Report for Week 11

November 12, 2021

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

Written by: *Zihao Xu*

E-mail: *xu1376@purdue.edu*

1 Tests on developed sample and fitting method

As stated in the report for week 10, more curves should be tested with the sampling and fitting method, while the mean square error should be calculated to evaluate the performance of the re-fitting process. I need to appreciate Yukun for providing me the curves for testing.

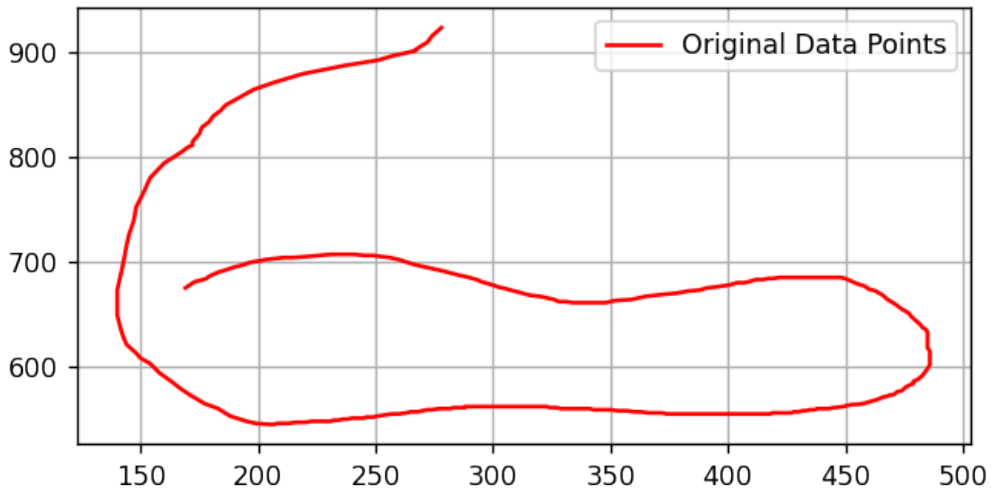
1.1 Curves to be fitted

Here are the curves provided by Yukun for testing the developed algorithm. I coded the data points from the pdf he sent to me to txt files which can be conveniently loaded by scripts. It needs to be mentioned that Yukun actually provided me with three curves while the first curve and the third curve are exactly the same. Therefore, I only focused on the two curves which are slightly different from each other.

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

[2]:
```
# First curve
Curve1 = np.loadtxt('.\Resources\Curve1.txt')
print('The number of data points is %d' % Curve1.shape[0])
# Draw out the spline with computed control points
fig, ax = plt.subplots(figsize=(6, 3), dpi=125)
ax.plot(Curve1[:, 0], Curve1[:, 1], "r", label='Original Data Points')
plt.grid()
plt.legend(loc='best')
plt.show()
```

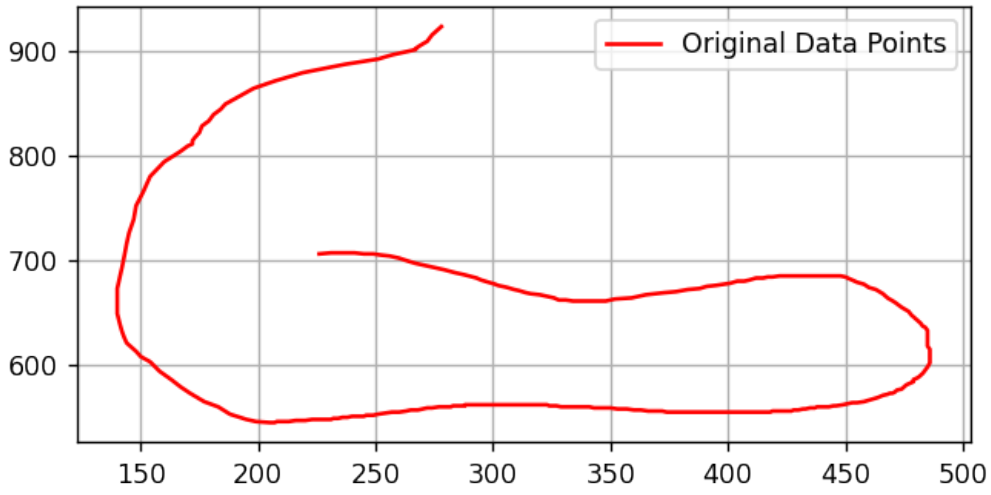
```
The number of data points is 330
```



The second curve is almost the same as the first one, but it's shorter. Therefore, we would expect less sample points from this curve.

