ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 46 за 2020 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

Борисов В.Е., Зипунова Е.В., Иванов А.В., Критский Б.В., Савенков Е.Б.

Программный комплекс HFrac3D++ для решения задач геомеханики с учетом крупномасштабных флюидонаполненных трещин

Рекомендуемая форма библиографической ссылки: Программный комплекс HFrac3D++ для решения задач геомеханики с учетом крупномасштабных флюидонаполненных трещин / В.Е.Борисов [и др.] // Препринты ИПМ им. М.В.Келдыша. 2020. № 46. 20 с. http://doi.org/10.20948/prepr-2020-46
URL: http://library.keldysh.ru/preprint.asp?id=2020-46

РОССИЙСКАЯ АКАДЕМИЯ НАУК
ОРДЕНА ЛЕНИНА
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М. В. КЕЛДЫША

В.Е. Борисов, Е.В. Зипунова, А.В. Иванов,
Б.В. Критский, Е.Б. Савенков

Программный комплекс HFrac3D++ для решения задач геомеханики с учетом крупномасштабных флюидонаполненных трещин

Москва, 2020

В.Е. Борисов, Е.В. Зипунова, А.В. Иванов, Б.В. Критский, Е.Б. Савенков
Программный комплекс HFrac3D++ для решения задач геомеханики с учетом крупномасштабных флюидонаполненных трещин

Аннотация. В работе представлено описание программного комплекса HFrac3D++, предназначенного для моделирования задач механики, в том числе при наличии флюидонаполненных трещин. Кратко описана использованная математическая модель, вычислительные алгоритмы, архитектура комплекса и способы его применения. Программный комплекс HFrac3D++ реализован в виде вычислительного ядра на языке C++11 и программного интерфейса на языке Python, функционирующих в операционной системе Linux.

Ключевые слова: метод конечных элементов, X-FEM, трещина гидроразрыва пласта

V.E. Borisov, E.V. Zipunova, A.V. Ivanov, B.V. Kritsky, E.B. Savenkov
HFrac3D++ code for geomechanics problems with large-scale fluid-filled fractures

Abstract. The work is devoted to description of HFrac3D++ computer code for finite element simulation of mechanical and geomechanical problems with large-scale fractures. Corresponding mathematical model and numverical algorithms are described as well as software architecture. The implementation uses C++11 programming language for core computational algorithms and Python for upper level software interface.

Key words and phrases: finite element method, X-FEM, hydraulic fracturing

Содержание

1 Введение

Программный комплекс HFrac3D++ [1] предназначен для анализа процесса гидроразрыва пласта (ГРП) [2] в рамках расширенной постановки, включающей в себя большинство основных эффектов, влияющих на динамику трещины ГРП [3]. Осуществляется согласованное решение следующих групп уравнений: (1) системы уравнений пороупругости, описывающей эволюцию напряженно-деформированного состояния среды и полей давления флюида в ней в ходе развития трещины; (2) уравнения течения жидкости в трещине; (3) механических условий развития трещины, определяющих направление ее развития в каждой точке ее фронта; (4) заданных на боковых поверхностях трещины условий согласования между полями давления в трещине и в среде, потоками массы, импульса и энергии, а также кинематическими условиями, связывающими раскрытие трещины и перемещение точек пласта. Программный комплекс имеет модульную структуру, позволяя при необходимости отключать учет несущественных для конкретной задачи эффектов.

Математическая модель, используемая в комплексе, подробно описана в работе [3]. Вычислительный алгоритм основан на применении метода проекции ближайшей точки для представления поверхности [4] и варианте расширенного метода конечных элементов X-FEM/CPP [5, 6]. Детали реализации и примеры расчетов представлены в работах [7–18].

Программный комплекс `HFrac3D++` реализован в виде вычислительного ядра на языке `C++11` и интерфейса на языке `Python` под `OS Linux`. Для связывания `C++` и `Python` используется утилита `SWIG`. При разработке комплекса широко использовалась библиотека `aiwlib`, см. [19, 20].

В вычислительном ядре реализован класс модели, хранящий глобальные параметры, неструктурированную расчетную тетраэдрическую сетку (и данные на ней), граничные условия и так далее. Кроме того класс модели предоставляет методы для инициализации сетки, перехода на следующий временной слой, вывода диагностики в различных форматах.

Поскольку с одним вычислительным ядром можно моделировать множество различных постановок задач, для каждой постановки создается отдельный головной скрипт на языке `Python`, содержащий создание экземпляра класса модели, его инициализацию, расчетный цикл по времени и сброс результатов расчета.

Для задания расчетной области сложной геометрии и генерации неструктурированной сетки используется широко распространенный пакет `Salome`. Данная платформа предоставляет средства для пред- и постобработки данных численного моделирования. В частности, в пакете есть встроенные средства твердотельного моделирования и интерфейс к пакетам создания расчетных сеток. После задания области в `Salome` запускается отдельный скрипт на

языке Python, сохраняющий всю необходимую информацию о сетке в один файл в формате pickle. При запуске расчета этот файл зачитывается и на его основе создается сетка, а для различных фрагментов задаются физические параметры счетной области, начальные и граничные условия. При этом используются имена групп (узлов, граней и ячеек сетки), заданные пользователем в Salome. В головном скрипте на языке Python для каждой группы тетраэдров можно задать физические параметры и начальные условия, а для групп узлов и граней можно задавать граничные условия различного типа.

