Описание пакета RustedSciThe (v. 0.2.12), свободного для скачивания и использования под лицензией MIT

Раздел IV.

(работа над руководством продолжается, могут быть ошибки и упущения!)

Задачи с граничным условием.

Научные основы BVP солвера, руководства разработчика и пользователя.

1. Научные основы.
   1. Общие замечания.

Решение задач с граничными условиями для целей моделирования горения, химических реакторов и смежных проблем реализовано в ряде пакетов, как коммерческих (CHEMKIN™), так и открытых (CANTERA). Такие задачи характеризуются жесткостью дифференциальных уравнений, большим их количеством (поскольку количество возможных реагентов и продукты может быть очень велико), что привело к необходимости разработки достаточно специфических алгоритмов их решения, каковые и будут обсуждаться далее.

1.2 Модифицированный метод Ньютона с параметром (damped Newton) .Данный алгоритм хорошо изложен в работах [1, 4], а образцы работающего кода соответственно на FORTRAN и С++ можно найти по ссылкам [2, 3]. Каким образом этот алгоритм реализован в пакете RST будет обсуждаться ниже.

1.3. Алгоритм удержания ньютоновского шага в физически-разумных границах.

Как в CHEMKIN™ так и в CANTERA реализован алгоритм удержания ньютоновского шага в физически-разумных границах. Поясним эту мысль. [5, стр. 261] Мы, например, знаем, что температура и массовый расход должны быть положительными, и что массовые доли веществ должны быть между нулем и единицей. Концентрации многих веществ, таких как топлива вдали от пламени, близки к нулю и часто и часто грозят вывести решение за пределы границ. Параметр затухания (коэффициент дампинга) изначально выбирается как можно большим, но таким чтобы не нарушать различные ограничения, накладываемые на переменные решения.

В ньютоновском солвере [3] пакета CANTERA (С++) этот алгоритм реализован в виде функции bound\_step, см. исходный код солвера строка 38. В свободно распространяемом солвере twopnt (fortran) [2], основанном в свою очередь на классическом фортрановском солвере этот алгоритм реализован в виде сабрутины newton\_damping, см. исходный код солвера, строка 3207.

Как можно видеть, при реализации этого алгоритма требуется указать границы физических величин. В пакете RST также реализован алгоритм удержания ньютоновского шага в физически-разумных границах.

1.4 Адаптивные сетки.

Авторы [5, 8] обнаружили, что начало расчет на крупной сетке имеет несколько важных преимуществ. Одно из них заключается в том, что ньютоновские итерации с большей вероятностью сходится на крупной сетке, чем на мелкой. Кроме того, на крупной сетке число переменных невелико и, следовательно, стоимость одной итерации относительно невелика. Поскольку итерация начинается с заданного пользователем приближения, «догадки» о решении, вероятно, потребуется много итераций. В конечном итоге, конечно, чтобы решение было точным, оно должно быть получено на мелкой сетке. Однако *по мере вычисления решения на каждой последующей более мелкой сетке начальные оценки становятся лучше, поскольку они получены из сходящегося решения на предыдущей крупной сетке*. В общем случае решение на одной сетке лежит в области сходимости метода Ньютона на следующей более мелкой сетке. Таким образом несмотря на то, что стоимость одной итерации растет, количество необходимых итераций уменьшается. Адаптивное размещение точек сетки для формирования более тонких сеток осуществляется таким образом, чтобы общее количество точек сетки, необходимое для точного представления решения минимизируется.

В работе [9, стр. 138] предлагается довольно простая техника пересчета сеток. Теперь новые точки сетки вставляются между любой парой соседних точек сетки - скажем, xi и xi+1 для которых | yi - yi+1 | превышает заранее определенный предел число таких точек сетки, вставленных (равномерно) между xi и xi+1, приблизительно равно | yi - yi+1 | /. Затем уравнения решаются снова, вставляются новые точки сетки и так далее; процесс продолжается итеративно до тех пор, пока не окажется | yi - yi+1 |< везде. Значение корректируется в процессе вычислений таким образом, чтобы всегда иметь фиксированное отношение (обычно 10-3) к вычисленному значению размаха (maxi(yi) - mini(yi)). Поскольку вставка новых точек сетки может привести к локальному резкому изменению размера интервала сетки с некоторой последующей потерей точности, с которой сеточные уравнения аппроксимирует уравнения дифференциальные, перед каждой новой гауссовой прогонкой выполняется процесс сглаживания. Этот процесс сглаживания заключается в замене каждой точки сетки xi на новую точку сетки x; через xi = 0.5(xi+1 + xi-1).

Необходимо сделать еще одно уточнение. Часто случается, что, если набор точек сетки не является плотным в нужных местах, решение сеточных уравнений не является хорошим приближением к решению дифференциальных уравнений, тогда процедура вставки точек сетки будет вставлять новые точки сетки в неподходящих областях. Чтобы избежать этой трудности, решение задачи выполняется поэтапно с разными значениями точности *e*. Сначала задача решается (с помощью описанного выше итерационного процесса) для умеренного значения *e* (например, *e* = 10-1 или 10-2), а затем, в свою очередь, для значений e, меньших предыдущего значения *e* примерно в 3 раза. Набор точек сетки, используемый при завершении предыдущего шага, формирует начальный набор для нового шага. Чтобы выявить нарушения в вычислениях, значения maxi(yi) и mini(yi) записываются для каждого значения e, встречающегося в процессе.

Таким образом, идея работы [9] в том, что каждый пересчет сетки происходит сгущение точек: особенно густыми они делаются в областях, где не выполняется условие на разность.

Автор [7, стр. 85], решая задачу о ламинарных пламенах, где вектор неизвестных представляет собой N концентраций веществ и одну температуру, усовершенствует технику работы [9] предлагают вводить новую сетку, добиваясь выполнения условий

(4.1)

Условие на разность

(4.2)

Условие на производную.

где и - малые числа, меньшие единицы, а значения максимумов и получены из сошедшегося численного решения на предыдущей сетке.

Потенциальным недостатком такой процедуры является формирование сетки, которая может не быть плавно изменяющейся. Например, соотношение последовательных интервалов сетки может отличаться на несколько порядков. Это может повлиять как на сходимость, так и на точность метода. В результате мы накладываем дополнительное ограничение на то, что сетка, полученная при использовании ограничений в уравнениях (4.1) и (4.2), должна быть локально ограниченной, то есть отношение соседних интервалов сетки должно быть ограничено сверху и снизу константами.

