СРСП №3

Численные метолы решения нелинейных уравнений

Бактыбеков Нурсултан

2 Октября, 2023

Содержание

Цель работы

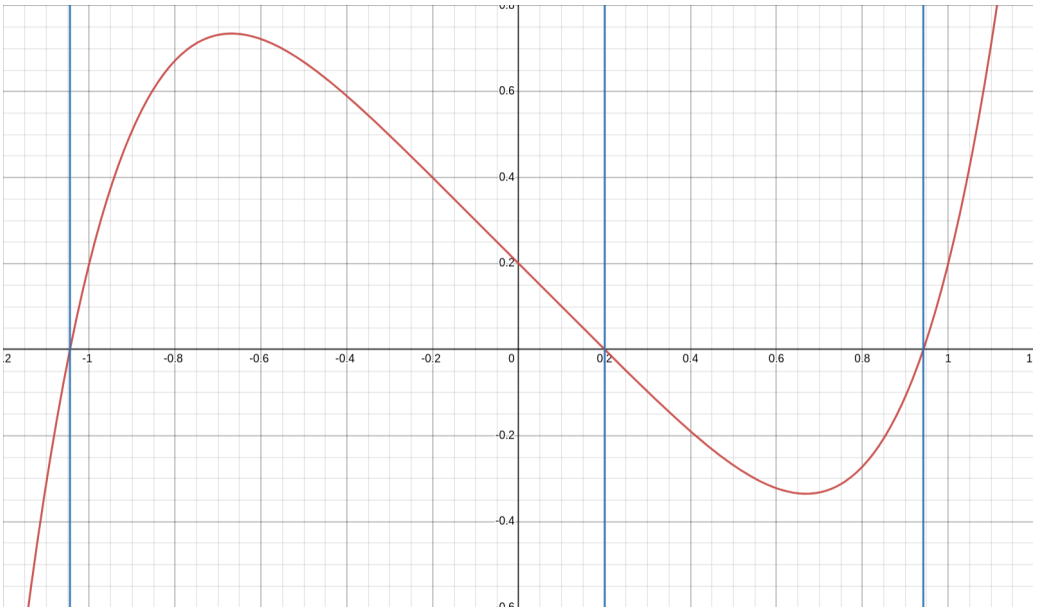
Научиться выполнять вычисления с приближенными числами с учетом погрешностей.

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

```go
package methods

import (
  "fmt"
  "math"
)

func HalfMethod(f func(float64) float64, a, b, tol float64) float64 {
  if f(a)*f(b) >= 0 {
    fmt.Println("Cannot use bisection method on this interval.")
    return math.NaN()
  }

  iterations := 0
  var c float64

  fmt.Printf("%-12s %-12s %-12s %-12s\n", "Iteration", "a", "b", "Tolerance")

  for {
    c = (a + b) / 2
    iterations++
    fmt.Printf("%-12d %-12.6f %-12.6f %-12.6f\n", iterations, a, b, (b-a)/2)

    if f(c) == 0 || (b-a)/2 < tol {
      break
    }

    if f(c)*f(a) < 0 {
      b = c
    } else {
      a = c
    }
  }

  return c
}
```

Метод Ньютона

Решение

```go
package methods

import (
  "fmt"
  "math"
)

func NewtonMethod(f func(float64) float64, df func(float64) float64, x0, tol
float64) float64 {
  iterations := 0

  fmt.Printf("%-12s %-12s %-12s\n", "Iteration", "x", "Tolerance")

  for {
    iterations++
    x1 := x0 - f(x0)/df(x0)
    tolerance := math.Abs(x1 - x0)

    fmt.Printf("%-12d %-12.6f %-12.6f\n", iterations, x1, tolerance)

    if tolerance < tol {
      return x1
    }

    x0 = x1
  }
}
```

Метод простой итерации

Решение

```go
package methods

import (
  "fmt"
  "math"
)

func SimpleIterationMethod(f func(float64) float64, x0, tol float64,
maxIterations int) float64 {
  iterations := 0

  fmt.Printf("%-12s %-12s %-12s\n", "Iteration", "x", "Tolerance")

  for iterations < maxIterations {
    iterations++
```

```
    x1 := f(x0)
    tolerance := math.Abs(x1 - x0)

    fmt.Printf("%-12d %-12.6f %-12.6f\n", iterations, x1, tolerance)

    if tolerance < tol {
      return x1
    }

    x0 = x1
  }

  fmt.Println("Maximum number of iterations reached.")
  return x0
}
```

Метод хорд

Решение

```
package methods

import (
  "fmt"
  "math"
)

func ChordMethod(f func(float64) float64, x0, x1, tol float64, maxIterations
int) float64 {
  iterations := 0

  fmt.Printf("%-12s %-12s %-12s %-12s\n", "Iteration", "x0", "x1",
"Tolerance")

  for iterations < maxIterations {
    iterations++
    fx0 := f(x0)
    fx1 := f(x1)

    x2 := x1 - (fx1 * (x1 - x0) / (fx1 - fx0))
    tolerance := math.Abs(x2 - x1)

    fmt.Printf("%-12d %-12.6f %-12.6f %-12.6f\n", iterations, x0, x1,
tolerance)

    if tolerance < tol {
      return x2
    }

    x0 = x1
```

```
    x1 = x2
  }

  fmt.Println("Maximum number of iterations reached.")
  return x1
}
```

