А. А. Ардентов

Параллельные и последовательные алгоритмы и программы решения систем уравнений в задачах оптимального управления

Научный руководитель: д.ф.-м.н. Ю. Л. Сачков

Аннотация. В статье описывается разработка параллельных и последовательных алгоритмов приближенного решения систем алгебраических уравнений, возникающих при решении задач оптимального управления. На основе алгоритмов созданы программы построения анимаций и вычисления оптимального синтеза в задаче Эйлера об эластиках. В дальнейшем описанные алгоритмы будут использоваться в разработке программ решения нильпотентной субримановой задачи с вектором роста (2,3,5).

1. Задачи оптимального управления и системы уравнений

Задачи оптимального управления относятся к теории экстремальных задач, то есть задач определения максимальных и минимальных значений. Некоторые из таких задач можно свести к решению системы уравнений. Рассмотрим задачу об эластиках, исследованию которой посвящены работы Л. Эйлера [1], М. Борна [2], а также [3]. Постановка задачи приведена далее в п. 2.1.

1.1. Глобально оптимальные эластики Эйлера

В работе [3] описан алгоритм нахождения глобально оптимальной эластики. Задача была сведена к численному решению нескольких систем алгебраических уравнений вида:

$$q(\chi) = q_1, \quad \chi \in N, \quad q_1 \in M, \tag{1}$$

где трехмерное пространство N — прообраз экспоненциального отображения, а $q(\chi)$ — параметризация эластик при фиксированном конечном времени t_1 (длине эластики). Глобально оптимальной является та эластика, соответствующая данным граничным условиям, у которой величина упругой энергии будет глобально минимальной. Поиск глобально оптимальной эластики заключается в переборе конечного числа локально оптимальных эластик. Причем корни, соответствующие локально оптимальным эластикам, ищутся в заданных

ограниченных подобластях трехмерного пространства, прямоугольных параллелепипедах.

1.2. Локально оптимальные эластики Эйлера

Для того чтобы найти произвольную локально оптимальную эластику, необходимо ограничить область поиска таким образом, чтобы в этой подобласти была лишь одна эластика, удовлетворяющая заданным граничным условиям (1). В качестве параметра, задающего локально оптимальную эластику можно взять приближенную величину ее упругой энергии и область C_i, в которой следует искать корни системы уравнений.

1.3. Нильпотентная субриманова задача с вектором роста (2,3,5)

Системы алгебраических уравнений возникают также в нильпотентной субримановой задаче с вектором роста (2,3,5). Эта задача ставится следующим образом:

$$\begin{cases} \dot{x} = u_1(t)X_1(x) + u_2(t)X_2(x), \\ x(0) = x_0, \\ x(T) = x_1, \\ x \in \mathbf{R}^5,\ (u_1, u_2) \in \mathbf{R}^2, \end{cases} \tag{2}$$

где $X_1,\ X_2$ — гладкие векторные поля, в каждой точке имеющие вектор роста $(2, 3, 5)$. Система (2) доставляет локальную аппроксимацию пятимерных систем с двумерным линейным управлением общего вида.

Сведение задачи (2) к решению систем уравнений было получено в [4–6]. Но если в задаче Эйлера возникала система из трех уравнений, то в данной задаче система состоит из 5 уравнений. Таким образом, решение этих задач можно обобщить, сведя его к поиску корней системы алгебраических уравнений численными методами.

1.4. Особенности и проблемы систем уравнений

Для каждой задачи необходимо найти ограниченную область поиска с единственным искомым корнем внутри. Также зачастую возникает потребность в решении нескольких систем. То есть производится поиск нескольких точек, после чего с помощью некоторого критерия отбирается искомое значение корня. Ограниченная область, в

которой производится поиск, может быть не связной, поэтому поиск производится либо параллельно в связных областях, либо в псевдопараллельном режиме. В псевдопараллельном режиме поиск происходит поочередно в каждой связной области, для этого необходимо создать критерий, с помощью которого программа завершает поиск в одной связной области и начинает искать корень в другой.

