Report Lab 7

Nguyen Tien Duc - ITITIU18029

December 20, 2019

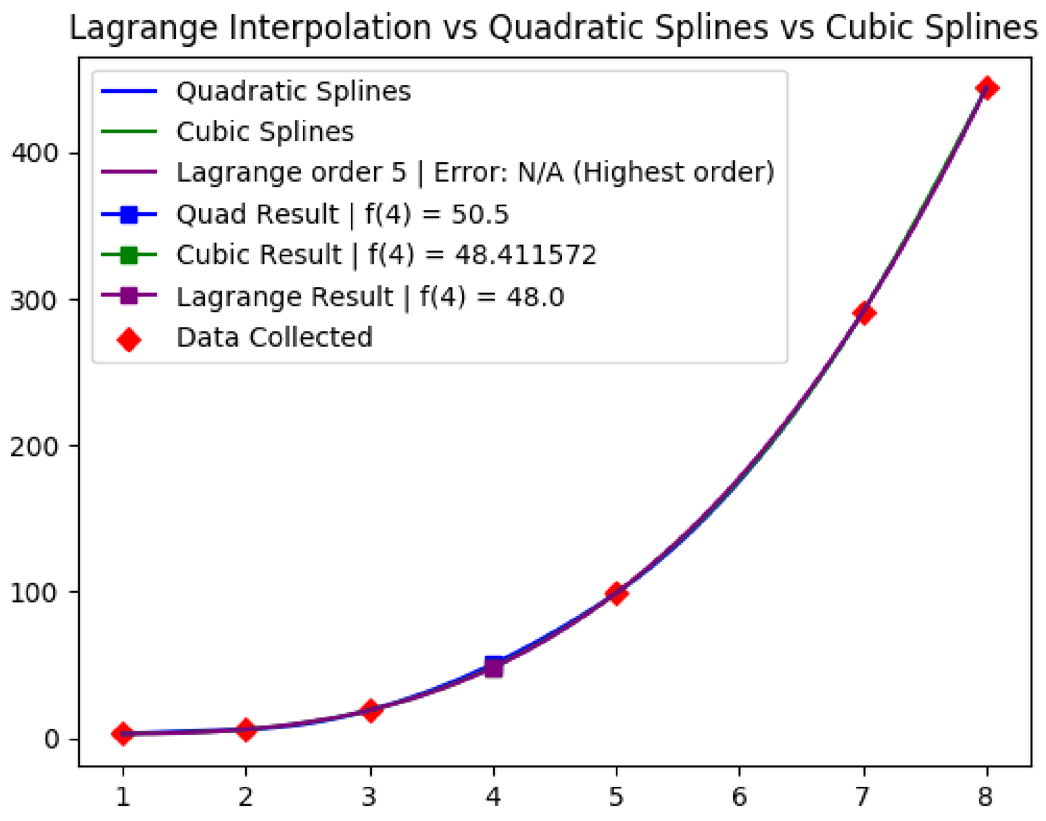
1/ Lagrange's interpolating polynomial

Code

```
import scipy.linalg
from matplotlib import pyplot as plt
from numpy import arange


class DataPoint:
  def __init__(self, x, y):
    self.x = x
    self.y = y


def spline_of_order(order, data):
  n = len(data) - 1
  n_unknowns = order + 1
  total_unknowns = n_unknowns * n
  left_matrix = [[0 for i in range(total_unknowns)] for j in range(total_unknowns)]
  right_matrix = [0 for i in range(total_unknowns)]
  row = 1
  col = 0
  data_index = 0
  while row < total_unknowns * 2 / n_unknowns:
    for i in range(2):
      right_matrix[row + i] = data[data_index + i].y
      for j in range(n_unknowns):
        left_matrix[row + i][col + j] = data[data_index + i].x ** (n_unknowns - j - 1
                                        )
    data_index += 1
    row += 2
    col += n_unknowns
  for u in range(1, order):
    col = 0
    data_index = 1
    for v in range(n - 1):
      for i in range(2):
        for j in range(n_unknowns):
          if u == 1:
            left_matrix[row][col + j + i * n_unknowns] = \
              data[data_index].x ** (n_unknowns - j - 2) * (n_unknowns - j - 1) * (-1
                                        ) ** i
          else:
            left_matrix[row][col + j + i * n_unknowns] = \
              data[data_index].x ** (n_unknowns - j - 3) * (n_unknowns - j - 1) * (
                                            n_unknowns - j - 2) * (-1) ** i
      data_index += 1
      row += 1
      col += n_unknowns
  if order == 3:
    left_matrix[0][0] = 6 * data[0].x
    left_matrix[0][1] = 2
    left_matrix[-1][-4] = 6 * data[-1].x
    left_matrix[-1][-3] = 2
    return left_matrix, right_matrix
  elif order == 2:
    for row in left_matrix:
      del row[0]
    return left_matrix[1:], right_matrix[1:]


def graph_spline_order(order, data, coefficients, spline_x, spline_y, values):
  for i in range(len(data)):
    min_idx = i
    for j in range(i + 1, len(data)):
      if data[min_idx].x > data[j].x:
        min_idx = j
    data[i], data[min_idx] = data[min_idx], data[i]
  if order == 2:
    coefficients = scipy.insert(coefficients, 0, [0])
  c_index = 0
  for i in range(len(data) - 1):
    for x in arange(data[i].x, data[i + 1].x, 0.001):
      y = 0
      for k in range(order + 1):
        y += coefficients[c_index + k] * x ** (order - k)
      values.append(DataPoint(x, y))
      spline_x.append(x)
      spline_y.append(y)
    c_index += order + 1


def value_at(target, values):
  for val in values:
    if abs(val.x - target) < 0.001:
      return val.y


def diff(data):
  if len(data) == 1:
    return data[0].y
  elif len(data) == 2:
    numerator = data[1].y - data[0].y
    denominator = data[1].x - data[0].x
    return numerator / denominator
  else:
    numerator = diff(data[1:]) - diff(data[:-1])
    denominator = data[-1].x - data[0].x
    return numerator / denominator
```

