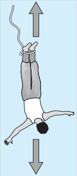
Lecture 3

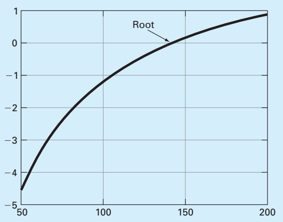
Finding Roots - Open Methods

Lecture in *Numerical Methods* from 17. March 2015

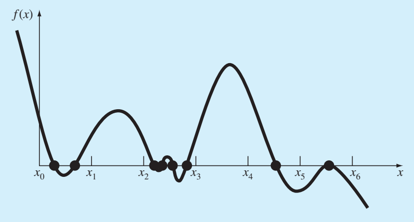
UVT



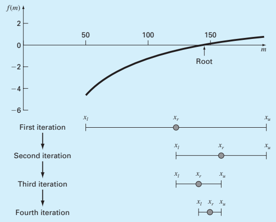
Bracketing methods - Overview



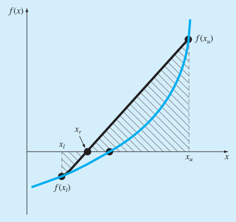
(a) Graphical method



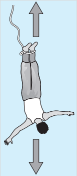
(b) Incremental search



(c) Bisection



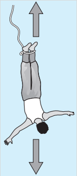
(d) False position



Agenda of today's lecture

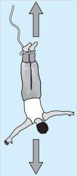
1. **Overview**

2. **Open methods**
   - Simple fixed-point iteration
   - Newton-Raphson
   - Secant methods
   - Brent's method



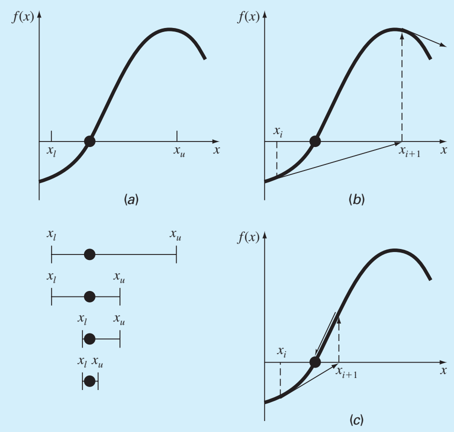
Open methods

For *bracketing methods* the root is located within an interval $\rightarrow$ repeated application always results in a closer estimates of the true value of the root $\rightarrow$ convergent methods

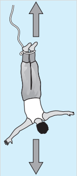


Open methods

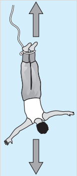
For *bracketing methods* the root is located within an interval → repeated application always results in a closer estimates of the true value of the root → convergent methods



Bisection (a) vs. Newton-Raphson (b) and (c)



Simple fixed-point iteration



Idea: Rearrange

$$f(x) = 0$$

to

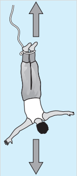
$$\mathbf{g}(\mathbf{x}) = \mathbf{x}$$

by algebraic manipulation or by adding x on both sides.
Utility: Easy-to-compute new estimates: $x_{i+1} = g(x_i)$.
Error estimator:

$$\varepsilon_a = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right| 100\%.$$

Example: Use simple fixed-point iteration to locate the root of $f(x) = \mathrm{e}^{-x} - x$. (initial guess $x_0 = 0$)

Fixed-point iteration: Example



$$f(x) = e^{-x} - x.$$

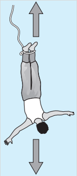
i	x_i	$\lvert\varepsilon_a\rvert$, %	$\lvert\varepsilon_t\rvert$, %	$\lvert\varepsilon_t\rvert_i / \lvert\varepsilon_t\rvert_{i-1}$
0	0.0000		100.000	
1	1.0000	100.000	76.322	0.763
2	0.3679	171.828	35.135	0.460
3	0.6922	46.854	22.050	0.628
4	0.5005	38.309	11.755	0.533
5	0.6062	17.447	6.894	0.586
6	0.5454	11.157	3.835	0.556
7	0.5796	5.903	2.199	0.573
8	0.5601	3.481	1.239	0.564
9	0.5711	1.931	0.705	0.569
10	0.5649	1.109	0.399	0.566

