Data visualization course

Laboratory work 1

Drawing tensor fields via Matplotlib

# Scalar fields visualisation

A scalar field (scalar function) on a finite space ***V*** is a function that corresponds to each point in some region of this space (the domain of definition) with a scalar, i.e. a real or complex number. With a fixed basis of space, a scalar field can be defined as a function of several variables on the coordinates of a point.

The difference between a numerical function of several variables and a scalar field is that in a different basis, the scalar field as a function of coordinates changes so that if a new set of arguments represents the same point in space in the new basis, the value of the scalar function does not change.

For example, if in some orthonormalized basis of a two-dimensional vector space a scalar function has the form  then in another basis rotated 45 degrees to this one, the same function in the new coordinates will have the form .

Most often, we consider scalar functions that are continuous or differentiable (smooth) a sufficient number of times. In practice, they are found to be widely used:

* A function of three variables:  (a scalar field in three-dimensional space).
* A function of two variables:  (a scalar field in two-dimensional space).

Examples of scalar fields in three dimensions are:

* temperature (if it is different in different points of space);
* electrostatic potential;
* potential in the Newtonian theory of gravitation;
* pressure field in a liquid medium.

Examples of flat (two-dimensional) scalar fields:

* sea depth marked in any way on a flat map;
* charge density on a flat surface of a conductor.

One of the important characteristics of a scalar field is its gradient. The gradient of a scalar field is a vector field with components:

 (1)

**Example**. Plot the visualization of a scalar field *u(x,y)* = sin(*x*2+*y*2),   
x ∈ [-3;3]; y ∈ [-3;3]. The implementation is presented in Listing 3.1.

**import** numpy **as** np

**from** matplotlib **import** pyplot **as** plt

n **=** 256

x **=** np**.**linspace**(-**3.**,** 3.**,** n**)** # X range

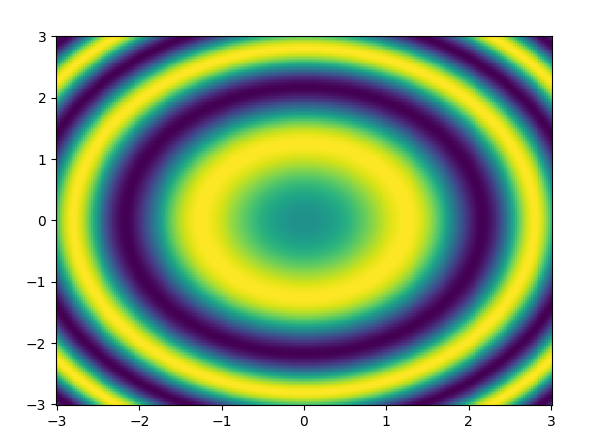
y **=** np**.**linspace**(-**3.**,** 3.**,** n**)** # Y range

X**,** Y **=** np**.**meshgrid**(**x**,** y**)**

Z **=** np**.**sin**(**X**\*\***2**+**Y**\*\***2**)** # scalar field equation

plt**.**pcolormesh**(**X**,** Y**,** Z**)**

plt**.**show**()**

Listing 3.1 - Example code for visualizing a scalar field

# Figure 3.1 - Visualization of a scalar field

# Vector field visualization

A vector field is a vector-valued function - a mapping that corresponds to each point of a given space with a vector. In the case of a Euclidean (finite-dimensional vector space with a scalar product) space, the concept of a vector field becomes clear, and the vector field is interpreted as a way to specify the movements of a certain dynamic system: the vector at a given point describes the direction and speed of the point along the phase curve. For the Cartesian coordinate system, the field can be represented as .

