

Engineering Analysis ENG 3420 Fall 2009

Dan C. Marinescu

Office: HEC 439 B

Office hours: Tu-Th 11:00-12:00

Lecture 9

- Last time:
 - Approximations
 - Finding the roots of the equation $f(x)=0$
 - Structured programming
 - File creation and file access
 - Relational operators
 - Project1
- Today:
 - Bracketing vs Open Methods for finding the roots of the equation $f(x) = 0$;
 - Convergence vs Divergence
 - Open methods for finding the roots of the equation $f(x) = 0$
 - Simple Fixed-Point Iteration
 - Newton-Raphson
 - Secant
- Next Time
 - Optimization

Numerical methods

- **Direct methods** → attempt to solve a numerical problem by a finite sequence of operations. In absence of round off errors deliver an exact solution; e.g., solving a linear system $Ax = b$ by *Gaussian* elimination.
- **Iterative methods** → attempt to solve a numerical problem (for example, finding the root of an equation or system of equations) by finding successive approximations to the solution starting from an initial guess.

- The stopping criteria: the relative error

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

is smaller than a pre-specified value.

- When used

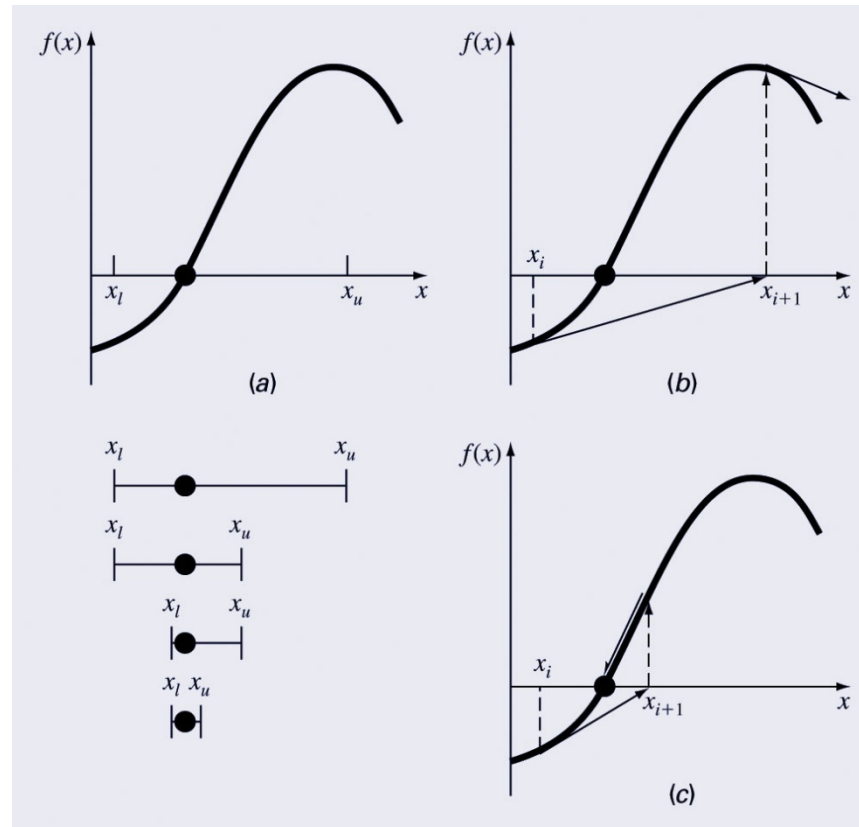
- The only alternative for non-linear systems of equations;
- Often useful even for linear problems involving a large number of variables where direct methods would be prohibitively expensive or impossible.

- **Convergence of a numerical methods** → if successive approximations lead to increasingly smaller relative error. Opposite to divergent.

Iterative methods for finding the roots

- *Bracketing methods*
- *Open methods:*
 - require only a single starting value or two starting values that do not necessarily bracket a root.
 - may diverge as the computation progresses, but when they do converge, they usually do so much faster than bracketing methods.

Bracketing vs Open; Convergence vs Divergence



a) Bracketing method \rightarrow start with an interval.

Open method \rightarrow start with a single initial guess

b) Diverging open method

c) Converging open method - note speed!