Мы требуем, чтобы

l/C =< hj/hj-1 =< C, j = 2, 3 ,... , M, (4.3)

где C - постоянная >=l. Такая буферизация сетки сглаживает быстрые изменения в размерах интервалов сетки. В алгоритме адаптивной сетки работы [7] сначала решается уравнения пламени на крупной сетке (4-5 подынтервалов). Затем получаются максимальные значения и

Далее проверялись неравенства в уравнениях (4.1) и (4.2) для каждой из N + 1 компонент S в узлах грубой сетки. Если ни одно из неравенств не выполняется, то точка сетки вставляется в середину рассматриваемого интервала.

После получения новой сетки мы проверяем, является ли она локально ограниченной. Если нет, то точка сетки вставляется в середину интервалов, в которых неравенство из уравнения (4.3) не выполняется. Ранее сходившееся численное решение интерполируется на эту новую сетку, и полученный результат служит оценкой начального решения для метода Ньютона на этой линейной сетке.

Работа [6, стр. 1777] продолжает и развивает идеи работы [9]. Новые точки сетки включаются в более тонкие сетки таким образом, чтобы минимизировать общее количество точек сетки, необходимое для точного представления решения. В частности, для разрешения градиентов мы ограничиваем изменение решения между точками сетки

(4.4)

и для устранения кривизны мы ограничиваем вариацию в производных решения между точками сетки точками.

(4.5)

Новая точка сетки помещается в середину в каждом интервале, где эти неравенства не удовлетворены. Параметры и имеют порядок порядка одной десятой, и мы часто используем = 0,2 и = 0,5. Алгоритм исключает из рассмотрения очень маленькие переменные и те чья суммарная вариация очень мала.

Авторы приходят к выводу, что метод, по-видимому, не чувствителен к используемой схеме интерполяции. Когда сетки очень грубые, решения не являются гладкими и могут иметь мало сходства как друг с другом, так и с физическим решением. Таким образом, использование схем интерполяции высокого порядка не представляется оправданным. Когда сетки становятся достаточно тонкими, и вычисленные решения начинают походить на физические пламена, то решения меняются от одной сетки к другой главным образом в изменении расположения и толщины фронта пламени. Таким образом, комплиментарная интерполяция просто помещает на более мелкой сетке высокоточную копию неправильного профиля решения. Как только решение устоялось метод Ньютона сходится настолько быстро на более тонких сетках, что нам не нужны более точные начальные оценки.

Разработчики CHEMKIN [5, стр. 264] следовали в основном работе [6], в их программе используются соотношения 4.4, 4.5 и при перестройке исключаются переменные, которые находятся ниже определенного минимального значения. Это позволяет избежать ненужного пересчета для переменных, которые, по сути, равны нулю, но из-за ошибок округления могут показывать локально высокие производные. Начальное приближение вектора неизвестных определяется линейной интерполяцией решения на грубой сетке на новую более тонкую сетку. После определения сходящегося решения на новой мелкой сетке процедура перестройки выполняется еще раз.

В пакете RST реализовано несколько алгоритмов перестройки сетки.

1.5 Решение системы линейных уравнений.

Блочно-диагональная система, получаемая при линеаризации системы разностных уравнений как правило [5 стр. 269, 6] решается с использованием алгоритмов, разработанных для библиотеки LINPACK, см. описание [10], код [11]. Код TWOPNT [2] содержит реализацию этих алгоритмов на современном фортране. Код на С++ является частью пакета CANTERA (<https://github.com/Cantera/cantera/blob/main/src/numerics/BandMatrix.cpp>).

Надо отметить, что временная сложность общих методов решения систем линейных уравнений есть O(N3), а временная сложность решения систем с матрицей диагонального вида (banded matrix) O(N\*l2), где l – “толщина” диагонали. Таким образом, последние дают существенный выигрыш в производительности когда N>>l, поскольку l – величина порядка количества неизвестных, а N = количество неизвестных\*количество шагов сетки, это условие выполняется. Таким образом, переход к специализированным для блочно-диагональных матриц методам даст выигрыш в производительности порядка N3/N\*l2 = количество шагов сетки2 раз, то есть от порядка сотен до порядка десяти тысяч раз.

Применяемым методом решения является метод Гаусса с частичным выбором главного элемента для блочно-диагональных матриц (gaussian eliminations with partial pivoting for banded matrix).

Наиболее эффективной и устойчивой является следующая алгоритмическая реализация.

Будем итерировать по столбцам i от 0 до n:

- В столбце i находим наибольший по модулю элемент, находящийся ниже строки i, т.е. ниже главной диагонали, иными словами, наибольший поддиагональный элемент, однако, наша матрица имеет блочно-диагональный вид, и число ненулевых элементов ниже главной диагонали не превышает kl, следовательно достаточно искать наибольший элемент в строках от i по i+kl+1. Поскольку число i+kl+1 очевидно не может превышать размерность матрицы N, то выражение для нижней границы поиска надо записать как

low = min(N, i+kl+1).

Найденный наибольший элемент называется главным или ведущим (pivot element) A[piv, i], а строка с номером *piv*, содержащая главный элемент, называется *главной строкой*;

- если ведущий элемент не является элементом на главной диагонали, иными словами если i!=piv и A[piv, i]! = A[i, i] меняем строки i и piv местами, если i=piv, значит A[piv, i] уже на главной диагонали и ничего переставлять не надо;

- запоминаем примененную в прошлом пункте перестановку в векторе перестановок, он далее будет использован для применения тех же самых перестановок к вектору правых частей b;

Далее следует алгоритм метода Гаусса.

