Московский авиационный институт
(Национальный Исследовательский Институт)

Институт №8 информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №2
по курсу «Численные методы»

Студент: Суханов Е.А

Группа: М8О-406Б-19

Оценка:

Преподаватель: Ревизников Д.Л.

Подпись:

Москва
2022

Задание 2.1

Реализовать методы простой итерации и Ньютона решения нелинейных уравнений в виде программ, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения найти положительный корень нелинейного уравнения (начальное приближение определить графически). Проанализировать зависимость погрешности вычислений от количества итераций.

Вариант:

$$e^{2x}+3x-4$$

Листинг

main.go:

```
package main

import (
  "fmt"
  "math"
)

type fn func(float64) float64

func newtonMethod(f, df fn, a, b, eps float64) (x float64, itcnt int) {
  var px float64
  x = a + (b-a)/2.

  for ; math.Abs(x-px) >= eps || itcnt == 0; itcnt++ {
    px = x
    x = x - f(x)/df(x)
    fmt.Println("iter: ", itcnt, "x: ", x)
  }

  return x, itcnt
}

func iterationMethod(f, df fn, a, b, eps float64) (x float64, itcnt int) {
  var px, q float64
  x = a + (b-a)/2.
  q = math.Max(math.Abs(df(a)), math.Abs(df(b)))
  fmt.Println("q: ", q)
  q = q / (1 - q)

  for ; math.Abs(x-px)*q >= eps || itcnt == 0; itcnt++ {
    px = x
    x = f(x)
```

```
    fmt.Println("iter: ", itcnt, "x: ", x)
  }

  return x, itcnt
}

func main() {
  var eps float64
  fmt.Scan(&eps)

  // f(x) = e ^ (2x) + 3x - 4
  f := func(x float64) float64 {
    return math.Exp(2*x) + 3*x - 4
  }
  df := func(x float64) float64 {
    return math.Exp(2*x) + 3
  }

  // f(x) = e ^ (2x) + 3x - 4 => x = ln(4-3x)/2
  phi := func(x float64) float64 {
    return math.Log(4-3*x) / 2
  }
  dphi := func(x float64) float64 {
    return -3 / (2 * (4 - 3*x))
  }

  {
    fmt.Println("Метод простых итераций")
    x, itcnt := iterationMethod(phi, dphi, 0.4, 0.6, eps)
    fmt.Println("Количество итераций: ", itcnt, "\tx:", x)
  }
  {
    fmt.Println("Метод Ньютона")
    x, itcnt := newtonMethod(f, df, 0.4, 0.6, eps)
    fmt.Println("Количество итераций: ", itcnt, "\tx:", x)
  }
}
```

Вывод программы

```
0.001
Метод простых итераций
q:  0.6818181818181818
iter:  0 x:  0.45814536593707755
iter:  1 x:  0.4826478464272677
iter:  2 x:  0.46844974467184275
```

```
iter:  3 x:  0.47672596398955785
iter:  4 x:  0.4719183389232763
iter:  5 x:  0.47471669458899934
iter:  6 x:  0.47308977210644126
iter:  7 x:  0.47403628478294113
iter:  8 x:  0.47348583964270274
iter:  9 x:  0.4738060251451568
Количество итераций:  10        x: 0.4738060251451568
Метод Ньютона
iter:  0 x:  0.461827374899102
iter:  1 x:  0.47923070202801515
iter:  2 x:  0.47113942118190705
iter:  3 x:  0.47486930552289197
iter:  4 x:  0.47314294387714
iter:  5 x:  0.47394050732493836
Количество итераций:  6         x: 0.47394050732493836
```

Задание 2.2

Реализовать методы простой итерации и Ньютона решения систем нелинейных уравнений в виде программного кода, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения решить систему нелинейных уравнений (при наличии нескольких решений найти то из них, в котором значения неизвестных являются положительными); начальное приближение определить графически. Проанализировать зависимость погрешности вычислений от количества итераций.

Вариант:

