IDEAS TO BE TAUGHT TODAY: INTERPOLATION

FIRST WE REVIEW: NEWTON'S METHOD

NEWTON'S METHOD IN $\mathbb{R}^n$ (NONLINEAR) SOLVES $F(x) = 0$

$$x_{n+1} = x_n - DF(x_n) \cdot F(x_n)$$

DF is the JACOBIAN MATRIX FOR $F$, $DF = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$

EXAMPLE: USE NEWTON'S METHOD WITH STARTING GUESS $(1,2)$ TO SOLVE $y = x^3$ WHERE THE VARIABLES $x$ AND $y$ ARE RESTRICTED ON THE UNIT CIRCLE.

Solution:

First we redefine the problem as $\sum y = x^3$
and reshape it to look like $F(u,v) = 0$
in other words $\sum y - x^3 = 0$
$\sum x^2 + y^2 - 1 = 0$

Now calculate $DF = \begin{pmatrix} -3x^2 & 1 \\ 2x & 2y \end{pmatrix}$
and then $DF(x_0) = \begin{pmatrix} -3 & 1 \\ 4 & 4 \end{pmatrix}$

and $DF^{-1}(x_0) = \frac{1}{16} \begin{pmatrix} 4 & -1 \\ -4 & -3 \end{pmatrix}$

So the first iteration is
\[ x_1 = \left( \frac{1}{2} \right) - \frac{1}{16} \left( \frac{4}{1} - 1 \right) \left( \frac{1}{4} \right) = \left( \frac{1}{1} \right) \]

Note our current approximate solution is \( \left( \frac{1}{1} \right) \) but will improve with more iterations.

**Note:** Exact solution: 
(from Matlab)
\[
\begin{align*}
 u &= 0.8260313577, \\
 v &= 0.5636241622
\end{align*}
\]

**Practical Problems with the Jacobian**

- Jacobian may be hard to calculate or even impossible. Instead we might have to approximate it.

- We always avoid calculating the inverse \( DF^{-1} \).

  Instead, we define \( Y = DF^{-1} \cdot F \) and re-write as \( DF \cdot Y = F \).

**Note:** We know both matrices \( DF \) and \( F \) and wish to calculate \( Y \). We can do this now by performing LU factorization or another similar such method.

**Polynomial Interpolation**

Def: A polynomial \( p \in P_n \) interpolates the points \((x_i, y_i)\) for \( i = 0, \ldots, n \) if \( p(x_k) = y_k \) for \( k \in \{0, \ldots, n\} \).

Why things work (with interpolation): Weierstrass approximation.
Theorem: Suppose \( f(x) \) is a continuous function on \([a, b]\). Then for every \( \varepsilon > 0 \), there exists a polynomial \( P(x) \) defined on \([a, b]\) such that:
\[
|f(x) - P(x)| \leq \varepsilon \quad \forall x \in [a, b]
\]
that is, we can find a polynomial as close as we like to any given function \( f(x) \).

**Lagrange Interpolation**

Given \( n+1 \) points or support \((x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\), we would like to find a polynomial of degree \( n \)
\[
P_n(x) = a_0 + a_1 x + \cdots + a_n x^n
\]
which passes through all of them, i.e., \( P_n(x_k) = y_k \quad \forall k \).

**Problem:** A single polynomial of degree \( n \) satisfying the \( n+1 \) points has already been solved by Lagrange!

**Solution:** Let \( \quad P_n(x) = \sum_{i=0}^{n} y_i \cdot L_i(x) \), note that \( y_i \) is given!

- **Example:** Given the points \((-1, -7), (1, 7), (2, -4), (5, 35)\) construct the Lagrange interpolating polynomial.

Through these points, use it to approximate \( f(3) \).

- \( L_1(x) = \frac{x - x_0}{x_1 - x_0} \cdot \frac{(x - x_1)(x - x_2)(x - x_3)}{(x_1 - x_0)(x_1 - x_2)(x_1 - x_3)} \)
- \( L_2(x) = \frac{x - x_0}{x_2 - x_0} \cdot \frac{(x - x_0)(x - x_1)(x - x_3)}{(x_2 - x_0)(x_2 - x_1)(x_2 - x_3)} \)
- \( L_3(x) = \frac{x - x_0}{x_3 - x_0} \cdot \frac{(x - x_0)(x - x_1)(x - x_2)}{(x_3 - x_0)(x_3 - x_1)(x_3 - x_2)} \)

\( P_4(x) = -7L_1(x) + 7L_2(x) - 4L_3(x) + 35L_4(x) \)

\( P_4(3) = 4 \approx f(3) \)

**Lagrange Interpolation Error Estimate**

No numerical result has any meaning without an estimate for the error. We can prove that the error for Lagrange interpolation of a smooth function \( f(x) \) is
Advantages:
- Error estimate is provided
- Does not restrict nodes to be evenly spaced at $x$ (other methods require that)

Disadvantages:
- The error is very difficult to know in advance since normally we do not know the actual function $f(x)$
- We must recompute everything from the beginning if an extra point is added to the data (this does not happen with some other clever interpolating methods).

