```go
package main

import (
    "errors"
    "fmt"
    "math"
    "myReader"
    "os"
)

func main(){
    n, A, e, l0, x0, err := myReader.ReadM4("5.txt")
    if err != nil{
        panic(err)
    }
    fmt.Println("1. Max l:")
    l1, ev1, count1, err := eignvalue1(n, A, e, x0, 1, 999999)
    if err != nil {
        fmt.Println(err)
    } else {
        fmt.Println("Результаты (число, вектор, итерации): ", l1, ev1, count1)
        fmt.Println("Невязка: ", r(n, A, ev1, l1))
    }
    fmt.Println("2. Second max l:")
    l2, ev2, count2, err := eignvalue2(n, A, e, x0, ev1, 1, 1, 10000)
    if err != nil {
        fmt.Println(err)
    } else {
        fmt.Println("Результаты (число, веткор, итерации): ", l2, ev2, count2)
        fmt.Println("Невязка: ", r(n, A, ev2, l2))
    }
    fmt.Println("3. Nearest l:")
    nl, count3, err := nearestEignvalue(n, A, e, x0, l0, 10000)
    if err != nil {
        fmt.Println(err)
    } else {
        fmt.Println("Результаты (число, итерации): ", nl, count3)
    }
    fmt.Println("4. Min l:")
    minl, eMin, countMin, err := minEignvalue(n, A, e, x0, 10, -100, 10000)
    if err != nil {
        fmt.Println(err)
    } else {
        fmt.Println("Результаты (число, вектор, итерации): ", minl, eMin, countMin)
        fmt.Println("Невязка: ", r(n, A, eMin, minl))
    }
}

//вспомогательная процедура невязки
func r(n int, A [][]float64, x []float64, l float64)(res []float64){
    A_x := Ax(n, A, x)
    for i := 0; i < n; i++{
        res = append(res, A_x[i] - l * x[i])
    }
    return res
}
//попробуем организовать поиск минимального собственного числа
//вход: размерность матрицы, матрица, точность, начальный вектор, период нормализации, с, критическое число итераций
//выход:
func minEignvalue(n int, A [][]float64, e float64, x0 []float64, np int, c float64, crucialCount int) (l float64, ev []float64, count int, err error){
    B := make([][]float64, n)
    E := func(i int, j int) float64 {if i == j {return 1} else {return 0} }
    for i := 0; i < n; i++{                                                    //получить матрицу B
        B[i] = make([]float64, n)
```

```
 65         for j := 0; j < n; j++{
 66             B[i][j] = A[i][j] + c * E(i, j)
 67         }
 68     }
 69     lb, eb, countB, err := eignvalue1(n, B, e, x0, 1, crucialCount)
 70     if err != nil {
 71         return -1, nil, -1, err
 72     }
 73     l = lb - c
 74     return l, eb, countB, nil
 75 }
 76
 77 //процедура нахождения собственного числа, ближайшего к данному
 78 //вход: размерность матрицы, матрица, точность, начальное приближение, l0,
    crucialCount
 79 //выход:
 80 func nearestEignvalue(n int, A [][]float64, e float64, x0 []float64, l0 float64,
    crucialCount int)(l float64, count int, err error){
 81     isFirst := true;
 82     var ln1 float64
 83     ln := l0
 84     xn := make([]float64, n)
 85     copy(xn, x0)    //скопировать начальное приближение
 86     var xn1 []float64
 87     count = 0
 88     delta := e + 1
 89     var oldDelta float64
 90     for isFirst || math.Abs(ln - ln1) >= e{
 91         if (delta >= oldDelta && crucialCount == 0 && oldDelta != e + 1) || (count >
    crucialCount) {
 92             return -1, -1, errors.New("Метод расходится")
 93         }
 94         if isFirst {
 95             isFirst = false
 96         } else {
 97             ln = ln1
 98         }
 99         subA := make([][]float64, n)
100         for i := 0; i < n; i++{
101             subA[i] = make([]float64, n) //получить матрицу системы
102             for j := 0; j < n; j++ {
103                 if i == j {
104                     subA[i][j] = A[i][j] - ln
105                 } else {
