To see how good and bad various interpolation methods can be, use Matlab’s interpolation routines on data generated from Runge’s function:

\[ f(x) = \frac{1}{1+x^2}. \]

In Matlab, do the following:

1. Problem setup:

   Generate \( N + 1 = 11 \) equally-spaced nodes \( x_i \) in the interval \([-5, 5]\)

   \[
   \begin{align*}
   N &= 10; \\
   x &= \text{linspace}(-5, 5, N+1); \quad \% \text{to see values, omit the ;}
   \end{align*}
   \]

   and then evaluate \( f(x) \) at these nodes

   \[
   \begin{align*}
   f &= \text{inline}('1./(1+x.*x)', 'x'); \\
   y &= f(x);
   \end{align*}
   \]

   The \( N + 1 \) points \((x_i, y_i)\) are the data points to be interpolated by various methods.

   Plot them to see where they are

   \[
   \begin{align*}
   \text{plot}(x, y, 'o') \\
   \text{title}('N+1 = 11 \text{ equally-spaced data points}')
   \end{align*}
   \]

   Also generate lots of points \( t_i \) at which to evaluate \( f \), and the interpolants, for plotting

   \[
   t = [-5:.1:5];
   \]

   Evaluate \( f \) at these \( t_i \)’s and plot \( f(t) \) in a new figure window

   \[
   \begin{align*}
   \text{figure;} \\
   \text{plot}(t, f(t), '-') \\
   \text{title('Runge function')}
   \end{align*}
   \]
2. Nth degree interpolating polynomial:

Use Matlab’s `polyfit` to construct (the coefficients of) the Nth degree interpolating polynomial using the equally spaced nodes

```matlab
PN = polyfit(x,y,N);
```

Now this can be evaluated anywhere in the interval [-5,5], e.g., at the \( t_i \)’s

```matlab
v = polyval(PN,t);
```

Find the inf-norm error \( \| f(t) - PN(t) \|_\infty \)

```matlab
err = norm(f(t)-v,inf)
```

and plot both \( f(t) \) and \( PN(t) \) on the same plot as the data points

```matlab
figure;
plot(x,y,'o',t,f(t),'-',t,v,'--')
title(sprintf('f(t) and PN_{10}(t), err=%g',err))
```

3. Interpolation at Chebychev nodes:

Generate \( N + 1 = 11 \) Chebychev nodes

```matlab
K = N+1;
a=-5;
b=5;
xcheb=zeros(1,K);
for i=1:K
xcheb(i)=(a+b)/2 + (b-a)/2 * cos( (i-.5)*pi/K );
end
ycheb = f(xcheb);
```

Follow the steps in 2 to produce the Nth degree interpolating polynomial \( PN_{\text{cheb}} \) based on the Chebychev nodes \( x_{\text{cheb}} \) and the data \( y_{\text{cheb}} \). Then compute the function values \( v_{\text{cheb}} \) at the \( t_i \)’s and the error \( \| f(t) - PN_{\text{cheb}}(t) \|_\infty \), and plot both \( f(t) \) and \( PN_{\text{cheb}}(t) \) on the same plot as the Chebychev data. Compare the error and the plot with those from 2. Comment on why one works better than the other.

4. Repeat 1, 2 and 3 with \( N = 20 \) and \( N = 50 \). Explain what behavior you observe.
5. *(Optional)* Piecewise linear interpolation:

Use Matlab’s `interp1` to construct the piecewise linear interpolant of the original data points from 1 evaluated at the \( t_i \)’s

\[
\text{vlin} = \text{interp1}(x,y,t,\text{'linear'});
\]

Repeat the steps of 2 to compute the error and plot. Compare error and plot with those from the previous examples.

6. *(Optional)* Piecewise cubic interpolation:

Use Matlab’s `interp1` to construct the piecewise cubic interpolant of the original data points from 1 evaluated at the \( t_i \)’s

\[
\text{vcub} = \text{interp1}(x,y,t,\text{'cubic'});
\]

Repeat the steps of 2 to compute the error and plot. Compare error and plot with those from the previous examples.

7. *(Optional)* Cubic spline interpolation:

Use Matlab’s `interp1` to construct the cubic spline interpolant of the original data points from 1 evaluated at the \( t_i \)’s

\[
\text{vspl} = \text{interp1}(x,y,t,\text{'spline'});
\]

Repeat the steps of 2 to compute the error and plot. Compare error and plot with those from the previous examples.