Описание генератора функций

Введение. Текущие возможности и актуальные файлы

Сейчас (07.08.15) генератор умеет генерировать функции в таких случаях:

- одномерный случай с соединениями, ни в одном уравнении нет производных порядка выше второго;
- двумерный случай с соединениями, ни в одном уравнении не содержится смешанных производных и производных порядка выше второго;
- двумерный случай, в уравнениях могут содержаться смешанные производные второго порядка, ни в одном уравнении нет производных порядка выше второго — правильно может быть обработана только ситуация, когда нет соединяющихся блоков (т.е. при генерировании функций не приходится приближать смешанные производные в местах соединений).
- трехмерный случай с соединениями, ни в одном уравнении не содержится смешанных производных и производных порядка выше второго.

Для генерирования функций используются файлы, находящиеся в папке domain-model:

1. customOfficer.py;
2. equationParser.py;
3. newFuncGenerator.py;
4. rhsCodeGenerator.py;
5. derivCodeGenerator.py;
6. someFuncs.py;
7. DerivHandler.py.

Файлы 6 и 7 используются для генерации, но функции, лежащие в них, могут использоваться и для других целей. Из файла 6 для генерации используются функции **NewtonBinomCoefficient()**, **generateCodeForMathFunction()**, **determineNameOfBoundary()**, **RectSquare()**, **determineCellIndexOfStartOfConnection2D()**, **getRanges()**. Функции **factorial()** и **getCellCountAlongLine()** из этого же файла используются функциями **NewtonBinomCoefficient()**, **determineCellIndexOfStartOfConnection2D()**, **getRanges()**. Файл 7 содержит функцию, которая находит порядок старшей производной.

1 Описание основных файлов

1.1 Файл customOfficer.py и проверка входных данных

Файл 2 (customOfficer.py) создан для проверки некоторых введенных в *.json данных на корректность перед запуском генерации функций. Этот файл содержит класс **Reviewer**, который связан с внешним миром методом **ReviewInput()**. В методе **createCPPandGetFunctionMaps()** класса **Model** создается экземпляр класса **Reviewer** и вызывается его метод **ReviewInput()**. Этот метод проверит правильность введения блоков и параметров.

Основные методы класса **Reviewer**:

- **ReviewParameters()**. Проверит, нет ли в списке параметров "Params"повторяющихся имен; совпадает ли множество ключей каждого из словарей в "ParamValues"с множеством элементов списка "Params"; правильно ли установлен "DefaultParamsIndex".
- **ReviewBlocks()**. Проверит каждый из блоков по таким критериям:
  - размеры блока ("Size") заданы неотрицательными числами;
  - индекс уравнения по умолчанию и индекс начального условия по умолчанию заданы корректно;
  - корректно заданы границы регионов уравнений, регионов начальных условий, регионов граничных условий (при этом нет контроля за наложением одного региона на другой, это остается на совести пользователя); этим будет заниматься метод **ReviewEqRegOrInitRegOrBoundReg()**;
  - одинаково количество уравнений в каждой системе, участвующей в задаче (система участвует в задаче, если она задана в списке "Equations"и при этом есть блок или часть блока, на котором задана именно эта система) (метод **ReviewEquations()**);
  - количество компонент для каждого начального условия совпадает с количеством уравнений в каждой системе (метод **ReviewInitials()**). То же самое и для граничных условий (метод **ReviewBounds()**).
- **ReviewInput()**. Вызывает методы **ReviewParameters()** и **ReviewBlocks()**.

Недостатки проверяльщика:

- Не определено до конца, какие данные нужно проверять на корректность, а какие нет. Поэтому некоторые проверки могут быть лишними, некоторые — отсутствовать.
- Корректность ввода интерконнектов пока что нигде не проверяется!

1.2 Файл equationParser.py и парсинг уравнений, начальных и граничных условий

Мозги парсера — библиотека pyparsing. Этот файл (equationParser.py) содержит три класса:

- **CorrectnessController**. Занимается проверкой уравнений и математических функций (например, используемых в качестве начальных или граничных условий) на корректность ввода (например, правильность расстановки скобок, правильность расстановки операторов $+$, $-$, $*$, $/$ и т.д.)