106                     subA[i][j] = A[i][j]
107                 }
108             }
109             subA[i] = append(subA[i], xn[i]) //добавить столбец b для получения
    системы
110         }
111         xn1, _, _, _, _, err = systemProcessing(subA, n, true) //попытаться решить
    сисmкму
112         if err != nil {
113             return -1, -1, err
114         }
115         ln1 = ln + scalar(n, xn, xn) / scalar(n, xn1, xn)
116         norm := math.Sqrt(scalar(n, xn1, xn1))
117         for i := 0; i < n; i++{ //обновить предшествующий вектор с нормированием
118             xn[i] = xn1[i] / norm
119         }
120         count++
121         oldDelta = delta
122         delta = math.Abs(ln - ln1)
123     }
124     return ln1, count, nil
125 }
126 //процедура нахождения второго по модулю собственного числа матрицы
```

```
127 //вход: размерность матрицы, матрица, точность, x0, собственный вектор max L, период
    обновления, периодн нормирования, crucial count
128 //выход:
129 func eignvalue2(n int, A [][]float64, e float64, x0 []float64, e1 []float64, up int,
    np int, crucialCount int)(l float64, ev []float64, count int, err error){
130     At := make([][]float64, n)
131     for i := 0; i < n; i++{
132         At[i] = make([]float64, n)
133         for j := 0; j < n; j++{
134             At[i][j] = A[j][i] //получить транспонированную матрицу
135         }
136     }
137     _, g1, _, err := eignvalue1(n, At, e, x0, np, crucialCount)
138     if err != nil{
139         return -1, nil, -1, err
140     }
141     y0 := make([]float64, n)    //начальное приближение для запуска алгоритма
142     scalarConst := scalar(n, x0, g1) / scalar(n, e1, g1)
143     for i := 0; i < n; i++{
144         y0[i] = x0[i] - scalarConst * e1[i]
145     }
146     //повторить аналогичный алгоритм
147     xn := make([]float64, n)
148     copy(xn, y0)    //скопировать начальное приближение
149     var xn1 []float64
150     var ln, ln1 float64 //приближения собственных чисел
151     //провести первую итерацию
152     xn1 = Ax(n, A, xn)
153     ln = scalar(n, xn, xn1) / scalar(n, xn, xn)
154     //провести вторую итерацию
155     xn = xn1
156     xn1 = Ax(n, A, xn)
157     ln1 = scalar(n, xn, xn1) / scalar(n, xn, xn)
158     count = 2   //счетчик итерации
159     xn = xn1 //обновить предыдущий вектор
160     delta := math.Abs(ln1 - ln)
161     oldDelta := delta + 1
162     for delta >= e { //основной цикл итерации
163         if (delta >= oldDelta && crucialCount == 0) || (count > crucialCount) {
164             return -1, nil, -1, errors.New("Метод расходится")
165         }
166         xn1 = Ax(n, A, xn)                          //получить следующий вектор
167         ln = ln1                                    //сохранить старое l
168         ln1 = scalar(n, xn, xn1) / scalar(n, xn, xn) //получить новое l
169         count++                                     //увлечить число итераций
170         if count%up == 0 {                          //если достигли периода
    обновления
171             scalarConst = scalar(n, xn1, g1) / scalar(n, e1, g1)
172             for i := 0; i < n; i++ {
173                 xn1[i] -= scalarConst * e1[i] //обновить вектор
174             }
175         }
176         xn = xn1 //обновить предыдущий вектор
177         oldDelta = delta
178         delta = math.Abs(ln1 - ln)
179     }
180     norm := math.Sqrt(scalar(n, xn, xn))
181     for i := 0; i < n; i++{
182         ev = append(ev, xn[i] / norm)   //нормализировать вектор
183     }
184     return ln1, ev, count, nil
185 }
186 //процедура нахождения маскимального по модулю собственного числа матрицы
187 //вход: размерность матрицы, матрица, точность, x0, период нормирования, критическое
