FMNF10 Numerical analysis Project 1

Robin Öhrnberg

April 15, 2024

The influence of the basis

Problem 1

The linear system $Ac = y$ that needs to be solved to find the coefficients c_i can be seen below:

$$\underbrace{\begin{bmatrix} \phi_0(t_1) & \phi_1(t_1) & \phi_2(t_1) & \phi_3(t_1) & \phi_4(t_1) & \phi_5(t_1) & \phi_6(t_1) \\ \phi_0(t_2) & \phi_1(t_2) & \phi_2(t_2) & \phi_3(t_2) & \phi_4(t_2) & \phi_5(t_2) & \phi_6(t_2) \\ \phi_0(t_3) & \phi_1(t_3) & \phi_2(t_3) & \phi_3(t_3) & \phi_4(t_3) & \phi_5(t_3) & \phi_6(t_3) \\ \phi_0(t_4) & \phi_1(t_4) & \phi_2(t_4) & \phi_3(t_4) & \phi_4(t_4) & \phi_5(t_4) & \phi_6(t_4) \\ \phi_0(t_5) & \phi_1(t_5) & \phi_2(t_5) & \phi_3(t_5) & \phi_4(t_5) & \phi_5(t_5) & \phi_6(t_5) \\ \phi_0(t_6) & \phi_1(t_6) & \phi_2(t_6) & \phi_3(t_6) & \phi_4(t_6) & \phi_5(t_6) & \phi_6(t_6) \\ \phi_0(t_7) & \phi_1(t_7) & \phi_2(t_7) & \phi_3(t_7) & \phi_4(t_7) & \phi_5(t_7) & \phi_6(t_7) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{bmatrix}}_{c} = \underbrace{\begin{bmatrix} 6{,}371{,}432 \\ 7{,}041{,}829 \\ 7{,}497{,}967 \\ 8{,}081{,}229 \\ 8{,}317{,}937 \\ 8{,}590{,}630 \\ 8{,}882{,}792 \end{bmatrix}}_{y} \tag{1}$$

There will be four versions of the matrix A from equation (1). One for each basis function. I implemented the basis functions as anonymous functions as seen below.

```
phi_a = @(x) (x*ones(1,7)).^[0:6];
phi_b = @(x) ((x-1940)*ones(1,7)).^[0:6];
phi_c = @(x) ((x-1970)*ones(1,7)).^[0:6];
phi_d = @(x) (((x-1970)/30)*ones(1,7)).^[0:6];
```

Listing 1: The basis functions as anonymous functions

The anonymous functions seen in listing **??** are designed to each give one row of the A-matrix for the respective basis functions. I construct one A-matrix for each basis function with a loop and store them in a cell, as seen in listing **??**.

```
%fill out matricies
for i=1:7
    Aa(i,1:7) = phi_a(t(i));
    Ab(i,1:7) = phi_b(t(i));
    Ac(i,1:7) = phi_c(t(i));
    Ad(i,1:7) = phi_d(t(i));
end
%store result in a cell to simplyfy future use:
Acell = {Aa Ab Ac Ad};
```

Listing 2: Constructing the A matrices.

I then make three simple loops and calculate and print the condition numbers for each basis function with MATLABs cond function. I did this for the l_1-, l_2- and l_∞-norms. All norms for all versions of the basis function can be found in table 1.

Basis	Norms	Condition numbers
a)	l_1	$6.6723 \cdot 10^{32}$
	l_2	$2.2856 \cdot 10^{28}$
	l_∞	$3.3262 \cdot 10^{28}$
b)	l_1	$8.5213 \cdot 10^{10}$
	l_2	$6.4789 \cdot 10^{10}$
	l_∞	$1.3159 \cdot 10^{11}$
c)	l_1	$1.6097 \cdot 10^{9}$
	l_2	$1.0350 \cdot 10^{9}$
	l_∞	$7.5413 \cdot 10^{8}$
d)	l_1	455.0000
	l_2	189.8141
	l_∞	630.0000

