Examination Computer Algebra and LaTeX

Summer Semester 2020

Debojit Das

Matriculation Number: 51866
Seminar Group : MA19w1-B

University of Applied Sciences Mittweida
August 20, 2020

Contents

Chapter 1
Polynomial Interpolation

Abstract The first thing to realize when fitting a polynomial (exactly) to discreet *data* is that the polynomial must have as many parameters (i.e. coefficients) we are free to vary as there are conditions (i.e. points) in our *dataset.*

Thus, for example, to run a polynomial through all points of the table, it needs to have atleast 4 coefficients (i.e. degree 3) such that,

$$c_0 + c_1 x_i + c_2 x_i^2 + c_3 x_i^3 + c_4 x_i^4 = y_i \tag{1.1}$$

where i = 1, 2, ... 5. The resulting 4 equations for 4 unknowns are linear, having a unique solution (unless there are two or more identical values in the x row). Later on we will solve such system of linear equations but the general technique is quite lengthy and tedious

1.1 Newton Interpolation

Polynomial interpolation is a method of estimating values between known data points. When graphical data contains a gap, but data is available on either side of the gap or at a few specific points within the gap, an estimate of values within the gap can be made by interpolation.

The simplest method of interpolation is to draw straight lines between the known data points and consider the function as the combination of those straight lines. This method, called linear interpolation, usually introduces considerable error. A more precise approach uses a polynomial function to connect the points. A polynomial is a mathematical expression comprising a sum of terms, each term including a variable or variables raised to a power and multiplied by a coefficient. The simplest polynomials have one variable. Polynomials can exist in factored form or written out in full.

1.1.1 General Explanation / Procedure

The Lagrange interpolation relies on the n+1 interpolation points $\{x_i,\ y_i = f(x_i),\ i = 0, \cdots, n\}$ all of which need to be available to calculate each of the basis polynomials $l_i(x)$ If additional points are to be used when they become available, all basis polynomials need to be recalculated.

In comparison, in the Newton interpolation, when more data points are to be used, additional basis polynomials and the corresponding coefficients can be calculated, while all existing basis polynomials and their coefficients remain unchanged. Due to the additional terms, the degree of interpolation polynomial is higher and the approximation error may be reduced (e.g., when interpolating higher order polynomials).

Specifically, the basis polynomials of the Newton interpolation are calculated as below:

$$n_0(x) = 1, \quad n_i(x) = \prod_{j=0}^{i-1}(x - x_j), \qquad (i = 1, \cdots, n) \tag{1.2}$$

And the newton interpolated polynomial is constructed:

$$\begin{aligned} N_n(x) &= \sum_{i=0}^{n} c_i n_i(x) = c_0 + \sum_{i=1}^{n} c_i \left(\prod_{j=0}^{i-1}(x - x_j) \right) \\ &= c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots + c_n \prod_{j=0}^{n-1}(x - x_j) \end{aligned} \tag{1.3}$$

When the next data points are available, not used in any of the basis polynomials but it is used for calculating the last coefficient c_n, as shown below. For this nth degree polynomial to pass all n+1 points, it needs to satisfy the following n+1 equations.

$$\begin{cases} y_0 = c_0 \\ y_1 = c_0 + c_1(x_1 - x_0) \\ y_2 = c_0 + c_1(x_2 - x_0) + c_2(x_2 - x_0)(x_2 - x_1) \\ y_3 = c_0 + c_1(x_3 - x_0) + c_2(x_3 - x_0)(x_3 - x_1) + c_3(x_3 - x_0)(x_3 - x_1)(x_3 - x_2) \\ \cdots\cdots\cdots\cdots \\ y_n = c_0 + c_1(x_n - x_0) + c_2(x_n - x_0)(x_n - x_1) + \cdots + c_n \prod_{i=0}^{n-1}(x_n - x_i) \end{cases} \tag{1.4}$$

Now the Newton polynomial interpolation can be written as:

$$N_n(x) = \sum_{i=0}^{n} c_i n_i(x) = \sum_{i=0}^{n} f[x_0, \cdots, x_i] n_i(x) = f[x_0] + \sum_{i=1}^{n} f[x_0, \cdots, x_i] \left(\prod_{j=0}^{i-1} (x - x_j) \right) \tag{1.5}$$

1.1.2 Explanation with example

Say, we have the function, $f(x) = e^{(x+sinx)}$ by a polynomial degree of n = 3, based on the following n + 1 = 4 points:

i	0	1	2	3
x_i	1	2	3	4
$f(x_i)$	2.76	7.65	21.16	58.54

Based on $f[x_i] = f(x_i)$, (i = 0, ... , n), we can find all other divided differences recursively in tabular form as shown below, In general it can be found based on its left neighbour and top left neighbour:

$$f[x_i, \cdots, x_j] = \frac{f[x_{i+1}, \cdots, x_j] - f[x_i, \cdots, x_{j-1}]}{x_j - x_i}$$

x_i	$0th$	$1st$	$2nd$	$3rd$
$x_0 = 1$	$f[x_0] = 2.76$			
$x_1 = 2$	$f[x_1] = 7.65$	$f[x_0, x_1] = 4.89$		
$x_2 = 3$	$f[x_2] = 21.16$	$f[x_1, x_2] = 13.51$	$f[x_0, x_2] = 4.31$	
$x_3 = 4$	$f[x_3] = 54.54$	$f[x_2, x_3] = 37.38$	$f[x_1, x_3] = 12$	$f[x_0, x_3] = 1.28$

