Plots of input features

Features were proposed by Wang et al. (Wang, Wu, & Xiao, 2017)

The features are normalized according to:

$$q_\beta = \frac{\hat{q}_\beta}{|\hat{q}_\beta| + |q^*_\beta|} \tag{1}$$

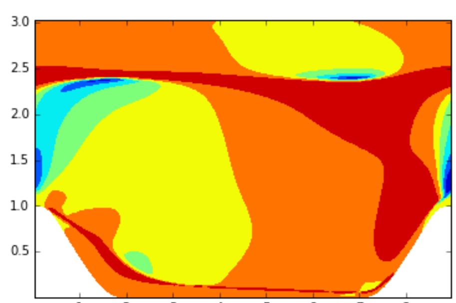
RANS data from openFOAM simulation (Re=700 and kOmega) with end-time 3000.

1. Ratio of excess rotation rate to strain rate

$$\hat{q}_\beta = \frac{1}{2}(||\Omega||^2 + ||S||^2)$$

$$q^*_\beta = ||S||^2$$

```
def q1(S, Omega):
    a = np.shape(S)
    q1 = np.zeros((a[2],a[3]))
    for i1 in range(a[2]):
        for i2 in range(a[3]):
            raw = 0.5*(np.abs(np.trace(np.dot(S[:,:,i1,i2],S[:,:,i1,i2]))) - np.abs(np.trace(np.dot(Omega[:,:,i1,i2],-1*(Omega[:,:,i1,i2])))))
            norm = np.trace(np.dot(S[:,:,i1,i2],S[:,:,i1,i2]))
            q1[i1,i2] = raw/(np.abs(raw) + np.abs(norm))
    return q1
```

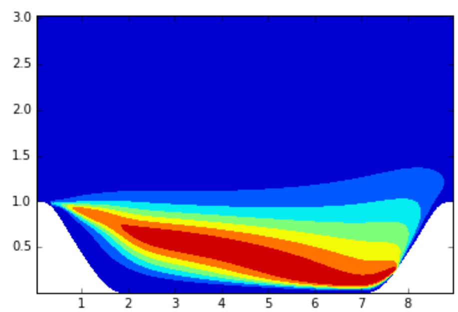


2. Turbulence intensity

$$\hat{q}_\beta = k$$

$$q^*_\beta = \frac{1}{2} U_i U_i$$

```
def q2(k, U):
    a = np.shape(k)
    b= np.shape(U)
    q2 = np.zeros((a[1],a[2]))
    for i1 in range(a[1]):
        for i2 in range(a[2]):
            raw = k[0,i1,i2]
            norm = 0.5*(np.inner(U[:, i1, i2], U[:, i1, i2])) # inner is equivalent to sum UiUi
            q2[i1,i2] = raw/(np.abs(raw) + np.abs(norm))
    return q2
```



3. Wall-distance based on Reynolds number

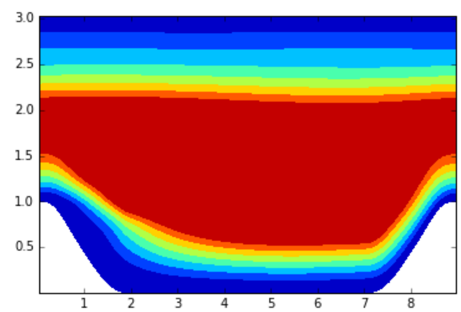
$$q_\beta = min(\frac{\sqrt{k}d}{50\nu}, 2)$$

Normalization is not necessary since this feature is already non dimensional.

```
nu=1.4285714285714286e-03

def q3(k, yWall, nu):
    a = np.shape(k)
```

```python
    q3 = np.zeros((a[1],a[2]))
    for i1 in range(a[1]):
        for i2 in range(a[2]):
            q3[i1,i2] = np.minimum((np.sqrt(k[:,i1,i2][0])*yWall[:, i1, i2])/(50*nu), 2)
    return q3
```



