Introduction to computational thinking and programming for CFD (13251)

Dr. rer. nat. Marten Klein

Chair of Numerical Fluid and Gas Dynamics, BTU Cottbus-Senftenberg

Sheet 3

Goals

- Matrices, 2-D arrays, and n-D arrays
- Structured grids and algebraic grid generation
- Data output and data input
- Interpolation on structured and unstructured meshes

Basic recipes for n-D array

Declare a 3-D ($N_1 \times N_2 \times N_3$) array and initialize it with zeros

```
matrix = np.zeros((N1, N2, N3))
```

Read a specific element form the matrix, here $i = 2$, $j = 3$, $k = 1$

```
element = matrix[2,3,1]
```

Read a 1-D slice along the i direction for fixed $j = 0$ and $k = 0$

```
slice  = matrix[:,0,0]
```

Overwrite a single element (here $i = 2$, $j = 3$, $k = 1$) of the 3-D array with the value 99

```
matrix[2,3,1] = 99.
```

Overwrite a 1-D slice along i ($j = 0$, $k = 0$) with linearly increasing values from $-\pi$ to $+\pi$

```
matrix[:,0,0] = np.linspace( -np.pi, np.pi, len(matrix[:,0,0]) )
```

Reshape a 1-D vector to a 2-D (3×9) matrix

```
vector = np.arange(3*9)
matrix = vector.reshape((3, 9))
```

Tasks

1. Consider the 2×3 matrix $\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} = \begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{pmatrix}$.

   (a) Store $\boldsymbol{A}$ in a 2-D array. Confirm the storage scheme by printing $\boldsymbol{A}$.

   (b) Overwrite the element a_{23} with the value 99. Confirm the result by printing $\boldsymbol{A}$.

   (c) What happens if you try to print the (nonexisting) element a_{55}?

2. Algebraic grid generation. A mesh is needed for a 2-D channel.

   - Domain size (in meters): $0 \le x \le 4$ $\quad -1 \le y \le 1$ $\quad$ (walls at $y = \pm 1$)
   - Number of cells: $N_x = 50$ $\quad N_y = 30$

   (a) Generate an **equidistant 2-D grid** for the (x, y) plane.
   Hint: Consider using the `meshgrid` command from the `numpy` module.
   https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.html

   (b) Visualize the grid by plotting all grid vertices $\boldsymbol{x}_{ij} = (x_{ij}, y_{ij})$ as 2-D scatter plot.
   https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html

   (c) Modify your program so that the number of cells can be provided as input on runtime.

   (d) Store the grid into a *comma separated variables* (CSV) file using `savetxt`.

```
np.savetxt('grid_x.csv', X, delimiter=',')  # all X coordinates
np.savetxt('grid_y.csv', Y, delimiter=',')  # all Y coordinates
```

3. Grid stretching. Vinokur (NASA Contractor Report 3313, 1980, p. 14 and Appendix A – see moodle) developed an analytical 1-D stretching. By application to the y coordinate of the 2-D channel, as shown in the lecture, clustering of grid points toward $y = +1$ and $y = -1$ is readily achieved with the following **stretching function**,

$$y(\xi) = \frac{\tanh\big(b \cdot (\xi - 1/2)\big)}{\tanh\big(b/2\big)},$$

   where

$$b = \frac{1}{2} \ln\left(\frac{1+a}{1-a}\right), \qquad a = 0.99, \qquad \xi_j = \frac{j}{N_y} \quad \text{for} \quad j = 0, 1, \ \ldots \ , N_y.$$

   (a) Implement grid stretching for the y coordinate in your grid generator.
   Hint: Implementations of tanh and ln (named log) are available in `numpy`.

   (b) Visualize the stretched grid by a 2-D scatter plot.

   (c) Now read the equidistant grid from file using `loadtxt` and plot it together with the stretched grid.
   https://www.sharpsightlabs.com/blog/numpy-loadtxt/

4. Interpolation and gridded data. Consider the 2-D spatial distribution of a scalar property that is described by the function $z = f(x, y)$ over the unit square $(x, y) \in [0, 1] \times [0, 1]$.

   - $f(x, y) = \sin(2\pi x)\cos(8\pi y)\exp(-4y^2)$
   - High-resolution Cartesian grid with 40,000 vertices ($N_x = 200$, $N_y = 200$)
   - Low-resolution unstructured mesh with 100 randomly distributed nodes

```
from numpy import random as rnd
points = rnd.random_sample((100, 2))  # sample 100 pairs (x,y)
```

   (a) Implement $f(x, y)$ in a Python function.

   (b) Generate the high-resolution equidistant Cartesian grid.

   (c) Evaluate $z = f(x, y)$ on the high-resolution grid. Visualize the 2-D distribution by filled contours using 256 levels.

```
import matplotlib.pyplot as plt
plt.contourf( X, Y, Z, 256 )
```

   (d) Plot the points of the low-resolution mesh as scatter plot of black bullets (•) on top of the contours.

   (e) Now evaluate $z = f(x, y)$ for the low-resolution grid.

   (f) Interpolate the low-resolution data to the high-resolution grid and plot the interpolated data. Plot the interpolated data in another figure. Also use filled contours.

```
from scipy.interpolate import griddata
Z_interp = griddata( (X_low, Y_low), Z_low, (X, Y))
```

   (g) Select another **interpolation method** by adding the keyword argument `method=...` to the function call. See the link for available alternatives. https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.griddata.html