- **ParsePatternCreater**. Чтобы распарсить некое выражение средствами pyparsing, нужно составить подходящую под это выражение грамматику. Нам надо парсить уравнения (выделять их правые части) и отдельно функции. Для этого надо создать 2 разных (но очень похожих) грамматики. Еще надо получать из левых частей уравнений список компонент искомой функции. Надо поэтому уравнения парсить уже по-другому (выделять их левые части), т.е. есть необходимость создания третьей грамматики. Этот класс отвечает за создание этих грамматик.

- **MathExpressionParser**. С помощью созданной предыдущим классом грамматики либо парсит уравнение, либо математическую функцию, либо возвращает список компонент искомой функции.

Клиент использует только класс 1.2. Это происходит в методе **generateAllPointInitials()** класса **abstractGenerator** для того, чтобы парсить начальные условия; в методе **generateCentralFunctionCode()** того же класса для парсинга уравнений, необходимых для генерации центральных функций; в реализациях матода **generateBoundsAndIcs()** классов **generator1D** и **generator2D** (а в будущем и **generator3D**).

Недостатки парсера:

- Работает долго.

- Проверка на корректность ввода уравнений и математических функций (расстановка скобок и т.д.) сделана без использования средств pyparsing. Это порождает целый дополнительный класс, который занимается проверкой.

1.3 Файл newFuncGenerator.py, выполняющий основную работу

Класс **abstractGenerator** и его наследники **generator1D** и **generator2D** выполняют почти всю работу, связанную с генерацией (оставшуюся, но все же весьма значительную часть работы выполняют классы, находящиеся в файлах 4, 5).

В методе **createCPPandGetFunctionMaps()** класса **Model** создается экземпляр класса **FuncGenerator**, описание которого находится в рассматриваемом файле 3. В конструкторе этого класса в зависимости от размерности задачи выбирается конкретная реализация генератора (т.е. полем этого класса становится экземпляр одного из трех классов **generator1D**, **generator2D**, **generator3D**). В дальнейшем метод **generateAllFunctions()** класса **FuncGenerator** работает именно с этой конкретной реализацией. Также этот метод формирует список словарей **functionMaps**, по которому формируется матрица пересчета и структура которго описана в файле definition в папке doc.

Вот полный список классов файла newFuncGenerator.py с их описанием:

- **FuncGenerator**. Оперируя генератором для нужной размерности, определенная в нем функция **generateAllFunctions()** генерирует:
  - определения всех констант (дефайнов);
  - функции для работы с начальными условиями и параметрами;
  - для каждого блока — набор центральных функций, набор граничных, угловых и соединительных (интерконнектов) функций.

- **abstractGenerator**. Содержит общие для всех трех генераторов поля и методы. Например, в нем определены методы генерирования констант (дефайнов) (метод **generateAllDefinitions()**), функций для начальных условий (**generateAllPointInitials()**), параметров (**generateParamFunction()**); методы генерирования функций для граничных условий Дирихле (**generateDirichlet()**) или Неймана, а также интерконнектов (**generateNeumannOrInterconnect()**) (функция, генерирующая интерконнект, совпадает с функцией, генерирующей Неймана, просто ей передаются параметры с другим смыслом); метод генерирования центральных функций **generateCentralFunctionCode()**.

- **generator1D**, **generator2D**, **generator3D**. Содержат специфические методы и поля, нужные для обработки задачи указанной размерности. Генератор **generator3D** не работает и не изменялся очень очень давно. Важно то, что у этих классов есть общие методы **getBlockInfo()** и **generateBoundsAndIcs()**, которые вызываются в методе **generateAllFunctions()** класса **FuncGenerator**. В первом методе определяются все уравнения, заданные на блоке (т.е. копится информация для генерации центральных функций), и для каждой границы определяются граничные условия и интервалы в клетках, на которых они действуют. А также начинает формироваться словарь из списка **functionMaps**. Во втором методе генерируются граничные, угловые, соединительные функции и завершается формирование словаря из списка **functionMaps**.

- **InterconnectRegion**. Создан для того, чтобы унифицировать обработку границ двумерного (и может, трехмерного) блока, т.к. граничные условия и соединения с другими блоками для массива **functionMaps** надо представлять в одном и том же формате. Просто при создании экземпляра генератора для каждого блока создается список его соединительных регионов.

- **BoundCondition** и **Connection**. Метод **getBlockInfo()** в двумерном (и может, в трехмерном) случае составляет список элементов, которые есть экземпляры этих классов.