Alternatively, They can be represented in the expanded form:

$$c_i = f[x_0, \cdots, x_i] = \sum_{j=0}^{n} \frac{f(x_j)}{\prod_{i=0, i \neq j}^{n} (x_j - x_i)} \tag{1.6}$$

Now the Newton interpolating polynomial can be obtained as:

$$\begin{aligned} N_3(x) &= \sum_{i=0}^{3} c_i n_i(x) \\ &= 2.76 + 4.89(x+1) + 0.643(x+1)(x-0) - 0.663(x+1)(x-0)(x-1) \\ &= 8.73 + 19.88\,x - 10.92\,x^2 + 2.53\,x^3 \end{aligned}$$

1.1.3 Realization of Newton Interpolation with Sage

In the Sage part, we have done the calculation of our Polynomial Interpolation using Newton's procedure, as well as we have calculated the graphical representation.

First, We have written a function which will return $p(x)$. For this reason, we need to calculate $p(x)$ and $dt(x)$. Then, we have used a loop so that we can calculate the value of C, $p(x)$ and $dt(x)$. In the last part of the function we have used the built in command full simplify to simplify the functional value.

Then we have defined our function, xx values and yy values and the points. After that, we recalled our function to simply calculate the values. Later that we plotted the graph of the main function and the interpolated function together using different legend labels. Axes labels, frame, different color and different linestyle was also added.

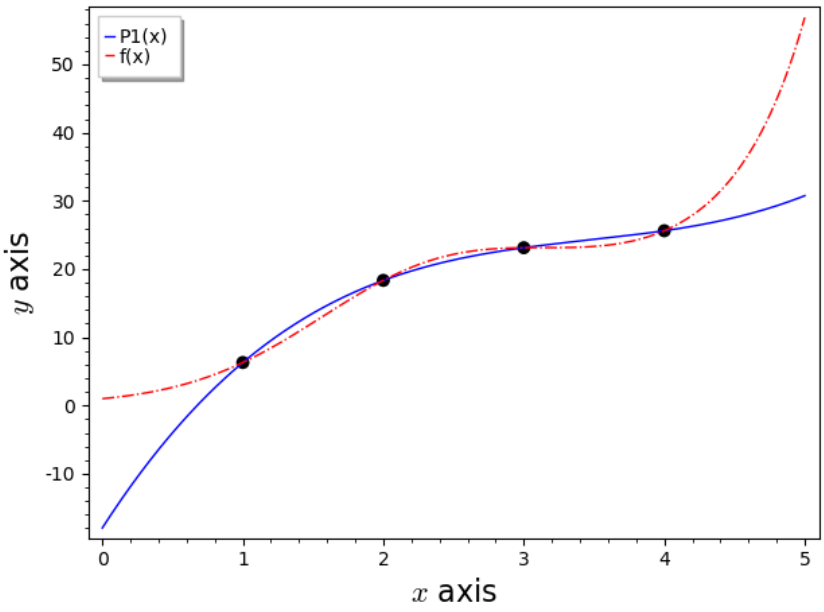

Fig. 1.1 Original and interpolation polynomial functions with different colors in one image.

Then we have calculated for the extended two more points.

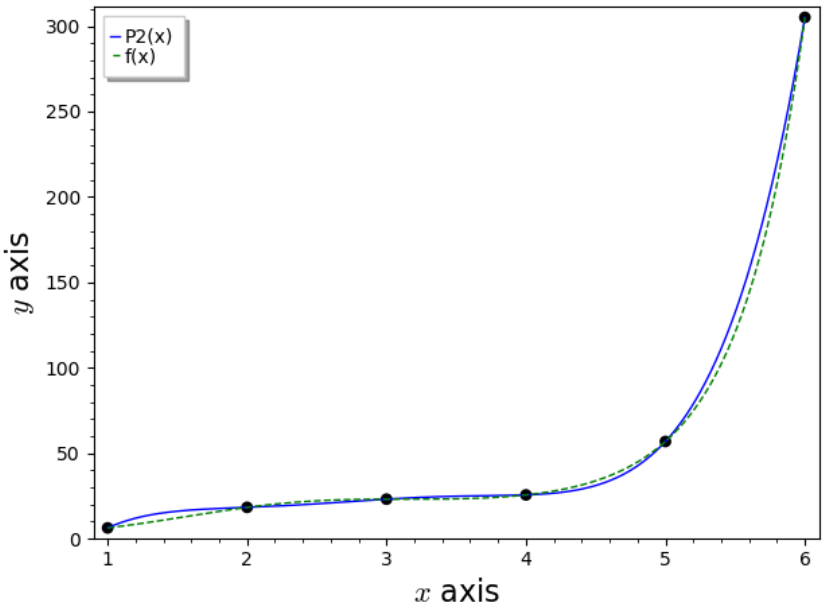

Fig. 1.2 Original and second interpolation polynomial functions with different colors in one image.

