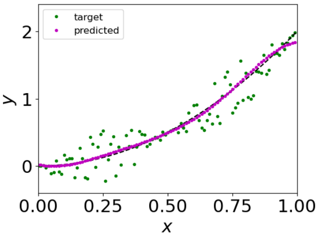


FIGURE: Hyperplane, ε tube and slack, ξ_i. $\cdot$: predicted (SVR) data point

- How close to the Hyperplane should the predicted data points be? Answer: ε
- How far outside of the ε tube can a predicted point be? Answer: ξ (large C, promotes small ξ)

Testing SVR for a line

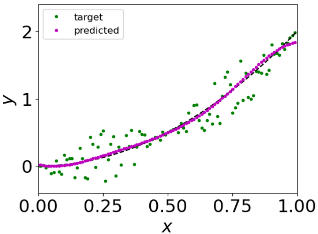
$$y = 2x^2 + \sin(2\pi) \cdot rand(-0.4, 0.4)$$



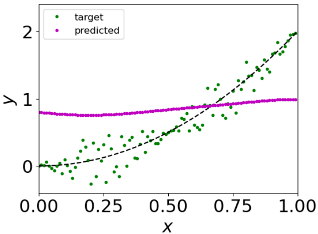
(A) $C = 0.1$, $\varepsilon = 0.1$

Testing SVR for a line

$$y = 2x^2 + \sin(2\pi) \cdot rand(-0.4, 0.4)$$



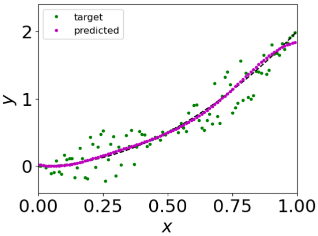
(a) $C = 0.1$, $\varepsilon = 0.1$



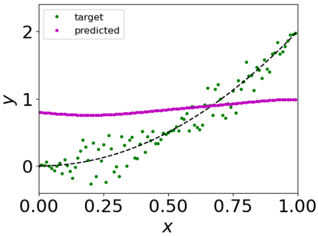
(b) $C = 0.1$, $\varepsilon = 1.0$

Testing SVR for a line

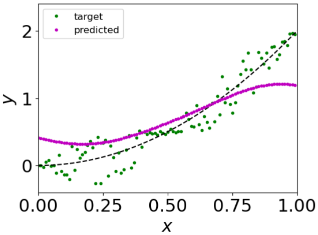
$$y = 2x^2 + \sin(2\pi) \cdot rand(-0.4, 0.4)$$



(A) $C = 0.1,\ \varepsilon = 0.1$



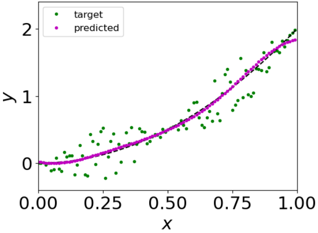
(B) $C = 0.1,\ \varepsilon = 1.0$



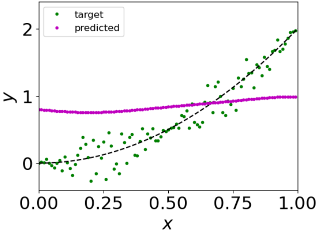
(C) $C = 0.1,\ \varepsilon = 0.4$

Testing SVR for a line

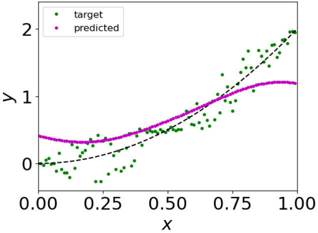
$$y = 2x^2 + \sin(2\pi) \cdot rand(-0.4, 0.4)$$



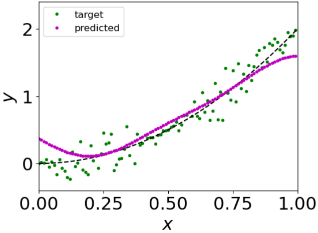
(A) $C = 0.1,\ \varepsilon = 0.1$



(B) $C = 0.1,\ \varepsilon = 1.0$



(C) $C = 0.1,\ \varepsilon = 0.4$



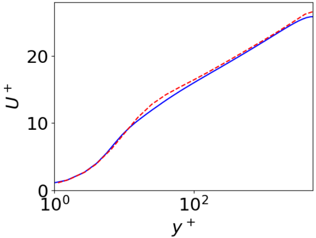
(D) $C = 1.0,\ \varepsilon = 0.4$

Channel flow

- We will use DNS data of channel flow to develop C_ω in $k-\omega$ model based on SVR

- We will use DNS data of channel flow to develop C_ω in $k-\omega$ model based on SVR
- Note that the standard $k-\omega$ models gives very good velocity profiles but a less good k, see below

