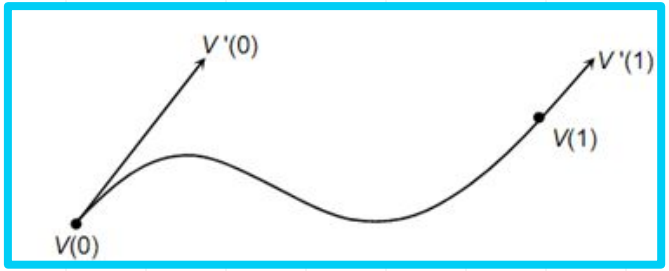
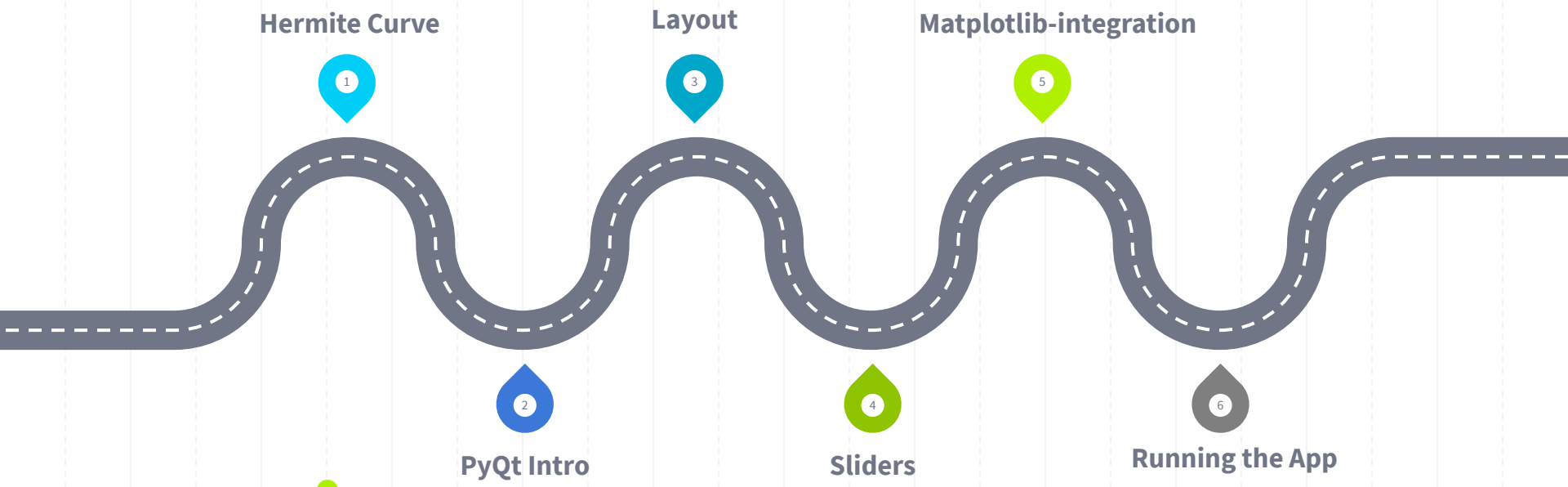


HERMITE CURVE

Made By :
Visharad NA18B102
Lokesh NA18B024
Govind NA18B016

ROADMAP


Hermite Curve
1
PyQt Intro
2
Layout
3
Sliders
4
Matplotlib-integration
5
Running the App
6

Introduction to Hermite Curve

A cubic Hermite spline or cubic Hermite interpolator is a spline where each piece is a third-degree polynomial specified in Hermite form, that is, by its values and first derivatives at the end points of the corresponding domain interval.

A Hermite Curve is made by fitting:

- ➔ The Endpoints (P1, P2)
- ➔ The 1st Derivative at the Endpoints (R1, R2)

The 2 forms of Hermite Curves are as follows:- Algebraic form and Matrix form

Algebraic form is given by :

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$

$$z(t) = a_z t^3 + b_z t^2 + c_z t + d_z$$

This algebraic for is reduced in vector form as **p(t)= at^3 + bt^2 + ct + d**

Where **t** varies from 0 to 1.