    число итераций
188 //выход:
189 func eignvalue1(n int, A [][]float64, e float64, x0 []float64, np int, crucialCount
```

```
189 int) (l float64, ev []float64, count int, err error){
190     xn := make([]float64, n)
191     copy(xn, x0)    //скопировать начальное приближение
192     var xn1 []float64
193     var ln, ln1 float64 //приближения собственных чисел
194     //провести первую итерацию
195     xn1 = Ax(n, A, xn)
196     ln = scalar(n, xn, xn1) / scalar(n, xn, xn)
197     //провести вторую итерацию
198     xn = xn1
199     xn1 = Ax(n, A, xn)
200     ln1 = scalar(n, xn, xn1) / scalar(n, xn, xn)
201     count = 2   //счетчик итерации
202     xn = xn1 //обновить предыдущий вектор
203     delta := math.Abs(ln1 - ln)
204     oldDelta := delta + 1
205     for delta >= e{ //основной цикл итерации
206         if (delta >= oldDelta && crucialCount == 0) || (crucialCount != 0 && count >
    crucialCount){
207             return -1, nil, -1, errors.New("Метод расходится")
208         }
209         xn1 = Ax(n, A, xn)                          //получить следующий вектор
210         ln = ln1                                    //сохранить старое l
211         ln1 = scalar(n, xn, xn1) / scalar(n, xn, xn) //получить новое l
212         count++                                     //увлечить число итераций
213         if count%np == 0 {                          //если достигли периода
    нормализации
214             norm := math.Sqrt(scalar(n, xn1, xn1))
215             for i := 0; i < n; i++ {
216                 xn1[i] /= norm //нормализировать вектор
217             }
218         }
219         xn = xn1 //обновить предыдущий вектор
220         oldDelta = delta
221         delta = math.Abs(ln1 - ln)
222     }
223     norm := math.Sqrt(scalar(n, xn, xn))
224     for i := 0; i < n; i++{
225         ev = append(ev, xn[i] / norm)   //нормализировать вектор
226     }
227     return ln1, ev, count, nil
228 }
229
230 //вспомогательная процедура скалярного произведения
231 func scalar(n int, x1 []float64, x2 []float64) float64{
232     var res float64 = 0
233     for i := 0; i < n; i++{
234         res += x1[i] * x2[i]
235     }
236     return res
237 }
238 //вспомогательная процедура оператора
239 func Ax(n int, A [][]float64, x[]float64)(res []float64){
240     for i := 0; i < n; i++{
241         var sum float64 = 0
242         for j := 0; j < n; j++{
243             sum += A[i][j] * x[j]
244         }
245         res = append(res, sum)
246     }
247     return res
248 }
249
250
251 //ниже копипаст из lr1 для решения системы методом гаусса
252 //
253 //
```

```
254 //
255 //
256 //
257
258 //процедура обработки системы: получает решение системы, велечину невязки,
    определитель матрицы системы и обратную матрицу системы
259 //входные параметры: исходная матрица уравнения Ab, кол-во неизвестных, признак
    использования главного элемента в алгоритме приведения к треугольному виду
260 //результат: решение системы, вектор невязки для решения системы, определитель,
    обратная матрица, невязка для обратной матрицы, информация об ошибке
261 func systemProcessing(A [][]float64, n int, withMajor bool) ([]float64, []float64,
    float64, [][]float64, [][]float64, error){
262     //скопировать матрицу A и вектор b для вычисления невязки обратной матрицы ниже
263     AbE := A //сформируем матрицу вида A|b|E - расширенную марице уравнения
264     A = make([][]float64, n)
265     for i := 0; i < n; i++{
266         A[i] = make([]float64, n)
267         copy(A[i], AbE[i][:n])
268     }
269     b := make([]float64, n)
270     for i := 0; i < n; i++{
271         b[i] = AbE[i][n]
272     }
273     for i := 0; i < n; i++{ //цикл по строкам