- В варианте метода для полных матриц мы выделяли бы подматрицу As = A[i.. i.. ], начинающуюся с диагонального элемента, однако в варианте для диагональной матрицы мы можем выбрать такую подматрицу, чтобы в ней не было ни одного нулевого столбца или строки: As = A[i..low i.. ], где right = min(N, i + ku + kl + 1). Действительно, kl+ku+1 это ширина диагональной полосы;

- вырезаем из подматрицы As первый столбец C (напомним, что верхний элемент у него будет лежать на главной диагонали), который соответствует столбцу i исходной матрицы As, оставшуюся подматрицу назовем As1

- первый столбец C за исключением первого элемента делим на значение первого элемента;

- с оставшейся матрицей As1 проделываем следующее: вырезаем из нее первую строку R (pivot row), оставшуюся матрицу назовем As2

- итерируем по столбцам As2 для каждого столбца k выполняем:

В конце описанной процедуры получаем LU матрицу, т.е. такую матрицу, которая является суммой L и U матриц за вычетом единиц в главной диагонали матрицы U, иными словами, LU = L + U – E, где E единичная матрица. Эта матрица затем используется для решения линейной системы

Литература

1. The Twopnt Program for Boundary Value Problems Version 3.10 of March 1992. Sandia National Laboratories Report SAND91-8230, Livermore, California, April 1992. Reprinted February 1996.

2. <https://github.com/perazz/twopnt?tab=readme-ov-file>

3. MultiNewton.cpp (https://github.com/Cantera/cantera/blob/main/src/oneD/MultiNewton.cpp)

4.  Ascher, U. M., Mattheij, R. M., & Russell, R. D. 1988. Numerical solution of boundary value problems for ordinary differential equations. Society for Industrial and Applied Mathematics.

5. Chemkin Theory Manual 17.0. Chemkin® Software. January 2016  
6. Joseph F. Grcar, Robert R.  Kee, Mitchell D. Smooke and James A. Miller. A hybrid newton/time-integration procedure for the solution of steady, laminar, one-dimensional, premixed flames. // Twenty-first Symposium (International) on Combustion/The Combustion Institute, 1986, pp. 1773-1782

7. Mitchell D. Smooke. Solution of Burner-Stabilized Premixed Laminar Flames by Boundary Value Methods// Journal of Computational Physics Volume 48, Issue 1, October 1982, pp. 72-105

8. M. D. Smooke and R. M. M. Mattheij, “On the solution of nonlinear two point boundary value problems on successively reﬁned grids,” Applied Numerical Mathematics, volume 1,

1985, pages 463–487.

9. Carl E. Pearson. On a differential equation of boundary layer type //Journal of Mathematics and Physics, volume 47, issue 1-4, pp. 134-154

10. J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, “LINPACK Users' Guide”, Society of Industrial and Applied Mathematics, 1979.

11. <https://www.netlib.org/linpack/>

1. Руководство разработчика.

Солвер задач с граничными условиями реализует следующие возможности (features):

* Точный аналитический Якобиан;
* Адаптивные сетки (несколько алгоритмов пересчета сеток);
* Модифицированный метод Ньютона с параметром ( Damped Newton);
* Выбор между стандартным методом Гаусса решения СЛАУ и вариантом метода для блочно-диагонального якобиана;
* Выбор между плотными и разреженными матрицами;
* Логгирование всего процесса счета;

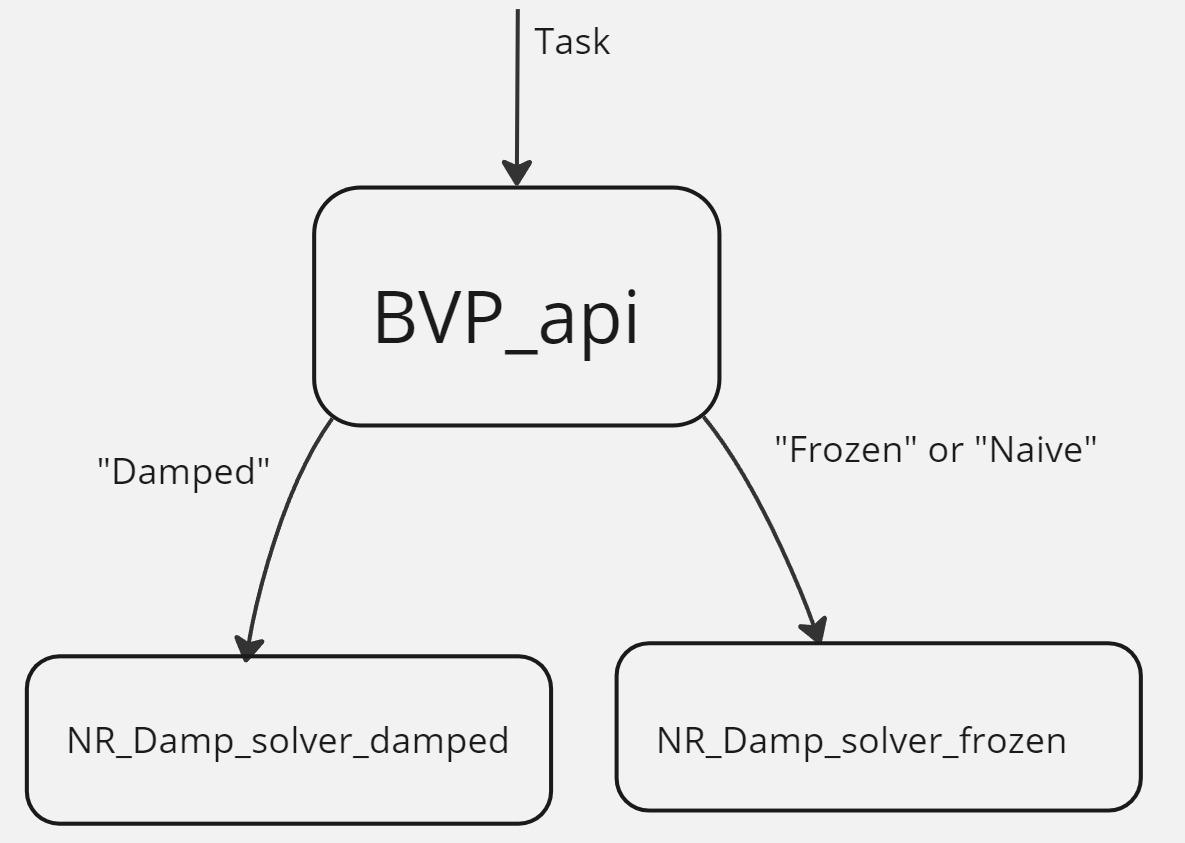
Кратко опишем логику работы основных алгоритмов, устройство модулей, основные структуры данных.