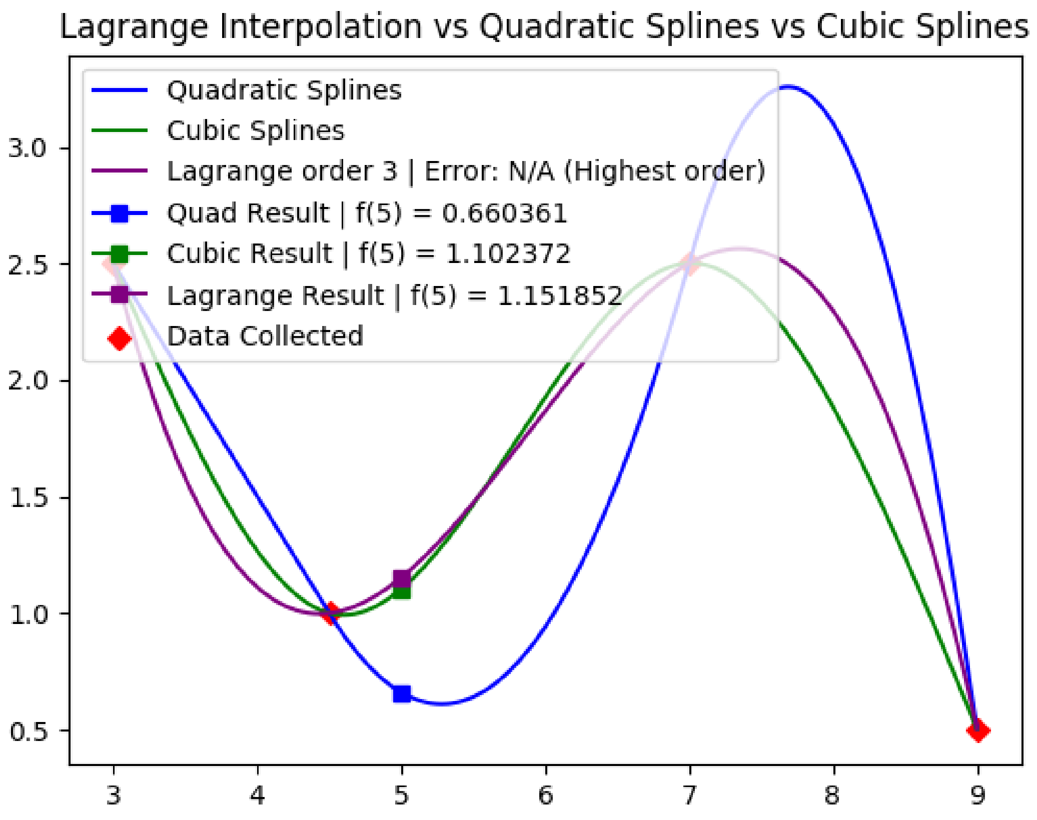
```python
def lagrange_interpolation_error(data, target):
  product = 1
  for i in range(len(data) - 1):
    product *= target - data[i].x
  return abs(diff(data) * product)


def lagrange_interpolation_value_by_degree(
  order, data, target):
  result = 0
  if order < len(data) - 1:
    limit = order + 1
    error = str(lagrange_interpolation_error(data[:limit + 1], target).__round__(6))
  else:
    limit = len(data)
    error = "N/A (Highest order)"
  for i in range(limit):
    L = 1
    for j in range(limit):
      if j != i:
        L *= (target - data[j].x) / (data[i].x - data[j].x)
    result += L * data[i].y
  return result, error


def lagrange_interpolation_graph_by_degree(order, data, value_graph_x, value_graph_y)
                                            :
  x_min, x_max = data[0].x, data[-1].x
  for k in range(len(data)):
    if data[k].x > x_max:
      x_max = data[k].x
    if data[k].x < x_min:
      x_min = data[k].x
  for k in arange(x_min, x_max, 0.001):
    value_graph_x.append(k)
    value_graph_y.append(lagrange_interpolation_value_by_degree(order, data, k)[0])


def scatter_data(data, data_graph_x, data_graph_y):
  for k in range(len(data)):
    data_graph_x.append(data[k].x)
    data_graph_y.append(data[k].y)


# data_input = [
#   DataPoint(3.0, 2.5),
#   DataPoint(4.5, 1.0),
#   DataPoint(7.0, 2.5),
#   DataPoint(9.0, 0.5),
# ]
# chosen_target = 5

# data_input = [
#   DataPoint(0, 2),
#   DataPoint(1, 5.4375),
#   DataPoint(2.5, 7.3516),
#   DataPoint(3, 7.5625),
#   DataPoint(4.5, 8.4453),
#   DataPoint(5, 9.1875),
#   DataPoint(6, 12)
# ]
# chosen_target = 3.5

data_input = [
  DataPoint(1, 3),
  DataPoint(2, 6),
  DataPoint(3, 19),
  DataPoint(5, 99),
  DataPoint(7, 291),
  DataPoint(8, 444)
]
chosen_target = 4

scatter_data_x = []
scatter_data_y = []
plot_quad_spline_x = []
plot_quad_spline_y = []
quad_splines_value = []
plot_cubic_spline_x = []
plot_cubic_spline_y = []
cubic_splines_value = []
plot_lagrange_x = []
plot_lagrange_y = []
lagrange_value = []
lagrange_order = 5

quad = spline_of_order(order=2, data=data_input)
cubic = spline_of_order(order=3, data=data_input)
lagrange = \
  lagrange_interpolation_value_by_degree(order=lagrange_order, data=data_input,
                                             target=chosen_target)
quad_coefficients = scipy.linalg.solve(quad[0], quad[1])
cubic_coefficients = scipy.linalg.solve(cubic[0], cubic[1])

scatter_data(data_input, scatter_data_x, scatter_data_y)
graph_spline_order(2, data_input, quad_coefficients, plot_quad_spline_x,
