

Содержание

Введение	5
Часть 1. Постановка задачи	8
1.1 Применимость программы	8
1.2 Выбор метода расчета	9
1.2.1 Метод конечных разностей (МКР)	10
1.2.2 Метод граничных элементов (МГЭ)	10
1.2.3 Метод конечных элементов (МКЭ)	11
Часть 2. Технический проект	12
2.1 Алгоритм автоматической генерации конечно–элементной сетки	12
2.1.1 Обзор методов	12
2.1.2 Алгоритм Рапперта	14
2.1.3 Модификация алгоритма Рапперта	16
2.1.4 Алгоритм построения первичного разбиения	22
2.2 Расчетная часть программы.	25
2.2.1 Выбор конечного элемента	26
2.2.2 Алгоритм перенумерации узлов	29
2.2.3 Алгоритм поиска начального узла нумерации	30
2.2.4 Процедуры собственно перенумерации узлов	35
2.2.5 Оценка эффективности перенумерации узлов	38
2.2.6 Выбор схемы хранения и решения системы уравнений	39
2.2.7 Реализация профильного метода хранения	39
2.2.8 Заполнение глобальной матрицы жесткости	43
2.2.9 Метод решения для профильной схемы	44
2.2.9 Реализация алгоритма решения системы уравнений	46
2.2.10 Эффективность алгоритма решения системы.	50
2.2.11 Решение задачи в напряжениях	50

Часть 3. Оценка эффективности программы в целом

3.1 Контрольный пример	54
3.2 Анализ сходимости решения и его устойчивости	56
Приложения	59
Руководство пользователя	59
Ссылки на литературу	71

Введение

В инженерной практике очень часто встречается такой класс конструкций как плиты. Это различные плиты покрытий и перекрытий, металлические настилы, плитные фундаменты, стены под гидростатическим давлением, базы колонн и др. И все эти конструкции приходится рассчитывать инженеру при проектировании тех или иных объектов. Часто расчет облегчается использованием уже рассчитанных типовых конструкций с известной допустимой нагрузкой и не требующих сложного расчета. Особенно часто в гражданском строительстве применяются типовые плиты перекрытий и покрытий, имеющих большой типовой ряд.

Но встречаются и такие ситуации, когда необходимо запроектировать уникальную конструкцию, предназначенную для какой-то определенной ситуации. Особенно часто такая необходимость возникает при проектировании промышленных объектов, их монолитных железобетонных и, особенно, металлических конструкций. Это могут быть перекрытия под технологическое оборудование, перекрытия нестандартной формы с отверстиями, перекрытия с нестандартным опиранием и т.п. В таких ситуациях полного расчета не избежать.

СНИПы дают рекомендации как действовать в таких случаях, но как правило такие рекомендации не универсальны. Безусловно, они рассчитаны на ручной счет, так как инженер всегда должен владеть ручным счетом, хотя бы для того, чтобы всегда можно было бы проверить результаты, полученные другими методами. Ручной счет почти всегда не универсален, так как основан на упрощениях общей методики применительно для данного конкретного случая. Кроме трудоемкости и требований квалификации к инженеру у такого подхода есть еще один недостаток. Ведь количество описываемых в СНИПе случаев ограничено и невозможно предусмотреть все случаи заранее. Поэтому приходится «сводить» каждый конкретный случай к описанному в СНИПе и при этом не всегда остается точное соответствие расчетной модели и реальной рассчитываемой конструкции. И как следствие – довольно большие погрешности.

Для простых тонких плит существуют аналитические решения и для них нет необходимости делать сложную программу расчета, так как при

простейшем ручном счете получаются точные решения. Но этот класс простых тонких плит весьма ограничен, всего несколько конструкций имеют аналитические решения (прямоугольные плиты без отверстий, эллиптические и т.п). Аналитические решения не существуют для плит произвольного очертания с отверстиями, для плит со сложной схемой задания нагрузки.

В последнее время с развитием вычислительной техники появилась альтернатива ручному счету — машинные методы расчета. Машинный счет имеет ряд преимуществ перед ручным: большая универсальность, большая точность, быстрота расчета, простота и удобство использования. Но кроме преимуществ есть и недостатки, самым существенным из которых является то, что инженер не видит всего расчета, он проходит для него скрыто, и он не может убедиться в правильности результата не рассчитав вручную. Кроме того, часто возникает иллюзия абсолютности результатов полученных на машине. Но, несмотря на эти недостатки, использования машинных методов сейчас не избежать, необходимо лишь помнить об их общих недостатках при использовании.

Кроме этих общих недостатков существуют еще и конкретные недостатки самого метода численного расчета и программы его реализующей.

Сейчас существует множество программ и для расчета строительных конструкций и для различных прикладных задач, но большинство из них предназначены для применения лишь в узких областях, т.е. для расчета определенного типа конструкций. Из программ общего применения обращают на себя внимание несколько программных комплексов.

Зарубежные:

- Cosmos/M компании Structural Research and Analysys Corp.
- MSK/NASTRAN компании McNeal-Schwendler Corp.
- ANSYS компании ANSYS Corp.

и отечественные:

- Лира НИИАСС
- Зенит

Эти программные комплексы существуют и развиваются уже не одно десятилетие и прошли в своем развитии от командно-текстового

интерфейса до графического на базах современных графических операционных систем и оболочек. К сожалению все эти проекты являются коммерческими и стоимость последних версий продуктов делает их совсем не доступными для использования в наших условиях. А те версии, которые доступны, являются весьма устаревшими.

У нас сейчас используется программный комплекс Cosmos/m 1989 года выпуска. В свое время это была современная программа, использующая современные по тем временам технологии, но сейчас она уже сильно устарела. Она позволяет получить необходимый результат и нельзя так категорично заявить, что расчетный модуль ее устарел. В нем есть лишь недостаток, связанный с работой с памятью, тогда было возможным использование лишь 1 мегабайта памяти, а этого мало при расчетах даже не очень сложных конструкций. Поэтому используется подкачка с жесткого диска, а это сильно замедляет работу. Кроме того, в этой версии Cosmos/m существует ограничения максимального количества узлов. Также есть недостатки сложности использования этой программы.

Но все же основным недостатком программных комплексов, которые доступны для применения в России, является то, что в них слабо развиты возможности автоматической генерации конечно-элементной сетки. Ни в одной из них нет реально работающего алгоритма генерации сетки, который бы работал без помощи человека. В них заявлены такие возможности, но реально ими нельзя воспользоваться. В программе Cosmos/m можно сравнительно легко генерировать регулярные сетки только лишь для очень ограниченного числа случаев. Например, для прямоугольной плиты сетка строится путем разбиения прямоугольника на части в двух направлениях. Но такая задача становится неосуществимой при усложнении геометрии или при появлении отверстий в конструкции.

Ручная генерация сетки это очень трудоемкая задача, часто гораздо проще посчитать вручную. Это очень сильно ограничивает применение всех вышеперечисленных программ, это, пожалуй, самая серьезная причина ограниченности их применения.

1. Постановка задачи

Одной из основных задач поставленных передо мной было обязательное осуществление возможностей автоматической генерации сетки в моей программе. Она должна быть предназначена для расчета такого класса конструкций как жесткие плиты и построена на одном из современных численных методов решения. Кроме того в нее должны входить возможности наглядного представления и окончательных и промежуточных результатов работы. Обязательным была возможность задания всех видов статической нагрузки и закреплений в любых комбинациях.

Интерфейс должен быть наглядным и простым, основанным на возможностях, предоставляемых графической средой Windows. Ко всему прочему это должна быть самостоятельная программа, не требующая для себя настройки среды и не занимающая много места.

В процессе работы я добавил для себя еще ряд требований, касающихся применения современных технологий. Это решения по алгоритму генерации сетки и необязательный, но желательный алгоритм перенумерации узлов и пр.

1.1 Применимость программы

Первый вопрос с которым необходимо было определиться сразу это четкое ограничение класса конструкций, для расчета которых предназначается данная программа. Уже говорилось о том, что она предназначена для расчета тонких жестких плит, и теперь необходимо дать точное определение данного класса конструкции и установить границы применимости.

Плита, это призматическое тело, ограниченное двумя параллельными плоскостями, расстояние между которыми называется толщиной плиты. Плоскость, делящая толщину плиты пополам называют срединной.

Пусть L - характерный размер плиты, т.е. минимальный ее габарит, тогда:

Если $\frac{L}{h} \leq 5$, то плиту называют толстой.

Если $\frac{L}{h} \leq 50$, то плиту называют тонкой жесткой, в них можно пренебречь т.н. мембранными усилиями (продольными растягивающими и сдвигающими усилиями в срединной плоскости).

Если $50 < \frac{L}{h} \leq 100$, то плиту называют тонкой гибкой (здесь на напряженное состояние влияют мембранные силы и изгиб).

Если $\frac{L}{h} > 100$ - мембрана.

В строительстве наибольшее распространение получили тонкие жесткие плиты. В дальнейшем под плитой мы будем понимать именно этот вид плит.

В программу не закладывается никаких ограничений по геометрии конструкций, кроме здравого смысла. Это могут быть плиты полигональных очертаний, или очертаний, аппроксимированных полигонами. В них могут быть любые отверстия полигональных очертаний, это могут быть не отверстия, а зоны большей жесткости, толщины, приложения распределенной нагрузки.

Также возможно задавать любые виды статического воздействия. Разные их типы могут прикладываться к разным типам объектов, изменяя свой смысл в зависимости от типа объекта. Например нагрузка, прикладываемая к грани, является распределенной, а к узлу — сосредоточенной. То же можно сказать и о закреплении.

1.2 Выбор метода расчета

Самыми распространенными методами численного решения на сегодняшний день являются три метода:

- Метод конечных разностей (МКР)
- Метод граничных элементов (МГЭ)
- Метод конечных элементов (МКЭ)

Во всех численных методах решение описывается набором численных значений, обозначающих значение некоего параметра, описывающего поведение конструкции в целом, в точках рассчитываемой конструкции, или функций, описывающих поведение этого параметра на отдельных

участках конструкции. Поэтому для получения такого решения переходят от континуальной постановки задачи к дискретной, т.е. мысленно разбивают непрерывную конструкцию на участки, для которых можно получить численное решение. При достаточно большом количестве таких “контрольных” участков, распределенных по конструкции, численное решение будет приближаться к точному. Но, так как ресурсы современной вычислительной техники ограничены и не позволяют бесконечно увеличивать количество участков, всегда ограничиваются таким количеством участков, которое обеспечивает приемлемую точность результата, т.е. достаточную близость его к точному решению.

Свойства каждого из этих методов достаточно хорошо изучены и, поэтому, выбор параметров начального разбиения не составляет труда.

1.2.1 Метод конечных разностей.

В основе этого метода лежит чисто математический подход к решению разрешающих уравнений. Он заключается в том, что дифференциальные уравнения аппроксимируются алгебраическими зависимостями на дискретных участках конструкции и подстановке этих аппроксимаций в разрешающие уравнения. К достоинствам данного метода можно отнести следующее:

- Простота алгоритма
- При использовании МКР в некоторых задачах матрица разрешающих уравнений получается наиболее разреженной. Поэтому в этих задачах обычно применяют МКР.
- По оценкам специалистов с помощью МКР можно решать задачи наибольших размерностей.

Но, в то же время у МКР есть серьезный недостаток:

- Сложность расчета комбинированных конструкций

1.2.2 Метод граничных элементов

Этот метод исходит из интегрального уравнения задачи, а не из дифференциального. Для задач, описываемых дифференциальными уравнениями всегда существует эквивалентная постановка в интегральном виде. Поэтому здесь дискретизации подвергается не область определения задачи, а ее границы. Здесь всегда получается матрица меньшего порядка, но она всегда заполнена. С вычислительной точки зрения, почти всегда выгоднее решение разреженных систем более высокого порядка при сопоставимой точности результатов. Этот

метод распространен лишь в специальных областях, таких как расчет оснований как области с бесконечно удаленной границей, фильтрация воды через грунт.

Основным недостатком этого метода, кроме того, что он дает не выгодную матрицу уравнений, является то, что он достаточно сложно алгоритмируется.

1.2.3 Метод конечных элементов (МКЭ).

Метод конечных элементов впервые появился в современном виде в работах по строительной механике в 40-х 50-х годах. МКЭ тесно связан с идеологией строительной механики, хотя применяется во многих других областях. Его можно интерпретировать как метод перемещений для не стержневых конструкций.

К его достоинствам можно отнести следующее:

- Ясный физический смысл на всех этапах расчета, что позволяет легко проконтролировать результаты.
- Легкость расчета комбинированных конструкций, т.е. конструкций, включающих в себя элементы различной размерности (например, оболочка + стержень).
- Ввиду большой популярности МКЭ существует множество его реализаций.

Для программы расчета было рекомендовано использовать именно МКЭ благодаря его преимуществам и широкой распространенности и этот метод был положен в основу расчетной части программы.

2. Технический проект

2.1 Алгоритм автоматической генерации конечно–элементной сетки

Самым первым вопросом вставшим при решении этой задачи был выбор подходящего алгоритма, предназначенного именно для нашей задачи. Я провел небольшую исследовательскую работу по изучению самых известных из существующих алгоритмов генерации сетки. В основном я использовал англоязычные материалы интернет, так как русскоязычной литературы по данному вопросу я не нашел.

2.1.1 Обзор методов

Основными понятиями теории триангуляции и плоских треугольных сеток являются диаграммы Вороного (см. рис. 1) и триангуляция Делоне (Delaunay triangulation – см. рис. 2).

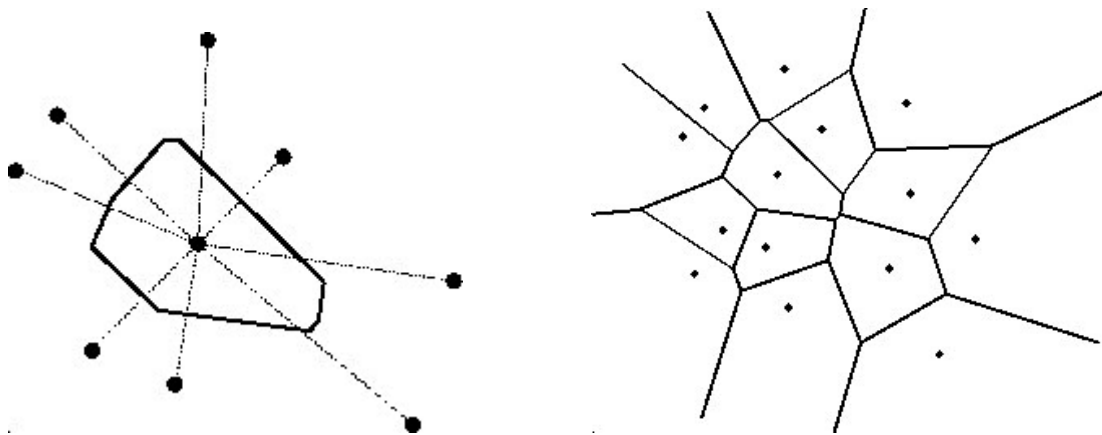


рис. 1. Диаграммы Вороного.

Диаграммами Воронова на плоскости для множества точек называют системы полигональных фигур, образуемых линиями, перпендикулярными отрезкам, соединяющими соседние точки и проходящим через середины этих отрезков. Триангуляция Делоне на плоскости - это множество не пересекающихся треугольников, в котором ни одна точка, не принадлежащая данному треугольнику не попадает в окружность, описанную вокруг этого треугольника.

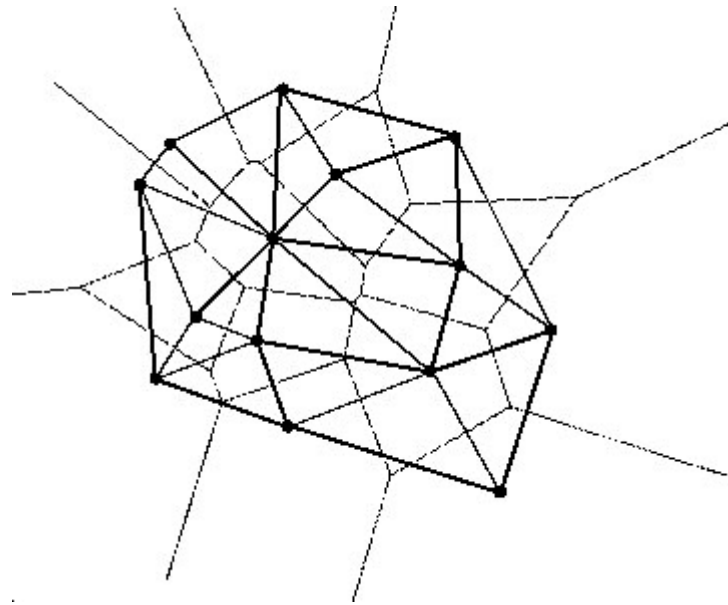


Рис. 2. Триангуляция Делоне (Delaunay triangulation).

Все рассматриваемые методы оперируют этими понятиями в том или ином виде.

Приведу лишь названия наиболее известных алгоритмов генерации сетки:

- 1.Radial sweep algorithm.
- 2.Recursive split algorithm (Алгоритм последовательного разбиения).
- 3.Divide-and-conquer algorithm (Алгоритм деления-и-включения).
- 4.Step-by-step algorithm.
- 5.Modified hierarchical algorithm (Модифицированный иерархический алгоритм).
- 6.Incremental algorithm.
- 7.Incremental delete-and-build algorithm.