Структура, содержащая задание на расчет называется BVP и находится а модуле numerical::BVP\_api. Эта структура является посредником между пользователем и различными вариантами метода Ньютона и у нее самой нет никакого функционала, кроме передачи условий задачи в модули реализующие конкретные стратегии метода Ньютона. Для инициализации решения необходимо вызвать экземпляр этой структуры и с помощью функции *new* передать в структуру условия задачи, описанные в табл. 1

Табл.1

|  |  |
| --- | --- |
| Публичное поле (заполняется при заполнении задания): тип данных поля | Описание. |
| eq\_system: Vec<Expr> | Вектор правых частей системы уравнений. Задана в символьной (аналитической) форме (NB!). |
| initial\_guess: DMatrix<f64> | Начальное приближение к решению  (матрица размерности, равна количество неизвестных переменных\*количество шагов сетки) |
| values: Vec<String> | Вектор неизвестных |
| arg: String | Название аргумента (x, t и т.д.) |
| BorderConditions: HashMap<String, (usize, f64) | Хэш-таблица (словарь) начальных условий, в котором ключ – название неизвестной, для которой вводится условие, значение – пара чисел: первое число 0 или 1 – соответственно начальное или граничное условие, второе число - собственно значение |
| t0: f64 | Начальное значение аргумента |
| t\_end: f64 | Конечное значение аргумента |
| n\_steps: usize | Количество шагов дискретизации (должно быть согласовано с размерностью матрицы нач. приближения см. выше). В случае если выбран алгоритм с адаптивной сеткой указанное количество шагов дискретизации будет в ходе решения изменено. |
| scheme: String | Параметр для задания численной схемы:  «forward» - простая прямая разность,  «trapezoid» - см. например https://en.wikipedia.org/wiki/Trapezoidal\_rule\_(differential\_equations) |
| strategy: String | Стратегия метода Ньютона:  «Naive» Якобиан пересчитывается каждую итерацию, дампинга нет, удержания ньютоновского шага в заданных границах нет, адаптивных сеток нет – простейший вариант метода;  “Frozen” – Якобиан пересчитывается не каждую итерацию, а по некоторому условию, задаваемому параметрами в хэш-таблице strategy\_params, однако, дампинга нет, удержания ньютоновского шага в заданных границах нет, адаптивных сеток нет –более продвинутый вариант метода;  «Damped» - самый продвинутый вариант метода: дампинг, удержание ньютоновского шага в заданных границах, адаптивные сетки, пересчет Якобиана по условию. Требуемые управляющие параметры должны быть указаны в хэш-таблице strategy\_params |
| strategy\_params: Option<HashMap<String, Option<Vec<f64>>>>, | Параметры стратегии решения. Специфичны для каждой стратегии и будут описаны отдельно. |
| linear\_sys\_method: Option<String> | Параметр для указания солвера решения системы линейных уравнений. |
| method: String | Параметр для указания типа конкретной программной реализации базовых структур линейно алгебры (фактически библиотеки (крейта) линейной алгебры, фактически определяет способ хранения данных матриц)  «Dense» - применяются плотные матрицы;  “Sparse” – разреженные (подходят для больших и очень больших задач); |
| max\_iterations: usize | Максимально допустимое количество ньютоновских итераций, если за это количество не достигается решения следует выход из программы. |
| tolerance: f64 | Точность расчета абсолютная |
| rel\_tolerance: Option<HashMap<String, f64>>, | Относительная точность. Хэш-таблица, где ключ — это название переменной, а число значение точности для этой переменной. |
| Bounds: Option<HashMap<String, (f64, f64)>> | Допустимые физические границы физической величины (например, очевидно, что мольная доля заключена между 0 и 1, а абсолютная температура в физике горения между значением «в комнате» и (условно) 5000 К). Хэш-таблица, где ключ – название неизвестной переменной, а значение – пара чисел (верхняя и нижняя границы). Данная информация используется в алгоритме удержания ньютоновского шага в заданных границах. |
| loglevel: Option<String> | Выбор уровня ведения (и сохранения в файл) лога:   * None - уровень ведения info - все сообщения лога выводятся на печать (тоже самое если loglevel==Some(“info”.to\_string())); * Some(“warn”.to\_string()) - выводятся только предупреждения и ошибки; * Some(“error”.to\_string()) - выводятся только ошибки; |

В программе существуют две «ветки» с частично непересекающимся набором модулей, одна для варианта метода «Damped» и одна для двух остальных: “Frozen” и “Naive”. Конечно, существование двух вариаций кода противоречит известному принципу программирования DRY (don’t repeat yourself), однако, такая организация кода была выбрана умышленно иначе при попытке объединения всех трех вариантов, итак непростой алгоритм усложнился бы еще сильнее.



Ниже будет обсуждаться ветка «Damped», как самая совершенная.

