Roots of Equations

Mark A. Austin

University of Maryland

austin@umd.edu

ENCE 201, Fall Semester 2023

September 30, 2023

Overview

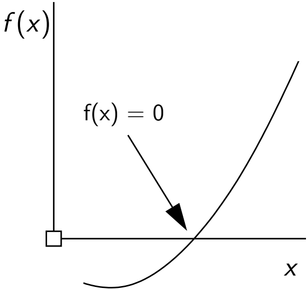
1. Numerical Solution of Equations
2. Iterative Methods
3. Method of Bisection
   - Numerical Procedure, Examples
4. Newton Raphson Iteration
5. Modified Newton Raphson Iteration
6. Python Code Listings
   - Method of Bisection
   - Newton Raphson Algorithm
   - Modified Newton Raphson

Part 2

Numerical Solution of Equations

Numerical Solution of Equations

Math Problem. Given f(x), find a value of x such that f(x) = g(x), f(x) = constant, or f(x) = 0.



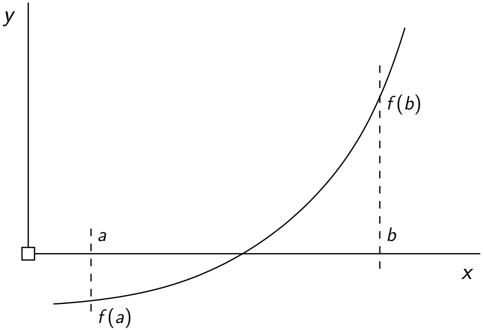
All forms may be put in the format F(x) = 0.

Problem Solving

Strategies

Problem Solving Strategies

Bracketing Methods: Requires two initial guesses that bracket the solution.



- Various algorithms for computing estimates to $f(x) = 0$, e.g, Bisection, Secant stiffness.

Method of Bisection

Method of Bisection

A reliable method for solving $f(x) = 0$.

Fact. Suppose we have continuous function $f(x)$. If $f(a) < 0$ and $f(b) > 0$ then there exists a point c in $[a, b]$ such that $f(c) = 0$.

Numerical Procedure. Find initial points a and b such that $f(a)$ and $f(b)$ have opposite signs. Let $x_{left} = a$ and $x_{right} = b$.

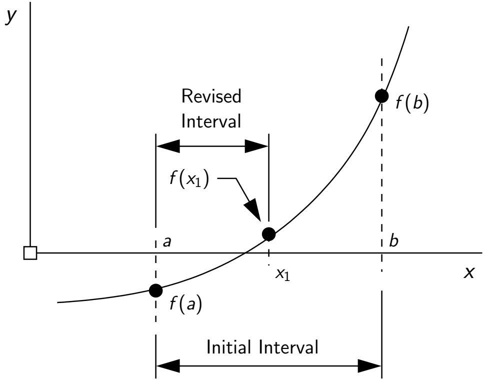
- Evaluate at mid-point: $x_{new} = \frac{1}{2}[x_{left} + x_{right}]$.
- Look for change in sign in function evaluation.

  Keep $f(x_{left})$ if $f(x_{new}).f(x_{left}) < 0$.

  Otherwise, keep $f(x_{right})$ if $f(x_{new}).f(x_{right}) < 0$.
- Repeat until solution converges.

Method of Bisection

Schematic: One iteration of Bisection:



For iteration 2, we set $x_{left} = f(a)$ and $x_{right} = f(x_1)$.

Method of Bisection

Example 1. Demonstrate use of bisection method to compute roots of the quadratic.

$$f(x) = (x-3)*(x-3) - 2 = 0; \tag{3}$$

Analytic Solution: From equation 3:

$$(x-3)^2 = 2 \Longrightarrow [x_1, x_2] = \left[3 - \sqrt{2}, 3 + \sqrt{2}\right]. \tag{4}$$

Source Code:

- TestBisection01.py: Test program and functions for bisection algorithm ...
- Solutions.py: Python code for bisection algorithm.

Method of Bisection

Test Program Source Code:

```
# ==========================================================================
# TestBisection01.py: Use bisection algorithm to compute roots of equations.
#
# Written By: Mark Austin                                      February 2023
# ==========================================================================

import math;
import Solutions;

# Define mathematical functions ...

def f1(x):
    return (x-3)*(x-3)-2;

# main method ...

def main():
    print("--- Enter TestBisection01.main()                    ... ");
    print("--- ========================================= ... ");

    print("--- ");
    print("--- Case Study 1: Solve (x-3)*(x-3)-2 = 0 ... ");
    print("--- ======================================= ... ");

    # Initialize problem setup ...
```