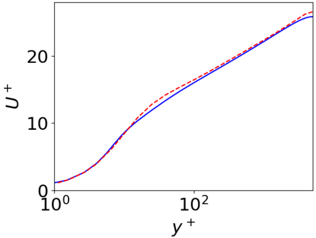
- We will use DNS data of channel flow to develop C_ω in $k-\omega$ model based on SVR
- Note that the standard $k-\omega$ models gives very good velocity profiles but a less good k, see below



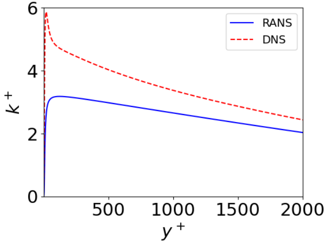
(A) Velocity

Channel flow

- We will use DNS data of channel flow to develop C_ω in $k-\omega$ model based on SVR
- Note that the standard $k-\omega$ models gives very good velocity profiles but a less good k, see below

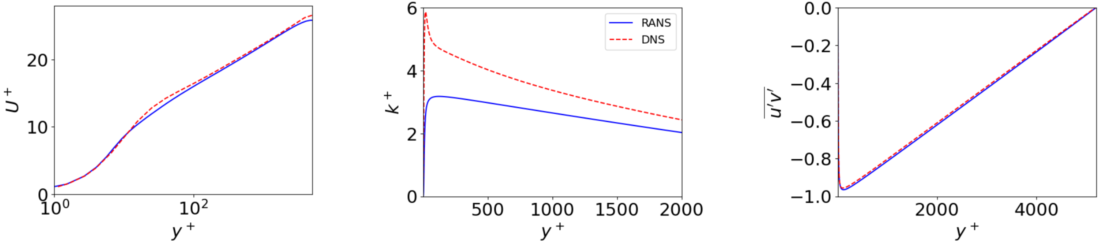


(A) Velocity



(B) Turbulent kinetic energy

- We will use DNS data of channel flow to develop C_ω in $k-\omega$ model based on SVR
- Note that the standard $k-\omega$ models gives very good velocity profiles but a less good k, see below



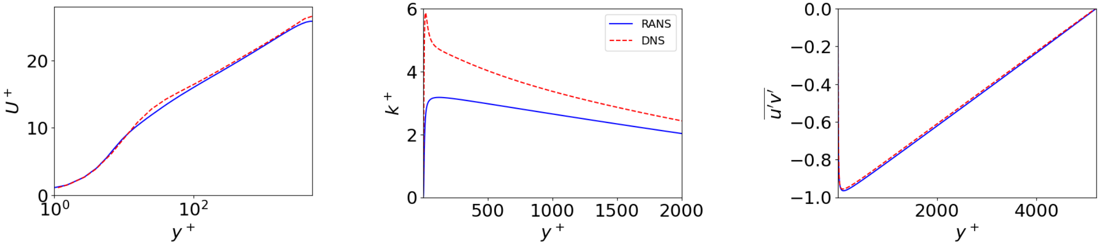
(A) Velocity

(B) Turbulent kinetic energy

(C) Turbulent shear stress

Channel flow

- We will use DNS data of channel flow to develop C_ω in $k-\omega$ model based on SVR
- Note that the standard $k-\omega$ models gives very good velocity profiles but a less good k, see below

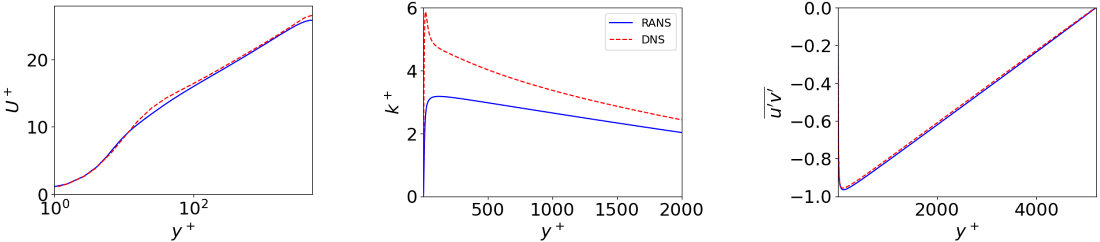


(A) Velocity (B) Turbulent kinetic energy (C) Turbulent shear stress

- The $k-\omega$ model predicts accurate Reynolds shear stress, $-\overline{v_1'v_2'} = \nu_t \dfrac{\partial \bar{v}_1}{\partial x_2}$ (all turbulence models predict a linear total shear stress, see Eq. 6.20 in the eBook)

Channel flow

- We will use DNS data of channel flow to develop C_ω in $k-\omega$ model based on SVR
- Note that the standard $k-\omega$ models gives very good velocity profiles but a less good k, see below



(A) Velocity (B) Turbulent kinetic energy (C) Turbulent shear stress

- The $k-\omega$ model predicts accurate Reynolds shear stress, $-\overline{v_1'v_2'} = \nu_t \dfrac{\partial \bar{v}_1}{\partial x_2}$ (all turbulence models predict a linear total shear stress, see Eq. 6.20 in the eBook)
- The velocity profile above is correct; hence also the turbulent viscosity is correct.

- Now we will create our SVR.