Метод метод секущих

Решение

```
package methods

import (
  "fmt"
  "math"
)

func SecantMethod(f func(float64) float64, x0, x1, tol float64, maxIter int)
(float64, error) {
  fmt.Printf("%-12s %-12s %-12s %-12s\n", "Iteration", "x", "f(x)",
"Tolerance")
  for i := 0; i < maxIter; i++ {
    fx0 := f(x0)
    fx1 := f(x1)

    if math.Abs(fx1) < tol {
      fmt.Printf("\nConverged with tolerance: %e\n", tol)
      return x1, nil
    }

    x2 := x1 - (fx1 * (x1 - x0) / (fx1 - fx0))

    fmt.Printf("%-12d %-12.6f %-12.6f %-12.6f\n", i, x2, fx1, math.Abs(x2-
x1))

    if math.Abs(x2-x1) < tol {
      fmt.Printf("\nConverged with tolerance: %e\n", tol)
      return x2, nil
    }

    x0 = x1
    x1 = x2
  }

  return 0, fmt.Errorf("Maximum number of iterations reached")
}
```

Метод Гаусса

Решение

```go
package methods

func GaussMethod(matrix [][]float64, constants []float64) []float64 {
  n := len(matrix)
  solution := make([]float64, n)

  // Step 1: Forward Elimination
  for i := 0; i < n; i++ {
    // Find the row with the maximum element in the current column
    maxRow := i
    for j := i + 1; j < n; j++ {
      if matrix[j][i] > matrix[maxRow][i] {
        maxRow = j
      }
    }

    // Swap the current row with the row containing the maximum element
    matrix[i], matrix[maxRow] = matrix[maxRow], matrix[i]
    constants[i], constants[maxRow] = constants[maxRow], constants[i]

    // Make the diagonal element of the current row equal to 1
    pivot := matrix[i][i]
    for j := i; j < n; j++ {
      matrix[i][j] /= pivot
    }
    constants[i] /= pivot

    // Eliminate all other elements in the current column
    for j := i + 1; j < n; j++ {
      factor := matrix[j][i]
      for k := i; k < n; k++ {
        matrix[j][k] -= factor * matrix[i][k]
      }
      constants[j] -= factor * constants[i]
    }
  }

  // Step 2: Back Substitution
  for i := n - 1; i >= 0; i-- {
    // Solve for the variable at the current row
    solution[i] = constants[i]
    for j := i + 1; j < n; j++ {
      solution[i] -= matrix[i][j] * solution[j]
    }
  }


```

```
48    return solution
}
```

Метод Крамера

Решение

```
package methods

import (
  "fmt"
  "math"
)

func HalfMethod(f func(float64) float64, a, b, tol float64) float64 {
  if f(a)*f(b) >= 0 {
    fmt.Println("Cannot use bisection method on this interval.")
    return math.NaN()
  }

  iterations := 0
  var c float64

  fmt.Printf("%-12s %-12s %-12s %-12s\n", "Iteration", "a", "b", "Tolerance")

  for {
    c = (a + b) / 2
    iterations++
    fmt.Printf("%-12d %-12.6f %-12.6f %-12.6f\n", iterations, a, b, (b-a)/2)

    if f(c) == 0 || (b-a)/2 < tol {
      break
    }

    if f(c)*f(a) < 0 {
      b = c
    } else {
      a = c
    }
  }

  return c
}
```