Method of Bisection

Test Program Source Code: Continued ...

```
    a =  -1.0;
    b =   2.0
    tolerance     = 0.01
    maxiterations = 100

    print("--- Inputs:")
    print("---   a = {:5.2f} ...".format(a) )
    print("---   b = {:5.2f} ...".format(b) )
    print("---   tolerance      = {:8.5f} ...".format(tolerance) )
    print("---   max iterations = {:8.2f} ...".format(maxiterations) )

    # Compute roots to equation ...

    print("--- Execution:")
    root, i, converged = Solutions.bisection(f1, a, b, tolerance, maxiterations )

    # Summary of computations ...

    print("--- Output:")
    print("---   root = {:10.5f} ...".format(root) )
    print("---   f(root) --> {:12.5e} ...".format( f1(root)) )
    print("---   no iterations = {:d} ...".format(i) )
    print("---   converged: {:s} ...".format( str(converged) ) )

    print("--- ");
    print("--- Case Study 2: Solve 2x^3 - cos(x+1) - 3 = 0 ... ");
    print("--- ========================================== ... ");
```

Method of Bisection

Test Program Source Code: Continued ...

```

    # Initialize problem setup ...

    a = -1.0;
    b =  2.0
    tolerance     = 0.01
    maxiterations = 100

    print("--- Inputs:")
    print("---   a = {:5.2f} ...".format(a) )
    print("---   b = {:5.2f} ...".format(b) )
    print("---   tolerance      = {:8.5f} ...".format(tolerance) )
```

Abbreviated Output:

```
--- Case Study 1: Solve (x-3)*(x-3)-2 = 0 ...
--- ===================================== ...
--- Inputs:
---   a = -1.00 ...
---   b =  2.00 ...
---   tolerance      =  0.01000 ...
---   max iterations =   100.00 ...
```

Method of Bisection

Abbreviated Output: Continued ...

```
--- Execution:
---   Initial Conditions:
---   f(a) -->  1.40000e+01 ...
---   f(b) --> -1.00000e+00 ...
---   Main Loop for Root Computation:
---   Iteration 00: dx = 1.50000e+00, x =  5.00000e-01, f(x) ->  4.25000e+00
---   Iteration 01: dx = 7.50000e-01, x =  1.25000e+00, f(x) ->  1.06250e+00
---   Iteration 02: dx = 3.75000e-01, x =  1.62500e+00, f(x) -> -1.09375e-01
---   Iteration 03: dx = 1.87500e-01, x =  1.43750e+00, f(x) ->  4.41406e-01
---   Iteration 04: dx = 9.37500e-02, x =  1.53125e+00, f(x) ->  1.57227e-01
---   Iteration 05: dx = 4.68750e-02, x =  1.57812e+00, f(x) ->  2.17285e-02
---   Iteration 06: dx = 2.34375e-02, x =  1.60156e+00, f(x) -> -4.43726e-02
---   Iteration 07: dx = 1.17188e-02, x =  1.58984e+00, f(x) -> -1.14594e-02
---   Iteration 08: dx = 5.85938e-03, x =  1.58398e+00, f(x) ->  5.10025e-03
--- Output:
---   root =    1.58398 ...
---   f(root) -->  5.10025e-03 ...
---   no iterations = 8 ...
---   converged: True ...
```

Method of Bisection

Example 2. The test function

$$f(x) = \left[\frac{(x^{20} + 1)x(x-2)}{1000} \right] \tag{5}$$

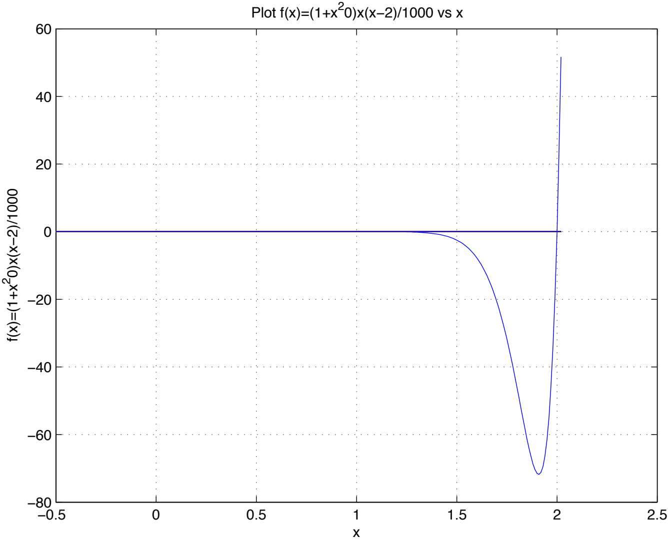
has two roots within the interval $[-1, 3]$.

