Numerical Analysis Home Assignment 4

Barak-Nadav Diker

March 6, 2024

Contents

1 Question 1

We have the following data on the number of unemploy in a certain city Evaluate using interpolation the number of unemploy at the year 1955

year	1951	1961	1971	1981
unemployed	35	42	58	84

1.1 Solution Newton interpolation

In order to use newton interpolation we should calculate the divided difference

Here is a sample code on how to calculate it , It's a simple recursion by definition

```
def div_diff(x , y):
    if len(x) == 1:
        return y[0]
    return (div_diff(x[1:],y[1:]) - div_diff(x[:-1],y[:-1]))/(x[-1] -
    ↪  x[0])
#example
#return div_diff([8.1,8.3],[16.9446 , 17.56492])
```

```
None
```

```
def div_diff(x , y):
    if len(x) == 1:
        return y[0]
    return (div_diff(x[1:],y[1:]) - div_diff(x[:-1],y[:-1]))/(x[-1] -
    ↪  x[0])
#example
#return div_diff([8.1,8.3],[16.9446 , 17.56492])
x = my_data[0][1:] #Get the data from the table
y = my_data[1][1:] # Get the data from the table
newton_coeff = [div_diff(x[:i], y[:i]) for i in range(1,len(x))]
return newton_coeff
```

```
# newton_coeff = | 35 | 0.7 | 0.045 |
# [y_0] = 35 , [y_0,y_1] = 0.7 , [y_0,y_1,_2] = 0.045
```

Here I am creating the function interpolation.
Please note that interpolation is function from $\mathbb{R}$ to $\mathbb{R}$ which is polynomial

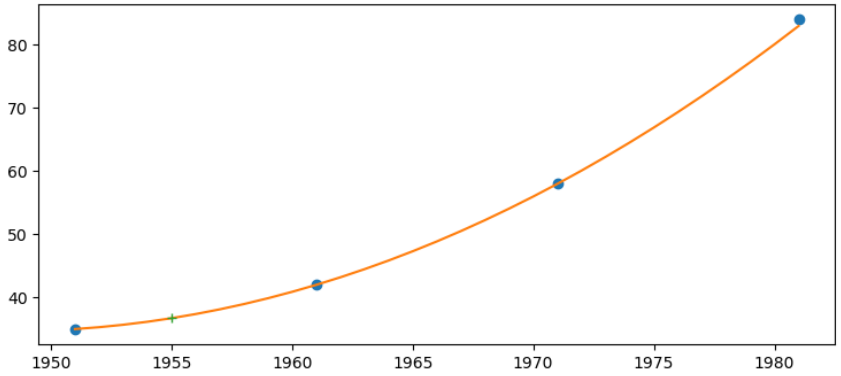
```
x_d = my_data[0][1:] # x_d =  [1951 ,1961 , 1971 ,1981]
interpolation = lambda x: coeff[0]*1 + coeff[1]*(x-x_d[0]) +
↪  coeff[2]*(x-x_d[0])*(x-x_d[1])
```

```
#return interpolation(1955)
#36.72
```

1.2 Visualize the Solution

Here is some basic code for viewing the function

```
import numpy as np
import matplotlib.pyplot as plt
x_d = my_data[0][1:] # x_d =  [1951 ,1961 , 1971 ,1981]
interpolation = lambda x: coeff[0]*1 + coeff[1]*(x-x_d[0]) +
↪  coeff[2]*(x-x_d[0])*(x-x_d[1])
plt.rcParams["figure.figsize"] = [7.50, 3.50]
plt.rcParams["figure.autolayout"] = True
x = my_data[0][1:] # x =  [1951 ,1961 , 1971 ,1981]
y = my_data[1][1:] # y = [ 35 , 42, 58, 84 ]
xnew = np.arange(1951, 1982 ) # xnew = [1951 , 1952 ,... , 1981]
ynew = interpolation(xnew)   # use interpolation
plt.plot(x, y, 'o', xnew, ynew, '-',[1955] ,[36.72] , '+')
plt.savefig("question1_a.png")
return "question1_a.png"
```