Запуск всех методов

Код

```go
package main

import (
  "fmt"
  "math"

  "github.com/orenvadi/7methods/methods"
)

// деления пополам, Ньютона, простой итерации, метод хорд, метод секущих,
func f(x float64) float64 {
  return math.Pow(x, 5) - x + 0.2
}

func derivativeF(x float64) float64 {
  return math.Pow(x, 4)*5 - 1
}

func main() {
  a := -2.0 // Lower bound of the interval
  b := 2.0  // Upper bound of the interval
  x0 := (a + b) / 2
  // x0 := a
  tol := 0.0001 // Tolerance
  maxIterations := 20

  fmt.Println(derivativeF(100.0))

  // Half method
  fmt.Println("\n----------------------------
Half----------------------------")
  root := methods.HalfMethod(f, a, b, tol)
  fmt.Printf("\nApproximate root: %.6f\n", root)

fmt.Println("\n------------------------------------------------------------")

  // Newton method
  fmt.Println("\n----------------------------
Newton----------------------------")
  rootNewton := methods.NewtonMethod(f, derivativeF, x0, tol)
  fmt.Printf("\nApproximate root: %.6f\n", rootNewton)

fmt.Println("\n------------------------------------------------------------")

  // Simple Iteration method
  fmt.Println("\n----------------------------
SimpleIteration----------------------------")
```

```
  rootSimpleIter := methods.SimpleIterationMethod(f, x0, tol, maxIterations)
  fmt.Printf("\nApproximate root: %.6f\n", rootSimpleIter)

fmt.Println("\n------------------------------------------------------------------------")

  // Chord method
  fmt.Println("\n---------------------------
Chord-----------------------------")
  rootChord := methods.ChordMethod(f, x0, a, tol, maxIterations)
  fmt.Printf("\nApproximate rootChord: %.6f\n", rootChord)

fmt.Println("\n------------------------------------------------------------------")

  // Secant method
  fmt.Println("\n---------------------------
Secant-----------------------------")
  rootSecant, err := methods.SecantMethod(f, x0, a, tol, maxIterations)
  if err != nil {
    fmt.Println(err)
  } else {
    fmt.Printf("Approximate root: %.8f\n", rootSecant)
  }

fmt.Println("\n------------------------------------------------------------------")

  //
  //
  // For algebraic equations

  fmt.Printf("\n\n")
  fmt.Println("+
------------------------------------------------------------------------+")
  fmt.Println("|                              Algebraic Equations
|")
  fmt.Println("+
------------------------------------------------------------------------+")
  fmt.Printf("\n\n")
  //
  //

  fmt.Println("\n---------------------------
Kramer-----------------------------")

  AKram := [][]float64{
    {3, -2, 4},
    {3, 4, -2},
    {2, -1, -1},
  }

  bKram := []float64{21, 9, 10}

```

```
  rootsKr, err := methods.KramerMethod(AKram, bKram)
  if err != nil {
    fmt.Println("Error:", err)
    return
  }

  fmt.Println("Roots:")
  for i, root := range rootsKr {
    fmt.Printf("x%d = %.2f\n", i+1, root)
  }

fmt.Println("\n------------------------------------------------------------------")

  fmt.Println("\n---------------------------
Gauss-----------------------------")
  AGs := [][]float64{
    {3, -2, 4},
    {3, 4, -2},
    {2, -1, -1},
  }

  bGs := []float64{21, 9, 10}

  rootsGs := methods.GaussMethod(AGs, bGs)

  fmt.Println("Roots:")
  for i, root := range rootsGs {
    fmt.Printf("x%d = %.2f\n", i+1, root)
  }

fmt.Println("\n------------------------------------------------------------------")
}
```

Результат запуска

```
4.99999999e+08

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

```
12           -1.044922    -1.042969    0.000977
13           -1.044922    -1.043945    0.000488
14           -1.044922    -1.044434    0.000244
15           -1.044922    -1.044678    0.000122
16           -1.044800    -1.044678    0.000061

Approximate root: -1.044739

------------------------------------------------------------

---------------------------Newton---------------------------
Iteration    x            Tolerance
1            0.200000     0.200000
2            0.200323     0.000323
3            0.200323     0.000000

Approximate root: 0.200323

------------------------------------------------------------

-------------------------SimpleIteration--------------------------
Iteration    x            Tolerance
1            0.200000     0.200000
2            0.000320     0.199680
3            0.199680     0.199360
4            0.000637     0.199043
5            0.199363     0.198725
6            0.000952     0.198410
7            0.199048     0.198095
8            0.001265     0.197783
9            0.198735     0.197470
10           0.001575     0.197160
11           0.198425     0.196850
12           0.001882     0.196543
13           0.198118     0.196235
14           0.002188     0.195930
15           0.197812     0.195625
16           0.002491     0.195322
17           0.197509     0.195019
18           0.002791     0.194718
19           0.197209     0.194418
20           0.003089     0.194120
Maximum number of iterations reached.

Approximate root: 0.003089

------------------------------------------------------------------------

---------------------------Chord----------------------------
Iteration    x0           x1           Tolerance
1            0.000000     -2.000000    1.986667
```

```
2            -2.000000    -0.013333    0.014121
3            -0.013333    -0.027454    0.227455
4            -0.027454    0.200000     0.000320
5            0.200000     0.200320     0.000002

Approximate rootChord: 0.200323

------------------------------------------------------------

---------------------------Secant-----------------------------
Iteration    x            f(x)         Tolerance
0            -0.013333    -29.800000   1.986667
1            -0.027454    0.213333     0.014121
2            0.200000     0.227454     0.227455
3            0.200320     0.000320     0.000320

Converged with tolerance: 1.000000e-04
Approximate root: 0.20032045

------------------------------------------------------------


+-------------------------------------------------------------------------+
|                          Algebraic Equations                            |
+-------------------------------------------------------------------------+



---------------------------Kramer-----------------------------
Roots:
x1 = 5.00
x2 = -1.00
x3 = 1.00

------------------------------------------------------------

---------------------------Gauss------------------------------
Roots:
x1 = 5.00
x2 = -1.00
x3 = 1.00

------------------------------------------------------------
```