Все эти алгоритмы не предназначены непосредственно для генерации конечно-элементной сетки. Они подразумевают наличие опорных точек, которые будут потом узлами сетки.

К сетке для метода конечных элементов предъявляется ряд особых требований. Во-первых, треугольники не должны быть сильно вытянутыми, т.к. наличие таких треугольников отрицательно сказывается на точности результатов расчета. Во-вторых, должен быть учтен характер работы конструкции и ее геометрия, т.е. в местах концентрации напряжений сетка должна сгущаться. Все эти требования учтены в алгоритме генерации треугольных иррациональных конечно-элементных сеток Рапперта.

2.1.2 Алгоритм Рапперта

Этот алгоритм довольно новый, он был опубликован в 1994 году [Ruppert 1994]. Его разработал Джим Рапперт еще будучи студентом в рамках проекта, поддерживаемым NASA. Алгоритм адаптирован для метода конечных элементов и переисчисленные выше требования, предъявляемые к конечно-элементным сеткам удовлетворяются автоматически. К его преимуществам можно отнести и то, что алгоритм легко параметризуется и управляется.

Если говорить точнее, этот алгоритм не является алгоритмом генерации, а лишь алгоритмом улучшения качества сетки. В нашем случае входными данными для него будут являться геометрия конструкции в виде замкнутых полигонов и первичное разбиение конструкции на треугольники. Для получения первичного разбиения при реализации алгоритма генерации сетки я использовал упрощенный вариант алгоритма step-by-step (см. выше) с использованием в качестве точек вершин полигонов исходной геометрии. Вкратце это действие можно описать так:

1. Выбирается начальный сегмент (любой) на внешнем полигоне исходной геометрии.
2. Из всех вершин выбирается такая, из которой лучи, соединяющие эту вершину и концы текущего сегмента образуют максимальный угол.
3. Добавляется полученный таким образом треугольник и текущим сегментом становится одна из созданных сторон этого треугольника.
4. Если существует подходящий текущий сегмент, то на шаг 2.

Алгоритм опирается на две базовые процедуры - разбиение треугольника с введением нового узла и разбиение сегмента, принадлежащего границе области разбиения с введением нового узла. При разбиении треугольника происходит следующее:

1. Вычисляются координаты центра окружности, описанной вокруг треугольника, подлежащего разбиению.
2. В эту точку добавляется новый узел.
3. Удаляется треугольник, подлежащий разбиению и прилегающие и добавляются новые, как показано на рисунке 3.

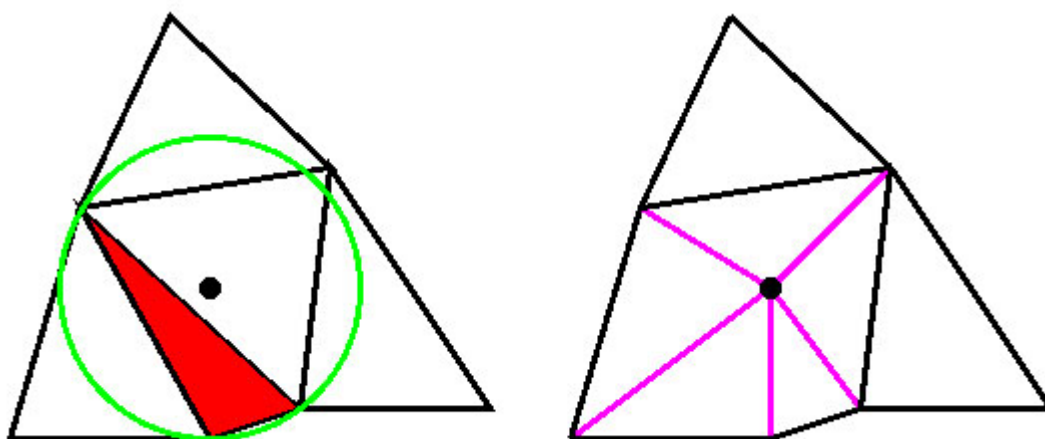


рис. 3. Иллюстрация работы процедуры разбиения «неправильного» треугольника.

Вторая базовая процедура состоит в следующем:

1. Если в окружность, диаметром которой является сегмент, принадлежащий границе области разбиения попадает точка, не принадлежащая этому сегменту, то сегмент делится на две части.
1. Удаляется треугольник, которому принадлежал первоначальный сегмент и добавляются два новых треугольника (см. рис. 4).

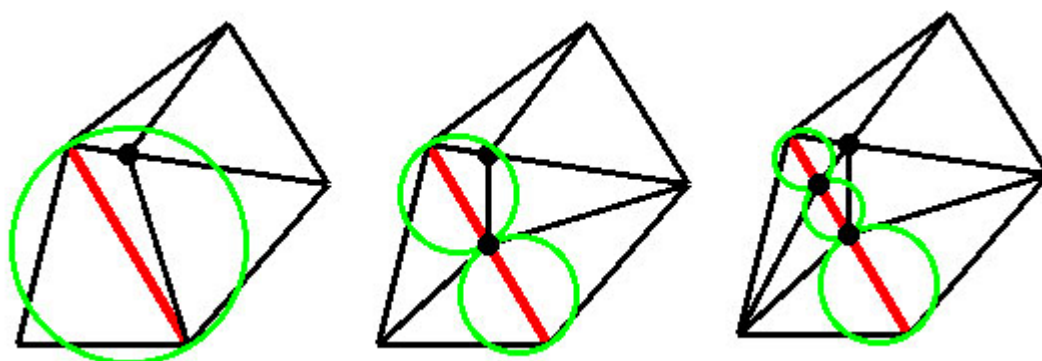


рис. 4. Работы процедуры разбиения сегмента. Удаляется треугольник, которому принадлежал первоначальный сегмент и добавляются два новых треугольника.

Принцип работы алгоритма состоит в цикле определения «неправильных» треугольников, т.е. не удовлетворяющих определенному критерию, и произведение над ним одной из базовых процедур. В качестве критерия я применил два условия: условие минимального угла и условие максимальной площади треугольника таким образом, что после работы алгоритма можно утверждать, что в сетке углы всех треугольников не менее значения, заданного пользователем, а площади всех треугольников не более другого значения, также задаваемого пользователем.

Математически доказано, что критерий минимального угла автоматически обеспечивает сгущение сетки вблизи мелких подробностей: отверстий, углов, вырезов, т.е. концентраторов напряжений. А критерий максимальной площади является дополнительным, регулирующим размер треугольников. Введение этого критерия было необходимо для адаптации этого алгоритма к методу конечных элементов.

2.1.3 Модификация алгоритма Рапперта

В этой работе для Journal of Algorithms [Ruppert 1994] раскрыт только лишь чисто математический аспект проблемы, хотя вероятней всего он разрабатывался для решения прикладных задач и для генерации конечно–элементной сетки в том числе. К тому же, и математическая сторона была раскрыта не полностью, а только лишь с точки зрения сходимости и некоторых особенностей. Ряд самых неочевидных подводных камней не поднимается на обсуждение. Вероятно это объясняется тем, что это проект NASA и, вероятно, имеет какую-то степень закрытости.

В том варианте, в котором он был предложен автором [Ruppert. 1994], алгоритм не обладал хорошей сходимостью и устойчивостью, а точнее углы более 25 градусов были недоступны для него. Для повышения качества работы алгоритма, помимо двух вышеперечисленных процедур была введена еще одна–процедура поворота диагонали.

Суть ее заключается в следующем. Два треугольника, имеющие общую грань, образуют четырехугольник, разделенный диагональю. Внутри такого образования диагональ имеет два возможных положения. На рисунке 5 показаны эти положения — отрезки ac и bd . В общей системе триангуляции положение этой диагонали является локальной характеристикой для четырехугольника. Действительно, смена положения, кроме некоторых случаев, обсуждаемых позже, изменит характеристики только этих двух треугольников.

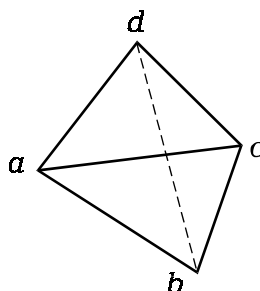


рис. 5. Диагональ внутри четырехугольника имеет два возможных положения.

Этой возможностью можно воспользоваться для локального улучшения свойств сетки. Для этого необходимо выяснить еще одно обстоятельство — критерий, который бы давал ответ, улучшится ли локальная характеристика сетки при смене положения диагонали, т.е. дал ответ надо или не надо производить эту операцию над каждым конкретным кластером.

Мне пришлось перепробовать три возможных варианта. Первый, самый очевидный критерий это длина диагонали. Действительно, зрительно, диагональ ac кажется гораздо более удачной диагонали bd для минимального угла двух треугольников (см. рис. 5). Но здесь есть подводный камень. На рисунке 6 показан случай, где использование критерий минимальной длины диагонали может привести к серьезному ухудшению свойств сетки.

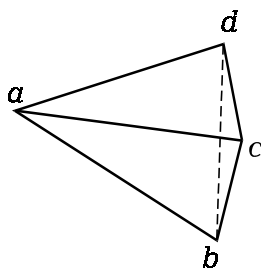


рис. 6. Случай, где использование критерия минимальной длины диагонали может привести к серьезному ухудшению свойств сетки.

Диагональ bd короче диагонали ac , поэтому, по критерию минимальной длины диагонали, диагональ ac надо поменять на bd . Но это приведет к появлению очень *тонкого* треугольника, а их появление крайне нежелательно, а часто становится критическим. Этот случай довольно сложно описать математически, на машинном языке, так, чтобы легко было отсекал подобные ситуации, поэтому этот критерий пришлось отбросить.

Следующий очевидный критерий можно получить сравнивая минимальные углы в двух треугольниках до и после смены диагонали, для предсказания эффективности этого действия (см. рис. 7). Такой критерий действительно является объективным, т.к. он совпадает с основным глобальным критерием сетки — критерием минимального угла.

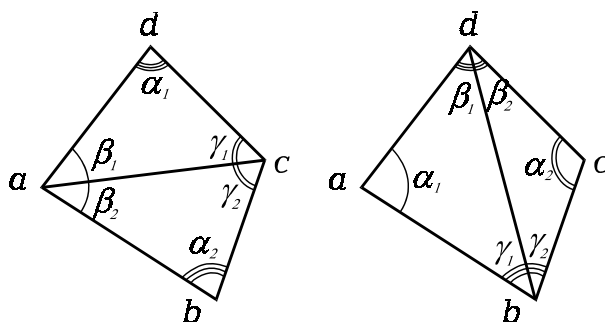


рис. 7. Сравнение минимальных углов в двух треугольниках до и после смены диагонали.

Применение такого критерия действительно кажется заманчивым в силу его объективности, но у него есть один специфический недостаток — трудоемкость. Для него необходимо вычислить 6 углов, а по объему операций трудоемкость этого превосходит более чем в 10 раз вычисление двух длин отрезков. Поэтому пришлось использовать компромиссное решение.

Действие вычисления площади по трудоемкости сравнимо с нахождением одного угла. Поэтому третьим критерием является сравнение площадей двух треугольников, точнее не сравнение самих площадей, а сравнение с единицей площадей до и после смены диагонали. Если после смены диагонали отношение площадей ближе к

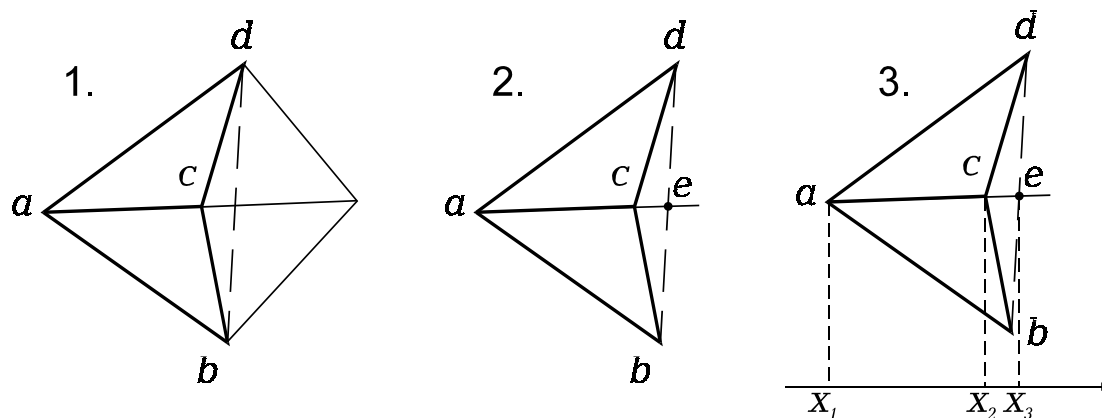


рис. 8. 1 — новая диагональ пересекает грань уже существующих соседей, нарушая тем самым непрерывность описываемого треугольниками пространства. 2 — точка пересечения новой, мнимой диагонали и прямой, на которой лежит существующая диагональ. 3 — математическое подтверждение мнимости новой диагонали.

единице, например сверху, чем до смены (тоже сверху), то распределение площадей более равномерно и производится смена диагонали. Этот критерий теоретически не обоснован, он больше интуитивный, эмпирический, но тем не менее, на практике работает намного лучше двух предыдущих.

Выше было упомянуто о существовании исключения, при котором нарушается локальность изменения положения диагонали внутри четырехугольника. На рисунке 8 показан такой случай.

Как видно, новая диагональ выходит за границы кластера, пересекая грани уже существующих соседей. Из-за этого получается пересечение площадей с соседними треугольниками, что совсем недопустимо, т.к. вызывает нарушение непрерывности пространства, описываемого треугольниками. Поэтому, в дополнение к основному критерию, необходимо добавить еще один, запрещающий производить смену диагонали в подобных ситуациях. Самыми простыми являются два. Первый показан на рисунке, когда отыскивается точка пересечения прямых, на которых лежат существующая и новая диагональ и проверяется, лежит ли точка пересечения на этих отрезках. Но самым эффективным является второй, когда сравниваются суммы площадей двух треугольников, т.е. площади четырехугольников до и после смены диагоналей. Если суммы совпадают то, новая диагональ не является мнимой. Этот способ особенно эффективен с использованием критерия по площади, т.к. они совместно используют четыре площади треугольников.

Теперь необходимо установить в каком порядке включать эту процедуру. Самым простым решением было проверять циклом каждую грань триангуляции на каждом шаге и, если необходимо, производить смену диагонали. Такой подход и был осуществлен в первой версии рабочего алгоритма, и надежно работал. Но известно, что в треугольной сетке граней больше чем треугольников и узлов в 1,5-1,6 и в 3 раза соответственно, поэтому тотальная проверка всех граней стала узким местом.

Но по сути тотальной проверки производить не надо т.к. если, например, на предыдущем шаге грань i не нуждалась в смене и, если изменения в сетке, сделанные в этот шаг, не коснулись треугольников, которым принадлежит грань i , то, очевидно, что и на текущем шаге она не нуждается в смене.

Исходя из этого была применена другая технология — технология распространяющегося «вируса смены диагонали». Суть ее заключается

в следующем. После выполнения одной из базовых процедур точно известно, какие треугольники сетки подверглись изменениям, т.е. известна область, которая подверглась изменениям. Грани в этой области могут нуждаться в перемене своего направления и эти грани становятся «зараженными». Они помещаются в очередь для проверки – линейный массив индексов. Каждая грань в очереди проверяется и если проверяемая грань нуждалась в изменении, то все остальные грани треугольника, содержащих проверяемую грань также становятся зараженными и помещаются в очередь. Если же грань не нуждается в изменениях, то она удаляется из очереди. Такой процесс продолжается до тех пор, пока очередь не станет пустой.

Такой подход существенно ускорил работу алгоритма в целом. Например, в одном контрольном примере, для случая с тотальной проверкой время работы на Celeron 266 (333) составило около 18 секунд, а для второго случая время составило менее 10 секунд — сокращение трудоемкости более чем на 45%.

Кроме добавления в базовый алгоритм еще одной процедуры, необходимо было усовершенствовать и базовые алгоритмы. Как было сказано выше в базовом алгоритме есть ряд недочетов, которые потребовали своего устройства.

Первый недочет состоял в том, что алгоритм не работал правильно с симметричными или правильными геометрическими формами. На рисунке видно, при попытке разбить равнобедренный треугольник bcd с

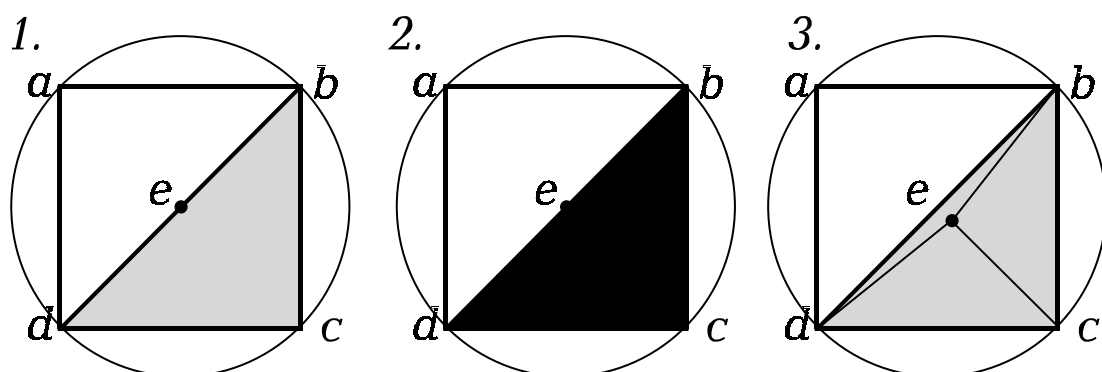


рис. 9. Возможность возникновения треугольников нулевой площади.