MATRIX FORM

```
def update_plot(self):

    p1 = self.s1.value()
    p2 = self.s2.value()
    t = np.loadtxt('data.csv', delimiter=',')
    r1 = self.s3.value()
    r2 = self.s4.value()

    s = (2*(t**3) - 3*(t**2) + 1)*p1 + (-2*(t**3) + 3*(t**2))*p2 + (t**3 - 2*(t**2) + t)*r1 + (t**3 - t**2)*r2
```

- The [a b c d] matrix formed is known as Hermite basis transformation matrix. Equation **p(t)= at^3 + bt^2 + ct + d** can be written in matrix form as: **$p(t)= [t^3\ t^2\ t\ 1] * [a\ b\ c\ d]^T$**
- After applying end conditions that is the values of tangents and points are known we get the following equation for the Hermite Curve as:

  $H(t) = (2t^3-3t^2 + 1)P1 + (-2t^3 + 3t^2)P2 + (t^3-2t^2+t)R1 + (t^3-t^2)R4$, where **t** varies from 0 to 1.
- Input for t is taken from the 'data.csv' file.
- Values for the points P1, P2, R1 & R2 are assigned using **value()** function which gives the value of the slider.

PyQt5

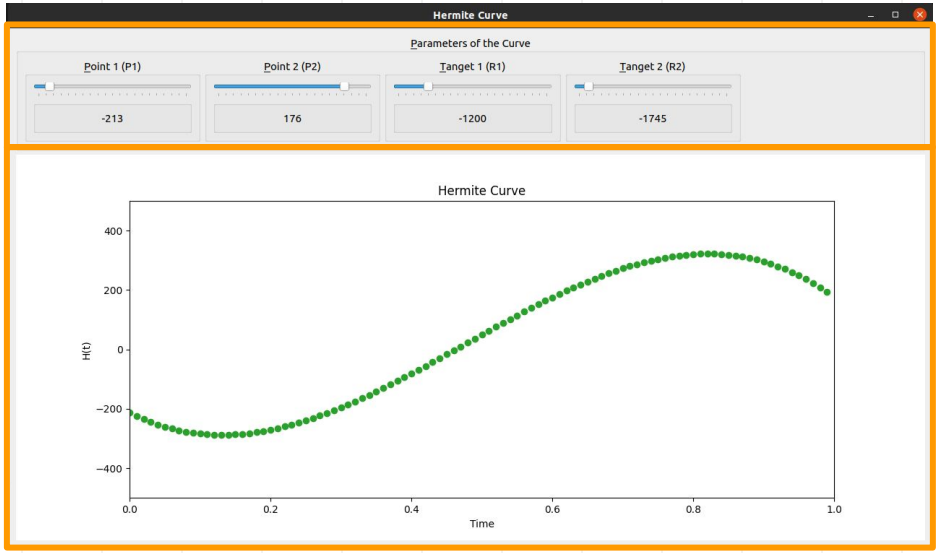
- PyQt5 is a library that can be used to create graphical user interfaces (GUI).
- In our project, we'll be using predefined classes and widgets available in PyQt5 to create the GUI for our program.

We've made the use of following widgets:

`QGridLayout()`, `QGroupBox()`, `QLabel()`, `QSlider()`, `QPushButton()`, `QHBoxLayout()`, `QApplication()`, `QVBoxLayout()`, `QWidget()` imported from `QtWidgets`, and `Qt` imported from `Qt.core()`

```
from PyQt5.QtWidgets import QGridLayout, QGroupBox, QLabel, QSlider, QPushButton, QHBoxLayout, QApplication, QVBoxLayout, QWidget
from PyQt5.QtCore import Qt
```

Layout




```python
def init_ui(self):
    #visual box for the buttons and the graph
    grid_box = QGridLayout()
    grid_box.addWidget(self.create_grp_coeff())
    self.canvas = Figurecanvas(plt.Figure(figsize = (15,6)))
    grid_box.addWidget(self.canvas)
    self.insert_plot()

    self.setLayout(grid_box)
    self.setWindowTitle("Hermite Curve")
    self.show()
```

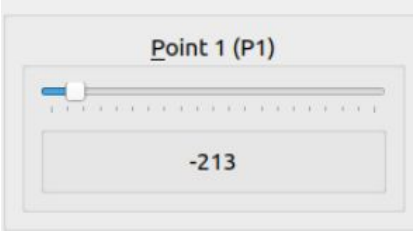
This layout is created using QGridLayout() widget.

In Which two another widgets are added namely create_grp_coeff() and canvas to plot graph.

Code of slider for Point (P1) :

```
def grp_1(self):
    #slider 1 --->
    self.s1 = QSlider(Qt.Horizontal)
    self.s1.setMinimum(-250)
    self.s1.setMaximum(250)
    self.s1.setValue(0)
    self.s1.setTickInterval(25)
    self.s1.setTickPosition(QSlider.TicksBelow)
    self.s1.valueChanged.connect(self.update_label_1)

    self.label1 = QLabel('0', self)
    self.label1.setAlignment(Qt.AlignCenter | Qt.AlignVCenter)
    self.label1.setMinimumWidth(80)

    self.s1.valueChanged.connect(self.update_plot)

    grp_box_1 =QGroupBox("&Point 1 (P1)")
    grp_box_1.setAlignment(Qt.AlignHCenter)
    h_box_1 = QVBoxLayout()
    h_box_1.addWidget(self.s1)

    grp_label_1 = QGroupBox()
    H_box_label_1 = QVBoxLayout()
    H_box_label_1.addWidget(self.label1)

    grp_label_1.setLayout(H_box_label_1)
    grp_box_1.setLayout(h_box_1)
    h_box_1.addWidget(grp_label_1)
    return grp_box_1
```