- Now we will create our SVR.
- Since we know that k and ω cannot be correctly predicted by the $k-\omega$ model (only their ratio k/ω is) we should not use them as input parameters.

- Now we will create our SVR.
- Since we know that k and ω cannot be correctly predicted by the $k-\omega$ model (only their ratio k/ω is) we should not use them as input parameters.
- Recall than in the standard $k-\omega$ model then $\nu_t = \dfrac{k}{\omega}$.

- Now we will create our SVR.
- Since we know that k and ω cannot be correctly predicted by the $k-\omega$ model (only their ratio k/ω is) we should not use them as input parameters.
- Recall than in the standard $k-\omega$ model then $\nu_t = \dfrac{k}{\omega}$.
- We choose ν_t and $\dfrac{\partial \bar{v}_1}{\partial x_2}$ as input parameters.

- Now we will create our SVR.
- Since we know that k and ω cannot be correctly predicted by the $k-\omega$ model (only their ratio k/ω is) we should not use them as input parameters.
- Recall than in the standard $k-\omega$ model then $\nu_t = \dfrac{k}{\omega}$.
- We choose ν_t and $\dfrac{\partial \bar{v}_1}{\partial x_2}$ as input parameters.
- The target is the shear stress $-\overline{v_1' v_2'}$.

- Now we will create our SVR.
- Since we know that k and ω cannot be correctly predicted by the $k-\omega$ model (only their ratio k/ω is) we should not use them as input parameters.
- Recall than in the standard $k-\omega$ model then $\nu_t = \dfrac{k}{\omega}$.
- We choose ν_t and $\dfrac{\partial \bar{v}_1}{\partial x_2}$ as input parameters.
- The target is the shear stress $-\overline{v_1' v_2'}$.
- We then compute C_ω where C_ω is defined from $\nu_t = C_\omega \dfrac{k}{\omega}$.

- Now we will create our SVR.
- Since we know that k and ω cannot be correctly predicted by the $k-\omega$ model (only their ratio k/ω is) we should not use them as input parameters.
- Recall than in the standard $k-\omega$ model then $\nu_t = \dfrac{k}{\omega}$.
- We choose ν_t and $\dfrac{\partial \bar{v}_1}{\partial x_2}$ as input parameters.
- The target is the shear stress $-\overline{v_1' v_2'}$.
- We then compute C_ω where C_ω is defined from $\nu_t = C_\omega \dfrac{k}{\omega}$.
- Download the Python script `ML-channel.py` and all datafiles. (in `ML-channel.py` I use $\partial \bar{v}_1/\partial x_2$ as input and $-\overline{v_1' v_2'}$ as output)

ML model (SVR)

- The main difference to the script `ML-python.py` is that we have two influence parameters instead of one i.e. two StandardScaler

ML MODEL (SVR)

- The main difference to the script `ML-python.py` is that we have two influence parameters instead of one i.e. two StandardScaler

```
# scale input data
scaler_dudy=StandardScaler()
scaler_vist=StandardScaler()
dudy_in=scaler_dudy.fit_transform(dudy_in)
vist_in=scaler_vist.fit_transform(vist_in)
```

ML MODEL (SVR)

- The main difference to the script `ML-python.py` is that we have two influence parameters instead of one i.e. two StandardScaler

```
# scale input data
scaler_dudy=StandardScaler()
scaler_vist=StandardScaler()
dudy_in=scaler_dudy.fit_transform(dudy_in)
vist_in=scaler_vist.fit_transform(vist_in)
```

- and higher dimension of `X`, i.e.

ML model (SVR)

- The main difference to the script `ML-python.py` is that we have two influence parameters instead of one i.e. two StandardScaler

```
# scale input data
scaler_dudy=StandardScaler()
scaler_vist=StandardScaler()
dudy_in=scaler_dudy.fit_transform(dudy_in)
vist_in=scaler_vist.fit_transform(vist_in)
```

- and higher dimension of `X`, i.e.

```
# setup X (input) and y (output)
X=np.zeros((n_svr,2))
y=uv_out
X[:,0]=dudy_in[:,0]
X[:,1]=vist_in[:,0]
```

ML model (SVR)

```
# output uv
uv_all_data=uv_DNS
# input dudy
dudy_all_data=dudy_DNS
# input vist
vist_all_data=vist_DNS
```

ML model (SVR)

```
# output uv
uv_all_data=uv_DNS
# input dudy
dudy_all_data=dudy_DNS
# input vist
vist_all_data=vist_DNS
```

- Now pick 20% as test data

ML model (SVR)

```
# output uv
uv_all_data=uv_DNS
# input dudy
dudy_all_data=dudy_DNS
# input vist
vist_all_data=vist_DNS
```

- Now pick 20% as test data

```
# create indices for all data
index= np.arange(0,len(uv_all_data), dtype=int)
# number of elements of test data, 20 \%
n_test=int(0.2*len(uv_all_data))
# pick 20\% elements randomly (test data)
index_test=np.random.choice(index, size=n_test, replace=False)
# pick every 5th elements
#index_test=index[::5]
```