После создания и инициализации класса модели в основном скрипте запускается цикл расчета по времени. При этом есть возможность сохранять результаты расчета в различных форматах. Для сохранения мгновенных снимков распределения физических полей (перемещений и давлений в узлах тетраэдрической сетки) доступны форматы .vtk (текстовый, может отрисовываться в стандартном вьювере paraview) и бинарный формат библиотеки aiwlib. Кроме того, в текстовые файлы может сохраняться эволюция значений полей в отдельных точках или на скважинах.

Таким образом, программный комплекс состоит из:

1. набора скриптов на языке Python2.7 для конвертации данных из пакета Salome во внутренний формат кода (или, при необходимости, формат CSV), генерации тестовых сеток;

2. вычислительного ядра — разделяемой библиотеки _model.so компилируемого из исходных C++ кодов при помощи утилиты GNU Make;

3. интерфейса вычислительного ядра на языке Python2.7 — модуля model.py и импортирующего его модуля geomech.py;

4. создаваемых конечным пользователем скриптов на языке Python2.7, реализующих загрузку сетки, задание начальных и граничных условий, запуск расчета и сохранение результатов.

Под различные постановки задач конечные пользователи могут создавать различные скрипты на языке Python. При этом в Python необходимо создать и настроить (задать ключевые параметры задачи) экземпляр C++–класса Model, реализующий инициализацию, хранение данных, выполнение различных этапов численной схемы на одном шаге (расчет полей напряжений, фильтрации, движения трещины и т.д.), сохранение результатов расчетов в формате VTK. Вызов соответствующих методов в Python обеспечивает проведение расчета для конкретной постановки задачи.

2 Задание формы расчетной области и генерация сетки

Расчетная область задается в пакете Salome с помощью его внутренних процедур. После генерации сетка может быть выгружена из пакета Salome при помощи скрипта salome2pickle.py в формат, предназначенный для кода HFrac3D++. Для выгрузки сетки в текстовый (CSV) формат (например для загрузки в Matlab) предназначен скрипт salome2matlab.py.

При задании расчетной области в пакете Salome область может быть разбита на несколько подобластей. Для последующего задания физических параметров и начальных условий узлы, тетраэдры (элементы) и грани каждой подобласти необходимо объединить в группы и дать каждой группе имя собственное, начинающееся с phys_. Аналогично узлы и грани границ подобластей, на которых будут ставиться нетривиальные граничные условия, также необходимо объединить в группы и дать группам имена, начинающиеся с bound_.

Для передачи сетки в программный комплекс HFrac3D++ используется стандартный формат сериализации pickle языка Python. При этом в один файл сохраняется следующая структура данных языка Python:

```
[ [(x0,y0,z0), (x1,y1,z1), ... ],    # координаты узлов сетки
  [(ID00, ID10, ID20, ID30), ... ],  # ID узлов тетраэдров
  [(ID00, ID10, ID20), ... ],        # ID узлов граней
  ['name1', set(nodesIDs), set(tetrsIDs), set(facesIDs)],
  # описание групп: имя, ID узлов, тетраэдров, граней
]
```

Для экспорта данных в текстовый формат необходимо вызвать в Salome скрипт salome2matlab.py, при этом в отдельной поддиректории будет сформирован аналогичный набор текстовых файлов с координатами узлов, ID тетраэдров и граней, а также разбивкой ID по группам.

Трехмерная равномерная картезианская сетка, каждая ячейка которой разбита на пять тетраэдров, может быть сгенерирована в формате .pickle при помощи скрипта

```
create-cube-mesh.py Nx,Ny,Nz Lx,Ly,Lz filename.pickle
```

принимающего через аргументы командной строки размеры сетки в ячейках, линейные размеры сетки и имя файла.

Выгруженный из Salome .pickle-файл с сеткой автоматически загружается в Python при создании экземпляра класса GeoMechModel, объявленном в модуле geomech:

```
from geomech import *
M = GeoMechModel('mesh.pickle'),
```

при этом экземпляр C++-класса `Model` находится в поле `M.core`.

3 Задание физических параметров модели, начальных и граничных условий

Для задания физических параметров для каждой подобласти необходимо вызвать метод

```
M.set_phys('phys_1',     # имя подобласти
           rhoS0=2.5e3,  # плотность скелета при PS0 [кг/м^3]
           PS0=1e5,      # давление при котором задается rhoS0 [Па]
           comprS0=1e-9, # сжимаемость скелета [1/Па]
           E=3.7E+9,     # модуль Юнга [Па]
           nu=0.2,       # дренированный к-т Пуассона
           # параметры Био
           b=0.7,        # коэффициент Био
           M=2.5E+8,     # модуль Био [Н/м^2]
           # свойства жидкости
           mu=1e-3,      # вязкость [Па*с]
           rhoF0=1e3,    # плотность флюида [кг/м^3] при PF0
           PF0=1e5,      # давление при котором задается rhoF0 [Па]
           comprF0=1e-6, # сжимаемость флюида [1/Па]
           g=vec(0.,0.,0.), # гравитация
           k=2.4E-15,    # проницаемость [м^2]
           phi=.19       # пористость
           )
```

Здесь `vec(0.,0.,0.)` создает трехмерный вектор библиотеки `aiwlib`.