**Example.** Create a visualization of a flat vector field

, *x, y* ∈ [-5; 5]; (2)

**import** numpy **as** np

**import** matplotlib**.**pyplot **as** plt

**def** u**(**x**,** y**):**

**return** **-**y **/** np**.**sqrt**(**x **\*\*** 2 **+** y **\*\*** 2**)**

**def** v**(**x**,** y**):**

**return** x **/** np**.**sqrt**(**x **\*\*** 2 **+** y **\*\*** 2**)**

xx**,** yy **=** np**.**meshgrid**(**np**.**linspace**(-**5**,** 5**,** 10**),**

np**.**linspace**(-**5**,** 5**,** 10**))**

u\_val **=** u**(**xx**,** yy**)**

v\_val **=** v**(**xx**,** yy**)**

plt**.**quiver**(**xx**,** yy**,** u\_val**,** v\_val**)**

plt**.**show**()**

plt**.**streamplot**(**xx**,** yy**,** u\_val**,** v\_val**)**

plt**.**show**()**

Listing 3.2 - Example code for visualizing a vector field

|  |  |
| --- | --- |
| png |  |
| а | б |

Figure 3.2 - Visualization of a vector field: a - vectors; b - flow lines

It is also quite easy to construct current lines by finding the trajectory of a massless point by performing integration according to the following principle (see Listing 3.3):

 (3)

The result of building the stream lines can be seen in Fig. 3.3.

**import** numpy **as** np

**import** matplotlib**.**pyplot **as** plt

**def** u**(**x**,** y**):**

**return** **-**y **/** np**.**sqrt**(**x **\*\*** 2 **+** y **\*\*** 2**)**

**def** v**(**x**,** y**):**

**return** x **/** np**.**sqrt**(**x **\*\*** 2 **+** y **\*\*** 2**)**

**def** create\_stream\_line**(**x0**,** y0**,** u**,** v**,** t0**=**0**,** t1**=**10**,** dt**=**0.001**):**

t **=** np**.**arange**(**t0**,** t1**,** dt**)**

xx\_new **=** np**.**zeros\_like**(**t**)**

yy\_new **=** np**.**zeros\_like**(**t**)**

xx\_new**[**0**]** **=** x0

yy\_new**[**0**]** **=** y0

**for** i **in** range**(**1**,** t**.**size**):**

xx\_new**[**i**]** **=** x0 **+** u**(**x0**,** y0**)** **\*** dt

yy\_new**[**i**]** **=** y0 **+** v**(**x0**,** y0**)** **\*** dt

x0**,** y0 **=** xx\_new**[**i**],** yy\_new**[**i**]**

**return** xx\_new**,** yy\_new

**for** i **in** range**(**1**,** 10**):**

x1**,** y1 **=** create\_stream\_line**(**i**,** 0**,** u**,** v**)**

plt**.**plot**(**x1**,** y1**)**

plt**.**show**()**

Listing 3.3 - Example code for calculating and visualizing stream lines

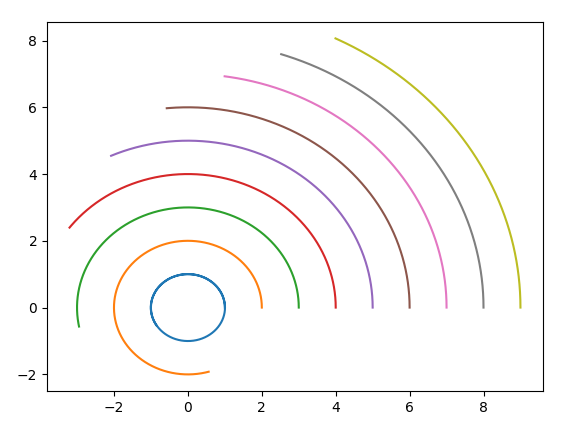


Figure 3.3 - Visualization of current lines

**Example.** Build a visualization of a vector field

, x, y, z ∈ [-0.8; 0.8]; (4)