1 – Центр описанной окружности треугольника bcd находится точно на отрезке bd . 2 – процедура разобьет треугольник на три: dce , bec , bde . Треугольник bde имеет нулевую площадь. 3 – поправки к точным координатам центра описанной окружности.

прямым углом по базовой процедуре образуются треугольники нулевой площади (см. рис. 9). В данном случае процедура разобьет треугольник

на три: dce , bec , bde . Центр описанной окружности треугольника bcd находится точно на отрезке bd , поэтому треугольник bde имеет нулевую площадь. При попытке вычислить координаты центра его описанной окружности происходит деление на ноль. Поэтому пришлось принять ряд мер.

Во-первых и в первую и во вторую базовую процедуру были введены небольшие относительные поправки к точным координатам середины отрезка и центра описанной окружности, не нарушающие принципы работы алгоритма. Теперь точки стали добавляться не точно в середины сегментов и центры окружностей, а в точки, расположенные вблизи них. Такое действие добавляет какую-то степень «иррациональности» в правильную геометрию, и ликвидирует такую проблему.

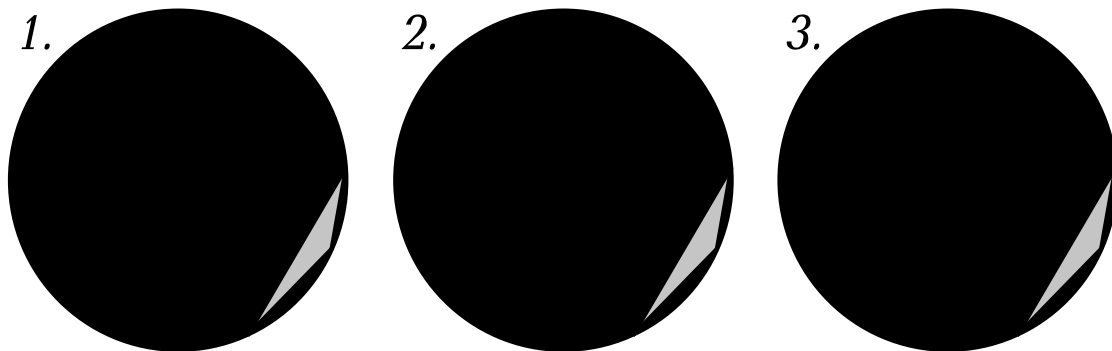


рис. 10. 1 — центр описанной окружности находится далеко от самого треугольника. 2 — в триангуляцию вводится новая точка и два новых треугольника. 3 — первоначальный треугольник не претерпевает никаких изменений.

Второй недочет заключается в том, что не учитывается возможность существования таких треугольников, у которых центр описанной вокруг них окружности находится далеко от самого треугольника. В таком случае, при делении треугольников с наименьшим углом в первую очередь, возникает следующая ситуация (см. рис. 10). При первом разделении такого треугольника добавляется точка в центр описанной вокруг него окружности, но так как точка находится далеко от самого треугольника, то сам треугольник не претерпевает никаких изменений и он по-прежнему остается первым в очереди для разбиения.

При следующей попытке разбиения опять вводится новая точка с теми же координатами, что и на предыдущем шаге. В итоге получаются треугольники нулевой площади и грани нулевой длины. Для ликвидации

подобных ситуаций была введена проверка на принадлежность вновь вводимой точки к одному из соседних треугольников.

Принадлежность к треугольнику устанавливается следующим способом. Для треугольника, зная вершины 1, 2 и 3 треугольника проверяется с какой стороны от грани 1-2 находится проверяемая точка, далее аналогично для граней 2-3 и 3-1. Если для всех граней точка находится с одной стороны, то точка находится внутри треугольника.

Все эти усовершенствования коснулись самого алгоритма улучшения сетки. Они объективно позволили улучшить стабильность работы и качество генерируемой сетки, но в нем все же есть небольшие недостатки. Но они не критичны, самым заметным из них является появление неоправданных сгущений сетки. Такие сгущения появляются не всегда, но если и появляются то их можно ликвидировать изменив параметры (минимальный угол или максимальная площадь) и сгенерировав заново сетку. А при разумных установках этих параметров, алгоритм всегда сходится, т.е. достигает этих параметров. Безусловно при более тщательном его исследовании можно добиться еще лучших результатов.

2.1.4 Алгоритм построения первичного разбиения

Ранее говорилось об алгоритме генерации сетки, но по сути все, о чем говорилось выше, является не алгоритмом генерации сетки, а лишь алгоритмом ее улучшения (Delaunay Refinement Algorithm). Для него входными данными является уже готовое разбиение. Первоначальное разбиение полигональной фигуры производится другим алгоритмом. Входными данными для него являются массивы точек (координаты, нагрузка, закрепление) и граней полигонов, в терминологии Рапперта [Ruppert 1994] — сегментов, соединяющих эти точки. Выходные данные — это массив вершин без изменений, массив граней треугольников и массив собственно треугольников, заполняющих весь объем конструкции.

Здесь используется технология, похожая на «вирус смены диагонали». Первым действием является нахождение первого «правильного»

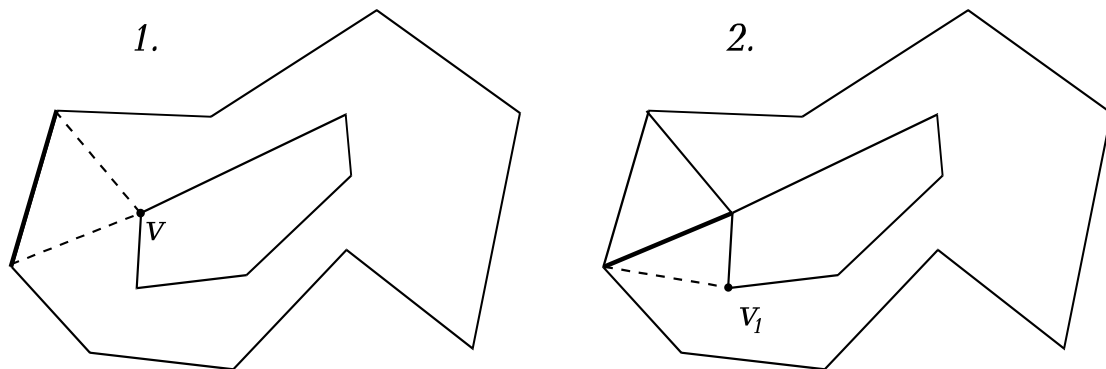


рис. 11. Генерация начального разбиения.

треугольника. Таким образом появляются две первых грани (не сегмент, сегментом называется такая грань, которая принадлежит к границам первоначальных полигональных областей – очертаний). Далее треугольники образуются путем правильного добавления к грани еще двух граней соединяющих концы этой грани и определенную точку, принадлежащую сегменту (границе области), добавляя в сетку еще две грани, и т.д. (см. рис. 11)

При выборе третьей вершины для формирования треугольника возникает множество вопросов. Во-первых, необходим критерий, позволяющий легко избежать пересечений вновь создаваемых граней с уже существующими. Во-вторых, как ограничить треугольники от распространения за пределы полигональной области. В-третьих, как правильно сгенерировать первый треугольник. И в-четвертых, как учесть то обстоятельство, что в конструкции присутствуют как отверстия так и заполненные области описываемые одинаково.

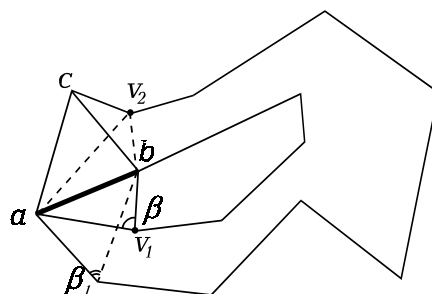


рис. 12. Выбор третьей точки треугольника. Угол β является максимальным, поэтому выбирается эта точка.

Критерием для оптимального выбора третьей точки треугольника является угол, под которым видна грань из этой точки, т.е. угол образуемый двумя лучами, выходящими из выбираемой точки и проходящими через концы грани. Для каждой грани выбирается такая точка, для которой этот угол максимален. Такой выбор критерия позволил сэкономить время на том, что нет необходимости проверять вновь создаваемые грани на пересечение с уже существующими. Для того чтобы сразу исключить проверки, подобные проверке точки v_2 , перед перебором всех точек устанавливается с какой стороны лежит третья точка треугольника, которому принадлежит грань ab (точка c , см. рис. 12). Далее каждая точка проверяется на то, с какой стороны от грани ab она находится и если она находится с той же стороны, то такая точка заведомо не верна и она сразу отвергается. Я не находил точного обоснования этой особенности, но, тем не менее, такой подход надежно работает.

Чтобы треугольники не распространялись за пределы конструкции, вводится простое правило: треугольники не формируются от сегментов a

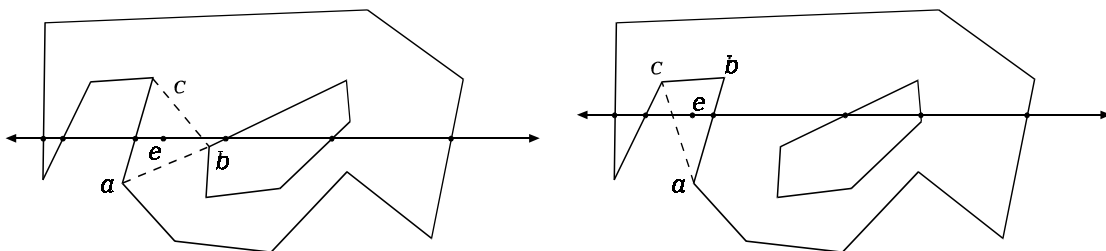


рис. 13. Установление принадлежности точки e контуру.

только от граней уже существующих треугольников. На грани также накладывається ограничение, треугольники формируются только от граней, которые принадлежат только одному треугольнику. Эти правила нарушаются только в одном случае — при генерации первого треугольника.

Первый треугольник генерируется от любого сегмента внешнего полигона, а третья точка выбирается также как и для грани. Для первой проверяемой точки устанавливается принадлежность центра треугольника, образуемого этой точкой и сегментом, конструкции (см. рис. 13). Для этого через этот центр проводится прямая, параллельная оси X и находятся все ее точки пересечения с сегментами. Далее эти точки сортируются по мере возрастания координаты X и формируются интервалы. Каждый нечетный интервал принадлежит контуру.

Все вышеперечисленные процедуры и формируют алгоритм генерации сетки. Сюда еще была добавлена полезная функция вывода файла с разбиением и граничными условиями задачи для программного комплекса Cosmos/m, но эта задача была осуществлена в другом модуле.

2.2 Расчетная часть программы.

Следующей задачей работы было создание расчетной части программы, основанной на методе конечных элементов. Последовательность действий для метода конечных элементов стандартна и довольно хорошо описана. В нее входят:

- Построение матрицы инцидентий
- Перенумерация узлов элементов по алгоритму Катхилла-Макки [Джордж 1984].
- Генерация локальных матриц жесткости для элементов построенных алгоритмом генерации сетки
- Учет граничных условий задачи и создание структур, описывающих глобальную матрицу жесткости
- Сборка глобальной матрицы, включение локальных матриц жесткости в глобальную с помощью списка инцидентий, учитывая список перенумерации узлов
- Учет условий загрузки и построение матрицы свободных членов
- Решение системы уравнений
- Решение в перемещениях

В этой части программы были осуществлены все эти пункты в полном объеме с использованием одних из самых эффективных методов [Джордж 1984]. Такой же объем осуществлен в программе Cosmos/m, в той его версии, что доступна у нас. Но основное преимущество расчетного модуля перед расчетной частью программы Cosmos/m состоит в том, что работая под Windows она использует возможности машины максимально. Во-первых, используются 10-байтовые числа с плавающей точкой типа *long double*, во-вторых вся глобальная матрица жесткости размещается в оперативной памяти машины, тогда как Cosmos/m использует только лишь один мегабайт оперативной памяти.

Эти преимущества дают высокую скорость счета и более низкую погрешность вычислений.

2.2.1 Выбор конечного элемента

В данной реализации метода конечных элементов используется треугольный конечный элемент с десятью степенями свободы, предназначенный для расчета плоских тонких анизотропных плит и пластин [А. Александров 1983]. Выбор данного конечного элемента обусловлен в основном тем, что он дает наибольшую точность при относительной своей простоте.

Этот треугольный элемент имеет десять степеней свободы - девять узловых и одно перемещение в центре тяжести треугольника. Эта «лишняя» степень свободы является локальной, т.к. при перемещении, соответствующему этой степени свободы, деформируется только данный треугольник, не затрагивая других.

Наличие «лишней» десятой степени свободы обусловлено необходимостью симметрии относительно координат x и y аппроксимирующего полинома.

$$w = a_1 + a_2x + a_3y + a_4x^2 + a_5xy + a_6y^2 + a_7x^3 + a_8x^2y + a_9xy^2 + a_{10}y^3$$

или в матричной форме

$$w = [1 \quad x \quad y \quad x^2 \quad xy \quad y^2 \quad x^3 \quad x^2y \quad xy^2 \quad y^3] \bar{a}$$

Дифференцируя по x и y , получим выражения для полей углов поворота и перемещений:

$$\begin{bmatrix} w \\ \varphi^x \\ \varphi^y \end{bmatrix} = \begin{bmatrix} w \\ \frac{dw}{dy} \\ -\frac{dw}{dx} \end{bmatrix} = \begin{bmatrix} 1 & x & y & x^2 & xy & xy^2 & x^3 & x^2y & xy^2 & y^3 \\ 0 & 0 & 1 & 0 & x & 2y & 0 & x^2 & 2xy & 3y^2 \\ 0 & -1 & 0 & -2x & -y & 0 & -3x^2 & -2xy & -y^2 & 0 \end{bmatrix} \bar{a}$$

Подставив координаты точек 0, 1, 2, 3 элемента, получим:

$$\begin{bmatrix} w_0 \\ w_1 \\ w_1^x \\ w_1^y \\ w_2 \\ w_2^x \\ w_2^y \\ w_3 \\ w_3^x \\ w_3^y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & x_1 & y_1 & x_1^2 & x_1 y_1 & y_1^2 & x_1^3 & x_1^2 y_1 & x_1 y_1^2 & y_1^3 \\ 0 & 0 & 1 & 0 & x_1 & 2y_1 & 0 & x_1^2 & 2x_1 y_1 & 3y_1^2 \\ 0 & -1 & 0 & -2x_1 & -y_1 & 0 & -3x_1^2 & -2x_1 y_1 & -y_1^2 & 0 \\ 1 & x_2 & y_2 & x_2^2 & x_2 y_2 & y_2^2 & x_2^3 & x_2^2 y_2 & x_2 y_2^2 & y_2^3 \\ 0 & 0 & 1 & 0 & x_2 & 2y_2 & 0 & x_2^2 & 2x_2 y_2 & 3y_2^2 \\ 0 & -1 & 0 & -2x_2 & -y_2 & 0 & -3x_2^2 & -2x_2 y_2 & -y_2^2 & 0 \\ 1 & x_3 & y_3 & x_3^2 & x_3 y_3 & y_3^2 & x_3^3 & x_3^2 y_3 & x_3 y_3^2 & y_3^3 \\ 0 & 0 & 1 & 0 & x_3 & 2y_3 & 0 & x_3^2 & 2x_3 y_3 & 3y_3^2 \\ 0 & -1 & 0 & -2x_3 & -y_3 & 0 & -3x_3^2 & -2x_3 y_3 & -y_3^2 & 0 \end{bmatrix} \bar{a}$$

$$\bar{Z} = L\bar{a}, \text{ откуда } \bar{a} = L^{-1}\bar{Z},$$

$$w = [1 \quad x \quad y \quad x^2 \quad xy \quad y^2 \quad x^3 \quad x^2y \quad xy^2 \quad y^3] L^{-1}\bar{Z}$$

По вектору w , используя зависимости Коши построим вектор относительных деформаций:

$$\varepsilon = -z \begin{bmatrix} \frac{\partial^2 w}{\partial x^2} \\ \frac{\partial^2 w}{\partial y^2} \\ 2 \frac{\partial^2 w}{\partial x \partial y} \end{bmatrix} = -z \begin{bmatrix} 0 & 0 & 0 & 2 & 0 & 0 & 6x & 2y & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 2x & 6y \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 4x & 4y & 0 \end{bmatrix} L^{-1}\bar{Z} = BL^{-1}\bar{Z}$$

Матрицу реакций можно получить из следующего выражения:

$$R^* = \frac{h^3}{12} (L^{-1})^T \int_{\Delta} B^T DB \cdot dx dy \cdot L^{-1}$$

Подынтегральное выражение представляет матрицу 10x10 (см. табл. 1).

