Numerical analysis laboratory work №2

Ivan Zhytkevych FI-91

System

$$A = \begin{bmatrix} 2.12 & 0.42 & 1.34 & 0.88 \\ 0.42 & 3.95 & 1.87 & 0.43 \\ 1.34 & 1.87 & 2.98 & 0.46 \\ 0.88 & 0.43 & 0.46 & 4.44 \end{bmatrix}$$

$$b = \begin{bmatrix} 11.172 & 0.115 & 0.009 & 9.349 \end{bmatrix}$$

LU-decomposition calculation

Listing the following program output of C matrix:

```
C after c_11 and c_1j inits:
| +1.456022  +0.288457  +0.920316  +0.604386   |
| +0.000000  +0.000000  +0.000000  +0.000000   |
| +0.000000  +0.000000  +0.000000  +0.000000   |
| +0.000000  +0.000000  +0.000000  +0.000000   |

C on 1,0 step
| +1.456022  +0.288457  +0.920316  +0.604386   |
| +0.000000  +0.000000  +0.000000  +0.000000   |
| +0.000000  +0.000000  +0.000000  +0.000000   |
| +0.000000  +0.000000  +0.000000  +0.000000   |

C on 1,1 step
| +1.456022  +0.288457  +0.920316  +0.604386   |
| +0.000000  +0.000000  +0.000000  +0.000000   |
| +0.000000  +0.000000  +0.000000  +0.000000   |
| +0.000000  +0.000000  +0.000000  +0.000000   |

C on 1,2 step
| +1.456022  +0.288457  +0.920316  +0.604386   |
| +0.000000  +1.966416  +0.000000  +0.000000   |
| +0.000000  +0.000000  +0.000000  +0.000000   |
| +0.000000  +0.000000  +0.000000  +0.000000   |
```

```
C on 1,3 step
| +1.456022  +0.288457  +0.920316  +0.604386   |
| +0.000000  +1.966416  +0.815966  +0.000000   |
| +0.000000  +0.000000  +0.000000  +0.000000   |
| +0.000000  +0.000000  +0.000000  +0.000000   |

C on 2,0 step
| +1.456022  +0.288457  +0.920316  +0.604386   |
| +0.000000  +1.966416  +0.815966  +0.130013   |
| +0.000000  +0.000000  +0.000000  +0.000000   |
| +0.000000  +0.000000  +0.000000  +0.000000   |

C on 2,1 step
| +1.456022  +0.288457  +0.920316  +0.604386   |
| +0.000000  +1.966416  +0.815966  +0.130013   |
| +0.000000  +0.000000  +0.000000  +0.000000   |
| +0.000000  +0.000000  +0.000000  +0.000000   |

C on 2,2 step
| +1.456022  +0.288457  +0.920316  +0.604386   |
| +0.000000  +1.966416  +0.815966  +0.130013   |
| +0.000000  +0.000000  +0.000000  +0.000000   |
| +0.000000  +0.000000  +0.000000  +0.000000   |

C on 2,3 step
| +1.456022  +0.288457  +0.920316  +0.604386   |
| +0.000000  +1.966416  +0.815966  +0.130013   |
| +0.000000  +0.000000  +1.211288  +0.000000   |
| +0.000000  +0.000000  +0.000000  +0.000000   |

C on 3,0 step
| +1.456022  +0.288457  +0.920316  +0.604386   |
| +0.000000  +1.966416  +0.815966  +0.130013   |
| +0.000000  +0.000000  +1.211288  -0.167023   |
| +0.000000  +0.000000  +0.000000  +0.000000   |

C on 3,1 step
| +1.456022  +0.288457  +0.920316  +0.604386   |
| +0.000000  +1.966416  +0.815966  +0.130013   |
| +0.000000  +0.000000  +1.211288  -0.167023   |
| +0.000000  +0.000000  +0.000000  +0.000000   |

C on 3,2 step
| +1.456022  +0.288457  +0.920316  +0.604386   |
| +0.000000  +1.966416  +0.815966  +0.130013   |
| +0.000000  +0.000000  +1.211288  -0.167023   |
```

```
| +0.000000  +0.000000  +0.000000  +0.000000   |

C on 3,3 step
| +1.456022  +0.288457  +0.920316  +0.604386   |
| +0.000000  +1.966416  +0.815966  +0.130013   |
| +0.000000  +0.000000  +1.211288  -0.167023   |
| +0.000000  +0.000000  +0.000000  +0.000000   |
```

