1 Постановка задачи

1.1 Цели и задачи модификации

- обеспечить по итогам текущей модернизации кода удобство интегрирования новых реологических моделей МСС, новых численных алгоритмов, выполняющих расчёт этих моделей;
- сделать код более структурированным, более понятным и простым для ознакомления и будующих расширений.

1.2 Постановка задачи

На данный момент задача в самой общей постановке представляет собой систему N квазилинейных гиперболических уравнений в частных производных:

$$\frac{\partial \vec{u}}{\partial t} + \mathbf{A}_x \frac{\partial \vec{u}}{\partial x} + \mathbf{A}_y \frac{\partial \vec{u}}{\partial y} + \mathbf{A}_z \frac{\partial \vec{u}}{\partial z} = \vec{f}, \tag{1.1}$$

где $\mathbf{A}_x = \mathbf{A}_x(\vec{u}, \vec{r}, t)$, $\mathbf{A}_y = \mathbf{A}_y(\vec{u}, \vec{r}, t)$, $\mathbf{A}_z = \mathbf{A}_z(\vec{u}, \vec{r}, t)$, $\vec{f} = \vec{f}(\vec{u}, \vec{r}, t)$.

Также движение сетки описывается уравнением:

$$\frac{\partial \vec{r}}{\partial t} = \vec{v}. \tag{1.2}$$

Кинетика разрушений в общем случае описывается системой эволюционных уравнений:

$$\frac{\partial \vec{\chi}}{\partial t} = \vec{F}(\vec{r}, t, \vec{u}, \vec{\chi}), \tag{1.3}$$

где $\vec{\chi}$ – так называемые внутренние параметры, характеризующие внутреннюю структуру материала (пористость, размер пор, повреждённость, параметр упрочнения и пр.).

Уравнение (1.2) подразумевает замену дифференциального оператора разностным нужного порядка.

Уравнения (1.3) могут иметь различный вид. Пока у нас простейшие модели – решение этих(этого) уравнения не составит труда. Дальше будем думать. Вместо этих уравнений могут быть различные критерии.

2 Модель

2.1 DeformationModel

Для пущей структурированности и удобства имплементации различной физики предлагаю ввести класс **DeformationModel**.

Переменная **DeformationModelName**.

В этот класс предлагаю ввести следующие сущности:

- **Material** material – материал,

- **typedef Node**'a – тип **Node**'a ассоциированного с расчитываемой моделью, далее при создании сетки будут использоваться **Node**'ы этого типа,
- **MatrixSetter** matrixSetter – заполняет матрицы текущей модели,
- **InGomogeniousSetter** inGomogeniousSetter – заполняет правую часть уравнений,

Варианты моделей:

- Elasticity,
- ElasticityFiniteStrains,
- NonLinearElasticity,
- Plasticity,
- ThermoElasticity.

2.2 FailureModel

По аналогии **FailureModel**.

Переменная **FailureModelName**. Переменная **FailureModelType** – = **discrete** *or* = **continuum**.

Тут могла бы быть ВАША модель разрушения с обслуживающим персоналом:

- **Setter** – если нужен
- **failureConstants** – любые критические параметры разрушения,
- **failureSolverPtr** – ссылочка на используемую в методе сущность, отвечающую за разрушение.

И прочее.

Примерные варианты реализаций(дочерние классы):

- FailureDiscrete,
  - MisesCriterion,
  - MohrCoulombCriterion,
  - HashinCriterion,
  - TsaiHillCriterion,
  - TsaiWuCriterion;
- FailureContinuum,
  - Failure1Order,
  - Failure2Order;

Я представляю это себе так: **Engine** создаёт **DeformationModel**, **FailureModel** и **GCMsolver**, указанные в таске. Причём у нас в программе есть "библиотека моделей"(со всеми константами) и мы по сути выбираем её или создаём свою, указывая все необходимые параметры.

3 Метод

3.1 GCMsolver

Родительский класс – **GCMsolver**. Его предлагаю сделать обобщённым.

Содержит переменные **spaceOrder**, **timeOrder**, **bufNodes**. А также **bufCoefficients** – наклоны характеристик, которые уточняются в процессе решения.

Далее, флаг **HomogeniousSystem**. Необходимы отдельно метод для однородных систем и отдельно для неоднородных систем.

Далее, поскольку у нас сеточно-характеристический метод и больше никого (появится FEM будет FEMsolver) разумно определить сюда базовые кирпичи GCM'a.

