Announcements

• Office hours before class.
• Math Majors: Applications for scholarships are due 2/5:
  http://www.math.utah.edu/undergraduate/scholarships.php

Math 5600 Spring 2021 Notes of 2/1/21

• Recall the idea of transforming a root finding problem into a fixed point problem

\[ f(x) = 0 \iff x = g(x) \]

and solving that problem via the fixed point iteration

\[ x_0 \text{ given, } \quad x_{k+1} = g(x_k), \quad k = 0, 1, 2, \ldots, \]

where we iterate until some suitable error criterion, depending on the problem, is satisfied.

• Recall our drawings:

![Graph](image)

**Figure 1.** Fixed Point Iteration 1.
Figure 2. Fixed Point Iteration 2.

Figure 3. Fixed Point Iteration 3.
Figure 4. Fixed Point Iteration 4.
• So when do we get convergence?
• Suppose
  \[ x_{k+1} = g(x_k) \quad \text{and} \quad \alpha = g(\alpha) \]
• \( \alpha \) is the fixed point to which the iteration should converge.
• The \( k \)-th error is simply the difference between the true solution and the \( k \)-th approximation:
  \[ e_k = \alpha - x_k. \]
• A crucial item to study is error propagation, i.e., the precise nature of the dependence of the error on the previous error. For fixed point iteration we get:
  \[ e_{k+1} = \alpha - x_{k+1} \]
  \[ = \alpha - g(x_k) \]
  \[ = g(\alpha) - g(x_k) \]
  \[ = g'(\xi)(\alpha - x_k) \]
  \[ \text{by the mean value theorem, for some } \xi \text{ between } \alpha \text{ and } x_k \]
  \[ e_{k+1} = g'(\xi)e_k \]
• Thus
  \[ |e_{k+1}| = |g'(\xi)||e_k| \]
• We will have convergence if
  \[ |g'(\xi)| < 1 \]
  for all relevant values of \( \xi \).
• In particular we will have convergence if \( g' \) is continuous, \( |g'(\alpha)| < 1 \), and we start sufficiently close to \( \alpha \).
• What about Newton’s Method?

\[ g'(x) = x - \frac{f(x)}{f'(x)} \]

Math 5600 Spring 2021  Notes of 2/1/21  page 4
Order of Convergence

- A Taylor Series gives a more precise result:

\[
e_{k+1} = g(\alpha) - g(x_k)
= g(\alpha) - \left( g(\alpha) + g'(\alpha)(x_k - \alpha) + \frac{1}{2}g''(\alpha)(x_k - \alpha)^2 \\
+ \frac{1}{6}g'''(\alpha)(x_k - \alpha)^3 + \ldots \right)
= \frac{1}{p!}g^{(p)}(\alpha)e_k^p + O(e_k^{p+1})
\]

where

\[
g(\alpha) = \alpha, \quad g'(\alpha) = g''(\alpha) = \ldots = g^{(p-1)}(\alpha) = 0, \quad g^{(p)}(\alpha) \neq 0.
\]

If (1) holds we say that the iteration

\[
x_{k+1} = g(x_k)
\]

converges to \(\alpha\) of order \(p\).

- Newton’s Method is convergent of order 2.

\[\text{If } g'(\alpha) \neq 0 \text{ and } |g'(\alpha)| < 1 \text{ we say that the iteration converges linearly.}\]
Aitken Extrapolation

• Linearly convergent methods can be accelerated.
• If we knew \( g'(\alpha) \) we could improve the convergence:

\[
\alpha - x_{n+1} \approx g'(\alpha)(\alpha - x_n) = g'(\alpha)\alpha - g'(\alpha)x_n
\]

Thus

\[
(1 - g'(\alpha))\alpha \approx x_{n+1} - g'(\alpha)x_n
\]

and hence

\[
\alpha \approx \frac{x_{n+1} - g'(\alpha)x_n}{1 - g'(\alpha)}.
\]

• The basic idea of Aitken Extrapolation is to use this formula combined with an approximation of \( g'(\alpha) \):

\[
g'(\alpha) \approx \frac{g(x_n) - g(x_{n-1})}{x_n - x_{n-1}} = \frac{x_{n+1} - x_n}{x_n - x_{n-1}}.
\]

• We get the improved approximation

\[
\hat{x}_{n+1} = \frac{x_{n+1} - x_{n+1} - x_n}{1 - \frac{x_{n+1} - x_n}{x_{n} - x_{n-1}}} = \frac{x_{n+1}(x_n - x_{n-1}) - (x_{n+1} - x_n)x_n}{x_n - x_{n-1} - x_{n+1} + x_n} = \frac{x_{n+1}x_{n-1} - x_n^2}{x_{n+1} - 2x_n + x_{n-1}}
\]  \[ \tag{2} \]

• Of course, we can make this into a method in its own right:

\[
\hat{x}_{n+1} = \frac{g(g(x_{n-1}))x_{n-1} - g^2(x_{n-1})}{g(g(x_{n-1})) - 2g(x_{n-1}) + x_{n-1}}
\]
• Dropping the dependence on the original sequence gives

\[ x_{n+1} = G(x_n) \quad \text{where} \quad G(x) = \frac{g(g(x))x - g^2(x)}{g(g(x)) - 2g(x) + x}. \] (3)

• Exercise: Show that this iteration converges of order 2.
Example: \( g(x) = \cos x \)

\[ x_0 = 1, \quad x_{n+1} = \cos x_n \quad (4) \]

Clearly there is a fixed point \( \alpha \approx 0.7 \). Since \( g'(x) = -\sin x \) and \( -\sin 0.7 \approx -0.64 \) we expect the error in the iteration in the iteration (4) to be reduced by 36 percent at each step. To get an accuracy of \( 10^{-20} \), say, we require about 100 steps (exercise). This is born out by the following (abbreviated) maple calculation:
> restart;
> Digits := 20:
> x0 := 1:
> n := 0:
> lprint (n, x0):
> 0, 1
> for i from 1 to 120 do
> n := n + 1;
> x0 := evalf(cos(x0)):
> lprint (n, x0):
> end do:
>
> ... 