                                             plot_quad_spline_y, quad_splines_value)
graph_spline_order(3, data_input, cubic_coefficients, plot_cubic_spline_x,
                                             plot_cubic_spline_y, cubic_splines_value)
lagrange_interpolation_graph_by_degree(
  order=lagrange_order, data=data_input, value_graph_x=plot_lagrange_x, value_graph_y
                                             =plot_lagrange_y
)

quad_result = value_at(chosen_target, quad_splines_value).__round__(6)
cubic_result = value_at(chosen_target, cubic_splines_value).__round__(6)
lagrange_result = lagrange[0].__round__(6)

plt.title("Lagrange Interpolation vs Quadratic Splines vs Cubic Splines")
plt.scatter(scatter_data_x, scatter_data_y, marker="D", c="red", label="Data
                                             Collected")
plt.plot(plot_quad_spline_x, plot_quad_spline_y, c="blue", label="Quadratic Splines")
plt.plot(plot_cubic_spline_x, plot_cubic_spline_y, c="green", label="Cubic Splines")
plt.plot(
  plot_lagrange_x, plot_lagrange_y, c="purple",
  label="Lagrange order " + str(lagrange_order) + " | Error: " + lagrange[1])
plt.plot(
  chosen_target, quad_result, marker="s", c="blue",
    label="Quad Result | f(" + str(chosen_target) + ") = " + str(quad_result))
plt.plot(
    chosen_target, cubic_result, marker="s", c="green",
    label="Cubic Result | f(" + str(chosen_target) + ") = " + str(cubic_result))
plt.plot(
    chosen_target, lagrange_result, marker="s", c="purple",
    label="Lagrange Result | f(" + str(chosen_target) + ") = " + str(lagrange_result))
plt.legend()
plt.savefig("Result0.png")
plt.show()
```

Running



2/ Analyze and Comments

As the data given in Lab 7 is not good enough to demonstrate the differences between lagrange interpolation, quadratic spline, and cubic spline, I used the data given in the lesson's example and it clearly distinguished the curves.



Overall, Lagrange interpolation gives us the best curve. The next closest curve is the curve created using cubic splines. The curve created with quadratic splines, in most cases, is not enough to describe the data accurately.
In conclusion, we have to agree about 3 things:

- Newton and Lagrange interpolating polynomials better describe the data compared to splines in most cases,
- However, in cases when Newton and Lagrange does not work, splines could be used and it will give a fair enough result.
- The higher order of the polynomials, for both interpolation and splines, always guarantee us a more accurate model.