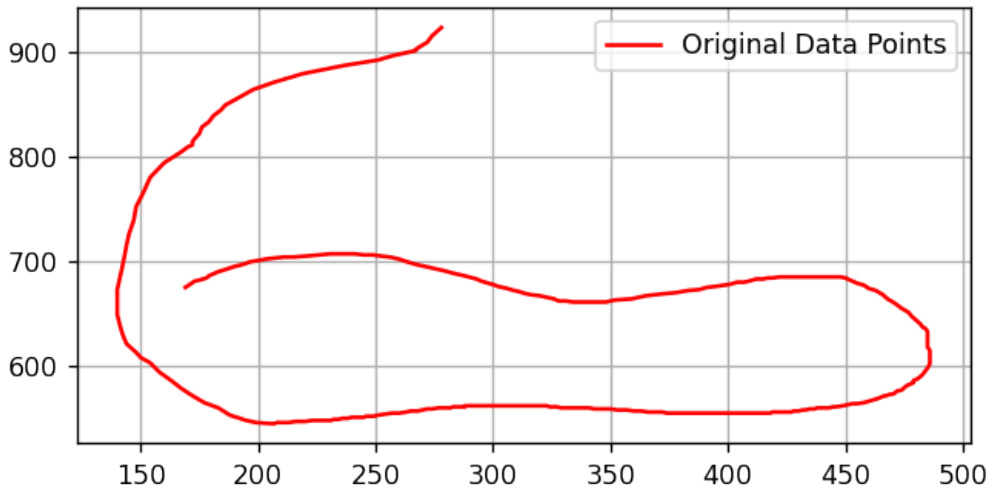
[3]:
```
# Second curve
Curve2 = np.loadtxt('.\Resources\Curve2.txt')
print('The number of data points is %d' % Curve2.shape[0])
# Draw out the curve with computed control points
fig, ax = plt.subplots(figsize=(6, 3), dpi=125)
ax.plot(Curve2[:, 0], Curve2[:, 1], "r", label='Original Data Points')
plt.grid()
plt.legend(loc='best')
plt.show()
```

```
The number of data points is 312
```



[4]:
```
# Third curve
Curve3 = np.loadtxt('.\Resources\Curve3.txt')
print('The number of data points is %d' % Curve3.shape[0])
# Draw out the curve
fig, ax = plt.subplots(figsize=(6, 3), dpi=125)
ax.plot(Curve3[:, 0], Curve3[:, 1], "r", label='Original Data Points')
plt.grid()
plt.legend(loc='best')
plt.show()
```

```
The number of data points is 330
```



1.2 Modified Algorithms

Considering the repetitive data points in the given curves, a function to obtain the non-repeating points from a given curve is developed.

[5]:
```
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
```

The re-sample algorithm is also modified to embed the function of removing repetitive points. In the meanwhile, considering the different dimensions of input curves, a normalization process is also added into the function.

