# Numerical Analysis. Lab 3

## Thibaud Germain (GTZJ79) & Baglan Aitu (G956V1)

# **Exercise 1 – Runge’s phenomenon**

## Classic method

We observe huge errors at the edges of the interval (close to -5 and 5) with the classical method.



The results of the norm aren’t conclusive, we have values too close to 0, while we should have values that grows.

## Better choice of points

As we use a higher degree, the error decrease with a smarter choice of points.



Results of infinity norm with this method.

|  |  |
| --- | --- |
|  |  |
| 1 | 0.7547 |
| 2 | 0.5811 |
| 3 | 0.3937 |
| 4 | 0.2423 |
| 5 | 0.1369 |
| 6 | 0.0697 |
| 7 | 0.0323 |
| 8 | 0.0084 |
| 9 | 0.0054 |
| 10 | 0.0095 |
| 11 | 0.0088 |
| 12 | 0.0081 |

## Code

|  |
| --- |
| %% Ex1 - part1  fun = @(x) 1./(1 + x.^2);  x = linspace(-5,5,12);  y = fun(x);    nb\_points = linspace(2,24,12);  results = zeros(size(nb\_points,2), size(x,2));    index = 1;  for i = nb\_points  x\_in = linspace(-5,5,i);  y\_in = fun(x\_in);  results(index,:) = lagrange\_interpolation(x\_in,y\_in,x);  index = index +1;  end  inf\_norm = zeros(1, numel(nb\_points));    %Calculation of infinity norm  for i = 1:numel(nb\_points)  diff = abs(y-results(i,:));  inf\_norm(i) = max(diff);  end    disp(inf\_norm);    %% Ex1 - part 2  results\_2 = zeros(size(nb\_points,2), size(x,2));    for j = 1:numel(nb\_points)  x\_in = 1:nb\_points(j);  x\_in = 5\*cos( ((2\*x\_in - 1)\*pi) /(2\*(nb\_points(j))) );  y\_in = fun(x\_in);    results\_2(j,:) = lagrange\_interpolation(x\_in, y\_in,x);  end    inf\_norm\_2 = zeros(1, numel(nb\_points));    for j = 1:numel(nb\_points)  diff = abs(y-results\_2(j,:));  inf\_norm\_2(j) = max(diff);  end      function y = lagrange\_interpolation(x\_in, y\_in, x)  length\_in = length(x\_in);  length\_base = numel(x);  y = zeros(size(x));    for i = 1:length\_base  for j = 1:length\_in  lk = 1;  for k = 1:length\_in  if(j ~= k)  lk = lk \* (x(i) - x\_in(k))/(x\_in(j)-x\_in(k));  end  end  y(i) = y(i) + lk\*y\_in(j);  end  end  end |

**Exercise 2. Spline interpolation**

|  |
| --- |
| t = 0:1/50:1;  f = @(x, t) sin(5\*pi\*x) \* cos(10\*pi\*t) + 2 \* sin(7\*pi\*x) \* cos(14\*pi\*t);  df = @(x, t) pi \* (5\*cos(10\*pi\*t)\*cos(5\*pi\*x) + 14\*cos(14\*pi\*t)\*cos(7\*pi\*x));  knot\_vector = t;  x = linspace(0,1,100); |