1.5. Параллельные программы в системе gridMathematica

Для решения уравнений в данной работе использовалась система gridMathematica [7] — параллельная версия системы Mathematica [8]. gridMathematica — это технико-вычислительная параллельная система для решения сложных проблем науки, инженерии, финансовой сферы и делового анализа. Она предоставляет самую большую в мире коллекцию алгоритмов в одной интегрированной системе, оптимизированной для современных многопроцессорных машин, кластеров, грид-систем и суперкомпьютеров.

gridMathematica [7] состоит из набора ядер Mathematica [8]: одно ядро менеджера и пул рабочих ядер. Ядра Mathematica работают как образующий единое целое модуль, координирующийся менеджером и сообщающийся через технологию Mathematica's MathLink technology.

Для пользователей, которые знакомы с системой Mathematica, чтобы освоить систему gridMathematica, необходимо изучить принцип работы пакета Parallel Computing Toolkit [9], с помощью которого происходит связь между основным ядром и остальными узлами системы. Основное ядро обрабатывает входящие и исходящие данные, а также отвечает за составление графиков заданий. Им можно управлять с любого клиентского интерфейса системы Mathematica или используя командные файлы на локальном или удаленном компьютере. Пользователи могут запускать удаленные ядра (remote kernels) из основного ядра с помощью соединений на основе протоколов RSH или SSH. Сразу после запуска удаленного ядра, оно готово принимать команды, поступающие с основного компьютера.

Рассмотрим пример программы, которая демонстрирует некоторые механизмы распараллеливания вычислений. Эта программа на основном ядре создает k задач типа $fun(i, 100)$, после чего помещает их в очередь и рассылает вычислять на узлы (команда $Queue$). Во время вызова функции $Wait$ главное ядро ожидает результатов выполнения вычислений с удаленных узлов. Функция $chkSl$ запускает $nnodes$ удаленных узлов из списка $avMachines$.

```
chkSl = Function[{nnodes, avMachines},
    If[Length[$Slaves] != nnodes,
        $AvailableMachines = {};
        For[i = 1, i <= nnodes,
            AppendTo[$AvailableMachines,RemoteMachine[avMachines[[i]]]];
            ++i;
        ];
        LaunchSlaves[LinkHost -> "192.168.0.253"];
    ];
];

For[n = 1; times = {};, n <= 8,
    times = Append[times, n <> "nodes: " <> ToString[AbsoluteTiming[
        CloseSlaves[];
        chkSl[n, avMachines];
        RemoteEvaluate[
          fun = Function[{a, n}, Factor[Expand[(x^3 - x + a)^n]]];
        ];
k = 840;
        For[i = 0; pd = {};, i < k,
            pd = Append[pd, Queue[fun[i, 100]]];
            ++i;
        ];
        res = Wait[pd];
        ++n;
    ][[1]]] <> "s."];
];
Print[times];
```

Результат выполнения программы:

```
{1nodes: 904.191188s.,
 2nodes: 453.402125s.,
 3nodes: 306.684497s.,
 4nodes: 359.843528s.,
 5nodes: 288.083183s.,
 6nodes: 161.209873s.,
 7nodes: 141.560466s.,
 8nodes: 126.670936s.}
```

Таким образом, в этом примере наблюдается эффективное распараллеливание решения множества независимых задач. Данный алгоритм обладает свойством масштабируемости (возможностью ускорения вычислений пропорционально числу процессоров).

2. Вычисление глобально оптимальных эластик Эйлера

2.1. Постановка задачи

В классическом вариационном исчислении и оптимальном управлении хорошо известна задача о стационарных профилях упругого стержня. Леонард Эйлер, впервые рассмотревший эту задачу в 1744 г., описал все возможные стационарные профили; они называются эйлеровыми эластиками. Известно, что эластики параметризуются эллиптическими функциями. Однако задача оптимального управления оставалась нерешенной; одной из целей данной работы является ее исследование.

