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: , в  , с правой частью вида:

, 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: ,

тогда исходное уравнение можно представить в виде системы: 

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



Следом проведём аппроксимацию: , 

Подставив выражения в систему, получим СЛАУ из  уравнений.

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



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

Локальная правая часть тоже буде иметь блочную структуру: это будет вектор из 8 двухмерных векторов вида: , причём вектора  и  вычисляются через соответствующие значения правой части: 

Локальные матрицы массы краевых условий будут иметь блочную структуру с видом блоков: , где - матрица массы для билиныйных функциях на прямоугольниках, добавки в правую часть будут вычисляться умножением этой матрицы на вектор значений функции или .

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  и  , т.е. решается СЛАУ вида  , где  . Выбирается начальное приближение  и полагается: , .

Далее выполняются итерации метода, на каждой итерации вычисляются элементы двух вспомогательных матриц  размера  и  размера , где  – размерность решаемой СЛАУ, а  – глубина метода.

Пусть  –  -й столбец матрицы . Первый столбец формируется по формуле: 

Далее для  выполняется:



, 





Если , то процесс построения  заканчивается, и за  принимается , иначе 

Новое приближение  определяется следующим образом:

Вычисляется вектор  размера  .

Находится вектор параметров :  .

После чего вычисляется  .

Новая невязка рассчитывается как  .

После окончания итерационного процесса решение исходной СЛАУ определяется как  .

4. Исследования и тесты

4.1. Полином первой степени.

Сетка: 

Шаг:  (число узлов – 216)

Аналитическое решение: , 

Параметры уравнения: , , , 

Вид правой части: , 

Параметры решателей:  ,  , 

|  |  |  |  |
| --- | --- | --- | --- |
|  | *LU* | *LOS* | *GMRES* |
| Отн. погрешность | 2.370e-16 | 4.519e-16 | 2.461e-15 |
| Отн. погрешность | 2.470e-16 | 4.653e-16 | 2.318e-15 |
| Отн. погрешность | 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: 

Шаг:  (число узлов – при  - 4913, при  - 35937)

Аналитическое решение: , 

Параметры уравнения: , , , 

Вид правой части: , 

Параметры решателей:  ,  , 

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | *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 | 1.123e-03 | 3.285e-04 | 1.123e-03 | 3.285e-04 | 1.123e-03 | 3.285e-04 |
| Отн. погрешность | 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 |



4.3. Исследования для сеток с небольшим количеством узлов.

Сетка: 

Шаг:  (число узлов – 729)

Аналитическое решение: , 

Вид правой части: , 

Параметры решателей:  ,  , 

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | *LU, мс* | *LOS, ит* | *LOS, мс* | *GMRES, ит* | *GMRES, мс* |
|  |  |  |  | 30.410000 | 87 | 22.525000 | 13 | 16.545000 |
|  |  |  |  | 29.640000 | 96 | 24.180000 | 14 | 17.191000 |
|  |  |  |  | 29.701000 | 91 | 22.742000 | 13 | 16.795000 |
|  |  |  |  | 29.609000 | 92 | 23.552000 | 13 | 16.455000 |
|  |  |  |  | 1548.4240 | 10000 | 170.75200 | 396 | 262.04100 |
|  |  |  |  | 1549.0610 | 10000 | 163.07900 | 10000 | 6380.1530 |
|  |  |  |  | 33.114000 | 1238 | 200.08300 | 50 | 40.279000 |
|  |  |  |  | 29.726000 | 10000 | 1555.6310 | 10000 | 6427.9090 |

4.4. Исследования для сеток с большим количеством узлов.

Сетка: 

Шаг:  ,  ,  (число узлов – 35301)

Аналитическое решение: , 

Вид правой части: , 

Параметры решателей:  ,  , 

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | *LU, мс* | *LOS, ит* | *LOS, мс* | *GMRES, ит* | *GMRES, мс* |
|  |  |  |  | 151831.54 | 2575 | 25063.307 | 207 | 8845.5030 |
|  |  |  |  | 152323.37 | 2584 | 25506.373 | 208 | 8904.6220 |
|  |  |  |  | 152895.02 | 921 | 9463.2200 | 190 | 8450.9180 |
|  |  |  |  | 150848.38 | 917 | 9396.1100 | 172 | 7441.6760 |
|  |  |  |  | 152075.70 | 1439 | 14304.545 | 252 | 10647.963 |
|  |  |  |  | 152120.07 | 10000 | 98062.427 | 10000 | 405763.26 |
|  |  |  |  | 151097.01 | 1570 | 15491.485 | 32 | 1820.7110 |
|  |  |  |  | 153620.45 | 10000 | 97067.476 | 10000 | 365335.06 |

4.5. Исследования на глубину метода в GMRES.

Сетка: 

Шаг:  (число узлов – 9261)

Аналитическое решение: , 

Параметры уравнения: , , , 

Вид правой части: , 

Параметры решателей:  , 

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | *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 | 7.636e-04 | 7.636e-04 | 7.636e-04 | 7.636e-04 | 7.636e-04 | 7.636e-04 |
| Отн. погрешность | 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 ,  и сходимость итерационных методов ухудшается, а при увеличении 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 - 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, faces\_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 :: faces\_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 ! количество 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 элементов 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 резульатата 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) ! локальные 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/),shape=(/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() \* sigma\_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, number\_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 количества итераций

procedure, private :: dec ! Простроение 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 количества 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 ! Обновление 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 количества 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) \* vector(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) \* vector(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