Plot 3d surface with colormap as 4th dimension, function of x,y,z

Asked 7 years, 11 months ago Modified 4 months ago Viewed 33k times

19

I'm trying to plot a 3d surface where each of the three dimensions in a separate array of values and the colouring of the surface at each coordinate is a function of x,y,z. A sort of numpy.pcolormesh but in 4D, rather than 3D. The 3D plot is given by:

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
fig = plt.figure()
ax = fig.gca(projection='3d')
x = np.logspace(-1.,np.log10(5),50)
y = np.linspace(6,9,50)
z = np.linspace(-1,1,50)
colors = LikeBeta(y,range(50),range(50))
ax.plot_trisurf(x,y,z,cmap=colors,linewidth=0.2)
```

where

```
def LikeBeta(rho0,r0,beta):
    M0 = 10**rho0*r0_array[r0]**3
    I = cst*M0*sigma_los_beta[beta,:,r0]
    S = dv**2+I
    res = (np.log(S) + (v-u)**2/S).sum()
    return res/2.
```

Probably the `cmap=colors` is wrong, but the problem lies elsewhere. I get the following error:

```
----> 8 colors = LikeBeta(y,range(50),range(50))
----> 4     I = cst*M0*sigma_los_beta[beta,:,r0]
    ValueError: operands could not be broadcast together with shapes (50,) (50,353)
```

Indeed `sigma_los_beta` is an array that I evaluate separately and has shape `(50,353,50)` and those 353 are data that I must have.

How can I cast this function into a form that is compatible with the other entries of `plot_trisurf` ?

Sorry, but I can't supply a minimal working code, because dv,v and u are data. Thank you very much for your help. Cheers

python matplotlib matplotlib-3d Edit tags

Share Edit Follow Close Flag

edited Apr 26 at 1:25
Trenton McKinney
56.9k 33 143 158

asked Sep 8, 2015 at 15:15
andrea
525 1 5 21

You could try slicing perhaps? You have quite a few undefined variables within your sample code. It is difficult to help without understanding what everything is. – tnknepp Sep 8, 2015 at 15:56

4 Answers

Sorted by:
Reset to default

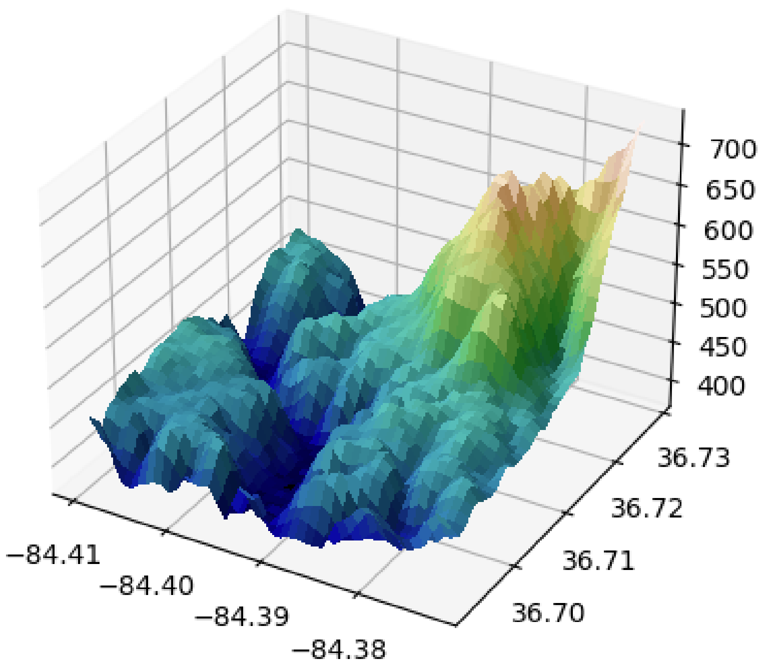
Date modified (newest first)

2

As of May 2022 the top three answers to this question each have various issues. I found the example provided in the matplotlib 3.5.0 documentation to be far simpler and actually work as expected to calculate `facecolors` with shading using the `LightSource` class.

Just override the specific `z` passed into `ls.shade`:

```
from matplotlib import cbook
from matplotlib import cm
from matplotlib.colors import LightSource
import matplotlib.pyplot as plt
import numpy as np