Sliders

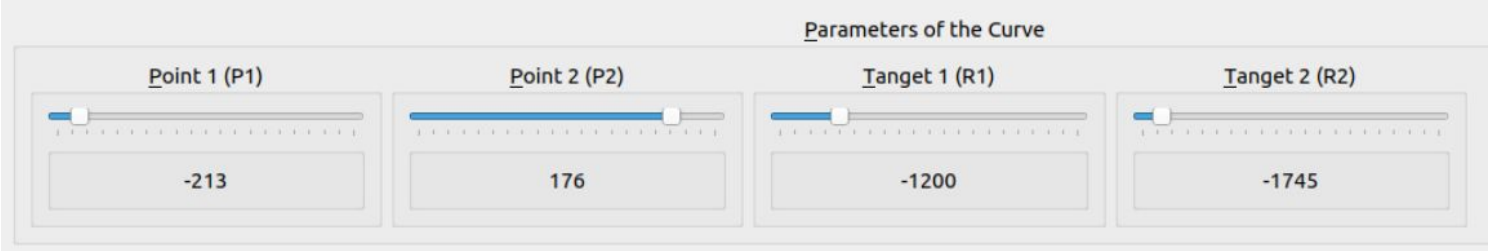


- Sliders are made using the using the Widget QSlider from the class QWidgets.
- Sliders are connected to the function update_Label_(i) function which defines the value of the slider with the help of the label attached to it.
- Label is made using the Widget QLabel.
- Also sliders are connected to the function update_plot which updates the plot when the slider is moved.
- Sliders are defined in a vertical box adding slider on the top of the label.
- Similarly all the sliders are defined and then added together in a horizontal box defined using QHBoxLayout().

All Sliders together

```
def create_grp_coeff(self):
    group_box = QGroupBox("&Parameters of the Curve")
    group_box.setAlignment(Qt.AlignHCenter)
    H_box = QHBoxLayout()
    H_box.setAlignment(Qt.AlignCenter)
    H_box.addWidget(self.grp_1())
    H_box.addWidget(self.grp_2())
    H_box.addWidget(self.grp_3())
    H_box.addWidget(self.grp_4())
    H_box.addStretch()
    group_box.setLayout(H_box)
    return group_box
```

- Using QGroupBox() we define group_box layout to add a H_box defined using QHBoxLayout()
- In the H_Box we added all the slider widgets
- This Layout looks like this



Matplotlib Integration

```
def init_ui(self):
    #visual box for the buttons and the graph
    grid_box = QGridLayout()
    grid_box.addWidget(self.create_grp_coeff())
    self.canvas = Figurecanvas(plt.Figure(figsize = (15,6)))
    grid_box.addWidget(self.canvas)
    self.insert_plot()

    self.setLayout(grid_box)
    self.setWindowTitle("Hermite Curve")
    self.show()
```

```
def insert_plot(self):
    self.ax = self.canvas.figure.subplots()
    self.ax.set_xlim([0,1])
    self.ax.set_ylim([-500,500])
    self.ax.set_title("Hermite Curve")
    self.ax.set_xlabel("Time")
    self.ax.set_ylabel("H(t)")
    self.graph = None
```

- Using Figurecanvas, canvas for for Hermite is defined. Then widget is added in the grid_box defined using QGridLayout().
- insert_graph() function defines the basic properties of the graph such as xlimit, ylimit, title and labels for the axes.
- Self.graph is "**None**" initially but it plots the graph once the values of the parameters are changed through the sliders

Exception Handling:

```
try:
    p1 = int(p1)
except ValueError:
    p1 = 0

try:
    p2 = int(p2)
except ValueError:
    p2 = 0

try:
    r1 = int(r1)
except ValueError:
    r1 = 0

try:
    r2 = int(r2)
except ValueError:
    r2 = 0

if self.graph:
   self.graph.remove()

self.graph = self.ax.scatter(t,s)
self.canvas.draw()
```

Running Application

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    a_window = Window()
    a_window.init_ui()
    sys.exit(app.exec_())
```

- Using QApplication an app is declared and a_window object is defined
- Using init_ui() GUI for graph is displayed.
- App can be run by running the command in the terminal : $ python3 app.py
- Github Link : Here

Thank you!