[6]:
```
def resample(curve, num=0, closed=False):
    """ Resample a curve to the desired number of points
        curve:   Sampled data points. Array with the size [num, dim]
        num:     Number of re-sampled data points
                 If set to 0, automatically determine the number
        closed:  Input closed curves would have a repeating data point
                 at the end. Set to true to implicate this.
    """
    cv_ori = curve.copy()
    # Get the curve with no repetitive points
    cv_ori = rm_rep(cv_ori)
    # The initial parametrisation is uniform on [0,1]
    du = 1 / (cv_ori.shape[0] - 1)
    # Normalize the curve
    cv = cv_ori.copy()
    for ii in range(cv.shape[1]):
        cv[:,ii] = cv[:,ii] / max(cv[:,ii])
    # Calculate the approximated arc length parametrisation
    l = np.zeros(cv.shape[0])
    distance = 0
    for ii in range(1, cv.shape[0]):
        cv_dot = (cv[ii, :] - cv[ii - 1, :]) / du
        distance += norm(cv_dot, 2) * du
        l[ii] = distance
    # Calculate the approximated curvature parametrisation
    k = np.zeros(cv.shape[0])
    distance = 0
    cv_dot = (cv[1, :] - cv[0, :]) / du
    cv_ddot = (cv[1, :] - cv[0, :]) / (du**2)
    curvature = np.abs(np.cross(cv_dot, cv_ddot)) / (norm(cv_dot, 2)**3)
    distance += (l[1] - l[0]) * curvature
    k[1] = distance
    for ii in range(2, cv.shape[0]):
        cv_dot = (cv[ii, :] - cv[ii - 1, :]) / du
        cv_ddot = (cv[ii, :] - 2 * cv[ii - 1, :] + cv[ii - 2]) / (du**2)
        # Add a tiny number for stability
        curvature = np.abs(np.cross(cv_dot, cv_ddot)) / (norm(cv_dot, 2)**3)
        distance += (l[ii] - l[ii - 1]) * curvature
        k[ii] = distance
    # The overall parametrisation
```

```
    p = 0.9 * l + 0.1 * k
    # Check whether the number of resampled points is assigned
    if num == 0:
        step = 0.25
        num = int(p[-1] // step + 1)
        step = p[-1] / (num - 1)
    else:
        step = p[-1] / (num - 1.0)
    # Select the resampled points
    pts = np.zeros((num, cv.shape[1]))
    pts[0, :] = cv_ori[0, :]
    pts[num - 1, :] = cv_ori[-1, :]
    for ii in range(1, num - 1):
        for jj in range(0, cv.shape[0]):
            if p[jj] >= step * ii:
                pts[ii, :] = cv_ori[jj, :]
                break
    # Again, delete the possible repetitive points
    pts = rm_rep(pts)
    # Return the resampled points
    if closed:
        # Add an additional point for closed point
        pts_a = np.zeros((num + 1, cv.shape[1]))
        pts_a[0:num + 1, :] = pts
        pts_a[num + 1, :] = pts[0, :]
        return pts_a
    else:
        return pts
```

Finally, I compile all the functions as one function for convenient testing the algorithms.

[7]:
```
def bsf(curve, num=0, closed=False, plot=False):
    """ Resample a curve to the desired number of points and fit a
        B-spline curve to it.
        curve:   Sampled data points. Array with the size [num, dim]
        num:     Number of re-sampled data points.
                 If set to 0, automatically determine the number.
        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.
    """
    # Resample the first curve
    pts = resample(curve, num, closed)
    # Fit the B-Spline Curve from resampled points
    if closed == True:
        tck, u = interpolate.splprep(pts.T, u=None, s=0, k=3, per=1)
    else:
```

```
        tck, u = interpolate.splprep(pts.T, u=None, s=0, k=3)
    cof = tck[1]
    print('Obtain %d control points.'%u.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)
    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(pts[:, 0], pts[:, 1], 'kx', label='Resample Data Points')
        ax.plot(spl[0], spl[1], 'g', label='Fitted B-Spline Curve')
        ax.plot(cof[0], cof[1], '.c', label='Control Points')
        ax.plot(cof[0], cof[1], '--c')
        plt.grid()
        plt.legend(loc='best')
        plt.show()
    return pts, tck, u
```