Now we will generate a graphics array consisting of 3 rows and 1 column where the first array (position (3; 1)) contains $P_1(x)$ and the original function $f(x)$. The second array (position (2; 1)) contains $P_2(x)$ and the original function $f(x)$. Finally the third one displays the three functions $P_1(x)$, $P_2(x)$ and $f(x)$. Here, we are using different colors, linestyles, label boxes and axis labels.

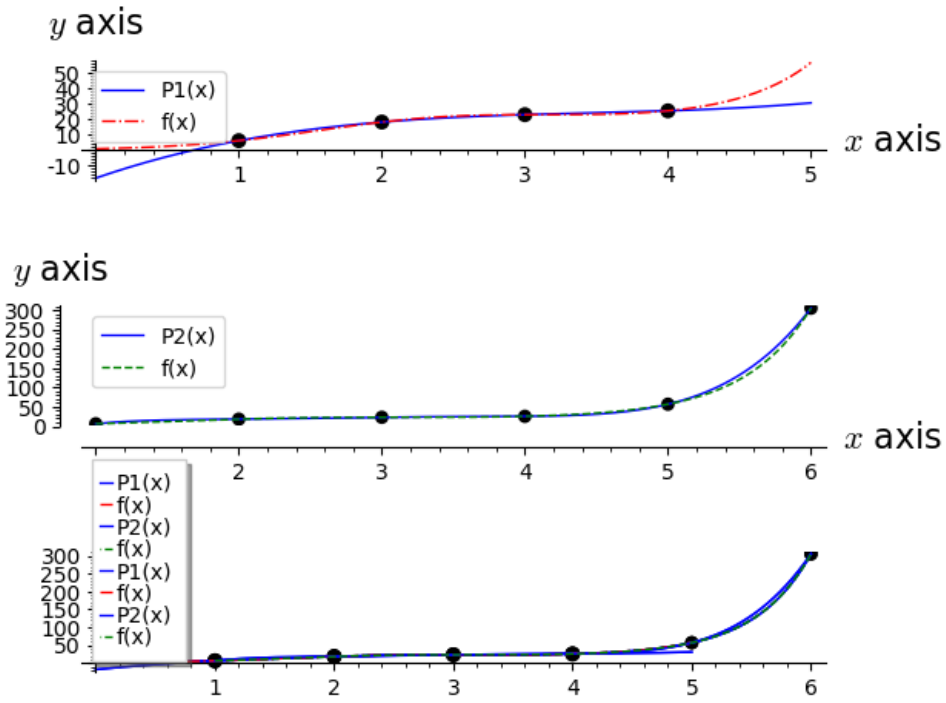

Fig. 1.3 Original, first interpolation function and second interpolation polynomial functions with different colors in one image.

1.2 Divided Differences

From here, we will discuss about the Polynomial Interpolation using Divided Difference. Previously, we established that for each n = 0, 1, ... , there exists a unique polynomial P_n such that, the degree of P_n is at most n. It was shown that P_n can be expressed in the Newton form:

$$P_n(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + \cdots + b_n(x - x_0)\cdots(x - x_{n-1}) \quad (1.7)$$

1.2.1 General Explanation / Procedure

The general form of Newton's divided-difference interpolating polynomial is

$$p_n(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + \cdots + b_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}),$$

where the b_i are recursively computed.

To define the b_i, $i = 0, \ldots, n$, we first define the quantities $f[x_i, x_{i-1}, \ldots, x_{i-j}]$, where $0 \leq j \leq i$.

Base Case 1. $f[x_i] = y_i$ is the y-value associated with x_i.
Base Case 2. The average rate of change from x_{i-1} to x_i is,

$$f[x_i, x_{i-1}] = \frac{y_i - y_{i-1}}{x_i - x_{i-1}} .$$

In other words, we may think of $f[x_i, x_{i-1}]$ as a first derivative approximation.

Recursive Case. Suppose $f[x_i, x_{i-1}, \ldots, x_{i-j+1}]$ and $f[x_{i-1}, x_{i-2}, \ldots, x_{i-j}]$ have both been defined, while the first represents the approximation of a $(j-1)$ th derivative over the interval $[x_i, x_{i-j+1}]$, while the second represents the approximation of a $(j-1)$ th derivative over the interval $[x_{i-1}, x_{i-j}]$. Then $f[x_i, x_{i-1}, \ldots, x_{i-j}]$ is defined as:

$$f[x_i, x_{i-1}, \ldots, x_{i-j}] = \frac{f[x_i, x_{i-1}, \ldots, x_{i-j+1}] - f[x_{i-1}, x_{i-2}, \ldots, x_{i-j}]}{x_i - x_{i-j}} \quad [3]$$

and represents the approximation of a j th derivative over the interval $[x_i, x_{i-j}]$.

1.2.2 Explanation with example

Our main function here is $y = e^{x+sin(x)}$
In order to calculate the interpolation using Divided Difference method first we have to take the inputs os x values and functional values of x. then we need to apply the formula

$$N_n(x) = \sum_{i=0}^{n} c_i n_i(x) = \sum_{i=0}^{n} f[x_0, \cdots, x_i] n_i(x) = f[x_0] + \sum_{i=1}^{n} f[x_0, \cdots, x_i] \left(\prod_{j=0}^{i-1} (x - x_j) \right) \tag{1.8}$$

Then we need to write a nested loop first one from 1 to n and the index of inner loop should be 1 to iteration variable in my case which is k. Then we set the formula. If we print the output, we shall see the result.