1.4 Файл rhsCodeGenerator.py и генерирование правых частей уравнений, заданных пользователем

Файл содержит класс **RHSCodeGenerator**, который отвечает за генерацию правой части одного уарвнения системы, заданной пользователем, с учетом краевых условий или соединений. Пример возврата такой функции — строка:

```
result[idx + 0] = params[0] * ((2.0 * DXM2 * (
        source[idx - Block0StrideX * Block0CELLSIZE + 0] -
        source[idx + 0] + (0.0) * DX)));
```

Этим занимается метод **generateRightHandSideCode()**. Методы **generateDirichlet()** и **generateNeumannOrInterconnect()** класса **abstractGenerator** используют именно этот метод. Отдельная задача — сгенерировать производную с нужным конечно-разностным приближением. Подготовкой к такой генерации занимаются методы **callDerivGenerator()** и **callSpecialDerivGenerator()**. А саму производную генерируют уже методы одного из двух классов, описанных в файле 5.

1.5 Файл derivCodeGenerator.py и генерирование производных

Т.к. в уравнении могут встретиться и чистые и смешанные производные, а смешанные производные в случае соединения блоков порождают много разных видов функций, которые необходимо сгенерировать, то удобно поручить генерирование чистых и смешанных производных разным классам. Эти классы:

- **PureDerivGenerator**;

- **MixDerivGenerator**.

1.5.1 PureDerivGenerator

Для центральной функции способен сгенерировать чистую производную любого порядка (потому что конечно-разностные приближения для них в этом случае содержат однотипные элементы, различно только их количество и коэффициенты перед слагаемыми). В зависимости от того, какими значениями был проинициализирован экземпляр этого класса, за генерацию производной отвечает одна из трех функций:

- **commonPureDerivativeAlternative()** — генерировать производную для центральной функции или для граничной в случае, когда граничное условие не влияет на производную (например, первая производная по y вдоль границы $x = 0$);
- **specialPureDerivativeAlternative()** — генерировать производную для граничной функции, когда граничное условие влияет на производную (например, производная по x вдоль границы $x = x_{max}$);
- **interconnectPureDerivAlternative()** — генерировать производную для функции-соединения (для интерконнекта).

1.5.2 MixDerivGenerator

Пока что способен генерировать смешанные производные только второго порядка для несоединенных блоков. Функции-генераторы:

- **commonMixedDerivativeAlternative()** — генерировать производную для центральной функции;
- **specialMixedDerivativeAlternative()** — генерировать производную для граничной функции.

Производные в углах двумерного блока генерируются по частям, а потом объединяются в обну строку в методе **callDerivGenerator()** класса **RHSCodeGenerator**.

2 Конечно-разностные формулы, используемые при генерировании производных

Расчетные функции (которые и генерирует генератор) отличаются друг от друга только выражениями, аппроксимирующими производные по пространственным переменным. Если рассматриваемая система уравнений n-мерна (т.е. искомая функция — это вектор-функция $u = (u_1, \dots, u_n)$, $u_k = u_k(t, x, y, z)$, $k = \overline{1, n}$) и ни одно из уравнений не содержит смешанных производных и производных порядка выше второго, то i-ое уравнение этой системы ($i = \overline{1, n}$) имеет вид:

$$\frac{\partial u_i}{\partial t} = f\left(u_1, \frac{\partial u_1}{\partial x}, \frac{\partial u_1}{\partial y}, \frac{\partial u_1}{\partial z}, \frac{\partial^2 u_1}{\partial x^2}, \frac{\partial^2 u_1}{\partial y^2}, \frac{\partial^2 u_1}{\partial z^2}, \dots, u_n, \frac{\partial u_n}{\partial x}, \frac{\partial u_n}{\partial y}, \frac{\partial u_n}{\partial z}, \frac{\partial^2 u_n}{\partial x^2}, \frac{\partial^2 u_n}{\partial y^2}, \frac{\partial^2 u_n}{\partial z^2}\right).$$

Для простоты записи можно считать, что система одномерна, т.е. есть всего одно уравнение:

$$\frac{\partial u}{\partial t} = f\left(u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial u}{\partial z}, \frac{\partial^2 u}{\partial x^2}, \frac{\partial^2 u}{\partial y^2}, \frac{\partial^2 u}{\partial z^2}\right). \tag{1}$$

Надо аппроксимировать все производные, стоящие в правой части (1). Нужны аппроксимации для:

1. центральных функций;
2. функций для рассчета границ блока;
3. функций для рассчета вершин блока (в двумерном и трехмерном случаях);
4. функций для рассчета ребер блока (в трехмерном случае).