Подробнее, задача состоит в следующем. На плоскости даны точки (x_0, y_0) и (x_1, y_1), и в каждой из точек закреплен некоторый вектор $(v_0, v_1$ соответственно). Требуется соединить точки гладкой кривой γ, выходящей из (x_0, y_0) с вектором скорости v_0 и попадающей в (x_1, y_1) с вектором скорости v_1. В такой постановке задача имеет очень много различных решений. Наложим на кривую γ условие: потребуем, чтобы вдоль искомой кривой квадрат кривизны имел наименьший интеграл:

$$\int_\gamma k^2 dt \to \min. \tag{3}$$

Эта задача имеет простой физический смысл: рассмотрим упругий стержень с закрепленными концами и направлениями стержня в концах (держим стержень руками за концы), функционал имеет смысл упругой энергии. Какую форму примет стержень? Возможные формы, которые может принимать упругий стержень, открыл Эйлер, они называются эйлеровыми эластиками. Однако заданным граничным условиям удовлетворяет не единственная эластика. Вопрос в том, как отобрать эластики, доставляющие решение нашей задаче.

2.2. Вычисление одной эластики

В работе [3] предложен алгоритм поиска глобально оптимальной эластики. Напомним основные этапы этого алгоритма:

(1) Выбор двух случайных точек q_b, q_e.
(2) Поиск корня методом хорд на точках q_b, q_e, результат — q_{i_1}.

(3) Поиск корня методом Ньютона на точке q_{i_j}, результат — $q_{i_{j+1}}$. Если $q_{i_{j+1}}$ искомый корень, то поиск завершается, возвращается результат. Если точка $q_{i_{j+1}}$ находится дальше от корня, чем q_{i_j}, значит переходим к пункту (1). Иначе повторяем эту процедуру с точкой $q_{i_{j+1}}$.

Описанный алгоритм поиска оптимальной эластики основывается на методе случайного поиска. Поэтому время работы программы, реализующей этот алгоритм, зависит от начальных точек, которые задаются с помощью функции Random. Чем ближе к искомому корню выбрана пара случайных точек (начальное приближение корня), тем зачастую ближе к нему будет результат поиска методом хорд. Так как для вычисления следующего шага необходим результат предыдущего, то единственным способом распараллеливания является запуск итераций на узлах с разными точками в качестве начального приближения.

Распараллеливание производится по схеме, схожей с приведенной в качестве примера в предыдущем пункте. То есть для n узлов создаются и рассылаются n задач с основного узла. После того как один из узлов завершил вычисление (нашел искомый корень), результат вычисления передается на основное ядро. Алгоритм реализован в программной среде gridMathematica. После тестирования выяснилось, что программа хорошо работает на 2 узлах. На большем количестве узлов эффективность резко снижается (в таблице 1 приведены результаты решения 300 независимых задач, каждые из которых решаются параллельно).

Таблица 1. Серия из 300 тестов

Число узлов: n	Время работы программы: t, с	Ускорение: t_1/t
1	20875	1
2	8624	2.4
3	7792	2.6
4	6596	3.1
5	6428	3.2
6	7167	2.9

2.3. Вычисление нескольких эластик

Для тестирования алгоритма, описанного в предыдущем пункте, создавался набор случайных точек, для которых необходимо найти оптимальные эластики. Так как алгоритм не является эффективным для большого количества узлов, то есть смысл реализовать алгоритм для параллельного поиска серии оптимальных эластик. Пусть имеется N точек, которым соответствуют N задач. Задачи по очереди рассылаются на узлы для вычислений. Как только какой-то узел завершил вычисление, результат пересылается за основное ядро. Если в очереди еще есть задачи, то на простаивающий узел посылается еще задача и так далее, пока задачи не закончатся. В этом случае распараллеливание происходит эффективно при любом количестве узлов (ускорение близко к линейному).

2.4. Создание фильма в параллельном режиме

Описанная в предыдущем пункте программа может быть применена для создания фильма с непрерывным движением второго конца эластики. Иными словами набор состоит из последовательности точек на некоторой кривой $q_1(s) = (x_1(s), y_1(s), \theta_1(s))$. Несмотря на то, что реализованный параллельный алгоритм работает довольно эффективно, поставленная задача немного отличается от изначальной. Известно, что в большинстве случаев, если близки координаты второго конца эластики, то близки и искомые корни решения системы уравнений. Таким образом, можно в качестве начального приближения задавать корень, вычисленный на предыдущем шаге. При этом время поиска сокращается в десятки раз. Но существуют точки, для которых не годится такого рода начальное приближение. Это — точки, в которые приходят несколько оптимальных эластик (точки Максвелла). Подмножеством таких точек являются точки, задаваемые уравнением $P(x, y, \theta) = x \sin \frac{\theta}{2} - y \cos \frac{\theta}{2} = 0$. Общая формула, с помощью которой можно было бы определить, является ли данная точка точкой Максвелла, не известна.