Note that so far we have examined the following interpolating method:

**Lagrange** $\phi(x) = \sum_{k=0}^{n} y_k \cdot L_k^n(x)$ Basis: $\{L_0^n(x), L_1^n(x), \ldots, L_n^n(x)\}$

**Question:** Can we find a way to minimize our error of interpolation even more (without changing the number of points used)?

**Error:** \( \left| \frac{f^{(n+1)}(\xi)}{(n+1)!} \right| \prod_{i=0}^{n} (x-x_i) \) where $x, \xi \in [x_0, x_n]$

This can become smaller if $x_i$'s are chosen cleverly!

Let us look at an applet regarding placing the $x_i$'s at different locations and discover what is Runge's phenomenon.
Question:
So how do we pick the nodes \( x_0, x_1, \ldots, x_n \) in 
\[ w(x) = \prod_{i=0}^{n} (x-x_i) = (x-x_0)(x-x_1) \cdots (x-x_n) \] 
in such a way so that we can minimize \( w(x) \) and 
avoid Runge's Phenomenon?

Answer: Not equidistant is a start, 
but how exactly?

Runge studied this and found that the 
oscillations diminish if we increase the number of nodes used 
towards the ends asymptotically given by the formula 
\[ \frac{\sqrt{1-x^2}}{\sqrt{n}}. \]

A standard example of such a set of nodes is the 
Chebyshev nodes where the maximum error is guaranteed to 
diminish as we increase the order of the polynomial.

Note: We still have not discussed how to find such 
nodes!
The nodes are the roots of the Chebyshev polynomials!

Def (Chebyshev Polynomials)
Chebyshev polynomials are orthogonal with respect to 
the weight \( \frac{1}{\sqrt{1-x^2}} \) on the interval \([-1, 1]\).

These are some of the Chebyshev polynomials on \([-1, 1]\):
\[
\begin{align*}
T_0(x) &= 1 \\
T_1(x) &= x \\
T_2(x) &= 2x^2-1 \\
T_3(x) &= 4x^3-3x
\end{align*}
\]

In general it is true that 
\[ T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \]

An alternate way to obtain the polynomials:
An alternate way to obtain the polynomials:

DEF: (Chebyshev Polynomials)

The Chebyshev polynomials are defined as

\[ T_n(x) = \cos \left[ n \arccos(x) \right] \text{ on } [-1, 1] \]

The roots of the Chebyshev polynomials are

\[ x_k = \cos \left( \frac{\pi (2k+1)}{2n+2} \right), k = 0, 1, \ldots, n \]

If we wish to apply Chebyshev to an interval \([a, b]\) instead of \([-1, 1]\) we use the change of variables

\[ x_{\text{new}} \rightarrow \frac{1}{2} (b-a)x + a + b \]

Example: Find the roots of the 3rd order Chebyshev polynomial in the interval \([1, 3]\).

Solution: note that here \(n=2\) and to get the 3 roots we must use the formula above with \(k=0, 1, 2\).

So let \(k=0\): \(x_0 = \cos \left( \frac{\pi (2 \cdot 0 + 1)}{2 \cdot 2 + 2} \right) = \cos \left( \frac{\pi}{6} \right) = 0.866\)

\(k=1\): \(x_1 = \cos \left( \frac{\pi (2 \cdot 1 + 1)}{2 \cdot 3 + 2} \right) = \cos \left( \frac{3\pi}{6} \right) = 0\)

\(k=2\): \(x_2 = \cos \left( \frac{\pi (2 \cdot 2 + 1)}{2 \cdot 4 + 2} \right) = \cos \left( \frac{5\pi}{6} \right) = -0.866\)

But those are the 3 roots on \([-1, 1]\)!

We must now convert them to roots on the interval \([1, 3]\) using the formula.

\[ x_{\text{new}} = \frac{1}{2} (3 - (-1)) x_{\text{old}} + (-1) + 3 = 3x_{\text{old}} + 1 \]

\[ x_{\text{new}} = 3 \times 0.866 + 1 = 3.598 \]
\[ x_{1,\text{new}} = \frac{1}{2} \left( 4 \times x_{1,\text{old}} + 2 \right) = 2.732 \]

\[ x_{1,\text{new}} = \frac{1}{2} \left( 4 \times x_{1,\text{old}} + 2 \right) = 1 \]

\[ x_{2,\text{new}} = \frac{1}{2} \left( 4 \times x_{2,\text{old}} + 2 \right) = -0.732 \]

So the 3 Chebyshev roots in the interval \([-1, 3]\) are: \(-0.732, 1, 2.732\).

Thus to interpolate with a Chebyshev polynomial we use the Lagrange interpolating polyn. **ALTHOUGH FOR THE NODES IN LAGRANGE WE USE THE CHEBYSHEV NODES IN ORDER TO MINIMIZE OUR ERROR.**