MA228 NUMERICAL ANALYSIS - EXERCISE SHEET 2

Joe Tilley 1403899

January 28, 2016

Question 1

This question is not assessed.

Question 2

The most appropriate way to choose $n = 11$ points in the range $[0, 2\pi]$ is with regular intervals. If we let $h = \frac{2\pi}{n-1} = \frac{\pi}{5}$, then we let $x_i = (i-1)h$ for $i = 1, 2, ..., 11$, these will be our 11 points with regular intervals (including 0 and 2π). Note that, unlike in regular theory, I have started at x_1 instead of x_0, to match MATLAB's code indexing starting at 1 instead of 0. We can easily adjust the points for greater n, as it will be done automatically by the code.

Using these defined points, we can Lagrange interpolate $f(x)$ by defining 11 Lagrange interpolating polynomials, since $f(x)$ is approximately equal to $P(x)$, defined by

$$P(x) = \sum_{i=1}^{11} f(x_i) L_i(x)$$

where

$$L_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{11} \frac{(x - x_j)}{(x_i - x_j)}$$

Firstly, we will define the function which finds the Lagrange polynomials with the following code

```
function output = Lagrange(x, points, i, n)
%Finds the ith Lagrange Polynomial of n points
Lx = 1;             %Lx is lagrange polynomial, initially set as multipicative ...
    identity
    for j=1:n
        if j≠i          %If i not equal to j, multiply by (x-x_j)/(x_i-x_j)
            Lx= Lx.*(x-points(j))/(points(i)-points(j));
        end
    end
 output = Lx;           %Final output after all products done
end
```

Note the inputs are:
'x' the x-range (it will be from 0 to 2π for our case).
'points' which will be an array of x_i for $i = 1, 2, ..., n$.
'i' the number of the Lagrange polynomial (the number 'i' in $L_i(x)$).
'n' the number of data points used.

Now we can call upon this function to find the Lagrange polynomials we want and hence find $P(x)$ as done in the code below.

```
1  clear
2
3  len=10000;              %Number of intervals in x-range
4  nlen=2*pi/len;          %Length of each interval in x-range
5  x = 0:nlen:2*pi;        %x-range
6
7  n=11;                   %Number of points
8  h=(2*pi)/(n-1);         %Equal length between points
9  for k=1:n
10     points(k)=(k-1)*h;    %Defines each point
11 end
12 fpoints = sin(points);  %Defines the sin of each of those points
13
14 Px=0;             %Px is approximation of fx, initally set as additive identity
15
16 for i=1:n
17     LagrangeList(i,1:len+1)=Lagrange(x, points, i, n);     %Calls the function
18     %Saves the output as the ith Lagrange polynomial, in an array
19     Px = Px + fpoints(i).*LagrangeList(i,:); %Add f(x_i)*L_i(x) to Px
20 end
```

Hence, we have defined the 11 Lagrange interpolating polynomials on the x-range, the i$^{\text{th}}$ Lagrange interpolating polynomial, $L_i(x)$, which is stored in the i^{th} index of the array 'LagrangeList' (Line 17). We have also defined the polynomial approximating $f(x)$ on the x-range, $P(x)$, labelled 'Px' (Line 19).

Question 3

We can plot all the Lagrange interpolating polynomials relatively easily, and I have plotted these on two separate figures (so that they are not all cluttered on one figure). They are shown on Figures 1 and 2.

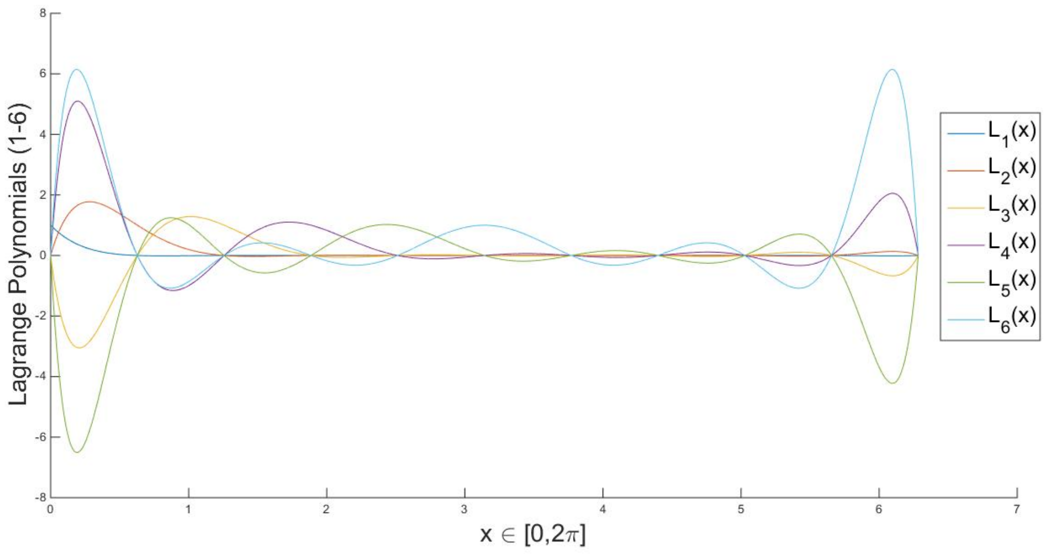

Figure 1: A graph displaying the Lagrange interpolating polynomials $L_i(x)$ for $i = 1, 2, ..., 6$ as defined in the code