Далее все значения задаются либо в виде числа, либо в виде строки. Строка трактуется как выражение языка `Python`, которое вычисляется функцией `eval` и может включать переменные `r0`, `r1`, `r2` — координаты узла.

Можно устанавливать значения отдельных полей для группы узлов (подобласти) — это трактуется как начальные условия, аналогичная операция может быть проведена в процессе расчета для задания источников, временных зависимостей (например, роста давления в скважине) и т.д.

```
M.set_fields('phys_1',       # имя подобласти
             p=0,            # давление
             u0=0,u1=0,u2=0) # компоненты перемещений
```

Для задания граничных условий в узлах необходимо вызвать метод

```
M.set_bound_nodes('bound_z0',      # имя граничной группы
                  p=1e6,           # давление
                  u0=0,            # компонента перемещения
                  )
```

или

```
M.set_bound_nodes('bound_z0',      # имя граничной группы
                  L0=(1, 0, 0, 0), # матрица линейных уравнений
                  L1=(0, 1, 0, 0), # в узле на перемещение вида
                  L2=(0, 0, 1, 0)  #     a*u0+b*u1+c*u2=d
                  )                # задаются как LX=(a, b, c, d)
```

или

```
M.set_bound_nodes_f('bound_z0',     # имя граничной группы
                    user_node_func  # функция пользователя
                   )
```

здесь `user_node_func` — некоторая функция на языке `Python`, определяемая пользователем. Функция принимает два параметра — экземпляр `C++`-структуры `Node` и экземпляр `C++`-структуры `BoundNode`, и должна произвести настройку (задать поля) экземпляра `BoundNode`.

Структуры `Node` и `BoundNode` описаны далее.

Для задания граничных условий на гранях необходимо вызвать метод

```
M.set_bound_faces('bound_z0', # имя граничной группы
                  sigma=1e3,  # напряжение на грани
                  flux=1.5,   # поток флюида через грань
                  )
```

здесь можно задать либо поток, либо напряжение, либо и поток и напряжение.

Можно также вызывать

```
M.set_bound_faces_f('bound_z0',      # имя граничной группы
                    user_face_func  # функция пользователя
                   )
```

`user_face_func` — некоторая функция на языке `Python` определяемая пользователем. Функция принимает два параметра — экземпляр `C++`-структуры `Node` и экземпляр `C++`-структуры `Face`, и должна произвести настройку (задать поля) экземпляра `Face`. Структура `Face` описана далее.

Вызов методов задания граничных условий может производиться в процессе расчета, но при этом заданные ранее граничные условия для группы сбрасываются.

4 Задание начального положения и движения трещины

Для задания зародыша трещины в форме диска используется вызываемый из Python метод вычислительного ядра M.core

```
Model::add_frac_disk(Vec<3> center, // центр диска
                     Vec<3> normal, // нормаль к плоскости диска
                     double R       // радиус диска
                     )
```

где Vec<3> — трехмерные вектора библиотеки aiwlib, которые могут быть созданы при помощи функции vec(x,y,z).

Для задания зародыша трещины в форме эллипса используется вызываемый из Python метод вычислительного ядра M.core

```
Model::add_frac_ellipse(Vec<3> center, // центр эллипса
                        Vec<3> normal, // нормаль к плоскости
                        double R,      // полуось вдоль Y
                        double eps     // эксцентриситет
                        )
```

Для задания зародыша трещины в форме сегмента сферы используется вызываемый из Python метод вычислительного ядра M.core

```
Model::add_frac_sph(Vec<3> center, // центр сферы
                    Vec<3> normal, // направление в центр сегмента
                        double R,  // радиус сферы
                        double th  // угол сегмента
                        )
```

Вызов этих методов задает проекции на трещину, нормали на фронте трещины и типы узлов сетки. При этом играют роль следующие параметры:

```
M.core.cloud_Rf          # радиус сферы, в которой берутся точки
                         # при пересчете проекции cp_front
M.core.cloud_R_max       # макс. расстояние от узла до трещины
M.core.cloud_R_xfem      # макс. расстояние от узла до трещины
                         # в XFEM (должно быть <=Rf_max)
M.core.epsilon_f = 1e-14 # критерий обусловленности при расчете
                         # проекции на фронт трещины
M.core.cloud_R_tips      # макс. расстояние до трещины
                         # в тетраэдрах, если 0, то игнорируется
```

Для движения трещины в процессе расчета необходимо вызывать из Python метод вычислительного ядра M.core

```
Model::move_frac(const BaseFracVelFunc &vf, # поле скоростей
                 double dt                  # шаг по времени
                 )
```

где BaseFracVelFunc — абстрактный базовый класс–функтор поля скоростей

```
struct BaseFracVelFunc{
virtual Vec<3> operator ()(const Vec<3> &r) const = 0;
virtual ~BaseFracVelFunc(){}
};
```

принимающий единственный аргумент — координату точки, в которой необходимо вычислить скорость.

