Министерство образования и науки Российской Федерации

Новосибирский государственный технический университет

Кафедра прикладной математики

Уравнения математической физики

Лабораторные работы №3,4

Факультет	ПМИ
Группа	ПМ-01
Студент	Жигалов П.С.
Преподаватель	Задорожный А.Г.
	Персова М.Г.
Вариант	9, 3

Новосибирск

2013

1. Цель работы

Разработать программу решения гармонической задачи методом конечных элементов. Провести сравнение прямого и итерационного методов решения получаемой в результате конечноэлементной аппроксимации СЛАУ.

2. Задание

Решить трехмерную гармоническую задачу в декартовых координатах, базисные функции – трилинейные.

3. Анализ

3.1. Постановка задачи

Дано уравнение вида: $-div(\lambda grad(u)) + \chi \frac{\partial^2 u}{\partial t^2} + \sigma \frac{\partial u}{\partial t} = f$, в Ω, с правой частью вида:

$f = f^s \sin(\omega t) + f^c \cos(\omega t)$, не зависящими от времени остальными коэффициентами.

Решить МКЭ на параллелепипедах, базис - трилиннейный.

3.2. Вариационная постановка и конечноэлементная аппроксимация задачи

Решение может быть представлено в виде: $u = u^s \sin(\omega t) + u^c \cos(\omega t)$,

тогда исходное уравнение можно представить в виде системы: $\begin{cases} -div(\lambda grad(u^s)) - \omega^2 \chi u^s - \omega\sigma u^c = f^s \\ -div(\lambda grad(u^c)) - \omega^2 u^c + \omega\sigma u^s = f^c \end{cases}$

Умножим скалярно каждое уравнение системы на пробную функцию v, и применим формулу Грина:

$$\begin{cases} \int\limits_\Omega \left(\lambda grad(u^s) grad(v) - \omega\sigma u^c v - \omega^2 \chi u^s v\right) d\Omega + \int\limits_{S_3} \beta u^s v dS = \int\limits_\Omega f^s v d\Omega + \int\limits_{S_3} \theta^s v dS + \int\limits_{S_3} \beta u_\beta^s v dS \\ \int\limits_\Omega \left(\lambda grad(u^c) grad(v) + \omega\sigma u^s v - \omega^2 \chi u^c v\right) d\Omega + \int\limits_{S_3} \beta u^c v dS = \int\limits_\Omega f^c v d\Omega + \int\limits_{S_3} \theta^c v dS + \int\limits_{S_3} \beta u_\beta^c v dS \end{cases}$$

Следом проведём аппроксимацию: $u^s = \sum\limits_{i=0}^{n-1} q_{2i} \psi_{2i}$, $u^c = \sum\limits_{i=0}^{n-1} q_{2i+1} \psi_{2i+1}$

Подставив выражения в систему, получим СЛАУ из $2n$ уравнений.

Локальная матрица системы, без учёта краевых условий, будет выглядеть как блочная матрица размером 8×8 со следующим видом блока:

$$A_{ij} = \begin{bmatrix} G_{ij} - \omega^2 \overline{\chi} M_{ij} & -\omega \overline{\sigma} M_{ij} \\ \omega \overline{\sigma} M_{ij} & G_{ij} - \omega^2 \overline{\chi} M_{ij} \end{bmatrix}$$

G - матрица жесткости для трилинейных функций на параллелепипедах, с учётом коэффициента λ, M - матрица массы.

Локальная правая часть тоже буде иметь блочную структуру: это будет вектор из 8 двухмерных векторов вида: $b_i = \begin{bmatrix} b_i^s \\ b_i^c \end{bmatrix}$, причём вектора b^s и b^c вычисляются через соответствующие $b^s = Mp^s$ значения правой части:

$b^s = Mp^s \qquad b^c = Mp^c$

Локальные матрицы массы краевых условий будут иметь блочную структуру с видом блоков: $A_{ij}^{S_3} = \beta \begin{bmatrix} M_{ij} & 0 \\ 0 & M_{ij} \end{bmatrix}$,

где M - матрица массы для билиныйных функциях на прямоугольниках, добавки в правую часть будут вычисляться умножением этой матрицы на вектор значений функции u_β или θ.

3.3. Методы решения СЛАУ

LU-факторизация

Формат представления матрицы: профильный, выполняется конвертация из разреженного.

Локально-оптимальная схема

Формат представления матрицы: разреженный, предобуславливание: неполная *LU*-факторизация.

GMRES с LU-предобуславливанием

Формат представления матрицы: разреженный, предобуславливание: неполная *LU*-факторизация.

3.4. GMRES

Пусть используется предобуславливание СЛАУ матрицами S и Q, т.е. решается СЛАУ вида $\left(S^{-1}AQ^{-1}\right)\tilde{x} = S^{-1}b$, где $\tilde{x} = Qx$. Выбирается начальное приближение x^0 и полагается: $\tilde{x}^0 = Qx^0$, $\tilde{r}^0 = S^{-1}(b - Ax^0)$.

Далее выполняются итерации метода, на каждой итерации вычисляются элементы двух вспомогательных матриц V размера $n \times m$ и H размера $(m+1) \times m$, где n – размерность решаемой СЛАУ, а m – глубина метода.

Пусть v^i – i-й столбец матрицы V. Первый столбец формируется по формуле: $v^1 = \tilde{r}^{k-1} / \left\|\tilde{r}^{k-1}\right\|$

Далее для $\mu = 1, 2, ..., m$ выполняется:

$$w = (S^{-1}AQ^{-1}v^\mu)$$

$$H_{\lambda\mu} = v^\lambda w, \qquad \lambda = 1, ..., \mu$$

$$\breve{v}^{\mu+1} = S^{-1}AQ^{-1}v^\mu - \sum_{\lambda=1}^{\mu} H_{\lambda\mu}v^\lambda$$

$$H_{\mu+1,\mu} = \left\|\breve{v}^{\mu+1}\right\|$$

Если $H_{\mu+1,\mu} = 0$, то процесс построения H заканчивается, и за m принимается μ, иначе $v^{\mu+1} = \frac{1}{H_{\mu+1,\mu}}\breve{v}^{\mu+1}$

Новое приближение $\tilde{x}^k$ определяется следующим образом:

Вычисляется вектор $d^k = \left(\left\|\tilde{r}^{k-1}\right\|, 0, ..., 0\right)^T$ размера $m+1$.

Находится вектор параметров z^k: $\left\|d^k - Hz^k\right\| = \min_z \left\|d^k - Hz\right\|$.

После чего вычисляется $\tilde{x}^k = \tilde{x}^{k-1} + Vz^k$.

Новая невязка рассчитывается как $\tilde{r}^k = S^{-1}(b - AQ^{-1}\tilde{x}^k)$.

После окончания итерационного процесса решение исходной СЛАУ определяется как $x = Q^{-1}\tilde{x}$.

4. Исследования и тесты

4.1. Полином первой степени.

Сетка: $[0,1]\times[0,1]\times[0,1]$

Шаг: $h = 0.2$ (число узлов – 216)

Аналитическое решение: $u^s = x + y + z$, $u^c = x - y - z$

Параметры уравнения: $\lambda = 1$, $\sigma = 1$, $\chi = 0.1$, $\omega = 1$

Вид правой части: $f^s = -1.1x + 0.9y + 0.9z$, $f^c = 0.9x + 1.1y + 1.1z$

Параметры решателей: $\varepsilon = 10^{-14}$, $maxiter = 100000$, $m = 3$

	LU	*LOS*	*GMRES*
Отн. погрешность	2.370e-16	4.519e-16	2.461e-15
Отн. погрешность u^s	2.470e-16	4.653e-16	2.318e-15
Отн. погрешность u^c	1.876e-16	3.888e-16	3.003e-15
Число итераций	1	37	8
Время, мс	2.1160000	3.5620000	3.1120000

4.2. Оценка порядка аппроксимации.

Сетка: $[0,4]\times[0,4]\times[0,4]$

Шаг: $h = 0.25$ (число узлов – при h - 4913, при $h/2$ - 35937)

Аналитическое решение: $u^s = 2e^{x+y+z}$, $u^c = 3e^{x-y-z}$

Параметры уравнения: $\lambda = 10^4$, $\sigma = 10^3$, $\chi = 10^{-11}$, $\omega = 10^{-2}$

Вид правой части: $f^s = -6\lambda e^{x+y+z} - 3\omega\sigma e^{x-y-z} - 2\omega^2\chi e^{x+y+z}$, $f^c = -9\lambda e^{x+y+z} - 2\omega\sigma\chi e^{x+y+z} - 3\omega^2 e^{x-y-z}$

Параметры решателей: $\varepsilon = 10^{-14}$, $maxiter = 10000$, $m = 3$

	LU, h	*LU, h/2*	*LOS, h*	*LOS, h/2*	*GMRES, h*	*GMRES, h/2*
Отн. погрешность	1.123e-03	3.285e-04	1.123e-03	3.285e-04	1.123e-03	3.285e-04
Отн. погрешность u^s	1.123e-03	3.285e-04	1.123e-03	3.285e-04	1.123e-03	3.285e-04
Отн. погрешность u^c	7.973e-03	2.348e-03	7.973e-03	2.348e-03	7.973e-03	2.348e-03
Число итераций	1	1	243	765	34	124
Время, мс	2580.1820	245052.17	390.18100	8022.3580	247.60900	5750.2370

$$\log_2\left(\|u^* - u_h\| / \|u^* - u_{h/2}\|\right) \approx 1.77$$

4.3. Исследования для сеток с небольшим количеством узлов.