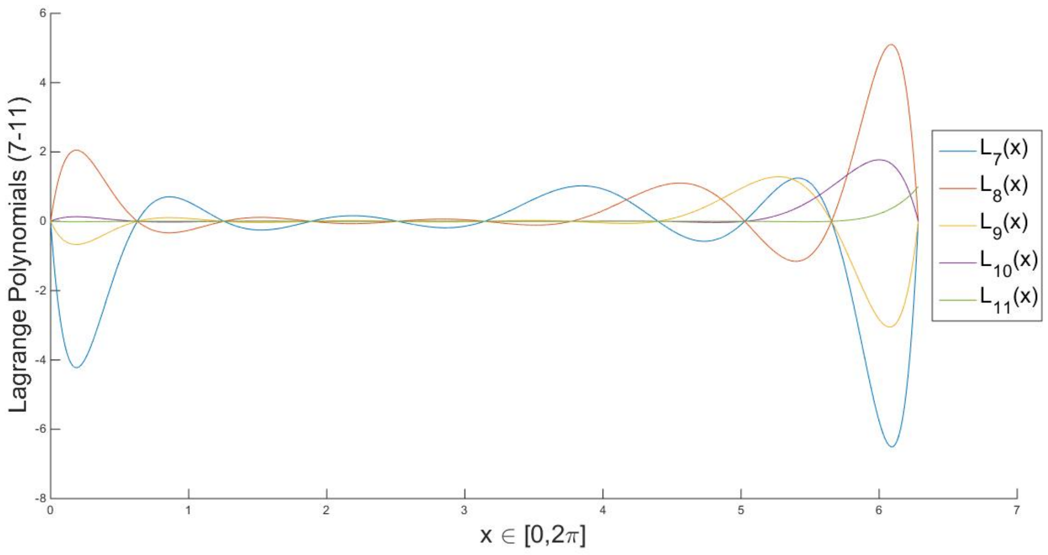

Figure 2: A graph displaying the Lagrange interpolating polynomials $L_i(x)$ for $i = 7, 8, ..., 11$ as defined in the code

Plotting $P(x)$, the polynomial of order 10 approximating $f(x)$ gives an extremely good approximation (as seen by the error in Question 4), indistinguishable by eye (the lines overlap everywhere), so I have plotted $P(x)$ and $f(x)$ on two different graphs on Figure 3 and Figure 4 respectively. Note $f(x)$ was generated by simply taking $\sin(x)$ where 'x' is the x-range from Line 5.