1.2.3 Realization of Interpolation based on Divided Difference with Sage

In order to avoid simply listing headings of different levels we recommend to let every heading be followed by at least a short passage of text. Use the LaTeX automatism for all your cross-references and citations citations as has already been described in Sect. 1.1.

Please note that the first line of text that follows a heading is not indented, whereas the first lines of all subsequent paragraphs are.

Chapter 2
Latex Part

We are going to explain the LaTeXcode here. I have used a book format of a4 paper where both sides of the paper can have texts. The other packages that I have used here is type1cm, makeidx (it will allow the index generation), graphicx (for graphical interpretation), newtxmath and many others.

This LaTeXproject has four main components. These are,

- Author - Chapter, Appendix and References are written here
- Images - All the images that I have used are here
- Style - svmono.cls is kept here
- Book.tex - This is the main file where I am calling the other components

In the very beginning I have made a cover page for the project. Then I have written the contents. We have two chapters and two appendix and one bibliography.

As LaTeXis a markup language, every command we use here starts with a backslash. For example, if we want to write a new chapter, then the command will start with a backslash and then we need to write Chapter followed by the curly braces. Inside the curly braces the heading of the chapter will be written. This way we can write a new chapter.

Similar way we can write down most of the builtin commands. For mathematical notations we need to write an equation block which will start with a backslash begin then curly braces and then backslash end followed by curly braces. between the curly braces equation should be written. then every equation we write it has be within this block.

Appendix A
Appendix 1 Complete Sage-Math source code

The solution of 1(a) is given below:

```
#Program 1(a)
def Newton_Interpolation(x):

    p(x)=points[0][1]
    dt(x)=x-points[0][0]

    for k in [1,2, ..,n]:
        c=((points[k][1]-p(points[k][0]))/dt(points[k][0]))
        p(x)=dt(x)*c+p(x)
        dt(x)=(x-points[k][0])*dt(x)

    return p

    show(Newton_Interpolation(x).full_simplify())
```

The solution of 1(b+c) is given below:

```
#1(b+c)
F(x)=exp(x+sin(x))
xx=[1,2,3,4]
yy=[F(xx[0]),F(xx[1]),F(xx[2]),F(xx[3])]
points=[(xx[0],yy[0]),(xx[1],yy[1]),(xx[2],yy[2]),(xx[3],yy[3])
    ↪ ]
n=len(points)-1

def Newton_Interpolation(x):

    p(x)=points[0][1]
    dt(x)=x-points[0][0]

    for k in [1,2, ..,n]:
        c=((points[k][1]-p(points[k][0]))/dt(points[k][0]))
        p(x)=dt(x)*c+p(x)
        dt(x)=(x-points[k][0])*dt(x)

    return p

Newton_Interpolation(x).full_simplify()

p1=plot(Newton_Interpolation(x),x,points[0][0]-1,points[n
    ↪ ][0]+1,legend_label="P1(x)", frame=True, axes_labels=['
    ↪ $x$ axis','$y$ axis'],axes=True)
p1+=plot(F(x),x,points[0][0]-1,points[n][0]+1,legend_label="f(x
    ↪ )",color='red',linestyle="-.")
p1+=list_plot(points,color='black',size=50)
show(p1)
```

The solution of 1(d+e) is given below:

```
#1(d+e)
F(x)=exp(x+sin(x))
xx=[1,2,3,4,5,6]
yy=[F(xx[0]),F(xx[1]),F(xx[2]),F(xx[3]),F(xx[4]),F(xx[5])]
points=[(xx[0],yy[0]),(xx[1],yy[1]),(xx[2],yy[2]),(xx[3],yy[3])
    ↪ ,(xx[4],yy[4]),(xx[5],yy[5])]
n=len(points)-1


def Newton_Interpolation(x):

    p(x)=points[0][1]
    dt(x)=x-points[0][0]

    for k in [1,2, ..,n]:
        c=((points[k][1]-p(points[k][0]))/dt(points[k][0]))
        p(x)=dt(x)*c+p(x)
        dt(x)=(x-points[k][0])*dt(x)

    return p

Newton_Interpolation(x).full_simplify()
p2=plot(Newton_Interpolation(x),x,points[0][0],points[n][0],
    ↪ legend_label="P2(x)")
p2+=plot(F(x),x,points[0][0], points[n][0],color='green',
    ↪ legend_label="f(x)", linestyle="--", frame=True,
    ↪ axes_labels=['$x$ axis','$y$ axis'],axes=True)
p2+=list_plot(points,color='black',size=40)
show(p2)
```

The solution of 1(g) is given below:

```
#1(g)
of=plot(F(x),(x,0,6))
ni1=plot(p1)
ni2=plot(p2)
com=plot(of+ni1+ni2)
graphics_array([[ni1],[ni2],[com]])
```

