Root finding continued

Eugeniy E. Mikhailov

The College of William & Mary

Lecture 06
Secant method

Secant method converges with \( m = \frac{1 + \sqrt{5}}{2} \approx 1.618 \).

Need to provide two starting points \( x_1 \) and \( x_2 \).

\[
x_{i+2} = x_{i+1} - f(x_{i+1}) \frac{x_{i+1} - x_i}{f(x_{i+1}) - f(x_i)}
\]
Newton-Raphson method

Newton-Raphson method converges quadratically \((m = 2)\).

\[
x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}
\]

Need to provide a starting points \(x_1\) and the derivative of the function. Newton-Raphson method converges quadratically \((m = 2)\).
Numerical derivative of a function

Mathematical definition

\[ f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h} \]

The initial intent is to calculate it at very small \( h \).

Remember about roundoff errors (HW01).

For computers with \( h \) small enough

\[ f(x + h) - f(x) = 0. \]

Let's be smarter. Recall Taylor series expansion

\[ f(x + h) = f(x) + f'(x) \frac{1}{1!} h + f''(x) \frac{1}{2!} h^2 + \cdots \]

So we can see

\[ f'(x) = \frac{f(x + h) - f(x)}{h} = f'(x) + f''(x) \frac{1}{2} h + \cdots \]

Here computed approximation and algorithm error.

There is a range of optimal \( h \) when both the round off and the algorithm errors are small.
Numerical derivative of a function

Mathematical definition

\[ f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h} \]

The initial intent is to calculate it at very small \( h \).
Remember about roundoff errors (HW01).
Numerical derivative of a function

Mathematical definition

\[ f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h} \]

The initial intent is to calculate it at very small \( h \).
Remember about roundoff errors (HW01).
For computers with \( h \) small enough \( f(x + h) - f(x) = 0 \).
Numerical derivative of a function

Mathematical definition

\[ f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h} \]

The initial intent is to calculate it at very small \( h \).
Remember about roundoff errors (HW01).
For computers with \( h \) small enough \( f(x + h) - f(x) = 0 \).
Let’s be smarter. Recall Taylor series expansion

\[ f(x + h) = f(x) + \frac{f'(x)}{1!}h + \frac{f''(x)}{2!}h^2 + \cdots \]
Numerical derivative of a function

Mathematical definition

\[ f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h} \]

The initial intent is to calculate it at very small \( h \).
Remember about roundoff errors (HW01).

For computers with \( h \) small enough \( f(x + h) - f(x) = 0 \).

Let’s be smarter. Recall Taylor series expansion

\[ f(x + h) = f(x) + \frac{f'(x)}{1!} h + \frac{f''(x)}{2!} h^2 + \cdots \]

So we can see

\[ f_c'(x) = \frac{f(x + h) - f(x)}{h} = f'(x) + \frac{f''(x)}{2} h + \cdots \]

Here computed approximation and algorithm error.
Numerical derivative of a function

Mathematical definition

\[ f'(x) = \lim_{{h \to 0}} \frac{f(x + h) - f(x)}{h} \]

The initial intent is to calculate it at very small \( h \).
Remember about roundoff errors (HW01).
For computers with \( h \) small enough \( f(x + h) - f(x) \approx 0 \).
Let’s be smarter. Recall Taylor series expansion

\[ f(x + h) = f(x) + \frac{f'(x)}{1!}h + \frac{f''(x)}{2!}h^2 + \cdots \]

So we can see

\[ f'_c(x) = \frac{f(x + h) - f(x)}{h} = f'(x) + \frac{f''(x)}{2}h + \cdots \]

Here computed approximation and algorithm error. There is a range of optimal \( h \) when both the round off and the algorithm errors are small.
Derivative via Forward difference

\[ f'_c(x) = \frac{f(x + h) - f(x)}{h} \]

Algorithm error for small \( h \)

\[ \varepsilon_{fd} \approx \frac{f''(x)}{2} h \]
Derivative via Forward difference

\[ f'_c(x) = \frac{f(x + h) - f(x)}{h} \]

Algorithm error for small \( h \)

\[ \varepsilon_{fd} \approx \frac{f''(x)}{2} h \]

This is quite bad since error is proportional to \( h \).
Derivative via Forward difference

\[ f'_c(x) = \frac{f(x + h) - f(x)}{h} \]

Algorithm error for small \( h \)

\[ \varepsilon_{fd} \approx \frac{f''(x)}{2} h \]

This is quite bad since error is proportional to \( h \).