Сетка: $[0,4]\times[0,4]\times[0,4]$

Шаг: $h = 0.5$ (число узлов – 729)

Аналитическое решение: $u^s = 2e^{x+y+z}$, $u^c = 3e^{x-y-z}$

Вид правой части: $f^s = -6\lambda e^{x+y+z} - 3\omega\sigma e^{x-y-z} - 2\omega^2\chi e^{x+y+z}$, $f^c = -9\lambda e^{x+y+z} - 2\omega\sigma\chi e^{x+y+z} - 3\omega^2 e^{x-y-z}$

Параметры решателей: $\varepsilon = 10^{-14}$, $maxiter = 10000$, $m = 3$

ω	λ	σ	χ	*LU, мс*	*LOS, ит*	*LOS, мс*	*GMRES, ит*	*GMRES, мс*
10^{-2}	$3\cdot10^{2}$	10	10^{-11}	30.410000	87	22.525000	13	16.545000
10^{-2}	$3\cdot10^{2}$	10^{4}	10^{-11}	29.640000	96	24.180000	14	17.191000
10^{-2}	10^{4}	10	10^{-11}	29.701000	91	22.742000	13	16.795000
10^{-2}	10^{4}	10^{4}	10^{-11}	29.609000	92	23.552000	13	16.455000
10^{5}	$3\cdot10^{2}$	10	10^{-11}	1548.4240	10000	170.75200	396	262.04100
10^{5}	$3\cdot10^{2}$	10^{4}	10^{-11}	1549.0610	10000	163.07900	10000	6380.1530
10^{5}	10^{4}	10	10^{-11}	33.114000	1238	200.08300	50	40.279000
10^{5}	10^{4}	10^{4}	10^{-11}	29.726000	10000	1555.6310	10000	6427.9090

4.4. Исследования для сеток с большим количеством узлов.

Сетка: $[0,4]\times[0,4]\times[0,4]$

Шаг: $h_x = 0.2$, $h_y = 0.1$, $h_z = 0.1$ (число узлов – 35301)

Аналитическое решение: $u^s = 2e^{x+y+z}$, $u^c = 3e^{x-y-z}$

Вид правой части: $f^s = -6\lambda e^{x+y+z} - 3\omega\sigma e^{x-y-z} - 2\omega^2\chi e^{x+y+z}$, $f^c = -9\lambda e^{x+y+z} - 2\omega\sigma\chi e^{x+y+z} - 3\omega^2 e^{x-y-z}$

Параметры решателей: $\varepsilon = 10^{-14}$, $maxiter = 10000$, $m = 3$

ω	λ	σ	χ	*LU, мс*	*LOS, ит*	*LOS, мс*	*GMRES, ит*	*GMRES, мс*
10^{-2}	$3\cdot10^{2}$	10	10^{-11}	151831.54	2575	25063.307	207	8845.5030
10^{-2}	$3\cdot10^{2}$	10^{4}	10^{-11}	152323.37	2584	25506.373	208	8904.6220
10^{-2}	10^{4}	10	10^{-11}	152895.02	921	9463.2200	190	8450.9180
10^{-2}	10^{4}	10^{4}	10^{-11}	150848.38	917	9396.1100	172	7441.6760
10^{5}	$3\cdot10^{2}$	10	10^{-11}	152075.70	1439	14304.545	252	10647.963
10^{5}	$3\cdot10^{2}$	10^{4}	10^{-11}	152120.07	10000	98062.427	10000	405763.26
10^{5}	10^{4}	10	10^{-11}	151097.01	1570	15491.485	32	1820.7110
10^{5}	10^{4}	10^{4}	10^{-11}	153620.45	10000	97067.476	10000	365335.06

4.5. Исследования на глубину метода в GMRES.

Сетка: $[0,4]\times[0,4]\times[0,4]$

Шаг: $h = 0.2$ (число узлов – 9261)

Аналитическое решение: $u^s = 2e^{x+y+z}$, $u^c = 3e^{x-y-z}$

Параметры уравнения: $\lambda = 10^4$, $\sigma = 10^3$, $\chi = 10^{-11}$, $\omega = 10^{-2}$

Вид правой части: $f^s = -6\lambda e^{x+y+z} - 3\omega\sigma e^{x-y-z} - 2\omega^2\chi e^{x+y+z}$, $f^c = -9\lambda e^{x+y+z} - 2\omega\sigma\chi e^{x+y+z} - 3\omega^2 e^{x-y-z}$

Параметры решателей: $\varepsilon = 10^{-14}$, $maxiter = 10000$

	m = 3	*m = 5*	*m = 10*	*m = 15*	*m = 20*	*m = 30*
Отн. погрешность	7.636e-04	7.636e-04	7.636e-04	7.636e-04	7.636e-04	7.636e-04
Отн. погрешность u^s	7.636e-04	7.636e-04	7.636e-04	7.636e-04	7.636e-04	7.636e-04
Отн. погрешность u^c	5.438e-03	5.438e-03	5.438e-03	5.438e-03	5.438e-03	5.438e-03
Число итераций	46	10	9	6	4	3
Время, мс	600.18300	467.62600	427.60800	453.67300	453.27400	546.26200

5. Выводы

При увеличении ω, σ и χ сходимость итерационных методов ухудшается, а при увеличении λ улучшается. Возможно, это связанно с тем, что при увеличении указанных параметров исходный дифференциальный оператор теряет свойство положительной определённости, следовательно и дискретный аналог (матрица системы) тоже теряет это свойство. Среднее время решения с помощью LU - факторизации от параметров системы не зависит, только её размерности и в отличие от *LOS* и *GMRES* сходится всегда, хотя и гораздо медленнее. *GMRES* решается наиболее быстро, особенно на задачах большой размерности, при этом не теряя в сходимости относительно *LOS*, а манипулируя параметром глубины, можно еще больше выиграть во времени.

6. Код программы (основные модули)