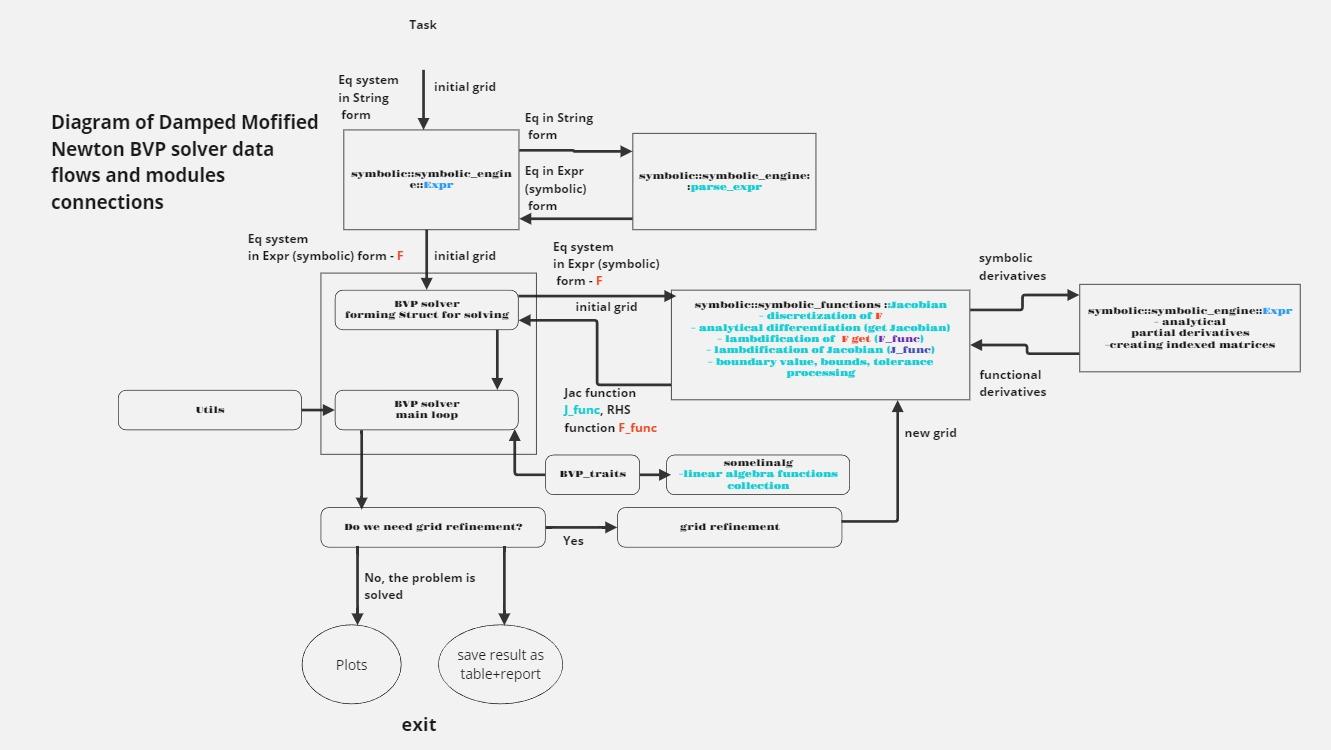
Если мы выбрали стратегию «Damped», то все эти условия задачи тут же будут переписаны в структуру NRBVP (через ее собственный метод new), расположенную в модуле *numerical::BVP\_Damp::NR\_Damp\_solver\_damped*, которая и реализует наиболее совершенный из доступных в пакете RST вариант метода Ньютона. Эта структура содержит помимо задания данные итераций, функции Якобиана и системы уравнений (residual) и пр. данные для расчета.

Охарактеризуем пары ключ-значение хэш-таблицы *strategy\_params*, которую требуется задать пользователю в числе остальных обязательных полей.

|  |  |  |
| --- | --- | --- |
| Название ключа | Значение по умолчанию (задействовано если пользователь указал вместо значения None) | Математический смысл |
| max\_jac | 3 | Количество итераций без пересчета Якобиана, т.е. со старым Якобианом. |
| maxDampIter | 5 | Предельное количество итераций поиска дампинг-фактора |
| DampFacor | 0.5 | Фактор уменьшения предполагаемого дамп-фактора, если предыдущая итерация поиска не была успешна |
| adaptive | Если пользователь указал None перерасчета сеток не будет.  Либо в качестве ключа можно задать Vec[m, n], где m и n - f64 числа:   * m - 1.0 (зарезервировано) * n - количество допустимых перерасчетов сетки. | Требуется или нет перерасчет сеток. |
| Нижеследующие пары характеризуют алгоритм пересчета сеток и имеют смысл, если значение для adaptive не None. | | |
| grcar\_smooke | Значений по умолчанию нет | Алгоритм авторов GrCar и Smooke |
| pearson | Значений по умолчанию нет | Алгоритм автора Pearson |
| two\_point | Значений по умолчанию нет. Требуется задать вектор: Vec[d, g, C]  d, g – соответственно константы из выражений 4.4, 4.5  C – константа из неравенств (4.3) | Алгоритм вдохновлен источником [2] |

На рис. 1 нарисована общая схема потока данных при решении метода Ньютона при помощи указанной стратегии.

Рис. 1



Опишем поток данных, т.е. преобразование информации от задания на расчет до вывода результатов.

Перед запуском решения функция структуры *NRBVP*, а именно *task\_check* проверит правильность заполнения всех полей. Если, что-либо не указано, или указано неверно, программа выдаст диагностическое сообщение и вылетит.

Далее функция *eq\_generate* создаст экземпляр структуры *Jacobian* обратившись к модулю *symbolic::symbolic\_functions* с помощью функции структуры под названием *new*. Название структуры *Jacobian* не должно вводить в заблуждение – ее методы отвечают и за дискретизацию системы, и за формирование вектор-функций и матриц-функций. Дальнейшее развитие событий будет происходить внутри движка символьных вычислений, реализованного в модулях *symbolic::symbolic\_functions* и *symbolic::symbolic\_engine* (назад в солвер *NR\_Damp\_solver\_damped* мы вернемся когда сгенерируем все необходимые структуры данных) поэтому вплоть до возвращения в солвер, называя функцию мы будем иметь в виду, что она является методом структуры *Jacobian* и находится в модуле *symbolic::symbolic\_functions.*

Дальнейшее развитие событий зависит от значения параметра *method:*

- если выбрано значение “Sparse” - это означает что будут использована библиотека *faer* и характерные для нее структуры данных для матриц, векторов и т.д. (мы будем называть их «примитивы линейной алгебры») Функция, которая вызывается при этом называется *generate\_BVP\_SparseColMat*

*-* если выбрано значение “Dense” — это означает что будут использована библиотека *nalgebra* и характерные для нее примитивы линейной алгебры. Функция, которая вызывается при этом называется

*generate\_BVP*

Обе функции, по сути, делают одно и то же (но с разными, по выбору пользователя примитивами линейной алгебры). Внутри функций запускается череда следующих генераций.

1. Выполняется дискретизация системы уравнений с имплементацией граничных условий. Иными словами у нас есть вектор правых частей дифференциальных уравнений, заданный в виде переменной *eq\_system: Vec<Expr>* вектора символьных выражений, далее зная сетку значений аргумента, принимая во внимание схему дискретизации (параметр *scheme*), а так же граничные условия (*BorderConditions: HashMap<String, (usize, f64)>*) в двойном цикле (по шагам сетки и по уравнениям) генерируются сеточные (дискретные) уравнения. Описанный алгоритм имеет место в функции *discretization\_system\_BVP.* Данная функция, таким образом, возвращает дискретизированную систему (все еще символьную!) уравнений. Кроме того, эта функция возвращает вектор дискретных (сеточных) переменных, количество которых, разумеется, определяется количеством неизвестных и шагов сетки. Кроме того, генерируются:

- Вектор границ допустимых значений для каждой из сеточных переменных (на базе параметра задания *Bounds*);

- Вектор относительных точностей для каждой из дискретных переменных (на базе параметра задания *rel\_tolerance*).