ML MODEL (SVR)

```
# output uv
uv_all_data=uv_DNS
# input dudy
dudy_all_data=dudy_DNS
# input vist
vist_all_data=vist_DNS
```

- Now pick 20% as test data

```
# create indices for all data
index= np.arange(0,len(uv_all_data), dtype=int)
# number of elements of test data, 20 \%
n_test=int(0.2*len(uv_all_data))
# pick 20\% elements randomly (test data)
index_test=np.random.choice(index, size=n_test, replace=False)
# pick every 5th elements
#index_test=index[::5]
```

```
dudy_test=dudy_all_data[index_test]
vist_test=vist_all_data[index_test]
uv_out_test=uv_all_data[index_test]
```

ML model (SVR)

- The remaining 80% are used as training data

ML model (SVR)

- The remaining 80% are used as training data

```
# delete testing data from 'all data' => training data
dudy_in=np.delete(dudy_all_data,index_test)
vist_in=np.delete(vist_all_data,index_test)
uv_out=np.delete(uv_all_data,index_test)
```

ML MODEL (SVR)

- The remaining 80% are used as training data

```
# delete testing data from 'all data' => training data
dudy_in=np.delete(dudy_all_data,index_test)
vist_in=np.delete(vist_all_data,index_test)
uv_out=np.delete(uv_all_data,index_test)
```

- The input arrays to `svr` must be reshaped

ML model (SVR)

- The remaining 80% are used as training data

```
# delete testing data from 'all data' => training data
dudy_in=np.delete(dudy_all_data,index_test)
vist_in=np.delete(vist_all_data,index_test)
uv_out=np.delete(uv_all_data,index_test)
```

- The input arrays to `svr` must be reshaped

```
# re-shape
dudy_in=dudy_in.reshape(-1, 1)
vist_in=vist_in.reshape(-1, 1)
```

ML model (SVR)

- The remaining 80% are used as training data

```
# delete testing data from 'all data' => training data
dudy_in=np.delete(dudy_all_data,index_test)
vist_in=np.delete(vist_all_data,index_test)
uv_out=np.delete(uv_all_data,index_test)
```

- The input arrays to `svr` must be reshaped

```
# re-shape
dudy_in=dudy_in.reshape(-1, 1)
vist_in=vist_in.reshape(-1, 1)
```

- It's a good habit to re-scale input variables because different inputs may have widely different magnitudes

ML MODEL (SVR)

- The remaining 80% are used as training data

```
# delete testing data from 'all data' => training data
dudy_in=np.delete(dudy_all_data,index_test)
vist_in=np.delete(vist_all_data,index_test)
uv_out=np.delete(uv_all_data,index_test)
```

- The input arrays to `svr` must be reshaped

```
# re-shape
dudy_in=dudy_in.reshape(-1, 1)
vist_in=vist_in.reshape(-1, 1)
```

- It's a good habit to re-scale input variables because different inputs may have widely different magnitudes

```
# scale input data
scaler_dudy=StandardScaler()
scaler_vist=StandardScaler()
dudy_in=scaler_dudy.fit_transform(dudy_in)
vist_in=scaler_vist.fit_transform(vist_in)
```

ML MODEL (SVR)

- Next, I create the input array to be sent to the SVR

ML model (SVR)

- Next, I create the input array to be sent to the SVR

```
# setup X (input) and y (output)
X=np.zeros((n_svr,2))
y=uv_out
X[:,0]=dudy_in[:,0]
X[:,1]=vist_in[:,0]
```

ML model (SVR)

- Next, I create the input array to be sent to the SVR

```
# setup X (input) and y (output)
X=np.zeros((n_svr,2))
y=uv_out
X[:,0]=dudy_in[:,0]
X[:,1]=vist_in[:,0]
```

- I choose type of SVR

ML MODEL (SVR)

- Next, I create the input array to be sent to the SVR

```
# setup X (input) and y (output)
X=np.zeros((n_svr,2))
y=uv_out
X[:,0]=dudy_in[:,0]
X[:,1]=vist_in[:,0]
```

- I choose type of SVR

```
# choose Machine Learning model
C=1
eps=0.001
model = SVR(kernel='rbf', epsilon = eps, C = C)
```

ML model (SVR)

- Next, I create the input array to be sent to the SVR

```
# setup X (input) and y (output)
X=np.zeros((n_svr,2))
y=uv_out
X[:,0]=dudy_in[:,0]
X[:,1]=vist_in[:,0]
```

- I choose type of SVR

```
# choose Machine Learning model
C=1
eps=0.001
model = SVR(kernel='rbf', epsilon = eps, C = C)
```

- and then I train (fit) the SVR

ML model (SVR)

- Next, I create the input array to be sent to the SVR

```
# setup X (input) and y (output)
X=np.zeros((n_svr,2))
y=uv_out
X[:,0]=dudy_in[:,0]
X[:,1]=vist_in[:,0]
```

- I choose type of SVR

```
# choose Machine Learning model
C=1
eps=0.001
model = SVR(kernel='rbf', epsilon = eps, C = C)
```

- and then I train (fit) the SVR

```
# Fit the model
svr = model.fit(X, y.flatten())
```

ML model (SVR)

- Now I will use the SVR model to *predict* $\overline{v_1' v_2'}$.

