Report for Week 12

November 19, 2021

Information: *A brief report listing the work I've done in the last week.*

Written by: *Zihao Xu*

E-mail: *xu1376@purdue.edu*

1 Problem Statement

1.1 Goal

- Find a way to provide precise descriptions for a given curve from hand sketches.

1.2 Current status

- While we can use the algorithms in the Scipy package to fit B-spline curves conveniently, it's hard to automatically determine the number of control points required for the curve.
- In the meanwhile, it's hard to resample from the given curve since the B-spline curve fitting algorithm will provide the same number of control points as the number of sampled data points. Right now, the rule based re-sampling algorithm can only work well for a some simple curves after well-fitted.

1.3 Possible directions

1.3.1 Machine learning methods

- Construct a dataset of various B-spline curves labeled by their number of control points, use some kind of recurrent neural networks (the lengths of input curve are generally different) to learn the optimal number of control points.
- Given the optimal number of control points, solve for optimization problem where we would like to minimize the reconstruction error by changing sampled points.

1.3.2 Local features

- Smooth the curve to be re-sampled and see if the local features such as the curvatures would be better.

2 Explorations

2.1 Machine leaning method: DeepSpline

2.1.1 Summary

The paper proposing **DeepSpline** provides a deep learning architecture that adapts to perform spline fitting tasks accordingly. It highlights the feature that it can handle spline fitting tasks with undetermined number of control points using data-driven approaches, which sounds very promising to the application to our spline fitting tasks. Here are the main contributes as stated in the paper.

- Defines two single-layer RNNs, Curve RNN and Point RNN that can be used to perform curve predictions and control point predictions respectively.
- Provides a Hierarchical RNN architecture model that performs reconstruction of 2D spline curves of variable number, each of which contains a variable number of control points using nested Curve RNN and Point RNN units, and an algorithm to train it effectively.
- Provides an unsupervised parametric reconstruction model that performs reconstruction of 3D surfaces of extrusion or revolution.

2.1.2 Relevance to our task

- **The paper's target:** This part related to 2D shapes in this paper aims at the reconstruction of 2D parametric curves from images. It divide the problem into two stages, prediction of spine curves in the image to reconstruct, and the prediction of the actual control points that reconstruct the afore-mentioned spline curve. In the meanwhile, since there might be multiple spline curves in images, it need to consider not only the number of control points, but also the number of spline curves.
- **Our target:** In out sketch interface, we're trying to get the parametric descriptions once the user input some shapes. The interface will then divide the shapes into primitives. The spline fitting algorithm will only be invoked when some shapes are considered as a spline curve. That is to say, the input to this algorithm will only be one preprocessed spline curve. This target is same as the second stage of this paper's target.

2.1.3 Inspirations

- **Use synthetically generated curves to train the model predicting required number of control points.**
- Th recurrent neural network with attention network might be referred to when we would like to build our own models.
- A neural network model directly outputs the control points might not be stable enough for design interactions.

2.2 Re-sampling: optimization problem

As stated earlier, I would like to view the re-sampling method as an optimization problem. Denote $\mathbf{S}$ as the input curve we would like to perform B-spline fitting to. Its dimension is $d \in \{2, 3\}$ and there are $m > 3$ sampled data points. That is to say, the input curve is a $m \times d$ matrix.

$$\text{Input curve: } \mathbf{S} \in \mathbb{R}^{m \times d}$$

Select n points from the input curve and get the re-sampled curve as a $n \times d$ matrix.

$$\text{Re-sampled curve: } \mathbf{S}_r \in \mathbb{R}^{n \times d}$$

With this re-sampled curve $\mathbf{S}'$, use the cubic B-spline fitting method and get the vector of knots $\mathbf{t}$ and the B-spline coefficients (control points) $\mathbf{c}$. Then use these obtained spline parameters, reconstruct the B-spline curve.

$$\text{Re-constructed curve: } \mathbf{S}_c \in \mathbb{R}^{m \times d}$$

Calculate the reconstruction error.

$$l = \|\mathbf{S} - \mathbf{S}_c\|_F^2$$

Thus defining the desired re-sampling result.

$$\mathbf{S}_r^* = \underset{\mathbf{S}_r}{\operatorname{argmin}} \{\|\mathbf{S} - \mathbf{S}_c\|_F^2\}$$

However, I found out that this straight-forward optimization problem is not seemingly easy to solve. On one hand, there's no straight derivative method such as gradient descent which can help locating the optimum. On the other hand, grid search requires huge computations and is not feasible.