The solution of problem number 2 is given below:

```
#Problem number 2
x = [1,2,3,4]
y = [2.76, 7.65, 21.16, 54.54]
def DividedDiff(x, y, xi):
    #length/number of datapoints
    n = len(x)
    #divided difference initialization
    fdd = [[None for x in range(n)] for x in range(n)]
    #f(X) values at different degrees
    yint = [None for x in range(n)]
    #error value
    ea = [None for x in range(n)]

    #finding divided difference
    for i in range(n):
        fdd[i][0] = y[i]
    for j in range(1,n):
        for i in range(n-j):
            fdd[i][j] = (fdd[i+1][j-1]-fdd[i][j-1])/(x[i+j]-x[i])

    #interpolating xi
    xterm = 1
    yint[0] = fdd[0][0]
    for order in range(1, n):
        xterm = xterm * (xi - x[order-1])
        yint2 = yint[order-1] + fdd[0][order]*xterm
        ea[order-1] = yint2 - yint[order-1]
        yint[order] = yint2


    return yint[order]

a = DividedDiff(x, y, yint)
```

Appendix B
Appendix 2 Complete LaTeXsource code

——————book.tex——————

```
\documentclas[graybox,envcountchap,sectrefs,a4paper,oneside,
    ↪ openany]{svmono}
\usepackage{lipsum}
\usepackage{xpatch}
\makeatletter
\makeatother
\usepackage{type1cm}
\usepackage{makeidx} % allows index generation
\usepackage{graphicx} % standard LaTeX graphics tool
\usepackage{multicol} % used for the two-column index
\usepackage[bottom]{footmisc}% places footnotes at page bottom
\usepackage{newtxtext} %
\usepackage{newtxmath} % selects Times Roman as basic font
\makeindex
\usepackage{listings} % used for the subject index
\begin{document}



\begin{titlepage}
   \begin{center}
       \vspace*{1cm}

       \Huge
       \textbf{Examination Computer Algebra and \LaTeX}

       \vspace{0.5cm}
       \LARGE
       Summer Semester 2020
```

```
       \vspace{1.5cm}

       \textbf{Debojit Das}

       \vfill

       Matriculation Number: 51866\\
       Seminar Group : MA19w1-B\\


       \vspace{0.8cm}


       \Large
       University of Applied Sciences Mittweida\\
       20 August, 2020



   \end{center}
\end{titlepage}


\frontmatter%%%%%%%%%%%%%%%

\tableofcontents

\include{author/acronym}

\mainmatter%%%%%%%%%%%%%%%%%
\include{author/part}
\include{author/chapter}
\include{author/appendix}
\include{author/references}

\backmatter%%%%%%%%%%%%%%%%%
\include{author/glossary}

\printindex

\end{document}
```

