ISC3, Fall 2020 (A22)
Computer works report 001

Pierre Lague

September 23, 2022

1 Exercice 1 : Runge phenomenon, cubic splines

In this exercice we consider the function f defined in [-1, 1] by :

$$f(x) = \frac{1}{1+25x^2} \tag{1}$$

From f we want to generate a dataset made of the couples $(x_i, y_i)_{i=0,\ldots,n}$ defined by $x_i = -1 + \frac{2*i}{n}$, $y_i = f(x_i)$. We are looking for the polynomial p_n of degree less than or equal to n such that $p_n(x_i) = f(x)$. For that we use the Lagrange polynomials $L_i(x)$ seen in the course.

1.1 a) Implementing the *poly_interp* function

Using the Lagrange Polynomial function given in the course, write a

`Scilab`

function :

```
function y = poly_interp(x, xi, yi)
// (xi, yi) -> dataset
// returns the interpolation value y at point x
...
```

The function is as follow :

```
 function y=poly_interp(x, xi, yi)
        y = zeros(x) //initialize y as 0 matrix of size x
        l = length(xi);
        for i=1:l
           y = y + yi(i) * LagrangePol(x,xi,i);
        end;
    endfunction
```

After implementing the function we have to write the main part of the script. I've decided to do a $for\ n = 5 : 3 : 15$ loop in order to have 3 interpolation polynomials and to observe the *Runge Unstability Phenomenon* as N increases. The script is as follows :

```
for n=5:3:15

    //data
    xval=linspace(-1,1,200)';
    yi = 0, xi = 0;
    for i = 1:n
        xi(i) = (-1 + ((2*i)/n))
        yi(i) = (1 ./ (1+25*xi(i).^2))
    end

    //plotting
    xx_p = xval;
    yy_p = poly_interp(xx_p, xi, yi)
    plot2d(xx_p,yy_p, rect=[-1,-0.5,1,1.5], n);//interpolation polynomial
    plot2d(xi,yi,rect=[-1,-0.5,1,1.5], -1);//interpolation points

end
xlabel("x");ylabel("y");
title("Polynomial interpolation");
```

The following plot is the result we obtained :

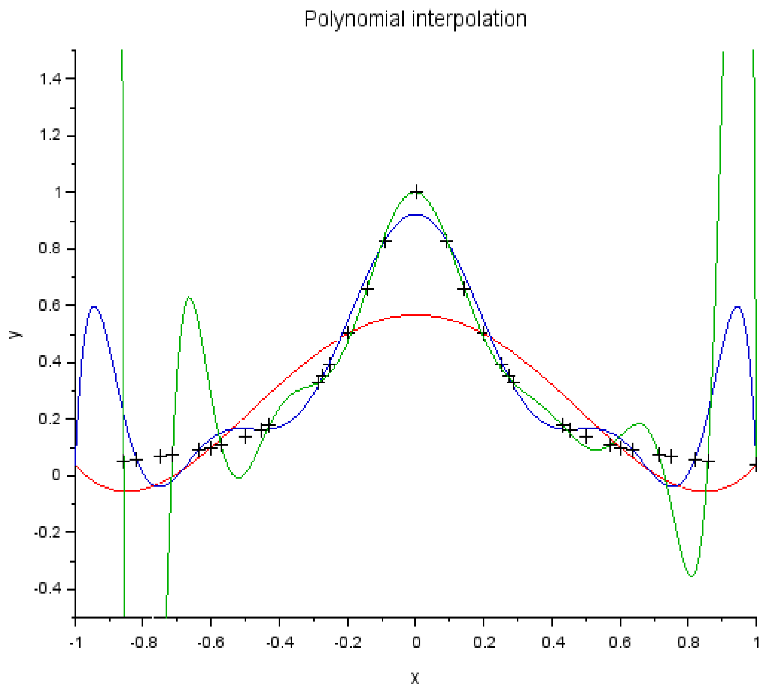

Figure 1: Observing the Runge Unstability Phenomenon as N increases. Lack of convergence is obivous.

1.2 b) Use the Scilab pre-implemented splin() and interp() to interpolate the data by means of cubic splines. Compare with polynomial interpolation.

For this question I used the pre-implemented functions in scilab to see if the interpolation with cubic splines presents the same Unstability phenomenon as interpolation with polynomials.