harm_fem.f95

```
#if __GFORTRAN__ == 1 && __GNUC__ == 4 && __GNUC_MINOR__ < 5
#define class type
#endif
#define SOLVERTYPE gmres
#define SOLVERMODULE solver_gmres
    module harm_fem_
        use addition_classes
        use SOLVERMODULE
        use timer_module
        implicit none

        type :: harm_fem
            integer, private :: elements_n  ! Количество узлов
            integer, private :: nodes_n     ! Количество
элементов
            integer, private :: faces_fir_n, faces_sec_n, fac-
es_thi_n ! Количество граней с 1, 2 и 3 краевыми соответственно
            type(node), private, allocatable :: nodes(:)    !
Массив узлов
            type(fe), private, allocatable :: elements(:)   !
Массив элементов
            type(face), private, allocatable :: faces_sec(:),
faces_thi(:) ! Массивы для краевых граней
            double precision, private, allocatable :: fac-
es_fir(:)  ! Значения первых краевых
            integer, private, allocatable :: faces_fir_node(:)
! Узлы первых краевых
            double precision, private, allocatable :: betta(:)
            type(SOLVERTYPE), private :: solver      ! Решатель
СЛАУ
            integer, private :: solver_iters         ! количе-
ство итераций за которое было решено СЛАУ
            type(slae_port_gen), private :: port_gen !
Генератор портрета СЛАУ
            double precision, private :: time        ! Время
решения СЛАУ
            ! Массивы, для хранения СЛАУ
            integer, pointer, private :: ig(:), jg(:)
            integer, private :: slae_el_n ! Количество элемен-
тов в СЛАУ(элементов в jg, gl, gu)
            double precision, pointer, private :: gl(:), gu(:),
di(:)
            double precision, pointer, private :: right_part(:)
! правая часть
            double precision, pointer, private :: solution(:)
! Решение
        contains
            procedure :: transf_grid    ! преобразовывает сетку
в файлах
            procedure :: init_hf        ! Ввод данных
            procedure :: form_matrix    ! Формирование матрицы
СЛАУ
            procedure :: solve_         ! Решение СЛАУ
            procedure :: out_rez        ! Вывод результата в
файл file_name
            procedure :: out_diff       ! Вывод погрешности в
файл
            procedure, private :: u_betta_s     ! Вычисление
краевого условия третьего рода, для синуса
            procedure, private :: u_betta_c     ! Вычисление
краевого условия третьего рода, для косинуса
            procedure, private :: tetta_s       ! Вычисление
краевого условия второго рода, для синуса
            procedure, private :: tetta_c       ! Вычисление
краевого условия второго рода, для косинуса
            procedure, private :: lambda        ! коэффициенты
уравнения
            procedure, private :: sigma
            procedure, private :: hi
            procedure, private :: f_sin         ! правая часть
уравнения
            procedure, private :: f_cos
            procedure, private :: w             ! Частота
колебаний
            procedure, public :: u_sin
            procedure, public :: u_cos
            procedure, private :: form_gmass    ! Формирование
массивов ig и jg
            procedure, private :: find_el_pos   ! Определяет
положение элемента в матрице
            procedure, private :: form_loc      ! Формирование
локальной матрицы для элемента el_n
            procedure, private :: gen_loc_thi   ! Формирование
локальной матрицы и вектора для третих краевых, для грани
face_n
            procedure, private :: gen_loc_sec   ! Формирование
локального вектора для вторых краевых, для грани face_n
            ! Вычисление специальных индексов
            procedure, private :: mu
            procedure, private :: nu
            procedure, private :: v
        end type

    contains

        function w(this)
            implicit none
            class(harm_fem) :: this
            double precision :: w
            w = 1d-2
        end function

        function lambda(this, x, y, z)
            implicit none
            class(harm_fem) :: this
            double precision :: x, y, z, lambda
            lambda = 1d4
        end function

        function sigma(this, x, y, z)
            implicit none
            class(harm_fem) :: this
            double precision :: x, y, z, sigma
            sigma = 1d4
        end function

        function hi(this, x, y, z)
            implicit none
            class(harm_fem) :: this
            double precision :: x, y, z, hi
            hi = 1d-11
        end function

        function u_sin(this, x, y, z)
            implicit none
            class(harm_fem) :: this
            double precision :: x, y, z, u_sin
            u_sin = 2d0*exp(x+y+z)
        end function

        function u_cos(this, x, y, z)
            implicit none
            class(harm_fem) :: this
            double precision :: x, y, z, u_cos
            u_cos = 3d0*exp(x-y-z)
        end function

        function f_sin(this, x, y, z)
            implicit none
            class(harm_fem) :: this
            double precision :: x, y, z, f_sin
            f_sin = - this%lambda(x,y,z) * 3d0 *
this%u_sin(x,y,z) &
                    - this%w()**2 * this%hi(x,y,z) *
this%u_sin(x,y,z) &
                    - this%w() * this%sigma(x,y,z) *
this%u_cos(x,y,z)
        end function

        function f_cos(this, x, y, z)
            implicit none
            class(harm_fem) :: this
            double precision :: x, y, z, f_cos
            f_cos = - this%lambda(x,y,z) * 3d0 *
this%u_cos(x,y,z) &
                    - this%w()**2 * this%hi(x,y,z) *
this%u_cos(x,y,z) &
                    + this%w() * this%sigma(x,y,z) *
this%u_sin(x,y,z)
        end function

        function u_betta_s(this, x, y, z, face_n)
            implicit none
            class(harm_fem) :: this
            double precision :: x, y, z, u_betta_s
            integer :: face_n
            u_betta_s = 0d0
        end function
```

```
        function u_betta_c(this, x, y, z, face_n)
            implicit none
            class(harm_fem) :: this
            double precision :: x, y, z, u_betta_c
            integer :: face_n
            u_betta_c = 0d0
        end function

        function tetta_s(this, x, y, z, face_n)
            implicit none
            class(harm_fem) :: this
            double precision :: x, y, z, tetta_s
            integer :: face_n
            tetta_s = 0d0
        end function

        function tetta_c(this, x, y, z, face_n)
            implicit none
            class(harm_fem) :: this
            double precision :: x, y, z, tetta_c
            integer :: face_n
            tetta_c = 0d0
        end function

        subroutine init_hf(this, file_cords, file_elements,
file_faces)
            implicit none
            class(harm_fem) :: this
            character(len=255) :: file_cords, file_elements,
file_faces
            integer :: i, j
            open(10, file=file_cords, status='old')
            ! Ввод координат вершин
            read(10,*) this%nodes_n
            allocate(this%nodes(this%nodes_n))
            do i = 1, this%nodes_n
                read(10,*) this%nodes(i)%x, this%nodes(i)%y,
this%nodes(i)%z
            end do
            close(10)
            open(10, file=file_elements, status='old')
            ! Ввод элементов
            read(10,*) this%elements_n
            allocate(this%elements(this%elements_n))
            do i = 1, this%elements_n, 1
                do j = 1, 16, 1
                    read(10,*) this%elements(i)%node_n(j)
                end do
            end do
            close(10)
            open(10, file=file_faces, status='old')
            ! Ввод первых краевых
            read(10,*) this%faces_fir_n
            allocate(this%faces_fir(this%faces_fir_n))
            allocate(this%faces_fir_node(this%faces_fir_n))
            do i = 1, this%faces_fir_n, 1
                read(10,*) this%faces_fir_node(i),
this%faces_fir(i)
            end do
            ! Ввод вторых краевых
            read(10,*) this%faces_sec_n
            allocate(this%faces_sec(this%faces_sec_n))
            do i = 1, this%faces_fir_n, 1
                do j = 1, 8, 1
                    read(10,*) this%faces_sec(i)%node_n(j)
                end do
            end do
            ! Ввод третьих краевых
            read(10,*) this%faces_thi_n
            allocate(this%faces_sec(this%faces_thi_n))
            do i = 1, this%faces_thi_n, 1
                do j = 1, 8, 1
                    read(10,*) this%faces_thi(i)%node_n(j)
                end do
            end do
            close(10)
        end subroutine

        subroutine out_diff(this, file_name)
            implicit none
            class(harm_fem) :: this
            character*255 :: file_name
            double precision :: diff1 = 0d0 ! погрешность
            double precision :: u_norm = 0d0
            double precision :: diff_s = 0d0, diff_c = 0d0,
u_ns = 0d0, u_nc = 0d0
            double precision :: x, y, z, us, uc
            integer :: i
            open(10, file=file_name, status='unknown')
            do i = 1, this%nodes_n / 2, 1
                x = this%nodes(2*i)%x
                y = this%nodes(2*i)%y
                z = this%nodes(2*i)%z
                us = this%u_sin(x,y,z)
                uc = this%u_cos(x,y,z)
                diff_s = diff_s + (us - this%solution(2*i))*(us
- this%solution(2*i))
                diff_c = diff_c + (uc -
this%solution(2*i+1))*(uc - this%solution(2*i+1))
                u_ns = u_ns + us**2
                u_nc = u_nc + uc**2
            end do
            diff1 = diff_s + diff_c
            u_norm = u_ns + u_nc
            write(10, fmt='( a8e10.3 )'), 'Total:',
dsqrt(diff1/u_norm)
            write(10, fmt='( a8e10.3 )'), 'Sin:',
dsqrt(diff_s/u_ns)
            write(10, fmt='( a8e10.3 )'), 'Cos:',
dsqrt(diff_c/u_nc)
            write(10, fmt='( a8i10 )'), 'Iters:',
this%solver_iters
            write(10, fmt='( a8e10.3 )'), 'Time:', this%time
            close(10)
        end subroutine

        subroutine solve_(this)
            implicit none
            class(harm_fem) :: this
            class(timer_) :: t
            ! Начало замера времени
            call t%start_
            call this%solver%init(this%ig, this%jg, this%gu,
this%gl, this%di, this%nodes_n)
            call this%solver%set_rp(this%right_part)
            call this%solver%solve(this%solution,
this%solver_iters)
            ! Конец замера времени
            this%time = t%stop_()
        end subroutine

        subroutine form_gmass(this)
            implicit none
            class(harm_fem) :: this
            integer :: i
            call this%port_gen%init_g(this%nodes_n)
            ! Все собираем
            allocate(this%ig(this%nodes_n + 1))
            do i = 1, this%elements_n, 1
                call this%port_gen%add_el_g(this%elements(i))
            end do
            call this%port_gen%gen(this%ig, this%jg,
this%slae_el_n) ! получаем портрет
            allocate(this%gl(this%slae_el_n))
            allocate(this%gu(this%slae_el_n))
            allocate(this%di(this%nodes_n))
            allocate(this%right_part(this%nodes_n))
            allocate(this%solution(this%nodes_n))
            ! Обнуление
            this%gl = 0d0
            this%gu = 0d0
            this%di = 0d0
            this%right_part = 0d0
            this%solution = 0d0
            call this%port_gen%destruct_g()
        end subroutine

        subroutine form_matrix(this)
            implicit none
            class(harm_fem) :: this
            double precision :: a_loc(16,16), b_loc(16) ! ло-
кальные матрица и вектор правой части
            integer :: cur_row ! текущая строка
            integer :: pos ! Позиция в gu и gl
            integer :: k, i, j, i_s, i_e
            double precision :: val
            call this%form_gmass() ! формируем массивы
            ! Генерация основной СЛАУ
            do k = 1, this%elements_n, 1
                call this%form_loc(a_loc, b_loc, k) ! Получаем
локальные матрицы
                do i = 1, 16, 1
                    cur_row = this%elements(k)%node_n(i) !
Определяем элемент строки
                    do j = 1, i - 1, 1
                        if(cur_row .gt.
this%elements(k)%node_n(j)) then ! Если элементы содержаться в
строке
                            pos = this%find_el_pos(cur_row,
this%elements(k)%node_n(j)) ! Находим позицию в gu и gl
                            this%gl(pos) = this%gl(pos) +
a_loc(i, j)
                            this%gu(pos) = this%gu(pos) +
a_loc(j, i)
                        else
                            pos =
this%find_el_pos(this%elements(k)%node_n(j), cur_row) ! Находим
позицию в gu и gl
                            this%gl(pos) = this%gl(pos) +
a_loc(i, j)
                            this%gu(pos) = this%gu(pos) +
a_loc(j, i)
                        end if
                    end do
                    this%di(cur_row) = this%di(cur_row) +
a_loc(i, i)
                    this%right_part(cur_row) =
this%right_part(cur_row) + b_loc(i)
                end do
            end do
            ! Учёт третьих краевых условий
            do k = 1, this%faces_thi_n, 1
                call this%gen_loc_thi(a_loc, b_loc, k)
                do i = 1, 8, 1
                    cur_row = this%faces_thi(k)%node_n(i)
                    do j = 1, i-1, 1
                        if(cur_row .gt.
this%faces_thi(k)%node_n(j)) then ! Если элементы содержаться в
строке
                            pos = this%find_el_pos(cur_row,
this%faces_thi(k)%node_n(j)) ! Находим позицию в gu и gl
                            this%gl(pos) = this%gl(pos) +
a_loc(i, j)
                            this%gu(pos) = this%gu(pos) +
a_loc(j, i)
                        else
```

```
                            pos =
this%find_el_pos(this%faces_thi(k)%node_n(j), cur_row) !
