THE FAMOUS RUNGE EXAMPLE

Even if the function is extremely nice, the interpolating polynomial may not be a very good representation of the function at points other than the points where it interpolates. The function $f(x) = \frac{1}{1+25x^2}$ on $[-1, 1]$ will illustrate this. This function is very smooth: it has infinitely many derivatives, its graph is symmetric about the y-axis, and it decreases to zero in both directions as $|x| \to \infty$.

We begin by showing the programs for the Newton Form of Polynomial of Interpolation. The first generates the divided difference table for given data:

```
function  DD=dd_table(FUN,xn);
%%  function  DD=dd_table(FUN,xn);
%%
%%  THE DIVIDED DIFFERENCE TABLE for the function FUN at the distinct points xn
%%
%%  INPUT:
%%   xn  - the points at which the divided difference is taken as a row
%%         vector ,   e.g. [0,1,2,3,4]
%%    FUN  - the data for the difference which may be entered EITHER as
%%           a string, enclosed in ' ', with the name the m-file of the
%%           function,  OR as a row vector of data in the same order as xn
%%  OUTPUT:
%%    DD  - the matrix containing the divided difference table with xn
%%          as first column, the values of FUN at xn as 2nd column and
%%          k-th order divided difference as (k+2)nd column


n=length(xn)-1;  % determine the number of points

DD(:,1)=xn';   % points as first column in the table

if isstr(FUN)==1;  % test whether from a function file or from given data
   DD(:,2)=feval(FUN,xn'); %function values as 2nd column (0th order difference)
else
    DD(:,2)=FUN';  % given data as 2nd column
end;

for i=2:n+1;       %% an adjustment is made for indexing in MATLAB
  for j=3:i+1;
   DD(i,j)=(DD(i,j-1)-DD(i-1,j-1))/(xn(i)-xn(i-j+2));
  end;
end;
```

The next program gnerates the Newton Interpolating polynomial evaluated at a prescribed points for graphing or comparing values. It uses Matlab's ability to generate the values of P for a whole vector of points rather than individual values:

```
function P=int_poly(FUN,xn,x);
%%  function P=int_poly(FUN,xn,x);
%%
%%  THE POLYNOMIAL INTERPOLATING FUN at xn, generated from the divided
%%  difference table in Newton form and evaluated using a nested Horner
%%  type algorithm
%%
%%  INPUT:
%%    xn   - the interpolation points as a row vector e.g. [0,1,2,3,4]
%%    FUN  - the data to be interpolated which may be entered EITHER as
%%           a string, enclosed in ' ', with the name the m-file of the
%%           function ,  OR as a row vector of data in the same order as xn
%%     x   - the points at which the interpolating polynomial is to be
%%           evaluated as a row vector e.g., x=0:.01:1
%%  OUTPUT
%%     P   - values of the polynomial at x
%%


DD=dd_table(FUN,xn);  % get the divided difference table
n=length(xn)-1; % find the number of points

A=diag(DD(:,2:n+2)); % Get the coefficients of P from the top diagonal in
                     % the divided difference table

%%Now to evaluate the Newton polynomial with the Horner algorithm
P=A(n+1)*ones(size(x));
for j=n:-1:1;
      temp=P.*(x-xn(j))+A(j)*ones(size(x));
      P=temp;
end;
```

Now we use these two programs to show the example. Both of these programs are available on the course website. The function $f(x) = \frac{1}{1+25x^2}$ on $[-1, 1]$ is interpolated at equally spaced points of step size $1/5$, a total of 11 points for a polynomial of degree 10.

```
>> 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')
```

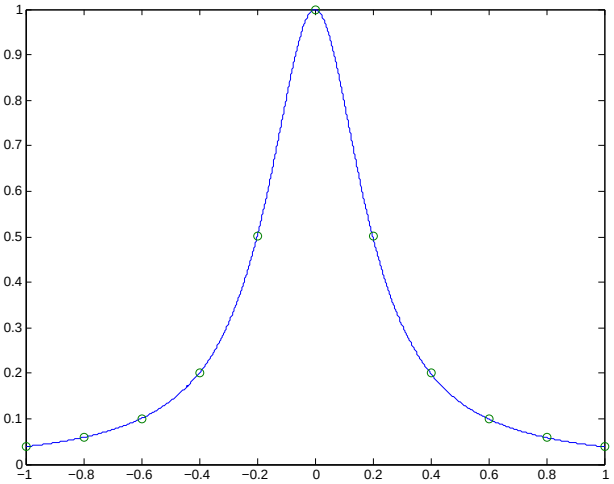

Figure 1: The Runge Function $f(x) = \frac{1}{1+25x^2}$ and the 11 interpolation points.