ML model (SVR)

- Now I will use the SVR model to *predict* $\overline{v_1' v_2'}$.
- I will use the test data (20% of the DNS data of $\partial U/\partial y$ and ν_t)

ML MODEL (SVR)

- Now I will use the SVR model to *predict* $\overline{v_1' v_2'}$.
- I will use the test data (20% of the DNS data of $\partial U/\partial y$ and ν_t)
- The target will be the corresponding $\overline{v_1' v_2'}$ of DNS (20%), i.e. `uv_out_test`

ML model (SVR)

- Now I will use the SVR model to *predict* $\overline{v_1' v_2'}$.
- I will use the test data (20% of the DNS data of $\partial U / \partial y$ and ν_t)
- The target will be the corresponding $\overline{v_1' v_2'}$ of DNS (20%), i.e. `uv_out_test`
- I reshape test data $\partial U / \partial y$ and ν_t

ML model (SVR)

- Now I will use the SVR model to *predict* $\overline{v_1' v_2'}$.
- I will use the test data (20% of the DNS data of $\partial U/\partial y$ and ν_t)
- The target will be the corresponding $\overline{v_1' v_2'}$ of DNS (20%), i.e. `uv_out_test`
- I reshape test data $\partial U/\partial y$ and ν_t

```
#  re-shape test data
dudy_test=dudy_test.reshape(-1, 1)
vist_test=vist_test.reshape(-1, 1)
```

ML model (SVR)

- Now I will use the SVR model to *predict* $\overline{v_1' v_2'}$.
- I will use the test data (20% of the DNS data of $\partial U/\partial y$ and ν_t)
- The target will be the corresponding $\overline{v_1' v_2'}$ of DNS (20%), i.e. `uv_out_test`
- I reshape test data $\partial U/\partial y$ and ν_t

```
#  re-shape test data
dudy_test=dudy_test.reshape(-1, 1)
vist_test=vist_test.reshape(-1, 1)
```

- I re-scale it (note: the *same* `StandardScaler` that I used when I trained the SVR)

ML MODEL (SVR)

- Now I will use the SVR model to *predict* $\overline{v'_1 v'_2}$.
- I will use the test data (20% of the DNS data of $\partial U/\partial y$ and ν_t)
- The target will be the corresponding $\overline{v'_1 v'_2}$ of DNS (20%), i.e. `uv_out_test`
- I reshape test data $\partial U/\partial y$ and ν_t

```
#  re-shape test data
dudy_test=dudy_test.reshape(-1, 1)
vist_test=vist_test.reshape(-1, 1)
```

- I re-scale it (note: the *same* `StandardScaler` that I used when I trained the SVR)

```
# scale test data
dudy_test=scaler_dudy.transform(dudy_test)
vist_test=scaler_vist.transform(vist_test)
```

ML MODEL (SVR)

- Now I will use the SVR model to *predict* $\overline{v_1' v_2'}$.
- I will use the test data (20% of the DNS data of $\partial U / \partial y$ and ν_t)
- The target will be the corresponding $\overline{v_1' v_2'}$ of DNS (20%), i.e. `uv_out_test`
- I reshape test data $\partial U / \partial y$ and ν_t

```
# re-shape test data
dudy_test=dudy_test.reshape(-1, 1)
vist_test=vist_test.reshape(-1, 1)
```

- I re-scale it (note: the *same* `StandardScaler` that I used when I trained the SVR)

```
# scale test data
dudy_test=scaler_dudy.transform(dudy_test)
vist_test=scaler_vist.transform(vist_test)
```

- and setup the `X_test` array

ML model (SVR)

- Now I will use the SVR model to *predict* $\overline{v_1' v_2'}$.
- I will use the test data (20% of the DNS data of $\partial U/\partial y$ and ν_t)
- The target will be the corresponding $\overline{v_1' v_2'}$ of DNS (20%), i.e. `uv_out_test`
- I reshape test data $\partial U/\partial y$ and ν_t

```
#  re-shape test data
dudy_test=dudy_test.reshape(-1, 1)
vist_test=vist_test.reshape(-1, 1)
```

- I re-scale it (note: the *same* `StandardScaler` that I used when I trained the SVR)

```
# scale test data
dudy_test=scaler_dudy.transform(dudy_test)
vist_test=scaler_vist.transform(vist_test)
```

- and setup the `X_test` array

```
# setup X (input) for testing (predicting)
X_test=np.zeros((n_test,2))
X_test[:,0]=dudy_test[:,0]
X_test[:,1]=vist_test[:,0]
```

ML MODEL (SVR)

- and then I make the prediction

ML model (SVR)

- and then I make the prediction

```
# predict uv
uv_predict= model.predict(X_test)
```

