СУРС на тему “Метод конечных элементов”

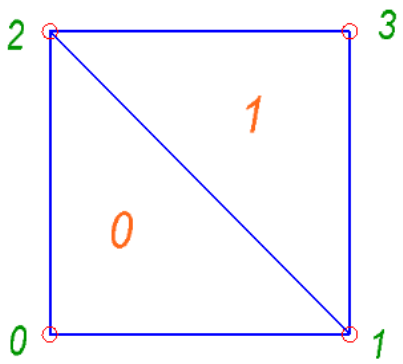
Метод конечных элементов (*МКЭ*) — это численный метод решения дифференциальных уравнений с частными производными, а также интегральных уравнений, возникающих при решении задач прикладной физики. Метод широко используется для решения задач механики деформируемого твёрдого тела, теплообмена, гидродинамики, электродинамики и топологической оптимизации.

Суть метода заключена в его названии. Область, в которой ищется решение дифференциальных уравнений, разбивается на конечное количество подобластей (элементов). В каждом из элементов произвольно выбирается вид аппроксимирующей функции. В простейшем случае это полином первой степени. Вне своего элемента аппроксимирующая функция равна нулю. Значения функций на границах элементов (в узлах) являются решением задачи и заранее неизвестны. Коэффициенты аппроксимирующих функций обычно ищутся из условия равенства значения соседних функций на границах между элементами (в узлах). Затем эти коэффициенты выражаются через значения функций в узлах элементов. Составляется система линейных алгебраических уравнений. Количество уравнений равно количеству неизвестных значений в узлах, на которых ищется решение исходной системы, прямо пропорционально количеству элементов и ограничивается только возможностями ЭВМ. Так как каждый из элементов связан с ограниченным количеством соседних, система линейных алгебраических уравнений имеет разрежённый вид (**Разрежённая матрица** — это матрица с преимущественно нулевыми элементами), что существенно упрощает ее решение.

Если говорить в матричных терминах, то собираются так называемые матрицы жёсткости (или матрица Дирихле) и масс. Далее на эти матрицы накладываются граничные условия (например, при условиях Неймана в матрицах не меняется ничего, а при условиях Дирихле из матриц вычеркиваются строки и столбцы, соответствующие граничным узлам, так как в силу краевых условий значение соответствующих компонент решения известно). Затем собирается система линейных уравнений и решается одним из известных методов.

Пример написания программы для МКЭ на языке программирования C++ с использованием математической библиотеки Eigen.

Создадим входной файл с полным описанием задачи. Для тестового входного файла, мы создадим самую простую сетку из возможных:



Первая строка будет описание свойств материала. Например, для стали, коэффициент Пуассона v = 0,3 и модуль Юнга E = 2000 МПа:

```
0.3 2000
```

Затем следует строка с количеством узлов и затем сам список узлов:

```
4
0.0 0.0
1.0 0.0
0.0 1.0
1.0 1.0
```

Затем следует строка с количеством элементов, далее список элементов:

```
2
0 1 2
1 3 2
```

Теперь, нам нужно задать закрепления, или как говорят, условия на границе первого рода. В качестве таких граничных условий мы будем фиксировать перемещения узлов. Фиксировать можно перемещения по осям независимо друг от друга, т.е. мы можем запретить перемещения по оси x или по оси y или по обоим сразу. В общем случае, можно задавать некоторое начальное перемещение, однако мы ограничимся лишь нулевым начальным перемещением. Таким образом, у нас будет список узлов с заданным типом ограничений на перемещение. Тип ограничения будут указаны номером:

- 1 — запрещено перемещение в направлении оси x
- 2 — запрещено перемещение в направлении оси y
- 3 — запрещено перемещение в обоих, x и y направлениях

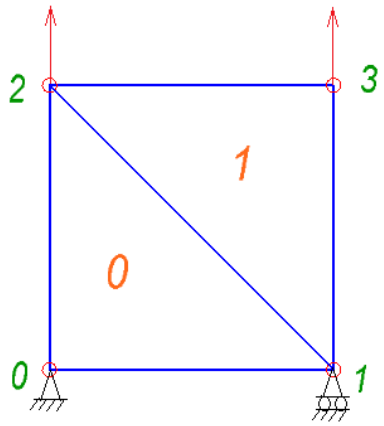
Первая строка определяет количество ограничений:

```
2
0 3
1 2
```

Затем, мы должны задать нагрузки. Мы будем работать только с узловыми силами. Нам нужен список с индексами узлов и соответствующими векторами сил. Первая строка определяет количество приложенных сил:

```
2
2 0.0 1.0
3 0.0 1.0
```

Можно легко видеть, что рассматриваемая задача, это тело квадратной формы, низ которого закреплен, а на верхнюю грань действуют растягивающие усилия.



Входной файл целиком:

```
0.3 2000
4
0.0 0.0
1.0 0.0
0.0 1.0
1.0 1.0
2
0 1 2
1 3 2
2
0 3
1 2
2
2 0.0 1.0
3 0.0 1.0
```

Для хранения данных, которые мы собираемся читать из входного файла, а также промежуточных данных, мы должны объявить свои структуры. Простейшая структура данных описывающая конечный элемент показана ниже. Она состоит из массива трех элементов, которые хранят индексы узлов образующих конечный элемент, а также матрицы B:

```
struct Element
{
    void CalculateStiffnessMatrix(const Eigen::Matrix3f& D, std::vector<Eige
n::Triplet<float> >& triplets);

    Eigen::Matrix<float, 3, 6> B;
    int nodesIds[3];
};
```