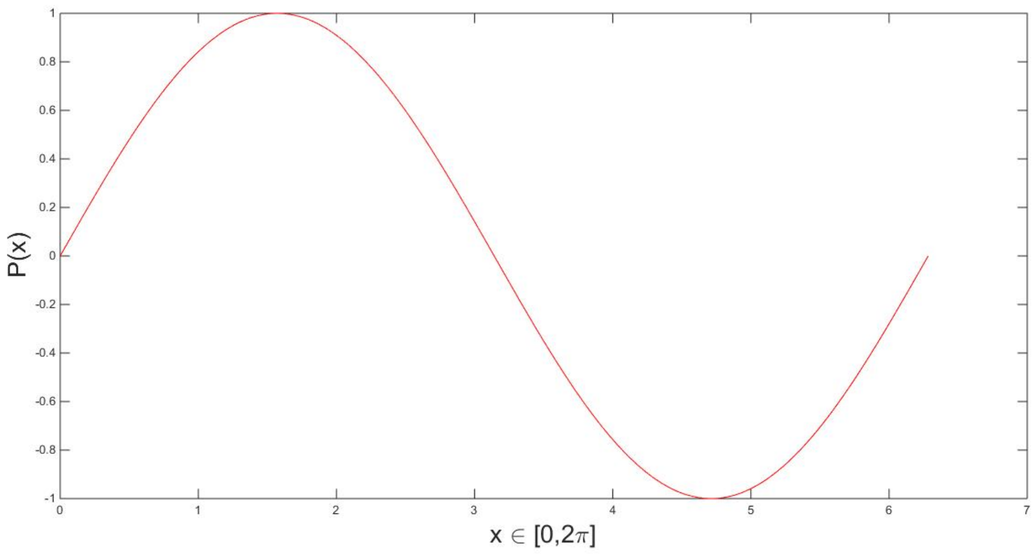

Figure 3: A graph displaying $P(x)$

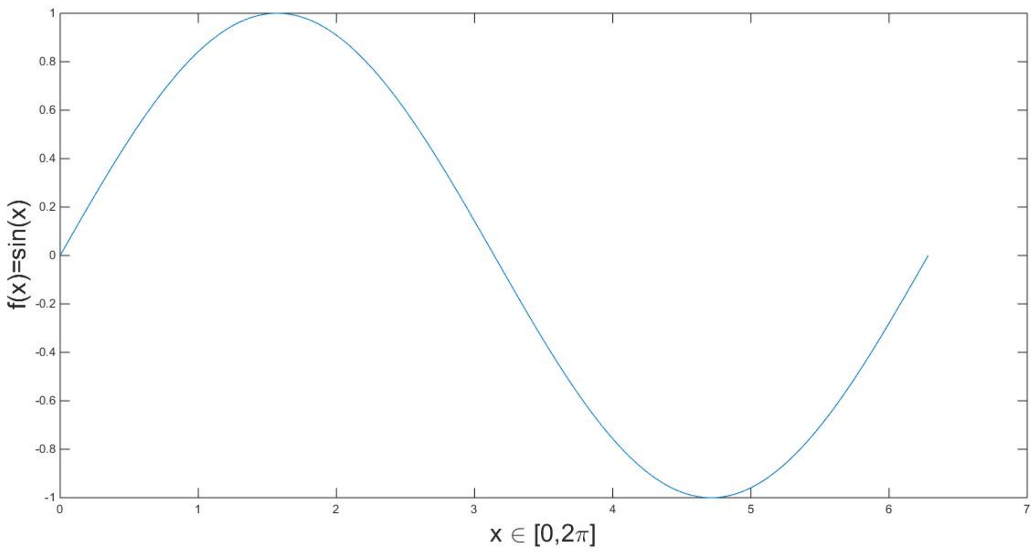

Figure 4: A graph displaying $f(x)$

Question 4

As proven in lectures, for 11 points, the error is given by

$$|f(x) - P(x)| = \left| \frac{f^{(11)}(\xi(x))}{11!} \prod_{i=1}^{11} (x - x_i) \right|$$

for some $\xi(x)$ in $(0, 2\pi)$. We can bound this error by taking the supremum. Noting $f(x) = \sin(x)$ so $f^{(11)}(x) = -\cos(x)$, we have

$$|f(x) - P(x)| = \left| \frac{-\cos(\xi(x))}{11!} \prod_{i=1}^{11} (x - x_i) \right| \leq \left| \frac{\sup_{\xi(x)\in(0,2\pi)}(-\cos(\xi(x)))}{11!} \prod_{i=1}^{11} (x - x_i) \right| = \left| \frac{1}{11!} \prod_{i=1}^{11} (x - x_i) \right|$$

So, the following code (when added the end of the algorithm from Question 2) will find the error function on the x-range

```
error=1;                  %Theoretical error initially multiplicative identity
for i=1:n             %n=11
    producterror=error.*(x-points(i));
end
error=abs(producterror/(factorial(n))); %n=11
```

We can now change to 21 data points easy by setting $n = 21$ in Line 7 of the algorithm, as the points are based entirely on this and are of equal distance apart, so the points array will adjust automatically. Additionally, the code calculating the error may remain exactly the same, since

$$|f(x) - P(x)| = \left| \frac{f^{(21)}(\xi(x))}{21!} \prod_{i=1}^{21} (x - x_i) \right| = \left| \frac{\cos(\xi(x)))}{21!} \prod_{i=1}^{21} (x - x_i) \right| \leq \left| \frac{1}{21!} \prod_{i=1}^{21} (x - x_i) \right|$$

and the change from 11 to 21 will automatically adjust by n in the code. We should run the code for $n = 11$ and $n = 21$ and compare the two errors on a graph by plotting each (against the x-range). Plotting these two errors on separate graphs gives Figure 4 and 5 (I have plotted them on separate figures, else the smaller error is so small that it is indistinguishable to 0 when on a scale with the larger error).