**from** mpl\_toolkits**.**mplot3d **import** axes3d

**import** matplotlib**.**pyplot **as** plt

**import** numpy **as** np

ax **=** plt**.**figure**().**add\_subplot**(**projection**=**'3d'**)**

x**,** y**,** z **=** np**.**meshgrid**(**np**.**arange**(-**0.8**,** 1**,** 0.2**),**

np**.**arange**(-**0.8**,** 1**,** 0.2**),**

np**.**arange**(-**0.8**,** 1**,** 0.8**))**

u **=** np**.**sin**(**np**.**pi **\*** x**)** **\*** np**.**cos**(**np**.**pi **\*** y**)** **\*** np**.**cos**(**np**.**pi **\*** z**)**

v **=** **-**np**.**cos**(**np**.**pi **\*** x**)** **\*** np**.**sin**(**np**.**pi **\*** y**)** **\*** np**.**cos**(**np**.**pi **\*** z**)**

w **=** **(**np**.**sqrt**(**2.0 **/** 3.0**)** **\*** np**.**cos**(**np**.**pi **\*** x**)** **\*** np**.**cos**(**np**.**pi **\*** y**)** **\*** np**.**sin**(**np**.**pi **\*** z**))**

ax**.**quiver**(**x**,** y**,** z**,** u**,** v**,** w**,** length**=**0.2**,** color **=** 'black'**)**

plt**.**show**()**

Listing 3.4 - Visualization of a three-dimensional vector field

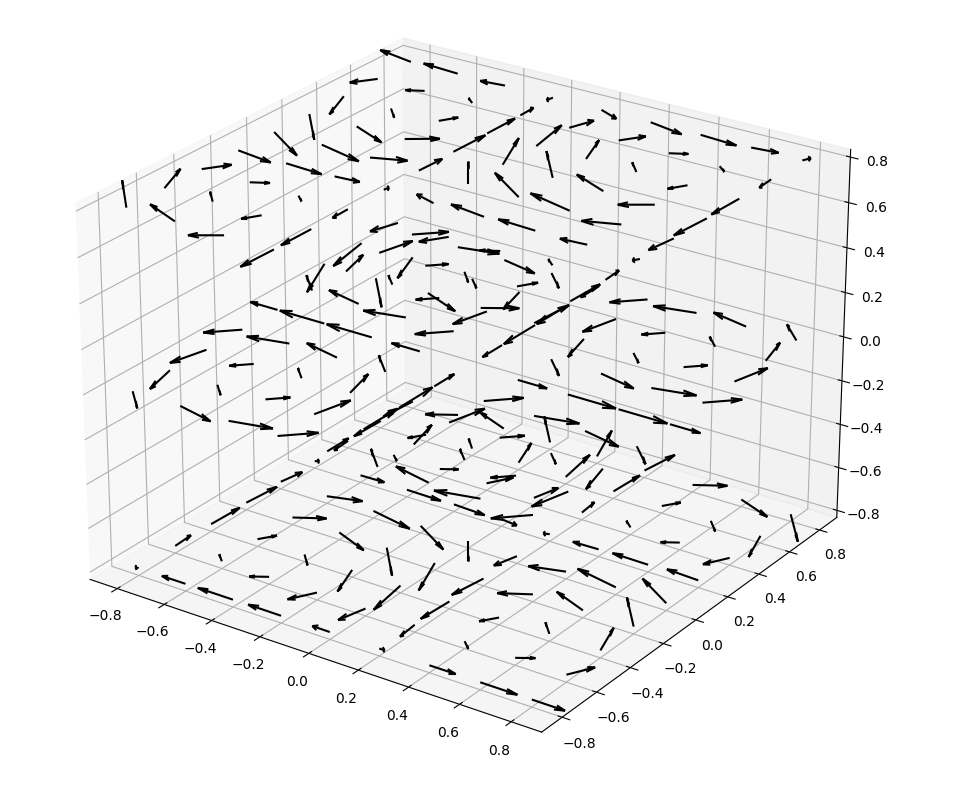


Figure 3.4 - Visualization of a three-dimensional vector field

# Visualization of tensor fields

A tensor field, by analogy with a vector field, is a mapping that corresponds to a tensor at each point in space. The simplest tensor is a tensor of rank 2 and it is a "combination" of two vectors into one object. Since a vector in 3-dimensional space has 3 components, a tensor of rank 2 contains 3×3 elements and is represented by a corresponding matrix. If the tensor is symmetric, then only 6 of its 9 components are independent. One of the most common tensors is the mechanical stress tensor (Fig. 3.5):