274         for j := n + 1; j <= n * 2; j++{ //цикл по столбцам
275             if i == j - (n + 1) { //если элемент на главной диагонали подматрицы E
276                 AbE[i] = append(AbE[i], 1) //то записать 1
277             } else {
278                 AbE[i] = append(AbE[i], 0) //иначе 0
279             }
280         }
281     }
282     AbE, swapStringNumb, err := reductionToTriangular(AbE, n, withMajor) // привести
    всё это дело к треугольному виду
283     if err != nil { //проверка на возникновение ошибок
284         return nil, nil, 0, nil, nil, err
285     }
286
287     solution, err := solve(AbE, n, n)//далее получить решение для системы уравнений
288     if err != nil {
289         return nil, nil, 0, nil, nil, err
290     }
291
292     //определить вектор невязки для полученного решения
293     solutionDelta := make([]float64, n) //вектор невязки
294     for i:= 0; i < n; i++{
295         var sum float64 = 0
296         for j := 0; j < n; j++ {
297             sum += A[i][j] * solution[j]
298         }
299         solutionDelta[i] = b[i] - sum
300     }
301
302     //далее получить решения являющиеся столбцами обратной матрицы и сформировать
    обратную матрицу
303     A_ := make([][]float64, n) //обратная матрица
304     for i := n + 1; i < n + 1 + n; i++ { // цикл по всем столбцам подматрицы E
305         column, err := solve(AbE, n, i) //решить систему и получить очередной столбец
    обратной матрицы
306         if err != nil {
307             return nil, nil, 0, nil, nil, err
308         }
309         for j := 0; j < n; j++{
310             A_[j] = append(A_[j], column[j]) //заполнить столбец обратной матрицы
311         }
312     }
313
```

```
314     //определить невязку для обратной матрицы
315     R := make([][]float64, n) //произведение прямой и обратной матрицы A, затем вектор
     невязки
316     for i := 0; i < n; i++{ //цикл по строкам
317         for j := 0; j < n; j++ { //цикл по столбцам
318             R[i] = append(R[i], 0) //добавить элемент AA_[i][j]
319             for k := 0; k < n; k++ { //цикл умножения iой строки A на jый столбец A_
320                 R[i][j] += A[i][k] * A_[k][j]
321             }
322         }
323     }
324
325     //преобразовать AA_ к вектору невязки (AA_ = E - AA_)
326     for i := 0; i < n; i++ {
327         for j := 0; j < n; j++{
328             if i == j { //если элемент на главной диагонали
329                 R[i][j] = 1 - R[i][j]
330             } else {
331                 R[i][j] = 0 - R[i][j]
332             }
333         }
334     }
335
336
337     detA := det(AbE, n, swapStringNumb) // вычислить определитель матрицы A
338     return solution, solutionDelta, detA, A_, R, nil
339 }
340
341
342 //процедура получения корней системы линейных уравнений по треугольной матрице системы
343 //входные параметры: расширенная матрица системы, кол-во неизвестных, индекс столбца
    вектора правой части системы
344 func solve(AbE [][]float64, n int, bIndex int) ([]float64, error) {
345     res := make([]float64, n) // результат
346     for i := n - 1; i >= 0; i-- { // цикл подъема по матрице вверх для нахождения
    корней
347         var sum float64 = 0.0; // получить отнимаемую сумму
348         for j := i + 1; j < n; j++ {
349             sum += AbE[i][j] * res[j]
350         }
351         res[i] = (AbE[i][bIndex] - sum) / AbE[i][i]
352         if math.IsNaN(res[i]) || math.IsInf(res[i], 1) || math.IsInf(res[i], -1) { //
    проверка на выход за пределы множества машинных чисел
353             return nil, errors.New("Одно из чисел за пределами множества машинных
    чисел, получить решение невозможно. Попробуйте использовать алгоритм с выбором
    главного элемента.")