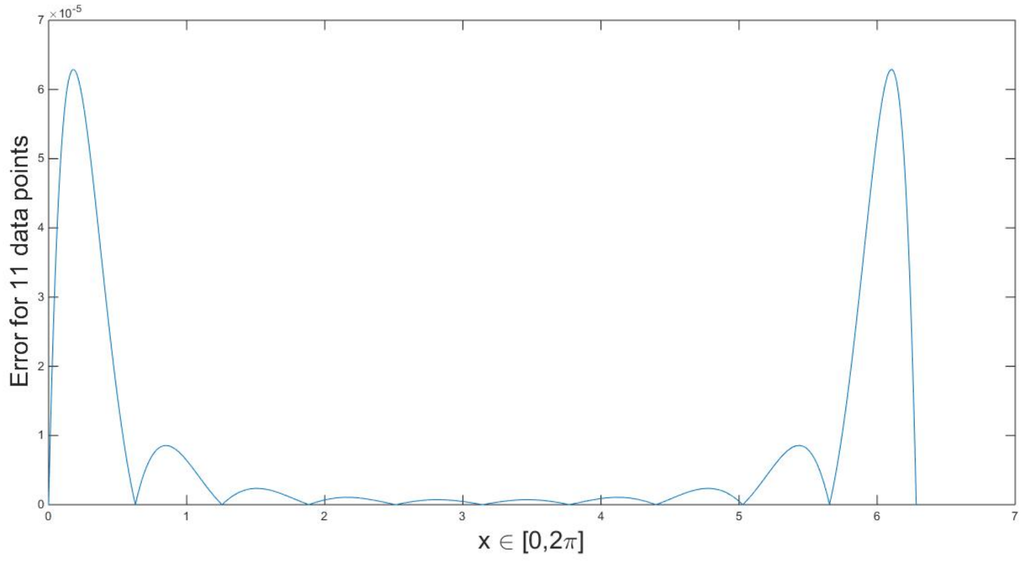

Figure 5: A graph displaying the error for 11 data points ($n = 11$)

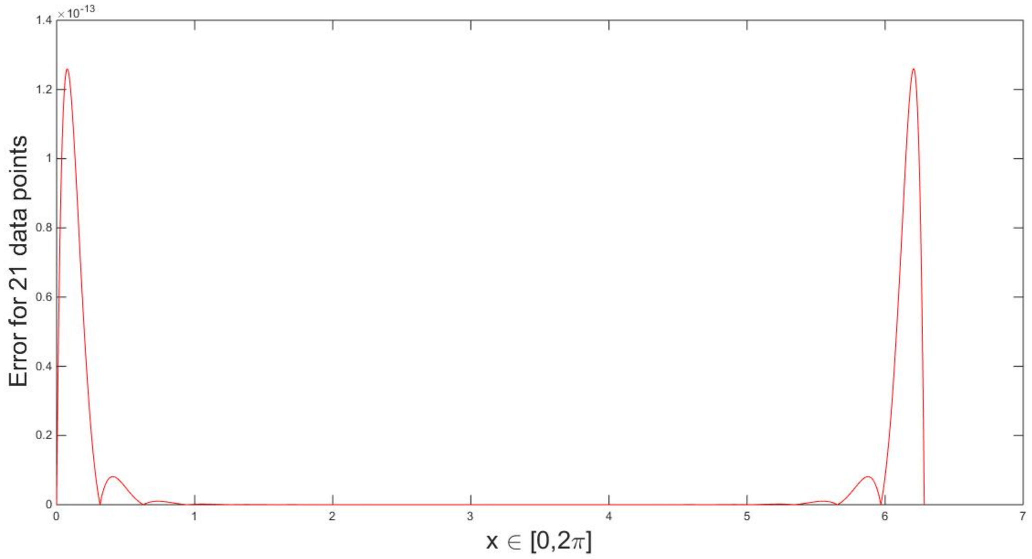

Figure 6: A graph displaying the error for 21 data points ($n = 21$)

As expected, the error is 0 at the data points (due to the nature of how the Lagrange polynomial is generated). Comparing these two estimates, we see that the error estimate for 11 points has a maximum of the order 10^{-5}, whereas for 21 points, this error reduces significantly, having a maximum of the order 10^{-13}. This is partially explained by the $n!$ in the denominator of the error bound, which increases for the more points we use, so the error bound decreases as n increases. So we have shown that using more data points gives a more accurate estimation of the function (in this example).

The errors appear to have a very similar pattern for each n, having greater error in intervals nearer to the boundary and having a small error in the centre (comparative to the rest of the error). Furthermore, the error in both number of points is very large (compared to the other parts of the error) in the range $[x_1, x_2]$ and $[x_{n-1}, x_n]$. For larger n, the error becomes far sharper during these intervals compared to the rest of the function (likewise to a Gibbs Phenomenon of Fourier series as more terms are used). Since the interval sizes get smaller for increasing n, these 'peaks' will become increasingly sharp (since their 'base' will shorten). However, whilst the peaks are sharper for increasing n, their maximum is still smaller (so the overall error is smaller everywhere). This is called a Runge Phenomenon.