——————chapter.tex——————

```
\chapter{Polynomial Interpolation}
\label{intro} % Always a unique label
% need to use \chaptermark{}
% to alter or adjust the chapter heading in the running head

\abstract{The first thing to realize when fitting a polynomial
    ↪ (exactly) to discreet \textit{data} is that the
    ↪ polynomial must have as many parameters (i.e.
    ↪ coefficients) we are free to vary as there are
    ↪ conditions (i.e. points) in our \textit{dataset.} \
    ↪ newline\indent
Thus, for example, to run a polynomial through all points of
    ↪ the table, it needs to have atleast 4 coefficients (i.e.
    ↪  degree 3) such that,
\begin{equation}
   c_{0} + c_{1}x_{i} + c_{2}x_{i}^{2} + c_{3}x_{i}^{3} + c_
       ↪ {4}x_{i}^{4} = y_{i}
\end{equation}
\newline\indent
where i = 1, 2, ... 5. The resulting 4 equations for 4 unknowns
    ↪  are linear, having a unique solution (unless there are
    ↪ two or more identical values in the x row). Later on we
    ↪ will solve such system of linear equations but the
    ↪ general technique is quite lengthy and tedious}

\section{Newton Interpolation}
\label{sec:2}

Polynomial interpolation is a method of estimating values
    ↪ between known data points. When graphical data contains
    ↪ a gap, but data is available on either side of the gap
    ↪ or at a few specific points within the gap, an estimate
    ↪ of values within the gap can be made by interpolation.

The simplest method of interpolation is to draw straight lines
    ↪ between the known data points and consider the function
    ↪ as the combination of those straight lines. This method,
    ↪  called linear interpolation, usually introduces
    ↪ considerable error. A more precise approach uses a
    ↪ polynomial function to connect the points. A polynomial
    ↪ is a mathematical expression comprising a sum of terms,
    ↪ each term including a variable or variables raised to a
    ↪ power and multiplied by a coefficient. The simplest
```

```
    ↪ polynomials have one variable. Polynomials can exist in
    ↪ factored form or written out in full.


\subsection{General Explanation / Procedure}
\label{subsec:2}
The Lagrange interpolation relies on the n+1 interpolation
    ↪ points $\{x_i,\,y_i=f(x_i),\;i=0,\cdots,n\}$ all of
    ↪ which need to be available to calculate each of the
    ↪ basis polynomials $l_{i}(x)$ If additional points are to
    ↪  be used when they become available, all basis
    ↪ polynomials need to be recalculated. \newline\indent
In comparison, in the Newton interpolation, when more data
    ↪ points are to be used, additional basis polynomials and
    ↪ the corresponding coefficients can be calculated, while
    ↪ all existing basis polynomials and their coefficients
    ↪ remain unchanged. Due to the additional terms, the
    ↪ degree of interpolation polynomial is higher and the
    ↪ approximation error may be reduced (e.g., when
    ↪ interpolating higher order polynomials). \newline\indent
Specifically, the basis polynomials of the Newton interpolation
    ↪  are calculated as below:
\begin{equation}
n_0(x)=1,\;\;\;\;\;n_i(x)=\prod_{j=0}^{i-1}(x-x_j),\;\;\;\;\;\;\;(i
    ↪ =1,\cdots,n)
\end{equation}
And the newton interpolated polynomial is constructed:
\begin{eqnarray}
N_n(x)&=&\sum_{i=0}^n c_i n_i(x)=c_0+\sum_{i=1}^n c_i\;
  \left(\prod_{j=0}^{i-1}(x-x_j)\right)
  \nonumber \\
  &=&c_0+c_1(x-x_0)+c_2(x-x_0)(x-x_1)+\cdots+c_n\prod_{j=0}^{n
      ↪ -1}(x-x_j)
\end{eqnarray}
When the next data points are available, not used in any of the
    ↪  basis polynomials but it is used for calculating the
    ↪ last coefficient $c_{n}$, as shown below. For this nth
    ↪ degree polynomial to pass all n+1 points, it needs to
    ↪ satisfy the following n+1 equations.
\begin{equation}
\left\{\begin{array}{l}
  y_0=c_0\\
  y_1=c_0+c_1(x_1-x_0)\\
  y_2=c_0+c_1(x_2-x_0)+c_2(x_2-x_0)(x_2-x_1)\\
```

```
  y_3=c_0+c_1(x_3-x_0)+c_2(x_3-x_0)(x_3-x_1)+c_3(x_3-x_0)(x_3-
      ↪ x_1)(x_3-x_2)\\
  \cdots \cdots \cdots \cdots \\
  y_n=c_0+c_1(x_n-x_0)+c_2(x_n-x_0)(x_n-x_1)+\cdots+c_n\prod_{i
      ↪ =0}^{n-1}(x_n-x_i)
  \end{array}\right.
\end{equation}
\eject
Now the Newton polynomial interpolation can be written as:
\begin{equation}
N_n(x)=\sum_{i=0}^nc_i n_i(x)=\sum_{i=0}^n f[x_0,\cdots,x_i]
    ↪ n_i(x)
  =f[x_0]+\sum_{i=1}^n\;f[x_0,\cdots,x_i]\left(\prod_{j=0}^{i
      ↪ -1}(x-x_j)\right)
\end{equation}



\subsection{Explanation with example}
\label{subsec:3}
Say, we have the function, $f(x)$ = $e^{(x+sinx)}$ by a
    ↪ polynomial degree of n = 3, based on the following n + 1
    ↪  = 4 points:
\begin{equation}
\begin{array}{c||c|c|c|c}\hline
   i & 0 & 1 & 2 & 3 \\\hline \hline
   x_i & 1 & 2 & 3 & 4 \\\hline
   f(x_i) & 2.76 & 7.65 & 21.16 & 58.54 \\\hline
  \end{array}
  \nonumber\\
\end{equation}
Based on $f[x_{i}]$= $f(x_{i})$, (i = 0, ... , n), we can find
    ↪ all other divided differences recursively in tabular
    ↪ form as shown below, In general it can be found based on
    ↪  its left neighbour and top left neighbour:
 \begin{equation}
f[x_i,\cdots,x_j]=\frac{f[x_{i+1},\cdots,x_j]-f[x_i,\cdots,x_{j
    ↪ -1}]}{x_j-x_i}
  \nonumber\\
\end{equation}
 \begin{equation}
\begin{array}{c||l|l|l|l}\hline
   x_i & 0th & 1st & 2nd & 3rd \\\hline\hline
   x_0=1& f[x_0]=2.76 & & & \\\hline
   x_1=2 & f[x_1]=7.65 & f[x_0,x_1]=4.89 & & \\\hline
```

```
    x_2=3 & f[x_2]=21.16 & f[x_1,x_2]=13.51 & f[x_0,x_2]=4.31 &
        ↪  \\\hline
    x_3=4 & f[x_3]=54.54 & f[x_2,x_3]=37.38 & f[x_1,x_3]=12 & f
        ↪ [x_0,x_3]=1.28 \\\hline
  \end{array}
  \nonumber\\
\end{equation}
Alternatively, They can be represented in the expanded form:
\begin{equation}
c_i=f[x_0,\cdots,x_i]=\sum_{j=0}^n\frac{f(x_j)}{\prod_{i=0,\,i\
    ↪ ne j}^n(x_j-x_i)}
\end{equation}
Now the Newton interpolating polynomial can be obtained as:
\begin{eqnarray}
N_3(x)&=&\sum_{i=0}^3 c_i n_i(x)
  \nonumber \\
  &=&2.76+4.89(x+1)+0.643(x+1)(x-0)-0.663(x+1)(x-0)(x-1)
  \nonumber \\
  &=&8.73+19.88\,x-10.92\,x^2+2.53\,x^3
  \nonumber
\end{eqnarray}


\subsection{Realization of Newton Interpolation with Sage}
\label{subsec:4}
In the Sage part, we have done the calculation of our
    ↪ Polynomial Interpolation using Newton's procedure, as
    ↪ well as we have calculated the graphical representation.
    ↪  \newline\indent
First, We have written a function which will return $p(x)$. For
    ↪  this reason, we need to calculate $p(x)$ and $dt(x)$.
    ↪ Then, we have used a loop so that we can calculate the
    ↪ value of C, $p(x)$ and $dt(x)$. In the last part of the
    ↪ function we have used the built in command full simplify
    ↪  to simplify the functional value. \newline\indent

Then we have defined our function, xx values and yy values and