Изначально в программном комплексе есть следующие аналитические варианты полей скоростей для тестирования:

```
struct SphFracVelFunc: public BaseFracVelFunc {
Vec<3> r0; // центр сферы
virtual Vec<3> operator ()(const Vec<3> &r) const;
};
```

создает поле единичной скорости направленной из точки r0.

```
struct CylFracVelFunc: public BaseFracVelFunc {
Vec<3> r0;     // точка через которую проходит ось цилиндра
Vec<3> n;      // направление оси цилиндра, |n|=1
double v_n;    // скорость вдоль оси цилиндра
double v_r;    // скорость убегания от оси цилиндра
double omega;  // угловая скорость вращения
               // вокруг оси цилиндра

virtual Vec<3> operator ()(const Vec<3> &r);
};
```

Для реальных расчетов используется поле скоростей

```
struct RealVelFunc: public BaseFracVelFunc;
```

создаваемое методом вычислительного ядра M.core

```
RealVelFunc Model::get_real_vel_func();
```

Это поле скоростей вычисляет направление движения трещины на основе анализа поля напряжений в окрестностях фронта трещины.

При любом вызове метода move_frac пересчитываются проекции, базисы на фронте трещины и нормали к поверхности трещины.

5 Управление расчетом

Управление расчетом производится с помощью задания следующих параметров:

```
# тип граничных условий на трещине по упругости
M.core.fracbcinfo.neuman_on_frac_elast = True
M.core.fracbcinfo.dirichlet_on_frac_elast = False
# тип граничных условий на трещине по фильтрации
M.core.fracbcinfo.neuman_on_frac_filt = False
M.core.fracbcinfo.dirichlet_on_frac_filt = True

# параметр метода штрафа для задания условий Дирихле
M.core.fracbcinfo.on_frack_eps = 1.e-3

# параметры предобуславливателя GMM++
M.core.precondinfo.n_elast = 60
M.core.precondinfo.pivot_elast = 1.e-4
M.core.precondinfo.n_filt = 20
M.core.precondinfo.pivot_filt = 1.e-8
```

Цикл по времени выглядит следующим образом

```
for it in range(max_iters): # цикл по времени
    # расчет течения в трещине
    M.core.do_step_frac(dt_frac,1,File("err.dat","w"),
                                  File("frac-%03i.vtk"%it,"w"),
                                  File("all-%03i.vtk"%it,"w"))
    M.core.setUseSavedPrecond(False)
    M.core.do_xfem_step(1.) # шаг по XFEM
    M.core.dump2vtk(File("finish-%03i.vtk"%it, "w"))

    #движение трещины
    M.core.move_frac(0.2)
```

6 Структура данных вычислительного ядра

Данные хранятся на неструктурированной тетраэдрической сетке.

Широко используются структуры библиотеки `aiwlib` `Vec<D>` (D–мерный вектор) и `Matr<Y,X>` (матрица).

Физические поля, задаваемые в узлах сетки, перечислены в структуре

```
struct Fields{
    Vec<3> u;     // перемещения
    Vec<3> u_old; // перемещения на предыдущем слое
    Matr<3, 5> u_enr;     // значения перемещений для
    Matr<3, 5> u_enr_old; // разных степеней свободы XFEM
    double p;     // давление
    double p_old; // давление на предыдущем слое
    Matr<1, 5> p_enr;     // значения давления для
    Matr<1, 5> p_enr_old; // разных степеней свободы XFEM
    double pf;     // давление в трещине
    double pf_old; // давление в трещине на предыдущем слое
};
```

Для граничных узлов дополнительно создаются экземпляры структуры

```
struct BoundNode{
    std::string gname; // имя группы
    Matr<3,3>   M; // матрица и правая часть СЛАУ
    aiw::Vec<3> R; // задающей условие на перемещение
    int usage; // битовая маска задающая тип гран.условия,
               // биты [0,1,2] - условие Дирихле для u,
               // бит[3] - условие Дирихле для p
};
```

Для граней, на которых устанавливаются граничные условия, создаются экземпляры структуры

```
struct Face{
    std::string gname;  // имя группы
    Node* nodes[3];     // указатели на узлы грани
    aiw::Vec<3> sigma;  // напряжение
    double flux;        // поток флюида
    bool use_sigma, use_flux; // флаги, задающие тип условия
};
```

Для описания проекций на трещину и фронт трещины используется структура

```
struct ClosestProjection {
    int cp_usage = 0;  // маска, определяющая состояние узла
        //  1 - используется cp_frac, 2 - используется cp_front
        //  4 - is_cut,  8 - is_tip
        // 16 - граничная точка в облаке задающем трещину
```

```
        // 32 - cp_front==cp_frac, т.е. точка за пределами
        //      площади трещины

    Tetr *cp_tetr;  // указатель на инцидентный узлу тетраэдр,
                    // в который попадет вектор cp_frac

    Vec<3> cp_frac;  // проектор на поверхность трещины
    Vec<3> cp_front; // проектор на фронт трещины
    Vec<3> nf_phi;   // нормаль к поверхности трещины на фронте
    Vec<3> nf_psi;   // нормаль к фронту (в плоскости трещины)
    Vec<3> ns;       // нормаль к поверхности к трещины

    // классификация узлов
    inline bool is_cp() const;  // есть ли актуальная проекция
    inline bool is_cut() const; // тип узла
    inline bool is_tip() const; // тип узла
    bool is_cp_front() const;   // все проекции попадают на фронт
    bool is_frac_cloud_bound() const; // граничная точка в облаке,
                                      // задающем трещину
};
```