2 Question 2

Show that given a linear function i.e $y = ax + b$. You'll need at most 1 iteration of newton-rhapson to find the root

2.1 Solution

Let $x \in \mathbb{R}$ if $x = \frac{-b}{a}$ we are done

else , let $x \neq \frac{-b}{a}$ run 1 iteration of newton-rhapson and show that $x_1 = \frac{-b}{a}$

$$x_1 = x - \frac{y(x)}{y'(x)}$$

$$x_1 = x - \frac{ax + b}{a}$$

$$x_1 = x - x + \frac{-b}{a}$$

$$x_1 = \frac{-b}{a}$$

□

3 Question 3

Given the function f and let $Q(x)$ be the interpolation of f at
$(x_0, x_1, ..., x_n)$
and let $P(x)$ be the newton interpolation of points $(x_0, x_1, ..., x_n, t, t)$ for arbitrary $t \in \mathbb{R}$

Writing $Q(x)$ explicitly we'll have

$$Q(x) = [y_0] + [y_0, y_1](x - x_0) + \cdots + [y_0, \ldots, y_n](x - x_0)(x - x_1) \cdots (x - x_{n-1}) \tag{1}$$

and writing $P(x)$ explicitly grants us

$$\begin{aligned} P(x) = [y_0] + [y_0, y_1](x - x_0) + \cdots + [y_0, \ldots, y_n](x - x_0)(x - x_1) \cdots (x - x_{n-1}) + \\ [y_0, \ldots, y_n, t](x - x_0)(x - x_1) \cdots (x - x_{n-1})(x - t) + \\ [y_0, \ldots, y_n, t, t](x - x_0)(x - x_1) \cdots (x - x_{n-1})(x - t)^2 \end{aligned}$$

Clearly we can see that

$$\begin{aligned} P(x) - Q(x) = &[y_0, \ldots, y_n, t](x - x_0)(x - x_1) \cdots (x - x_{n-1})(x - t) + \\ &[y_0, \ldots, y_n, t, t](x - x_0)(x - x_1) \cdots (x - x_{n-1})(x - t)^2 \end{aligned}$$

4 Question 4

Evaluate the integral

$$\int_0^1 \frac{1}{1+x^3} dx$$

using the trapezoidal rule we have the following formula

$$\int_a^b f(x)dx = \frac{b-a}{n}\left(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(a+\frac{b-a}{n}i)\right)$$

Using python code we have

```
def f(x):
    return 1/(1+x**3)
a = 0 ; b = 1 ; n =4
return (b - a )/n *( (f(b)-f(a))/2 + sum([f(a+(b-a)*i/n) for i in
↪  range(1,n-1) ]) )
```

```
0.4058760683760684
```

```
def f(x):
    return 1/(1+x**3)
a = 0 ; b = 1 ; n =40
return (b - a )/n *( (f(b)-f(a))/2 + sum([f(a+(b-a)*i/n) for i in
↪  range(1,n-1) ]) )
```

```
0.7976353006745376
```

Happily we can see that after 40 iteration we are really close to the true solution which is 0.83525

$$relativeerror = \frac{\Delta y}{y} = \frac{0.83525 - 0.7976353006745376}{0.83525} = 0.045$$

5 Question 5

Write a program in C++ that can calculate the integral from 0 to 1 divided by n seqments

```cpp
#include <iostream>
double f(double x){ // Example function , one can pick anyfunction he
↪  wants
    return 1/(x*x*x + 1);
}
double integral_calculator(int n , double a = 0 , double b=1){
    double my_sum = 0;
   for(int i =1; i < n-1; i++){
       my_sum += f(a+(b-a)*i/n);
   }
    return (b-a)/n*((f(b)+f(a))/2 + my_sum);
}
int main() {
    std::cout << "The answer is :" << integral_calculator(19);
    return 0;
}
```

```
The answer is :0.80703
```

6 Question 6

Write a Program in C++ that given n+1 point on $\mathbb{R}^2$ Creates interpolation polinom