1. Теперь, когда у нас есть вектор сеточных уравнений и сеточных переменных, заданных мы можем вычислить символьный Якобиан, вычисление символьного (аналитического) Якобиана реализовано в функции *calc\_jacobian*, а соответствующие низкоуровневые процедуры (алгоритмическая имплементация правил дифференцирования, частные производные, упрощения символьных выражений и пр.) реализованы в модуле движка аналитических вычислений *symbolic::symbolic\_engine.*

Итак, теперь у нас есть вектор сеточных уравнений и матрица Якоби, заданные в символьном виде. Это отлично!

1. Превратим вектор сеточных уравнений, заданных в символьном виде в вектор-функцию для дальнейших вычислений. Это производится в функциях *vector\_funvector\_IVP\_Col* и *vector\_funvector\_IVP\_DVector* соответственно для опций «Sparse» и «Dense»
2. Превратим матрицу якобиана, заданную в символьном виде в матрицу -функцию для дальнейших вычислений. Это производится в функциях  *jacobian\_generate\_IVP\_SparseColMat* и *jacobian\_generate\_IVP\_DMatrix* соответственно для опций «Sparse» и «Dense».
3. Вычислим ширину блочной диагонали (bandwidth) символьной матрицы и запишем ширину ненулевых диагоналей ниже главной (subdiagonal width) и выше главной в кортеж (*kl, ku*)

Выходим из функции *generate\_BVP.*

Итак, теперь у нас есть все необходимое:

* Вектор-функция сеточных уравнений;
* Матрица-функция Якобиана;
* Вектор сеточных переменных;
* Вектор допустимых значений для каждой из сеточных переменных;
* Вектор относительных точностей для каждой из дискретных переменных;
* Ширина полосы ненулевых элементов

, мы можем вернутся в главный модуль метода Ньютона *numerical::BVP\_Damp::NR\_Damp\_solver\_damped* и записать в главную структуру *NRBVP* сгенерированные функции и вектора.

Они будут расположены в соответствующих полях. См. Табл. 2

Табл. 2. Поля заполняемые по возвращении из модуля *symbolic::symbolic\_functions*

|  |  |
| --- | --- |
| fun: Box<dyn Fun>, | Вектор-функция сеточных уравнений; |
| jac: Option<Box<dyn Jac>> | Матрица-функция Якобиана; |
| bounds\_vec: Vec<(f64, f64)>, | Вектор границ допустимых значений для каждой из сеточных переменных; |
| rel\_tolerance\_vec: Vec<f64>, | Вектор относительных точностей для каждой из дискретных переменных; |
| variable\_string: Vec<String> | Вектор сеточных переменных; |
| bandwidth:(usize, isize) | Ширина полосы ненулевых элементов |

Теперь все готово для запуска главного цикла итераций, который реализован в виде функции *main\_loop\_damped.* Обсудим главный цикл:

1. Записываем начальное приближение к решению *initial\_guess* в поле структуры *NRBVP* под названием *y* – это вектор, который содержит текущую итерацию метода Ньютона для сеточных переменных. Выставляем счетчик «возраста Якобиана» (количество итераций без пересчета якобиана) на ноль *m*=0. Общий счетчик итераций *i* выставляется на ноль. Это нулевой шаг алгоритма. Все готово к вычислениям.
2. Вычисляем булеву функцию *jac\_recalc*, чье возвращаемое значение (очевидно булево) записывается в одноименное поле структуры *NRBVP*. Когда *jac\_recalc==true,* якобиан пора пересчитывать, в обратном случае можно и дальше вести расчет со старым якобианом. Поведение этой функции задается значением, соответствующем ключу *max\_jac* из хэш-таблицы *strategy\_params* (один из пользовательских параметров). Это значение содержит «предельный возраст Якобиана» (количество итераций без пересчета якобиана). По достижении его, функция *jac\_recalc* выдает *true*. Так же флаг пересчета Якобиана будет *true*, когда значение матрица якобиана еще не вычислено – на 0-й итерации
3. Заходим в функцию *recalc\_jacobian*, если *jac\_recalc*==*false* тут же выходим, продолжая расчеты со старым якобианом, в обратном случае вызываем из поля структуры *jac* матрицу-функцию якобиана и подставляем в нее *y* вектор, который содержит текущую итерацию значений сеточных переменных, вычисляем якобиан, записываем матрицу чисел в поле *old\_jac*.
4. Инкрементируем счетчик итераций i и счетчик итераций без пересчета Якобиана m.
5. Вызываем функцию *damped\_step*.

Делаем не модифицированный (то есть не умноженный на коэффициент дампинга) ньютоновский шаг, вызывая функцию *step*. Вычисляем шаг dyund[k-1] зная вектор значений y[k-1].

* 1. Функция *step* вычисляет простой (не модифицированный) ньютоновский шаг. Вызываем вектор-функцию *fun*, и подставляем в нее *y* вектор, который содержит текущую итерацию значений сеточных переменных, вычисляем, получаем вектор невязок. Как мы помним, поле *old\_jac* уже содержит вычисленную матрицу Якобиана. Теперь вызываем функцию *solve\_sys* для решения линейной системы, которая принимает Якобиан, вектор правых частей, а также пользовательский параметр *linear\_sys\_method*, а возвращает вектор Ньютоновского шага.

Вычисляем следующую итерацию *y[k] = y[k-1]- dyund [k-1]*.

Вычисляем немодифицированный шаг *dyund[k]*.

4.2 Вычисляем *fbound* множитель, на который умножается ньютоновский шаг из предыдущего пункта, чтобы результат расчета сеточных переменных оставался в пределах границ, определенных для каждой из сеточных переменных, такие пары (левая и правая границы) содержатся в поле *bounds\_vec*, а вычисление *fbound* производит функция *bound\_step*.

4.3 Входим в цикл с целью нахождения модифицированного шага Ньютона (damped step), а также коэффициента дампинга *lambda* для него, внутри цикла действует счетчик циклов k:

- Начальным приближением к конечному коэффициенту дампинга будет *fbound*:

*lambda = fbound*.