Fixed-point iteration: Example

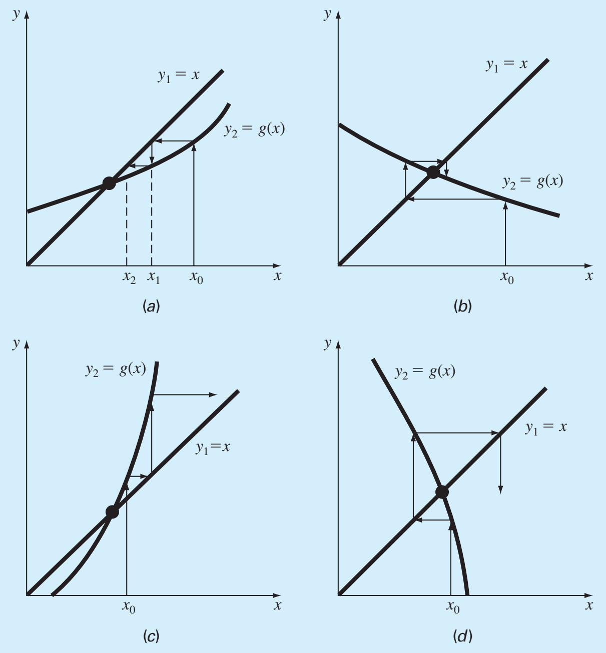
$$f(x) = e^{-x} - x.$$

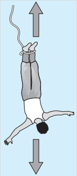
i	x_i	$\|\varepsilon_a\|$, %	$\|\varepsilon_t\|$, %	$\|\varepsilon_t\|_i / \|\varepsilon_t\|_{i-1}$
0	0.0000		100.000	
1	1.0000	100.000	76.322	0.763
2	0.3679	171.828	35.135	0.460
3	0.6922	46.854	22.050	0.628
4	0.5005	38.309	11.755	0.533
5	0.6062	17.447	6.894	0.586
6	0.5454	11.157	3.835	0.556
7	0.5796	5.903	2.199	0.573
8	0.5601	3.481	1.239	0.564
9	0.5711	1.931	0.705	0.569
10	0.5649	1.109	0.399	0.566

Each iteration brings the estimate closer to the true value of the root: 0.5671.

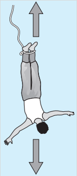


Convergence vs. Divergence





Simple fixed-point iteration: Error estimates

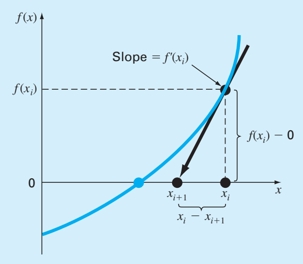


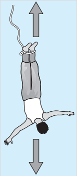
Error for any iteration is proportional to error from previous iteration multiplied by the absolute value of the slope of g:

$$E_{i+1} = g'(\xi)E_i.$$

If $|g'| < 1$ the errors decrease with each iteration.
If $|g'| > 1$ the errors grow with each iteration.
Remark: If $g' > 0$ the errors are positive, if $g' < 0$, the errors change sign!

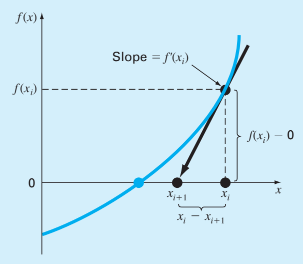
Newton-Raphson: most widely used of all root-finding methods

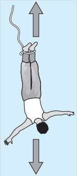




Newton-Raphson: most widely used of all root-finding methods

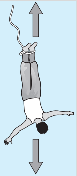
$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$





Newton-Raphson: Example

Use Newton-Raphson to estimate the root of $f(x) = \mathrm{e}^{-x} - x$, with the initial guess $x_0 = 0$.