Simple Fixed-Point Iteration

- Rearrange the function $f(x)=0$ so that x is on the left-hand side of the equation:

$$x=g(x)$$

- Use the new function g to predict a new value of x . The recursion equation:

$$x_{i+1}=g(x_i)$$

- The approximate error is:

$$\mathcal{E}_a = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right| 100\%$$

- Graphically, the root is at the intersection of two curves:

$$y_1(x) = g(x)$$

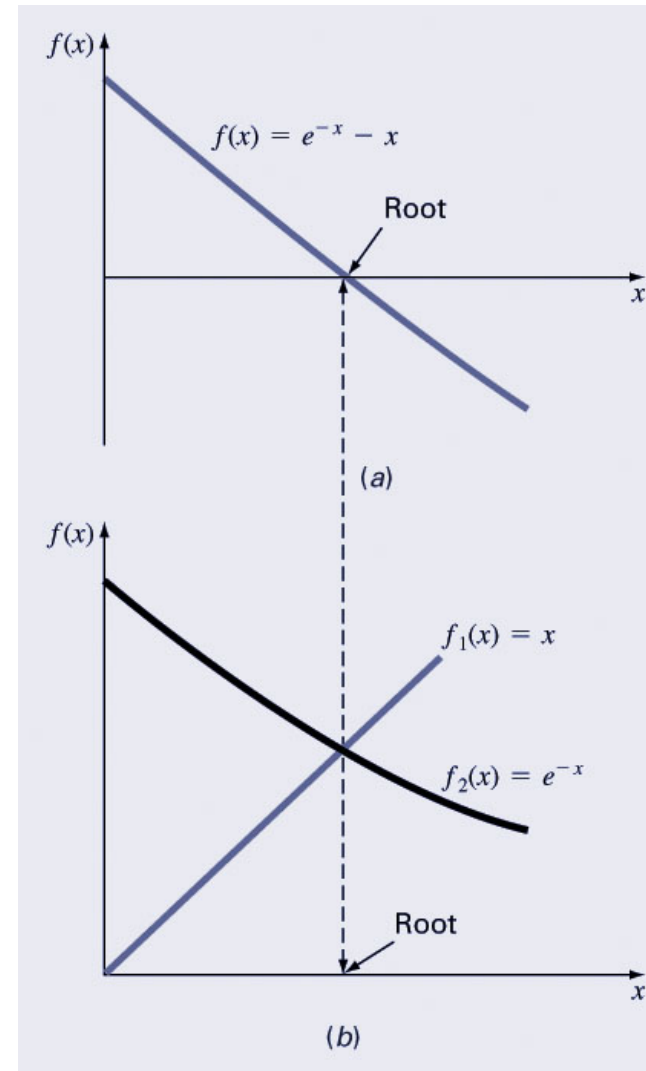
$$y_2(x) = x.$$

Example

- Solve $f(x)=e^{-x}-x$
- Re-write as: $x=g(x) \rightarrow x=e^{-x}$
- Start with an initial guess (here, 0)

i	x_i	$ \varepsilon_a \%$	$ \varepsilon_t \%$	$ \varepsilon_t / \varepsilon_{t-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