В 2. - 4. надо учитывать краевые условия (в случае краевых условий Неймана как раз и возникает необходимость аппроксимировать производные, а в случае условий Дирихле вместо правой части уравнения (1) просто подставляется значение производной функции, являющейся этим условием), соединения блоков и то, что на разных частях одного блока могут быть заданы разные уравнения. Для приближения производных во всех функциях используются центральные конечные разности.

Рассмотрим трехмерный блок — параллелепипед

$$\Pi = \{(x, y, z) \in R^3 | x \in [x_0, x_1], y \in [y_0, y_1], z \in [z_0, z_1]\}.$$

Пусть $m, n, p \in N$ и на Π введена сетка с шагами $\Delta x = (x_1 - x_0)/m$, $\Delta y = (y_1 - y_0)/n$, $\Delta z = (z_1 - z_0)/p$ по пространственным переменным x, y, z и введен шаг Δt по времени t. Значения функции u в узлах сетки далее обозначаются обычным образом:

$$u_{ijk}^{\ell} := u(\ell \Delta t, x_0 + i\Delta x, y_0 + j\Delta y, z_0 + k\Delta z),$$

а значения производных в узлах сетки пишутся похожим образом, например:

$$\left(\frac{\partial^2 u}{\partial y^2}\right)_{ijk}^{\ell} := \frac{\partial^2 u}{\partial y^2}(\ell \Delta t, x_0 + i\Delta x, y_0 + j\Delta y, z_0 + k\Delta z).$$

Здесь $\ell = \overline{0, L}$, $i = \overline{0, m}$, $j = \overline{0, n}$, $k = \overline{0, p}$.

Замечание: т.к. генератор все-таки умеет генерировать функции для задачи, когда в уравнении есть смешанные производные и при этом рассматриваются блоки без соединений, то для этой ситуации ниже также описаны формулы, используемые генератором для аппроксимации смешанных производных. Для такой задачи уравнение (1) выглядит по-другому:

$$\frac{\partial u}{\partial t} = f\left(u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial u}{\partial z}, \frac{\partial^2 u}{\partial x \partial y}, \frac{\partial^2 u}{\partial x \partial z}, \frac{\partial^2 u}{\partial z \partial y}, \frac{\partial^2 u}{\partial x^2}, \frac{\partial^2 u}{\partial y^2}, \frac{\partial^2 u}{\partial z^2}\right). \tag{2}$$

2.1 Формулы для производных в центральных функциях

В центральных функциях для приближения производных используется самый обычный вид конечно-разностных формул.

Первая производная (например, по z):

$$\left(\frac{\partial u}{\partial z}\right)_{ijk}^{\ell} \approx \frac{u_{ijk+1}^{\ell} - u_{ijk-1}^{\ell}}{2\Delta z} \tag{3}$$

Вторая чистая производная (например, по yy):

$$\left(\frac{\partial^2 u}{\partial y^2}\right)_{ijk}^{\ell} \approx \frac{u_{ij+1k}^{\ell} - 2u_{ijk}^{\ell} + u_{ij-1k}^{\ell}}{(\Delta y)^2} \tag{4}$$

Для центральных функций смешанные производные генерировать очень легко. Смешанная производная (например, по xz):

$$\left(\frac{\partial^2 u}{\partial x \partial z}\right)_{ijk}^{\ell} \approx \frac{u_{i+1jk+1}^{\ell} - u_{i-1jk+1}^{\ell} - u_{i+1jk-1}^{\ell} + u_{i-1jk-1}^{\ell}}{4\Delta x \Delta z} \tag{5}$$

2.2 Формулы для производных в функциях рассчета границ блока

2.2.1 Левая граница

Рассмотрим, например, границу $y = y_0$ блока. Пусть на каком-то куске

$$\Pi_0 = \{(x, y, z) \in \Pi | \quad y = y_0; \quad x_0 \leqslant a \leqslant x \leqslant b \leqslant x_1; \quad z_0 \leqslant c \leqslant z \leqslant d \leqslant z_1\}$$

этой границы пользователь задал краевое условие Неймана:

$$\frac{\partial u}{\partial y}(t, x, y_0, z) = \varphi(t, x, z).$$

И пусть на Π_0 также задано уравнение (2). Тогда при аппроксимации производных в функциях для рассчета куска Π_0 используются такие формулы.