В итоге, для узлов сетки используется структура

```
struct Node: public Fields, public ClosestProjection {
    ClosestProjection oldCP; // проекция на предыдущем слое

    int ID;    // ID узла (позиция в общей таблице)
    Vec<3> r;  // координата узла
    BoundNode *bound; // граничные условия в узле (или nullptr)

    Matr<3, 1, int> base_dof_place;
    Matr<3, 5, int> dof_place;
    Matr<1, 1, int> base_dof_place_press;
    Matr<1, 5, int> dof_place_press; // переменные со
                                     // степенями свободы в
                                     // обогащенных узлах
    int n_enr;

    double w_frac;      // раскрытие трещины
    double w_frac_old;  // раскрытие трещины на предыдущем слое
};
```

Для элементов сетки (тетраэдров) используется структура, в которой узлы и грани тетраэдра имеют согласованную нумерацию — i-му узлу отвечает противолежащая i-я грань

```
struct Tetr{
    Node *nodes[4]; // узлы сетки, образующие тетраэдр
    PhysPar *phys;  // физические параметры тетраэдра

    void init();       // рассчитывает все поля, определенные ниже
    Vec<3> n[4];       // внутренние нормали к граням
    Tetr *tetrs[4];    // соседние тетраэдры
    double _height[4]; // обратные высоты для расчета значений
                       // базисных функций
    double V;          // объем тетраэдра

    // проверяет, попала ли точка внутрь тетраэдра
    inline bool check_in(const Vec<3>& r) const;

    // возвращает веса интерполяции
    // для расчета значения внутри тетраэдра
    inline Vec<4> weights(const Vec<3> &r) const;

    // возвращает пару цилиндрических координат
    // (r, theta) в точке point
    // (внутри тетраэдра) относительно фронта трещины
    Vec<2> theta_r(const Vec<3> point) const;
};
```

Класс модели, инкапсулирующий все данные и методы работы с ними

```
class Model{
    std::vector<Node> nodes;  // массив узлов
    std::vector<Tetr> tetrs;  // массив тетраэдров

    std::vector<PhysPar> phys_par;    // физические параметры
    std::list<Face> bound_faces;      // ГУ на гранях
    std::list<BoundNode> bound_nodes; // ГУ в узлах
public:
    Model();
    ~Model();

    // задает число подобластей с физическими параметрами
```

```
void init_phys(int phys_sz);

// задает число узлов сетки
void init_nodes(int nodes_sz);

// добавляет один тетраэдр
void add_tetr(Ind<4> nodeIDs, int physID);

// инициализирует соседей для тетраэдров и таблицу
// пограничных граней, вызывается  после создания всей сетки
void init_tetrs_nb();

// сбрасывает граничные условия для узлов и граней
void clear_bound_nodes(std::string gname);
void clear_bound_faces(std::string gname);

// устанавливает граничные условия
void add_bound_node(int nodeID, BoundNode bnode);
void add_bound_face(Ind<3> nodeIDs, Face face);

// обеспечивает возможность изменения физических
// параметров в процессе расчета
PhysPar& get_phys(int ID);

// сборка упругой части матрицы системы уравнений
void assambleElasticPart(RowMatr& M, std::vector<double>
                         &B, bool add_undrained);
void assambleElasticXFEMPart(RowMatr& M, std::vector<double>
                         &B, bool add_undrained);

// сборка фильтрационной части матрицы системы уравнений
void assambleFiltrationPart(RowMatr& M, std::vector<double>
                         &B, const double tau);
void assambleFiltrationXFEMPart(RowMatr& M, RowMatr& Mpr,
                  std::vector<double>& B, const double tau);

// сборка фильтрационной части матрицы
// системы уравнений течения в трещине
void assambleFracturePart(RowMatr& M, RowMatr& E,
                  std::vector<double>& B, const double tau);
```

```
// считает одну итерацию между группами уравнений
// упругости и фильтрации
void do_step(double tau);

// считает несколько итераций по времени для течения в трещине
void do_step_frac(double tau, int max_cycles,
                IOstream &Serr, IOstream &Sfrac, IOstream &S);

// итерирует уравнения упругости и фильтрации до сходимости
void do_step_it(double tau, bool if_undrained);

// считает одну недренированную итерацию между группами
// уравнений упругости и фильтрации
void do_undarined_step(double tau);

// считает один шаг по времени для уравнений упругости
void do_elast_step(double tau, bool add_undrained);

// считает один шаг по времени для уравнений упругости  XFEM
void do_elast_xfem_step(double tau, bool add_undrained);

// считает одну итерацию между группами уравнений
// упругости и фильтрации для XFEM
void do_xfem_step(double tau);

// считает один шаг по времени для уравнений фильтрации
void do_filt_step(double tau);

// считает один шаг по времени для уравнений фильтрации с XFEM
void do_filt_xfem_step(double tau);

// режим использования сохраненного предобуславливателя
void setUseSavedPrecond(bool flag);

// задает максимальное число итераций
void setMaxNlIters(int nit);

// сброс результатов в формат VTK
void dump2vtk(IOstream &S);
void dumpfrac2vtk(IOstream &S);
```

```
    double cloud_Rf;     // радиус сферы, в которой берутся точки
                         // при пересчете проекции cp_front
    double cloud_R_max;  // макс. расстояние от узла до трещины
    double cloud_R_xfem; // макс. расстояние от узла до трещины
                         // в XFEM (<=Rf_max)
    int cloud_R_tips=0;  // макс. расстояние от фронта
                         // в тетраэдрах (дискретный критерий)
    double source_r;     // радиус источника в трещине
    double epsilon_f;    // критерий обусловленности при расчете
                         // проекции на фронт трещины

    // добавляет различные зародыши трещин
    void add_frac_disk(Vec<3> center, Vec<3> normal, double R);
    void add_frac_sph(Vec<3> center, Vec<3> normal, double R,
                      double theta_max);
    void add_frac_ellipse(Vec<3> center, Vec<3> normal,
                          double R, double a);
};
```