From a numerical standpoint, this problem is challenging:

- In the neighborhood of $x = 0$, the test function values and slope are very close to zero.
- In the neighborhood of $x = 2$, the test function slope is extremely high.

We can break the solution into blocks:

Method of Bisection



Method of Bisection

Test Program Source Code:

```
# ==========================================================================
# TestBisection02.py: Use bisection algorithm to compute roots of equations:
#
# Written By: Mark Austin                                      February 2023
# ==========================================================================

import math;
import Solutions;

# Define mathematical functions ...

def f1(x):
    return (x**20 + 1)*x*(x-2)/1000.0;

# main method ...

def main():
    print("--- ");
    print("--- Case Study 1: Solve f(x) = ((x^20 + 1)x(x-2))/1000 = 0 ... ");
    print("--- ====================================================== ... ");

    # Initialize problem setup ...

    a =  0.5;
    b =  2.5
    tolerance     = 0.0001
```

Method of Bisection

Test Program Source Code: Continued ...

```
    maxiterations = 100

    print("--- Inputs:")
    print("---   a = {:5.2f} ...".format(a) )
    print("---   b = {:5.2f} ...".format(b) )
    print("---   tolerance      = {:8.5f} ...".format(tolerance) )
    print("---   max iterations = {:8.2f} ...".format(maxiterations) )

    # Compute roots to equation ...

    print("--- Execution:")
    root, i, converged = Solutions.bisection(f1, a, b, tolerance, maxiterations )

    # Summary of computations ...

    print("--- Output:")
    print("---   root = {:12.7f} ...".format(root) )
    print("---   f(root) --> {:14.7e} ...".format( f1(root)) )
    print("---   no iterations = {:d} ...".format(i) )
    print("---   converged: {:s} ...".format( str(converged) ) )

# call the main method ...

main()
```

Method of Bisection

Abbreviated Output: Solve f(x) = $((x^{20} + 1)$x(x-2))/1000 = 0

```
--- Inputs:
---   a =  0.50 ...
---   b =  2.50 ...
---   tolerance      =  0.00010 ...
---   max iterations =   100.00 ...
--- Execution:
---   Initial Conditions:
---   f(a) --> -7.50001e-04 ...
---   f(b) -->  1.13687e+05 ...
---   Main Loop for Root Computation:
---   Iteration 00: dx = 1.0000e+00, x = 1.500000e+00, f(x) -> -2.494692e+00
---   Iteration 01: dx = 5.0000e-01, x = 2.000000e+00, f(x) ->  0.000000e+00
...
---   Iteration 24: dx = 5.9605e-08, x = 1.999999e+00, f(x) -> -1.250000e-04
---   Iteration 25: dx = 2.9802e-08, x = 2.000000e+00, f(x) -> -6.250004e-05
--- Output:
---   root =    2.0000000 ...
---   f(root) --> -6.2500040e-05 ...
---   no iterations = 25 ...
---   converged: True ...
```

Method of Bisection

Summary

- A reliable method for solving $f(x) = 0$.

Limitations

- Need to find two bracketing points before iteration can begin.
- Convergence can be slow.

Python Code Listings

Code 1: Method of Bisection

```
# ==========================================================================
# Solutions.bisection(): Compute Roots of an equation by the Bisection method.
#
# Args: f (function): equation f(x).
#       a (float): lower limit.
#       b (float): upper limit.
#       toler (float): tolerance (stopping criterion).
#       iter_max (int): maximum number of iterations (stopping criterion).
#
# Returns:
#       root (float): root value.
#       iter (int): number of iterations used by the method.
#       converged (boolean): flag to indicate if the root was found.
# ==========================================================================

import math

def bisection(f, a, b, toler, iter_max):

    fa = f(a)
    fb = f(b)

    # Check that the function changes sign ....

    print("---   Initial Conditions: ")
    print("---   f(a) --> {:12.5e} ...".format( f(a) ) );
    print("---   f(b) --> {:12.5e} ...".format( f(b) ) );
```

Code 1: Method of Bisection

```
    if fa * fb > 0:
        raise ValueError("--- The function does not change signal at \
              the ends of the given interval.")

    delta_x = math.fabs(b - a) / 2

    # Main loop for bisection iteration ..

    print("---   Main Loop for Root Computation: ")

    x = 0
    converged = False
    for i in range(0, iter_max + 1):
        x  = (a + b) / 2
        fx = f(x)

        print("---   Iteration {:03d}: dx = {:10.5e}, x = {:14.7e}, f(x) --> {:14.7e} ..

        if delta_x <= toler and math.fabs(fx) <= toler:
            converged = True
            break

        if fa * fx > 0:
            a = x
            fa = fx
        else:
            b = x

        delta_x = delta_x / 2
```