ML MODEL (SVR)

- and then I make the prediction

```
# predict uv
uv_predict= model.predict(X_test)
```

- Then I find the difference between target (`uv_out_test`) and prediction (`uv_predict`)

ML model (SVR)

- and then I make the prediction

```
# predict uv
uv_predict= model.predict(X_test)
```

- Then I find the difference between target (`uv_out_test`) and prediction (`uv_predict`)

```
# find difference between ML prediction and target
uv_error=np.std(uv_predict-uv_out_test)/\
(np.mean(uv_predict**2))**0.5
print('\nRMS error using ML turbulence model',uv_error)
```

ML model (SVR)

- and then I make the prediction

```
# predict uv
uv_predict= model.predict(X_test)
```

- Then I find the difference between target (`uv_out_test`) and prediction (`uv_predict`)

```
# find difference between ML prediction and target
uv_error=np.std(uv_predict-uv_out_test)/\
(np.mean(uv_predict**2))**0.5
print('\nRMS error using ML turbulence model',uv_error)
```

- The errors with `ML-channel.py` are large; when we here add ν_t as input and use $|\overline{v_1' v_2'}|$ as output, it gets much better.

ML MODEL (SVR)

- and then I make the prediction

```
# predict uv
uv_predict= model.predict(X_test)
```

- Then I find the difference between target (`uv_out_test`) and prediction (`uv_predict`)

```
# find difference between ML prediction and target
uv_error=np.std(uv_predict-uv_out_test)/\
(np.mean(uv_predict**2))**0.5
print('\nRMS error using ML turbulence model',uv_error)
```

- The errors with `ML-channel.py` are large; when we here add ν_t as input and use $|\overline{v_1' v_2'}|$ as output, it gets much better.
- One way to improve the accuracy is to use data only where the flow is fully turbulent, e.g. $100 < y^+ < 1000$

ML MODEL (SVR)

- and then I make the prediction

```
# predict uv
uv_predict= model.predict(X_test)
```

- Then I find the difference between target (`uv_out_test`) and prediction (`uv_predict`)

```
# find difference between ML prediction and target
uv_error=np.std(uv_predict-uv_out_test)/\
(np.mean(uv_predict**2))**0.5
print('\nRMS error using ML turbulence model',uv_error)
```

- The errors with `ML-channel.py` are large; when we here add ν_t as input and use $|\overline{v_1'v_2'}|$ as output, it gets much better.
- One way to improve the accuracy is to use data only where the flow is fully turbulent, e.g. $100 < y^+ < 1000$

```
index=np.nonzero((yplus_DNS > 100 ) & (yplus_DNS< 2000 ))
```

ML model (SVR)

- Now I'll export the model to disk. Export the model, the scalers and min/max of $\partial U/\partial y$ and ν_t.

ML model (SVR)

- Now I'll export the model to disk. Export the model, the scalers and min/max of $\partial U/\partial y$ and ν_t.

```
dump(model, 'model-svr.bin')
dump(scaler_dudy,'scalar-dudy-svr.bin')
dump(scaler_vist,'scalar-vist-svr.bin')
np.savetxt('min-max-svr.txt', [dudy_max, dudy_min, vist_max, vist_min])
```

ML model (SVR)

- Now I'll export the model to disk. Export the model, the scalers and min/max of $\partial U/\partial y$ and ν_t.

```
dump(model, 'model-svr.bin')
dump(scaler_dudy,'scalar-dudy-svr.bin')
dump(scaler_vist,'scalar-vist-svr.bin')
np.savetxt('min-max-svr.txt', [dudy_max, dudy_min, vist_max, vist_min])
```

- Next step is to import the SVR model into your Python CFD code (either `rans-k-omega.py`, pyCALC-RANS or your own)

ML model (SVR)

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from joblib import dump, load

folder='./'
filename=str(folder)+'model-svr.bin'
model = load(tr(folder)+'model-svr.bin')
scaler_dudy = load(str(folder)+'scalar-dudy-svr.bin')
scaler_vist = load(str(folder)+'scalar-vist-svr.bin')
dudy_max,dudy_min,vist_max,vist_min = np.loadtxt(str(folder)+'min-max-svr.
 txt')
```

ML MODEL (SVR)

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from joblib import dump, load

folder='./'
filename=str(folder)+'model-svr.bin'
model = load(tr(folder)+'model-svr.bin')
scaler_dudy = load(str(folder)+'scalar-dudy-svr.bin')
scaler_vist = load(str(folder)+'scalar-vist-svr.bin')
dudy_max,dudy_min,vist_max,vist_min = np.loadtxt(str(folder)+'min-max-svr.
 txt')
```

- Then you insert the SVR coding corresponding to the test/predict part in the previous slides. Note: you should NOT include the transform fitting, `scaler_dudy.fit_transform(dudy_in)`

ML model (SVR)