7 Пример расчета

В ранее опубликованных работах были представлены результаты валидационных и верификационных расчетов [5, 8, 11–14, 16, 18], представленная ниже задача приведена в иллюстративных целях.

Схема постановки задачи представлена на рисунке 1. В начальный момент времени трещина представляла собой круглый диск радиуса 10 м, расположенный в центре расчетной области, которая имела форму куба с длиной ребра 40 м. Для расчета использовалась равномерная сетка из тетраэдров. Во внешней области создавалось неоднородное поле напряжений, согласованное с трещиной. На нижней поверхности расчетной области задавалось постоянное перемещение вдоль оси $\mathcal{O}y$: $\mathbf{u} = (0, 0.1, 0)$, тогда как верхняя поверхность кубика была зафиксирована. В качестве граничных условий для внутренней задачи течения жидкости в трещине задавалось известное давление в центре трещины, а на ее берегах — условие непротекания. Для определения направления развития трещины в точках ее фронта использовался предложенный ранее критерий разрушения. Для этого на каждом временном шаге после решения связанной задачи течения в трещине и пороупругости определялось направление роста трещины. Далее геометрия трещины пересчитывалась с использованием соответствующих алгоритмов расчета эволюции трещины.

Деформация расчетной области из-за граничных условий представлена на рисунке 2. На рисунке 3 видна несимметричность поля главных макси-

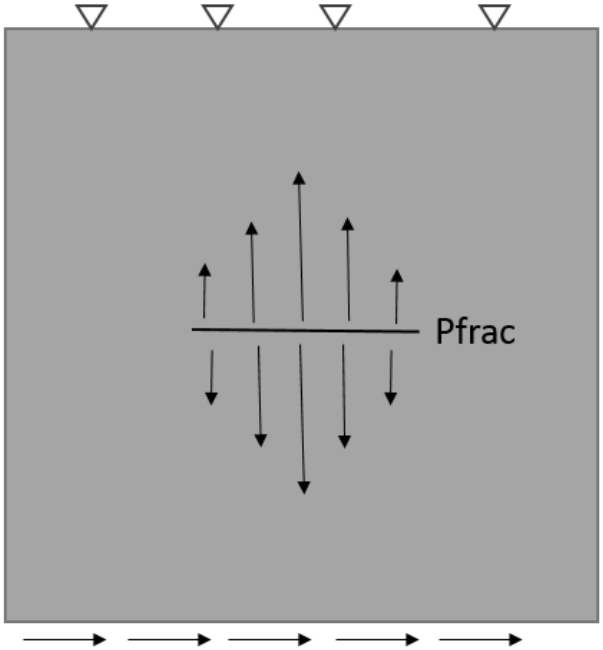

Рис. 1. Схематичный вид задачи.

мальных напряжений, за счет чего происходит искривление трещины.

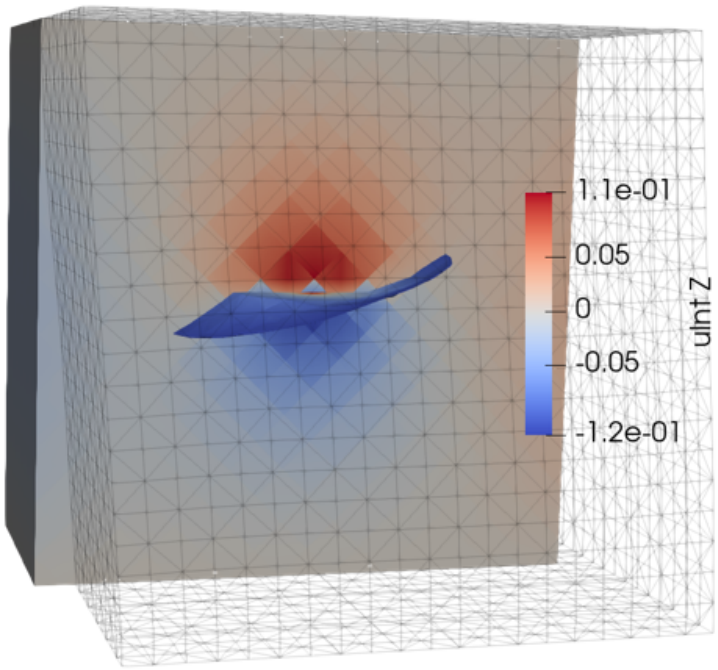

Рис. 2. Деформация расчетной области.