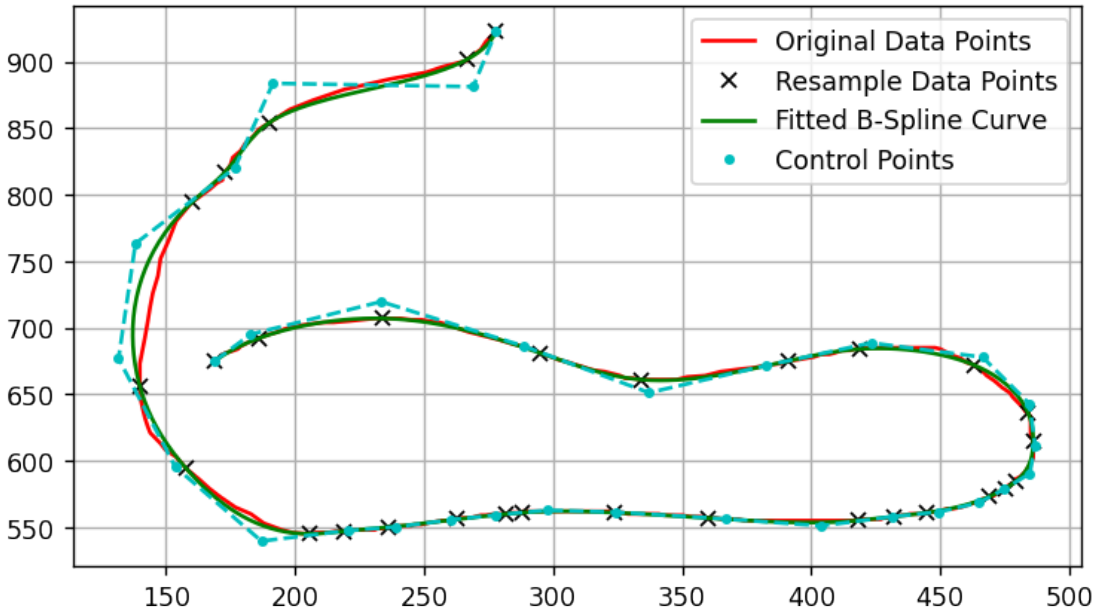
1.3 Sampling and refitting

Let's check the performance of the develop algorithms. The MSE loss is calculated to evaluate the modifications to the developed algorithm.

Sample and fit the first curve.

[8]:
```
pts1, tck1, u1 = bsf(Curve1, num=0, closed=False, plot=True)
```

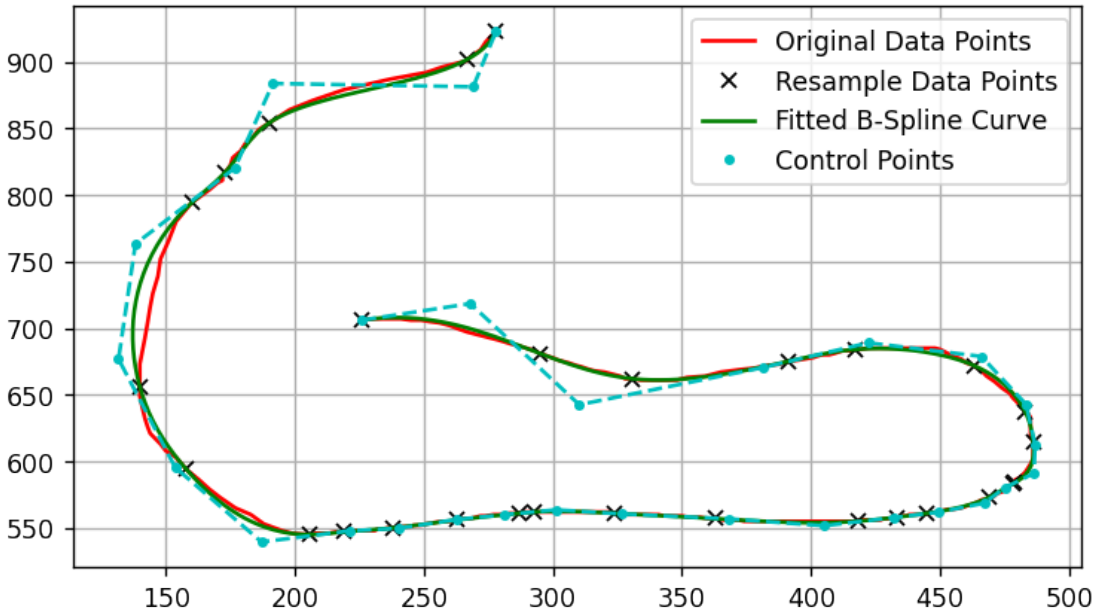
```
Delete 1 repetitive point(s).
Delete 1 repetitive point(s).
Obtain 31 control points.
The mean square error is:  [4165.68115476 5937.01680446]
```



Sample and fit the second curve.