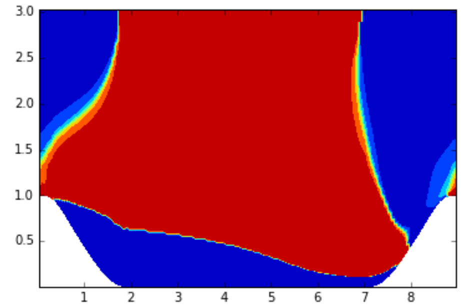
4. Pressure gradient along streamline

$$\hat{q}_\beta = U_i \frac{\partial P}{\partial x_i}$$

$$q^*_\beta = \sqrt{\frac{\partial P}{\partial x_j}\frac{\partial P}{\partial x_j} U_i U_i}$$

```python
def q4(U, gradP):
    a = np.shape(gradP)
    q4 = np.zeros((a[1],a[2]))
    for i1 in range(a[1]):
        for i2 in range(a[2]):
            raw  = np.einsum('k,k', U[:,i1,i2], gradP[:,i1,i2])
            norm = np.einsum('j,j,i,i', gradP[:,i1,i2], gradP[:,i1,i2], U[:, i1, i2],U[:, i1, i2])

            q4[i1,i2] = raw / (np.fabs(norm) + np.fabs(raw));
    return q4
```

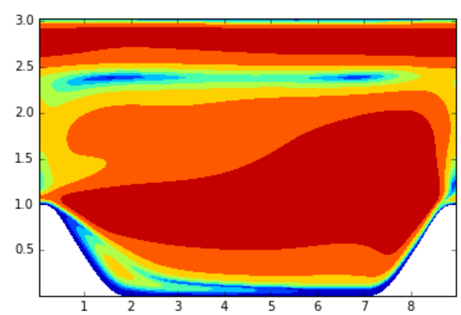


5. Ratio of turbulent time scale to mean strain time scale

$$\hat{q}_\beta = \frac{k}{\epsilon}$$

$$q_\beta^* = \frac{1}{||S||}$$

```
Cmu=0.09
def q5(k, S, Cmu, omega):
    a = np.shape(k_RANS)
    q5 = np.zeros((a[1],a[2]))
    for i1 in range(a[1]):
        for i2 in range(a[2]):
            epsilon = Cmu * k[:, i1, i2] * omega[:,
        i1, i2]
            raw = k[:, i1, i2] / epsilon
            norm = 1 / np.sqrt(np.trace(np.dot(S[:,
        :, i1, i2],S[:, :, i1, i2])))
            q5[i1,i2] = raw/(np.fabs(raw) + np.fabs(
        norm))
    return q5
```



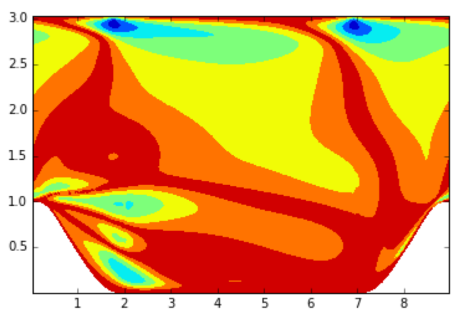
6. Cratio of pressure normal stresses to shear stresses

$$\hat{q}_\beta = \sqrt{\frac{\partial p}{x_i}\frac{\partial p}{\partial x_i}}$$

$$q^*_\beta = \frac{1}{2}\rho(\frac{\partial U_k}{\partial x_k})^2$$

```python
def q6(gradP, gradU, p, U):
    a = np.shape(gradP)
    q6 = np.zeros((a[1],a[2]))
    for i1 in range(a[1]):
        for i2 in range(a[2]):
            raw  = np.sqrt(np.einsum('i,i', gradP[:, i1, i2], gradP[:, i1, i2]))
            norm = np.einsum('k,kk', U[:, i1, i2], gradU[:, :, i1, i2])

            norm *= 0.5 * p[0, i1, i2]
            q6[i1,i2] = raw/(np.fabs(raw) + np.fabs(norm))
    return q6
```