6.1 Answer a - lagrange interpolation

Here is code snippet that implement lagrange interpolation

```cpp
#include <iostream>
#include <vector>
#include <tuple>
#include <utility>
using namespace std;
double lagrange_basis(double x,int j,vector<tuple<double,double>    > data
↪  ){
    double p = 1;
    auto [x_j,y_j] = data[j];
    data.erase(data.begin()+j);
        for(auto [x_i,y_i] : data){
            p *= (x-x_i)/(x_j-x_i);
    }
    data.insert(data.begin()+j,make_tuple(x_j,y_j));
    return p;
}
double lagrange_interpolation(double x ,vector<tuple<double,double>    >
↪  data){
    double sum_lagrange = 0;
    int j = 0;
    for(auto [x_i,y_i] :data){
        sum_lagrange += y_i*lagrange_basis(x,j,data);
        j++;
    }
    return sum_lagrange;
}
int main(){
    vector<tuple<double,double>    > data ;
    data.push_back(make_tuple(1,2));
    data.push_back(make_tuple(3,4));

      //lagrange_basis(30,0,data);
        cout << "Given the x = 2 the interpolation is "
        ↪  <<lagrange_interpolation(2,data);
}
```

```
Given the x = 2 the interpolation is 3
```

But It makes sense that the point (2,3) is on the interpolation of the data (1,2) , (3,4) because it's a simple line !

6.2 Answer B - Use Newton Interpolation

Implement via the programming language c++ the newton interpolation

```cpp
// Calculate the finite difference function
#include <iostream>
#include <vector>
using namespace std;
double div_diff(vector<double>x, vector<double>y){
    if (x.size() == 1)
        return y[0];
    vector<double>::const_iterator start = x.begin() + 1;
    vector<double>::const_iterator end  = x.end();
    vector<double> x_from_1(start, end);
    start = y.begin() +1;
    end = y.end();
    vector<double> y_from_1(start, end);
    start = x.begin();
    end = x.end() - 1;
    vector<double> x_no_last_element(start, end);
    start = y.begin();
    end = y.end() - 1;
    vector<double> y_no_last_element(start, end);
    return ((div_diff(x_from_1 , y_from_1) -
    ↪  div_diff(x_no_last_element,y_no_last_element)))/(x.back() -
    ↪  x.front());
    }
int main(){
    // example of usage
    vector <double> x = {8.1 , 8.3};
    vector <double> y = {16.9446 , 17.56492};

    std::cout << "The div different is " << div_diff(x,y);
    return 1;
}
```

```
The div different is 3.1016
```

```python
def div_diff(x , y):
    if len(x) == 1:
        return y[0]
    return (div_diff(x[1:],y[1:]) - div_diff(x[:-1],y[:-1]))/(x[-1] -
    ↪  x[0])
#example
return div_diff([8.1,8.3],[16.9446 , 17.56492])
```

```
3.1015999999999813
```

Clearly in python it's more elegant In order to calculate the the final newton interpolation I'll use the same code from question just in c++

```cpp
#include <vector>
#include <iostream>
#include <cmath>
using namespace std;
double newton_interpolation(double x ,vector<double> x_arr ,
↪  vector<double>div_diff_arr){
  double acc = 0;
  for (int i =0;i < div_diff_arr.size() ;i++){
      double mal = div_diff_arr[i];
      for(int j =0 ; j < i ; j++){
          mal *=  std::pow((x-x_arr[j]),j) ;
        }
          acc += mal;
}
  return acc;
}
int main(){
    vector <double > x = {1951 , 1961 , 1971 , 1981}; // from question 1
    vector <double > div_diff_arr = {35 , 0.7 , 0.045}; // from question 1
    std::cout << "Interpolatoin is " << newton_interpolation(1955 , x ,
    ↪  div_diff_arr);
}
```

```
Interpolatoin is 35.43
```

Please note that the result 35.43 is close to 36.72 , now I'll combine the 2 function together to have entire pipe