Instead, there are optimization algorithms directly optimizing the positions of the control points. One example is the **Fast B-spline Curve Fitting by L-BFGS**. The optimization problem is stated as follows:

- Given a set of data points $\{\mathbf{X}_i\}_{i=1}^N \subset \mathbb{R}^2$ sampled from the outline of a planar shape, the aim of curve fitting is to find a B-spline curve $\mathbf{P}(t) = \sum_{i=1}^n \mathbf{P}_i N_i(t)$ that best approximates the shapes' outline.
- Denote $\mathcal{P} = \{\mathbf{P}_i\}_{i=1}^n \subset \mathbb{R}^2$ as the set if B-spline control points, $\{N_i(t)\}_i^n = 1$ are B-spline basis functions.
- Suppose the knots of the B-spline curve are fixed and therefore not subject to optimization. Similarly, suppose all the basis functions are thus defined on fixed, uniform spaced knots throughout the curve fitting process.
- For a data point $\mathbf{X}_k$, let $\mathbf{P}(t_k)$ denote the nearest point of $\mathbf{X}_k$ on the fitting curve. Then the distance between data point $\mathbf{X}_k$ and the fitting curve is $\|\mathbf{P}(t_k) - \mathbf{X}_k\|$. Here t_k is called the location parameter of $\mathbf{X}_k$, $\mathbf{P}(t_k)$ is called the foot point corresponding to $\mathbf{X}_k$.
- Denote $\mathcal{T} = \{t_1, \ldots, t_k\}$ as the collection of the location parameters of all the data points.
- Formulate the fitting problem as

$$\min_{\mathcal{P},\mathcal{T}} \frac{1}{2} \sum_{k=1}^{N} \|\mathbf{P}(\mathcal{P}; t_k) - \mathbf{X}_k\|^2 + \lambda F_{fairing}$$

  where $F_{fairing}$ is a fairing term which defines the fairness of a curve.

Right now I do **not** think we should consider this optimization problem since this method is also well solved given fixed number of control points while it does not solve for variable number of control points. But this optimization idea inspired me to look into the scipy algorithms again and found that the algorithm is also doing some optimization computations instead of simple matrix operations.

2.3 Control number of knots in scipy

When I look again into the B-spline definition to gain some insights, I noticed the connection:

$$\text{number of knots} = \text{number of coefficients} + \text{degree} + 1$$

That is to say, the number of coefficients (control points) might be able to be controlled by the number of knots. The scipy package would be able to optimize the control points given fixed knot vectors.

Therefore, I did some simple tests to check that. Here are the results.

First import all the necessary modules.

[1]:
```
import warnings
import time
import numpy as np
import matplotlib.pyplot as plt
import scipy.interpolate as interpolate
from numpy.linalg import norm
```

Then modify the codes for spline fitting. It needs to be mentioned that here I ignored the warnings reporting too small smoothing condition. As long as I set this parameter small to ensure the fitting as close as possible to the original curve, the warning occurs. Otherwise, it may use fewer control points just to satisfy the upper bound of tolerance.

[2]:
```
def rm_rep(curve):
    """ Remove all the repetitive points in a sampled curve from the sketch
        curve:    Sampled data points. Array with the size [num, dim]
    """
    cv = curve.copy()
    pts = []
    pts.append(cv[0, :])
    for ii in range(1, cv.shape[0]):
        if not (cv[ii, :] == cv[ii - 1, :]).all():
            pts.append(cv[ii, :])
    pts = np.array(pts)
    if cv.shape[0] - pts.shape[0] != 0:
        print("Delete %d repetitive point(s)." % (cv.shape[0] - pts.shape[0]))
    return pts
```