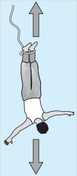


Newton-Raphson: Example

Use Newton-Raphson to estimate the root of $f(x) = \mathrm{e}^{-x} - x$, with the initial guess $x_0 = 0$.

First derivative $f'(x) = -\mathrm{e}^{-x} - 1$, giving

$$x_{i+1} = x_i - \frac{\mathrm{e}^{-x_i} - x_i}{-\mathrm{e}^{-x_i} - 1}.$$



Newton-Raphson: Example

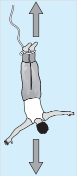
Use Newton-Raphson to estimate the root of $f(x) = \mathrm{e}^{-x} - x$, with the initial guess $x_0 = 0$.

First derivative $f'(x) = -\mathrm{e}^{-x} - 1$, giving

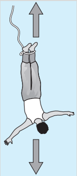
$$x_{i+1} = x_i - \frac{\mathrm{e}^{-x_i} - x_i}{-\mathrm{e}^{-x_i} - 1}.$$

i	x_i	$\lvert\varepsilon_t\rvert$, %
0	0	100
1	0.500000000	11.8
2	0.566311003	0.147
3	0.567143165	0.0000220
4	0.567143290	$<10^{-8}$

Remark: the exact root of f is given via the product logarithm (a special function) and is 0.56714.



Newton-Raphson: Convergence rate



Rate of convergence:

$$E_{t,i+1} = \frac{-f''(x_r)}{2f'(x_r)} E_{t,i}^2$$

The error is roughly proportional to the square of the previous error.

QUADRATIC CONVERGENCE $\rightarrow$ one of the main reasons for the popularity of the method

A slowly converging Newton-Raphson algorithm

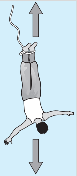
Determine the positive root of

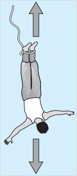
$$f(x) = x^{10} - 1$$

using Newton-Raphson and an initial guess of $x = 0.5$.
Iteration

$$x_{i+1} = x_i - \frac{x_i^{10} - 1}{10x_i^9}$$

i	x_i	$\lvert\varepsilon_a\rvert$, %
0	0.5	
1	51.65	99.032
2	46.485	11.111
3	41.8365	11.111
4	37.65285	11.111
⋮		
40	1.002316	2.130
41	1.000024	0.229
42	1	0.002





A slowly converging Newton-Raphson algorithm

Determine the positive root of

$$f(x) = x^{10} - 1$$

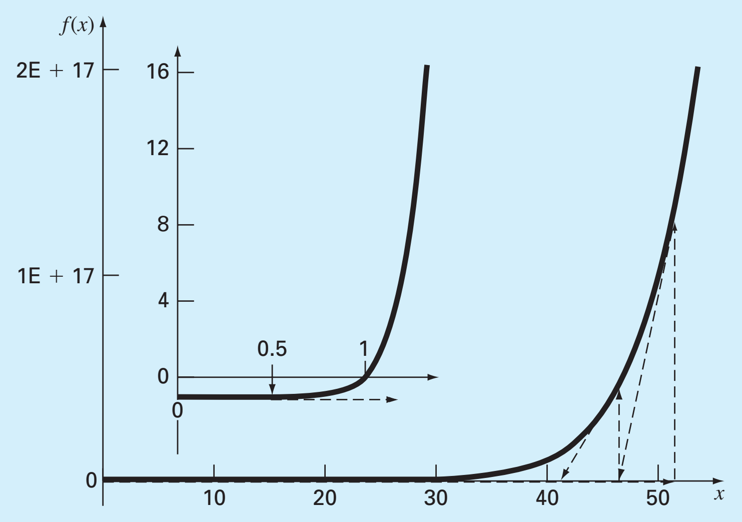
using Newton-Raphson and an initial guess of $x = 0.5$.
Iteration