The following code looks the same but it combines both of the functions

```cpp
#include <vector>
#include <iostream>
#include <cmath>
using namespace std;

double div_diff(vector<double>x, vector<double>y){
    if (x.size() == 1)
        return y[0];
    vector<double>::const_iterator start = x.begin() + 1;
    vector<double>::const_iterator end  = x.end();
    vector<double> x_from_1(start, end);
    start = y.begin() +1;
    end = y.end();
    vector<double> y_from_1(start, end);
    start = x.begin();
    end = x.end() - 1;
    vector<double> x_no_last_element(start, end);
```

```
    start = y.begin();
    end = y.end() - 1;
    vector<double> y_no_last_element(start, end);
    return ((div_diff(x_from_1 , y_from_1) -
    ↪  div_diff(x_no_last_element,y_no_last_element)))/(x.back() -
    ↪  x.front());
    }

double newton_interpolation(double x ,vector<double> x_arr ,
↪  vector<double>div_diff_arr){
  double acc = 0;
  for (int i =0;i < div_diff_arr.size() ;i++){
      double mal = div_diff_arr[i];
      for(int j =0 ; j < i ; j++){
          mal *=  std::pow((x-x_arr[j]),j) ;
        }
          acc += mal;
}
  return acc;
}
vector <double> subarray(vector <double> arr , auto first , auto last ){
    vector<double> subarr(first , last);
    return subarr;
}
int main(){
    vector <double > x = {1951 , 1961 , 1971 , 1981}; // from question 1
    vector <double > y = {35,42,58,84}; // from question 1
    vector <double> div_diff_arr;
    vector <double>sub_x;
    vector <double> sub_y;
    auto first_x = x.begin();
    auto last_x = x.begin();
    auto first_y = y.begin();
    auto last_y = y.begin();
    for( int i =0 ; i < x.size() ; i++ ){
           first_x = x.begin();
           last_x = x.begin() + i + 1;
           sub_x = subarray(x, first_x, last_x);
           first_y = y.begin();
           last_y = y.begin() + i + 1;
           sub_y = subarray(y,first_y, last_y);
        div_diff_arr.push_back(div_diff(sub_x , sub_y   ));
    }
    std::cout << "Interpolatoin is " << newton_interpolation(1955 , x ,
    ↪  div_diff_arr);
}
```

```
Interpolatoin is 35.174
```

7 Question 7

Calculate the newton interpolation polinom of the function

$$f(x) = x^3$$

using the the set of point

$$(1,1),(2,8),(3,27),(4,64)$$

using the function from question the

x	1	2	3	4	5
y	1	8	27	64	125

I'll calculate the coefficient

$$[y_0],[y_0,y_1],[y_0,y_1,y_2],[y_0,y_1,y_2,y_3]$$

```
def div_diff(x , y):
    if len(x) == 1:
        return y[0]
    return (div_diff(x[1:],y[1:]) - div_diff(x[:-1],y[:-1]))/(x[-1] -
    ↪  x[0])
#example
#return div_diff([8.1,8.3],[16.9446 , 17.56492])
x = my_data[0][1:] #Get the data from the table
y = my_data[1][1:] # Get the data from the table
newton_coeff = [div_diff(x[:i], y[:i]) for i in range(1,len(x))]
return newton_coeff
```

$$1 \quad 7.0 \quad 6.0 \quad 1.0$$

which is exactly

$[y_0]$	$[y_0 , y_1]$	$[y_0 , y_1 , y_2]$	$[y_0 , y_1 , y_2 , y_3]$
1	7.0	6.0	1.0

From the formula of newton interpolation

$$N(x) = [y_0] + [y_0,y_1](x-x_0) + \cdots + [y_0,\ldots,y_k](x-x_0)(x-x_1)\cdots(x-x_{k-1}) \tag{2}$$

$$N(x) = 1 + 7(x-1) + 6(x-1)(x-2) + 1(x-1)(x-2)(x-3) \tag{3}$$

```
from sympy import symbols , init_printing , simplify
x = symbols('x')
init_printing(use_unicode=True)
return simplify(1+7*(x-1)+6*(x-1)*(x-2) + 1*(x-1)*(x-2)*(x-3))
```

```
x**3
```

Which means that our interpolation simplify to x^3
Note that the error from the function $f(x) = x^3$ is

$$E = |f(x) - N(x)| = |x^3 - x^3| = 0$$

which means that for all $x \in \mathbb{R}$ the error is 0