

СРСП N°3

Численные методы решения нелинейных уравнений

Бактыбеков Нурсултан

2 Октября, 2023

Содержание

Цель работы	3
Задание	3
Выполнение заданий	3
График функции уравнения	3
Метод деления пополам	4
Решение	4
Метод Ньютона	5
Решение	5
Метод простой итерации	5
Решение	5
Метод хорд	6
Решение	6
Метод метод секущих	7
Решение	7
Метод Гаусса	8
Решение	8
Метод Крамера	9
Решение	9
Запуск всех методов	10
Код	10
Результат запуска	12

Цель работы

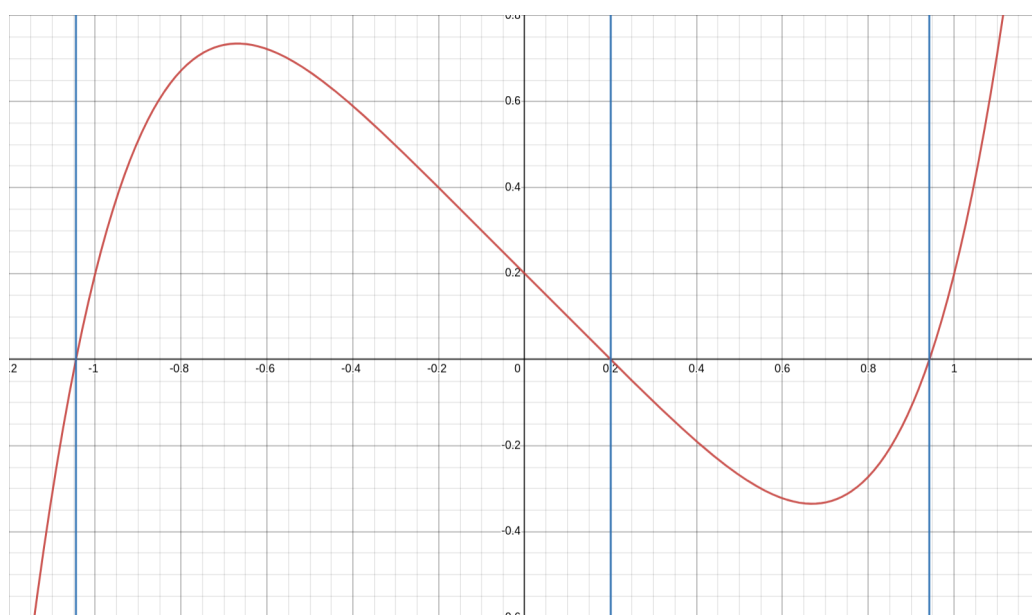
Научиться выполнять вычисления с приближенными числами с учетом погрешностей.

Задание

- Усвоить понятия абсолютной и относительной погрешности и их границ;
- Научиться определять верные, сомнительные, значащие цифры в приближенном числе и определять по ним погрешности приближений;
- Научиться находить погрешности вычислений;
- Вычислить корни уравнения $x^5 - x + 0.2 = 0$

Выполнение заданий

График функции уравнения



Метод деления пополам

Решение

```
1 package methods
2
3 import (
4     "fmt"
5     "math"
6 )
7
8 func HalfMethod(f func(float64) float64, a, b, tol float64) float64 {
9     if f(a)*f(b) >= 0 {
10         fmt.Println("Cannot use bisection method on this interval.")
11         return math.NaN()
12     }
13
14     iterations := 0
15     var c float64
16
17     fmt.Printf("%-12s %-12s %-12s %-12s\n", "Iteration", "a", "b", "Tolerance")
18
19     for {
20         c = (a + b) / 2
21         iterations++
22         fmt.Printf("%-12d %-12.6f %-12.6f %-12.6f\n", iterations, a, b, (b-a)/2)
23
24         if f(c) == 0 || (b-a)/2 < tol {
25             break
26         }
27
28         if f(c)*f(a) < 0 {
29             b = c
30         } else {
31             a = c
32         }
33     }
34
35     return c
36 }
```

