splinart Documentation

Release

Loic Gouarin

Nov 19, 2017

Contents

splinart is a package used for a tutorial which explains how to do the Python packaging using

- PyPi
- conda build
- pytest
- Pylint
- Sphinx

And automate the process to distribute this package using github.

The original idea of splinart is found on the great invonvergent website.

If you want to install splinart:

```
pip install splpinart
```

or:

```
conda install -c gouarin splinart
```

CHAPTER 1

User manual

1.1 Cubic Spline

We consider here a cubic spline passing through the points (x_i, y_i) with $a = x_1 < \ldots < x_n = b$, that is, a class function $\mathcal{C}^2$ on $[a, b]$ and each restriction at the interval $[x_{i-1}, x_i]$, $1 \leq i \leq n$, is a polynomial of degree less than 3. We will note S such a spline. His equation is given by

$$S_i(x) = Ay_i + By_{i+1} + Cy_i'' + Dy_{i+1}'', \qquad x_i \leq x \leq x_{i+1},$$

where

$$A = \frac{x_{i+1} - x}{x_{i+1} - x_i} \qquad \text{et} \qquad B = \frac{x - x_i}{x_{i+1} - x_i},$$

$$C = \frac{1}{6}\left(A^3 - A\right)(x_{i+1} - x_i)^2 \qquad \text{et} \qquad D = \frac{1}{6}\left(B^3 - B\right)(x_{i+1} - x_i)^2.$$

If we derive this equation twice with respect to x, we get

$$\frac{d^2S(x)}{dx} = Ay_i'' + By_{i+1}''.$$

Since $A = 1$ in x_i and $A = 0$ in x_{i+1} and conversely for B, we can see that the second derivative is continuous at the interface of the two intervals $[x_{i-1}, x_i]$ and $[x_i, x_{i+1}]$.

It remains to determine the expression of y_i''. To do this, we will calculate the first derivative and impose that it is continuous at the interface of two intervals. The first derivative is given by

$$\frac{dy}{dx} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{3A^2 - 1}{6}(x_{i+1} - x_i)y_i'' + \frac{3B^2 - 1}{6}(x_{i+1} - x_i)y_{i+1}''.$$

We therefore want the value of the first derivative in $x = x_i$ over the interval $[x_{i-1}, x_i]$ to be equal to the value of the first derivative in $x = x_i$ over the interval $[x_i, x_{i+1}]$; which gives us for $i = 2, \ldots, n-1$

$$a_i y_{i-1}'' + b_i y_i'' + c_i y_{i+1}'' = d_i,$$

with

$$
\begin{aligned}
a_i &= \tfrac{x_i - x_{i-1}}{x_{i+1} - x_{i-1}} \\
b_i &= 2 \\
c_i &= \tfrac{x_{i+1} - x_i}{x_{i+1} - x_{i-1}} \\
d_i &= \tfrac{6}{x_{i+1} - x_{i-1}} \left(\tfrac{y_{i+1} - y_i}{x_{i+1} - x_i} - \tfrac{y_i - y_{i-1}}{x_i - x_{i-1}} \right).
\end{aligned}
$$

So we have $n-2$ linear equations to calculate the n unknowns y_i'' for $i = 1, \ldots, n$. So we have to make a choice for the first and last values and we will take them equal to zero. We can recognize the resolution of a system with a tridiagonal matrix. It is then easy to solve it by using the algorithm of Thomas which one recalls the principle

$$
c_i' = \begin{cases} \tfrac{ci}{b_i} & i = 1 \\ \tfrac{c_i}{b_i - a_i c_{i-1}'} & i = 2, \ldots, n. \end{cases}
$$

$$
d_i' = \begin{cases} \tfrac{di}{b_i} & i = 1 \\ \tfrac{d_i - a_i d_{i-1}'}{b_i - a_i c_{i-1}'} & i = 2, \ldots, n. \end{cases}
$$

The solution is then obtained by the formula

$$
\begin{aligned}
y_n'' &= d_n' \\
y_i'' &= d_i' - c_i' y_{i+1}'' \qquad \text{pour} \qquad i = n-1, \ldots, 1.
\end{aligned}
$$

CHAPTER 2

Tutorial

2.1 Splinart on a circle

In this tutorial, we will see how to use splinart with a circle.