1. Linear spline interpolation

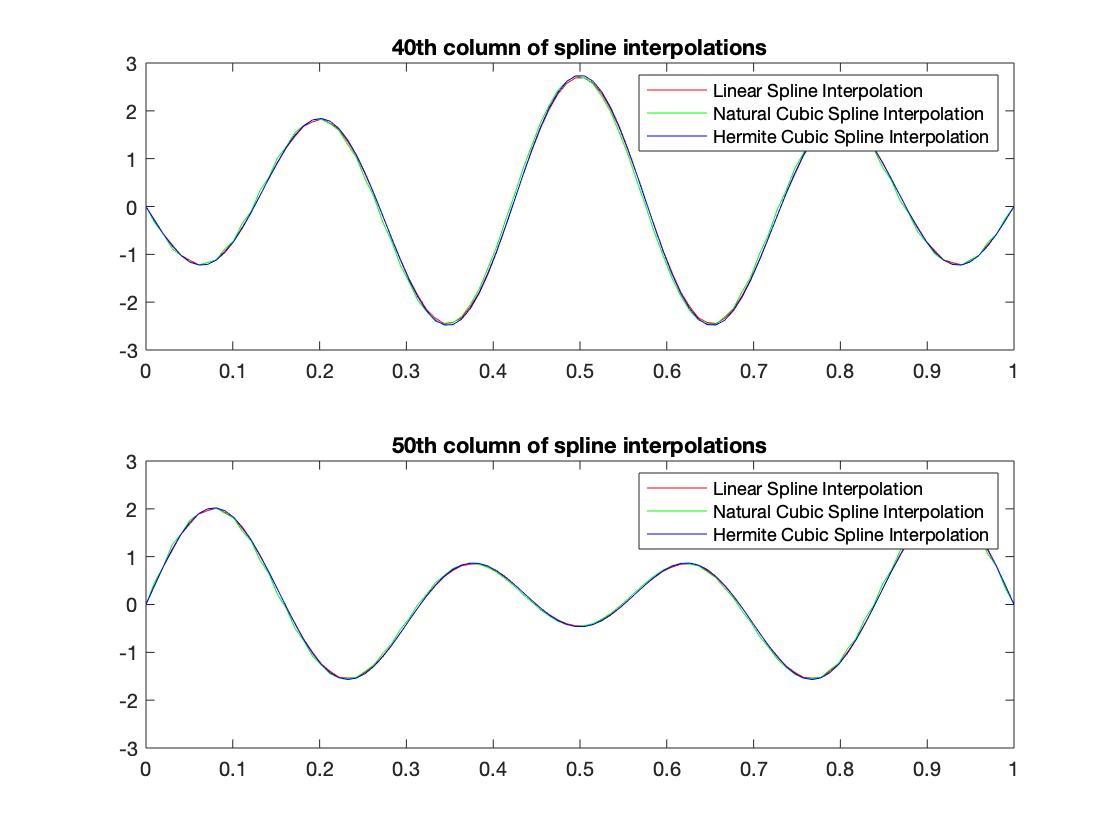
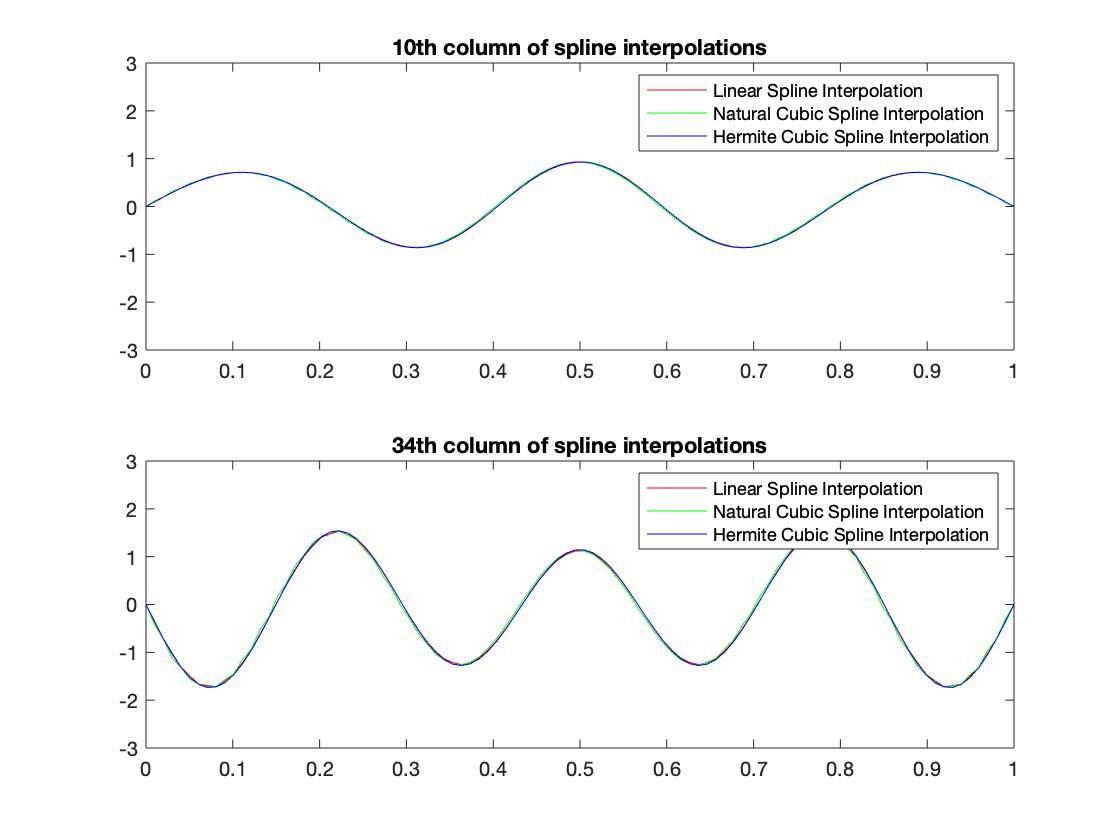
|  |
| --- |
| %% Linear spline interpolation and its plot.  linear\_output = [];  for i = 1:length(t)  linear\_output(i,:) = linear\_spline(f,knot\_vector,x,t(i));  %{  plot(x,linear\_output(i,:), 'r');  title(["t= ",t(i)]);  axis([0 1 -3 +3]);  %}  end  function [y] = linear\_spline(f,knot\_vector,x,t)  % LINEAR\_SPLINE Plots the interpolated function at a specific number of knot\_vector.  % f - the function reference  % knot\_vector - the knot values of X.  % x - the vector domain of the sampled function  % t - a time value for f  y= zeros(size(x));  for i = 1:length(x)  %Finding extreme case and searching for knots pair.  if(x(i) == knot\_vector(end))  y(i) = f(knot\_vector(end),t);  else  kn = x(i) >= knot\_vector;  knot\_pos = find(kn,1,'last');  x\_i = knot\_vector(knot\_pos);  x\_i\_1 = knot\_vector(knot\_pos+1);  % Linear interpolation between the knots..  y(i) = ((x\_i\_1-x(i))/(x\_i\_1-x\_i))\*f(x\_i,t)+ ((x(i)-x\_i)/(x\_i\_1-x\_i))\*f(x\_i\_1,t);  end  end  end |