[9]:
```
pts2, tck2, u2 = bsf(Curve2, num=0, closed=False, plot=True)
```

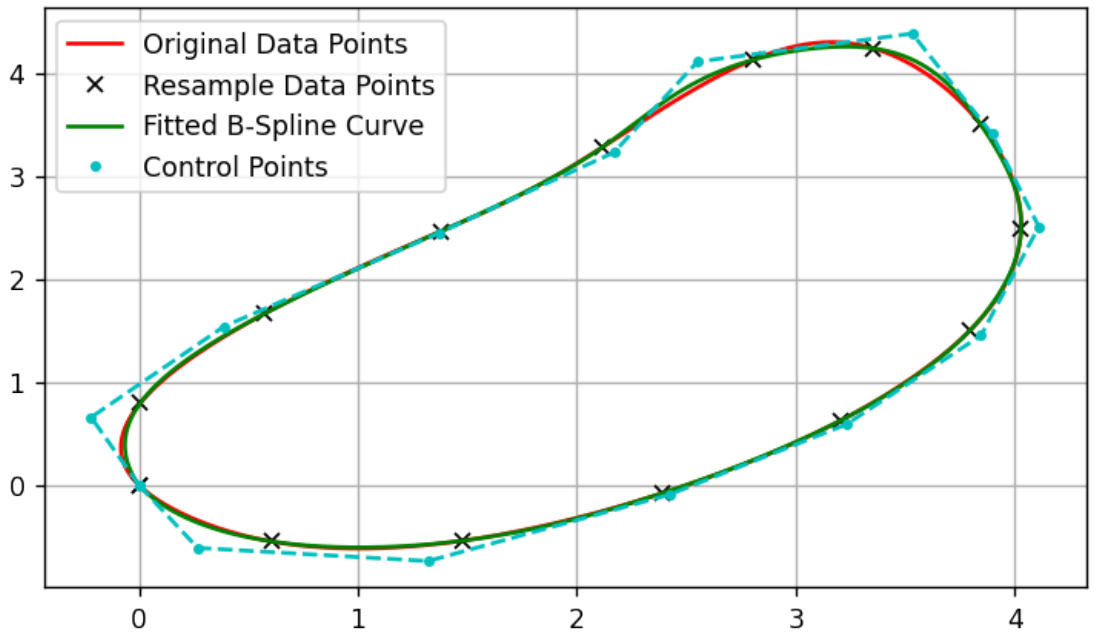
```
Delete 1 repetitive point(s).
Delete 1 repetitive point(s).
Obtain 29 control points.
The mean square error is:  [4515.87671316 6357.38721682]
```



Try the curve obtained last time with the same parameters.

[10]:
```
# Construct a curve to be fit
Curve = 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]])
# Fit with B-spline curve
tck, u = interpolate.splprep(Curve.T, u=None, s=0, k=3, per=1)
# Sample from the B-spline curve
T = np.linspace(u.min(), u.max(), 2000)
Spline = interpolate.splev(T, tck)
Curve4 = np.concatenate(
    (Spline[0].reshape(-1, 1), Spline[1].reshape(-1, 1)), axis=1)
# Resample and fit this curve
pts3, tck3, u3 = bsf(Curve4, num=0, closed=False, plot=True)
```

```
Obtain 15 control points.
The mean square error is:  [0.00246302 0.00278222]
```



1.4 Conclusion

- It's hard to set the parameters to be good for automatically fitting of all the curves with different dimensions.
- The approximation of the curvature shows a relatively large error which makes the sampling less reliable. The fitted B-spline curve sometimes hold great mean square errors.

- It's difficult to automatically get satisfactory re-sampling results by focusing on local features and fixed parameterization.

1.5 Possible work

- **Optimization problem solved by gradient descent:** Check if it is possible to view the problem as an optimization problem.
  - Divide the task into two parts:
    - Given desired number of control points, find the optimal position of re-sampling points for B-spline fitting.
    - Given a curve, find the optimal number of control points.
  - For the first part, check if the optimization process is supported by pytorch with gradient descent.
    - Parameters to be optimized: An array of the size [number of control points, dimension of the curve].
    - Performance measurement: The mean square error between the fitted B-spline and the original curve.
- **Get a fixed scale from the sketch interface:** Like what Min said, users will use similar scales in one drawings. Therefore, it might be feasible to obtain a fixed scale from the sketch interface for normalize the curve.
- **Smooth the curve to be re-sampled:** Since the error in computing curvature is a big problem, smoothing the curve before re-sampling it might be helpful.
- **Search for existing work:** Again, look for existing solutions, especially on the strategies determining the number of required control points.