Интегрировать это выражение удобно численно с использованием весовых коэффициентов, отдельно для каждого треугольника [А. Александров 1983]. Такое интегрирование основано на переходе к треугольным или, как их еще называют, однородным координатам. Некоторые авторы пытались уменьшить число коэффициентов полинома на единицу. Так, предлагалось, не нарушая симметрию полинома, принять коэффициент $a_5 = 0$, однако при этом теряются деформации,

соответствующие чистому кручению $M_{кр} = \frac{\partial^2 w}{\partial x \partial y}$, и элемент не дает

сходимости к точному решению. Предлагалось принять $a_8 = a_9$, но при этом для некоторых видов треугольников матрица A получается особенной и не имеет обратной, что не позволяет построить матрицу

жесткости R. Но десятая степень свободы не является проблемой, т.к. от локальной степени свободы можно избавиться, используя исключение Гаусса [А. Александров 1983].

$$\begin{vmatrix}
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 4 & 0 & 4\mu & 12x & 4y & 4\mu x & 12\mu y & \\
 0 & 0 & 0 & 0 & 2(1-\mu) & 0 & 0 & 4x(1-\mu) & 4y(1-\mu) & 0 & \\
 0 & 0 & 0 & 4\mu & 0 & 4 & 12\mu x & 4\mu y & 4x & 12y & \\
 0 & 0 & 0 & 12x & 0 & 12\mu x & 36x^2 & 12xy & 12\mu x^2 & 36\mu xy & \\
 0 & 0 & 0 & 4y & 4x(1-\mu) & 4\mu y & 12xy & 4y^2 + 8x^2(1-\mu) & 4\mu xy + 8xy(1-\mu) & 12\mu y^2 & \\
 0 & 0 & 0 & 4\mu x & 4y(1-\mu) & 4x & 12\mu x^2 & 4\mu xy + 8xy(1-\mu) & 4x^2 + 8y^2(1-\mu) & 12xy & \\
 0 & 0 & 0 & 12\mu y & 0 & 12y & 36\mu xy & 12\mu y^2 & 12xy & 36y^2 &
 \end{vmatrix}$$

табл. 1. Подынтегральное выражение в формуле для локальной матрицы жесткости.

У этого элемента существует одна важная особенность, о которой необходимо упомянуть. Матрица L описывающая геометрию конечного элемента в некоторых случаях не имеет обратной. Такая ситуация возникает при попытке сгенерировать матрицу жесткости для конечного элемента с геометрией стремящейся к равнобедренному треугольнику. В этом случае самым простым разумным решением было просто слегка «подправить» координаты его вершин. Осуществляется это простым циклом:

```

gen_L(); // сборка матрицы L по известным координатам.
while (L->invertGJ() == ide_vir) // ide_vir - сообщение о том, что
                                // матрица вырождена
{
    f = sqrt(area());
    x3 += f/1000; // небольшое изменение третьей координаты
    y3 += f/800; // на величину 0,001 от квадратного корня от
                // площади элемента.
    gen_L();
};

```

Такие ситуации возникают довольно редко, примерно 2 – 3 элемента в одном расчете и поэтому этот учет не сказывается заметно на скорости исполнения программы.

2.2.2 Алгоритм перенумерации узлов

Также была осуществлена необходимая для таких задач функция перенумерации узлов сетки. Она работает по известному и проверенному временем алгоритму Катхилла-Макки. Она состоит из двух подзадач: нахождение псевдопериферийного узла и собственно перенумерация. Понятие псевдопериферийного узла пришло из теории графов и для пояснения необходимо ввести еще несколько понятий: расстояние, диаметр графа, эксцентриситет узла и периферийный узел.

Расстояние между двумя узлами это есть длина кратчайшего пути, соединяющего эти узлы. *Диаметр графа* – это максимально возможное расстояние из всех возможных сочетаний узлов этого графа. *Эксцентриситетом* узла называется x называется расстояние из до самого «дальнего» узла, т.е. расстояние до которого максимально. *Периферийным* называется узел, эксцентриситет которого совпадает с диаметром графа.

Эти задачи были реализованы по алгоритмам предложенными американскими исследователями А. Джорджем и Дж. Лю [Джордж 1984], с переделкой под специфику языка С++ и моих схем хранения.

Входными данными для этого алгоритма являются два массива описывающих структуру графа, который в свою очередь отражает

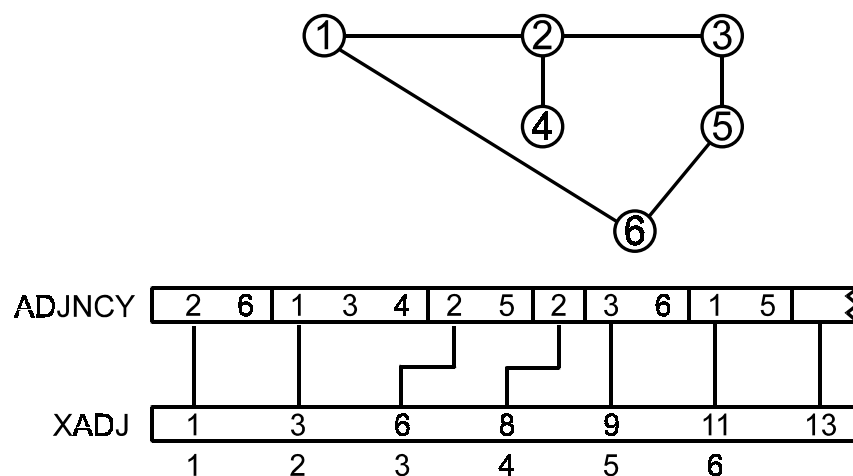


рис. 14. Пример структуры смежности графа

структуру смежности, т.е. связи между узлами. Этот граф строится с использованием списка инцидентий, и записывается в виде двух линейных массивов *xadj* и *adjncy* (см. рис. 14). Массив *adjncy* содержит

список узлов-соседей для каждого узла по порядку, а массив *xadj* структурирует его. Элемент массива *xadj* с индексом *i* содержит точку входа в массив *adjncy*, т.е. индекс с которого начинается перечисление соседей узла *i*. Для доступа ко всем соседям узла *node* удобно воспользоваться следующей процедурой:

```
jstrt = xadj[node];  
jstop = xadj[node+1];  
for (unsigned int j = jstrt; j < jstop; j++)  
{  
    nbr = adjncy[j];  
};
```

Как видно такая схема хранения довольно удачна с точки зрения доступа к данным.

Сам алгоритм состоит из двух подзадач: поиск псевдопериферийного узла и собственно перенумерация. В символьном виде его можно описать так:

Шаг 1. Определить начальный узел *r* и выполнить операцию присваивание $x_1 = r$, т.е. первым узлом в новой нумерации будет узел *r*.

Шаг 2 (основной цикл). Для $i = 1, \dots, N$ найти всех ненумерованных соседей узла x_i и занумеровать в порядке возрастания степеней

Шаг 3 (обратное упорядочение). Обратное упорядочение Катхтлла—Макки есть y_1, y_2, \dots, y_n , где $y_i = x_{N-i+1}$ для $i = 1, \dots, N$.

2.2.3 Алгоритм поиска начального узла нумерации

От выбора на первом шаге узла *r* критически зависит эффективность алгоритма в целом, поэтому необходимо рассмотреть процедуру его отыскания более подробно. Определить истинный периферийный узел можно лишь перебрав все узлы графа и определив эксцентриситет для каждого. Этот подход весьма неэффективен, ведь для того, чтобы определить эксцентриситет каждого узла, необходимо перебрать все остальные узлы. Но существует гораздо более быстрый эвристический метод отыскания псевдопериферийного узла (модифицированный алгоритм Гиббса) [Джордж 1984] [Gibbs 1976]. Этот метод не гарантирует, что будет найден периферийный узел или хотя бы близкий к нему узел.

Но, как правило, получаемые узлы имеют большой эксцентриситет и являются хорошими начальными значениями.

Основной конструкцией, с которой оперирует этот алгоритм, является *корневая структура уровней* (см. рис. 15). При заданном узле x структура уровней с корнем в x есть разбиение $L(x)$ множества X :

$$L(x) = \{L_0(x), L_1(x), \dots, L_{l(x)}(x)\},$$

такое, что

$$\begin{aligned} L_0(x) &= \{x\}, L_1(x) = Adj(L_0(x)), \\ L_i(x) &= Adj(L_{i-1}(x)) - L_{i-2}(x), i = 2, 3, \dots, l(x), \end{aligned}$$

где $Adj(L_i(x))$ – множество соседей узлов, принадлежащих множеству $L_i(x)$.

Если привлечь только что определенное понятие структуры уровней, то в словесной формулировке алгоритм можно описать так:

Шаг 1 (инициализация). Выбрать в X произвольный узел r

Шаг 2 (построение структуры уровней). Построить структуру уровней с корнем в r : $L(r) = \{L_0(r), L_1(r), \dots, L_{l(r)}(r)\}$

Шаг 3 (Стягивание последнего уровня). Выбрать в $L_{l(r)}(r)$ узел x с минимальной степенью

Шаг 4 (Построение структуры уровней). $L(x) = \{L_0(x), L_1(x), \dots, L_{l(x)}(x)\}$, если $l(x) > l(r)$, то положить $r = x$ и перейти к шагу 3.

Шаг 5 (Конец). Узел x является периферийным.

Структуру подпрограмм обратного алгоритма Катхилла-Макки и отношения между ними можно представить в схемы (см. рис. 16).

Ввиду краткости здесь будут приведены эти подпрограммы в виде исходного текста на языке C++.

Подпрограмма `fnroot` (FiNd ROOT) используется для нахождения псевдопериферийного узла, который используется в дальнейшем для переупорядочения. В свою очередь подпрограмма `fnroot` использует подпрограмму `rootls` (ROOTed Level Structure), которая по сути является шагом 2, вынесенным в отдельную процедуру.

Назначение подпрограммы `rootls` — генерировать структуру уровней для связного графа. Входными данными для нее являются: `root` — заданный корень структуры уровней, `mask` — массив, маскирующий узлы графа; массивы `xadj` и `adjncy`, описывающие структуру графа. На выходе

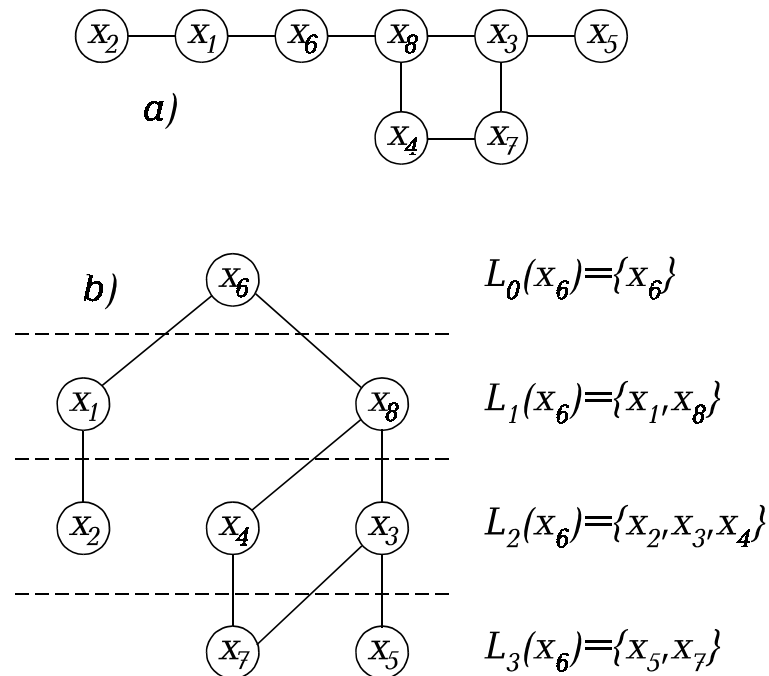


рис. 15. **a** — граф с 8 узлами. **b** — структура уровней с корнем в x_6 для него.

будет получена структура уровней с корнем в узле `root`; она задается парой массивов `xls` и `ls`. Формат описания похож на формат массивов

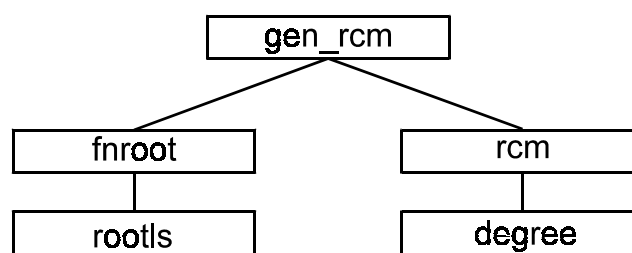


рис. 16. Структура подпрограмм обратного алгоритма Катхилла-Макки и отношения между ними

`xadj` и `adjncy`, $ls[j]$ есть узлы, принадлежащие уровню k , где $xls[k] \leq j < xls[k+1]$. Переменная `lvl` отражает количество уровней, т.е. длину массива `xls`. Массив булевых значений `beta` является вспомогательным, после того как узел помещается в массив `ls`,

соответствующее ему значение изменяется с истины на ложь, чтобы узел не попал в *ls* вторично.

```

unsigned int
TGlobMatrix::rootls(unsigned int root, bool* mask, unsigned int* xls,
                    unsigned int* ls, unsigned int nlvl)
{
    bool* beta = new bool[nnodes+1];
    unsigned int ccsize, lbegin, lvlend, node, jstrt, jstop, nbr;
    for (unsigned int i = 0; i < nnodes; i++) beta[i] = true;
    beta[root] = false;
    ls[0] = root;
    nlvl = 0;
    lvlend = 0;
    ccsize = 1;
    do
    {
        lbegin = lvlend;
        lvlend = ccsize;
        nlvl++;
        xls[nlvl-1] = lbegin;
        for (unsigned int i = lbegin; i < lvlend; i++)
        {
            node = ls[i];
            jstrt = xadj[node];
            jstop = xadj[node+1];
            for (unsigned int j = jstrt; j < jstop; j++)
            {
                nbr = adjncy[j];
                if (mask[nbr] && beta[nbr])
                {
                    ls[ccsize] = nbr;
                    beta[nbr] = false;
                    ccsize++;
                };
            };
        };
    } while (ccsize - lvlend > 0);
    delete[] beta;
    xls[nlvl] = ccsize;
    return nlvl;
};

```

В функции *fnroot* описанная подпрограмма вызывается из цикла, строя корневую структуру уровней для узлов, необходимую для отыскания узлов самого нижнего уровня. Входными параметрами для

подпрограммы `fnroot` являются: `root` – начальный узел, с которого начинается направленный поиск псевдопериферийного узла, и массивы `axdj` и `adjncy`.

```
unsigned int
TGlobMatrix::fnroot(unsigned int root, bool* mask)
{
    unsigned int* xls = new unsigned int[nnodes+1];
    unsigned int* ls = new unsigned int[nnodes+1];
    unsigned int nbr, onlvl, ccsz, nlvl, jstrt, mindeg, node, ndeg,
                kstrt, kstop;
    bool br = false;
    nlvl = rootls(root, mask, xls, ls, 0);
    if (nlvl > nnodes) return nnodes+1;
    ccsz = xls[nlvl];
    // если структура уровней линейна или количество уровней = 1, то
    // root - периферийный узел.
    if (ccsz == nlvl || nlvl == 1) return root;
    do
    {
        jstrt = xls[nlvl-1];
        mindeg = ccsz;
        root = ls[jstrt];
        if (ccsz > jstrt)
        {
            // перебор всех узлов нижнего уровня.
            for (unsigned int i = jstrt; i < ccsz; i++)
            {
                node = ls[i];
                ndeg = 0;
                kstrt = xadj[node];
                kstop = xadj[node+1];
                // перебор всех соседей узла нижнего уровня node и
                // определение его степени
                for (unsigned int j = kstrt; j < kstop; j++)
                {
                    nbr = adjncy[j];
                    if (mask[nbr]) ndeg++;
                };
                // выбор узла с минимальной степенью.
                if (ndeg < mindeg)
                {
                    root = node;
                    mindeg = ndeg;
                };
            };
        };
    };
};
```

```

};
// сохранение старого значения nlvl
onlvl = nlvl;
// вычисление нового
nlvl = rootls(root, mask, xls, ls, nlvl);
if (nlvl > nnodes) return nnodes+1;
if (onlvl >= nlvl) br = true;
} while (nlvl < ccsize && !br);
delete[] xls;
delete[] ls;
return root;
};

```

2.2.4 Процедуры собственно перенумерации узлов

При завершении работы подпрограмма возвращает значение найденного псевдопериферийного узла и процедура `gen_rcm` запускает подпрограмму собственно перенумерации `rcm` (Reverse Cuthill-McKee). Этот алгоритм является небольшой модификацией алгоритма Катхилла-Макки. Суть этой модификации состоит в том, что полученное базовым алгоритмом упорядочение обращается. Исследуя профильные методы, Джордж обнаружил [George 1971], что упорядочение, получаемое обращением упорядочения Катхилла-Макки, часто гораздо сильнее уменьшает профиль, чем первоначальное упорядочение, хотя ширина ленты остается неизменной. Позднее было доказано, что обратное упорядочение всегда не хуже прямого в отношении хранения и обработки оболочки [Liu 1975].