Метод Ньютона

Решение

```
1 package methods
2
3 import (
4     "fmt"
5     "math"
6 )
7
8 func NewtonMethod(f func(float64) float64, df func(float64) float64, x0, tol
float64) float64 {
9     iterations := 0
10
11     fmt.Printf("%-12s %-12s %-12s\n", "Iteration", "x", "Tolerance")
12
13     for {
14         iterations++
15         x1 := x0 - f(x0)/df(x0)
16         tolerance := math.Abs(x1 - x0)
17
18         fmt.Printf("%-12d %-12.6f %-12.6f\n", iterations, x1, tolerance)
19
20         if tolerance < tol {
21             return x1
22         }
23
24         x0 = x1
25     }
26 }
```

Метод простой итерации

Решение

```
1 package methods
2
3 import (
4     "fmt"
5     "math"
6 )
7
8 func SimpleIterationMethod(f func(float64) float64, x0, tol float64,
maxIterations int) float64 {
9     iterations := 0
10
11     fmt.Printf("%-12s %-12s %-12s\n", "Iteration", "x", "Tolerance")
12
13     for iterations < maxIterations {
14         iterations++
```

```

15     x1 := f(x0)
16     tolerance := math.Abs(x1 - x0)
17
18     fmt.Printf("%-12d %-12.6f %-12.6f\n", iterations, x1, tolerance)
19
20     if tolerance < tol {
21         return x1
22     }
23
24     x0 = x1
25 }
26
27 fmt.Println("Maximum number of iterations reached.")
28 return x0
29 }

```

Метод хорд

Решение

```

1  package methods
2
3  import (
4      "fmt"
5      "math"
6  )
7
8  func ChordMethod(f func(float64) float64, x0, x1, tol float64, maxIterations
9  int) float64 {
10
11     iterations := 0
12
13     fmt.Printf("%-12s %-12s %-12s %-12s\n", "Iteration", "x0", "x1",
14     "Tolerance")
15
16     for iterations < maxIterations {
17         iterations++
18         fx0 := f(x0)
19         fx1 := f(x1)
20
21         x2 := x1 - (fx1 * (x1 - x0) / (fx1 - fx0))
22         tolerance := math.Abs(x2 - x1)
23
24         fmt.Printf("%-12d %-12.6f %-12.6f %-12.6f\n", iterations, x0, x1,
25         tolerance)
26
27         if tolerance < tol {
28             return x2
29         }
30
31         x0 = x1
32     }
33 }

```

```

30     x1 = x2
31 }
32
33 fmt.Println("Maximum number of iterations reached.")
    return x1
}

```

Метод метод секущих

Решение

```

1  package methods
2
3  import (
4      "fmt"
5      "math"
6  )
7
8  func SecantMethod(f func(float64) float64, x0, x1, tol float64, maxIter int)
   (float64, error) {
9      fmt.Printf("%-12s %-12s %-12s %-12s\n", "Iteration", "x", "f(x)",
10     "Tolerance")
11     for i := 0; i < maxIter; i++ {
12         fx0 := f(x0)
13         fx1 := f(x1)
14
15         if math.Abs(fx1) < tol {
16             fmt.Printf("\nConverged with tolerance: %e\n", tol)
17             return x1, nil
18         }
19
20         x2 := x1 - (fx1 * (x1 - x0) / (fx1 - fx0))
21
22         fmt.Printf("%-12d %-12.6f %-12.6f %-12.6f\n", i, x2, fx1, math.Abs(x2-
23     x1))
24
25         if math.Abs(x2-x1) < tol {
26             fmt.Printf("\nConverged with tolerance: %e\n", tol)
27             return x2, nil
28         }
29
30         x0 = x1
31         x1 = x2
32     }
33
34     return 0, fmt.Errorf("Maximum number of iterations reached")
35 }

```

Метод Гаусса

Решение

```
1 package methods
2
3 func GaussMethod(matrix [][]float64, constants []float64) []float64 {
4     n := len(matrix)
5     solution := make([]float64, n)
6
7     // Step 1: Forward Elimination
8     for i := 0; i < n; i++ {
9         // Find the row with the maximum element in the current column
10        maxRow := i
11        for j := i + 1; j < n; j++ {
12            if matrix[j][i] > matrix[maxRow][i] {
13                maxRow = j
14            }
15        }
16
17        // Swap the current row with the row containing the maximum element
18        matrix[i], matrix[maxRow] = matrix[maxRow], matrix[i]
19        constants[i], constants[maxRow] = constants[maxRow], constants[i]
20
21        // Make the diagonal element of the current row equal to 1
22        pivot := matrix[i][i]
23        for j := i; j < n; j++ {
24            matrix[i][j] /= pivot
25        }
26        constants[i] /= pivot
27
28        // Eliminate all other elements in the current column
29        for j := i + 1; j < n; j++ {
30            factor := matrix[j][i]
31            for k := i; k < n; k++ {
32                matrix[j][k] -= factor * matrix[i][k]
33            }
34            constants[j] -= factor * constants[i]
35        }
36    }
37
38    // Step 2: Back Substitution
39    for i := n - 1; i >= 0; i-- {
40        // Solve for the variable at the current row
41        solution[i] = constants[i]
42        for j := i + 1; j < n; j++ {
43            solution[i] -= matrix[i][j] * solution[j]
44        }
45    }
46
47    return solution
48 }
```



```
48     return solution
    }
```