# Load and format data
dem = cbook.get_sample_data('jacksboro_fault_dem.npz', np_load=True)
z = dem['elevation']
nrows, ncols = z.shape
x = np.linspace(dem['xmin'], dem['xmax'], ncols)
y = np.linspace(dem['ymin'], dem['ymax'], nrows)
x, y = np.meshgrid(x, y)

region = np.s_[5:50, 5:50]
x, y, z = x[region], y[region], z[region]

# Set up plot
fig, ax = plt.subplots(subplot_kw=dict(projection='3d'))

ls = LightSource(270, 45)
# To use a custom hillshading mode, override the built-in shading and pass
# in the rgb colors of the shaded surface calculated from "shade".
rgb = ls.shade(z, cmap=cm.gist_earth, vert_exag=0.1, blend_mode='soft')
surf = ax.plot_surface(x, y, z, rstride=1, cstride=1, facecolors=rgb,
                       linewidth=0, antialiased=False, shade=False)

plt.show()
```



Share Edit Follow Flag

answered May 24, 2022 at 2:08

Teque5
414 7 7

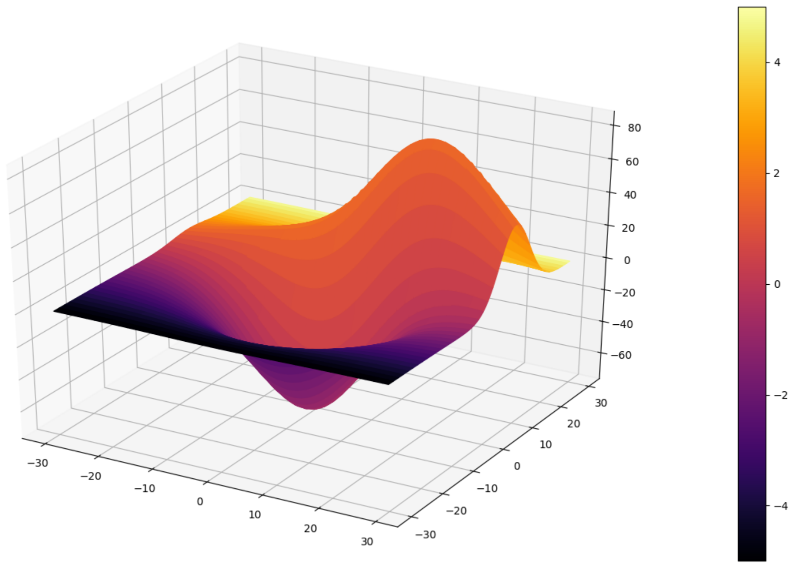
can you look at this question – A.E Nov 10, 2022 at 0:21

8

Many thanks to @Frik for his great answer, it helped me achieve a similar plot as requested by the OP.

However, I found that a few simplifications to the code may be done and could be of interest. Snippet and figure below.