- Continue until some tolerance is reached

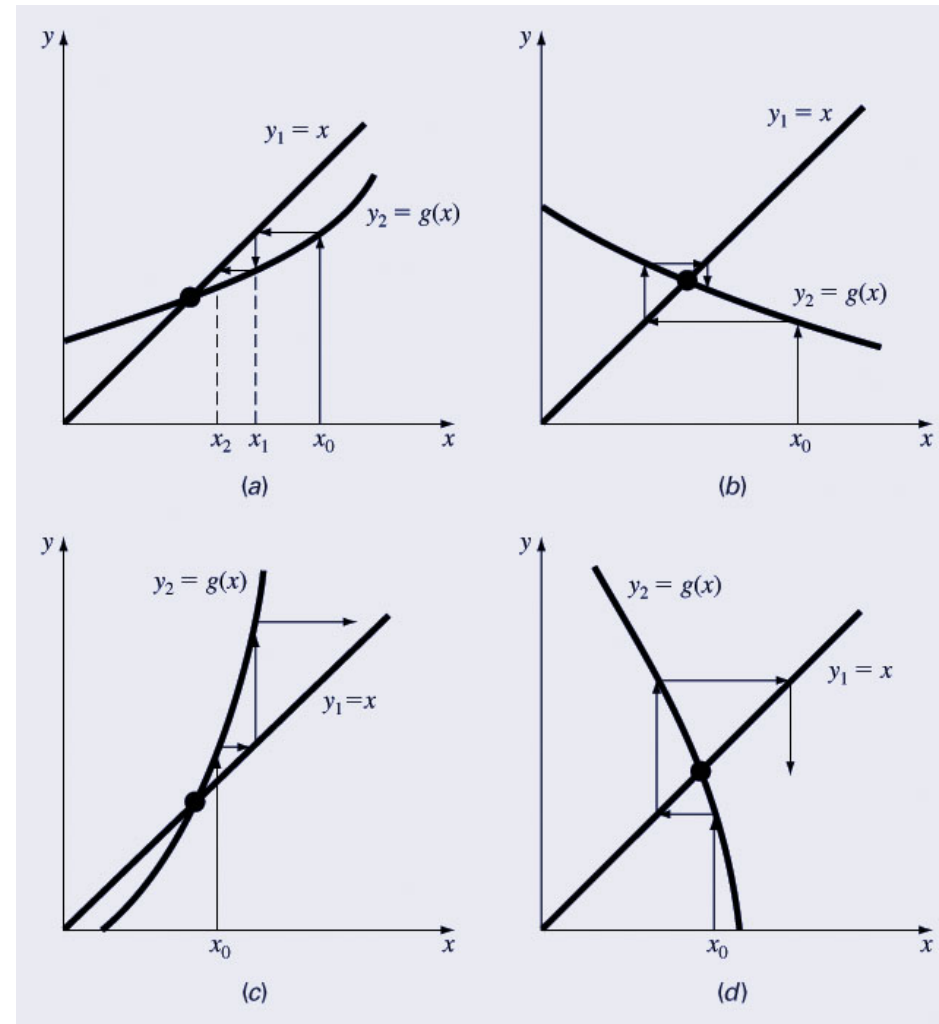


More on Convergence

Graphically → the solution is at the intersection of the two curves. We identify the point on y_2 corresponding to the initial guess and the next guess corresponds to the value of the argument x where $y_1(x) = y_2(x)$.

Convergence of the simple fixed-point iteration method requires that the derivative of $g(x)$ near the root has a magnitude less than 1.

- a) Convergent, $0 \leq g' < 1$
- b) Convergent, $-1 < g' \leq 0$
- c) Divergent, $g' > 1$
- d) Divergent, $g' < -1$



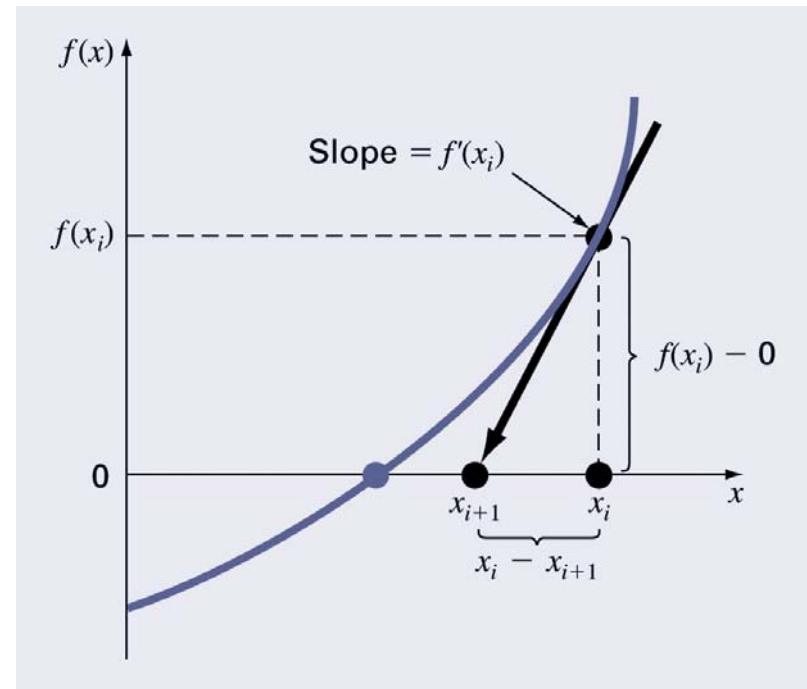
Newton-Raphson Method

- Express x_{i+1} function of x_i and the values of the function and its derivative at x_i .