- модифицированный шаг *dyd[k] =lambda\*dyund[k]*.

- квадратичная норма модифицированного шага *Sk =L2(dyd[k])*

- итерация с модифицированным шагом *y[k+1] = y[k]- dyd [k]*

*-* следующий немодифицированный шаг *dyund[k+1]*.

- квадратичная норма немодифицированного шага *Sk+1 =L2(dyund[k+1])*

*-* Принимаем коэффициент дампинга, если *Sk+1< Sk*, а так же если *Sk+1<условие сходимости*. В этом случае прерываем цикл нахождения коэффициента дампинга, в обратном случае продолжаем.

*-* уменьшаем текущее значение *lambda* в 2(k+DampFactor) раз.

- инкрементируем счетчик модифицированных шагов *k*, если этот счетчик в ходе циклов достигнет значения, соответствующем ключу *maxDampIter* из хэш-таблицы *strategy\_params* (один из пользовательских параметров) – произойдет выход из цикла со статусом -2.

- идем на следующий цикл по k.

Выходим из функции *damped\_step* возвращаем в функцию *main\_loop\_damped* результат расчета (None или вектор, статус решения). Статусы могут быть такими:

0 – сходимости еще не достигнуто, однако, вычисление модифицированного шага Ньютона прошло успешно, возвращаем новое приближение к решению и статус решения;

1 - сходимость достигнута, вычисление модифицированного шага Ньютона прошло успешно, возвращаем решение и статус решения;

-2 – коэффициент дампинга не найден, возвращаем None и статус решения.

5. В зависимости от статуса возвращаемого *damped\_step* в *main\_loop\_damped* происходит следующее:

Статус 1:

* Если не предполагается использование адаптивных сеток, т.е. значение, соответствующее ключу *adaptive* из хэш-таблицы *strategy\_params* – это None, то это означает, что решение найдено. Последний вектор значений сеточных переменных будет записан в поле *result*. Поздравляем!
* Если предполагается использование адаптивных сеток и флаг *new\_grid\_enabled==false,* это значит, что дополнительные условия сходимости на сетке выполнены и достигнуто полное решение задачи. Последний вектор значений сеточных переменных будет записан в поле *result*. Поздравляем!
* Если предполагается использование адаптивных сеток и флаг *new\_grid\_enabled==true* это значит, что хоть метод Ньютона и сошелся, но дополнительные условия сходимости на сетке не выполнены. Запускается алгоритм пересчета сетки и начального приближения – функция solve*\_with\_new\_grid*, который будет обсужден отдельно далее.

Статус 0:

Возвращенный *damped\_step* вектор значений сеточных переменных будет записан в поле *y* для использования в следующей итерации.

Статус <0:

Требуем перерасчета Якобиана: *jac\_recalc*==true;

Конец итерации, если раньше не вышли на шаге 5, то повторяем алгоритм с пункта. 1. Продолжаем итерации либо до решения, либо до исчерпания *max\_iter* – предельного количества итераций.

Подведем итоги. Итак, в ходе выполнения цикла *main\_loop\_damped* мы используем следующие поля главной структуры:

|  |  |
| --- | --- |
| y: Box<dyn VectorType>, | вектор значений сеточных переменных |
| m: usize, | Счетчик итераций без пересчета Якобиана |
| old\_jac: Option<Box<dyn MatrixType>>, | Численное значение Якобиана |
| Jac\_recalc:bool | Флаг пересчета Якобиана |

Обсудим построение новой сетки.

Хотя каждый алгоритм имеет свои особенности, нужно отметить общие черты:

- новая сетка производится из старой путем добавления точек в некоторых участках сетки;

- новая сетка естественно требует генерирования новой системы с большим количеством уравнений;

- первое приближение для новой системы (с более густой сеткой) нужно приготовить из решения старой системы уравнений, однако поскольку в новой системе будет больше точек дискретизации, чем есть в старой, то требуется выполнить интерполяцию как раз в тех ячейках, где были добавлены новые точки.

Посмотрим теперь, как сказанное реализовано в функции solve*\_with\_new\_grid,* которая запускается в пункте 5.

Запуск ее влечет следующую последовательность выполнения:

А) запускаем функцию *create\_new\_grid*, которая проверяет есть ли в хэш-таблицt *strategy\_params*, ключ, соответствующий одному из алгоритмов пересчета сетки, вектор значений же при этом ключе содержит параметры алгоритма для пересчета. Возможных ключей два: “*pearson*” и “*grcar\_smooke*”, соответствующие им алгоритмы будут рассмотрены ниже. Далее задействуется промежуточный между непосредственно алгоритмами и данной функцией интерфейс *grid\_api* и производятся вычисления.

Функция create\_new\_grid возвращает следующие величины:

- *new\_mesh* (новая сетка),

- *initial\_guess* (приготовленное из решения старой системы начальное приближение для новой системы) – записывается в поле *initial\_guess*,

- *number\_of\_nonzero\_keys* (количество участков, где не выполнены были дополнительные условия сходимости на сетке) записывается в поле главной структуры *number\_of\_refined\_intervals*.

Б) Вызывается функция генерирования уравнений *eq\_generate* (см. выше) в качестве аргумента которой передается новая сетка *new\_mesh.*

В) Вызывается функция *we\_need\_refinement*, которая проверяет, что если *number\_of\_refined\_intervals* == 0, то это значит, что дополнительные условия сходимости на сетке выполнены для всех интервалов, и флаг пересчета сеток *new\_grid\_enabled* делается false, таким образом решение только что сгенерированной системы будет последним этапом решения. Перезапуска задачи с новой сеткой не будет.

Г) Снова запускается основной цикл (см. выше) *main\_loop\_damped.*

Конец solve*\_with\_new\_grid.*

Подведем итоги. Итак, в ходе выполнения вычисления новой сетки мы используем следующие поля главной структуры:

|  |  |
| --- | --- |
| *new\_grid\_enabled: bool* | Флаг пересчета сетки |
| *grid\_refinemens: usize* | Счётчик пересчетов сетки |
| *number\_of\_refined\_intervals: usize,* | количество участков, где не были выполнены дополнительные условия сходимости на сетке |

Обсудим теперь модули, предназначенные для решения линейной системы.