Рассмотрим алгоритм, который учитывает результат, полученный в предыдущем кадре. В последовательной версии, корни вычисляются по порядку, причем корень, полученный в предыдущем шаге, является первым приближением для вычисления текущего корня. Если при таком приближении корень не был найден, то следующим

начальным приближением является случайная точка из области допустимых значений. Заметим, что время счета при этом будет выше среднего и в таких точках необязательно знать предыдущий кадр. На рис. 1 показано время счета фильма из 400 кадров. Красным цветом обозначены точки, для которых $P < 0$, синим - $P > 0$. На стыке цветов, то есть в точках $P = 0$, корень считается дольше обычного в десять раз.

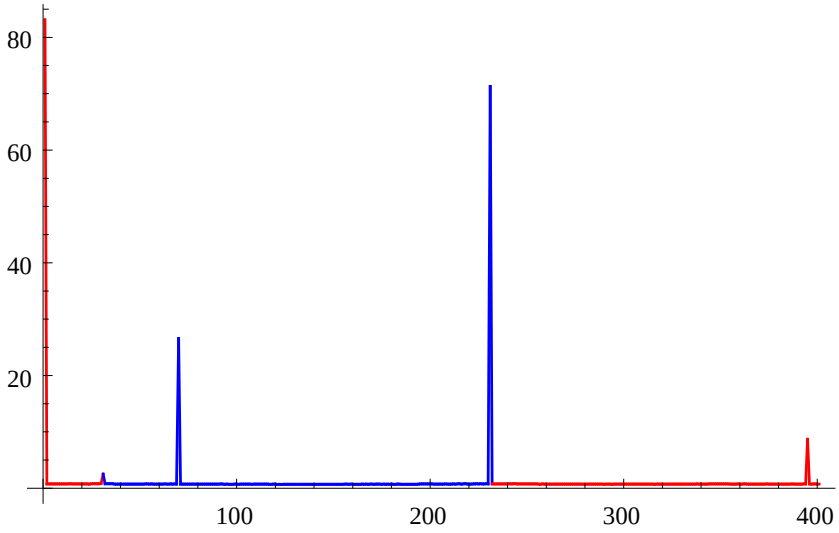

Рис. 1. Время вычисления кадров фильма

В параллельной версии программы на кривой выбирается набор точек, удовлетворяющих условию $P = 0$. Затем эти точки помещаются в очередь, после чего запускается параллельный счет. Причем первые точки, помещенные в очередь, вычисляются без информации о предыдущем кадре, в отличие от остальных. Поэтому после того как какой-то узел обработал точку q_i и выдал результат r_i, то в очередь помещается задача, соответствующая следующей точке q_{i+1} с начальным приближением r_i. И так далее, пока все точки не будут обработаны.

2.5. Результаты тестирования параллельной программы

Для описанных алгоритмов созданы программы, которые были протестирована на кластере «СКИФ Первенец-М». Кривая, на которой выбираются точки, задавалась с помощью функции с большим количеством точек типа $P = 0$. При таком типе входных данных количество отрезков, на которые разрезается кривая, достаточно велико. Следовательно, велико и число гранул параллелизма. Ниже приведена таблица 2, демонстрирующая эффективное распараллеливание алгоритма. В каждом испытании было измерено время работы программы t, с (см. рис. 2); по этим данным были вычислены ускорение времени работы программы $a = t_1/t_i$ (см. рис. 3). Заметим, что в алгориме используется метод случайного поиска, поэтому при повторных вычислениях возможны небольшие отклонения во времени выполнения алгоритма (см. ускорения для 6 и 7 узлов на рис. 3). В этом примере кривая задавалась следующей системой:

$$\begin{cases} x_1(s) = \frac{1}{2}\cos 2\pi s, \\ y_1(s) = \frac{1}{2}\sin 2\pi s, \\ \theta_1(s) = 50\pi s, \\ s \in [0,1]. \end{cases} \tag{4}$$

Анимация глобально оптимальной эластики с граничным условием на кривой (4) содержится в файле [10].