 (5)

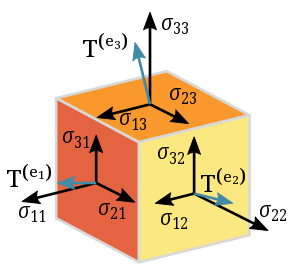


Figure 3.5 - Stress tensor components **σ**(Original image from <https://uk.wikipedia.org/wiki/Тензор>)

To build images (glyphs), it is assumed that the tensor can be represented in the form:

, (6)

where R – is the matrix of rotation of coordinates to the basis that coincides with the main directions; Λ = (λ1, λ2, λ3)– tensor’s eigenvalues. More details about the construction of glyphs can be found in the article by G. Kindelmann (Kindlmann G. Superquadric tensor glyphs // Proceedings of the Sixth Joint Eurographics-IEEE TCVG conference on Visualization. – 2004. – С. 147-154. <https://people.cs.uchicago.edu/~glk/pubs/pdf/Kindlmann-Superquad1-VisSym-2004.pdf>).

|  |  |
| --- | --- |
|  | (7) |

The sum of three metrics *cl*, *cp*, *cs* equals one and determines the barycentric parameterization of a triangular region with extreme values of linear, planar, and spherical shapes in the three corners.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| a | b | c |

Figure 3.6 - Visualization of the stress tensor at different values *cl*, *cp*, *cs:*a - cubes, b - cylinders, c - ellipsoids

To implement this idea, the Department of Dynamics and Strength of Machines of NTU "KhPI" developed the glyph\_visualization\_lib library (authors Ruslan Babudzhan, Oleksiy Vodka). This library supports two options for visualization:

* Using matplotlib (slower, but does not require additional dependencies);
* Using mayavi (faster option).

To install mayavi, you need to install the mayavi and pyqt5 packages.

pip install mayavi pyqt5

**Example.** Create a visualization of the tensor field:

 (8)

The code that performs the visualization is shown in Listing 3.5. For the code to work, you need to download the glyph\_visualization\_lib.py file and add it to the project directory. The result of visualization is shown in Fig. 3.7.

**import** numpy **as** np

**import** matplotlib**.**pyplot **as** plt

**from** mayavi **import** mlab

**import** glyph\_visualization\_lib **as** gvl

**def** main**():**

x **=** np**.**linspace**(**0**,** np**.**pi**,** 8**,** dtype**=**float**,** endpoint**=True)**

y **=** np**.**linspace**(-**np**.**pi**,** **-**1**,** 8**,** dtype**=**float**,** endpoint**=True)**

z **=** np**.**linspace**(**0**,** 2 **\*** np**.**pi**,** 8**,** dtype**=**float**,** endpoint**=True)**

X**,** Y**,** Z **=** np**.**meshgrid**(**x**,** y**,** z**)**

stress\_tensor **=** np**.**array**([**

**[-**np**.**sin**(**X**),** 5 **\*** np**.**sin**(**X **+** Y**),** 10 **\*** np**.**sin**(**X **\*** Z**)],**

**[**5 **\*** np**.**sin**(**X **+** Y**),** **-**np**.**sin**(**Y**),** 3 **\*** np**.**sin**(**Y **+** Z**)],**

**[**10 **\*** np**.**sin**(**X **\*** Z**),** 3 **\*** np**.**sin**(**Y **+** Z**),** np**.**cos**(**Z**)]**

**])**

**print(**stress\_tensor**.**shape**)**

vm\_stress **=** gvl**.**get\_von\_Mises\_stress**(**stress\_tensor**)**

glyph\_radius **=** 0.25

limits **=** **[**np**.**min**(**vm\_stress**),** np**.**max**(**vm\_stress**)]**