Первая производная по y:

$$\left(\frac{\partial u}{\partial y}\right)_{i0k}^{\ell} \approx \varphi_{ik}^{\ell} = \varphi(\ell\Delta t, x_0 + i\Delta x, z_0 + k\Delta z), \tag{6}$$

а первые производные по x и z будут выглядеть как в (3) (т.е. будут центральными).

Вторая производная по y:

$$\left(\frac{\partial^2 u}{\partial y^2}\right)_{i0k}^{\ell} \approx \frac{2\left(u_{i1k}^{\ell} - u_{i0k}^{\ell} - \Delta y \varphi_{ik}^{\ell}\right)}{(\Delta y)^2},$$

а вторые производные по x и z будут как в (4) (т.е. будут центральными).

Смешанная производная по xy:

$$\left(\frac{\partial^2 u}{\partial x \partial y}\right)_{i0k}^{\ell} \approx \frac{\varphi_{i+1k}^{\ell} - \varphi_{i-1k}^{\ell}}{2\Delta x}; \tag{7}$$

смешанная производная по zy:

$$\left(\frac{\partial^2 u}{\partial z \partial y}\right)_{i0k}^{\ell} \approx \frac{\varphi_{ik+1}^{\ell} - \varphi_{ik-1}^{\ell}}{2\Delta z}; \tag{8}$$

смешанная производная по xz будет как в (5) (т.е. будет центральной).

Предположим теперь, что область Π_0 является местом соединения рассматриваемого блока с каким-то другим. Это значит, что на Π_0 вообще не может быть задано краевых условий. Здесь генератор бессилен при генерировании смешанных производных, поэтому считаем, что на Π_0 задано уравнение (1). Формулы такие.

Первая производная по y:

$$\left(\frac{\partial u}{\partial y}\right)^{\ell}_{i0k} \approx \frac{u^{\ell}_{i1k} - ic[idx1][idx2]}{2\Delta y},$$

а первые производные по x и z опять будут выглядеть как в (3). Тут ic – массив интерконнектов, передающийся в рассчетную функцию, а $idx1$ и $idx2$ определяются генератором.

Вторая производная по y:

$$\left(\frac{\partial^2 u}{\partial y^2}\right)^{\ell}_{i0k} \approx \frac{u^{\ell}_{i1k} - 2u^{\ell}_{i0k} + ic[idx1][idx2]}{(\Delta y)^2},$$

а вторые производные по x и z опять будут как в (4).

Формулы для границ $x = x_0$, $z = z_0$ строятся по аналогии.

2.2.2 Правая граница

Рассмотрим границу $y = y_1$ блока. Пусть на куске

$$\Pi_1 = \{(x, y, z) \in \Pi| \quad y = y_1; \quad x_0 \leqslant A \leqslant x \leqslant B \leqslant x_1; \quad z_0 \leqslant C \leqslant z \leqslant D \leqslant z_1\}$$

этой границы пользователь задал краевое условие Неймана:

$$\frac{\partial u}{\partial y}(t, x, y_1, z) = \psi(t, x, z).$$

И пусть на Π_1 также задано уравнение (2). Формулы такие.

Для первых производных по x, z формулы будут такие же, как в (3), а по y — как в (6) с заменой функции φ на ψ и среднего индекса 0 на n в левой части. Для смешанной производной по xz формула как в (5), а для производных по xy и zy формулы как в (7), (8) с заменой функции φ на ψ и среднего индекса 0 на n в левой части.

Сильно изменяется вторая производная по y:

$$\left(\frac{\partial^2 u}{\partial y^2}\right)^{\ell}_{ink} \approx \frac{2\left(u^{\ell}_{in-1k} - u^{\ell}_{ink} + \Delta y \psi^{\ell}_{ik}\right)}{(\Delta y)^2},$$

а вторые производные по x и z опять будут как в (4).

Если область Π_1 является местом соединения рассматриваемого блока с каким-то другим, то на Π_0 вообще не может быть задано краевых условий. Считаем, что на Π_1 задано уравнение (1). Формулы такие.

Первая производная по y:

$$\left(\frac{\partial u}{\partial y}\right)^{\ell}_{ink} \approx \frac{ic[idx1][idx2] - u^{\ell}_{in-1k}}{2\Delta y},$$

а первые производные по x и z опять будут выглядеть как в (3).