Table 1

As can be seen in table 1 basis function *d)* consistently has the lowest conditional numbers by far. The result does clearly depend on the choice of matrix norm. As the the condition numbers them self they are quite high (some are enormous). I am wondering if I have done some mistake here but can't quite find anything. I did some research and read that it is not unusual to have very high condition numbers. In any case the that the size of the condition numbers is not surprising if one takes a moment to consider how the matrix norm is calculated, especially in conjunction with the basis functions. Basis function *a)* will set each element in the first column to 1 resulting in some large numbers in later columns. Functions *b)* and *c)* will set the first and fourth row to zero respectively except for column 1 which still has 1's for all elements. This removes large elements form the matrices. Function *d)* is similar to *c)* but has a division by a factor 30 in the basis of the exponential. I think it makes sens that the basis function *d)* gives the "lowest" conditional number.

The influence of the basis

Problem 2

In order to represent $p_6(t)$ in the best conditioned base, I simply rewrite the polynomial given in the project description and replace t with basis $\phi(t)$ which gave the best conditional number, which was *d)*.
With:

$$\phi(t) = \frac{t - 1970}{30}$$

This gives us:
$p_6(t) = c_0 + \phi(t)\,(c_1 + \phi(t)\,(c_2 + \phi(t)\,(c_3 + \phi(t)\,(c_4 + \phi(t)\,(c_5 + \phi(t)c_6)))))$ I had at this point not calculated the c coefficients. To do this I used the famous **mldivide** function in MATLAB.

```
c = Ad \ y;
```

Listing 3: Calculating the coefficient.

I then redefined the basis function to fit better with our goal here and defined $p_6(t)$:

```
%redefining phi_d;
phi_d2 = @(x) (((x-1970)./30));
% %Defining p_6(t) on nested form
p_6 = @(t) c(1) + phi_d2(t).*(c(2) + phi_d2(t).*(c(3) + phi_d2(t).*(c(4) + phi_d2(t
    ).*(c(5) + phi_d2(t).*(c(6) + phi_d2(t).*c(7))))));
```

Listing 4: The basis and the polynomial.

With all the components in place i evaluated the polynomial on one-year intervals and plotted the results, as seen in figure 1.

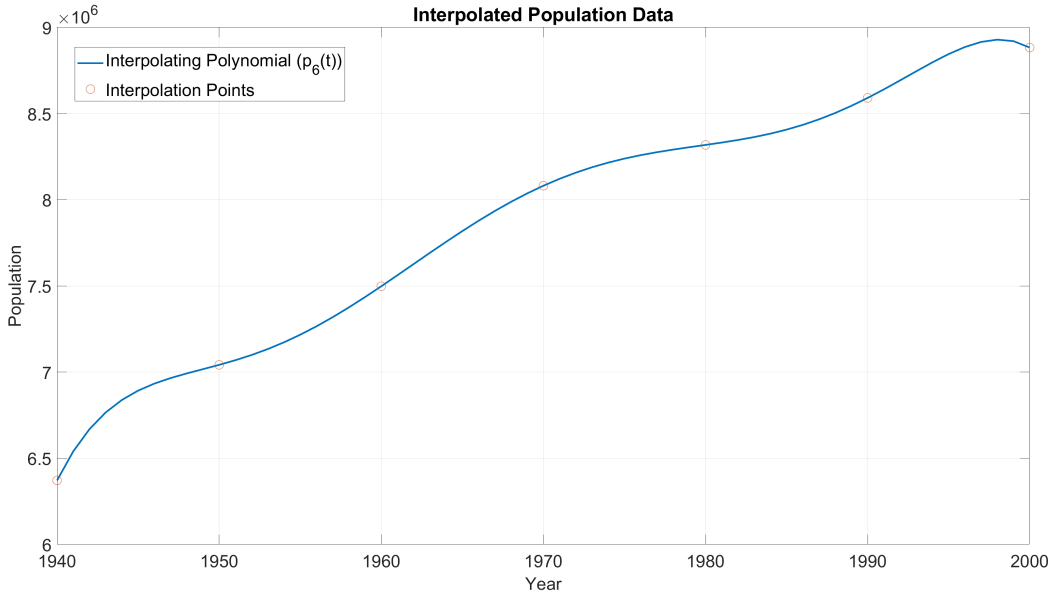

Figure 1: The $p_6(t)$ evaluated at one-year intervals from 1940 to 2000 plotted together with the interpolating plots.