Подпрограмма `rcm` генерирует новую нумерацию, и хранит ее в массиве `perm` так, что `perm[новый_номер] = старый_номер`. Напомним, что перенумерация основана на последовательной нумерации всех еще не пронумерованных соседей узла i в порядке возрастания их степеней, где $i=1, \dots, N$. Цикл `for (unsigned int j = jstrt; j < jstop; j++)` перебирает всех соседей узла `node` и, проверяя, не включен ли он уже в массив `perm`, с помощью массива булевых значений `mask`, вычисляет и упорядочивает их в порядке возрастания их степени.

```

int
TGlobMatrix::rcm(int root, bool* mask, unsigned int* perm)
{
    unsigned int lvlend = -1;
    unsigned int lnbr = 1;

```

```
unsigned int node, jstrt, jstop, fnbr, nbr, lbegin;
unsigned int mindeg;
unsigned int* deg = new unsigned int[nnodes+1];
degree(root, mask, deg);
mask[root]=0;
perm[0] = root;
bool first = true;
do{
    lbegin = lvlend;
    if (first) {lbegin = 0; first = false;};
    lvlend = lnbr;
    for (unsigned int i = lbegin; i < lvlend; i++)
    {
        node = perm[i];
        jstrt = xadj[node];
        jstop = xadj[node+1];
        fnbr = lnbr;
        for (unsigned int j = jstrt; j < jstop; j++)
        {
            nbr = adjncy[j];
            if (mask[nbr])
            {
                mask[nbr] = false;
                perm[lnbr] = nbr;
                lnbr++;
            };
        };
        // были ли добавлены номера в perm
        if (fnbr < lnbr)
        {
            // сортировка методом пузырька в порядке возрастания степени
            for (unsigned int w = fnbr; w < lnbr; w++)
            {
                for (unsigned int q = w; q < lnbr; q++)
                {
                    if (deg[perm[w]] > deg[perm[q]])
                    {
                        mindeg = perm[q];
                        perm[q] = perm[w];
                        perm[w] = mindeg;
                    };
                };
            }; // end of sorting
        };
    };
} while (lnbr > lvlend); // пока новые узлы были добавлены.
// Обращение упорядочивания
```

```

jstrt = 0;
jstop = (nnodes-1)/2;
lnbr = nnodes;
for (int i = jstrt; i < jstop; i++)
{
    node = perm[i];
    perm[i] = perm[lnbr];
    perm[lnbr] = node;
    lnbr--;
};
return ide_ok;
};

```

В своей работе ей необходимо вычислять степень узла и эта процедура вынесена в отдельную подпрограмму *degree* (см. рис. 16). Она вызывается один раз в начале работы *gsm* для построения массива описывающего степени узлов $deg[i]=degree(i)$. В дальнейшем *gsm* использует только массив *deg*.

Входными данными для подпрограммы *degree* являются: массивы *adj* и *adjncy*, *root* – корневой узел, найденный подпрограммой *fnroot*, пустой массив *deg*, в который записываются значения степеней. *ls* – вектор для хранения узлов уровень за уровнем. При добавлении узла в массив *ls*, соответствующий ему компонент в массиве *beta* меняется с истины на ложь, так же как и в подпрограмме *rootls*, это позволяет избежать повторного включения узла в массив *ls*. Здесь также, как и в *rootls* строится структура уровней, но здесь она не нужна как таковая, здесь лишь нужно обрабатывать узлы уровень за уровнем, и так как на текущем шаге обрабатывается уровень собранный на предыдущем и нет необходимости доступа к этому уровню в последующем, то ключевой массив структуры уровней *xls* не строится. *Lvlend* и *lbegin* обозначают начало и конец текущего уровня.

```

int
TGlobMatrix::degree(int root, bool* mask, unsigned int* deg)
{
    unsigned int* ls = new unsigned int[nnodes+1];
    bool* beta = new bool[nnodes+1];
    unsigned int node, jstrt, jstop, nbr, ideo, ccsize, lbegin, lvlend;
    for (int i = 0; i < nnodes; i++)
        beta[i] = true;
    ls[0] = root;
    beta[root] = false;
    ccsize = 1;

```

```

lvlend = 0;
bool first = true;
do
{
    lbegin = lvlend; // конец предыдущего уровня = начало текущего
    if (first) {first = false; lbegin = 0;};
    lvlend = ccsize; // конец текущего уровня = последний элемент в
    // массиве ls
    for (unsigned int i = lbegin; i < lvlend; i++)
    {
        node = ls[i];
        jstrt = xadj[node];
        jstop = xadj[node+1];
        ideg = 0;
        // вычисление степени узла - пересчет его соседей
        for (unsigned int j = jstrt; j < jstop; j++)
        {
            nbr = adjncy[j];
            if (mask[nbr])
            {
                ideg++;
                if (beta[nbr])
                {
                    beta[nbr] = false;
                    ls[ccsize] = nbr;
                    ccsize++;
                };
            };
        };
        deg[node] = ideg;
    };
} while (ccsize-lvlend > 0); // пока были добавлены узлы в ls
delete[] ls;
delete[] beta;
};

```

Главная процедура разбиения `gen_gcm` лишь объявляет и обнуляет массив `mask` последовательно исполняет подпрограмму `fnroot` и `gcm` и генерирует по готовой перенумерации обратную `iperm[perm[i]] = i`, т.е. `iperm[старый_номер] = новый_номер`. Она довольно проста, и описываться она nebude.

2.2.5 Оценка эффективности перенумерации узлов

Для оценки эффективности перенумерации узлов, с точки зрения экономии оперативной памяти, был введен некий интегральный критерий показывающий общее количество чисел, которые необходимо было держать в памяти при профильной схеме хранения. Для матрицы с количеством уравнений около 1300 этот критерий принимал значения порядка 250000 до перенумерации и 30000 после. В среднем, эффективность – 8 кратное уменьшение объемов используемой оперативной памяти и многократное сокращение времени решения системы уравнений.

2.2.6 Выбор схемы хранения и решения системы уравнений

Алгоритм перенумерации Катхилла-Макки прежде всего предназначен для повышения эффективности расчета с точки зрения экономии использования оперативной памяти и уменьшения времени решения. Безусловно применение его не было бы эффективным без применения соответствующих схем хранения и решения системы уравнений. Поэтому, для достижения максимального эффекта была выбрана одна из самых эффективных, и в то же время простых схем хранения — *профильная схема* [Джордж 1984].

2.2.7 Реализация профильного метода хранения

Профильный метод тесно связан с понятием ленточных методов и, в какой-то мере является продолжением их развития. Напомним, что ленточные методы используют разреженность матрицы коэффициентов и их группировку около главной диагонали симметричных, положительно определенных матриц. На рисунке 17 показан ленточный принцип хранения матрицы A . Помимо главной диагонали по этой схеме хранится еще и все элементы, попавшие в рамку. В таблице представлена зависимость от номера строки матрицы A ширины ленты для i -й строки. Максимальная ширина ленты i -й строки называется шириной ленты матрицы A . Эта величина определяет количество элементов, удерживаемых при хранении матрицы в памяти.

профильный метод будет не хуже ленточного. На рисунке 18 показан пример профильного представления матрицы A , той же матрицы, что и на рисунке 17.

Единственное негативное отличие от ленточного метода — некоторое усложнение схемы хранения с машинной точки зрения. Авторы Джордж и Лю [Джордж 1984] говорят о том, что наиболее применимой является схема Дженнинга [Jennings 1966]. В каждой строке матрицы хранятся все от первого ненулевого элемента до диагонального. Эти отрезки строк размещены в последовательных позициях одномерного массива (см. рис. 19).

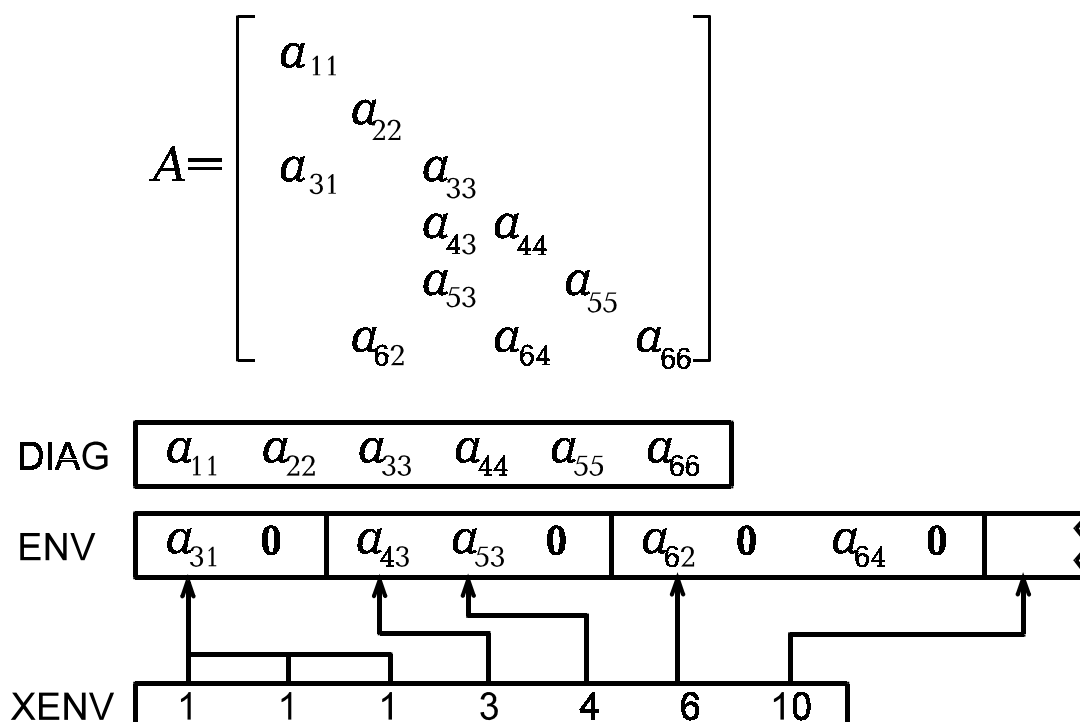


рис. 19. Пример профильной схемы хранения.

Но эта схема разрабатывалась, по всей видимости, не для счета на персональных компьютерах, и уж тем более без учета специфики Windows. В оригинале предлагается хранить отдельно диагональные элементы в линейном массиве и отдельно нижнюю треугольную матрицу также в линейном массиве. Структурирование линейного массива предлагалось осуществить дополнительным массивом, содержащем адреса начал каждой строки матрицы. Но Windows не позволяет выделить память для переменной занимающей больше 64 килобайт.

Поэтому я решил использовать двумерный массив. В C++ двумерные массивы в общем случае на совсем двумерные – более точное их определение это массивы массивов. Первый массив содержит ссылки (адреса) на начала других массивов, в которых и содержатся данные. Такая схема не требует одинаковой длины вложенных массивов, они

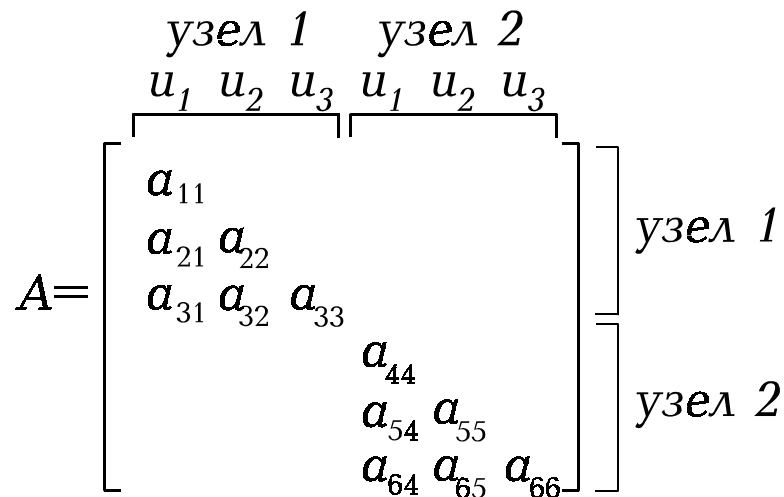


рис. 20. Блочная структура глобальной матрицы жесткости.

могут быть произвольными и я воспользовался этим обстоятельством. Каждый массив в моей схеме хранения представлял собой строку матрицы от первого ненулевого элемента до диагонали. Безусловно, предлагаемые алгоритмы решения системы изменились с учетом этой особенности, но принципы остались теми же.

Но глядя на схему сразу становится ясно, что вся структура, с точностью до каждого элемента, должна быть известна заранее, еще до заполнения матрицы. Эту структуру можно установить по матрице инцидентий и списку перенумерации узлов. Известно, что на каждый узел в глобальной матрице жесткости, приходится три столбца – три перемещения. Поэтому длина строки вычисляется исходя из максимальной разности номеров узлов связанных с тем, к которому относится заполняемая строка плюс величина зависящая от перемещения этого узла, к которому относится данная строка (см. рис. 20). Глобальная матрица жесткости имеет блочную структуру благодаря тому, что коэффициенты включающиеся в нее включаются блоками 3x3. На рисунке видно, что для первого узла условно не имеющего соседей максимальная разность номеров соседей равна нулю, поэтому, если отталкиваться исключительно от этих данных, то длина хранимых строк

1, 2, 3 тоже будет равняться нулю. Это приведет к потере элементов a_{21}, a_{31} и a_{32} , что недопустимо. Поэтому необходимо вводить вспомогательную величину, которая будет добавляться к длине строки. Она должна равняться 0 при строке соответствующей линейному перемещению u_1 , 1 — поворотному перемещению u_2 , 2 — поворотному перемещению u_3 .

Еще при сборке конструкции глобальной матрицы жесткости необходимо учитывать и тот факт, что при учете закреплений узлов конструкции, строки и столбцы в глобальной матрице вырезаются. Поэтому кроме массива перенумерации вводится еще два массива, отражающие эти изменения: *mask* и *cut*. Теперь чтобы получить индекс степени свободы конструкции с индексом i необходимо следующее обращение *cut*[i], но перед этим проверить по массиву булевых значений *mask* не исключена ли эта степень свободы.

2.2.8 Заполнение глобальной матрицы жесткости

После определения структуры глобальной матрицы можно приступить к ее заполнению. Генерация локальной матрицы жесткости уже была обсуждена выше и здесь она обсуждаться не будет, здесь нас уже интересует то, как собирать глобальную матрицу жесткости.

Эта операция на данном этапе, когда все необходимые данные уже известны, довольно проста. Она осуществляется функцией *place_elem*. В качестве входного параметра ей передается: номер элемента i , локальная матрица жесткости для него с размерностью 9x9, список инцидентий *inc* и вспомогательные массивы *mask* и *cut*. В глобальную матрицу локальная переносится по блокам, каждый блок 3x3 это матрица реакций возникающих в данном узле от перемещения другого узла элемента. Каждый блок заносится в глобальную матрицу функцией *place_block*, которой передается позиция блока в локальной нумерации элемента и смещения начала этого блока в глобальной матрице. Такие смещения вычисляются с использованием списка инцидентий следующим образом:

$$offs_col = inc[i,k], \quad offs_row = inc[i,l], \quad \text{где}$$

i — номер элемента, k — позиция блока в локальной нумерации элемента по столбцам, l — по строкам. $l, k = 0 \dots 2$.

Необходимо также еще отметить, что элементы блоков не присваиваются значениям в ячейках глобальной матрицы, а прибавляются к уже существующим значениям.

После заполнения матрицы жесткости необходимо заполнить столбец свободных членов, который отвечает узловой нагрузке, приложенной к конструкции. Здесь также нагрузка от различных типов загрузки сводится в узел, суммируется и записывается в вектор свободных членов. На этом этапе также используется информация массивов `mask` и `cut` об отброшенных степенях свободы.

2.2.9 Метод решения для профильной схемы

Для решения уже готовой системы, хранящейся по профильной схеме, необходимо применять профильные методы решения систем. Выбранный метод должен полностью использовать выгодные особенности решаемой системы: положительная определенность матрицы, ее симметричность, разреженность и ленточность. К тому же метод должен учитывать схему хранения и то обстоятельство, что в памяти удерживается только нижняя половина матрицы.

Авторы Джордж и Лю, для решения системы, хранящейся по профильной схеме, рекомендуют использовать метод Холесского [Джордж 1984]. Это точный, не итерационный метод. Отсюда и ряд достоинств: во-первых, это гарантированно точный результат, а при машинном счете это, как правило, результат, очень близкий к точному; а во-вторых, заранее известно количество операций, необходимых для получения ответа. В нем отсутствуют недостатки и проблемы итерационных методов, основной из которых является проблема сходимости к точному решению. Дело в том, что итерационные методы не дают гарантии, что будет получено точное решение за разумное количество операций, они лишь говорят, что решение при их применении будет приближаться к точному. К тому же метод Холесского легко приспособливается для профильной схемы хранения и позволяет для всех операций и преобразований использовать те же матрицы, что были для него входными данными, т.е. нет необходимости дублировать довольно ресурсоемкие данные.

Решение производится в три этапа: вычисление разложения Холецкого LL^T для симметричной матрицы методом окаймления, решение нижней треугольной системы и верхней треугольной

$$Ly = b$$

$$L^T x = y.$$

Метод окаймления основывается на представлении исходной матрицы A в виде

$$A = \begin{pmatrix} M & u \\ u^T & s \end{pmatrix},$$

где M ведущая главная подматрица A с разложением Холецкого $L_M L_M^T$. Тогда множитель L матрицы A можно записать в виде

$$L = \begin{pmatrix} L_M & 0 \\ w^T & t \end{pmatrix},$$

где $L_M w = u$ и $t = \sqrt{s - w^T w}$. Множитель Холецкого можно вычислять строка на строкой, начиная с матрицы a_{11} порядка 1 переходя к матрицам все большего порядка. На каждом цикле решается система уравнений $L_M w = u$ и определяется i -я строка разложения. При этом хранение новой строки осуществляется в той же матрице A , т.е. происходит как бы плавное превращение матрицы A в матрицу L . В той же книге [Джордж 1984] приводится доказательство того, что профиль матрицы L полностью соответствует профилю исходной матрицы A . Это полезное свойство полностью реализуется в используемой подпрограмме решения системы.