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from joblib import dump, load

folder='./'
filename=str(folder)+'model-svr.bin'
model = load(tr(folder)+'model-svr.bin')
scaler_dudy = load(str(folder)+'scalar-dudy-svr.bin')
scaler_vist = load(str(folder)+'scalar-vist-svr.bin')
dudy_max,dudy_min,vist_max,vist_min = np.loadtxt(str(folder)+'min-max-svr.
 txt')
```

- Then you insert the SVR coding corresponding to the test/predict part in the previous slides. Note: you should NOT include the transform fitting, `scaler_dudy.fit_transform(dudy_in)`
- You may find some useful Python coding at p. 14 in my report [1]

ML model: CFD code

- In the CFD code, we predict the shear stress in exactly the same way as in the testing phase above

ML model: CFD code

- In the CFD code, we predict the shear stress in exactly the same way as in the testing phase above
- Then we get C_ω as `cmu_omega=uv_predict/dudy/vist`

ML model: CFD code

- In the CFD code, we predict the shear stress in exactly the same way as in the testing phase above
- Then we get C_ω as `cmu_omega=uv_predict/dudy/vist`
- Note that you must invert the scaling for `vist` and `dudy` in the expression above, e.g. `scaler_vist.inverse_transform(vist)`

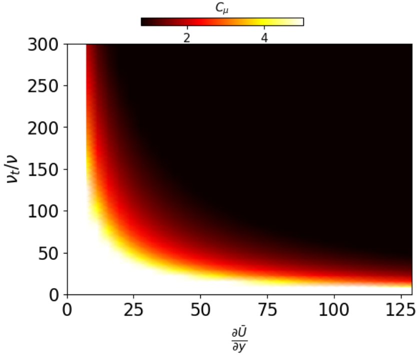
ML model: Problem 1

- Below I list some problems you're likely to encounter

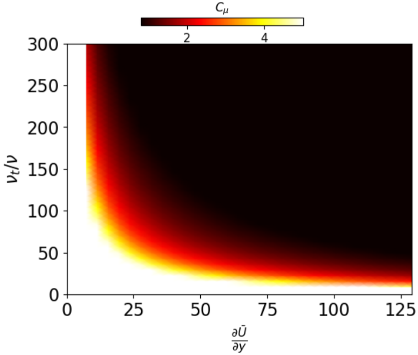
- Below I list some problems you're likely to encounter
- The figure below show the predicted C_ω (computed as $|\overline{v_1'v_2'}|_{SVR}/(\partial\bar{v}_1/\partial x_2)/\nu_t$ where $\partial\bar{v}_1/\partial x_2$ and ν_t cover the entire parameter space).

ML MODEL: PROBLEM 1

- Below I list some problems you're likely to encounter
- The figure below show the predicted C_ω (computed as $|\overline{v_1'v_2'}|_{SVR}/(\partial\bar{v}_1/\partial x_2)/\nu_t$ where $\partial\bar{v}_1/\partial x_2$ and ν_t cover the entire parameter space).



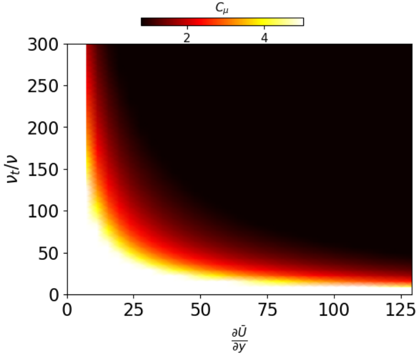
- Below I list some problems you're likely to encounter
- The figure below show the predicted C_ω (computed as $|\overline{v_1'v_2'}|_{SVR}/(\partial\bar{v}_1/\partial x_2)/\nu_t$ where $\partial\bar{v}_1/\partial x_2$ and ν_t cover the entire parameter space).



- When small $\partial\bar{v}_1/\partial x_2$ and ν_t are used (lower left corner), large C_ω are predicted.

ML model: Problem 1

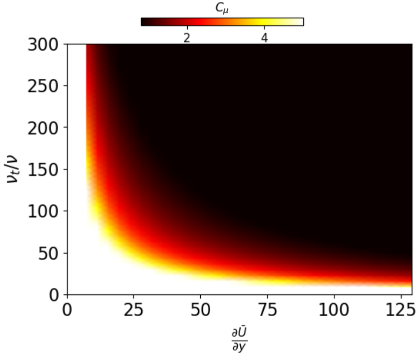
- Below I list some problems you're likely to encounter
- The figure below show the predicted C_ω (computed as $|\overline{v_1'v_2'}|_{SVR}/(\partial\bar{v}_1/\partial x_2)/\nu_t$ where $\partial\bar{v}_1/\partial x_2$ and ν_t cover the entire parameter space).



- When small $\partial\bar{v}_1/\partial x_2$ and ν_t are used (lower left corner), large C_ω are predicted.
- This may occur at the beginning of the CFD simulation and the solution is "stuck" (the product $\nu_t\partial\bar{v}_1/\partial x_2$ is correct but both ν_t and $\partial\bar{v}_1/\partial x_2$ are incorrect)

ML model: Problem 1

- Below I list some problems you're likely to encounter
- The figure below show the predicted C_ω (computed as $|\overline{v_1' v_2'}|_{SVR}/(\partial \bar{v}_1/\partial x_2)/\nu_t$ where $\partial \bar{v}_1/\partial x_2$ and ν_t cover the entire parameter space).



- When small $\partial \bar{v}_1/\partial x_2$ and ν_t are used (lower left corner), large C_ω are predicted.
- This may occur at the beginning of the CFD simulation and the solution is "stuck" (the product $\nu_t \partial \bar{v}_1/\partial x_2$ is correct but both ν_t and $\partial \bar{v}_1/\partial x_2$ are incorrect)