Находим позицию в gu и gl
                            this%gl(pos) = this%gl(pos) +
a_loc(i, j)
                            this%gu(pos) = this%gu(pos) +
a_loc(j, i)
                        end if
                    end do
                    this%di(cur_row) = this%di(cur_row) +
a_loc(i, i)
                    this%right_part(cur_row) =
this%right_part(cur_row) + b_loc(i)
                end do
            end do
            ! Учёт вторых краевых условий
            do k = 1, this%faces_sec_n, 1
                call this%gen_loc_sec(b_loc, k)
                do i = 1, 8, 1
                    this%right_part(this%faces_sec(k)%node_n) =
&

this%right_part(this%faces_sec(k)%node_n) + b_loc(i)
                end do
            end do
            ! Учёт первых краевых условий
            do k = 1, this%faces_fir_n, 1
                cur_row = this%faces_fir_node(k) ! Узел, в
котором заданно краевое
                val = this%faces_fir(k) ! Получаем значение
                this%di(cur_row) = 1d0
                this%right_part(cur_row) = val
                ! Обнуляем верхную часть столбца
                i_s = this%ig(cur_row)
                i_e = this%ig(cur_row + 1)
                do i = i_s, i_e - 1, 1
                    this%right_part(this%jg(i)) = &
                        this%right_part(this%jg(i)) -
this%gu(i) * val
                    this%gl(i) = 0d0
                    this%gu(i) = 0d0
                end do
                ! обнуляем нижную часть столбца
                do j = cur_row + 1, this%nodes_n, 1
                    i_s = this%ig(j)
                    i_e = this%ig(j + 1)
                    do i = i_s, i_e - 1, 1
                        if(this%jg(i) .eq. cur_row) then
                            this%right_part(j) = &
                                this%right_part(j) - this%gl(i)
* val
                            this%gl(i) = 0d0
                            this%gu(i) = 0d0
                        end if
                    end do
                end do
            end do
        end subroutine

        subroutine form_loc(this, a_loc, b_loc, el_n)
            implicit none
            class(harm_fem) :: this
            double precision :: a_loc(16,16), b_loc(16)
            integer :: el_n
            ! Матрицы жескости и массы
            double precision :: g(8,8), m(8,8)
            double precision :: hx, hy, hz
            double precision :: g1(2,2)=reshape(source=(/1d0,-
1d0,-1d0,1d0/),shape=(/2,2/)) ! Дополнительная матрица
            double precision ::
m1(2,2)=reshape(source=(/1d0/3d0,1d0/6d0,1d0/6d0,1d0/3d0/),shap
e=(/2,2/)) ! Ещё одна дополнительная матрица
            double precision :: lambda_aver = 0
            double precision :: sigma_aver = 0
            double precision :: hi_aver = 0
            double precision :: val_f_sin(8), val_f_cos(8),
b_sin(8), b_cos(8)
            double precision :: vec2(16), vals(16), diff(16)
            integer :: i, j
            double precision :: x, y, z
            hx =
dabs(this%nodes(this%elements(el_n)%node_n(3))%x - &

this%nodes(this%elements(el_n)%node_n(0))%x)
            hy =
dabs(this%nodes(this%elements(el_n)%node_n(5))%y - &

this%nodes(this%elements(el_n)%node_n(0))%y)
            hz =
dabs(this%nodes(this%elements(el_n)%node_n(9))%z - &

this%nodes(this%elements(el_n)%node_n(0))%z)
            do i = 1, 8, 1
                x =
this%nodes(this%elements(el_n)%node_n(2*i))%x
                y =
this%nodes(this%elements(el_n)%node_n(2*i))%y
                z =
this%nodes(this%elements(el_n)%node_n(2*i))%z
                lambda_aver = lambda_aver + this%lambda(x, y,
z)
                sigma_aver = sigma_aver + this%sigma(x, y, z)
                hi_aver = hi_aver + this%hi(x, y, z)
            end do
            lambda_aver = lambda_aver / 8d0
            sigma_aver = sigma_aver / 8d0
            hi_aver = hi_aver / 8d0
            ! Получение матриц G и M
            do i = 1, 8, 1
                do j = 1, 8, 1
                    g(i, j) = hy * hz * g1(this%mu(i),
this%mu(j)) * m1(this%nu(i), this%nu(j)) * m1(this%v(i),
this%v(j)) / hx + &
                                hx * hz * m1(this%mu(i),
this%mu(j)) * g1(this%nu(i), this%nu(j)) * m1(this%v(i),
this%v(j)) / hy
                    g(i, j) = g(i, j) + &
                                hx * hy * m1(this%mu(i),
this%mu(j)) * m1(this%nu(i), this%nu(j)) * g1(this%v(i),
this%v(j)) / hz
                    g(i, j) = g(i, j) * lambda_aver
                    m(i, j) = hx*hy*hz * m1(this%mu(i), this%mu(j)) *
m1(this%nu(i), this%nu(j)) * m1(this%v(i), this%v(j))
                end do
            end do
            ! Собираем локальную матрицу
            do i = 1, 8, 1
                do j = 1, 8, 1
                    a_loc(2 * i + 1, 2 * j + 1) = g(i, j) -
this%w()**2 * hi_aver * m(i, j)
                    a_loc(2 * i, 2 * j) = g(i, j) - this%w()**2
* hi_aver * m(i, j)
                    a_loc(2 * i, 2 * j + 1) = - this%w() *
sigma_aver * m(i, j)
                    a_loc(2 * i + 1, 2 * j) = this%w() * sig-
ma_aver * m(i, j)
                end do
            end do
            ! Вычисляем значения
            do i = 1, 8, 1
                x = this%nodes(this%elements(el_n)%node_n(2 *
i))%x
                y = this%nodes(this%elements(el_n)%node_n(2 *
i))%y
                z = this%nodes(this%elements(el_n)%node_n(2 *
i))%z
                val_f_sin(i) = this%f_sin(x, y, z)
                val_f_cos(i) = this%f_cos(x, y, z)
            end do
            ! Вычисляем подвекторы правой части
            do i = 1, 8, 1
                b_sin(i) = 0d0
                b_cos(i) = 0d0
                do j = 1, 8, 1
                    b_sin(i) = b_sin(i) + m(i, j) *
val_f_sin(j)
                    b_cos(i) = b_cos(i) + m(i, j) *
val_f_cos(j)
                end do
            end do
            ! Соединяем два вектора в один
            do i = 1, 8, 1
                b_loc(2 * i) = b_sin(i)
                b_loc(2 * i + 1) = b_cos(i)
            end do
            do i = 1, 8, 1
                x = this%nodes(this%elements(el_n)%node_n(2 *
i))%x
                y = this%nodes(this%elements(el_n)%node_n(2 *
i))%y
                z = this%nodes(this%elements(el_n)%node_n(2 *
i))%z
                vals(2 * i) = this%u_sin(x, y, z)
                vals(2 * i + 1) = this%u_cos(x, y, z)
            end do
            do i = 1, 16, 1
                vec2(i) = 0
                do j = 1, 16, 1
                    vec2(i) = vec2(i) + a_loc(i, j) * vals(j)
                end do
            end do
            do i = 1, 16, 1
                diff(i) = b_loc(i) - vec2(i)
            end do
        end subroutine

        function mu(this, i)
            implicit none
            class(harm_fem) :: this
            integer :: i, mu
            mu = mod(i, 2)
        end function

        function nu(this, i)
            implicit none
            class(harm_fem) :: this
            integer :: i, nu
            nu = mod(i / 2, 2)
        end function

        function v(this, i)
            implicit none
            class(harm_fem) :: this
            integer :: i, v
            v = mod(i / 4, 2)
        end function

        subroutine gen_loc_thi(this, a_loc, b_loc, face_n)
            implicit none
            class(harm_fem) :: this
            double precision :: a_loc(16,16), b_loc(16)
            integer :: face_n, i, j
            double precision :: hx, hy, loc_betta, x, y, z
            ! Матрица массы
            double precision :: m1(4,4) = reshape(source=(/&
                4d0, 2d0, 2d0, 1d0, &
                2d0, 4d0, 1d0, 2d0, &
                2d0, 1d0, 4d0, 2d0, &
```

```
                1d0, 2d0, 2d0, 4d0/),shape=(/4,4/))
            double precision :: b_both(8) ! Вектор значений
            ! Определяем оринтацию грани
            if(this%nodes(this%faces_thi(face_n)%node_n(3))%x
.eq. &
                this%nodes(this%faces_thi(face_n)%node_n(1))%x)
then ! Если в плоскости yOz
                hx =
dabs(this%nodes(this%faces_thi(face_n)%node_n(3))%y - &

this%nodes(this%faces_thi(face_n)%node_n(1))%y)
                hy =
dabs(this%nodes(this%faces_thi(face_n)%node_n(5))%z - &

this%nodes(this%faces_thi(face_n)%node_n(1))%z)
            else

if(this%nodes(this%faces_thi(face_n)%node_n(3))%y .eq. &

this%nodes(this%faces_thi(face_n)%node_n(1))%y) then ! Если в
плоскости xOz
                    hx =
dabs(this%nodes(this%faces_thi(face_n)%node_n(3))%x - &

this%nodes(this%faces_thi(face_n)%node_n(1))%x)
                    hy =
dabs(this%nodes(this%faces_thi(face_n)%node_n(5))%z - &

this%nodes(this%faces_thi(face_n)%node_n(1))%z)
                else ! Если в плоскости xOy
                    hx =
dabs(this%nodes(this%faces_thi(face_n)%node_n(3))%x - &

this%nodes(this%faces_thi(face_n)%node_n(1))%x)
                    hy =
dabs(this%nodes(this%faces_thi(face_n)%node_n(5))%y - &

this%nodes(this%faces_thi(face_n)%node_n(1))%y)
                end if
            end if
            loc_betta = this%betta(this%faces_thi(face_n)%area)
            ! Формируем матрицу массы
            do i = 1, 4, 1
                do j = 1, 4, 1
                    a_loc(2*i, 2*i) = loc_betta * hx * hy *
m1(i, i) / 36d0
                    a_loc(2*j+1, 2*j+1) = loc_betta * hx * hy *
m1(i, i) / 36d0
                    a_loc(2*i+1, 2*j) = 0d0
                    a_loc(2*i, 2*j+1) = 0d0
                end do
            end do
            ! Формируем правую часть
            do i = 1, 4, 1
                x = this%nodes(this%elements(face_n)%node_n(2 *
i))%x
                y = this%nodes(this%elements(face_n)%node_n(2 *
i))%y
                z = this%nodes(this%elements(face_n)%node_n(2 *
i))%z
                b_both(2*i) = this%u_betta_s(x, y, z, face_n)
                b_both(2*i+1) = this%u_betta_c(x, y, z, face_n)
            end do
            ! Получаем правую часть
            do i = 1, 8, 1
                b_loc(i) = 0d0
                do j = 1, 8, 1
                    b_loc(i) = b_loc(i) + a_loc(i, j) *
b_both(j)
                end do
            end do
        end subroutine

        subroutine gen_loc_sec(this, b_loc, face_n)
            implicit none
            class(harm_fem) :: this
            double precision :: b_loc(16), a_loc(8, 8)
            integer :: face_n, i, j
            double precision :: hx, hy, x, y, z
            double precision :: b_both(8) ! Вектор значений
            ! Матрица массы
            double precision :: m1(4,4) = reshape(source=(/&
                4d0, 2d0, 2d0, 1d0, &
                2d0, 4d0, 1d0, 2d0, &
                2d0, 1d0, 4d0, 2d0, &
                1d0, 2d0, 2d0, 4d0/),shape=(/4,4/))
            ! Определяем оринтацию грани
            if(this%nodes(this%faces_thi(face_n)%node_n(3))%x
.eq. &
                this%nodes(this%faces_thi(face_n)%node_n(1))%x)
then ! Если в плоскости yOz
                hx =
dabs(this%nodes(this%faces_thi(face_n)%node_n(3))%y - &

this%nodes(this%faces_thi(face_n)%node_n(1))%y)
                hy =
dabs(this%nodes(this%faces_thi(face_n)%node_n(5))%z - &

this%nodes(this%faces_thi(face_n)%node_n(1))%z)
            else

if(this%nodes(this%faces_thi(face_n)%node_n(3))%y .eq. &

this%nodes(this%faces_thi(face_n)%node_n(1))%y) then ! Если в
плоскости xOz
                    hx =
dabs(this%nodes(this%faces_thi(face_n)%node_n(3))%x - &

this%nodes(this%faces_thi(face_n)%node_n(1))%x)
                    hy =
dabs(this%nodes(this%faces_thi(face_n)%node_n(5))%z - &

this%nodes(this%faces_thi(face_n)%node_n(1))%z)
                else ! Если в плоскости xOy
                    hx =
dabs(this%nodes(this%faces_thi(face_n)%node_n(3))%x - &

this%nodes(this%faces_thi(face_n)%node_n(1))%x)
                    hy =
dabs(this%nodes(this%faces_thi(face_n)%node_n(5))%y - &

this%nodes(this%faces_thi(face_n)%node_n(1))%y)
                end if
            end if
            ! Формируем матрицу массы
            do i = 1, 4, 1
                do j = 1, 4, 1
                    a_loc(2*i, 2*i) = hx * hy * m1(i, i) / 36d0
                    a_loc(2*j+1, 2*j+1) = hx * hy * m1(i, i) /
36d0
                    a_loc(2*i+1, 2*j) = 0d0
                    a_loc(2*i, 2*j+1) = 0d0
                end do
            end do
            ! Формируем правую часть
            do i = 1, 4, 1
                x = this%nodes(this%elements(face_n)%node_n(2 *
i))%x
                y = this%nodes(this%elements(face_n)%node_n(2 *
i))%y
                z = this%nodes(this%elements(face_n)%node_n(2 *
i))%z
                b_both(2*i) = this%tetta_s(x, y, z, face_n)
                b_both(2*i+1) = this%tetta_c(x, y, z, face_n)
            end do
            ! Получаем правую часть
            do i = 1, 8, 1
                b_loc(i) = 0d0
                do j = 1, 8, 1
                    b_loc(i) = b_loc(i) + a_loc(i, j) *
b_both(j)
                end do
            end do
        end subroutine

        function find_el_pos(this, i, j)
            implicit none
            class(harm_fem) :: this
            integer :: i, j, find_el_pos, k_s, k_e, k
            logical :: find = .false.
            k_s = this%ig(i)
            k_e = this%ig(i+1)
            k = k_s
            do while(k .lt. k_e .and. .not. find)
                if(this%jg(k) .eq. j) then
                    find_el_pos = k
                    find = .true.
                end if
                k = k + 1
            end do
        end function

        subroutine out_rez(this, file_name)
            implicit none
            class(harm_fem) :: this
            character*255 :: file_name
            integer :: i
            open(10, file=file_name, status='unknown')
            do i = 1, this%nodes_n, 1
                write(10, fmt='( i10e27.16 )'), i,
this%solution(i)
            end do
            close(10)
        end subroutine
```