$p_6(t)$ fits then interpolating points very well. I have no intuitive feeling for how good of a fit one could expect from a basis with the condition numbers seen for *d)* in table 1. Therefor I'm not really sure what to comment.

Problem 3

To do this i simple extended the vector containing years and the vector containing the populations sizes and reran my functions and the plotting. The results can be seen in figure 2.

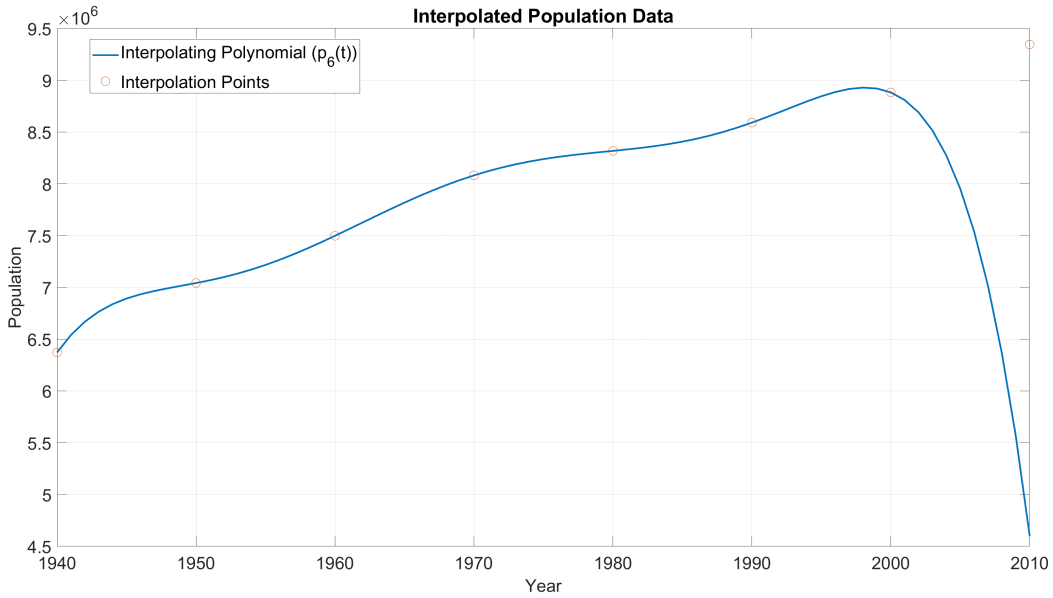

Figure 2: The $p_6(t)$ evaluated at one-year intervals from 1940 to 2010 plotted together with the interpolating plots.

In *Problem 2* I was a little suspicious on how good the fit was. To get a better fit here, we would need to resolve the problem with the new interpolation point added from the start. Thus getting new coefficients in c, and from there we would need to redefine everything.

Newton interpolation and cubic splines

Problem 4

I'm not really sure what to answer when it comes to the Newton form of $p_6(t)$. I used the recursive formulation that can be seen below:

$$p_6(t) = p_5(t) + c_6\phi_6(t) \tag{2}$$

Equation (2) follows the more general formulation in equation (3).

$$p_n(t) = p_{n-1}(t) + c_n\phi_n(t) \tag{3}$$

Here the basis functions $\phi_i(t)$ are the ones given in the project description for Newton interpolation.
In order to find the coefficients with divided differences I wrote a MATLAB function seen in listing 5.

```
function Cs = NewtonCoeff(time,pop)
%Base case
    if length(pop) ==1
        Cs = pop(1);
        return;
    end
%b is the term on the left in every interation of the
b = NewtonCoeff(time(2:end),pop(2:end));
%c is the term on the right in every iteration
c = NewtonCoeff(time(1:end-1),pop(1:end-1));
% temporery value for c
c_temp=(b(end)-c(end))/(time(end)-time(1));
%adding on and returning calls as a vector
Cs = [c;c_temp];
return;
end
```

Listing 5: Recursive function to find coefficients with divided differences.