Применение метода Холецкого было полезно еще и тем, что в нем

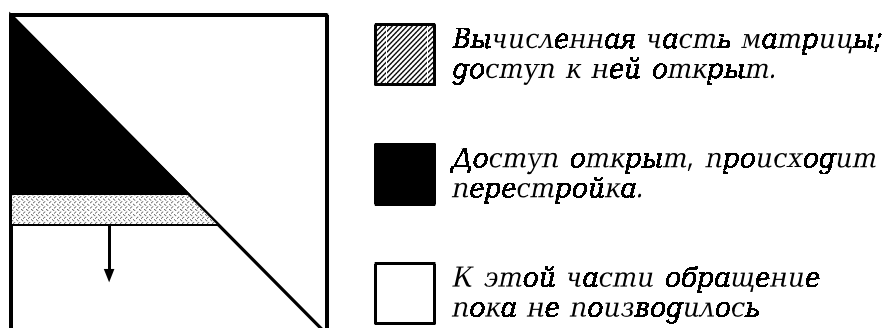


рис. 21. Иллюстрация работы одного из вариантов разложения методом Холецкого называемом методом окаймления

заложен показатель положительной определенности матрицы коэффициентов. Индикатором является положительность множителя Холесского t , это было в какой-то мере показателем правильности сборки глобальной матрицы.

2.2.9 Реализация алгоритма решения системы уравнений

Эта подпрограмма состоит из трех выполняемых последовательно процедур: разложения — подпрограмма `esfct` (Envelope Symmetric FaCTORIZATION), решения нижней треугольной матрицы — подпрограмма `elslv` (Envelope Lower SoLve) и верхней треугольной матрицы — подпрограмма `euslv` (Envelope Upper SoLve). Математическая формулировка алгоритма разложения была сформулирована выше. Для построения алгоритма необходимо еще установить схему работы этого алгоритма. Как было сказано выше разложение вычисляется строка за строкой и эти строки постепенно увеличивают матрицу L в размерах (см. рис. 21). Из рисунка видно, что применительно к нашей схеме хранения выбор этого варианта метода (из трех возможных, описываемых Джорджем [Джордж 1984]) разложения Холесского очень удачен. Дело в том, что для решения системы уравнений $L_M w = u$ (см. п. 2.2.9) на каждом шаге разложения можно использовать процедуру решения нижней треугольной системы, надо лишь слегка приспособить ее решать не целую систему, а только ее часть. Ведь на каждом шаге решается система, в которой коэффициентами являются вычисленная часть разложения (см. рис. 21), а свободными членами — вычисляемая строка без диагонального элемента. После получения этого частного решения строка заменяется на ответы решения этой системы, а множитель Холесского вычисляется по формуле $t = \sqrt{s - w^T w}$, где s — это исходный диагональный элемент, а $w^T w$ — это есть сумма произведений элементов новой строки самих на себя. Затем вычисленный множитель Холесского, который должен быть всегда положительным, заменяет собой диагональный элемент.

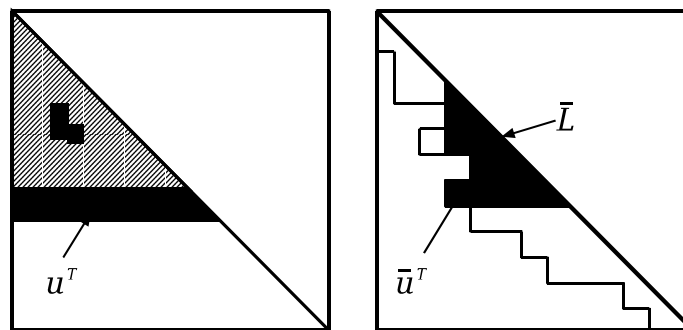


рис. 22. Иллюстрация к способу использования разреженности матрицы. В вычислении \bar{w} по \bar{u} участвует лишь \bar{L} .

Кроме того здесь замечательно учитывается разреженность и диагональность матрицы коэффициентов (см. рис. 22). В подпрограмме `esfct` учитывается то обстоятельство, что векторы \bar{u} короткие, так как мы работаем с профильной матрицей.

Необходимо также еще заметить, что моя схема хранения в отличие от авторской [Джордж 1984] еще более упрощает использование процедуры. Для пояснения приведу исходный код этой подпрограммы:

```
int
TGlobMatrix::esfct()
{
    if (diag[0] < EPSILON) return ide_zero_diag;
    unsigned int iband, ifirst, ixenv;
    long double t, s;
    s = diag[0];
    diag[0] = sqrtl(s);
    for (int i = 1; i < neq; i++)
    {
        iband = xenv[i]; // в массиве xenv содержатся длины каждой
                        // строки
        t = diag[i];
        if (iband > 0)
        {
            ifirst = i - iband;
            elslv(iband, ifirst, gk[i]);
            for (int j = 0; j < iband; j++)
            {
                s = gk[i][j];
                t -= s*s;
            };
            if (t < EPSILON)
```

```

        return ide_holessky;
    };
    diag[i] = sqrtl(t);
};
return ide_ok;
};

```

Вызов подпрограммы решения нижней треугольной матрицы выглядит следующим образом: `elslv(int neqs, int pstrt, long double* rhsl)`. Где `neqs` — количество уравнений, по сути это длина вектора \bar{u} , `pstrt` — индекс начало решения, то есть индекс первого ненулевого элемента вектора \bar{u} . Переменная `rhsl` — это ссылка на вектор \bar{u} , его адрес в памяти. В подпрограмме `esfct` эта переменная записывается как `gk[i]`.

Структурой глобальной матрицы является массив массивов и запись обращения к его элементу выглядит так: `gk[i][j]`. Символ `gk` обозначает ссылку на массив, в котором содержатся ссылки на другие массивы, а запись `gk[i]` ссылается на массив, в котором хранится i -я строка. Поэтому внутри подпрограммы `elslv` переменная `rhsl` будет обычным линейным массивом.

```

int
TGlobMatrix::elslv(int neqs, int pstrt, long double* rhsl)
{
    int ifirst, last, iband, imband, kstrt, kstop, l;
    long double s;
    ifirst = 0;
    while (rhsl[ifirst] < EPSILON && rhsl[ifirst] > -EPSILON
           && ifirst < neqs - 1)
        ifirst++;
    last = 0;
    for (int i = ifirst; i < neqs; i++)
    {
        iband = xenv[pstrt+i];
        if (iband > i) imband = i; else imband = iband;
        s = rhsl[i];
        l = i - imband;
        rhsl[i] = 0;
        if (iband != 0 && last >= l)
        {
            kstrt = iband - imband;
            kstop = iband;
            for (int j = kstrt; j < kstop; j++)
            {
                s -= gk[pstrt+i][j]*rhsl[l];
            }
        }
    }
}

```

```

        l++;
    };
};
if (s > EPSILON || s < -EPSILON)    // s != 0;
{
    rhs1[i] = s/diag[pstrt+i];
    last = i;
};
};
return ide_ok;
};

```

Подпрограмма euslv решает задачу $L^T x = y$, причем L хранится по той же схеме, что и в elslv. Это значит, что мы имеем удобный доступ к столбцам L^T (строкам L) и разреженность можно использовать полностью, применяя схему внешних произведений [Джордж 1984]. Схема его приведена на рисунке 23.

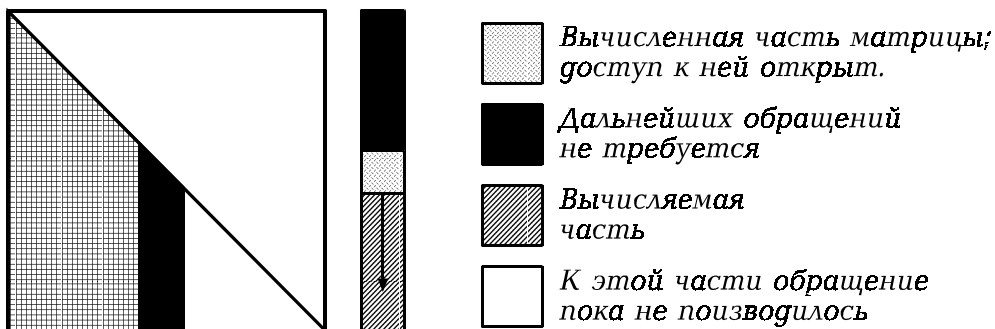


рис. 23. Схема метода внешних произведений.

Исходный код подпрограммы euslv:

```

int
TGlobMatrix::euslv()
{
    // решение производится для варианта со столбцовой схемой хранения
    // см. А.Джорж Дж.Лю "Численное решение больших разреженных систем
    // уравнений"
    // п 2.2.1
    int iband, k;
    long double s;
    for(int i = neq - 1; i >= 0; i--)
    {
        if (rhs[i] > EPSILON || rhs[i] < -EPSILON)

```

```
{
  s = rhs[i]/diag[i];
  rhs[i] = s;
  iband = xenv[i];
  if (iband > i) iband = i;
  if (iband != 0)
  {
    k = i - iband;
    for (int l = 0; l < iband; l++)
    {
      rhs[k] = rhs[k] - s*gk[i][l];
      k++;
    };
  };
};
return ide_ok;
};
```

2.2.10 Эффективность алгоритма решения системы.

Осуществленный в программе метод показывает стабильную работу при довольно точном решении и своей быстроте. Субъективно, само решение системы занимает примерно треть времени решения всей конечно-элементной задачи в перемещениях. После завершения работы над этой частью я провел некоторые тесты алгоритма решения системы уравнений. Были сохранены исходная матрица жесткости и исходный вектор свободных коэффициентов, а после получения решения оно было подставлено в исходные уравнения равновесия. Полученные значения этих уравнений сравнивались с исходно заданными. Получалось, что нулевые элементы отличались от нуля величиной порядка 10^{-4} от максимального значения перемещения в конструкции.

2.2.11 Решение задачи в напряжениях

После нахождения решения в перемещениях для полного инженерного решения задачи необходимо было решить задачу в напряжениях, ведь конечной целью большинства инженерных расчетов является нахождение максимальных напряжений. Инженеров при

расчете на прочность тем более интересуют прежде всего напряжения, а уж потом усилия и перемещения.

Для расчета по напряжениям у нас известны: узловые перемещения элементов, геометрия элементов, их физические характеристики. Для того, чтобы найти перемещения для нашего конечного элемента необходимо вспомнить принцип вывода матрицы жесткости для него. Сначала задаемся полиномом, аппроксимирующим функцию прогиба:

$$w = a_1 + a_2x + a_3y + a_4x^2 + a_5xy + a_6y^2 + a_7x^3 + a_8x^2y + a_9xy^2 + a_{10}y^3$$

или в матричной форме

$$w = [1 \ x \ y \ x^2 \ xy \ y^2 \ x^3 \ x^2y \ xy^2 \ y^3] \bar{a}$$

Дифференцируя по x и y , получим выражения для полей углов поворота и перемещений:

$$\begin{bmatrix} w \\ \varphi^x \\ \varphi^y \end{bmatrix} = \begin{bmatrix} w \\ \frac{dw}{dy} \\ -\frac{dw}{dx} \end{bmatrix} = \begin{bmatrix} 1 & x & y & x^2 & xy & xy^2 & x^3 & x^2y & xy^2 & y^3 \\ 0 & 0 & 1 & 0 & x & 2y & 0 & x^2 & 2xy & 3y^2 \\ 0 & -1 & 0 & -2x & -y & 0 & -3x^2 & -2xy & -y^2 & 0 \end{bmatrix} \bar{a}$$

Подставив координаты точек 0, 1, 2, 3 элемента, получим:

$$\begin{bmatrix} w_0 \\ w_1 \\ w_1^x \\ w_1^y \\ w_2 \\ w_2^x \\ w_2^y \\ w_3 \\ w_3^x \\ w_3^y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & x_1 & y_1 & x_1^2 & x_1y_1 & y_1^2 & x_1^3 & x_1^2y_1 & x_1y_1^2 & y_1^3 \\ 0 & 0 & 1 & 0 & x_1 & 2y_1 & 0 & x_1^2 & 2x_1y_1 & 3y_1^2 \\ 0 & -1 & 0 & -2x_1 & -y_1 & 0 & -3x_1^2 & -2x_1y_1 & -y_1^2 & 0 \\ 1 & x_2 & y_2 & x_2^2 & x_2y_2 & y_2^2 & x_2^3 & x_2^2y_2 & x_2y_2^2 & y_2^3 \\ 0 & 0 & 1 & 0 & x_2 & 2y_2 & 0 & x_2^2 & 2x_2y_2 & 3y_2^2 \\ 0 & -1 & 0 & -2x_2 & -y_2 & 0 & -3x_2^2 & -2x_2y_2 & -y_2^2 & 0 \\ 1 & x_3 & y_3 & x_3^2 & x_3y_3 & y_3^2 & x_3^3 & x_3^2y_3 & x_3y_3^2 & y_3^3 \\ 0 & 0 & 1 & 0 & x_3 & 2y_3 & 0 & x_3^2 & 2x_3y_3 & 3y_3^2 \\ 0 & -1 & 0 & -2x_3 & -y_3 & 0 & -3x_3^2 & -2x_3y_3 & -y_3^2 & 0 \end{bmatrix} \bar{a}$$

$$\bar{Z} = L\bar{a}, \text{ откуда } \bar{a} = L^{-1}\bar{Z},$$

$$w = [1 \ x \ y \ x^2 \ xy \ y^2 \ x^3 \ x^2y \ xy^2 \ y^3] L^{-1} \bar{Z}$$

По вектору w , используя зависимости Коши построим вектор относительных деформаций:

$$\varepsilon = -z \begin{bmatrix} \frac{\partial^2 w}{\partial x^2} \\ \frac{\partial^2 w}{\partial y^2} \\ 2 \frac{\partial^2 w}{\partial x \partial y} \end{bmatrix} = -z \begin{bmatrix} 0 & 0 & 0 & 2 & 0 & 0 & 6x & 2y & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 2x & 6y \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 4x & 4y & 0 \end{bmatrix} L^{-1} \bar{Z} = BL^{-1} \bar{Z}$$

Вектор напряжений:

$$\bar{\sigma} = DBL^{-1} \bar{Z},$$

где

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

Еще одной характерной особенностью этого конечного элемента является то, что в узлах конечного элемента нельзя найти напряжения, их можно найти лишь в его центре. Но беда в том, что при решении в перемещениях мы избавились от этой степени свободы и перемещение в этой точке неизвестно, а это перемещение должно быть в векторе \bar{Z} . Для его нахождения можно воспользоваться методом Гаусса. Локальную матрицу жесткости с не исключенной степенью свободы можно представить так:

$$\begin{array}{|c|c|} \hline R_{11} & R_{12} \\ \hline R_{21} & R_{22} \\ \hline \end{array} \begin{array}{|c|} \hline Z_1 \\ \hline Z_2 \\ \hline \end{array} = \begin{array}{|c|} \hline P_1 \\ \hline P_2 \\ \hline \end{array}$$

где Z_1 – это неизвестное перемещение в центре конечного элемента, а \bar{Z}_2 – девять узловых перемещений, полученных статическим расчетом. Так как вся нагрузка при статическом расчете сводится в узлы, то нагрузка на центр треугольника равна нулю. Поэтому:

$$R_{11}Z_1 + R_{12}Z_2 = P_1 = 0, \text{ отсюда}$$

$$Z_1 = R_{11}^{-1}P_1 - R_{11}^{-1}R_{12}Z_2$$

После определения перемещения Z_1 оно подставляется в вектор \bar{Z} .

Теперь, когда определены все десять перемещений конечного элемента вектор напряжений для середины элемента находится по формуле:

$$\bar{\sigma} = DBL^{-1}\bar{Z}$$

Матрица, определяющая в какой точке находятся напряжения — это матрица B . В нее подставляются координаты центра элемента (0,0).

Читатель может заметить неэффективность работы алгоритма нахождения напряжений в том что приходится генерировать локальные матрицы жесткости дважды: при решении в перемещениях и при решении в напряжениях. Но примененный подход оправдан тем, что для того, чтобы не считать матрицы жесткости дважды необходимо хранить их в памяти, что весьма накладно. Поэтому я сделал выбор в пользу экономии ресурсов, кроме того, несмотря на повторение самой трудоемкой части решения в перемещениях, решение а напряжениях получается намного быстрее.

3. Оценка эффективности программы в целом

3.1 Контрольный пример

После завершения рассмотрения всей расчетной части и оценки каждой ее части отдельно необходимо рассмотреть вопрос общей эффективности расчетной части. Насколько точным является решение и насколько быстро оно сходится к точному при сгущении сетки.

Для рассмотрения этого вопроса прежде всего надо определиться с точным решением, а это возвращает нас к уже обсуждавшемуся вопросу о проблемах существования точного решения для тонких жестких плит. Как было сказано выше для этого класса конструкций существуют точные аналитические решения, среди которых самым известным случаем является прямоугольная сплошная плита, защемленная по контуру и нагруженная равномерно распределенной нагрузкой, направленной перпендикулярно плоскости плиты. Существуют и другие случаи существования аналитического решения, но они или менее доступны в литературе, или более сложны для оперативной проверки. Поэтому я остановился именно на этом случае и протестировал численное

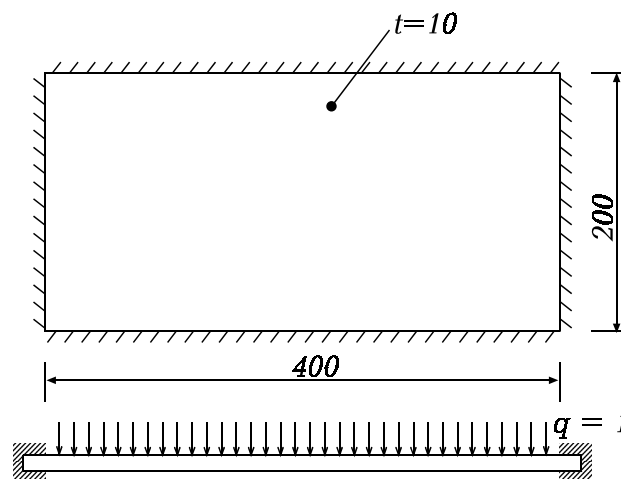


рис. 24. Расчетная схема контрольного примера

решение на нем.