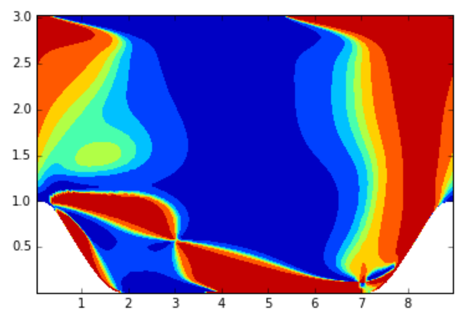


7. Non-orthogonality between velocity and its gradient

$$\hat{q}_\beta = |U_i U_j \frac{\partial U_i}{\partial x_j}|$$

$$q^*_\beta = \sqrt{U_l U_l U_i \frac{\partial U_i}{\partial x_j} U_k \frac{\partial U_k}{\partial x_j}}$$

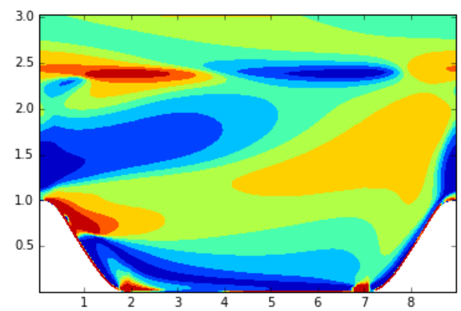
```python
def q7(U, gradU):
    a = np.shape(U)
    q7 = np.zeros((a[1],a[2]))
    for i1 in range(a[1]):
        for i2 in range(a[2]):
            raw = np.fabs(np.einsum('i, j, ij', U[:, i1, i2], U[:, i1, i2], gradU[:, :, i1, i2]))
            norm = np.sqrt(np.einsum('l, l, i, ij, k, kj', U[:, i1, i2], U[:, i1, i2],U[:, i1, i2], gradU[:, :, i1, i2], U[:, i1, i2], gradU[:, :, i1, i2]))
            q7[i1,i2] = raw/(np.fabs(raw) + np.fabs(norm))
    return q7
```



8. Ratio of convection to production of TKE

$$\hat{q}_\beta = U_i \frac{dk}{dx_i}$$
$$q^*_\beta = |\overline{u'_i u'_j} S_{jk}|$$

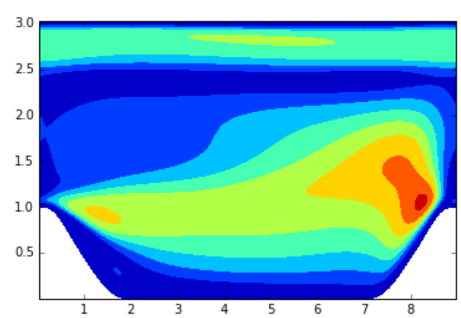
```python
def q8(U, gradK, Tau, S):
    a = np.shape(U)
    q8 = np.zeros((a[1],a[2]))
    for i1 in range(a[1]):
        for i2 in range(a[2]):
            raw  = np.einsum('i,i',U[:,i1,i2], gradK[:,i1,i2])
            norm = np.einsum('jk,jk', Tau[:,:, i1, i2], S[:,:, i1, i2])
            q8[i1,i2] = raw/(np.fabs(raw) + np.fabs(norm))
    return q8
```



9. Ratio of total to normal Reynolds stresses

$$\hat{q}_\beta = ||\overline{u_i' u_j'}||$$
$$q_\beta^* = k$$

```
def q9(tau, k):
    a = np.shape(k)
    q9 = np.zeros((a[1],a[2]))
    for i1 in range(a[1]):
        for i2 in range(a[2]):
            raw = np.sqrt(np.trace(np.dot(tau[:, :, i1, i2],np.transpose(tau[:, :, i1, i2]))))
            norm = k[:, i1, i2]
            q9[i1,i2] = raw/(np.fabs(raw) + np.fabs(norm))
    return q9
```



References

Wang, J.-X., Wu, J.-L., & Xiao, H. (2017). Physics-informed machine learning approach for reconstructing reynolds stress modeling discrepancies based on dns data. *Physical Review Fluids*, *2*(3), 034603.