Final result

```
Solution =
| +7.220064   |
| +1.083311   |
| -4.076517   |
| +0.992054   |

A \cdot solution =
| +11.172000   |
| +0.115000   |
| +0.009000   |
| +9.349000   |

b eps =
| +0.000000   |
| -0.000000   |
| -0.000000   |
| +0.000000   |
```

So the solution is

$$\begin{bmatrix} 7.220064 & 1.083311 & -4.076517 & 0.992054 \end{bmatrix}$$

Extending the output:

```
Solution =
| 7.220063842915758    |
| 1.0833107023256405    |
| -4.076517192429001    |
| 0.9920536317007763    |

A \cdot solution =
| 11.171999999999999    |
| 0.11500000000000027    |
| 0.009000000000000175    |
| 9.349    |

b eps =
```

```
| 1.7763568394002505e-15    |
| -2.636779683484747e-16    |
| -1.7520707107365752e-16    |
| 0    |
```

And the error is:

```
x eps
| 1.3598070311770764e-15    |
| 1.6136923231012026e-16    |
| -7.3932187240093535e-16    |
| -2.0854299460016857e-16    |
```

Listing

Written on Golang

```go
// matrix.go
package main

import (
    "fmt"
    "math"
)

type Matrix struct {
    data [][]float64
    m uint
    n uint
}

func MatrixInit(m, n uint) *Matrix {
    M := new(Matrix)
    M.m = m
    M.n = n
    M.data = make([][]float64, m)
    for mi := range M.data {
        M.data[mi] = make([]float64, n)
    }
    return M
}

func NewMatrix(m [][]float64) *Matrix {
    if len(m) == 0 {
        return MatrixInit(0, 0)
    }
    n := len(m[0])
```

```go
    M := MatrixInit(uint(len(m)), uint(n))
    for i := range m {
        copy(M.data[i], m[i])
    }
    return M;
}

func (m Matrix) String() (s string) {
    // s += fmt.Sprintf("--\n")
    s += "\n"
    for _, row := range m.data {
        s += fmt.Sprintf("| ")
        for _, item := range row {
            // s += fmt.Sprintf("%+02.6f   ", item)
            s += fmt.Sprint(item)
            s += "    "
        }
        s += fmt.Sprintf(" |\n")
    }
    // s += fmt.Sprintf("--\n")
    return
}


func (M Matrix) LUDecompose() *Matrix {
    // cannot decompose not square matrixes
    if M.m != M.n { return nil }

    C := MatrixInit(M.m, M.n)

    // c_{ii} = sqrt{ a_{ii} - sum_{k=1}^{i-1} c_{ki}^2 }      i = \overline{2, n}
    calc_diag := func(i uint) float64 {
        var cSum float64 = 0
        for k := uint(0); k < i; k++ {
            cSum += math.Pow(C.data[k][i], 2)
        }
        return math.Sqrt(M.data[i][i] - cSum)
    }

    calc_other := func(i, j uint) float64 {
        var cSum float64
        for k := uint(0); k < i; k++ {
            cSum += C.data[k][i] * C.data[k][j]
        }
        return (M.data[i][j] - cSum) / C.data[i][i]
    }
```

```
    // c_{11} = \sqrt{a_{11}}
    C.data[0][0] = math.Sqrt(M.data[0][0])

    // c_{1j} = \frac {a_{1j}} {c_{11}}     j = \overline{2, n}
    for j := uint(1); j < M.n; j++ {
        if C.data[0][0] == 0 { return nil }
        C.data[0][j] = M.data[0][j] / C.data[0][0]