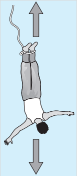
$$x_{i+1} = x_i - \frac{x_i^{10} - 1}{10x_i^9}$$

i	x_i	$\lvert\varepsilon_a\rvert$, %
0	0.5	
1	51.65	99.032
2	46.485	11.111
3	41.8365	11.111
4	37.65285	11.111
⋮		
40	1.002316	2.130
41	1.000024	0.229
42	1	0.002

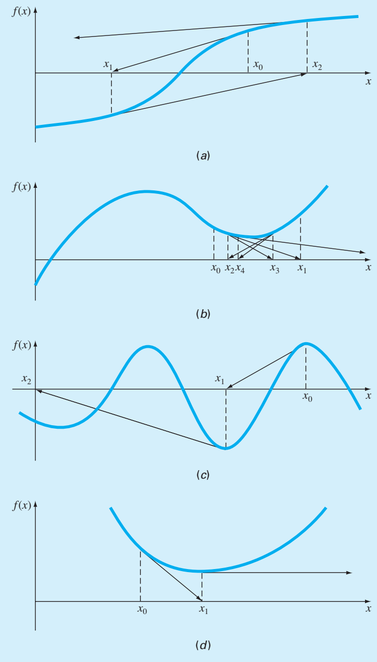
What happens?

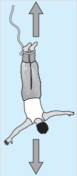
A slowly Newton-Raphson algorithm





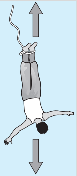
Newton-Raphson: weak spots





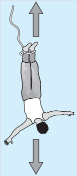
Secant method

Motivation: Difficult-to-evaluate derivative

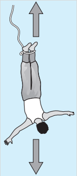


Secant method

Motivation: Difficult-to-evaluate derivative
Idea: Approximate derivative by backward finite difference



Secant method



Motivation: Difficult-to-evaluate derivative
Idea: Approximate derivative by backward finite difference

$$f'(x_i) \cong \frac{f(x_{i-1}) - f(x_i)}{x_{i-1} - x_i}$$

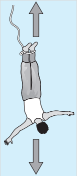
Secant method

Motivation: Difficult-to-evaluate derivative
Idea: Approximate derivative by backward finite difference

$$f'(x_i) \cong \frac{f(x_{i-1}) - f(x_i)}{x_{i-1} - x_i}$$

Secant method:

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$



Secant method

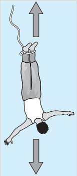
Motivation: Difficult-to-evaluate derivative
Idea: Approximate derivative by backward finite difference

$$f'(x_i) \cong \frac{f(x_{i-1}) - f(x_i)}{x_{i-1} - x_i}$$

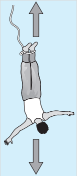
Secant method:

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$

Remark: We need 2 initial guesses for x!!



Modified secant method



Alternative approach: Instead of the second initial guess, use a fractional perturbation of x_i:

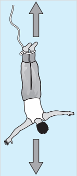
$$f'(x_i) \cong \frac{f(x_i + \delta x_i) - f(x_i)}{\delta x_i}.$$

Modified secant method:

$$x_{i+1} = x_i - \frac{\delta x_i f(x_i)}{f(x_i + \delta x_i) - f(x_i)}$$

Modified secant method - Bungee jumper problem

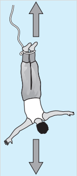
Determine mass of the bungee jumper that'll have a velocity of 36m/s in the 4th second of the free fall ($c_d = 0.25$kg/m) Use an initial guess of 50kg and a value of 10^{-6} for the perturbation fraction.