> 109, 0.73908513321516064161
> 110, 0.73908513321516064169
> 111, 0.73908513321516064163
> 112, 0.73908513321516064167
> 113, 0.73908513321516064165
> 114, 0.73908513321516064166
> 115, 0.73908513321516064165
> 116, 0.73908513321516064166
> 117, 0.73908513321516064165
> 118, 0.73908513321516064166
> 119, 0.73908513321516064165
> 120, 0.73908513321516064166
> quit

memory used=3.5MB, alloc=8.3MB, time=0.11
• When applying Aitken Extrapolation to this sequence we run into the problem that the denominator involves the difference of very close numbers which leads to a cancelation of significant digits, and thus severely diminishes the accuracy. This can be overcome (in maple) by carrying an increased number of digits. We get an accuracy of about 19 digits after 52 steps:
Math 5600 Spring 2021  Notes of 2/1/21  page 11
• With the second order method (3) We get the same accuracy after just 5 steps:

```maple
> restart:
> Digits := 50:
> x0 := 1:
> n := 0:
> lprint(n, x0):
0, 1
> for i from 1 to 120 do
> n := n + 1;
> x0 := evalf((cos(cos(x0))*x0-(cos(x0))^2) / (cos(cos(x0))-2*cos(x0)+x0))
> lprint(n, x0):
> end do:

1, .72801036146761709114866625005081662427456913301315
2, .739066966908673745824604779747442179470131610986
3, .73906513316607552665689165829064369503718913963434
4, .739065133121516064165495372799248441230680285366368
5, .73908513321516064165531208764204371128341454413679
...```

\[ f(x) = x - \cos x = 0 \]
Newton's method applied to the equivalent root finding problem

\[ f(x) = x - \cos x = 0 \]

behaves similarly:

```maple
> restart:
> Digits:=20:
> x0:=1:
> n:=0:
> for i from 1 to 5 do
>   n:=n+1;
>   x0:=evalf(x0-(x0-cos(x0))/(1+sin(x0))):
>   lprint(n,x0):
> end do:
1, .75036386784024389303
2, .73911289091136167036
3, .73908513338528396976
4, .73908513321516064166
5, .73908513321516064165
> quit
memory used=2.1MB, alloc=8.3MB, time=0.09
```

Note that Newton's Method does not require the division by small numbers.
How to Stop

• We would like to stop when

\[ |e_n| = |\alpha - x_n| \leq \epsilon \]

for some tolerance \( \epsilon \) that depends on the problem. (e.g., 1 cm in the term project).

• We know that

\[ |e_{n+1}| \approx |g'(\alpha)||e_n| \]

• Suppose that

\[ |e_{n+1}| \leq L|e_n| \]

for some \( L \) and all \( n \).

• We can estimate

\[ L \approx \left| \frac{x_{n+1} - x_n}{x_n - x_{n-1}} \right|. \quad (5) \]

• Here is a common trick. Do nothing by adding and subtracting a suitable term. We know that

\[
\begin{align*}
e_n &= \alpha - x_n \\
&= \alpha - x_{n+1} + x_{n+1} - x_n \\
&= e_{n+1} + x_{n+1} - x_n
\end{align*}
\]

By the triangle inequality:

\[
|e_n| \leq |e_{n+1}| + |x_{n+1} - x_n| \\
\leq L|e_n| + |x_{n+1} - x_n|
\]

Solving for \( |e_n| \) gives

\[
|e_n| \leq \frac{|x_{n+1} - x_n|}{1 - L}
\]
• We don’t usually know $L$ exactly, but we can use the approximation (5).

• For methods of order greater than 1, like Newton’s Method, we know that $g'(\alpha) = 0$. Thus we pretend that $L = 0$ and use the estimate

$$|e_n| \approx |x_{n+1} - x_n|$$

and stop when

$$|x_{n+1} - x_n| < \epsilon.$$
Newton’s Method again

- When precisely is Newton’s Method convergent of order at least 2? \( f'(x) \neq 0 \)
- When is it of order less than 2, and when it is of order greater than 2?

\[
\begin{align*}
g(x) &= x - \frac{f}{f'} \\
g' &= \frac{f f''}{f^2}
\end{align*}
\]
Example:

\[ f(x) = x^2 \]

\[ x_0 = 1 \]

\[ g(x_n) = x_n - \frac{f(x_n)}{f'(x_n)} \]

\[ = x_n - \frac{x_n^2}{2x_n} = \frac{x_n}{2} \]

\[ \lambda = 0 \]

\[ f'(x) \neq 0 \]

\[ g'(x) = \frac{f f''}{f'} - \frac{ff'' + f'f''}{f'} \]

\[ g'' = \frac{ff'' + f'f''}{f'} - \frac{ff''}{f'} - \frac{ff''}{f'} - \frac{ff''}{f'} \]

\[ \text{numerator} = 0 \text{ if } f''(x) = 0 \]
Higher Orders of Convergence

- Fixed point iterations of arbitrarily high order can be constructed by inverse interpolation.