additional_classes.f95

```
#if __GFORTRAN__ == 1 && __GNUC__ == 4 && __GNUC_MINOR__ < 5
#define class type
#endif
    module addition_classes
        implicit none

        ! Узел
        type :: node
            double precision :: x, y, z
            integer :: number
        end type

        ! Двойной куб
        type :: fe
            integer :: node_n(16)
            integer :: number
            integer :: area
        end type

        ! Грань
        type :: face
            integer :: node_n(8)
            integer :: number
            integer :: area
        end type

        ! элемент списка для геренации портрета СЛАУ
        type :: gen_l_el
            integer :: value
            type(gen_l_el), pointer :: next=>null()
```

```
        end type

        type :: slae_port_list
            type(gen_l_el), pointer, private :: begin => null()
            type(gen_l_el), pointer, private :: end => null()
            type(gen_l_el), pointer, private :: cash => null()
            integer, private :: l_size = 0, m_size, num-
ber_of_line
        contains
            procedure :: destruct_l
            procedure :: add_el_l ! Добавление элемента
            procedure :: init_l
            procedure :: set_num ! Установить номер линии
            procedure :: size_before
            procedure :: take_and_next
            procedure :: cash_off
            procedure :: get_m_size
            procedure, private :: exclude_last_el
            procedure, private :: add_l
        end type

        type :: slae_port_gen
            integer, private :: n
            type(slae_port_list), allocatable :: lists(:)
        contains
            procedure :: destruct_g
            procedure :: init_g
            procedure :: add_el_g
            procedure :: gen
        end type

    contains

        subroutine init_l(this)
            implicit none
            class(slae_port_list) :: this
            nullify(this%begin)
            nullify(this%end)
            nullify(this%cash)
            this%l_size = 0
        end subroutine

        subroutine set_num(this, s_num)
            implicit none
            class(slae_port_list) :: this
            integer :: s_num
            this%number_of_line = s_num
        end subroutine

        subroutine destruct_l(this)
            implicit none
            class(slae_port_list) :: this
            nullify(this%cash)
            do while(associated(this%begin))
                call this%exclude_last_el()
            end do
        end subroutine

        subroutine add_el_l(this, el_a)
            implicit none
            class(slae_port_list) :: this
            class(fe) :: el_a
            integer :: i
            do i = 1, 16
                call this%add_l(el_a%node_n(i))
            end do
        end subroutine

        subroutine cash_off(this)
            implicit none
            class(slae_port_list) :: this
            this%cash = this%begin
        end subroutine

        function take_and_next(this)
            implicit none
            class(slae_port_list) :: this
            integer :: take_and_next
            take_and_next = this%cash%value
            this%cash => this%cash%next
        end function

        function get_m_size(this)
            implicit none
            class(slae_port_list) :: this
            integer :: get_m_size
            get_m_size = this%m_size
        end function

        subroutine add_l(this, val)
            implicit none
            class(slae_port_list) :: this
            integer :: val
            class(gen_l_el), pointer :: add_el
            if(val .le. this%number_of_line) then
                allocate(add_el)
                add_el%value = val
                if(.not.associated(this%begin)) then
                    this%begin => add_el
                    nullify(this%begin%next)
                    this%end => this%begin
                    this%cash => this%begin
                else
                    if(val .lt. this%begin%value) then
                        add_el%next => this%begin
                        this%begin => add_el
                        this%cash => this%begin
                    else
                        if(val .gt. this%end%value) then
                            nullify(add_el%next)
                            this%end%next => add_el
                            this%end => this%end%next
                        else
                            this%cash => this%begin
                            do while(associated(this%cash%next)
.and. val .gt. this%cash%next%value)
                                this%cash => this%cash%next
                            end do
                            if(associated(this%cash%next) .and.
this%cash%next%value .ne. val .and. this%cash%value .ne. val)
then
                                add_el%next => this%cash%next
                                this%cash%next => add_el
                            end if
                        end if
                    end if
                end if
                this%l_size = this%l_size +1
                nullify(add_el)
            end if
        end subroutine

        subroutine exclude_last_el(this)
            implicit none
            class(slae_port_list) :: this
            if(associated(this%begin, this%end)) then
                this%cash => this%begin
                do while (.not.associated(this%cash%next,
this%end))
                    this%cash => this%cash%next
                end do
                deallocate(this%cash%next)
                nullify(this%cash%next)
                this%end => this%cash
            else
                deallocate(this%begin)
                nullify(this%begin)
                nullify(this%end)
                nullify(this%cash)
            end if
        end subroutine

        function size_before(this, n)
            implicit none
            class(slae_port_list) :: this
            integer :: size_before, n, tmp_s = 0
            this%cash => this%begin
            if(associated(this%begin)) then
                if(this%begin%value .lt. n) tmp_s = tmp_s + 1
                do while (associated(this%cash%next) .and.
this%cash%next%value .lt. n)
                    tmp_s = tmp_s + 1
                    this%cash => this%cash%next
                end do
            end if
            this%m_size = tmp_s
            size_before = tmp_s
        end function

        subroutine destruct_g(this)
            implicit none
            class(slae_port_gen), INTENT(inout) :: this
            deallocate(this%lists)
        end subroutine

        subroutine init_g(this, nodes_n)
            implicit none
            class(slae_port_gen) :: this
            integer :: nodes_n, i
            this%n = nodes_n
            allocate(this%lists(nodes_n))
            do i = 1, nodes_n, 1
                call this%lists(i)%set_num(i)
            end do
        end subroutine

        subroutine add_el_g(this, el_a)
            implicit none
            class(slae_port_gen) :: this
            class(fe) :: el_a
            integer :: i
            do i = 1, 16, 1
                call this%lists(el_a%node_n(i))%add_el_l(el_a)
            end do
        end subroutine

        subroutine gen(this, gi, gj, m)
            implicit none
            class(slae_port_gen) :: this
            integer :: gi(this%n + 1), m, i, j, shift, m1 = 0,
iters_m
            integer, pointer :: gj(:)
            m = 1
            gi(1) = 1;
            do i = 1, this%n, 1
                gi(i) = m
                m = m + this%lists(i)%size_before(i)
            end do
            gi(this%n + 1) = m
            allocate(gj(m))
            do i = 1, this%n, 1
                shift = m1
                iters_m = this%lists(i)%get_m_size()
                call this%lists(i)%cash_off()
                do j = 1, iters_m, 1
                    gj(shift + j) =
this%lists(i)%take_and_next()
                end do
```

```
            m1 = m1 + iters_m
        end do

    end subroutine

end module
```