$$f'(x_i) = \frac{f(x_i) - 0}{x_i - x_{i+1}}$$

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

- Graphically \rightarrow draw the tangent line to the $f(x)$ curve at some guess x , then follow the tangent line to where it crosses the x -axis.



```
function[root,relative_error,number_iterations]=newton_raphson(myfunction,derivative,  
initial_guess,desired_relative_error, max_number_iterations,varargin)
```

```
if(nargin<3) error('at least 3 input arguments, required'); end  
if (nargin<4) is_empty(desired_realtive_error), desired_realtive_error=0.0001; end  
    % set desired_realtive_error to the default, 0.0001, if none  
specified  
if (nargin<5) is_empty(max_number_iterations),max_number_iterations=50; end  
    % set max_number_iterations to the default, 50, if none  
specified  
number_iterations=0;  
current_guess = initial_guess;  
while(1)  
    next_guess = current_guess;  
    current_guess = current_guess - (myfunction(current_guess)/derivative(current_guess));  
    number_iterations=number_iterations+1;  
    %fprintf(' %6.0f number_iterations', number_iterations);  
    if (current_guess ~=0.0)  
        relative_error = abs((current_guess-next_guess)/current_guess)*100;  
    end  
    if (relative_error <=desired_relative_error) | (number_iterations >= max_number_iterations),  
        break,  
    end  
    fprintf('iteration= %2.0f guess= %8.5f relative_error=%8.5f \n', number_iterations,  
        current_guess, relative_error);  
end  
root = current_guess;
```

Example –fast converging case $f(x)=\exp(-x) - x$

```
>> fun=@(x) exp(-x)-x; der=@(x) -exp(-x)-1;  
[root,rel_error,niter]=newton_raphson(fun,der,0.0,0.000001,500); root,  
rel_error,niter
```

```
iteration= 1 guess= 0.50000 relative_error=100.00000
```

```
iteration= 2 guess= 0.56631 relative_error=11.70929
```

```
iteration= 3 guess= 0.56714 relative_error= 0.14673
```

```
iteration= 4 guess= 0.56714 relative_error= 0.00002
```

```
root = 0.5671
```

```
rel_error =5.0897e-013
```

```
niter = 5
```