Modified secant method - Bungee jumper problem

Determine mass of the bungee jumper that'll have a velocity of 36m/s in the 4th second of the free fall ($c_d = 0.25$kg/m) Use an initial guess of 50kg and a value of 10^{-6} for the perturbation fraction.

i	x_i	$\lvert\varepsilon_t\rvert$, %	$\lvert\varepsilon_a\rvert$, %
0	50.0000	64.971	
1	88.3993	38.069	43.438
2	124.0897	13.064	28.762
3	140.5417	1.538	11.706
4	142.7072	0.021	1.517
5	142.7376	4.1×10^{-6}	0.021
6	142.7376	3.4×10^{-12}	4.1×10^{-6}



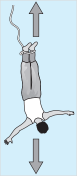
Modified secant method - Bungee jumper problem

Determine mass of the bungee jumper that'll have a velocity of 36m/s in the 4th second of the free fall ($c_d = 0.25$kg/m) Use an initial guess of 50kg and a value of 10^{-6} for the perturbation fraction.

i	x_i	$\|\varepsilon_t\|$, %	$\|\varepsilon_a\|$, %
0	50.0000	64.971	
1	88.3993	38.069	43.438
2	124.0897	13.064	28.762
3	140.5417	1.538	11.706
4	142.7072	0.021	1.517
5	142.7376	4.1×10^{-6}	0.021
6	142.7376	3.4×10^{-12}	4.1×10^{-6}

Care: choose δ

- not too small, avoiding subtractive cancelation
- not too large $\rightarrow$ inefficient and divergent algorithm



Modified secant method - Bungee jumper problem

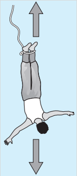
Determine mass of the bungee jumper that'll have a velocity of 36m/s in the 4th second of the free fall ($c_d = 0.25$kg/m) Use an initial guess of 50kg and a value of 10^{-6} for the perturbation fraction.

i	x_i	$\lvert\varepsilon_t\rvert$, %	$\lvert\varepsilon_a\rvert$, %
0	50.0000	64.971	
1	88.3993	38.069	43.438
2	124.0897	13.064	28.762
3	140.5417	1.538	11.706
4	142.7072	0.021	1.517
5	142.7376	4.1×10^{-6}	0.021
6	142.7376	3.4×10^{-12}	4.1×10^{-6}

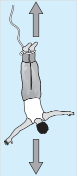
Care: choose δ

- not too small, avoiding subtractive cancelation
- not too large $\rightarrow$ inefficient and divergent algorithm

Particular use of secant methods: for complicated functions (ex. consisting of many lines of code)



Brent's method - generalities



Catch: it is a hybrid method combining the reliability of *bracketing methods* with the speed of *open methods*:

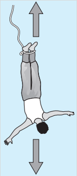
Developed by Richard Brent (1973) based on an earlier algorithm of Theodorus Dekker (1969).

Brent's method - generalities

Catch: it is a hybrid method combining the reliability of *bracketing methods* with the speed of *open methods*:

- bracketing method: bisection

Developed by Richard Brent (1973) based on an earlier algorithm of Theodorus Dekker (1969).

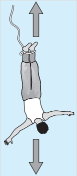


Brent's method - generalities

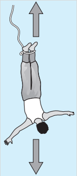
Catch: it is a hybrid method combining the reliability of *bracketing methods* with the speed of *open methods*:

- bracketing method: bisection
- open methods: secant method and quadratic interpolation

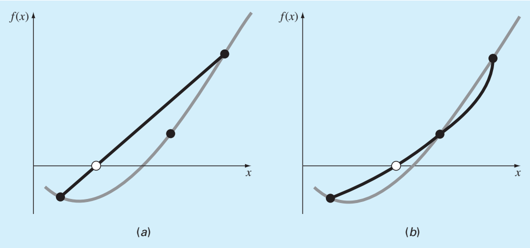
Developed by Richard Brent (1973) based on an earlier algorithm of Theodorus Dekker (1969).