First of all, we have to create a circle.

```
In [34]: import splinart as spl
         center = [.5, .5]
         radius = .3
         theta, path = spl.circle(center, radius)
```

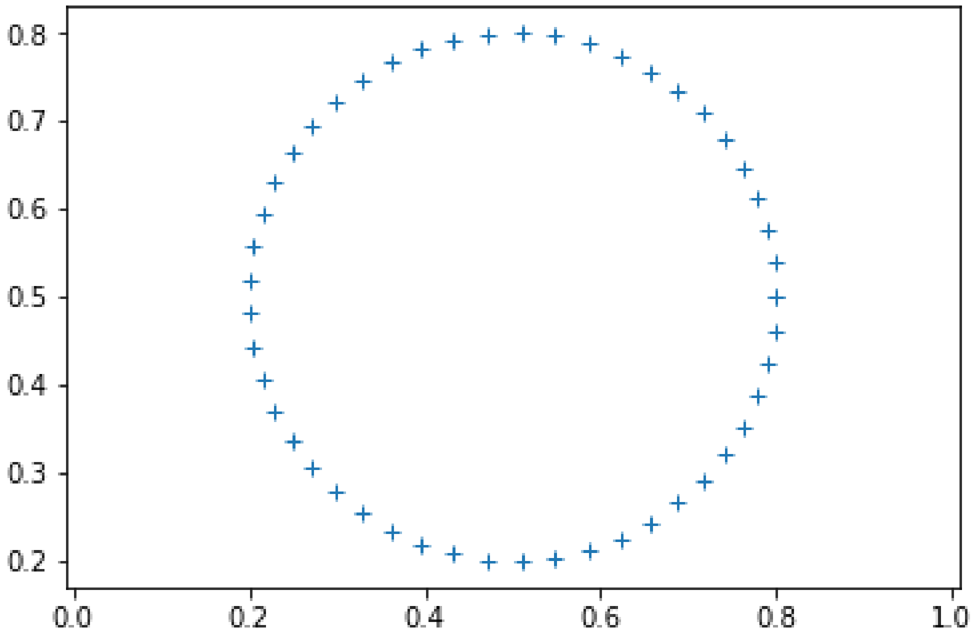
In the previous code, we create a discretization of a circle centered in $[0.5, 0.5]$ with a radius of 0.3. We don't specify the number of discretization points. The default is 30 points.

We can plot the points using matplotlib.

```
In [2]: %matplotlib inline
```

```
In [6]: import matplotlib.pyplot as plt
        plt.axis("equal")
        plt.plot(path[:, 0], path[:, 1], '+')
```

```
Out[6]: [<matplotlib.lines.Line2D at 0x7fc91dfe6048>]
```



2.1.1 The sample

In order to compute a sample on a given cubic spline equation, we need to provide a Python function that gives us the x coordinates. We can choose for example.

```
In [12]: import numpy as np
         def x_func():
             nsamples = 500
             return (np.random.random() + 2 * np.pi * np.linspace(0, 1, nsamples))%(2*np.pi)
```

We can see that the points are chosen between $[0, 2\pi]$ in a random fashion.

2.1.2 The cubic spline

Given a path, we can apply the spline function in order to compute the second derivative of this cubic spline.

```
In [13]: yder2 = spl.spline.spline(theta, path)
```