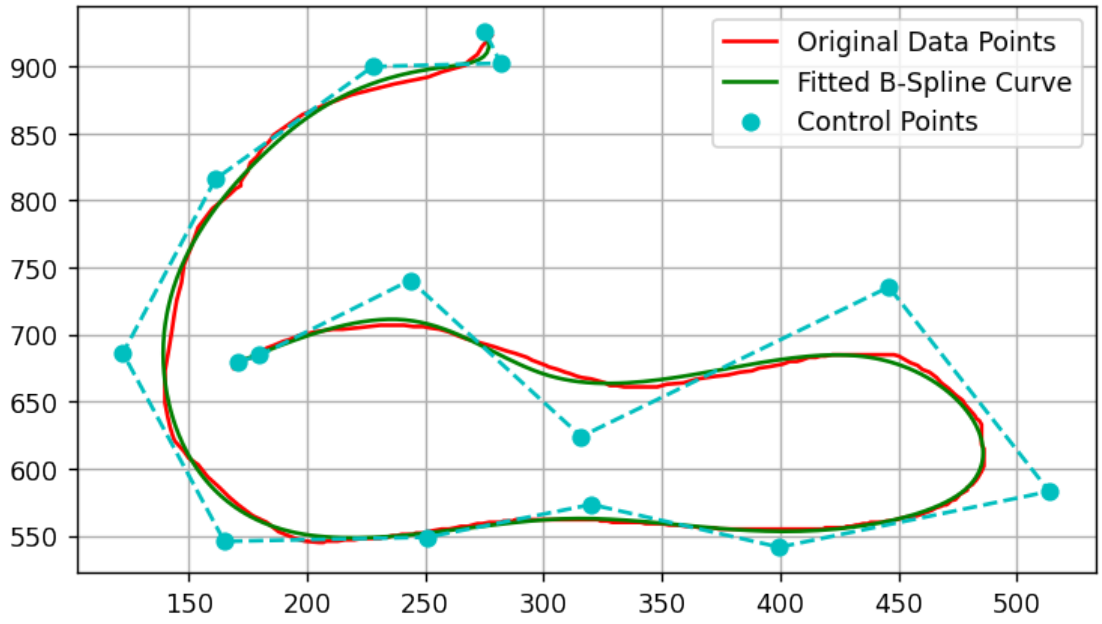
[3]:
```
def bsf(curve, m, closed=False, plot=False):
    """ Fit a B-spline curve to the sampled points with given number of
        control points.
        curve:   Sampled data points. Array with the size [num, dim]
```

```
        m:       Desired number of  control points.
        closed:  Closed curves would have a repetitive data point
                 at the end. Set to true to implicate this.
        plot:    Whether to plot it out or not.
    """
    # Record the start time
    start = time.time()
    pts = curve.copy()
    pts = rm_rep(pts)
    # Construct the knot vector given number of control points
    t3 = np.linspace(0, 1, num=(m - 2))
    t3 = np.zeros(m + 4)
    t3[3:-3] = np.linspace(0, 1, num=(m - 2))
    t3[-3:] = np.ones(3)
    # Fit the B-Spline Curve given desired number of control points
    # Ignore the fitting warning
    with warnings.catch_warnings():
        warnings.filterwarnings("ignore", message="The required storage space␣
↪exceeds the available storage space.")
        if closed == True:
            tck, u = interpolate.splprep(pts.T, u=None, t=t3, nest=m+4, s=1,␣
↪k=3, per=1)
        else:
            tck, u = interpolate.splprep(pts.T, u=None, t=t3, nest=m+4, s=1,␣
↪k=3)
    cof = tck[1]
    print('Obtain %d control points.' % cof[0].shape[0])
    T = np.linspace(u.min(), u.max(), curve.shape[0])
    spl = interpolate.splev(T, tck)
    mse = (np.square(np.array(spl).T - curve)).mean(axis=0)
    print('The mean square error is: ', mse,'.')
    # Calculate the time cost
    end = time.time()
    print('The process takes %f seconds'%(end-start))
    if plot == True:
        # Draw out the fit spline with computed control points
        fig, ax = plt.subplots(figsize=(7, 4), dpi=125)
        ax.plot(curve[:, 0], curve[:, 1], "r", label='Original Data Points')
        ax.plot(spl[0], spl[1], 'g', label='Fitted B-Spline Curve')
        ax.plot(cof[0], cof[1], '.c', markersize=12,label='Control Points')
        ax.plot(cof[0], cof[1], '--c')
        plt.grid()
        plt.legend(loc='best')
        plt.show()
    return pts, tck, u
```

Test the algorithm on previous curves.