The code is as follows:

```
    //working with built in methods
//creating the points
for n=5:3:15


    yi = 0, xi = 0;
        for i = 1:n
            xi(i) = (-1 + ((2*i)/n))
```

```
            yi(i) = (1 ./ (1+25*xi(i).^2))
        end

    //data
    xval=linspace(-1,1,101)';

    //spline
    d = splin(xi, yi, 'not_a_knot')
    xx_s=xval;

    //interpolation
    yy_s=interp(xx_s,xi,yi,d,"linear");

    plot2d(xi,yi,-1);
    plot2d(xx_s,yy_s,n);
    plot2d(xi,yi,n+3);

end
xlabel("x");ylabel("y");
title("Spline interpolation");

//Cubic spline interpolation applied to the Runge function seem to avoid the
//Runge Unstability phenomenon.
//When we put 15 points on the other one, the phenomenon was clearly visible.
//Here it's much smoother.
```

The plot we obtained is the following :

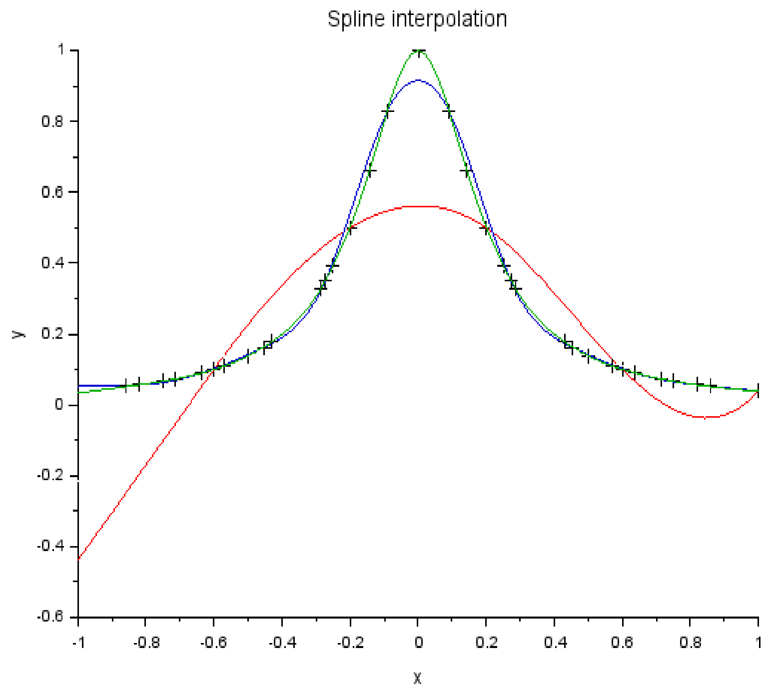

Figure 2: Spline Inerpolation prevents the Unstability Phenomenon from hapenning.

By comparing the two plots, we can see that the Polynomial Interpolation Method shows a lack of convergence very early for $N = 5 : 3 : 15$. However the Spline Interpolation Method shows that there is a much smoother convergence when N increases.

2 Exercice 2: Construction of a cubic spline by hand

Since the exercice was done in class, I will put my code and the plot I got after the execution.

```
     //important functions

function [r1, r2]=eval_spline(x, z)
    coefs = end_spline(z)
    r1=0, r2=0;
    for i=1:4
```

```
        r1 = r1 + coefs(i)*x.^(i-1)
        r2 = r2 + coefs(i+4)*x.^(i-1)
    end
endfunction

function render_spline(z, arg)
    space = linspace(0.5, 3.5, 200)
    x1 = linspace(0.5, 2, 200)
    x2 = linspace(2, 3.5, 200)
    [r1, _] = eval_spline(x1, z) //not the most optimal way, but it works
    [_, r2] = eval_spline(x2, z)
    plot(x1,r1, arg)
    plot(x2,r2, arg)
endfunction

plot([1, 2, 3], [1, 2, 0], 'o')
render_spline(5, '-b')
render_spline(7, '-r')
render_spline(10, '-g')
```

The resulting plot is the following :

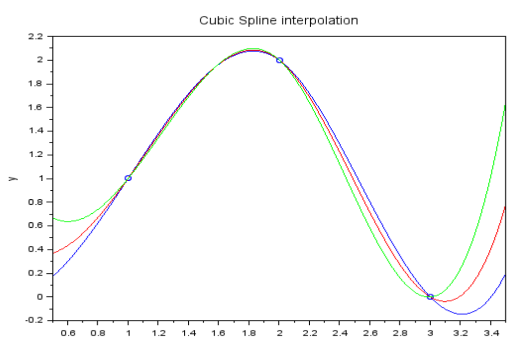

Figure 3: Interpolated points using a cubic spline. Multiple values for N.