The Runge function \( f(x) = \frac{1}{1+25x^2} \) on \([-1, 1]\) provided an very nice function that was not well-approximated by its polynomials of interpolation. In fact, a higher degree (more points used in the interpolation) gave larger errors. Let’s recall the graphs using an interpolation at equally 11 spaced points of step size 1/5 and at the Chebyshev nodes.

\[
>> f = @(x) (1./(1+25*x.^2));
>> xn=-1:.2:1;
>> yn=f(xn);
>> x=-1:.001:1;z=f(x);
>> plot(x,z,xn,yn,’o’)
\]

The Interpolating polynomials plotted against the graph of \( f(x) \)

\[
>> If=int_poly(yn,xn,x);
>> plot(x,z,x,If)
>> i=0:10;cn=cos(((2*i+1)./(22))*pi);
>> fcn=f(cn);
>> Ifcn=int_poly(fcn,cn,x);
>>
>> plot(x,z,x,Ifcn)
>> max(abs(z-Ifcn))
\]

ans = 0.1092
Figure 1: The Runge Function $f(x) = \frac{1}{1+25x^2}$ and its interpolating polynomial at the points $-1 : .02 : 1$ at equally spaced and at the Chebyshev nodes
Now we will look at the cubic spline interpolation of the Runge function at the same 11 points. We will do the natural spline. First we look at the programs:

The spline coefficients are determined by the data:

type spline3_coeff

function [a,b,c,d]=spline3_coeff(knots,data);
%%% function [a,b,c,d]=spline3_coeff(knots,data);
%%% Modification of Cheney Kincaid pseudo-code
%%% The natural spline for interpolating data at the knots a=x0<...<x_n=b;
%%% System of equations is solved using the tridiagonal nature
%%% WARNING: THIS PROGRAM REQUIRES AT LEAST 4 KNOTS
%%% INPUT:
%%% knots - distinct points (t_j) of interpolation as a row vector
%%% data - data at the interpolation points as a row vector
%%% OUTPUT:
%%% a - column vector of constant terms for the spline on [t_j,t_(j+1)]
%%% b - column vector of coefficients of x-t_j for the spline on [t_j,t_(j+1)]
%%% c - column vector of coefficients of (x-t_j)^2 for the spline on [t_j,t_(j+1)]
%%% d - column vector of coefficients of (x-t_j)^3 for the spline on [t_j,t_(j+1)]

% determine lengths of the intervals
n=length(knots)-1;
for j=1:n;
    h(j)=knots(j+1)-knots(j);
end;
% determine the right hand side of the system (without first and last eqn)
for i=2:n;
    rhs(i)=6*((data(i+1)-data(i))/h(i)-(data(i)-data(i-1))/h(i-1));
end;
% solve the system using tri_diag.m
z(1)=0;
z(n+1)=0;
dd=2*(knots(3:n+1)-knots(1:n-1));subd=h(1:n-2);supd=h(2:n-1);
z(2:n)=tri_diag(dd,subd,supd,rhs(2:n));
% find the other coefficients
for j=1:n;
    b(j,1)=(data(j+1)-data(j))/h(j) - h(j)*(z(j+1)+2*z(j))/6;
    d(j,1)=(z(j+1)-z(j))/(6*h(j));
end;
a=data(1:n)';
c=z(1:n)'/2;

%%% Notice that this program calls a program for computing a tri-diagonal system

type tri_diag

function x=tri_diag(d,subd,supd,b);
%%% function x=tri_diag(d,subd,supd,b);
% Solving a tri-diagonal system without pivoting
% INPUT
%    d -- main diagonal as a row vector
%    subd -- sub-diagonal as a row vector
%    supd -- superdiagonal as a row vector
%    b -- right hand side as a row vector
% OUTPUT
%    x -- solution as a row vector, or 'method fails'

%% Determine the size of the system
n=length(d);
flag=0;
for i=2:n;
    if flag==0
        if d(i-1)==0;
            flag=1;
        else
            xmult=subd(i-1)/d(i-1);
            d(i)=d(i)-xmult*supd(i-1);
            if d(i)==0
                flag=1;
            else
                b(i)=b(i)-xmult*b(i-1);
            end;
        end;
    end;
end;
if flag==0;
    x(n)=b(n)/d(n);
    for i=n-1:-1:1;
        x(i)=(b(i)-supd(i).*x(i+1))/d(i);
    end;
else
    x='method fails';
end;

%% Now we compute them for the Runge function at the 11 points:
[a,b,c,d]=spline3_coeff(xn,yn);

Now that we have the coefficients, if we want to plot the spline curve curve through those data points, we need to a program to evaluate the spline function:

type spline3_eval

function y=spline3_eval(a,b,c,d,knots,xx);
%% function y=spline3_eval(a,b,c,d,knots,xx);
%%
%% Compute the values at the points xx of the cubic spline with knots
%% 'knots' from the coefficients of its polynomial pieces
%%
%% INPUTS: (from output of spline3_coef.m)
%% knots - t_j
%% a - column vector of constant terms for the spline on [t_j,t_{j-1}]
%% b - column vector of coefficients of x-t_j for the spline on [t_j,t_{j-1}]
%% c - column vector of coefficients of (x-t_j)^2 for the spline on [t_j,t_{j-1}]
%% d - column vector of coefficients of (x-t_j)^3 for the spline on [t_j,t_{j-1}]
%% xx - points at which spline is to be evaluated
%% OUTPUT:
%% y - value of the spline

n=length(knots);
if length(a)~=n-1;
   message = 'knots and coefficients not matched'
end;
I=find(xx<knots(2));
if isempty(I)==0;
   y(I)=a(1)+(xx(I)-knots(1)).*(b(1)+... (xx(I)-knots(1)).*(c(1)+d(1)*(xx(I)-knots(1))));
end;clear I;
for j=2:n-2;
   I = find((xx>=knots(j))&(xx<knots(j+1)));
   if isempty(I)==0;
      y(I)=a(j)+(xx(I)-knots(j)).*(b(j)+... (xx(I)-knots(j)).*(c(j)+d(j)*(xx(I)-knots(j))));
   end;clear I;
end;
I=find( xx>=knots(n-1));
if isempty(I)==0;
   y(I)=a(n-1)+(xx(I)-knots(n-1)).*(b(n-1)+... (xx(I)-knots(n-1)).*(c(n-1)+d(n-1)*(xx(I)-knots(n-1))));
end;clear I;

%%% We set up the points to be 200 equally spaced points in the interval [-1,1]
%%% and then plot the interpolation points, the original function and the cubic spline interpolant:

xx=-1:.01:1;
yy=spline3_eval(a,b,c,d,xn,xx);
plot(xn,yn,'o',xx,yy,xx,f(xx))

%%% We also compute the error based on those 200 points
error=max(abs(yy-f(xx)))
error =
   0.0220

%%% As we will see in the graph, the spline function follows the curve very well
Figure 2: The Runge Function \( f(x) = \frac{1}{1 + 25x^2} \) and its interpolating cubic spline at the points \(-1 : .02 : 1\).