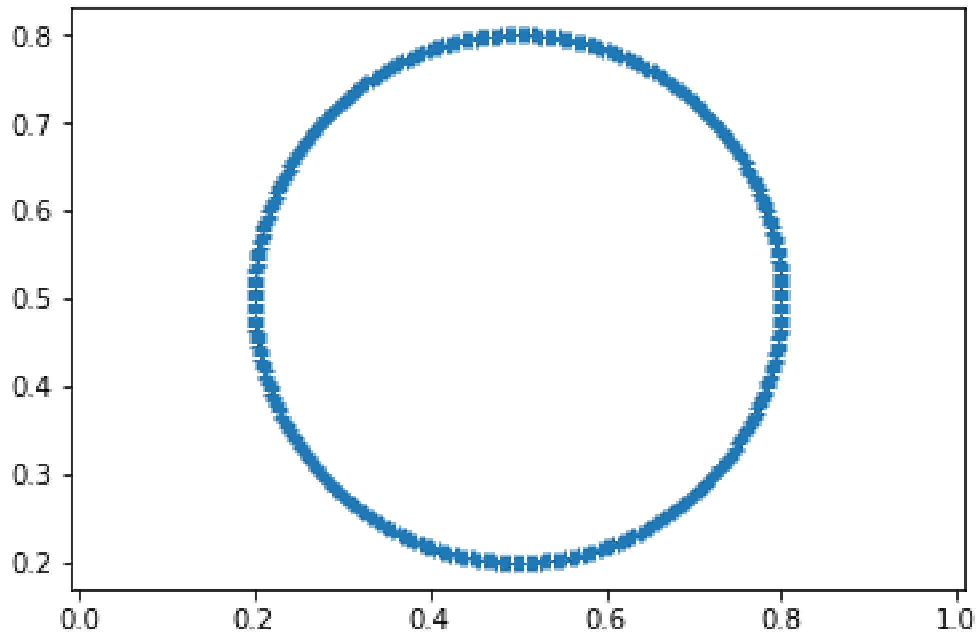
And apply the equation to the sample

```
In [14]: xsample = x_func()
         ysample = np.zeros((xsample.size, 2))
         spl.spline.splint(theta, path, yder2, xsample, ysample)
```

which gives

```
In [15]: import matplotlib.pyplot as plt
         plt.axis("equal")
         plt.plot(ysample[:, 0], ysample[:, 1], '+')

Out[15]: [<matplotlib.lines.Line2D at 0x7fc91e05feb8>]
```



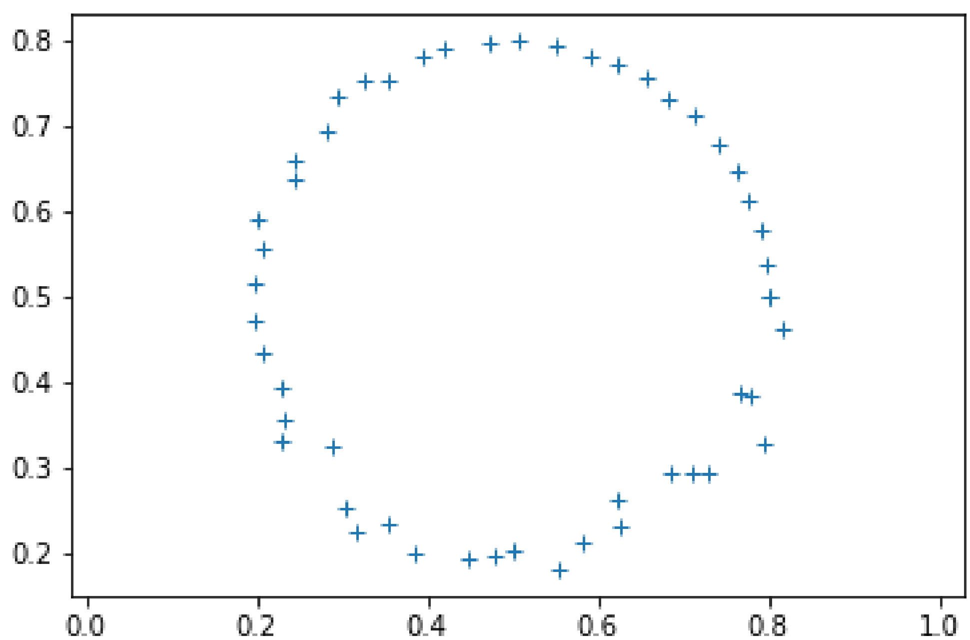
We can see the sample is well defined around the circle that we defined previously.