Распределение главных максимальных напряжений и поля давления после 1-го (a), 5-го (b), 30-го (c) и 70-го (d) шагов по времени представлено на рисунке 3.

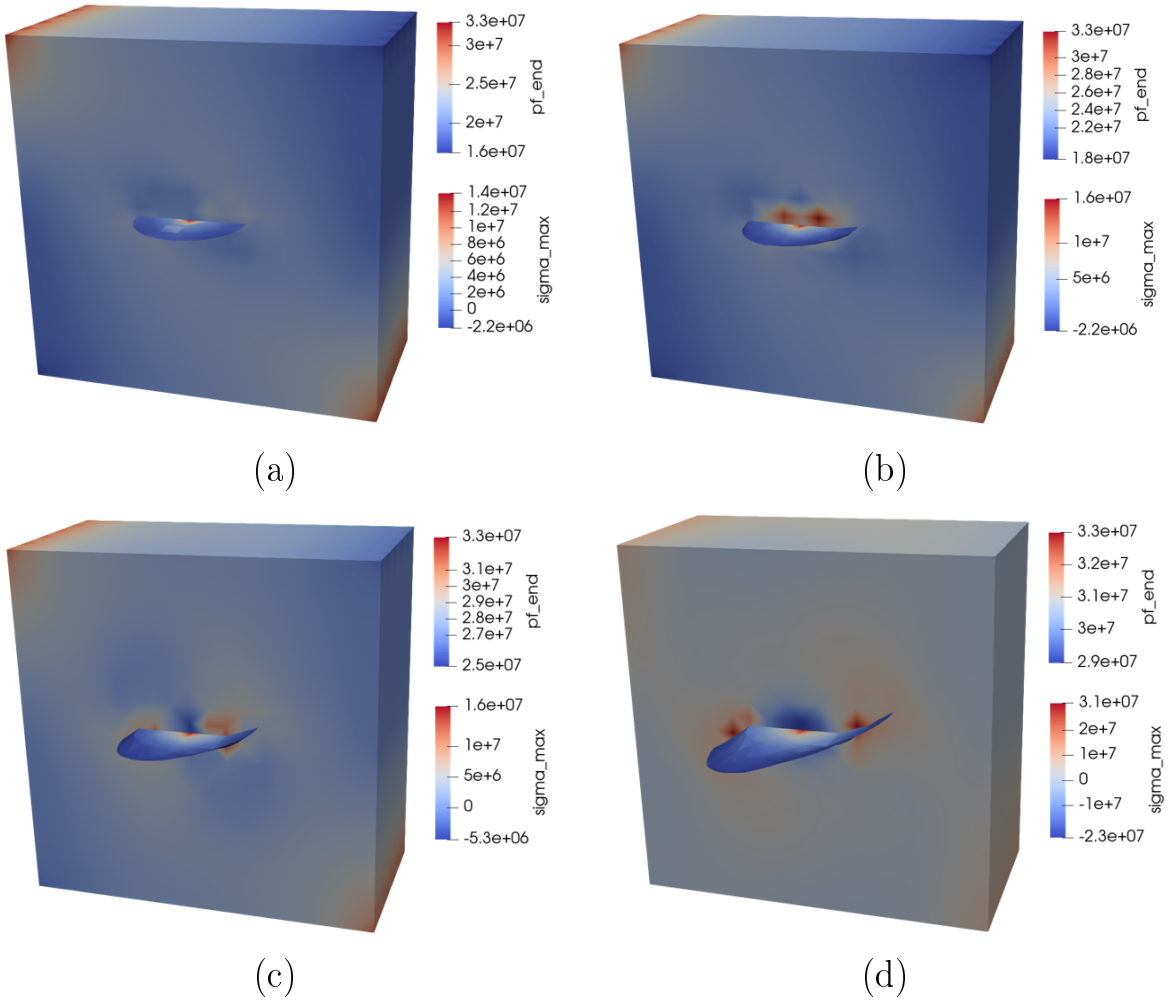

Рис. 3. Распределение главных максимальных напряжений и поля давления после 1-го (a), 5-го (b), 30-го (c) и 70-го (d) шагов по времени.

Список литературы

[1] Борисов В.Е., Зипунова Е.В., Иванов А.В., Критский Б.В., Савенков Е.Б. Свидетельство о государственной регистрации программы для ЭВМ № 2019666497 от 11.12.2019 г. «Программный комплекс HFrac3D++ для анализа процесса гидроразрыва пласта в рамках расширенной постановки на высокопроизводительных вычислительных системах».

[2] Экономидес М., Олини Р., Валько П. Унифицированный дизайн гидроразрыва пласта. От теории к практике. М.: Институт компьютерных исследований, 2007. 236 С.