The coefficients this function gave me are listed in table 2.

Coefficients	
c_0	6371432
c_1	67039.7
c_2	-1071.295
c_3	56.8971
c_4	-3.3960
c_5	0.1392
c_6	-0.004

Table 2

I then set my mind to write a function to preform the Newton Interpolation, this function can be found in listing 6.

```
function P = newton_interpolation(x,c)
    if size(c,1)==1
        P=c*ones(1,size(x,2));
    return;
    end
    Pnminusone = newton_interpolation(x(1:end-1,:),c(1:end-1));
    c_phi = c(end).*prod(x(1:end-1,:));

    Pn = Pnminusone +c_phi;
    P=Pn;
    return;
end
```

Listing 6: Recursive function to prefrom newton interpoation.

The function seen in listing 6 takes a matrix x. This matrix contains $(t - t_k)$ for all times t in the current one-year interval space. I first used the matrix on the one-year interval space between 1940 and 2000 the resulting interpolation can be seen in in figure 3.

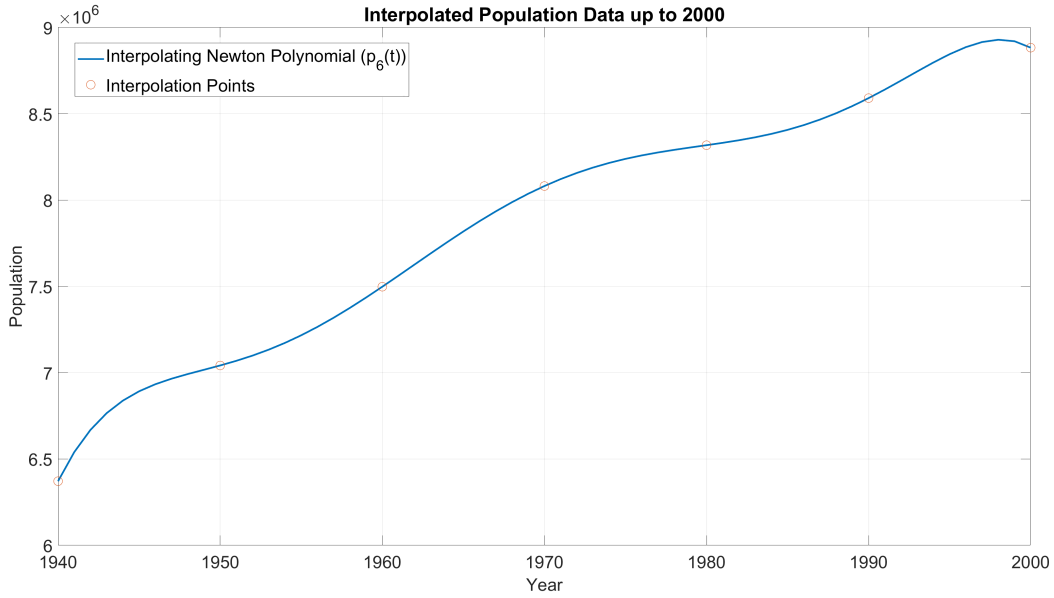

Figure 3: The Newton form $p_6(t)$ evaluated at one-year intervals from 1940 to 2000 plotted together with the interpolating plots.

Figure 3 looks identical to figure 1. I'm not sure this is how the result is supposed to look, but it seem to work.
I then tried to interpolate the one-year interval in the span 1940 to 2010 with the Newton $p_6(t)$ polynomial, the results are shown in figure 4