Метод Крамера

Решение

```
1  package methods
2
3  import (
4      "fmt"
5      "math"
6  )
7
8  func HalfMethod(f func(float64) float64, a, b, tol float64) float64 {
9      if f(a)*f(b) >= 0 {
10         fmt.Println("Cannot use bisection method on this interval.")
11         return math.NaN()
12     }
13
14     iterations := 0
15     var c float64
16
17     fmt.Printf("%-12s %-12s %-12s %-12s\n", "Iteration", "a", "b", "Tolerance")
18
19     for {
20         c = (a + b) / 2
21         iterations++
22         fmt.Printf("%-12d %-12.6f %-12.6f %-12.6f\n", iterations, a, b, (b-a)/2)
23
24         if f(c) == 0 || (b-a)/2 < tol {
25             break
26         }
27
28         if f(c)*f(a) < 0 {
29             b = c
30         } else {
31             a = c
32         }
33     }
34
35     return c
36 }
```

Запуск всех методов

Код

```
1 package main
2
3 import (
4     "fmt"
5     "math"
6
7     "github.com/orenvadi/7methods/methods"
8 )
9
10 // деления пополам, Ньютона, простой итерации, метод хорд, метод секущих,
11 func f(x float64) float64 {
12     return math.Pow(x, 5) - x + 0.2
13 }
14
15 func derivativeF(x float64) float64 {
16     return math.Pow(x, 4)*5 - 1
17 }
18
19 func main() {
20     a := -2.0 // Lower bound of the interval
21     b := 2.0  // Upper bound of the interval
22     x0 := (a + b) / 2
23     // x0 := a
24     tol := 0.0001 // Tolerance
25     maxIterations := 20
26
27     fmt.Println(derivativeF(100.0))
28
29     // Half method
30     fmt.Println("\n-----
31 Half-----")
32     root := methods.HalfMethod(f, a, b, tol)
33     fmt.Printf("\nApproximate root: %.6f\n", root)
34     fmt.Println("\n-----")
35
36     // Newton method
37     fmt.Println("\n-----
38 Newton-----")
39     rootNewton := methods.NewtonMethod(f, derivativeF, x0, tol)
40     fmt.Printf("\nApproximate root: %.6f\n", rootNewton)
41     fmt.Println("\n-----")
42
43     // Simple Iteration method
44     fmt.Println("\n-----
45 SimpleIteration-----")
```

```

45     rootSimpleIter := methods.SimpleIterationMethod(f, x0, tol, maxIterations)
46     fmt.Printf("\nApproximate root: %.6f\n", rootSimpleIter)
47
48     fmt.Println("\n-----")
49     // Chord method
50     fmt.Println("\n-----")
51     Chord-----")
52     rootChord := methods.ChordMethod(f, x0, a, tol, maxIterations)
53     fmt.Printf("\nApproximate rootChord: %.6f\n", rootChord)
54
55     fmt.Println("\n-----")
56     // Secant method
57     fmt.Println("\n-----")
58     Secant-----")
59     rootSecant, err := methods.SecantMethod(f, x0, a, tol, maxIterations)
60     if err != nil {
61         fmt.Println(err)
62     } else {
63         fmt.Printf("Approximate root: %.8f\n", rootSecant)
64     }
65
66     fmt.Println("\n-----")
67
68     //
69     // For algebraic equations
70
71     fmt.Printf("\n\n")
72     fmt.Println("+-----+")
73     fmt.Println("|                               Algebraic Equations")
74     fmt.Println("|")
75     fmt.Println("+-----+")
76     fmt.Printf("\n\n")
77     //
78     //
79
80     fmt.Println("\n-----")
81     Kramer-----")
82
83     AKram := [][]float64{
84         {3, -2, 4},
85         {3, 4, -2},
86         {2, -1, -1},
87     }
88
89     bKram := []float64{21, 9, 10}
90