Еще одна простая структура которая нам понадобится — это структура для описания закреплений. Это простая структура состоящая из перечисляемого типа, который определяет тип ограничения, и индекса узла на который накладывается ограничение.

```
struct Constraint
{
    enum Type
    {
        UX = 1 << 0,
        UY = 1 << 1,
        UXY = UX | UY
    };
    int node;
    Type type;
};
```

Чтобы не усложнять, давайте определим некоторые глобальные переменные. Создавать какие-либо глобальные объекты — это всегда плохая идея, но для этого примера вполне допустимо. Нам понадобятся следующие глобальные переменные:

- Количество узлов
- Вектор с x-координатой узлов
- Вектор с y-координатой узлов
- Вектор элементов
- Вектор закреплений
- Вектор нагрузок

В коде, мы определим их следующим образом:

```
int                         nodesCount;
int                         noadsCount;
Eigen::VectorXf             nodesX;
Eigen::VectorXf             nodesY;
Eigen::VectorXf             loads;
std::vector< Element >      elements;
std::vector< Constraint >   constraints;
```

Чтение входного файла:

```
int main(int argc, char *argv[])
{
    if ( argc != 3 )
    {
        std::cout << "usage: " << argv[0] << " <input file> <output file>\
n";
        return 1;
    }

    std::ifstream infile(argv[1]);
    std::ofstream outfile(argv[2]);
```

В первой строке входного файла находятся свойства материала, прочитаем их:

```
float poissonRatio, youngModulus;
infile >> poissonRatio >> youngModulus;
```

Этого достаточно, чтобы построить матрицу упругости изотропного материала для плоско-напряженного состояния, которая определена следующим образом:

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} = [D] \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix}$$

$$[D] = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix}$$

Эти выражения получаются из закона Гука в общем виде, мы можем найти выражение для матрицы D из следующих соотношений:

$$\varepsilon_x = \frac{1}{E}\sigma_x - \frac{\nu}{E}\sigma_y$$
$$\varepsilon_y = -\frac{\nu}{E}\sigma_x + \frac{1}{E}\sigma_y$$
$$\gamma_{xy} = \frac{2(1+\nu)}{E}\tau_{xy}$$

У нас есть все исходные данные чтобы задать матрицу упругости:

```
Eigen::Matrix3f D;
D <<
    1.0f,           poissonRatio,   0.0f,
    poissonRatio,   1.0,            0.0f,
    0.0f,           0.0f,           (1.0f - poissonRatio) / 2.0f;

D *= youngModulus / (1.0f - pow(poissonRatio, 2.0f));
```

Далее, мы читаем список с координатами узлов. Сначала читаем количество узлов, затем задаем размер динамических векторов x и y. Далее, мы просто читать координаты узлов в цикле, строка за строкой.

```
infile >> nodesCount;
nodesX.resize(nodesCount);
nodesY.resize(nodesCount);

for (int i = 0; i < nodesCount; ++i)
{
    infile >> nodesX[i] >> nodesY[i];
}
```

Затем, мы читаем список элементов. Все то-же самое, читаем количество элементов, а затем индексы узлов для каждого элемента:

```
int elementCount;
infile >> elementCount;

for (int i = 0; i < elementCount; ++i)
{
    Element element;
    infile >> element.nodesIds[0] >> element.nodesIds[1] >> element.nodesIds
[2];
    elements.push_back(element);
}
```

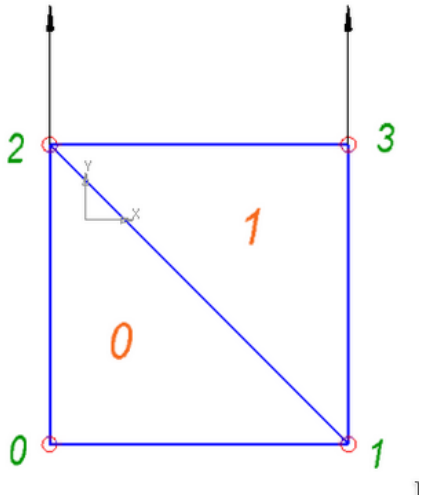
Далее, читаем список закреплений:

```
int constraintCount;
infile >> constraintCount;

for (int i = 0; i < constraintCount; ++i)
{
    Constraint constraint;
    int type;
    infile >> constraint.node >> type;
    constraint.type = static_cast<Constraint::Type>(type);
    constraints.push_back(constraint);
}
```

Наконец, нужно прочитать список нагрузок. Есть одна особенность связанная с нагрузкой, из-за которой мы будем представлять действующие нагрузки в виде вектора размерности двойного количества узлов. Рисунок ниже, иллюстрирует этот вектор нагрузки:

index	node	component	value
0	0	x	0.0
1		y	0.0
2	1	x	0.0
3		y	0.0
4	2	x	0.0
5		y	1.0
6	3	x	0.0
7		y	1.0



Таким образом, для каждого узла, у нас есть два элемента в векторе нагрузки, которые представляют x и y компоненты усилия. Таким образом, x-компонента усилия в определенном узле будет храниться в элементе с индексом id = 2 * node_id + 0, а y-составляющая усилия для этого узла будет храниться в элементе с индексом id = 2 * node_id + 1.

Сначала зададим размер вектора приложенных усилий вдвое больший, чем количество узлов, затем обнулим вектор. Читаем количество внешних сил и читаем их значения строка за строкой:

```
int loadsCount;
loads.resize(2 * nodesCount);
loads.setZero();

infile >> loadsCount;

for (int i = 0; i < loadsCount; ++i)
{
    int node;
    float x, y;
    infile >> node >> x >> y;
    loads[2 * node + 0] = x;
    loads[2 * node + 1] = y;
}
```