    ↪ the points. After that, we recalled our function to
    ↪ simply calculate the values. Later that we plotted the
    ↪ graph of the main function and the interpolated function
    ↪  together using different legend labels. Axes labels,
    ↪ frame, different color and different linestyle was also
    ↪ added.

\begin{figure}[h]
```

```
    \centering
    \includegraphics[scale=0.7]{images/sage0.png}
    \caption{Original and interpolation polynomial functions
        ↪ with different colors in one image.}
    \label{fig:mesh1}
\end{figure}
\eject

Then we have calculated for the extended two more points.

\begin{figure}[h]
    \centering
    \includegraphics[scale=0.7]{images/sage1.png}
    \caption{Original and second interpolation polynomial
        ↪ functions with different colors in one image.}
    \label{fig:mesh2}
\end{figure}

Now we will generate a graphics array consisting of 3 rows and
    ↪ 1 column where the first array (position (3; 1))
    ↪ contains $P_{1}(x)$ and the original function $f(x)$.
    ↪ The second array (position (2; 1)) contains $P_{2}(x)$
    ↪ and the original function $f(x)$. Finally the third one
    ↪ displays the three functions $P_{1}(x)$, $P_{2}(x)$ and
    ↪ $f(x)$. Here, we are using different colors, linestyles,
    ↪  label boxes and axis labels.
\eject

\begin{figure}[h]
    \centering
    \includegraphics[scale=0.7]{images/sage2.png}
    \caption{Original, first interpolation function and second
        ↪ interpolation polynomial functions with different
        ↪ colors in one image.}
    \label{fig:mesh3}
\end{figure}




\section{Divided Differences}
\label{sec:3}

From here, we will discuss about the Polynomial Interpolation
    ↪ using Divided Difference. Previously, we established
```

```
    ↪ that for each n = 0, 1, ... , there exists a unique
    ↪ polynomial $P_{n}$ such that, the degree of $P_{n}$ is
    ↪ at most n. It was shown that $P_{n}$ can be expressed in
    ↪  the Newton form:
\begin{equation}
  P_n(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + \cdots
      ↪  + b_n(x - x_0) \cdots (x - x_{n-1})
\end{equation}


\subsection{General Explanation / Procedure} %
The general form of Newton's divided-difference interpolating
    ↪ polynomial is
\[p_n(x) = b_0 + b_1(x-x_0) + b_2(x-x_0)(x-x_1)+ \cdots + b_n(x
    ↪ -x_0)(x-x_1)\cdots (x-x_{n-1}),\]
where the $b_i$ are recursively computed.

To define the $b_i$, $i=0,\ldots,n$, we first define the
    ↪ quantities $f[x_i,x_{i-1},\ldots,x_{i-j}]$, where $0\leq
    ↪  j\leq i$.

\begin{description}
\item[\underline{Base Case 1.}] $f[x_i]=y_i$ is the $y$-value
    ↪ associated with $x_i$.

\item[\underline{Base Case 2.}] $f[x_i,x_{i-1}]=\frac{y_i-y_{i
    ↪ -1}}{x_i-x_{i-1}}$ is the average rate of change from
    ↪ $x_{i-1}$ to $x_i$.
In other words, we may think of $f[x_i,x_{i-1}]$ as a first
    ↪ derivative approximation.
\item[\underline{Recursive Case.}] Suppose $f[x_i,x_{i-1},\
    ↪ ldots,x_{i-j+1}]$ and $f[x_{i-1},x_{i-2},\ldots,x_{i-j}]
    ↪ $ have both been defined, while the first represents the
    ↪  approximation of a $(j-1)$~th derivative over the
    ↪ interval $[x_i,x_{i-j+1}]$, while the second represents
    ↪ the approximation of a $(j-1)$~th derivative over the
    ↪ interval $[x_{i-1},x_{i-j}]$. Then $f[x_i,x_{i-1},\ldots
    ↪ ,x_{i-j}]$ is defined as:
\begin{equation}
   f[x_i,x_{i-1},\ldots,x_{i-j}] = \frac{f[x_i,x_{i-1},\ldots,
       ↪ x_{i-j+1}] - f[x_{i-1},x_{i-2},\ldots,x_{i-j}]}{x_i-
       ↪ x_{i-j}}
\end{equation}
```

```
and represents the approximation of a $j$~th derivative over
    ↪ the interval $[x_i,x_{i-j}]$.
\end{description}

\subsection{Explanation with example} %
Our main function here is $y = e^{x+sin(x)}$ \newline
In order to calculate the interpolation using Divided
    ↪ Difference method first we have to take the inputs os x
    ↪ values and functional values of x. then we need to apply
    ↪  the formula
\begin{equation}
N_n(x)=\sum_{i=0}^nc_i n_i(x)=\sum_{i=0}^n f[x_0,\cdots,x_i]
    ↪ n_i(x)
  =f[x_0]+\sum_{i=1}^n\;f[x_0,\cdots,x_i]\left(\prod_{j=0}^{i
      ↪ -1}(x-x_j)\right)
\end{equation}
Then we need to write a nested loop first one from 1 to n and
    ↪ the index of inner loop should be 1 to iteration
    ↪ variable in my case which is k. Then we set the formula.
    ↪  If we print the output, we shall see the result.

\subsection{Realization of Interpolation based on Divided
    ↪ Difference with Sage} %
In order to avoid simply listing headings of different levels
    ↪ we recommend to let every heading be followed by at
    ↪ least a short passage of text. Use the \LaTeX\
    ↪ automatism for all your cross-references and citations
    ↪ citations as has already been described in Sect.~\ref{
    ↪ sec:2}.

Please note that the first line of text that follows a heading
    ↪ is not indented, whereas the first lines of all
    ↪ subsequent paragraphs are.



\chapter{Latex Part}
\label{intro2} % Always give a unique label
% use \chaptermark{}
% to alter or adjust the chapter heading in the running head

We are going to explain the \LaTeX code here. I have used a
    ↪ book format of a4 paper where both sides of the paper
    ↪ can have texts. The other packages that I have used here
    ↪  is type1cm, makeidx (it will allow the index generation
```

```
    ↪ ), graphicx (for graphical interpretation), newtxmath
    ↪ and many others. \newline\indent

This \LaTeX project has four main components. These are,
\begin{itemize}
    \item Author - Chapter, Appendix and References are written
        ↪  here
    \item Images - All the images that I have used are here
    \item Style - svmono.cls is kept here
    \item Book.tex - This is the main file where I am calling
        ↪ the other components
\end{itemize}

In the very beginning I have made a cover page for the project.
    ↪  Then I have written the contents. We have two chapters
    ↪ and two appendix and one bibliography.\bigskip \newline\
    ↪ indent

As \LaTeX is a markup language, every command we use here
    ↪ starts with a backslash. For example, if we want to
    ↪ write a new chapter, then the command will start with a
    ↪ backslash and then we need to write Chapter followed by
    ↪ the curly braces. Inside the curly braces the heading of
    ↪  the chapter will be written. This way we can write a
    ↪ new chapter. \bigskip \newline\indent

Similar way we can write down most of the builtin commands. For
    ↪  mathematical notations we need to write an equation
    ↪ block which will start with a backslash begin then curly
    ↪  braces and then backslash end followed by curly braces.
    ↪  between the curly braces equation should be written.
    ↪ then every equation we write it has be within this block
    ↪ .
```