solver_lu.f95

```
#if __GFORTRAN__ == 1 && __GNUC__ == 4 && __GNUC_MINOR__ < 5
#define class type
#endif
    module solver_lu
        implicit none

        type :: lu
            integer, private :: n ! Размерность СЛАУ
            ! Основные массивы
            integer, pointer, private :: ig(:)
            double precision, pointer, private :: gu(:), gl(:),
di(:)
            double precision, pointer, private :: rp(:)
        contains
            procedure :: init   ! инициализация
            procedure :: set_rp ! Установка правой части
            procedure :: solve  ! Получение решения и количе-
ства итераций
            procedure, private :: dec ! Прострoение LU-
разложения
        end type

    contains

        subroutine init(this, s_ig, s_jg, s_gu, s_gl, s_di,
s_n)
            implicit none
            class(lu) :: this
            integer, pointer :: s_ig(:), s_jg(:)
            double precision, pointer :: s_gu(:), s_gl(:),
s_di(:)
            integer :: s_n, i, j, k, total_n, j_s, j_e, column,
s_point
            this%n = s_n
            allocate(this%di(this%n))
            allocate(this%ig(this%n+1))
            ! Перенос диагонали
            do i = 1, this%n, 1
                this%di(i) = s_di(i)
            end do
            ! Формируем новый массив ig
            this%ig(1) = 1
            do i = 2, this%n+1, 1
                k = s_ig(i) - s_ig(i-1)
                if(k .gt. 0)then
                    total_n = i - s_jg(s_ig(i-1))
                    this%ig(i) = this%ig(i-1) + total_n
                else
                    this%ig(i) = this%ig(i-1)
                end if
            end do
            allocate(this%gu(this%ig(this%n+1)))
            allocate(this%gl(this%ig(this%n+1)))
            ! Формируем новые gl и gu
            do i = 1, this%n, 1
                j_s = this%ig(i)
                j_e = this%ig(i+1)
                column = i - (j_e - j_s)
                s_point = s_ig(i)
                do j = j_s, j_e-1, 1
                    if(column .eq. s_jg(s_point))then
                        this%gu(j) = s_gu(s_point)
                        this%gl(j) = s_gl(s_point)
                        s_point = s_point + 1
                    else
                        this%gu(j) = 0d0
                        this%gl(j) = 0d0
                    end if
                    column = column + 1
                end do
            end do
            call this%dec()
        end subroutine

        subroutine set_rp(this, s_rp)
            implicit none
            class(lu) :: this
            double precision, pointer :: s_rp(:)
            integer :: i
            allocate(this%rp(this%n))
            do i = 1, this%n, 1
                this%rp(i) = s_rp(i)
            end do
        end subroutine

        subroutine dec(this)
            implicit none
            class(lu) :: this
            integer :: i, i0, i1, j, j0, j1, m, mi, mj, kol_i,
kol_j, kol_r
            double precision :: sd, sl, su
            do i = 1, this%n, 1
                i0 = this%ig(i)
                i1 = this%ig(i+1)
                j = i - (i1 - i0)
                sd = 0d0
                do m = i0, i1 - 1, 1
                    sl = 0d0
                    su = 0d0
                    j0 = this%ig(j)
                    j1 = this%ig(j+1)
                    mi = j0
                    mj = j0
                    kol_i = m - i0
                    kol_j = j1 - j0
                    kol_r = kol_i - kol_j
                    if(kol_r .lt. 0)then
                        mj = mj - kol_r
                    else
                        mi = mi + kol_r
                    end if
                    do mi = mi, m-1, 1
                        sl = sl + this%gl(mi) * this%gu(mj)
                        su = su + this%gu(mi) * this%gl(mj)
                        mj = mj + 1
                    end do
                    this%gl(m) = this%gl(m) - sl
                    this%gu(m) = (this%gu(m) - su) / this%di(j)
                    sd = sd + this%gl(m) * this%gu(m)
                    j = j + 1
                end do
                this%di(i) = this%di(i) - sd
            end do
        end subroutine

        subroutine solve(this, solution, its)
            implicit none
            class(lu) :: this
            double precision, pointer :: solution(:)
            integer :: its, i, j, j_start, j_end, vect_iter
            double precision :: sum
            allocate(solution(this%n))
            ! Прямой ход
            do i = 1, this%n, 1
                sum = 0d0
                j_start = this%ig(i)
                j_end = this%ig(i+1)
                vect_iter = i - (j_end - j_start)
                do j = j_start, j_end-1, 1
                    sum = sum + this%gl(j) * this%rp(vect_iter)
                    vect_iter = vect_iter + 1
                end do
                this%rp(i) = (this%rp(i) - sum) / this%di(i)
            end do
            ! Обраный ход
            do i = this%n, 1, -1
                j_start = this%ig(i)
                j_end = this%ig(i+1)
                vect_iter = i - (j_end - j_start)
                do j = j_start, j_end-1, 1
                    this%rp(vect_iter) = this%rp(vect_iter) -
this%gu(j) * this%rp(i)
                    vect_iter = vect_iter + 1
                end do
            end do

            do i = 1, this%n, 1
                solution(i) = this%rp(i)
            end do
            deallocate(this%ig)
            deallocate(this%gu)
            deallocate(this%gl)
            deallocate(this%di)
            deallocate(this%rp)
            its = 1
        end subroutine

    end module
```