Inverse quadratic interpolation



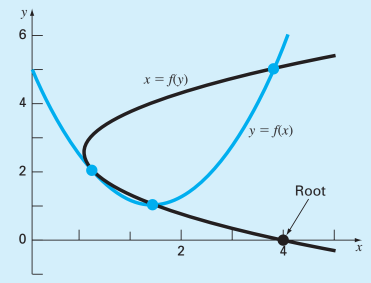
Idea:

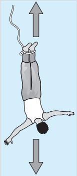


Secant (a) is in fact a linear interpolation method. *Inverse quadratic interpolation* (b)

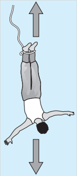
Inverse quadratic interpolation

Potential difficulties for quadratic interpolation: parabola does not intersect x-axis.

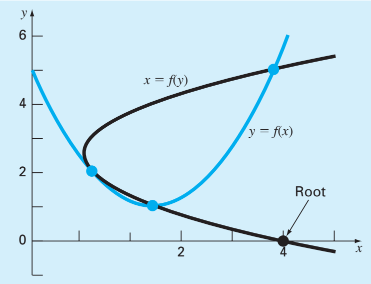




Inverse quadratic interpolation

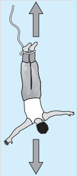


Potential difficulties for quadratic interpolation: parabola does not intersect x-axis.



Solution: inverse quadratic interpolation, i.e. "sideways" parabola $x = f(y)$. It always intersects the x-axis!

Inverse quadratic interpolation method



Given 3 points (x_{i-2}, y_{i-2}), (x_{i-1}, y_{i-1}), (x_i, y_i), a quadratic function of y that passes through them

$$g(y) = \frac{(y - y_{i-1})(y - y_i)}{(y_{i-2} - y_{i-1})(y_{i-2} - y_i)} x_{i-2} + \frac{(y - y_{i-2})(y - y_i)}{(y_{i-1} - y_{i-2})(y_{i-1} - y_i)} x_{i-1} + \frac{(y - y_{i-2})(y - y_{i-1})}{(y_i - y_{i-2})(y_i - y_{i-1})} x_i$$

This is a *Lagrange polynomial*.
The root corresponds to $y = 0$ above:

$$x_{i+1} = \frac{(y_{i-1})(y_i)}{(y_{i-2} - y_{i-1})(y_{i-2} - y_i)} x_{i-2} + \frac{(y_{i-2})(y_i)}{(y_{i-1} - y_{i-2})(y_{i-1} - y_i)} x_{i-1} + \frac{(y_{i-2})(y_{i-1})}{(y_i - y_{i-2})(y_i - y_{i-1})} x_i$$

Inverse quadratic interpolation method

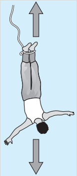
Given 3 points (x_{i-2}, y_{i-2}), (x_{i-1}, y_{i-1}), (x_i, y_i), a quadratic function of y that passes through them

$$g(y) = \frac{(y - y_{i-1})(y - y_i)}{(y_{i-2} - y_{i-1})(y_{i-2} - y_i)} x_{i-2} + \frac{(y - y_{i-2})(y - y_i)}{(y_{i-1} - y_{i-2})(y_{i-1} - y_i)} x_{i-1} + \frac{(y - y_{i-2})(y - y_{i-1})}{(y_i - y_{i-2})(y_i - y_{i-1})} x_i$$

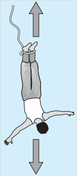
This is a *Lagrange polynomial*.
The root corresponds to $y = 0$ above:

$$x_{i+1} = \frac{(y_{i-1})(y_i)}{(y_{i-2} - y_{i-1})(y_{i-2} - y_i)} x_{i-2} + \frac{(y_{i-2})(y_i)}{(y_{i-1} - y_{i-2})(y_{i-1} - y_i)} x_{i-1} + \frac{(y_{i-2})(y_{i-1})}{(y_i - y_{i-2})(y_i - y_{i-1})} x_i$$

It does not work if y_{i-2}, y_{i-1}, y_i are not distinct; in this case we use the less efficient secant method.



Brent's method algorithm



General idea: whenever possible, use a fast *open method*; otherwise fall back on the conservative *bisection*.
Repeat until location tolerance is acceptable.
Remark: Bisection dominates in the beginning, but as the root is approached, the method shifts to open methods.