```

```

91 rootsKr, err := methods.KramerMethod(AKram, bKram)
92 if err != nil {
93     fmt.Println("Error:", err)
94     return
95 }
96
97 fmt.Println("Roots:")
98 for i, root := range rootsKr {
99     fmt.Printf("x%d = %.2f\n", i+1, root)
100 }
101
102 fmt.Println("\n-----")
103
104 fmt.Println("\n-----")
105 Gauss-----")
106 AGs := [][]float64{
107     {3, -2, 4},
108     {3, 4, -2},
109     {2, -1, -1},
110 }
111
112 bGs := []float64{21, 9, 10}
113
114 rootsGs := methods.GaussMethod(AGs, bGs)
115
116 fmt.Println("Roots:")
117 for i, root := range rootsGs {
118     fmt.Printf("x%d = %.2f\n", i+1, root)
119 }
120
121 fmt.Println("\n-----")
122 }

```

Результат запуска

```

1  4.99999999e+08
2
3  -----Half-----
4  Iteration    a          b          Tolerance
5  1           -2.000000    2.000000    2.000000
6  2           -2.000000    0.000000    1.000000
7  3           -2.000000   -1.000000    0.500000
8  4           -1.500000   -1.000000    0.250000
9  5           -1.250000   -1.000000    0.125000
10 6           -1.125000   -1.000000    0.062500
11 7           -1.062500   -1.000000    0.031250
12 8           -1.062500   -1.031250    0.015625
13 9           -1.046875   -1.031250    0.007812
14 10          -1.046875   -1.039062    0.003906
15 11          -1.046875   -1.042969    0.001953

```

```

16 12          -1.044922    -1.042969    0.000977
17 13          -1.044922    -1.043945    0.000488
18 14          -1.044922    -1.044434    0.000244
19 15          -1.044922    -1.044678    0.000122
20 16          -1.044800    -1.044678    0.000061
21
22 Approximate root: -1.044739
23
24 -----
25
26 -----Newton-----
27 Iteration    x          Tolerance
28 1            0.200000    0.200000
29 2            0.200323    0.000323
30 3            0.200323    0.000000
31
32 Approximate root: 0.200323
33
34 -----
35
36 -----SimpleIteration-----
37 Iteration    x          Tolerance
38 1            0.200000    0.200000
39 2            0.000320    0.199680
40 3            0.199680    0.199360
41 4            0.000637    0.199043
42 5            0.199363    0.198725
43 6            0.000952    0.198410
44 7            0.199048    0.198095
45 8            0.001265    0.197783
46 9            0.198735    0.197470
47 10           0.001575    0.197160
48 11           0.198425    0.196850
49 12           0.001882    0.196543
50 13           0.198118    0.196235
51 14           0.002188    0.195930
52 15           0.197812    0.195625
53 16           0.002491    0.195322
54 17           0.197509    0.195019
55 18           0.002791    0.194718
56 19           0.197209    0.194418
57 20           0.003089    0.194120
58 Maximum number of iterations reached.
59
60 Approximate root: 0.003089
61
62 -----
63
64 -----Chord-----
65 Iteration    x0          x1          Tolerance
66 1            0.000000    -2.000000    1.986667

```

```

67 2          -2.000000  -0.013333  0.014121
68 3          -0.013333  -0.027454  0.227455
69 4          -0.027454  0.200000  0.000320
70 5          0.200000  0.200320  0.000002
71
72 Approximate rootChord: 0.200323
73
74 -----
75
76 -----Secant-----
77 Iteration    x          f(x)          Tolerance
78 0          -0.013333  -29.800000  1.986667
79 1          -0.027454  0.213333  0.014121
80 2          0.200000  0.227454  0.227455
81 3          0.200320  0.000320  0.000320
82
83 Converged with tolerance: 1.000000e-04
84 Approximate root: 0.20032045
85
86 -----
87
88
89 +-----+
90 |                      Algebraic Equations                      |
91 +-----+
92
93
94
95 -----Kramer-----
96 Roots:
97 x1 = 5.00
98 x2 = -1.00
99 x3 = 1.00
100
101 -----
102
103 -----Gauss-----
104 Roots:
105 x1 = 5.00
106 x2 = -1.00
107 x3 = 1.00
108
109 -----

```