solver_los.f95

```
#if __GFORTRAN__ == 1 && __GNUC__ == 4 && __GNUC_MINOR__ < 5
#define class type
#endif
    module solver_los
        implicit none
        ! Параметры решателя
        integer, parameter, private :: max_iter = 100000
        double precision, parameter, private :: eps = 1d-16

        type :: los
            integer, private :: n ! Размерность СЛАУ
            ! Основные массивы
            integer, pointer, private :: ig(:), jg(:)
            double precision, pointer, private :: gu(:), gl(:),
di(:)
            double precision, pointer, private :: rp(:)
            ! Массивы для преобславливателя
            double precision, allocatable, private :: uu(:),
ll(:), ld(:)
        contains
            procedure, private :: precond  ! Вычисление матриц
L и U
            procedure, private :: dot_prod ! скалярное
произведение
            procedure, private :: mull_a   ! x = Af
            procedure, private :: solve_l  ! Lx = f, прямой ход
            procedure, private :: solve_u  ! Ux = f, обратный
ход
            procedure :: init   ! инициализация
            procedure :: set_rp ! Установка правой части
            procedure :: solve  ! Получение решения и количе-
ства итераций
        end type

    contains
```

```
        subroutine init(this, s_ig, s_jg, s_gu, s_gl, s_di,
s_n)
            implicit none
            class(los) :: this
            integer, pointer :: s_ig(:), s_jg(:)
            double precision, pointer :: s_gu(:), s_gl(:),
s_di(:)
            integer :: s_n
            this%ig => s_ig
            this%jg => s_jg
            this%gu => s_gu
            this%gl => s_gl
            this%di => s_di
            this%n = s_n
            call this%precond()
        end subroutine

        subroutine set_rp(this, s_rp)
            implicit none
            class(los) :: this
            double precision, pointer :: s_rp(:)
            this%rp => s_rp
        end subroutine

        subroutine precond(this)
            implicit none
            class(los) :: this
            double precision :: sum_l, sum_u, sum_d !
Промежуточные переменные, для вычисления сумм
            integer :: copy_end, i, j, k, k1, i_s, i_e, m, j_s,
j_e
            copy_end = this%ig(this%n + 1)
            allocate(this%ll(copy_end))
            allocate(this%uu(copy_end))
            allocate(this%ld(this%n))
            ! Копируем старые в новые
            do i = 1, copy_end, 1
                this%ll(i) = this%gl(i)
                this%uu(i) = this%gu(i)
            end do
            do i = 1, this%n, 1
                this%ld(i) = this%di(i)
            end do
            k1 = 1
            do k = 2, this%n + 1, 1
                sum_d = 0d0
                i_s = this%ig(k1)
                i_e = this%ig(k)
                do m = i_s, i_e - 1, 1
                    sum_l = 0d0
                    sum_u = 0d0
                    j_s = this%ig(this%jg(m))
                    j_e = this%ig(this%jg(m)+1)
                    do i = i_s, m - 1, 1
                        do j = j_s, j_e - 1, 1
                            sum_l = sum_l + this%ll(i) *
this%uu(j)
                            sum_u = sum_u + this%ll(j) *
this%uu(i)
                            j_s = j_s + 1
                        end do
                    end do
                    this%ll(m) = this%ll(m) - sum_l
                    this%uu(m) = (this%uu(m) - sum_u) /
this%ld(this%jg(m))
                    sum_d = sum_d + this%ll(m) * this%uu(m)
                end do
                this%ld(k1) = this%ld(k1) - sum_d
                k1 = k1 + 1
            end do
        end subroutine

        function dot_prod(this, a, b)
            implicit none
            class(los) :: this
            double precision :: a(*), b(*), dot_prod
            integer :: i
            dot_prod = 0d0
            do i = 1, this%n, 1
                dot_prod = dot_prod + a(i) * b(i)
            end do
        end function

        subroutine mull_a(this, f, x)
            implicit none
            class(los) :: this
            double precision :: f(*), x(*), v_el
            integer :: i, j, k, k1
            do i = 1, this%n, 1
                v_el = f(i)
                x(i) = this%di(i) * v_el
                k1 = this%ig(i + 1)
                do k = this%ig(i), k1 - 1, 1
                    j = this%jg(k)
                    x(i) = x(i) + this%gl(k) * f(j)
                    x(j) = x(j) + this%gu(k) * v_el
                end do
            end do
        end subroutine

        subroutine solve_l(this, f, x)
            implicit none
            class(los) :: this
            double precision :: f(*), x(*), sum
            integer :: i, k, k1
            k1 = 1
            do k = 2, this%n + 1, 1
                sum = 0d0
                do i = this%ig(k1), this%ig(k) - 1, 1
                    sum = sum + this%ll(i) * x(this%jg(i))
                end do
                x(k1) = (f(k1) - sum) / this%ld(k1)
                k1 = k1 + 1
            end do
        end subroutine

        subroutine solve_u(this, f, x)
            implicit none
            class(los) :: this
            double precision :: f(*), x(*), v_el
            double precision, allocatable :: f1(:)
            integer :: i, k, k1
            allocate(f1(this%n))
            do i = 1, this%n, 1
                f1(i) = f(i)
            end do
            k1 = this%n
            do k = this%n + 1, 2, -1
                x(k1) = f1(k1) / this%ld(k1)
                v_el = x(k1)
                do i = this%ig(k1), this%ig(k) - 1, 1
                    f1(this%jg(i)) = f1(this%jg(i)) -
this%uu(i) * v_el
                end do
                k1 = k1 - 1
            end do
            deallocate(f1)
        end subroutine

        subroutine solve(this, solution, its)
            implicit none
            class(los) :: this
            double precision, pointer :: solution(:)
            double precision :: rp_norm, discr, dot1, alpha,
betta
            double precision, allocatable :: x0(:), r(:), z(:),
p(:), s(:), t(:)
            integer :: its, end_cycle = 0, iter
            ! Норма правой части, для выхода
            rp_norm = dsqrt(dot_prod(this, this%rp, this%rp))
            ! Начинаем решение
            allocate(x0(this%n)) ! Приближение
            x0 = 0d0
            allocate(solution(this%n))
            allocate(r(this%n)) ! Вектор невязки
            allocate(z(this%n))
            allocate(p(this%n))
            allocate(s(this%n)) ! Вспомогательный вектор
            allocate(t(this%n)) ! Вспомогательный вектор
            ! r0 = L^(-1) * (f - Ax0)
            call this%mull_a(x0, s)
            s = this%rp - s
            call this%solve_l(s, r)
            ! z0 = U^(-1)r0
            call this%solve_u(r,z)
            ! p0 = L^(-1)Az0
            call this%mull_a(z, s)
            call this%solve_l(s, p)
            iter = 0
            do while(iter .lt. max_iter .and. end_cycle .eq. 0)
                discr = dsqrt(dot_prod(this, r, r)) !
Абсолютная невязка
                if(discr / rp_norm .gt. eps)then ! Проверка
условия выхода
                    dot1 = dot_prod(this, p, p) ! (p[k-1], p[k-
1])
                    alpha = dot_prod(this, p ,r) / dot1 ! a =
(p[k-1], r[k-1]) / (p[k-1], p[k-1])
                    x0 = x0 + alpha * z ! x[k] = x[k-1] +
a*z[k-1]
                    r = r - alpha * p ! r[k] = r[k-1] - a*p[k-
1]
                    ! betta = -(p[k-1], L^(-1)*A*U^(-1)r[k]) /
(p[k-1], p[k-1])
                    call this%solve_u(r, s) ! s = U^(-1)r[k]
                    call this%mull_a(s, t)
                    call this%solve_l(t, t);
                    betta = - dot_prod(this, p, t) / dot1
                    z = s + betta * z ! z[k] = U^(-1)r[k] +
b*z[k-1]
                    p = t + betta * p ! p[k] = L^(-1)*A*U^(-
1)r[k] + b*p[k-1]
                    if(mod(iter, this%n) .eq. 0) then ! Обнов-
ление метода
                        ! r0 = L^(-1) * (f - Ax0)
                        call this%mull_a(x0, s)
                        s = this%rp - s
                        call this%solve_l(s, r)
                        ! z0 = U^(-1)r0
                        call this%solve_u(r, z)
                        ! p0 = L^(-1)Az0
                        call this%mull_a(z, s)
                        call this%solve_l(s, p)
                    end if
                else
                    end_cycle = 1
                end if
                iter = iter + 1
            end do
            solution = x0
            its = iter
            deallocate(x0)
            deallocate(p)
            deallocate(r)
            deallocate(z)
            deallocate(s)
            deallocate(t)
        end subroutine
```

```
    end module
```