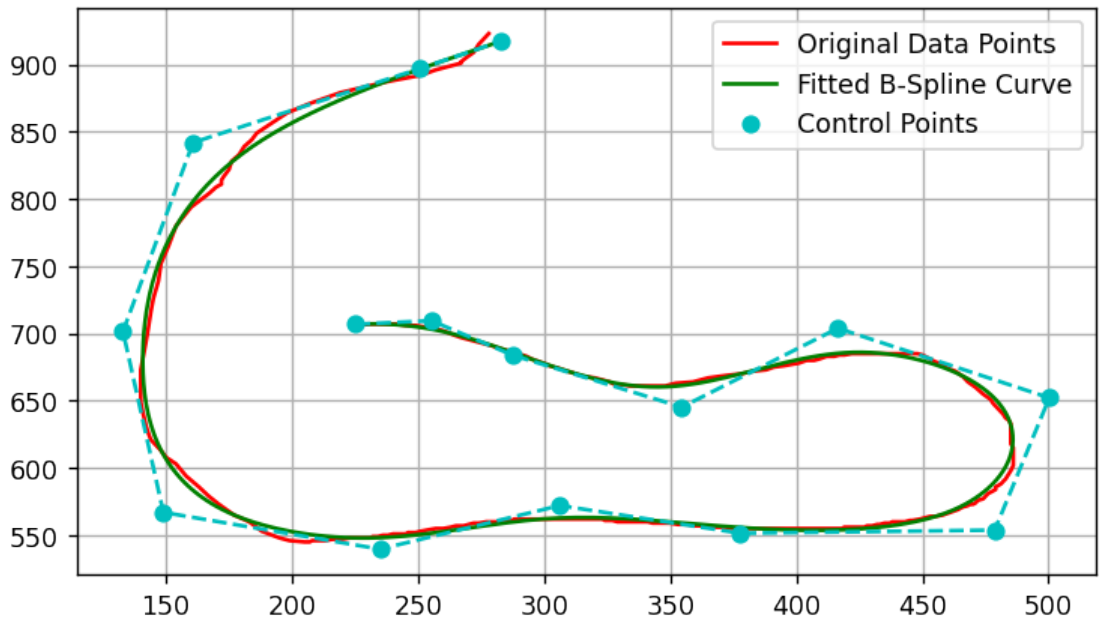
[4]:
```
# First curve
Curve1 = np.loadtxt('.\Resources\Curve1.txt')
print('The number of data points is %d' % Curve1.shape[0])
# Draw out the fitting result
pts1, tck1, u1 = bsf(Curve1, m=15, closed=False, plot=True)
```

```
The number of data points is 330
Delete 1 repetitive point(s).
Obtain 15 control points.
The mean square error is:  [4023.03945302 5820.31595253] .
The process takes 0.002954 seconds
```



[5]:
```
# Second curve
Curve2 = np.loadtxt('.\Resources\Curve2.txt')
print('The number of data points is %d' % Curve2.shape[0])
# Draw out the fitting result
pts2, tck2, u2 = bsf(Curve2, m=15, closed=False, plot=True)
```