1. Natural Cubic Spline Interpolation

|  |
| --- |
| %% Natural cubic spline interpolation and its plot.  natural\_output = [];  for i = 1:length(t)  natural\_output(i,:) = natural\_spline(f,knot\_vector,x,t(i));  hold off;  %{  plot(x,natural\_output(i,:), 'g');  title(["t= ",t(i)]);  axis([0 1 -3 +3]);  %}  end  function [y] = natural\_spline(f,knot\_vector,x,t)  %NATURAL\_SPLINE Plots the interpolated function at a specific number of knot\_vector.  % f - the function reference  % knot\_vector - the knot values of X.  % x - the vector domain of the sampled function  y = zeros(size(x));  m\_points = length(knot\_vector); %m+1 points  m = m\_points - 1;  %the matrix to solve.  M = zeros(m-1);  %the vector of dependent values.  V = zeros(m-1,1);  %the vector of steps. (m values)  h = zeros(m,1);  for i = 1:length(h)  h(i) = knot\_vector(i+1) - knot\_vector(i);  end  A = (h(1:m-1) + h(2:m,1))\*2;  B = h(2:m-1,1);  C = h(2:m-1,1);  %Fill tridiagonal matrix and vector  for i = 1:m-1  %Let's iterate over each row  M(i,i) = 2\*(h(i+1) + h(i));  if(1 ~= m)  M(i,1) = h(1);  end  %Take values of knot\_vector shifted.  v\_S1 = f(knot\_vector(i+2),t) - f(knot\_vector(i+1),t)/h(i+1);  v\_S2 = f(knot\_vector(i+1),t) - f(knot\_vector(i),t)/h(i);  V(i) = 6\*(v\_S1 - v\_S2);  end  %after solving this tridiagonal matrix, we add sigma\_0 =0 and sigma\_m = 0  %Get sigma from the linear equation system given by M and V.  sigma = zeros(m\_points,1);  sigma(2:end-1) = thomas\_algorithm(A,B,C,V);    %alpha and beta  alpha = zeros(m,1);  beta = zeros(m,1);  for i = 1:m  alpha(i) = 1/h(i)\*f(knot\_vector(i+1),t) - (1/6)\*(sigma(i+1)\*h(i));  beta(i) = 1/h(i)\*f(knot\_vector(i),t) - (1/6)\*(sigma(i)\*h(i));  end  for i = 1:length(x)  %Finding extreme case and searching for knots pair.  if(x(i) == knot\_vector(end))  y(i) = f(knot\_vector(end),t);  else  kn = x(i) >= knot\_vector;  knot\_pos = find(kn,1,'last');  x\_i = knot\_vector(knot\_pos);  x\_i\_1 = knot\_vector(knot\_pos+1);  hi = h(knot\_pos);  sigma\_0 = sigma(knot\_pos);  sigma\_1 = sigma(knot\_pos+1);    S1 = sigma\_0\*((x\_i\_1 - x(i))^3)/(6\*hi);  S2 = sigma\_1\*((x(i)-x\_i)^3)/(6\*hi);  S3 = alpha(knot\_pos)\*(x(i)-x\_i);  S4 = beta(knot\_pos)\*(x\_i\_1 - x(i));  y(i) = S1+S2+S3+S4;  end  end  end  function [y] = thomas\_algorithm(A,B,C,f)  n = length(f);  k = zeros(n,1);  y = k;  w = A(1);  y(1) = f(1)/w;  k(1)= C(1)/A(1);  for i=2:n-1  k(i) = C(i-1)/(A(i) - B(i)\*k(i-1));  end  for i = 2:n  y(i) = (f(i)-B(i-1)\*y(i-1))/(A(i)-B(i-1)\*k(i-1));  end  x(n)=y(n);  for j=n-1:-1:1  y(j) = y(j) - k(j)\*x(j+1);  end  end |

1. Hermit cubic spline interpolation

|  |
| --- |
| %% Hermite cubic spline interpolation and its plot.  hermite\_output = [];  for i = 1:length(t)  hermite\_output(i,:) = hermit\_spline(f,df,knot\_vector,x,t(i));  hold off;  %{  plot(x,hermite\_output(i,:),'b');  title(["t= ",t(i)]);  axis([0 1 -3 +3]);  %}  end  function [y] = hermit\_spline(f,df\_x,knot\_vector,x,t)    %HERMIT\_SPLINE Plots the interpolated function at a specific number of knot\_vector.  % f - the function reference  % knot\_vector - the knot values of X.  % x - the vector domain of the sampled function  % t - a time value for f  y= zeros(size(x));  for i = 1:length(x)  %Finding extreme case and searching for knots pair.  if(x(i) == knot\_vector(end))  y(i) = f(knot\_vector(end),t);  else  kn = x(i) >= knot\_vector;  knot\_pos = find(kn,1,'last');  x\_i = knot\_vector(knot\_pos);  x\_i\_1 = knot\_vector(knot\_pos+1);  % Polynomial coefficients  hi = x\_i\_1-x\_i;  C = zeros(4);  C(1) = f(x\_i,t);  C(2) = df\_x(x\_i,t);  C(3) = 3\*( f(x\_i\_1,t) -f(x\_i,t) )/(hi^2) - ( df\_x(x\_i\_1,t) + 2\*df\_x(x\_i,t) )/hi;  C(4) = ( df\_x(x\_i\_1,t) + df\_x(x\_i,t) )/(hi^2) - 2\*( f(x\_i\_1,t) - f(x\_i,t) )/(hi^3);    % Linear interpolation between the knots.  y(i) = C(1) + C(2)\*( x(i) - x\_i ) + C(3)\* ( x(i) - x\_i )^2 + C(4)\*( x(i) - x\_i )^3;  end  end  end |



As you can see on the plots above, we considered different cases of three spline interpolations. All interpolation methods gave approximately the same result. The reason for that high number of knots. If we decrease it, we can see clear the difference between them.

|  |
| --- |
| knot\_vector = linspace(0,1,15); |