$$x_1^2 + x_2^2 - 4 = 0$$

$$x_1 - e^{x_2} + 2 = 0$$

Листинг

main.go:

```go
package main

import (
  "fmt"
  "math"

  "github.com/Reterer/number_methods/internal/lu_decompose"
  "github.com/Reterer/number_methods/internal/utils"
  "github.com/Reterer/number_methods/pkg/matrix"
)

type fn func(x *matrix.RMatrix) float64
type sfn struct {
  n   int
  fns []fn
  jak [][]fn
}

func (s *sfn) calc(x *matrix.RMatrix) *matrix.RMatrix {
  res := matrix.MakeRealMatrix(s.n, 1)
  for i := 0; i < s.n; i++ {
    res.SetEl(i, 0, s.fns[i](x))
  }
  return res
}

func (s *sfn) Jak(x *matrix.RMatrix) *matrix.RMatrix {
```

```
  jak := matrix.MakeRealMatrix(s.n, s.n)
  for i := 0; i < s.n; i++ {
    coljak := s.jak[i]
    colres := jak.GetCol(i)
    for j := 0; j < s.n; j++ {
      colres[j] = coljak[j](x)
    }
  }

  return jak
}

func norm(x *matrix.RMatrix) float64 {
  var ans float64
  n, m := x.Shape()
  for i := 0; i < n; i++ {
    var sum float64
    colX := x.GetCol(i)
    for j := 0; j < m; j++ {
      sum += math.Abs(colX[j])
    }
    if sum > ans {
      ans = sum
    }
  }

  return ans
}

func calcDet(x *matrix.RMatrix) float64 {
  n, _ := x.Shape()
  lu := lu_decompose.MakeLU(lu_decompose.PermMin, x)
  det := float64(1)
  for i := 0; i < n; i++ {
    det *= lu.U.GetEl(i, i)
  }
  return det
}

func newtonMethod(a, b *matrix.RMatrix, s *sfn, eps float64) (x *matrix.RMatrix,
itcnt int) {
  n := s.n
  px := matrix.MakeRealMatrix(n, 1)
  x = b.Add(a).MulByConstant(1. / 2.) // x = (a + b) / 2

  for ; norm(x.Add(px.MulByConstant(-1))) > eps || itcnt == 0; itcnt++ {
    jak := s.Jak(x)
    detJak := calcDet(jak)

    dsf := matrix.MakeRealMatrix(n, 1)
    temp := matrix.MakeRealMatrix(n, 1)
```

```
    for j := 0; j < n; j++ {
      for i := 0; i < n; i++ {
        temp.SetEl(i, 0, jak.GetEl(i, j))
        jak.SetEl(i, j, s.fns[i](x))
      }
      dsf.SetEl(j, 0, -calcDet(jak)/detJak)
      for i := 0; i < n; i++ {
        jak.SetEl(i, j, temp.GetEl(i, 0))
      }
    }

    px = x
    x = x.Add(dsf)
    fmt.Println("iter:", itcnt, "\tx: ")
    utils.PrintMatrix(x)
  }

  return x, itcnt
}

func iterationMethod(a, b *matrix.RMatrix, s *sfn, eps float64) (x *matrix.RMatrix,
itcnt int) {
  n := s.n
  px := matrix.MakeRealMatrix(n, 1)
  x = b.Add(a).MulByConstant(1. / 2.) // x = (a + b) / 2

  var q float64
  {
    jak := s.Jak(x)
    q = norm(jak)
    fmt.Println(q)
    q = q / (1 - q)
  }

  for ; norm(x.Add(px.MulByConstant(-1)))*q > eps || itcnt == 0; itcnt++ {
    px = x
    x = s.calc(x)
    fmt.Println("iter:", itcnt, "\tx: ")
    utils.PrintMatrix(x)
  }

  return x, itcnt
}

func fisrtSfn() *sfn {
  return &sfn{
    n: 2,
    fns: []fn{
      func(x *matrix.RMatrix) float64 { return math.Pow(x.GetEl(0, 0), 2) +
math.Pow(x.GetEl(1, 0), 2) - 4 }, // f1
      func(x *matrix.RMatrix) float64 { return x.GetEl(0, 0) - math.Exp(x.GetEl(1,
```

```
0)) + 2 },                  // f2
    },
    jak: [][]fn{
      {
        func(x *matrix.RMatrix) float64 { return 2 * x.GetEl(0, 0) },
        func(x *matrix.RMatrix) float64 { return 2 * x.GetEl(1, 0) },
      },
      {
        func(x *matrix.RMatrix) float64 { return 1 },
        func(x *matrix.RMatrix) float64 { return -math.Exp(x.GetEl(1, 0)) },
      },
    },
  }
}

func secondSfn() *sfn {
  return &sfn{
    n: 2,
    fns: []fn{
      func(x *matrix.RMatrix) float64 { return math.Sqrt(4 - math.Pow(x.GetEl(1,
0), 2)) }, // f1
      func(x *matrix.RMatrix) float64 { return math.Log(x.GetEl(0, 0) + 2) },
// f2
    },
    jak: [][]fn{
      {
        func(x *matrix.RMatrix) float64 { return 0 },
        func(x *matrix.RMatrix) float64 { return -x.GetEl(1, 0) / math.Sqrt(4-
math.Pow(x.GetEl(1, 0), 2)) },
      },
      {
        func(x *matrix.RMatrix) float64 { return 1. / (x.GetEl(0, 0) + 2) },
        func(x *matrix.RMatrix) float64 { return 0 },
      },
    },
  }
}

func main() {
  var eps float64
  fmt.Scan(&eps)

  {
    system := fisrtSfn()
    a := matrix.MakeRealMatrix(2, 1)
    b := matrix.MakeRealMatrix(2, 1)
    a.SetEl(0, 0, 1.0)
    b.SetEl(0, 0, 2.0)
    a.SetEl(1, 0, 0.5)
    b.SetEl(1, 0, 1.5)
```

```
    fmt.Println("Newton Method")
    newtonMethod(a, b, system, eps)
  }

  {
    system := secondSfn()
    a := matrix.MakeRealMatrix(2, 1)
    b := matrix.MakeRealMatrix(2, 1)
    a.SetEl(0, 0, 1.0)
    b.SetEl(0, 0, 2.0)
    a.SetEl(1, 0, 0.5)
    b.SetEl(1, 0, 1.5)

    fmt.Println("Iteration Method")
    iterationMethod(a, b, system, eps)
  }
}
```

Вывод программы

```
0.001
Newton Method
iter: 8         x:
2 1
  1.5486
  1.2664
Iteration Method
iter: 8         x:
2 1
  1.5485
  1.2663
```