    }

    fmt.Println("C after c_11 and c_1j inits:", C)

    for i := uint(1); i < C.m; i++ {
        for j := uint(0); j < C.n; j++ {
            fmt.Printf("C on %d,%d step", i, j)
            fmt.Println(C)
            if j < i {
                C.data[i][j] = 0
            } else if j == i {
                C.data[i][j] = calc_diag(i)
            } else if j > i {
                C.data[i][j] = calc_other(i, j)
            }
        }
    }

    return C
}

func (M *Matrix) SolveSQRTMethod(b *Matrix) *Matrix {
    if b.m != M.m { return nil }
    if b.n != 1   { return nil }
    if M.m != M.n { return nil }

    n := M.m

    C := M.LUDecompose()
    if C == nil { return nil }

    y := MatrixInit(n, 1)
    // y_1 = b_1 / c_11
    y.data[0][0] = b.data[0][0] / C.data[0][0]

    var cSum float64;
    for i := uint(1); i < n; i++ {
        cSum = 0
```

```go
        for k := uint(0); k < i; k++ {
            cSum += C.data[k][i] * y.data[k][0]
        }
        y.data[i][0] = (b.data[i][0] - cSum) / C.data[i][i]
    }

    x := MatrixInit(n, 1)
    x.data[n-1][0] = y.data[n-1][0] / C.data[n-1][n-1]

    for i := uint(n-2); i < n-1; i-- {
        cSum = 0
        for k := uint(i+1); k < n; k++ {
            cSum += C.data[i][k] * x.data[k][0]
        }
        x.data[i][0] = (y.data[i][0] - cSum) / C.data[i][i]
    }

    return x
}

func (M Matrix) Dot(A *Matrix) *Matrix {
    if M.n != A.m { return nil }
    R := MatrixInit(M.m, A.n)
    for i := uint(0); i < M.m; i++ {
        for j := uint(0); j < A.n; j++ {
            for k := uint(0); k < M.n; k++ {
                R.data[i][j] += M.data[i][k] * A.data[k][j]
            }
        }
    }
    return R
}

func (M Matrix) Minus(A *Matrix) *Matrix {
    if M.m != A.m { return nil }
    if M.n != A.n { return nil }

    R := MatrixInit(M.m, A.n)
    for i := uint(0); i < M.m; i++ {
        for j := uint(0); j < A.n; j++ {
            R.data[i][j] = M.data[i][j] - A.data[i][j]
        }
    }
    return R
}
```

```go
// main.go
package main

import "fmt"

func main() {
    // {
    //     {2.12, 0.42, 1.34, 0.88},
    //     {0.42, 3.95, 1.87, 0.43},
    //     {1.34, 1.87, 2.98, 0.46},
    //     {0.88, 0.43, 0.46, 4.44},
    // };
    // { 11.172, 0.115, 0.009, 9.349 };

    A := NewMatrix([][]float64{
        {2.12, 0.42, 1.34, 0.88},
        {0.42, 3.95, 1.87, 0.43},
        {1.34, 1.87, 2.98, 0.46},
        {0.88, 0.43, 0.46, 4.44},
    })
    b := NewMatrix([][]float64{
        { 11.172 },
        {  0.115 },
        {  0.009 },
        {  9.349 },
    })

    solution := A.SolveSQRTMethod(b)
    fmt.Println("Solution = ", solution)

    b0 := A.Dot(solution)
    fmt.Println("A \\cdot solution = ", b0)
    fmt.Println("b eps = ", b.Minus(b0))

    eps := A.SolveSQRTMethod(b.Minus(b0))
    fmt.Println("x eps", eps)
}
```