354         }
355     }
356     return res, nil
357 }
358
359 //процедура нахождения определителя исходной матрицы A по приведенной треугольной
    матрице Ab
360 //входные параметры: приведенная к треугольному виду матрица Ab, число неизвестных,
    число перестановок строк в процессе приведения
361 //результат: определитель матрицы
362 func det(Ab [][]float64, n int, swapStringNumb int) float64 {
363     var det float64 = 1
364     for i := 0; i < n; i++ {
365         det *= Ab[i][i]
366     }
367     if swapStringNumb % 2 == 1{ //если необходимо изменить знак
368         det *= -1
369     }
370     return det
371 }
372 //процедура приведения к треугольному виду матрицы Ab
```

```
373 //треугольный вид имеет подматрица A
374 //входные параметры: матрица уравнения Ab, кол-во неизвестных n, признак использования
     главного элемента в алгоритме
375 //результат: матрица, приведённая к указанному виду, число перестановок строк в
    процессе алгоритма, сообщение об ошибках
376 func reductionToTriangular (Ab [][]float64, n int, withMajor bool) ([][]float64, int
    ,  error){
377     swapStringNumb := 0 // число перестановок строк в процессе алгоритма
378     for i := 0; i < n; i++ { // цикл по столбцам матрицы A
379         if withMajor { // если необходимо ставить на диагональ главный элемент
380             major := i // индекс первичного значения
381             for j := i + 1; j < n; j++ {
382                 if (math.Abs(Ab[major][i]) == 0.0) || ((math.Abs(Ab[j][i]) < math.Abs(
    Ab[major][i])) && math.Abs(Ab[j][i]) != 0.0) { // сравнить по модулю
383                     major = j
384                 }
385             }
386             if math.Abs(Ab[major][i]) == 0.0 { // если матрица вырожденая
387                 return nil, swapStringNumb, errors.New("Вырожденая матрица")
388             }
389             if i != major { //если выполняется перестановка
390                 Ab = swapString(Ab, i, major) // установить главный элемент если
    требуется
391                 swapStringNumb++ //увеличить число перестановок
392             }
393         }
394         for j := i + 1; j < n; j++ { // цикл по строкам для столбца i
395             Cji := Ab[j][i] / Ab[i][i] // строка i умножается на Cji и вычитается из
    строки j
396             if math.IsNaN(Cji) || math.IsInf(Cji, 1) || math.IsInf(Cji, -1) { //
    проверка на выход за пределы множества машинных чисел
397                 return nil, swapStringNumb, errors.New("Одно из чисел за пределами
    множества машинных чисел, получить решение невозможно. Попробуйте использовать
    алгоритм с выбором главного элемента.")
398             }
399             for k := i; k <= n * 2; k++ {  // цикл вычитания строки i из строки j
400                 Ab[j][k] -= Ab[i][k] * Cji
                                                        // вычесть очередной элемент
    строки
401             }
402         }
403     }
404     return Ab, swapStringNumb, nil
405 }
406
407 //вспомогательная процедура перестановки строк i и j в матрице m
408 func swapString(m [][]float64, i int, j int) [][]float64{
409     buf := m[i]
410     m[i] = m[j]
411     m[j] = buf
412     return m
413 }
414 //процедура чтения исодных данных:
415 //считывает расширенную матрицу уравнения из заданного файла
416 func readSourceData(fileName string) ([][]float64, int, error) {
417     file, err := os.Open(fileName) // открыть файл с данными
418     defer file.Close() // закрыть по завершению
419     if err != nil { // если ошибка открытия, то завершения с ошибкой
420         return nil, -1, err
421     }
422     var n int // кол-во неизвестных
423     _, err = fmt.Fscanf(file, "%d\n", &n) //считать кол-во неихвестныь
424     if err != nil { // если ошибка, то завершение
425         return nil, -1, err
426     }
427     var Ab [][]float64 = make([][]float64, n) //матрица n x (n + 1) - расширенная вида
     A|b
```

```
428     //цикл чтения матрицы из файла
429     for i := 0; i < n; i++{
430         Ab[i] = make([]float64, n + 1) // выделить память под строку матрицы
431         for j := 0; j <= n; j++ { // считтаь элементы строки
432             _, err = fmt.Fscanf(file, "%g", &Ab[i][j])
433             if err != nil {
434                 return nil, -1, err
435             }
436         }
437         fmt.Fscanf(file, "\n") //пропустить newline
438     }
439     return Ab, n, nil // вернуть результат
440 }
441
```