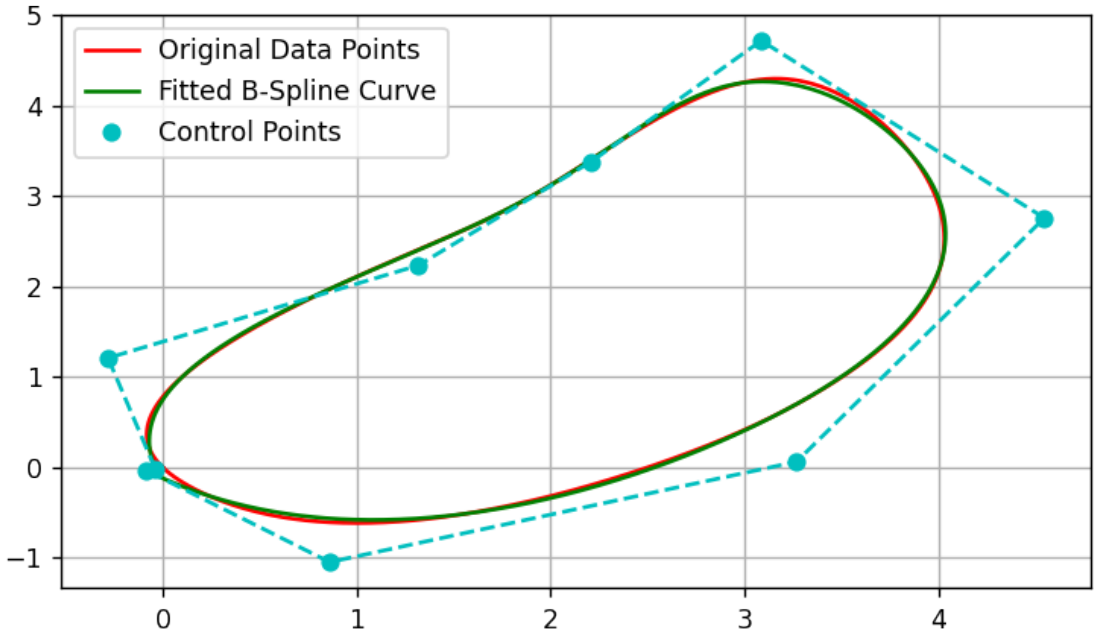
```
The number of data points is 312
Delete 1 repetitive point(s).
Obtain 15 control points.
The mean square error is:  [4345.224738   6217.76758492] .
The process takes 0.002993 seconds
```



For this closed curve, since I did not set the first element and the last element to be the same, I simply called the function for open curves. The first and last control point can be approximately viewed as one point. Further developments requires the standardization to the input form of closed curves.

[6]:
```
# Third curve
points = np.array([[0, 0], [1.2, 2.3], [1.9, 3], [3.2, 4.3], [4, 2.9], [3.5, 1],
                   [2.5, 0], [0.5, -0.5], [0, 0]])
tck, u = interpolate.splprep(points.T, u=None, s=0, k=3, per=1)
T = np.linspace(u.min(), u.max(), 2000)
Spline = interpolate.splev(T, tck)
Curve3 = np.concatenate(
    (Spline[0].reshape(-1, 1), Spline[1].reshape(-1, 1)), axis=1)
# Draw out the fitting result
pts3, tck3, u3 = bsf(Curve3, m=9, closed=False, plot=True)
```

```
Obtain 9 control points.
The mean square error is:  [0.0013757  0.00288336] .
The process takes 0.009014 seconds
```



As shown above, the current fitting algorithm could successfully construct the open curves well given desired number of control points. It does not require the sampling process to determine which points are 'important' to the reconstruction.

3 Summary and future works

As stated earlier, I divided the problem into two stages.

- Find the optimum number of control points for a curve.
- Find the best fitting curve given the number of control points.

So far, the second part has almost been finished except some tiny changes to adopt closed curves input, which should be standardized first (e.g. whether repeat the first element in the end). Those tiny changes can be made whenever this algorithm is to be embedded into the interface.

About the first part, according to the literature investigation, there are two possible directions:

- Given desired precision (reconstruction error), iteratively fit with increasing number of control points and find the minimal number of control points satisfying the error tolerance. As shown above, one fitting process only takes less than 0.01 seconds, which enables the line search for appropriate number of control points.
- Use machine learning methods to learn the optimal number of control points.

As the first approach would not require too much coding works and assigning the appropriate precision is simply a work of trial and errors, I prefer focusing on the machine learning methods right now and check the feasibilities. The detailed plans are:

- Generate different B-spline curves with different number of control points.
- Convert those splines of different length and with different number of control points to some standard images.
- With the images and corresponding number of control points as the training dataset, build a convolutional neural network model and train it.
- Check the performance of the trained model on real sketches.