We next plot polynomial of degree 10 that interpolates the function at equally spaced nodes against the graph of $f(x)$ itself. It turns out that equal spacing, though often what is necessarily done in practice, is not the best points to use when doing polynomial interpolation. In the second figure we generate a polynomial of degree 10 that interpolates the function at a more optimal set of points, the so-call Chebyshev nodes which we will discuss in a little more detail in class.

```
>> If=int_poly(yn,xn,x);
>> plot(x,z,x,If)
>> i=0:10;cn=cos(((2*i+1)./(22))*pi); %The chebyshev nodes
>> fcn=f(cn);
>> Ifcn=int_poly(fcn,cn,x);
>> plot(x,z,x,Ifcn)
>> max(abs(z-Ifcn))  %error in approximation in interpolation at the cheyshev nodes
=    0.1092
```

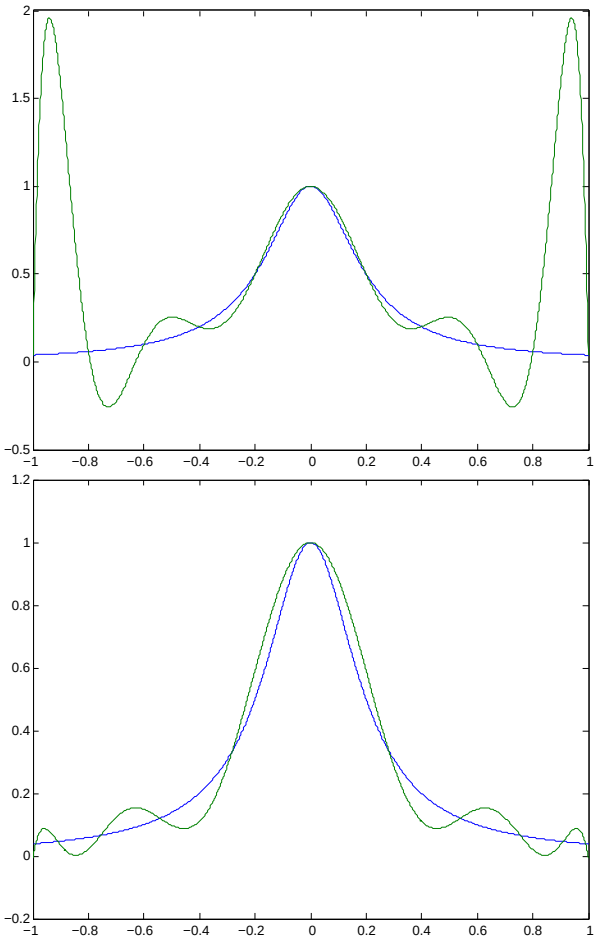

Figure 2: The Runge Function $f(x) = \frac{1}{1+25x^2}$ and its interpolating polynomial at the points $-1 : .02 : 1$ at equally spaced and Chebyshev nodes

For equally spaced points, the error gets worse as the number of interpolation points increases. It is because polynomials are stiff, it takes real effort to change direction as you can see in the first figure.

Error Estimate:

The maximum value of the 11th derivative of the Runge function in $[-1, 1]$ is approximately

$$\max_{-1\le x\le 1} \frac{1}{1+25x^2} = 1.772190773 \times 10^{15} \Longrightarrow \max_{-1\le x\le 1} \frac{f^{(11)}(x)}{11!} = 4.439711532 \times 10^7,$$

while the maximum in $[-1, 1]$ for the $\prod_{0\le j\le n} |x - x_j|$ in the case of equally spaced points is

$$\max_{-1\le x\le 1} (abs((x^2-1) * (x^2-.8^2) * (x^2-.6^2) * (x^2-.4^2) * (x^2-.2^2) * x)) = 0.008532263869$$

. Thus, the error estimate in the equally spaced case is

$$\max_{-1\le x\le 1} \frac{1}{(n+1)!(1+25x^2)} \max_{-1\le x\le 1} (abs((x^2-1)*(x^2-.8^2)*(x^2-.6^2)*(x^2-.4^2)*(x^2-.2^2)*x)) = 3.7880\ldots\times 10^5$$

which is essentially useless. The error estimate for the Chebyshev nodes isn't much better'

$$\max_{-1\le x\le 1} \frac{1}{(n+1)!(1+25x^2)} \frac{1}{2^{11}} = 21678.27896\ldots$$