Now, assume that we move randomly the points of the circle with a small distance.

```
In [35]: spl.compute.update_path(path, scale_value=.001, periodic=True)
```

```
In [36]: import matplotlib.pyplot as plt
         plt.axis("equal")
         plt.plot(path[:, 0], path[:, 1], '+')
```

```
Out[36]: [<matplotlib.lines.Line2D at 0x7fc91da92e10>]
```

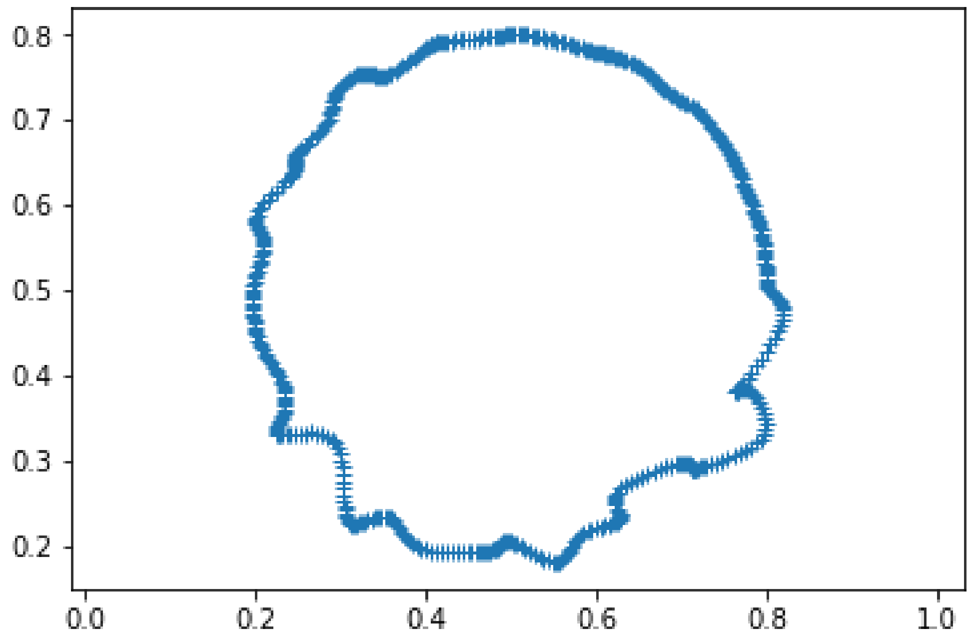


And we compute again the sample of the new cubic spline equation.

```
In [37]: yder2 = spl.spline.spline(theta, path)
         spl.spline.splint(theta, path, yder2, xsample, ysample)
         spl.compute.update_path(path, scale_value=.001, periodic=True)
```

```
In [39]: import matplotlib.pyplot as plt
         plt.axis("equal")
         plt.plot(ysample[:, 0], ysample[:, 1], '+')
```

```
Out[39]: [<matplotlib.lines.Line2D at 0x7fc91d94f748>]
```



The circle is deformed.

This is exactly how works splinart. We give a shape and at each step

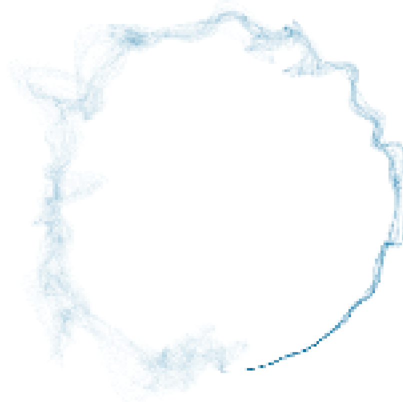
- we perturb the points of this shape
- we compute a sample an this new cubic spline equation
- we add the pixel with a given color on the output image

And we do that several time. We can have the following result

```
In [40]: img_size, channels = 1000, 4
         img = np.ones((img_size, img_size, channels), dtype=np.float32)

         theta, path = spl.circle(center, radius)
         spl.update_img(img, path, x_func, nrep=4000, x=theta, scale_value=.00005)
```

```
In [41]: spl.show_img(img)
```



CHAPTER 3

Reference manual

3.1 splinart

3.1.1 splinart package

Subpackages

splinart.shapes package

Submodules

splinart.shapes.base module

Define basic shapes.

`splinart.shapes.base.`**`circle`**(*center*, *radius*, *npoints=50*)

Discretization of a circle.

Parameters

- **`center`** (*`list (2)`*) – 2d coordinates of the center.
- **`radius`** (*`float`*) – Radius of the circle.
- **`npoints`** (*`int`*) – Number of discretization points (the default value is 50).

Returns

- *np.ndarray* – The theta angle.
- *np.ndarray* – The 2d coordinates of the circle.

`splinart.shapes.base.`**`line`**(*begin*, *end*, *ypos=0.5*, *npoints=50*)

Discretization of a horizontal line.