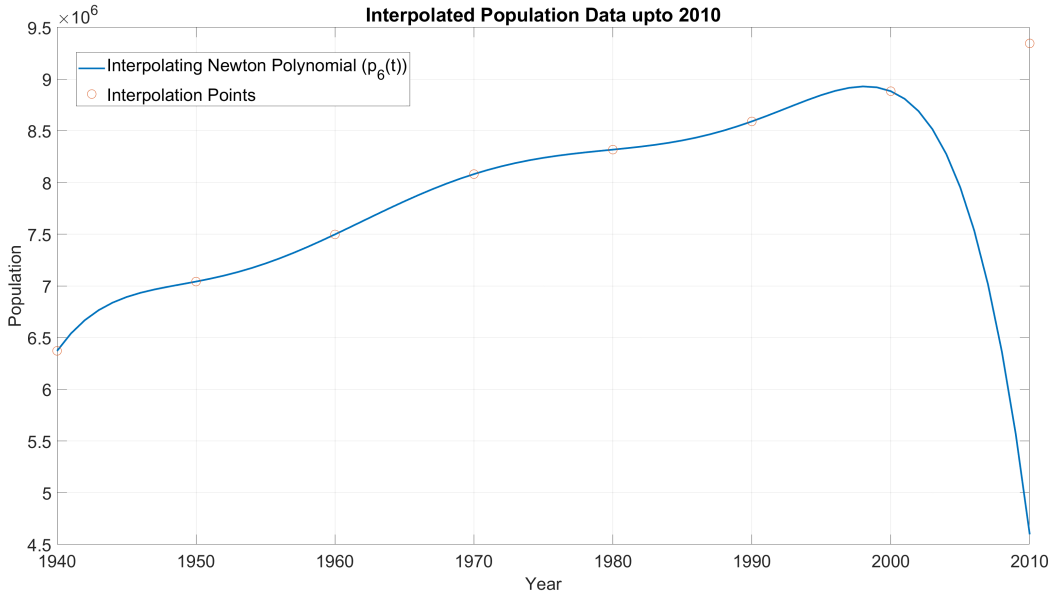

Figure 4: The Newton form $p_6(t)$ evaluated at one-year intervals from 1940 to 2010 plotted together with the interpolating plots.

Yet again the results are eerily similar to previous results, seen in figure 2.
To try to remedy the situation I formulate the Newton form of the seventh degree polynomial $p_7(t)$, using the recursive definition from equation (3):

$$p_7(t) = p_6(t) + c_7\phi_7(t) \tag{4}$$

We can use this to update the $p_6(t)$ polynomial without redoing everything from scratch, given some conditions. If $P_n(x_n) = f_n$ (which is true for my implementation, I triple checked) then:

$$f_n = P_{n-1}(x) + c_n\phi_n(x_n) \leftrightarrow c_n = \frac{f_n - P_{n-1}(x_n)}{\phi_n(x_n)} \tag{5}$$

Armed with the insights from equation (5) I formulated a function to do this. This function is seen in listing 7

```
%adding the 7th degree
 function [p7,c7] = P7(x,p6)
    f7=9345135;
    phi7 = prod(x(1:end-1,end));
    c7=(f7-p6(end))/phi7;
    p7=p6+c7.*prod(x(1:end-1,:));

 end
```

Listing 7: Function to add an eight interpolation point.

I ran this function, this gave me $c_7 = 9.4208 \cdot^{-5}$ and the resulting interpolation can be seen below in figure 5.