colormap **=** plt**.**get\_cmap**(**'rainbow'**,** 120**)**

fig **=** mlab**.**figure**(**bgcolor**=(**1**,** 1**,** 1**))**

fig2 **=** plt**.**figure**()**

ax **=** fig2**.**add\_subplot**(**111**,** projection**=**'3d'**)**

**for** i **in** range**(**x**.**size**):**

**for** j **in** range**(**y**.**size**):**

**for** k **in** range**(**z**.**size**):**

center **=** **[**x**[**i**],** y**[**j**],** z**[**k**]]**

data **=** stress\_tensor**[:,** **:,** i**,** j**,** k**]**

color **=** colormap**(**gvl**.**get\_colormap\_ratio\_on\_stress**(**vm\_stress**[**i**,** j**,** k**],** limits**))[:**3**]**

"""

glyph\_type = {0: 'cuboid', 1: 'cylinder', 2: 'ellipsoid', 3: 'superquadric'}

if glyph\_type == 3 (superquadric)

there are glyph shape type

0 - superquadrics,

1 - Kindlmann\_glyph,

2 - Kindlmann\_modified\_glyph

"""

x\_g**,** y\_g**,** z\_g **=** gvl**.**get\_glyph\_data**(**center**,** data**,** limits**,** glyph\_points**=**12**,** glyph\_radius**=**glyph\_radius**,**

glyph\_type**=**3**,**

superquadrics\_option**=**2**)**

# surf = ax.plot\_surface(x\_g, y\_g, z\_g, linewidth=0, antialiased=True, color=color)

# surf = ax.plot\_wireframe(x\_g, y\_g, z\_g, linewidth=1, antialiased=True, color=color)

mlab**.**mesh**(**x\_g**,** y\_g**,** z\_g**,** color**=**color**)**

mlab**.**move**(**forward**=**1.8**)**

mlab**.**savefig**(**"superquadric-Kindlmann\_modified-viz.png"**,** size**=(**100**,** 100**))**

mlab**.**show**()**

# plt.show()

**pass**

**if** \_\_name\_\_ **==** '\_\_main\_\_'**:**

main**()**

Listing 3.5 - Visualization of a tensor field

|  |  |  |
| --- | --- | --- |
| C:\Users\alex\AppData\Local\Microsoft\Windows\INetCache\Content.Word\cuboid-viz.png | C:\Users\alex\AppData\Local\Microsoft\Windows\INetCache\Content.Word\cylinder-viz.png | C:\Users\alex\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ellipsoid-viz.png |
| a. cuboid | b. cylinder | c. ellipsoid |
| C:\Users\alex\AppData\Local\Microsoft\Windows\INetCache\Content.Word\superquadric-superquadrics-viz.png | C:\Users\alex\AppData\Local\Microsoft\Windows\INetCache\Content.Word\superquadric-Kindlmann-viz.png | C:\Users\alex\AppData\Local\Microsoft\Windows\INetCache\Content.Word\superquadric-Kindlmann_modified-viz.png |
| d. superquadrature | e. Kindelman superquadrature | f. modified Kindelman superquadrature |

Figure 3.7 - Visualization of a tensor field

**Task.** Build according to your number in group list:

1. Visualize the scalar field. Find its gradient and visualize it as a flat vector field;
2. Visualize a plane vector field both with the help of vectors and current lines from the matplotlib library and with the code from the listing;
3. Construct a three-dimensional visualization of the vector field; For an additional point (optional), modernize the algorithm for constructing current lines in the case of a 3-dimensional field.
4. Construct a visualization of a tensor field using ellipsoids, cuboids, cylinders, and any superquadrant.

**Student 1.**

1. 

2. 

3. 

4. 

**Student 2.**

1. 

2. 

3. 

4. 

**Student 3.**

1. 

2. 

3. 

4. 

**Student 4.**

1. 

2. 

3. 

4. 

**Student 5.**

1. 

2. 

3. 

4. 

**Student 6.**

1. 

2. 

3. 

4. 

**Student 7.**

1. 

2. 

3. 

4. 

**Student 8.**

1. 

2. 

3. 

4. 

**Student 9.**

1. 

2. 

3. 

4. 