[3] Savenkov E.B., Borisov V.E. A mathematical model for hydraulic fracture propagation in three dimensional poroelastic medium. PNRPU Mechanics Bulletin, 2018, no. 1, pp. 5-17. DOI: 10.15593/perm.mech/2018.1.01

[4] Иванов А.В., Савенков Е.Б. Моделирование и визуальное представление динамики поверхности с подвижным краем на стационарной неструктурированной сетке // Научная визуализация. 2017, том 9, № 2, с. 64-81.

[5] Савенков Е.Б., Борисов В.Е., Критский Б.В. Представление поверхности с помощью проекции ближайшей точки в методе X-FEM // Математическое моделирование, 2019, том 31, № 6, с. 18–42. https://doi.org/10.1134/S0234087919060029.

[6] Савенков Е.Б., Борисов В.Е., Критский Б.В. Алгоритм метода X-FEM с представлением поверхности трещины на основе проекции ближайшей точки // Препринты ИПМ им. М.В.Келдыша. 2018. № 42. 36 с. https://doi.org/10.20948/prepr-2018-42.

[7] Рамазанов М.М., Критский Б.В., Савенков Е.Б. Формулировка J-интеграла для модели пороупругой среды Био // Инженерно-физический журнал, Т. 91, № 6. 2017. с. 1677-1684.

[8] Борисов В.Е., Иванов А.В., Критский Б.В., Меньшов И.С., Савенков Е.Б. Численное моделирование задач пороупругости // Препринты ИПМ им. М.В. Келдыша. 2017. № 81. 36 с. https://doi.org/10.20948/prepr-2017-81.

[9] Ramazanov M., Borisov V., Kritsky B., Savenkov E. Fracture growth criterion for poroelastic media // AIP Conference Proceedings, 2018, 2051, 020250. 2018.

[10] Рамазанов М.М., Савенков Е.Б. Критерий развития трещин в пороупругой среде // Вестник МГТУ им. Баумана. Сер. Естественные науки, № 5(80). с. 65-82. 2018.

[11] Borisov V., Ivanov A., Kritskiy B., Menshov I., Ramazanov M., Savenkov E. Fully coupled numerical simulation techniques for 3D hydraulic

fracturing // Journal of Physics: Conference Series, IOP Publishing Ltd, 2018, Vol. 1141 № 1. https://doi.org/10.1088/1742-6596/1141/1/012085.

[12] Borisov V.E., Ivanov A.V., Kritsky B.V., Menshov I.S., Savenkov E.B., Trimonova M.A., Turuntaev S.B., Zenchenko E.V. Analysis of Poroelastic Laboratory Experiments Using Numerical Simulation Techniques // Physical and Mathematical Modeling of Earth and Environment Processes. Springer Proceedings in Earth and Environmental Sciences. pp. 244-252. 2018.

[13] Borisov V.E., Ivanov A.V., Ramazanov M.M., Savenkov E.B. Poroelastic Hydraulic Fracture Simulation Using X-FEM/CPP Approach // Physical and Mathematical Modeling of Earth and Environment Processes. Springer Proceedings in Earth and Environmental Sciences. pp. 323-333. 2018.

[14] Borisov V.E., Zenchenko E.V., Kritsky B.V., Savenkov E.B., Trimonova M.A., Turuntaev S.B. Numerical Simulation of Laboratory Experiments on the Analysis of Filtration Flows in Poroelastic Media // Herald of the Bauman Moscow State Technical University, Series Natural Sciences, 2020, № 1 (88), pp 16-31. https://doi.org/10.18698/1812-3368-2020-1-16-31.

[15] Савенков Е.Б. Решение уравнений в частных производных на поверхностях: обзор алгоритмов // Препринты ИПМ им. М.В. Келдыша. 2020. № 5. 18 с. https://doi.org/10.20948/prepr-2020-5.

[16] Зипунова Е.В., Савенков Е.Б. Применение метода проекции ближайшей точки для решения уравнений гидродинамики в приближении смазочного слоя // Препринты ИПМ им. М.В. Келдыша. 2020. № 10. 32 с. https://doi.org/10.20948/prepr-2020-10.

[17] Савенков Е.Б. Конечноэлементный вариант метода проекции ближайшей точки для решения уравнений на поверхностях с краем // Препринты ИПМ им. М.В. Келдыша. 2020. № 8. 36 с. https://doi.org/10.20948/prepr-2020-8.

[18] Zipunova E., Ivanov A., Savenkov E. Application of the closest point projection method to solution of Reynold's lubrication equations on evolving surfaces // MATHEMATICA MONTISNIGRI, Vol. XLVII (2020).

[19] Иванов А.В. Использование библиотеки aiwlib на примере численного моделирования стохастического резонанса // Препринты ИПМ им. М.В.Келдыша. 2018. № 89. 30 с. https://doi.org/10.20948/prepr-2018-89.

[20] Иванов А.В., Хилков С.А. Библиотека aiwlib — инструмент для создания приложений численного моделирования, визуализации и анализа результатов // Научная визуализация, 2018, т. 10, № 1, с. 110-127. 2018.