```
import matplotlib.pyplot as plt
# This import registers the 3D projection, but is otherwise unused.
from mpl_toolkits.mplot3d import Axes3D  # noqa: F401 unused import
from mpl_toolkits.mplot3d.axes3d import get_test_data
import numpy as np
fig, ax = plt.subplots(subplot_kw={'projection': '3d'})
X, Y, Z = get_test_data(0.05)
C = np.linspace(-5, 5, Z.size).reshape(Z.shape)
scamap = plt.cm.ScalarMappable(cmap='inferno')
fcolors = scamap.to_rgba(C)
ax.plot_surface(X, Y, Z, facecolors=fcolors, cmap='inferno')
fig.colorbar(scamap)
plt.show()
```



Finally, I also wanted to comment on what @Frik wrote:

> The answer I referenced (and others) mentions that you should normalize your fourth dimension data. It seems that this may be avoided by explicitly setting the limits of the colormap as I did in the code sample.

I found this statement to be incorrect. Indeed, if one has a look at `to_rgba`, one can see that there is a `norm` keyword which is by default set to `True`. This is exactly where normalization occurs. The following statement is also included:

> If norm is False, no normalization of the input data is performed, and it is assumed to be in the range (0-1).

You indeed want your data to lie in (0-1).

Share Edit Follow Flag

answered Nov 25, 2019 at 3:54
Patol75
4,342 1 17 27

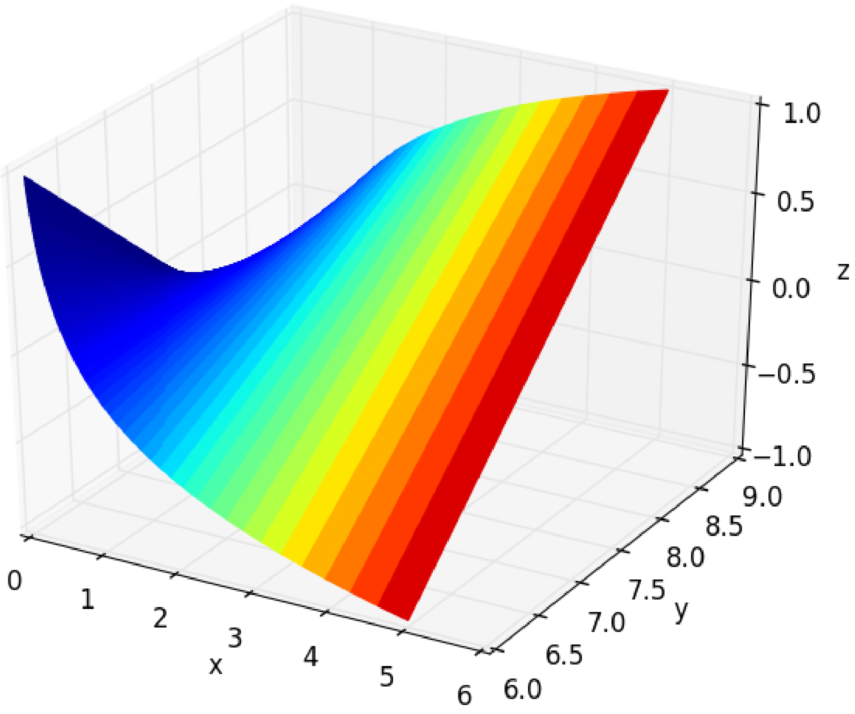
26

This answer addresses the 4d surface plot problem. It uses matplotlib's `plot_surface` function instead of `plot_trisurf`.

Basically you want to reshape your x, y and z variables into 2d arrays of the same dimension. To add the fourth dimension as a colormap, you must supply another 2d array of the same dimension as your axes variables.

Below is example code for a 3d plot with the colormap corresponding to the x values. The `facecolors` argument is used to alter the colormap to your liking. Note that its value is acquired from the `to_rgba()` function in the `matplotlib.cm.ScalarMappable` class.

```
import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

# domains
x = np.logspace(-1.,np.log10(5),50) # [0.1, 5]
y = np.linspace(6,9,50)             # [6, 9]
z = np.linspace(-1,1,50)            # [-1, 1]

# convert to 2d matrices
Z = np.outer(z.T, z)        # 50x50
X, Y = np.meshgrid(x, y)    # 50x50

# fourth dimention - colormap
# create colormap according to x-value (can use any 50x50 array)
color_dimension = X # change to desired fourth dimension
minn, maxx = color_dimension.min(), color_dimension.max()
norm = matplotlib.colors.Normalize(minn, maxx)
m = plt.cm.ScalarMappable(norm=norm, cmap='jet')
m.set_array([])
fcolors = m.to_rgba(color_dimension)