- **CrossPointFinder** crossPointFinder - ищет точку пересечения характеристики,
- **ModelPtr** – ссылка на текущую модель,
  - **MatrixSetter** matrixSetter – заполняет матрицы,
  - **InGomogeniousSetter** inGomogeniousSetter – заполняет правую часть уравнений,
- **Decomposer** decomposer – разлагает матрицы,
- **InGomogeniousSolver** ingomogeniousSolver – решает уравнение на перенос инвариантов $\vec{\xi}$:
$$\frac{\partial \vec{\xi}}{\partial t} = \mathbf{\Omega} \vec{F}, \tag{3.1}$$
- **Mesh** mesh *or* **MeshPtr** meshPtr – сетка(её область – шаблон) с обслуживающим персоналом:
  - **Interpolator** interpolator – интерполирует значения инвариантов в точке(из **Mesh**'a),
  - **MeshMover** meshMover – двигает сетку(из **Mesh**'a);
- **FailureSolver** failureSolver – corrector или что-то другое, решающее уравнения (1.3) и модифицирующий материал, решение, добавляющий трещину, изменяющий область трещины и пр.

Реализации второго порядка идёт либо с использованием **InterpolatorLimiter** interpolatorLimit либо отдельная реализация с гибридной схемой, использующая сущности **Interpolator1Order** interpolator1Order, **Interpolator2Order** interpolator2Order. Если кто знает TVD, ENO, аппроксимационную вязкость или любую вязкость – милости просим.

Основной метод **doNextTimeStep** – выполняет следующий шаг по времени.

Далее, **внимание, осторожно** – методы класса:

- VolumeCalculator,
- BorderCalculator,
- ContactCalculator.

Осторожно, потому, что не думаю, что **это** – лучшая идея внедрения всех калькуляторов в класс **GCMsolver**. Не лучшая потому, что логичнее их оставить отдельными сущностями – классами, но тогда их нужно включать в верхний список, а остальное включать в них. Незнаю как лучше это организовать. **ПАМАГИТЕ!**

Тут возможно стоит ещё что-то добавить про параллельность. Возможно много, потому что в моём представлении, реализации данного класса различны для последовательной и параллельной версии исполнения программы.

3.2 CrossPointFinder

Родительский класс. Производные классы – реализации конкретного порядка и просто разные реализации. В зависимости от реализации получает набор нодов и реологических параметров

Переменная **spaceOrder**.

Флаг **linearCharateristics** – если характеристики – прямые линии, в соответствии с моделью тут следует использовать первый порядок, дающий точное положение. Такакя же ситуация как и с **HomogeneusSystem**. Тут два варианта:

- отдельно метод второго порядка для прямых характеристик, с указанием нужного метода в модели;
- На этапе инициализации **Model** программа понимает, что нужно использовать линейный **Finder** для метода второго порядка.

Предполагаемые реализации:

- CrossPointFinder1Order;
- CrossPointFinder2Order.

3.3 Interpolator

Насколько я понимаю сущность из **Mesh**'а.

Неоходимы отдельные реализации классов:

- Interpolator1Order;
- Interpolator2Order;
  - Interpolator2Order;

- InterpolatorLimiter,
    - * InterpolatorLimiterMinMax,
    - * InterpolatorLimiterLinear.

3.4 MatrixSetter

Получает **curNode** и заполняет матрицы $\mathbf{A}_x$, $\mathbf{A}_y$, $\mathbf{A}_z$ в соответствии с моделью.

3.5 InGomogeniousSetter

Получает **curNode** и заполняет правую часть неоднородного уравнения $\vec{F}$ в соответствии с моделью.

3.6 MeshMover

Переменная **timeOrder**. Решает уравнение (1.2) в соответствии с порядком:

- MeshMover1Order;
- MeshMover2Order.

Получает **curNode** и двигает его в соответствии с решением.

3.7 Decomposer

Реализации:

- NumericalDecomposer;
- AnalyticalDecomposer,
    - IsotropicDecomposer,
    - GeneralAnalyticalDecomposer.

3.8 InGomogeniousSolver

Переменная **timeOrder**.
Решаем уравнение на инварианты (3.1).
Предполагаемые реализации:

- InGomogeniousSolver1Order;
- InGomogeniousSolver2Order.

3.9 FailureSolver

Переменная **timeOrder**. Переменная **FailureModelType** – = **discrete** *or* = **continuum**.
Решаем уравнение на инварианты (1.3) или запускаем корректоры.
Примерные варианты реализаций:

- FailureDiscrete,
  - MisesCriterionCorrector,
  - MohrCoulombCriterionCorrector,
  - HashinCriterionCorrector,
  - TsaiHillCriterionCorrector,
  - TsaiWuCriterionCorrector;
- FailureContinuumSolver,
  - Failure1OrderSolver,
  - Failure2OrderSolver;

Далее корректирует упругие постоянные, решение и пр.