———————references.tex———————-

```
% \bibliographystyle{}
% \bibliography{}
\begin{thebibliography}{99.}%

% Contribution
\bibitem{science-contrib} Broy, M.: SageMath --- from auxiliary
    ↪  to key technologies. In: Broy, M., Dener, E. (eds.)
    ↪ Software Pioneers, pp. 10-13. Springer, Heidelberg
    ↪ (2002)
%
\bigskip
% Online Document
\bibitem{science-online} Dod, J.: Latex Hacks For Dummies. In:
    ↪ The Dictionary of Substances and Their Effects. Royal
    ↪ Society of Mathematics (1999) Available via DIALOG. \\
\url{http://www.rsc.org/dose/title of LaTeX document. Cited 15
    ↪ Jan 1999}
%
\bigskip
\bibitem{science-online} Richard, N.: Latex Bible. Greyy
    ↪ Publication (2008) \\
\url{http://www.ptrb.org/main/bibLaTeX. Cited 21 Mar 2009}
\bigskip

% Monograph
\bibitem{science-mono} Geddes, K.O., Czapor, S.R., Labahn, G.:
    ↪ Algorithms for Computer Algebra. Kluwer, Boston (1992)
%
\bigskip
% Journal article
\bibitem{science-journal} Hamburger, C.: Mathematical Methods.
    ↪ Ann. Mat. Pura. Appl. \textbf{169}, 321--354 (1995)
%
\bigskip

\end{thebibliography}
```

References

1. Broy, M.: SageMath — from auxiliary to key technologies. In: Broy, M., Dener, E. (eds.) Software Pioneers, pp. 10-13. Springer, Heidelberg (2002)

2. Dod, J.: Latex Hacks For Dummies. In: The Dictionary of Substances and Their Effects. Royal Society of Mathematics (1999) Available via DIALOG.
`http://www.rsc.org/dose/titleofLaTeXdocument.Cited15Jan1999`

3. Richard, N.: Latex Bible. Greyy Publication (2008)
`http://www.ptrb.org/main/bibLaTeX.Cited21Mar2009`

4. Geddes, K.O., Czapor, S.R., Labahn, G.: Algorithms for Computer Algebra. Kluwer, Boston (1992)

5. Hamburger, C.: Mathematical Methods. Ann. Mat. Pura. Appl. **169**, 321–354 (1995)