- The reason is poor/incorrect initial b.c. You can, e.g., use the profiles of a solution without ML/SVR as initial b.c.

Problem 1 how to plot the figure at the previous slide

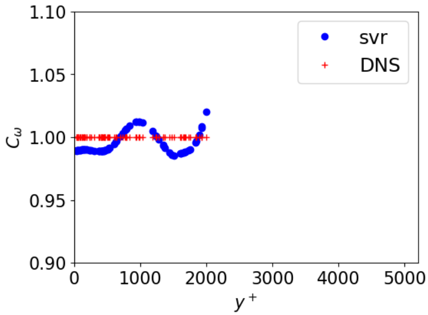
```
fig1,ax1 = plt.subplots();ax=plt.gca()
# Set Increments between points in a meshgrid
mesh_size = 0.05
# Identify min and max values for input variables
x_min, x_max = X[:,0].min(), X[:,0].max()
y_min, y_max = X[:,1].min(), X[:,1].max()
# Return evenly spaced values based on a range between min and max
xrange = np.arange(x_min, x_max, mesh_size)
yrange = np.arange(y_min, y_max, mesh_size)
# Create a meshgrid
xx, yy= np.meshgrid(xrange, yrange)
# Use model to create a prediction plane --- SVR
pred_svr = model.predict(np.c_[xx.ravel(), yy.ravel()])
pred_svr = pred_svr.reshape(xx.shape)
xx_no_scale=scaler_dudy.inverse_transform(xx)
yy_no_scale=scaler_vist.inverse_transform(yy)
# Make the color plot (excl. fig1,ax1 = plt.subplots() ...)
ax_plot=plt.pcolormesh(xx_no_scale,yy_no_scale/viscos, pred_svr, vmin=0.8,vmax
    =1,cmap=plt.get_cmap('hot'),shading='gouraud')
```

ML model: Problem 2

- Make sure that predicted C_ω agrees well with the target (that is not that case in the figure to the right)

ML model: Problem 2

- Make sure that predicted C_ω agrees well with the target (that is not that case in the figure to the right)



ML model number 2

- Now you should improve the mixing length model in which the turbulent viscosity is expressed as $\nu_t = f_m L_m^2 \left| \frac{\partial \bar{v}_1}{\partial x_2} \right|$.

ML model number 2

- Now you should improve the mixing length model in which the turbulent viscosity is expressed as $\nu_t = f_m L_m^2 \left| \frac{\partial \bar{v}_1}{\partial x_2} \right|$.
- We take the mixing length L_m as the wall distance and then let our function f_m adapt to get a reasonable lengthscale.

ML model number 2

- Now you should improve the mixing length model in which the turbulent viscosity is expressed as $\nu_t = f_m L_m^2 \left|\dfrac{\partial \bar{v}_1}{\partial x_2}\right|$.
- We take the mixing length L_m as the wall distance and then let our function f_m adapt to get a reasonable lengthscale.
- Use f_m as output and $|\partial \bar{v}_1/\partial x_2|$ and L_m^2 as input.

ML model number 2

- Now you should improve the mixing length model in which the turbulent viscosity is expressed as $\nu_t = f_m L_m^2 \left|\frac{\partial \bar{v}_1}{\partial x_2}\right|$.
- We take the mixing length L_m as the wall distance and then let our function f_m adapt to get a reasonable lengthscale.
- Use f_m as output and $|\partial \bar{v}_1/\partial x_2|$ and L_m^2 as input.
- You could also try ν_t as output instead of f_m.

References

[1] L. Davidson. Using Machine Learning for formulating new wall functions for Large Eddy Simulation: A second attempt. Technical report, Division of Fluid Dynamics, Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology, Gothenburg, 2022.

[2] Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, and Thomas Schön. *Machine Learning: A First Course for Engineers and Scientists*. Cambridge University Press, 2022.

[3] Menneni Rachana, Jegadeesan Ramalingam, Gajula Ramana, Adigoppula Tejaswi, Sagar Mamidala, and G Srikanth. Fraud detection of credit card using machine learning. *GIS-Zeitschrift für Geoinformatik*, 8:1421–1436, 10 2021.

[4] Sudarshana S Rao and Santosh R Desai. Machine learning based traffic light detection and ir sensor based proximity sensing for autonomous cars. In *Proceedings of the International Conference on IoT Based Control Networks & Intelligent Systems – ICICNIS*, 2021.