Вторая производная по y:

$$\left(\frac{\partial^2 u}{\partial y^2}\right)^{\ell}_{ink} \approx \frac{ic[idx1][idx2] - 2u^{\ell}_{ink} + u^{\ell}_{in-1k}}{(\Delta y)^2},$$

а вторые производные по x и z опять будут как в (4).

2.3 Формулы для производных в функциях рассчета вершин блока (в $2D$ и $3D$) и в функциях рассчета ребер (в $3D$)

Здесь для производных формулы те же самые, что и в п. 2.1 и 2.2. Отличие только в том, что для гарничных функций производные только по какому-нибудь одному аргументу будут приближены нецентральными формулами, а по всем остальным — центральными. Здесь же для вершин в двумерном случае производные по обоим аргументам будут приближены нецентральными формулами, а в трехмерном — по всем трем. Для ребер в $3D$ производные по двум аргументам будут приближены нецентральными формулами, а по третьему аргументу — центральными.

Особенность только в аппроксимации смешанных производных. Пусть рассматривается трехмерный блок Π (без соединений!) и в окрестности какого-то угла, например, $x = x_0$, $y = y_0$, $z = z_1$ задано уравнение (2) и заданы такие условия Неймана на соответствующие границы (можно сразу для простоты считать, что они заданы на каждой из границ полностью, а не на отдельных их частях):

$$\frac{\partial u}{\partial x}(t, x_0, y, z) = f(t, y, z), \quad \frac{\partial u}{\partial y}(t, x, y_0, z) = g(t, x, z), \quad \frac{\partial u}{\partial z}(t, x, y, z_1) = h(t, x, y).$$

Тогда производные по xy, xz, yz в рассматриваемой вершине будут аппроксимироваться формулами:

$$\left(\frac{\partial^2 u}{\partial x \partial y}\right)^{\ell}_{00p} \approx \left.\left(\frac{f^{\ell}_{j+1k} - f^{\ell}_{j-1k}}{4\Delta y} + \frac{g^{\ell}_{i+1k} - g^{\ell}_{i-1k}}{4\Delta x}\right)\right|_{i=0,j=0,k=p};$$

$$\left(\frac{\partial^2 u}{\partial x \partial z}\right)^{\ell}_{00p} \approx \left.\left(\frac{f^{\ell}_{jk+1} - f^{\ell}_{jk-1}}{4\Delta z} + \frac{h^{\ell}_{i+1j} - h^{\ell}_{i-1j}}{4\Delta x}\right)\right|_{i=0,j=0,k=p};$$

$$\left(\frac{\partial^2 u}{\partial y \partial z}\right)^{\ell}_{00p} \approx \left.\left(\frac{g^{\ell}_{ik+1} - g^{\ell}_{ik-1}}{4\Delta z} + \frac{h^{\ell}_{ij+1} - h^{\ell}_{ij-1}}{4\Delta y}\right)\right|_{i=0,j=0,k=p}.$$

Проблема тут в том, что приходится считать, что нам известны значения функций f, g и h за пределами блока Π.

Формулы для производных на ребрах чуть-чуть отличаются. Ребро — это место пересечения двух границ. Рассмотрим какое-нибудь ребро, например, $x = x_1$, $z = z_0$. Пусть на соответствующих границах заданы условия Неймана:

$$\frac{\partial u}{\partial x}(t, x_1, y, z) = \xi(t, y, z), \quad \frac{\partial u}{\partial z}(t, x, y, z_0) = \eta(t, x, y).$$

Тогда формулы для аппроксимации производных на рассматриваемом ребре такие:

$$\left(\frac{\partial^2 u}{\partial x \partial y}\right)^{\ell}_{mj0} \approx \left.\left(\frac{\xi^{\ell}_{j+1k} - \xi^{\ell}_{j-1k}}{2\Delta y}\right)\right|_{k=0};$$

$$\left(\frac{\partial^2 u}{\partial x \partial z}\right)^{\ell}_{mj0} \approx \left.\left(\frac{\xi^{\ell}_{jk+1} - f^{\ell}_{jk-1}}{4\Delta z} + \frac{\eta^{\ell}_{i+1j} - \eta^{\ell}_{i-1j}}{4\Delta x}\right)\right|_{i=m,k=0};$$

$$\left(\frac{\partial^2 u}{\partial y \partial z}\right)^{\ell}_{mj0} \approx \left.\left(\frac{\eta^{\ell}_{ij+1} - \eta^{\ell}_{ij-1}}{4\Delta y}\right)\right|_{i=m}.$$