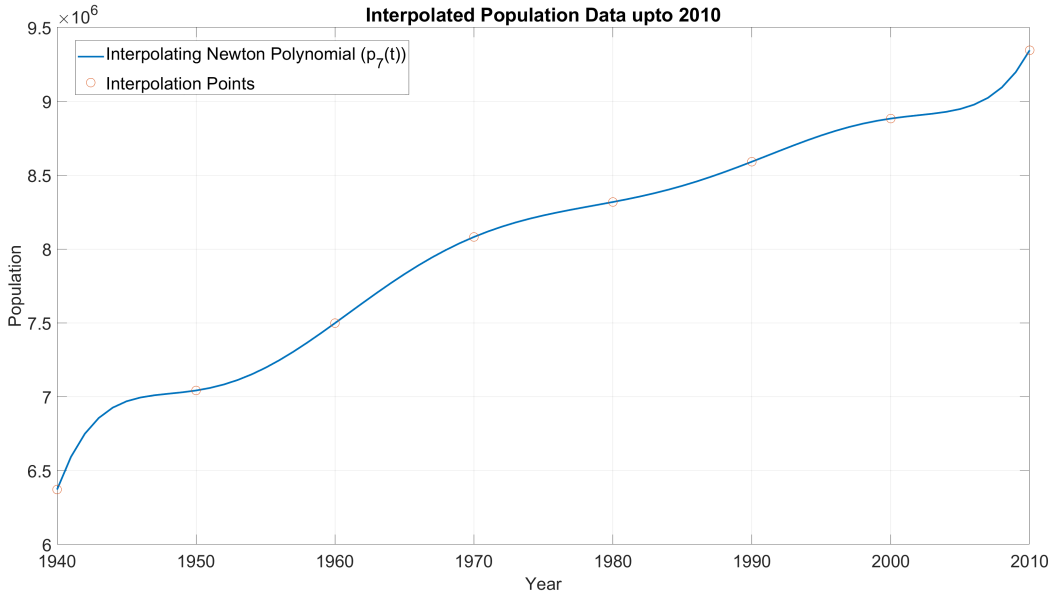

Figure 5: The Newton from $p_6(t)$ evaluated at one-year intervals from 1940 to 2010 plotted together with the interpolating plots.

The results are clearly better than running the interpolation with only seven interpolation points. I'm no expert in demographics but I'm unsure about that sharp rise at the end of the curve.

Problem 5

To solve this problem I used the spline function as seen in listing 8.

```
1 pline = spline((1940:10:2000),y,(1940:2000));
```

Listing 8: Using spline to interpolate.

Figure 6 shows the interpolation from the function spline.

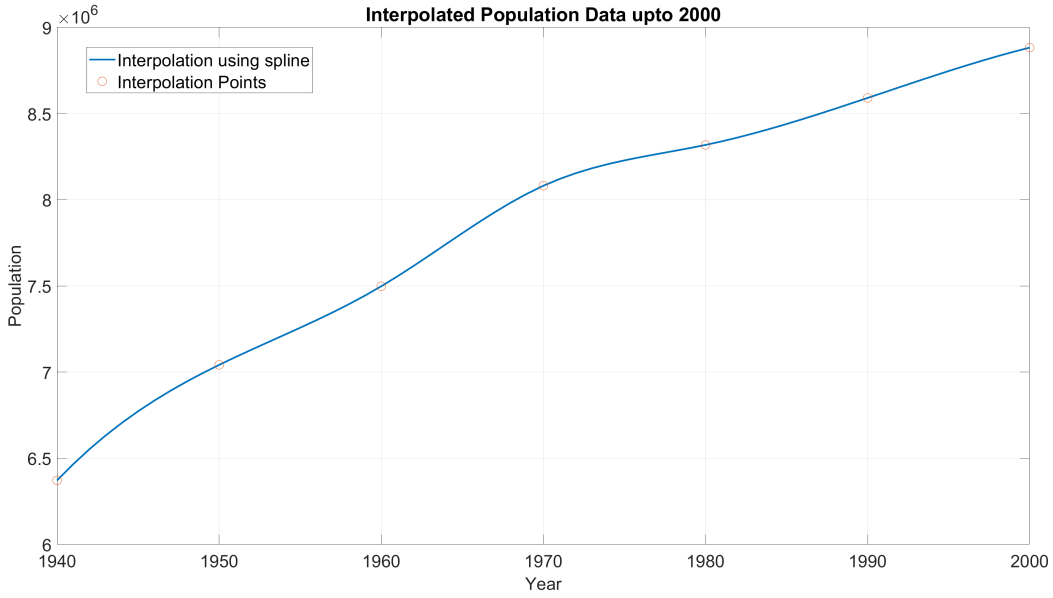

Figure 6: The interpolation from the built in MATLAB function spline for a one-year interval from 1940 to 2000.

The curve in figure 6 is less "curvy" than the corresponding interpolations from *Problem 2* and *Problem 4*. It might be more representative for the actual population growth in the time interval. If accuracy is what we are after then the spline interpolation might be the preferred option, built in MATLAB functions are usually great. But I feel that manually writing the other interpolation types were actually quite rewarding and gave me an insight into how this stuff works. Perhaps it's a trade off. If you want quick and accurate results, use MATLAB'S built in functions. If you want more control over the process and insight into what is going on, try writhing your on (given that you have the time to do so).