Модуль *linear\_sys\_sovers\_depot* работает таким образом:

Если в задании указан method= “Dense” – это означает, что мы хотим использовать аппарат плотных матриц из библиотеки (крейта) nalgebra, поэтому из модуля *linear\_sys\_sovers\_depot* выбирается функция *nalgebra\_sovers\_depot* , которая работает согласно следующей логике:

- параметр «full» отвечает использованию солвера для обыкновенных матриц;

- параметр «band» отвечает использованию солвера для блочно-диагональных матриц;

- если параметр задания linear\_sys\_method – это None, код сравнивает размерность матрицы якобиана с шириной диагонали (band), если это отношение превышает 10 выбирается параметр «band», в обратном случае «full»;

- если параметр задания linear\_sys\_method не None, то это либо «full» либо «band». Солвер выбирается соответственно параметру задания.

После успешного завершения решения пользователю доступны следующие опции:

* Печать результата расчета в виде таблицы.
* Построение графиков с результатами расчета для каждой из неизвестных.

Все время работы программы ведется подробный лог содержащий ход вычислений. Он будет также напечатан.

Таково краткое описание модулей решения BVP для разработчика.

1. Руководство пользователя.

Основная структура, содержащая задание на расчет называется BVP и находится а модуле numerical::BVP\_api. Для инициализации решения необходимо вызвать экземпляр этой структуры и с помощью функции *new* передать в структуру условия задачи, описанные в табл. 1

Табл.1

|  |  |
| --- | --- |
| Публичное поле (заполняется при заполнении задания): тип данных поля | Описание. |
| eq\_system: Vec<Expr> | Вектор правых частей системы уравнений. Задана в символьной (аналитической) форме (NB!). |
| initial\_guess: DMatrix<f64> | Начальное приближение к решению  (матрица размерности, равна количество неизвестных переменных\*количество шагов сетки) |
| values: Vec<String> | Вектор неизвестных |
| arg: String | Название аргумента (x, t и т.д.) |
| BorderConditions: HashMap<String, (usize, f64) | Хэш-таблица (словарь) начальных условий, в котором ключ – название неизвестной, для которой вводится условие, значение – пара чисел: первое число 0 или 1 – соответственно начальное или граничное условие, второе число собственно значение |
| t0: f64 | Начальное значение аргумента |
| t\_end: f64 | Конечное значение аргумента |
| n\_steps: usize | Количество шагов дискретизации (должно быть согласовано с размерностью матрицы нач. приближения см. выше). В случае если выбран алгоритм с адаптивной сеткой указанное количество шагов дискретизации будет в ходе решения изменено. |
| scheme: String | Параметр для задания численной схемы:  «forward» - простая прямая разность,  «trapezoid» - см. например https://en.wikipedia.org/wiki/Trapezoidal\_rule\_(differential\_equations) |
| strategy: String | Стратегия метода Ньютона:  «Naive» Якобиан пересчитывается каждую итерацию, дампинга нет, удержания ньютоновского шага в заданных границах нет, адаптивных сеток нет – простейший вариант метода;  “Frozen” – Якобиан пересчитывается не каждую итерацию, а по некоторому условию, задаваемому параметрами в хэш-таблице strategy\_params, однако, дампинга нет, удержания ньютоновского шага в заданных границах нет, адаптивных сеток нет –более продвинутый вариант метода;  «Damped» - самый продвинутый вариант метода: дампинг, удержание ньютоновского шага в заданных границах, адаптивные сетки, пересчет Якобиана по условию. Требуемые управляющие параметры должны быть указаны в хэш-таблице strategy\_params |
| strategy\_params: Option<HashMap<String, Option<Vec<f64>>>>, | Параметры стратегии решения. Специфичны для каждой стратегии и будут описаны отдельно. |
| linear\_sys\_method: Option<String> | Параметр для указания солвера решения системы линейных уравнений. |
| method: String | Параметр для указания типа конкретной программной реализации базовых структур линейно алгебры (фактически библиотеки (крейта) линейной алгебры, фактически определяет способ хранения данных матриц)  «Dense» - применяются плотные матрицы;  “Sparse” – разреженные (подходят для больших и очень больших задач); |
| max\_iterations: usize | Максимально допустимое количество ньютоновских итераций, если за это количество не достигается решения следует выход из программы. |
| tolerance: f64 | Точность расчета абсолютная |
| rel\_tolerance: Option<HashMap<String, f64>>, | Относительная точность. Хэш-таблица, где ключ — это название переменной, а число значение точности для этой переменной. |
| Bounds: Option<HashMap<String, (f64, f64)>> | Допустимые физические границы физической величины (например, очевидно, что мольная доля заключена между 0 и 1, а абсолютная температура в физике горения между значением «в комнате» и (условно) 5000 К). Хэш-таблица, где ключ – название неизвестной переменной, а значение – пара чисел (верхняя и нижняя границы). Данная информация используется в алгоритме удержания ньютоновского шага в заданных границах. |
| loglevel: Option<String> | Выбор уровня ведения (и сохранения в файл) лога:   * None - уровень ведения info - все сообщения лога выводятся на печать (тоже самое если loglevel==Some(“info”.to\_string())); * Some(“warn”.to\_string()) - выводятся только предупреждения и ошибки;   Some(“error”.to\_string()) - выводятся только ошибки; |

Охарактеризуем пары ключ-значение хэш-таблицы *strategy\_params*, которую требуется задать пользователю в числе остальных обязательных полей.

|  |  |  |
| --- | --- | --- |
| Название ключа | Значение по умолчанию (задействовано если пользователь указал вместо значения None) | Математический смысл |
| max\_jac | 3 | Количество итераций без пересчета Якобиана, т.е. со старым Якобианом. |
| maxDampIter | 5 | Предельное количество итераций поиска дампинг-фактора |
| DampFacor | 0.5 | Фактор уменьшения предполагаемого дамп-фактора, если предыдущая итерация поиска не была успешна |
| adaptive | Если пользователь указал None перерасчета сеток не будет | Требуется или нет перерасчет сеток. |
| Нижеследующие пары характеризуют алгоритм пересчета сеток и имеют смысл, если значение для adaptive не None. | | |
| grcar\_smooke | Значений по умолчанию нет | Алгоритм авторов GrCar и Smooke |
| pearson | Значений по умолчанию нет | Алгоритм автора Pearson |