Здесь единицы измерений не важны, применяется такая же схема, что и в Cosmos/m. Пользователь сам выбирает систему измерений (см. прил. 1. Руководство пользователя). Размеры плиты в плане 200x400, толщина

плиты 10, распределенная по площади нагрузка 1, модуль упругости $E = 2.06E^5$, коэффициент Пуассона $\mu = 0.25$.

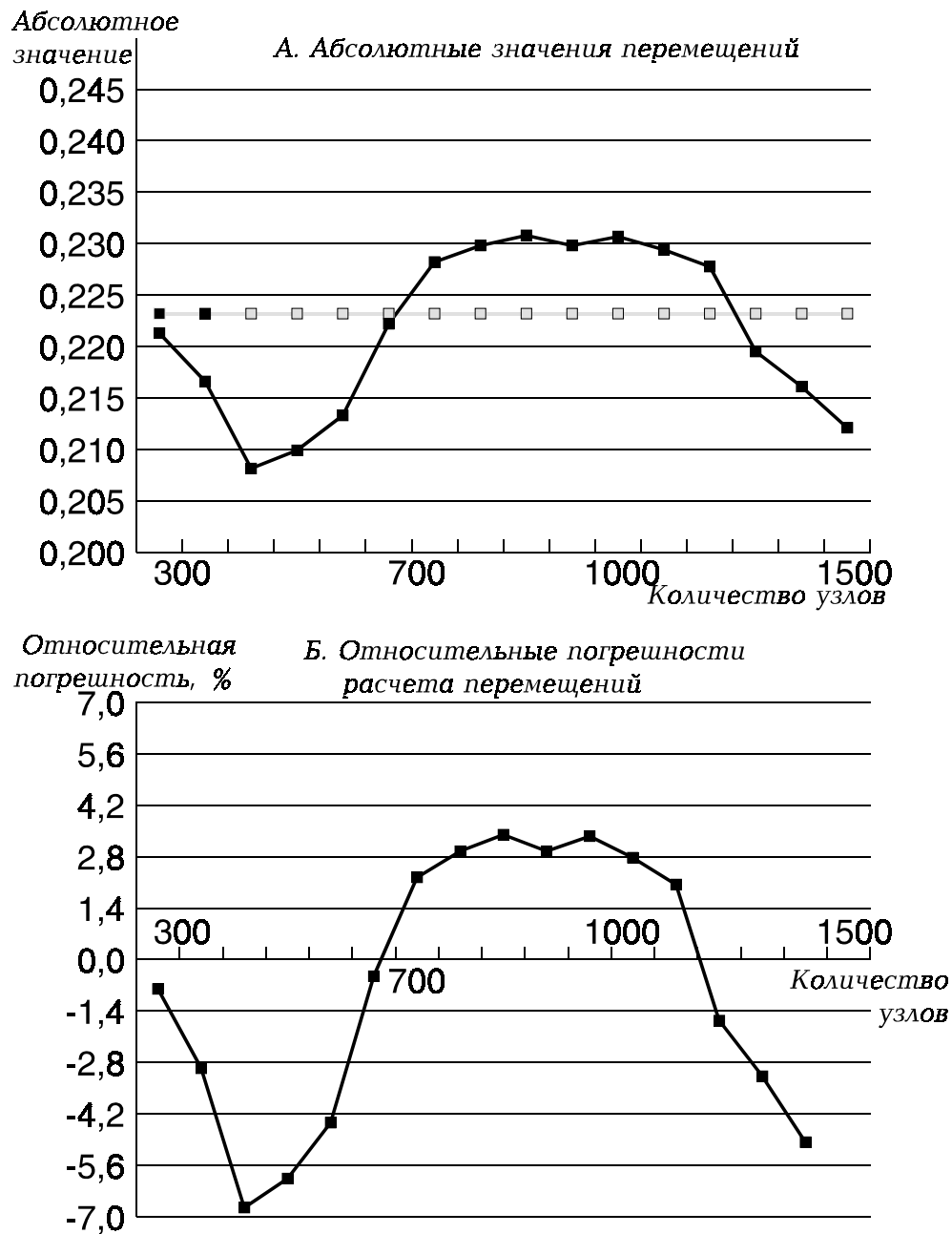


рис. 25. Тест на сходимость и устойчивость решения. А — Абсолютные значения перемещений для точного и полученных решений в зависимости от густоты сетки. Б — Значения относительной погрешности полученных решений в зависимости от густоты сетки.

Для такого случая расчета плиты существует аналитическое решение. Максимальные перемещения в центре плиты:

$$w = \frac{qa^4}{386D},$$

где $D = \frac{Eh^3}{12(1-\mu^2)}$ — цилиндрическая жесткость.

a — длина короткой стороны.

В нашем случае имеем:

$$D = \frac{Eh^3}{12(1-\mu^2)} = \frac{210000 \cdot 10^3}{12 \cdot (1-0.25^2)} = 186666666.6$$

$$w = \frac{1 \cdot 200^4}{386 \cdot 186666666.6} = 0.223214$$

Максимальный прогиб в центре плиты равен 0,223214. Можно вычислить и напряжения, зная наибольший момент можно рассчитать напряжения на единице ширины. Максимальный момент достигается у опоры ровно посередине длинной стороны:

$$M_y = -\frac{qa^2}{12} = -\frac{1 \cdot 200^2}{12} = -3333.333$$

Тогда напряжение равно:

$$\sigma = \frac{M}{W} = \frac{6 \cdot M}{b \cdot h^2} = -\frac{6 \cdot 3333.333}{1 \cdot 10^2} = -200$$

3.2 Анализ сходимости решения и его устойчивости

Сгущение сетки производилось от значений количества узлов 300 до 1000. Результаты расчетов и сравнение их с точным ответом приведены на графике (см. рис. 25).

Из графиков видно, что на первой части графика, когда количество узлов меньше 500, наблюдается некоторая неустойчивость решения. Затем от 500 до примерно 1200, погрешность решения фиксируется на уровне 3%. При дальнейшем увеличении густоты сетки точность решения опять теряется и достигает -5%. Такое поведение можно объяснить и на каждом участке причина своя.

При малой густоте сетки неустойчивость решения вполне понятна, мы пользуемся одним из самых простых из всего многообразия конечных элементов, предназначенных для решения нашей задачи. А для простых элементов, как известно, требуются более густые сетки.

Дело в том, что у конечных элементов есть недостатки, в нашем случае, у нашего треугольного конечного элемента, по всей видимости, не соблюдаются условия совместности. Наверное поэтому невозможно найти напряжения на границе конечного элемента, в его узлах. Поэтому,

для меньшего несоблюдения этих условий на границе какого-то конкретного конечного элемента приходится уменьшать его размеры, то есть сгущать сетку. Существует множество сложных элементов, не только треугольных, где вводятся дополнительные точки внутрь, и на его грани, для сведения этих погрешностей к нулю. Но такие элементы имеют большую размерность локальной матрицы жесткости и намного более сложны, чем наш конечный элемент. Получается, что хотя они позволяют понизить густоту сетки, размерность решаемой системы обычно не уменьшается.

Неустойчивость решения при увеличении числа узлов выше 1200 можно объяснить несовершенством алгоритма решения, нахождения

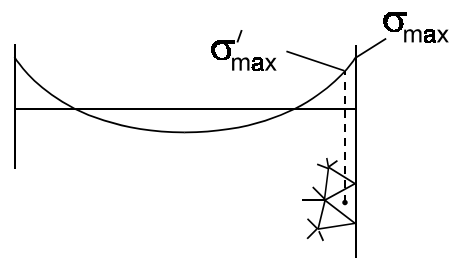


рис. 26. Ошибка получения максимального напряжения из-за перехода к дискретной задаче. Напряжения не могут быть найдены не границе конечного элемента и для получения напряжения на грани плиты необходимо было бы бесконечно сгущать сетку.

матриц жесткости, учета граничных условий. Ведь, если вспомнить не существовании локальной матрицы жесткости для равнобедренного конечного элемента, вырожденности одной из матриц ее составляющих, это сразу становится объяснимым, так как в этом случае слегка меняется положение одной из точек. Безусловно, это смещение крайне мало, но, вероятно эта ошибка накапливается при решении задач большой размерности.

Хочется еще упомянуть о еще одном вероятном источнике неустойчивости решения. При решении методом конечных элементов мы всегда переходим от континуальной к дискретной задаче, мы получаем лишь аппроксимацию точного решения в виде значений функции перемещений в наборе точек с известными координатами. После получения этих значений мы останавливаемся и целиком полагаясь на густоту сетки выбираем максимальное значение и считаем, что оно и есть максимальное на самом деле. Но мы забываем, что, если область экстремальных значений покрыта недостаточным количеством узлов, то максимальное значение можно и вовсе потерять. В своих экспериментах,

доподлинно зная точку наибольших перемещений (геометрический центр конструкции), я заранее, принудительно поставил туда узел.

То же можно сказать и о напряжениях, но с ними ситуация несколько отличается. Я не проводил тестов на сходимость решения в напряжениях по одной причине. Нахождение напряжений является локальной задачей для конечного элемента и напрямую не зависит от размерности системы, так как они находятся исключительно по данным, касающимся только этого элемента. Напряжения зависят от полученных глобальным расчетом перемещений и через них они косвенно зависят от размерности задачи. В целом поведение напряжений должно быть таким же по характеру, что и перемещений.

В данной конкретной задаче найти точно найти максимальные по модулю напряжения вообще не представляется возможным из-за той же дискретности задачи. Напряжения не могут быть найдены на границе конечного элемента и для получения напряжения на грани плиты необходимо было бы бесконечно сгущать сетку (см. рис. 26). При количестве узлов около 1500 максимальные напряжения получились минус 197,2.

Исходя из вышесказанного можно сделать вывод. *Необходимо всегда критически относиться к получаемым результатам и помнить, что в расчете мы имеем дело с математической моделью, основанной на ряде допущений и отступлений.*

Приложения

Приложение 1. Руководство пользователя.

FEPLAR - Finite Element PLate AnalyzeR. Руководство пользователя.

Программа FEPLAR предназначена для инженерного расчета на статическую нагрузку тонких жестких плит произвольной конфигурации. Под плитой понимается призматическое тело, ограниченное двумя параллельными плоскостями, расстояние между которыми называется толщиной плиты. Тонкая жесткая плита - это плита у которой отношение ее минимального габарита к толщине меньше 50 (в них можно пренебречь т.н. мембранными усилиями – продольными растягивающими и сдвигающими усилиями в срединной плоскости).

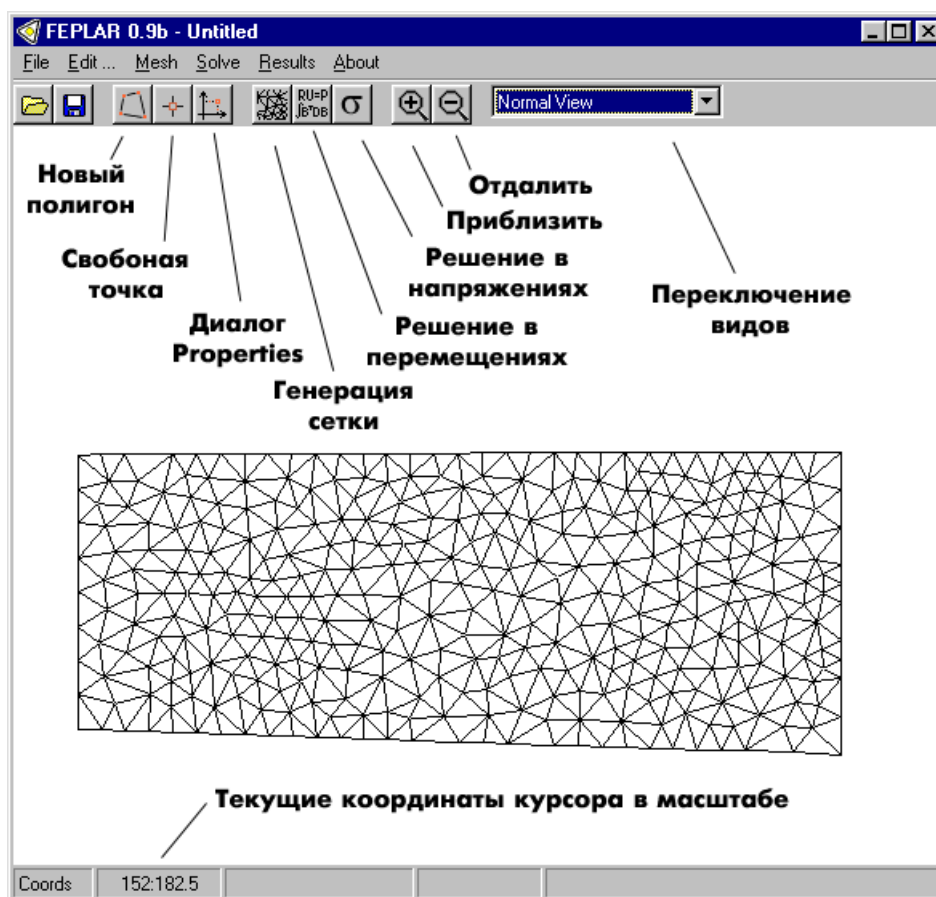
Основные особенности программы:

- на геометрию конструкции не налагается никаких ограничений (кроме здравого смысла). Она может иметь отверстия в любом количестве и конфигурации. Может иметь области с другими физическими характеристиками (модуль Юнга, коэффициент Пуассона, толщина) и распределенной нагрузкой.
- Формат сохраняемых файлов текстовый и имеет простую структуру.
- Возможность ввода мышью и точного позиционирования геометрических объектов.
- Встроенный адаптивный алгоритм генерации сетки, параметризуемый по минимальному углу в сетке (до 31-33 градусов) и по максимальной площади элемента.
- В основе численного решения лежит метод конечных элементов
- Перенумерация узлов по алгоритму Катхилла-Макки с целью сжатия глобальной матрицы жесткости, экономии памяти и сокращения времени решения.
- Профильная схема хранения глобальной матрицы жесткости позволяет максимально сократить необходимый объем памяти.
- Использование 10-байтовых чисел long double позволяет максимально сократить потерю точности при решении системы.
- Решение в перемещениях и напряжениях.

- Визуализация в виде изолиний по перемещениям (UZ, RotX, RotY) и по напряжениям (Sigma X, Sigma Y, Tau XY).
- Табличное представление с возможностью сортировки по каждому перемещению и напряжению
- Возможность вывода (поэлементного разбиения, закрепления и нагрузки) входного файла для программного комплекса Cosmos/m.
- Возможность вывода матрицы жесткости для любого элемента в текстовом виде.

Содержание:

1. Выбор необходимого масштаба.
2. Ввод исходных данных.
 - 2.1 Ввод геометрических параметров конструкции.
 - 2.2 Задание нагрузок.
 - 2.3 Наложение связей.
 - 2.4 Определение физических характеристик.
3. Генерация сетки.
4. Решение задачи.
 - 4.1 Решение в перемещениях.
 - 4.2 Решение в напряжениях.
5. Просмотр результатов расчета.
 - 5.1 Просмотр в табличной форме.
 - 5.2 Построение изолиний
6. Специальные возможности.
 - 6.1 Генерация входного файла для программного комплекса «Cosmos/m».
 - 6.2 Вывод локальной матрицы жесткости в файл



1. Выбор необходимого масштаба.

Перед вводом геометрии необходимо выбрать правильный масштаб. Зная габариты конструкции, кнопками на контрольной приблизить и отдалить устанавливается необходимый масштаб. Правильность контролируется просмотром координат курсора в нижней информационной полоске. В программе нет определенных единиц измерения, как и в программе Cosmos/m, пользователь сам выбирает в какой системе измерения работать. Она определяется тем, в каких единицах измеряется длина и сила, все производные величины опираются на эти размерности. Пользователь должен четко соблюдать выбранную систему измерений, в ней он вводит исходные данные и в ней он получает ответ.

2. Ввод исходных данных

2.1 Ввод геометрии конструкции.

Ввод полигональной фигуры осуществляется мышью после нажатия кнопки Новый полигон. Каждое нажатие левой кнопкой мыши в рабочем окне будет добавлять одну точку, вершину, и соединять их отрезками. При пересечении отрезка, соединяющего последнюю введенную точку и вновь вводимой с любым, уже существующим, пользователь получит сообщение об ошибке и ему будет предложено ввести точку заново. При щелчке мышью вблизи первой точки вводимого полигона, он замыкается и считается введенным. Можно вводить вложенные контуры, имеет смысл контур со степенью вложенности не больше одного, т.е. может быть только контур, вложенный во внешний и не может быть другого контура, вложенного в него. Другими словами, любая точка не может находиться внутри более чем двух контуров одновременно.

Не важно в какой последовательности вводить контуры – система сама определит какой из них внешний. После завершения ввода полигона можно приступить к точной установке его вершин.