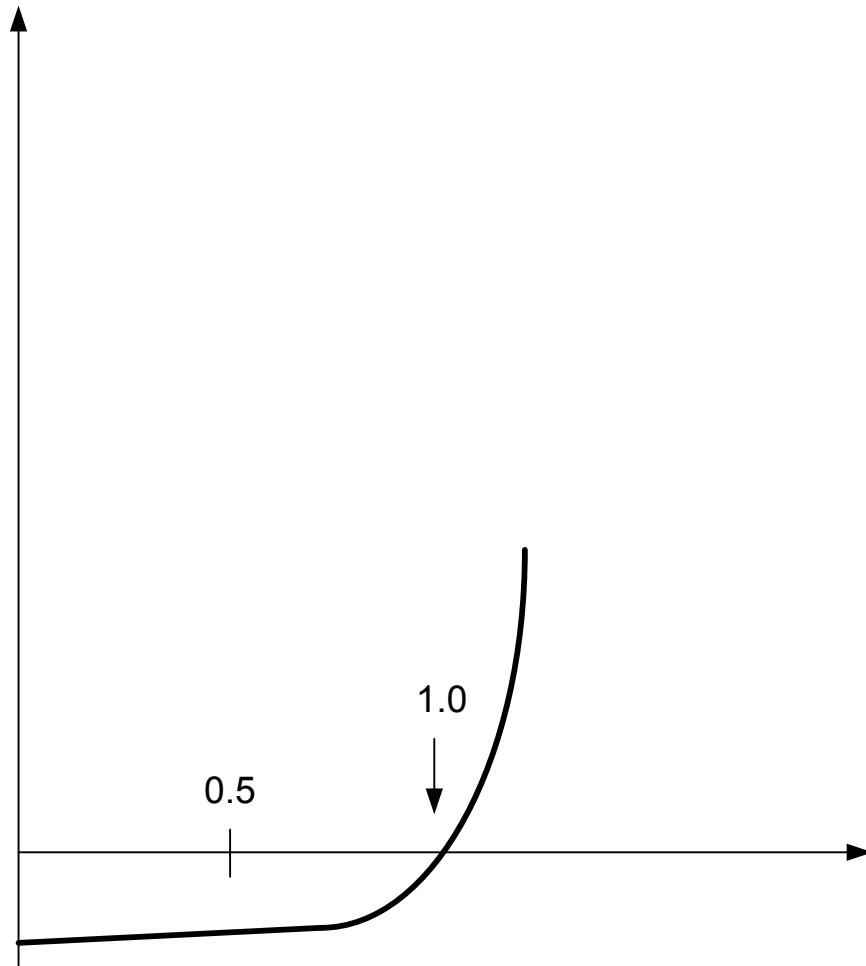
Slow converging case: $f(x)=x^{10}-1$ $\text{init_guess}=0.5$

```
>> fun=@(x) x^(10)-1; der=@(x) 10*x^(9); init_guess=0.5;  
[root,rel_error,niter]=newton_raphson(fun,der,init_guess,0.000001,500); root,  
rel_error,niter
```

```
iteration= 1 guess= 51.65000 relative_error=99.03195  
iteration= 2 guess= 46.48500 relative_error=11.11111  
iteration= 3 guess= 41.83650 relative_error=11.11111  
iteration= 4 guess= 37.65285 relative_error=11.11111  
iteration= 5 guess= 33.88757 relative_error=11.11111  
iteration= 6 guess= 30.49881 relative_error=11.11111  
iteration= 7 guess= 27.44893 relative_error=11.11111  
iteration= 8 guess= 24.70403 relative_error=11.11111  
iteration= 9 guess= 22.23363 relative_error=11.11111  
iteration= 10 guess= 20.01027 relative_error=11.11111  
iteration= 11 guess= 18.00924 relative_error=11.11111  
iteration= 12 guess= 16.20832 relative_error=11.11111  
iteration= 13 guess= 14.58749 relative_error=11.11111  
iteration= 14 guess= 13.12874 relative_error=11.11111  
iteration= 15 guess= 11.81586 relative_error=11.11111  
iteration= 16 guess= 10.63428 relative_error=11.11111  
iteration= 17 guess= 9.57085 relative_error=11.11111  
iteration= 18 guess= 8.61376 relative_error=11.11111  
iteration= 19 guess= 7.75239 relative_error=11.11111
```

```
iteration= 20 guess= 6.97715 relative_error=11.11111
iteration= 21 guess= 6.27943 relative_error=11.11111
iteration= 22 guess= 5.65149 relative_error=11.11111
iteration= 23 guess= 5.08634 relative_error=11.11111
iteration= 24 guess= 4.57771 relative_error=11.11111
iteration= 25 guess= 4.11994 relative_error=11.11111
iteration= 26 guess= 3.70794 relative_error=11.11110
iteration= 27 guess= 3.33715 relative_error=11.11109
iteration= 28 guess= 3.00344 relative_error=11.11104
iteration= 29 guess= 2.70310 relative_error=11.11090
iteration= 30 guess= 2.43280 relative_error=11.11052
iteration= 31 guess= 2.18955 relative_error=11.10941
iteration= 32 guess= 1.97069 relative_error=11.10624
iteration= 33 guess= 1.77384 relative_error=11.09714
iteration= 34 guess= 1.59703 relative_error=11.07110
iteration= 35 guess= 1.43881 relative_error=10.99684
iteration= 36 guess= 1.29871 relative_error=10.78735
iteration= 37 guess= 1.17835 relative_error=10.21396
iteration= 38 guess= 1.08335 relative_error= 8.76956
iteration= 39 guess= 1.02366 relative_error= 5.83053
iteration= 40 guess= 1.00232 relative_error= 2.12993
iteration= 41 guess= 1.00002 relative_error= 0.22920
iteration= 42 guess= 1.00000 relative_error= 0.00239
```

```
root = 1
rel_error = 2.5776e-007
niter = 43
```



The same function but a better initial guess

```
fun=@(x) x^(10)-1; der=@(x) 10*x^(9); init_guess=1.1;  
[root,rel_error,niter]=newton_raphson(fun,der,init_guess,0.000001,5  
00); root, rel_error,niter
```

```
iteration= 1 guess= 1.03241 relative_error= 6.54684  
iteration= 2 guess= 1.00422 relative_error= 2.80760  
iteration= 3 guess= 1.00008 relative_error= 0.41363  
iteration= 4 guess= 1.00000 relative_error= 0.00787  
iteration= 5 guess= 1.00000 relative_error= 0.00000
```

```
root = 1  
rel_error = 3.5527e-013  
niter = 6
```