# plot
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X,Y,Z, rstride=1, cstride=1, facecolors=fcolors, vmin=minn, vmax=maxx,
shade=False)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
fig.canvas.show()
```



The answer I referenced (and others) mentions that you should normalize your fourth dimension data. It seems that this may be avoided by explicitly setting the limits of the colormap as I did in the code sample.

Share Edit Follow Flag

edited May 23, 2017 at 12:24
Community Bot
1 1

answered Sep 9, 2015 at 12:52
Frik
1,054 1 13 16

Wonderful! But as I wrote in the question, my main problem lies in the evaluation of the fourth dimension array. Following your answer and my code, I should write `color_dimension = LikeBeta(y,range(50),range(50))` , i.e. a len(50) array. Its elements are evaluated in `LikeBeta` , a function of y and the indexes of x and z, which uses the components of sigma_los_beta, a (50,353,50) array. This leads to the broadcasting problem which I was mentioning. Once I solve this, I can apply your suggestion on how to colour the surface. Any idea on how to solve this problem? Thank you – andrea Sep 9, 2015 at 14:47

Can you perhaps provide details about the unknowns in your `LikeBeta` function? E.g. type and dimension of the following: `r0_array, cst, dv, u, v` . It is difficult to pinpoint the problem without the means to simulate it. The important thing is that the function should return an array that is of the same dimension as your axes variables. Also, do you intend to perform matrix or element-wise multiplication? – Frik Sep 9, 2015 at 15:57

I'm sorry, you're right. `r0_array` is `= x` , `cst` is simply a float and `dv` , `u` , `v` are all numpy.array of len 353 (these are datapoints). so, for simplicity, you can also remove those two lines of code and simply: `def LikeBeta(rho0,r0,beta):     M0 = 10**rho0*x[r0]**3     I = M0*sigma_los_beta[beta,:,r0]     return I` where `sigma_log_beta` is an np.array of shape (50,353,50). Given that I'm multiplying the whole array `sigma_los_beta[beta,:,r0]` by a scalar, it's element-wise – andrea Sep 9, 2015 at 17:28

You get the error because of size mismatches in the element-wise multiplication. Track the output dimensions in the function line by line to see where you need to apply changes. I cannot simply alter the function to correct the mismatch errors as I do not know what you wish to achieve. Have a look at `numpy.repeat` , and when you sum it might be necessary to do so along a single dimension only. – Frik Sep 9, 2015 at 19:00

I managed to solve the array dimensional problem by making `colors` a 3D array with a list comprehension. New question: how do I create a custom colormap from a 3D array? clearly, your `minn` , `maxx` from the above answer won't work with a 3D array. How can I create a mappable from my 3D array to the colormap? thanks – andrea Sep 14, 2015 at 13:58

3

This code is based on the trisurf demo
http://matplotlib.org/examples/mplot3d/trisurf3d_demo.html

I added a function make_colormap() based on the SO Create own colormap using matplotlib and plot color scale

Also added a sequence w=tan(-x*y) that generates a colour map based on that function, in the gray scale.
You can play with the construction of the cdict to add more colors to it but I think gray scale makes a good proof of concept...

Sorry I couldn't work directly with your example, due to lack of minimal working code.

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.colors as mcolors

###################

def make_colormap(seq):
    """Return a LinearSegmentedColormap
    seq: a sequence of floats and RGB-tuples. The floats should be increasing
    and in the interval (0,1).
    """
    #%
    cdict = {'red': [], 'green': [], 'blue': []}

    # make a lin_space with the number of records from seq.
    x = np.linspace(0,1, len(seq))
    #%
    for i in range(len(seq)):
        segment = x[i]
        tone = seq[i]
        cdict['red'].append([segment, tone, tone])
        cdict['green'].append([segment, tone, tone])
        cdict['blue'].append([segment, tone, tone])
    #%
    return mcolors.LinearSegmentedColormap('CustomMap', cdict)


#############################



n_angles = 36
n_radii = 8

# An array of radii
# Does not include radius r=0, this is to eliminate duplicate points
radii = np.linspace(0.125, 1.0, n_radii)

# An array of angles
angles = np.linspace(0, 2*np.pi, n_angles, endpoint=False)

# Repeat all angles for each radius
angles = np.repeat(angles[...,np.newaxis], n_radii, axis=1)

# Convert polar (radii, angles) coords to cartesian (x, y) coords
# (0, 0) is added here. There are no duplicate points in the (x, y) plane
x = np.append(0, (radii*np.cos(angles)).flatten())
y = np.append(0, (radii*np.sin(angles)).flatten())

# Pringle surface
z = np.sin(-x*y)



w = np.tan(-x*y)
colors = make_colormap(w)



fig = plt.figure()
ax = fig.gca(projection='3d')
```

```
ax.plot_trisurf(x, y, z, cmap=colors, linewidth=0.2)

plt.show()
```

Share Edit Follow Flag

edited May 23, 2017 at 11:53
Community Bot
1 1

answered Sep 8, 2015 at 17:03
PabTorre
2,878 21 30

Thank you very much PabTorre. however, if I try evaluating the equivalent of your w in my code, which is what drives the colouring of the 3D surface, I get the broadcasting error. If you examine my `LikeBeta` function, you notice that I'm calling it with three arrays of same length. However the function uses an array of different shape and this causes the broadcast error. I can't get around this. Do you know how can I fix this? – andrea Sep 9, 2015 at 13:28

When I try to run your function, r0_array is not defined, what is tge structure of that array? – PabTorre Sep 9, 2015 at 16:55