Parameters

- **begin** (*float*) – The left point of the line.
- **end** (*float*) – The right point of the line.
- **ypos** (*float*) – The position of the y coordinate (the default value is 0.5).
- **npoints** (*int*) – Number of discretization points (the default value is 50).

Returns The 2d coordinates of the line.

Return type np.ndarray

Module contents

Shape package

splinart.spline package

Submodules

splinart.spline.spline module

Cubic spline

`splinart.spline.spline.spline`(*xs*, *ys*)

Return the second derivative of a cubic spline.

Parameters

- **xs** (*np.ndarray*) – The x coordinate of the cubic spline.
- **ys** (*np.ndarray*) – The y coordinate of the cubic spline.

Returns The second derivative of the cubic spline.

Return type np.ndarray

splinart.spline.splint module

Integration of a cubic spline.

`splinart.spline.splint.splint`(*xs*, *ys*, *y2s*, *x*, *y*)

Evaluate a sample on a cubic pline.

Parameters

- **xs** – The x coordinates of the cubic spline.
- **ys** – The y coordinates of the cubic spline.
- **y2s** – The second derivative of the cubic spline.
- **x** – The sample where to evaluation the cubic spline.
- **y** – The y coordinates of the sample.

See also:

splinart.spline.spline()

Module contents

Spline package

Submodules

splinart.color module

Define the default color of the output.

splinart.compute module

Material to update the output image using a cunbic spline equation.

splinart.compute.**update_img**(*img*, *path*, *xs_func*, *x=None*, *nrep=300*, *periodic=True*, *scale_color=0.005*, *color=(0.0*, *0.41568627450980394*, *0.6196078431372549*, *1.0)*, *scale_value=1e-05*)

Update the image using a cubic spline on a shape.

Parameters

- **img** (*np.ndarray*) – The output image.
- **path** (*np.ndarray*) – The y coordinate of the cubic spline if x is not None, the coordinates of the cubic spline if x is None.
- **x** (*np.ndarray*) – The x coordinates of the cubic spline if given. (the default value is None)
- **xs_func** (*function*) – The function that return the x coordinate of the sampling points where to compute the y coordinates given the spline equation.
- **nrep** (*int*) – Number of iteration (default is 300).
- **periodic** (*bool*) – Define if the first and last points of the path must be equal (default is True).
- **scale_color** (*float*) – Scale the given color (default is 0.005).
- **color** (*list (4)*) – Define the RGBA color to plot the spline.
- **scale_value** (*float*) – Rescale the random radius (default value is 0.00001).

See also:

update_path()

splinart.compute.**update_path**(*path*, *periodic=False*, *scale_value=1e-05*)

Update the path of the spline.

We move each point of the path by a random vector defined inside a circle where

- the center is the point of the path
- the radius is a random number between [-1, 1]

Parameters

- **path** (*np.ndarray*) – The y coordinate of the cubic spline.
- **periodic** (*bool*) – If True, the first and the last points of the path are the same (the default value is False).

- **scale_value** (*float*) – Rescale the random radius (default value is 0.00001).

splinart.draw module

Material to update the image with given points and save or plot this image.

splinart.draw.**draw_pixel**(*img*, *xs*, *ys*, *scale_color=0.0005*, *color=(0.0*, *0.41568627450980394*, *0.6196078431372549*, *1.0)*)

Add pixels on the image.

Parameters

- **img** (*np.ndarray*) – The image where we add pixels.
- **xs** (*np.ndarray*) – The x coordinate of the pixels to add.
- **ys** (*np.ndarray*) – The y coordinate of the pixels to add.
- **scale_color** (*float*) – Scale the given color (default is 0.0005).
- **color** (*list (4)*) – Define the RGBA color of the pixels.

splinart.draw.**save_img**(*img*, *path*, *filename*)

Save the image in a png file.

Parameters

- **img** (*np.ndarray*) – The image to save.
- **path** (*str*) – The save directory.
- **filename** (*str*) – The file name with the png extension.

splinart.draw.**show_img**(*img*)

Plot the image using matplotlib.

Parameters **img** (*np.ndarray*) – The image to save.

splinart.version module

Module contents

Splinart package

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

C

D

L

S

U