solver_gmres.f95

```
#if __GFORTRAN__ == 1 && __GNUC__ == 4 && __GNUC_MINOR__ < 5
#define class type
#endif
    module solver_gmres
        implicit none
        ! Параметры решателя
        integer, parameter, private :: max_iter = 10000
        double precision, parameter, private :: eps = 1d-14
        integer, parameter, private :: m = 3

        type :: gmres
            integer, private :: n ! Размерность СЛАУ
            ! Основные массивы
            integer, pointer, private :: ig(:), jg(:)
            double precision, pointer, private :: gu(:), gl(:),
di(:)
            double precision, pointer, private :: rp(:)
            ! Массивы для преобславливателя
            double precision, allocatable, private :: uu(:),
ll(:), ld(:)
            ! Вспомогательное для GMRES
            double precision, allocatable, private :: h(:),
h2(:), v(:), w(:), y(:)
        contains
            procedure, private :: precond  ! Вычисление матриц
L и U
            procedure, private :: dot_prod ! скалярное
произведение
            procedure, private :: mull_a   ! x = Af
            procedure, private :: solve_l  ! Lx = f, прямой ход
            procedure, private :: solve_u  ! Ux = f, обратный
ход
            procedure :: init   ! инициализация
            procedure :: set_rp ! Установка правой части
            procedure :: solve  ! Получение решения и количе-
ства итераций
            procedure, private :: mult_v_y
            procedure, private :: insert_col
            procedure, private :: calc_h_w
            procedure, private :: calc_hth
            procedure, private :: gauss
        end type

    contains

        subroutine init(this, s_ig, s_jg, s_gu, s_gl, s_di,
s_n)
            implicit none
            class(gmres) :: this
            integer, pointer :: s_ig(:), s_jg(:)
            double precision, pointer :: s_gu(:), s_gl(:),
s_di(:)
            integer :: s_n
            this%ig => s_ig
            this%jg => s_jg
            this%gu => s_gu
            this%gl => s_gl
            this%di => s_di
            this%n = s_n
            call this%precond()
        end subroutine

        subroutine set_rp(this, s_rp)
            implicit none
            class(gmres) :: this
            double precision, pointer :: s_rp(:)
            this%rp => s_rp
        end subroutine

        subroutine precond(this)
            implicit none
            class(gmres) :: this
            double precision :: sum_l, sum_u, sum_d !
Промежуточные переменные, для вычисления сумм
            integer :: copy_end, i, j, k, k1, i_s, i_e, m, j_s,
j_e
            copy_end = this%ig(this%n + 1)
            allocate(this%ll(copy_end))
            allocate(this%uu(copy_end))
            allocate(this%ld(this%n))
            ! Копируем старые в новые
            do i = 1, copy_end, 1
                this%ll(i) = this%gl(i)
                this%uu(i) = this%gu(i)
            end do
            do i = 1, this%n, 1
                this%ld(i) = this%di(i)
            end do
            k1 = 1
            do k = 2, this%n + 1, 1
                sum_d = 0d0
                i_s = this%ig(k1)
                i_e = this%ig(k)
                do m = i_s, i_e - 1, 1
                    sum_l = 0d0
                    sum_u = 0d0
                    j_s = this%ig(this%jg(m))
                    j_e = this%ig(this%jg(m)+1)
                    do i = i_s, m - 1, 1
                        do j = j_s, j_e - 1, 1
                            sum_l = sum_l + this%ll(i) *
this%uu(j)
                            sum_u = sum_u + this%ll(j) *
this%uu(i)
                            j_s = j_s + 1
                        end do
                    end do
                    this%ll(m) = this%ll(m) - sum_l
                    this%uu(m) = (this%uu(m) - sum_u) /
this%ld(this%jg(m))
                    sum_d = sum_d + this%ll(m) * this%uu(m)
                end do
                this%ld(k1) = this%ld(k1) - sum_d
                k1 = k1 + 1
            end do
        end subroutine

        function dot_prod(this, a, b)
            implicit none
            class(gmres) :: this
            double precision :: a(*), b(*), dot_prod
            integer :: i
            dot_prod = 0d0
            do i = 1, this%n, 1
                dot_prod = dot_prod + a(i) * b(i)
            end do
        end function

        subroutine mull_a(this, f, x)
            implicit none
            class(gmres) :: this
            double precision :: f(*), x(*), v_el
            integer :: i, j, k, k1
            do i = 1, this%n, 1
                v_el = f(i)
                x(i) = this%di(i) * v_el
                k1 = this%ig(i + 1)
                do k = this%ig(i), k1 - 1, 1
                    j = this%jg(k)
                    x(i) = x(i) + this%gl(k) * f(j)
                    x(j) = x(j) + this%gu(k) * v_el
                end do
            end do
        end subroutine

        subroutine solve_l(this, f, x)
            implicit none
            class(gmres) :: this
            double precision :: f(*), x(*), sum
            integer :: i, k, k1
            k1 = 1
            do k = 2, this%n + 1, 1
                sum = 0d0
                do i = this%ig(k1), this%ig(k) - 1, 1
                    sum = sum + this%ll(i) * x(this%jg(i))
                end do
                x(k1) = (f(k1) - sum) / this%ld(k1)
                k1 = k1 + 1
            end do
        end subroutine

        subroutine solve_u(this, f, x)
            implicit none
            class(gmres) :: this
            double precision :: f(*), x(*), v_el
            double precision, allocatable :: f1(:)
            integer :: i, k, k1
            allocate(f1(this%n))
            do i = 1, this%n, 1
                f1(i) = f(i)
            end do
            k1 = this%n
            do k = this%n + 1, 2, -1
                x(k1) = f1(k1) / this%ld(k1)
                v_el = x(k1)
                do i = this%ig(k1), this%ig(k) - 1, 1
                    f1(this%jg(i)) = f1(this%jg(i)) -
this%uu(i) * v_el
                end do
                k1 = k1 - 1
            end do
            deallocate(f1)
        end subroutine

        subroutine mult_v_y(this, mr, res)
            implicit none
            class(gmres) :: this
            integer :: mr, k, p
            double precision :: res(*)
            do k = 1, this%n, 1
                res(k) = 0d0
                do p = 1, mr, 1
                    res(k) = res(k) + this%v(k*m+p) * this%y(p)
                end do
            end do
        end subroutine

        subroutine insert_col(this, j, x)
            implicit none
            class(gmres) :: this
            integer :: i, j
            double precision :: x(*)
            do i = 1, this%n, 1
                this%v(i*m+j) = x(i)
            end do
        end subroutine

        subroutine calc_h_w(this, i, j)
            implicit none
            class(gmres) :: this
            integer :: i, j, ind, k
```

```
            ind = i*m+j
            this%h(ind) = 0d0
            do k = 1, this%n, 1
                this%h(ind) = this%h(ind) + this%w(k) *
this%v(k*m+i)
            end do
            do k = 1, this%n, 1
                this%w(k) = this%w(k) - this%h(ind) *
this%v(k*m+i)
            end do
        end subroutine

        subroutine calc_hth(this, mr)
            implicit none
            class(gmres) :: this
            integer :: mr, i, j, k, idx
            this%h2 = 0d0
            do i = 1, mr, 1
                do j = 1, mr, 1
                    idx = i * m + j
                    do k = 1, mr + 1, 1
                        this%h2(idx) = this%h2(idx) + this%h(k
* m + i) * this%h(k * m + j)
                    end do
                end do
            end do
        end subroutine

        subroutine gauss(this, matr, vector, mr)
            implicit none
            class(gmres) :: this
            integer :: mr, i, j, k, ind_di, ind_maxstr
            double precision :: matr(:), vector(:), temp, max
            do i = 1, mr, 1
                ind_di = i * (m + 1)
                ind_maxstr = i
                max = dabs(matr(ind_di))
                do j = 2, mr - i, 1
                    if(dabs(matr(ind_di)) .gt. max) then
                        max = dabs(matr(ind_di + j * m))
                        ind_maxstr = i + j
                    end if
                end do
                do j = i, mr, 1
                    temp = matr(i * m + j)
                    matr(i * m + j) = matr(ind_maxstr * m + j)
                    matr(ind_maxstr * m + j) = temp
                end do
                temp = vector(i)
                vector(i) = vector(ind_maxstr)
                vector(ind_maxstr) = temp
                do j = 2, mr - i, 1
                    matr(ind_di + j) = matr(ind_di + j) /
matr(ind_di)
                end do
                vector(i) = vector(i) / matr(ind_di)
                matr(ind_di) = 1d0
                do j = i+1, mr, 1
                    do k = i+1, mr, 1
                        matr(j*m+k) = matr(j*m+k) - matr(i*m+k)
* matr(j*m+i)
                    end do
                    vector(j) = vector(j) - matr(j*m+i) * vec-
tor(i)
                    matr(j*m+i) = 0d0
                end do
            end do
            do i = mr, 1, -1
                do j = mr, i+1, -1
                    vector(i) = vector(i) - matr(i*m+j) * vec-
tor(j)
                end do
            end do
        end subroutine

        subroutine solve(this, solution, its)
            implicit none
            class(gmres) :: this
            double precision, pointer :: solution(:)
            double precision :: rp_norm, r_norm, h_last
            double precision, allocatable :: t(:), p(:)
            integer :: its, iter, newm, flag, i, j
            rp_norm = dsqrt(dot_prod(this, this%rp, this%rp))
            allocate(solution(this%n))
            solution = 0d0
            allocate(p(this%n))
            allocate(t(this%n))
            allocate(this%w(this%n))
            allocate(this%y(m))
            allocate(this%v(m * this%n))
            allocate(this%h(m * (m + 1)))
            allocate(this%h2(m * m))
            newm = m
            iter = 0
            ! вычисляем r0
            call this%mull_a(solution, t) ! t=A*x
            this%w = this%rp - t
            call this%solve_l(this%w, p) ! L*p=f-Ax
            r_norm = dsqrt(dot_prod(this, p, p)) ! p = r0
            do while(iter .lt. max_iter .and. r_norm / rp_norm
.gt. eps)
                flag = 0
                p = p / r_norm
                call this%insert_col(1, p)
                do j = 1, newm, 1
                    call this%solve_u(p, t) ! U*t=p
                    call this%mull_a(t, p) ! p=At
                    call this%solve_l(p, this%w) ! L*w=A*t
                    do i = 1, j, 1
                        call this%calc_h_w(i, j)
                    end do
                    this%h(j*m+j) = dsqrt(dot_prod(this,
this%w, this%w))
                    h_last = this%h(j*m+j)
                    if(dabs(h_last) .gt. eps**2 .and. j .ne.
newm)then
                        p(i) = this%w(i) / h_last
                        call this%insert_col(j+1, p)
                    end if
                    if(dabs(h_last) .lt. eps**2)then
                        newm = j
                        flag = 1
                    end if
                end do
                if(flag .ne. 0)then
                    this%y = 0d0
                    this%y(1) = r_norm
                    call this%gauss(this%h, this%y, newm)
                else
                    this%y = this%h * r_norm
                    call this%calc_hth(newm)
                    call this%gauss(this%h2, this%y, newm)
                end if
                call this%mult_v_y(newm, t) ! t=Vy
                call this%solve_u(t, p) ! U*p=t
                solution = solution + p ! x+=U(-1)*V*y
                call this%mull_a(solution, t) ! t=Ax
                r_norm = dsqrt(dot_prod(this, p, p)) ! p=r
                iter = iter + 1
            end do
            its = iter
            deallocate(p)
            deallocate(t)
            deallocate(this%w)
            deallocate(this%y)
            deallocate(this%v)
            deallocate(this%h)
            deallocate(this%h2)
        end subroutine

    end module
```