Точное позиционирование осуществляется из диалогового окна Properties, вызываемого кнопкой на панели или пунктом меню Edit/Geometry. На закладке Geometry расположены поля для ввода точных координат вершин. Для задания координат вершине необходимо выбрать нужную точку щелчком мыши (при этом выбранная точка станет красной). В пунктах для ввода координат появятся точные координаты вершины, их можно изменить и применить изменения кнопкой “Apply” на диалоговом окне. В этих пунктах должны вводиться числовые значения, в случае неверного ввода в эти пункты, равно как и в других пунктах всех остальных вкладок пользователь получит сообщение об ошибке с указанием в каком пункте введено неправильное значение.

Щелчком мыши на грани контура можно выбрать эту грань – она станет красной. В диалоговом окне Properties в пунктах координат появятся координаты одного из концов грани. Рядом будет находиться уменьшенное схематическое изображение выбранной грани, концы которой обозначаются квадратами. Закрашенный обозначает вершину, координаты которой будут меняться, сбоку написано началом или концом грани является эта точка. Другой конец грани можно выбрать щелкнув мышью по не закрашенному квадрату.

Это не полное выделение вершины – в других закладках (Constraints, Loads, Physics) считается, что выбрана грань, а не вершина. Выбрать вершину можно копкой “Select begin” или “Select endpoint”, при этом в

главном окне грань становится черной, а выбранная вершина – красной. Это уже полное выделение вершины.

В конструкции могут быть и не связанные точки, т.е. не находящиеся на грани. К ним может прилагаться нагрузка и они могут закрепляться. Считается, что такая точка принадлежит внешнему контуру, если она находится внутри него и не находится внутри никакого другого контура. Если эта точка находится внутри другого контура то считается, что она принадлежит этому контуру.

2.2 Задание нагрузок.

Для задания нагрузок предназначена закладка “Loads” диалогового окна “Properties”. В ней расположены три поля для задания трех нагрузок: сила вдоль оси Z, момент, вращающий относительно оси X и момент, вращающий относительно оси Y. За положительное направление оси Z принято направление от пользователя вглубь монитора. Нагрузка, задаваемая в этих полях прикладывается к выбранному в данный момент объекту, вершине или грани. При нажатии кнопки “Apply”, если была задана нагрузка отличная от нуля, в главном окне появится обозначение нагрузки вблизи выбранного объекта и нагрузка будет считаться приложенной.

Для точки любая из приложенных нагрузок будет сосредоточенной, а для грани распределенной. После разбиения распределенная нагрузка сводится к узлам по формуле:

$$p = q*d/2,$$

где p–нагрузка передаваемая на узел, q–распределенная нагрузка на грань, d–длина сегмента, на которые разбивается грань. Это действие производится со всеми сегментами, с обеими его точками, причем p прибавляется к нагрузке узла, а не присваивается.

С помощью этого инструмента можно просматривать значения нагрузок, т.к. на главном окне отображается только само наличие нагрузки, отличной от нуля. Для этого нужно просто выбрать интересующий объект и значения нагрузок на него отобразятся в полях ввода.

Для задания распределенной по площади нагрузки на контур используется закладка “Physics” (см. ниже).

2.3 Наложение связей.

Для этого существует вкладка “Constrains”. На ней расположены пункты

“Constrain Lin. Z”– наложить линейную связь вдоль оси Z, “Constrain Rot. X”– наложить поворотную связь вдоль оси X и “Constrain Rot. Y”–

наложить поворотную связь вдоль оси Y. Связи могут накладываться на вершины и на грани контуров, для этого необходимо выбрать один из этих объектов. В закладке “Constrains” соответствующие пункты станут выбранными в соответствии со связями, наложенными на данный объект. Кроме того, в главном окне эти связи отображаются в виде условных обозначений. После этого можно изменить условия закрепления с помощью изменения состояния соответствующих пунктов. Изменения вносятся после нажатия кнопки “Apply”.

2.4 Физические характеристики.

Определение физических характеристик конструкции осуществляется с помощью вкладки “Physics” диалогового окна “Properties”. На ней находятся поля задания давления (распределенной по площади нагрузки), модуля упругости, коэффициента поперечной деформации (коэффициента Пуассона), толщины области конструкции и прозрачности этой области. Область должна быть очерчена полигональным контуром. Для выбора всего контура необходимо выбрать один из его элементов с нажатой клавишей SHIFT, при этом весь контур станет красным. В соответствующих пунктах вкладки “Physics” появятся значения соответствующие текущим значениям. По умолчанию эти значения соответствуют конструкционной строительной стали ($E = 210000$, $\mu = 0.25$) [Е. Беленя 1986], давление нулевое, а толщина равна 10. **ВНИМАНИЕ!** по умолчанию эти характеристики даются в системе измерений Ньютоны/миллиметры, если пользователь выбрал другую, то он должен изменить модуль упругости (коэффициент Пуассона – безразмерная величина) в соответствии со своей системой измерений. В этой вкладке осуществляется контроль введенных значений. Модуль упругости и толщина могут принимать только положительные значения, а коэффициент Пуассона может принимать значения между 0 и 0.5. При попытке неверного ввода пользователь получит сообщение об ошибке и значение необходимо будет исправить.

Пункт “Hole” необходим для установки прозрачности областей конструкции, т.е. определения, является ли данная область отверстием или областью с другими характеристиками, давлением или толщиной. При установке галочки на этом пункте считается, что область является отверстием, это проигнорируется в том случае, если отверстием будет сделан внешний контур. При определении области как отверстия она не разбивается при генерации сетки и все остальные характеристики, введенные во вкладке “Physics” не учитываются.

3. Генерация сетки.

Алгоритм генерации сетки построен на основе алгоритма Рапперта [Jim Ruppert, «A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation», *NASA Ames Research Center, Submission to Journal of Algorithms, 1994*].

Встроенный алгоритм генерации сетки является адаптивным, т.е. он автоматически учитывает особенности конструкций, сгущаясь вблизи сужений конструкции или около мелких ее особенностей. Особенностью данного алгоритма является то, что математически доказано, что он может сгенерировать сетку с минимальным углом не превышающем только 20 градусов, т.е. любой угол в сетке будет больше 20 градусов. На практике же он обычно способен строить сетки с минимальным углом до 33 градусов, но из-за этого может проявиться неустойчивость, которая будет проявляться в виде неоправданных сгущений сетки.

В оригинале вся регулировка алгоритма заключается в задании минимального угла сетки. В программе этот алгоритм несколько видоизменен и добавлен еще один управляющий параметр–критерий максимальной площади элемента. Все настройки алгоритма осуществляются в диалоговом окне, вызываемом из меню Mesh/Parameters and Stats. В нем есть две секции, настройки и данных по уже сгенерированной сетке. Из настроек предлагается изменить параметры минимальный угол и максимальная площадь. Также можно убрать галочку с пункта анимация, отключив тем самым пошаговую прорисовку элементов, это весьма заметно ускоряет работу алгоритма. Сама генерация сетки запускается кнопкой “Сгенерировать сетку” на панели или пунктом меню Mesh/Generate Mesh.

Во время генерации сетки появляется информационное окно, показывающее текущее значение минимального угла, количество элементов и узлов. Работу алгоритма можно прервать нажав на клавиатуре клавишу ESC, после чего, можно начать генерацию заново.

Обычно такая операция необходима при оперативном изменении входных параметров при зацикливании или неудовлетворительных результатах.

4. Решение задачи.

4.1 Решение в перемещениях.

При решении производится перенумерация узлов, генерируются локальные матрицы жесткости, на их основе собирается глобальная матрица жесткости, производится учет граничных условий, условий нагружения и собственно решение системы уравнений. Все происходит без участия пользователя, ему лишь предоставляется информационное окно с указанием выполняемой операции и процентом его выполнения.

4.2 Решение в напряжениях.

После получения решения в перемещениях можно получить решение в напряжениях, что обычно и является целью расчета. Особенность этого расчета в том что из-за выбора типа треугольного конечного элемента невозможно найти напряжения на его границах, а можно лишь в центре конечного элемента. Поэтому решение в напряжениях дается в центрах элементов, а не в узлах как для решения в перемещениях. Напряжения даются для дальней от пользователя стороны.

5. Просмотр результатов

5.1 Просмотр результатов в табличной форме.

Возможен после выполнения расчета в перемещениях вызовом пункта меню Results/View Results. В нем расположена таблица и элементы управления. Шапка таблицы представляет собой заголовки с графическим элементом вблизи него в виде треугольника, символизирующего столбец, по которому ведется сортировка и ее порядок. Для того чтобы изменить столбец, по которому ведется сортировка достаточно щелкнуть мышью на заголовке столбца в шапке таблицы. Повторный щелчок изменит порядок сортировки.

На диалоговом окне есть элементы и другие элементы. Пункт ввода максимальных отклонений определяет размер выборки, по умолчанию это значение равно 100%. При значениях меньше 100% выборка будет ограничиваться значениями отклонения сортируемого параметра от максимального по модулю значения. Для переключения режима “по узлам” и “по элементам” существуют соответствующие переключатели: “by nodes” “by elements”. В режиме “by nodes” отображаются только номера узлов и все их перемещения, это режим по умолчанию. В режиме “by elements” отображаются номера элементов, их линейные перемещения и все напряжения.

5.2 Просмотр результатов в виде изолиний.

После расчета на перемещения становятся доступными другие виды, относящиеся к перемещениям, в переключателе видов на главной инструментальной панели, в главном окне. Для перемещений доступны три режима: “UZ Lin. Displacements isolines”, “RX Rot. Displacements isolines” и “RY Rot. Displacements isolines”, отвечающие соответствующим перемещениям. Для напряжений также доступны три режима, отвечающие напряжениям σ_X , σ_Y , τ_{XY} . В режимах для напряжений производится интерполяция значений из центров элементов в их узлы, поэтому может наблюдаться небольшое искажение картины.

В любом из режимов, кроме “Normal View” в информационной строке внизу главного окна отображается значение вблизи курсора мыши, соответствующее выбранному режиму. Это значение меняется от перемещения мыши, но не от нажатия клавиш на ней.

6. Специальные возможности

6.1 Генерация входного файла для программного комплекса «Cosmos/m».

В программе есть возможность генерировать входной файл для программного комплекса «Cosmos/m». Это может быть полезно для сравнения результатов расчета в исследовательской работе. На тестовых примерах, для которых существует аналитическое решение, программа «FEPLAR» показывает погрешность в два - три раза меньше чем «Cosmos/m». Например, для плиты, заземленной по контуру и нагруженной распределенной нагрузкой, при количестве элементов 900 и узлов 500 (порядок системы уравнений без учета граничных условий 1500) погрешности составляют соответственно 1% и 3%.

В файл выводится информация об узлах, объединении их в элементы, закреплении узлов, и нагрузки на них (закрепление и нагрузка передается уже сведенной в узлы).

Функция вызывается пунктом меню Solve/Cosmos input file, этот пункт становится доступным только после решения задачи в перемещениях.

Формат файла:

файл включает в себя следующий набор команд.

N - команда космоса NODE - узел, индекс узла, координаты X, Y, Z соответственно. В программе Cosmos/m применяется трехмерная система координат и в данном случае координата Z приравнивается к

нулю. Нумерация узлов вычислена алгоритмом перенумерации Катхилла-Макки.

```
N, 1, 500, 100, 0  
N, 2, 500, 125.15, 0  
N, 3, 474.9, 100, 0  
N, 4, 485.727, 114.295, 0  
.....  
N, 481, 100, 249.8, 0  
N, 482, 100, 262.35, 0  
N, 483, 100, 274.85, 0  
N, 484, 100, 287.4, 0  
N, 485, 112.575, 300, 0  
N, 486, 100, 300, 0
```

E - команда ELEMENT - элемент, индекс элемента (порядковый номер), три узла, его формирующих. В файле не содержится информации о типе конечного элемента, его физических характеристиках, поэтому до загрузки входного файла необходимо выполнить минимальный набор команд: EG - определение типа конечного элемента, RC - определение характеристик элемента (толщина) и команд, определяющих физические параметры материала.

```
E, 1, 4, 1, 3  
E, 2, 298, 296, 269  
E, 3, 429, 444, 449  
E, 4, 27, 26, 17  
.....  
E, 894, 383, 348, 382  
E, 895, 411, 382, 383  
E, 896, 348, 382, 379  
E, 897, 345, 379, 348
```

D - команда Displacement - перемещение, номер узла, направление перемещения, величина перемещения. При расчете плоских плит используются только перемещения UZ - линейные перемещения вдоль оси Z, ROTX - поворот относительно оси X, ROTY - поворот относительно оси Y.

D, 444, UZ, 0
D, 423, UZ, 0
D, 444, ROTX, 0
D, 423, ROTX, 0
D, 444, ROTY, 0
D, 423, ROTY, 0
.....
D, 397, ROTX, 0
D, 396, ROTY, 0
D, 397, ROTY, 0

F - команда FORCE - сила, номер узла, направление, величина силы.
При расчете плоских плит используются только усилия FZ - сила, действующая вдоль оси Z, MX - момент, вращающий относительно оси X, MY - относительно оси Y. В данном случае это распределенная по площади нагрузка, сведенная в узлы.

F, 4, FZ, 312.167
F, 6, FZ, 339
F, 8, FZ, 210.167
F, 9, FZ, 284.833
.....
F, 469, FZ, 284.167
F, 470, FZ, 306.5
F, 473, FZ, 233.5
F, 474, FZ, 325.833
F, 475, FZ, 204.667
F, 476, FZ, 320.5
F, 477, FZ, 187.333
F, 478, FZ, 268

6.2 Вывод локальной матрицы жесткости в файл.

Эта функция полезна в исследовательской работе при разработке программ, использующий треугольный конечный элемент (в свое время я столкнулся с проблемой отсутствия правильно посчитанной локальной матрицы жесткости). В программе FEPLAR применяется конечный элемент с десятью степенями свободы: по одной линейной и по две поворотных связи на каждый узел и одна линейная для центра элемента. Впоследствии эта последняя степень свободы исключается, т.к. она

является локальной, принадлежащей только данному элементу и перемещение в ней напрямую не влияет на состояние других элементов. В итоге локальная матрица жесткости имеет размерность 9x9 и не зависит от положения начала координат. Порядок перемещений для каждого узла: UZ, RotX, RotY.

Пример матрицы жесткости.

```
574391  3.22218e+06  -2.76598e+06  -145215  2.0788e+06  -616607  -
429176  1.23976e+06  -3.04191e+06

3.22218e+06  3.76319e+07  -3.90141e+06  -2.50629e+06  1.76303e+07
6.00248e+06  -715889  5.51119e+06  -6.72128e+06

-2.76598e+06  -3.90141e+06  3.02498e+07  -728498  -752917  8.7205e+06
3.49448e+06  -6.51759e+06  1.84852e+07

-145215  -2.50629e+06  -728498  301700  -2.15384e+06  -1.25035e+06  -
156485  -765665  -1.31857e+06

2.0788e+06  1.76303e+07  -752917  -2.15384e+06  2.52848e+07
3.59926e+06  75046.4  3.93606e+06  4.14486e+06

-616607  6.00248e+06  8.7205e+06  -1.25035e+06  3.59926e+06
1.41146e+07  1.86695e+06  3.39506e+06  1.01796e+07

-429176  -715889  3.49448e+06  -156485  75046.4  1.86695e+06  585661
-474092  4.36047e+06

1.23976e+06  5.51119e+06  -6.51759e+06  -765665  3.93606e+06
3.39506e+06  -474092  1.11079e+07  -2.34464e+06

-3.04191e+06  -6.72128e+06  1.84852e+07  -1.31857e+06  4.14486e+06
1.01796e+07  4.36047e+06  -2.34464e+06  4.41318e+07
```

Список литературы

[Ruppert 1994] **Jim Ruppert**,

«A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation»,
NASA Ames Research Center, Submission to Journal of Algorithms, 1994.

[Джордж 1984] **А. Джордж, Дж. Лю**,

Численное решение больших разреженных систем уравнений. — М.: Мир, 1984

[А. Александров] **Александров А.В., Лашенников Б.Я., Шапошников Н.Н.**,

Строительная механика. Тонкостенные пространственные системы: Учебник
для вузов; под. ред. А.Ф. Смирнова. — М.: Стройиздат, 1983

[СНиП Стальные конструкции]

СНиП II-23-81*. Стальные конструкции/ Госстрой СССР. - М.: ЦИТП Госстроя
СССР, 1988. - 96 с.

[Е. Беленя 1986] **Е.И.Беленя, В.А.Балдин, Г.С.Ведеников и др**

Металлические конструкции. Общий курс: Учебник для вузов; Под общ. ред.
Е.И.Беленя. 6-е изд., перераб. и доп. - М., Стройиздат, 1986. - 560 с., ил.

Ссылки на первоисточники

[George 1971] **Alan George**,

«Computer implementation of the finite element method», *Tech. Rept. STAN-CS-208*, Stanford University (1971)

[Gibbs 1976] **N.E. Gibbs, W.G. Poole Jr, and P.K. Stockmeyer**

«An algorithm for reducing the bandwidth and profile of sparse matrix», *SIAM J. Numer. Anal.* 13 (1976), pp 236-250.

[Jennings 1966] **A. Jennings**

«A compact scheme for the solution of the symmetric linear simultaneous equations», *Comput. J.* 9 (1966), pp 281–285

[Liu 1975] **Joseph W-H Liu, and Andrew H Sherman**,

«Comparative analysis of the Cuthill–McKee and the reverse Cuthill–McKee ordering algorithms of sparse matrices», *SIAM J. Numer. Anal.* 13 (1975), pp 198-213.