Example

\[ f(x) = a + bx^2 \]
Derivative via Forward difference

\[ f'_c(x) = \frac{f(x + h) - f(x)}{h} \]

Algorithm error for small \( h \)

\[ \varepsilon_{\text{fd}} \approx \frac{f''(x)}{2} h \]

This is quite bad since error is proportional to \( h \).

Example

\[ f(x) = a + bx^2 \]
\[ f(x + h) = a + b(x + h)^2 = a + bx^2 + 2bxh + bh^2 \]
Derivative via Forward difference

\[ f'_c(x) = \frac{f(x + h) - f(x)}{h} \]

Algorithm error for small \( h \)

\[ \varepsilon_{fd} \approx \frac{f''(x)}{2} h \]

This is quite bad since error is proportional to \( h \).

Example

\[
\begin{align*}
  f(x) &= a + bx^2 \\
  f(x + h) &= a + b(x + h)^2 = a + bx^2 + 2bxh + bh^2 \\
  f'_c(x) &= \frac{f(x + h) - f(x)}{h} = 2bx + bh
\end{align*}
\]
Derivative via Forward difference

\[ f'_c(x) = \frac{f(x + h) - f(x)}{h} \]

Algorithm error for small \( h \)

\[ \varepsilon_{fd} \approx \frac{f''(x)}{2} h \]

This is quite bad since error is proportional to \( h \).

Example

\[
\begin{align*}
  f(x) &= a + bx^2 \\
  f(x + h) &= a + b(x + h)^2 = a + bx^2 + 2bxh + bh^2 \\
  f'_c(x) &= \frac{f(x + h) - f(x)}{h} = 2bx + bh
\end{align*}
\]

So for small \( x \), the algorithm error dominate our approximation!
Derivative via Central difference

\[ f'_c(x) = \frac{f(x + h) - f(x - h)}{2h} \]
Derivative via Central difference

\[ f'_c(x) = \frac{f(x + h) - f(x - h)}{2h} \]

Algorithm error

\[ \varepsilon_{cd} \approx \frac{f'''(x)}{6} h^2 \]
Ridders method - smart variation of false position

Solve $f(x) = 0$ with the following approximation of the function $f(x) = g(x) \exp(-C(x - x_r))$, where $g(x) = a + bx$ i.e. linear. In this case if $g(x_0) = 0$ then $f(x_0) = 0$, but $g(x) = 0$ is trivial to solve.

One can say that

$$g(x) = f(x) \exp(C(x - x_1)) = a + bx$$

We chose $x_r = x_1$
Ridders method implementation

1. bracket the root between \( x_1 \) and \( x_2 \), i.e. function must have different signs at these points: \( f(x_1) \times f(x_2) < 0 \)
2. find the mid point \( x_3 = (x_1 + x_2)/2 \)
3. find new approximation for the root

\[
x_4 = x_3 + \text{sign}(f_1 - f_2) \frac{f_3}{\sqrt{f_3^2 - f_1 f_2}} (x_3 - x_1)
\]

where \( f_1 = f(x_1), f_2 = f(x_2), f_3 = f(x_3) \)

4. check if \( x_4 \) satisfies convergence condition and we should stop
5. rebracket the root, i.e. assign new \( x_1 \) and \( x_2 \), using old values
   - one end of the bracket is \( x_4 \) and \( f_4 = f(x_4) \)
   - the other is whichever of \((x_1, x_2, x_3)\) is closer to \( x_4 \) and provides proper bracket.
6. proceed to step 2

Nice features: \( x_4 \) is guaranteed to be inside the bracket, convergence of the algorithm is quadratic per cycle \((m = 2)\). But it requires evaluation of the \( f(x) \) twice for \( f_3 \) and \( f_4 \) thus it is actually \( m = \sqrt{2} \).
Bracketing algorithm are bullet proof and will always converge, however false position algorithm could be slow.

Newton-Raphson and secant algorithm are usually fast but starting points need to be close enough to the root.
Bracketing algorithm are bullet proof and will always converge, however false position algorithm could be slow.
Bracketing algorithms are bulletproof and will always converge, however false position algorithm could be slow.

Newton-Raphson and secant algorithm are usually fast but starting points need to be close enough to the root.
**Root finding algorithms summary**

**Root bracketing algorithms**
- bisection
- false position
- Ridders

Pro
- robust i.e. always converge.

Contra
- usually slower convergence
- require initial bracketing

**Non bracketing algorithms**
- Newton-Raphson
- secant

Pro
- faster
- no need to bracket (just give a reasonable starting point)

Contra
- may not converge

See Matlab built-in function `fzero` for equivalent tasks.