Таблица 2. Серия из 1000 тестов

Число узлов: n	Время работы программы: t, м	Ускорение: t_1/t
1	1374	1
2	693	1.98
3	467	2.94
4	355	3.87
5	297	4.63
6	214	6.42
7	188	7.31
8	173	7.94

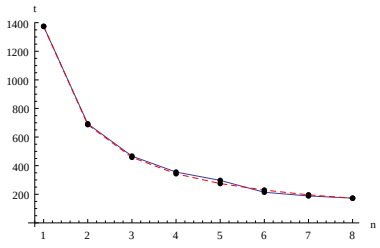

Рис. 2. Время

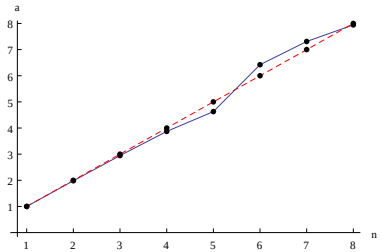

Рис. 3. Ускорение

Но для произвольной кривой количество точек вида $P = 0$ может быть не велико. Также данная кривая может содержать большое количество других точек разреза, для которых не известно аналитическое представление в виде формулы.

Отсюда возникает вероятность того, что на специальных входных данных алгоритм будет выполняться на 8 узлах примерно за такое же время, как на 1 узле. Если кривая не проходит через точки разреза, то параллельный алгоритм нет смысла использовать, так как такого типа фильм собирается за сравнительно небольшое время в последовательном режиме.

3. Вычисление деформации локально оптимальных эластик Эйлера

3.1. Постановка задачи

Вернемся к начальной постановке задачи [3], то есть рассмотрим упругий стержень, который подвергается деформации. Для заданной деформации конечной точки $(x_1(s), y_1(s), \theta_1(s))$ требуется вычислить деформацию всей эластики, оптимальной только локально (и не обязательно глобально). Именно такая деформация наблюдается при изгибе упругого стержня в руках. При непрерывном движении второго конца стержня упругая энергия меняется непрерывно. Следовательно, можно сопоставить такой деформации серию изображений с эластиками. Заметим, что у соседних эластик из этой серии величины упругой энергии будут близки, также как и формулы, которые задают функции, описывающие кривую.

Рис. 4. C_1

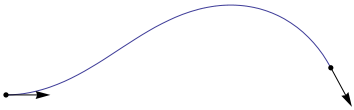

Рис. 5. C_2

3.2. Алгоритм создания фильма

Рассмотрим для начала кривые, которые не проходят через специальные точки, требующие трансформации формул для эластик. Для такого типа входных данных можно применить уже написанную программу. Но помимо кривой, которая задает движение второго конца эластики, необходимо задать начальное положение стержня, так как одним и тем же начальным условиям могут удовлетворять бесконечное количество эластик (локально оптимальных, см. рис. 4, 5).

Но если зафиксировать области поиска L_i и C_j и близкое значение упругой энергии эластики, тогда эластика находится однозначно.

3.3. Область успешной работы алгоритма, примеры

Но область успешной работы данного алгоритма довольно мала, так как множество точек, требующие трансформации формул для эластик, разбивает пространство входных значений на ограниченные подмножества. Фильмы, которые получаются (отдельные кадры см. на рис. 6, и полную анимацию в файлах [11], [12]), демонстрируют лишь небольшую деформацию стержня (например, поворот на угол $\phi < \pi$). Возникает необходимость в обработке этих точек. Для этого необходимо понимать, как переходят друг в друга области L_i, C_j.

Помимо этого неизвестно, каким образом использовать напрямую в решении уравнений величину энергии, которая должна быть близка к искомой. Поэтому удобно передавать величины параметров из предыдущего кадра, которые являются хорошим приближением для искомых. Но при этом возникает сложность перевода параметров одной области в другую, тогда как энергия инвариантна относительно перехода между областями.

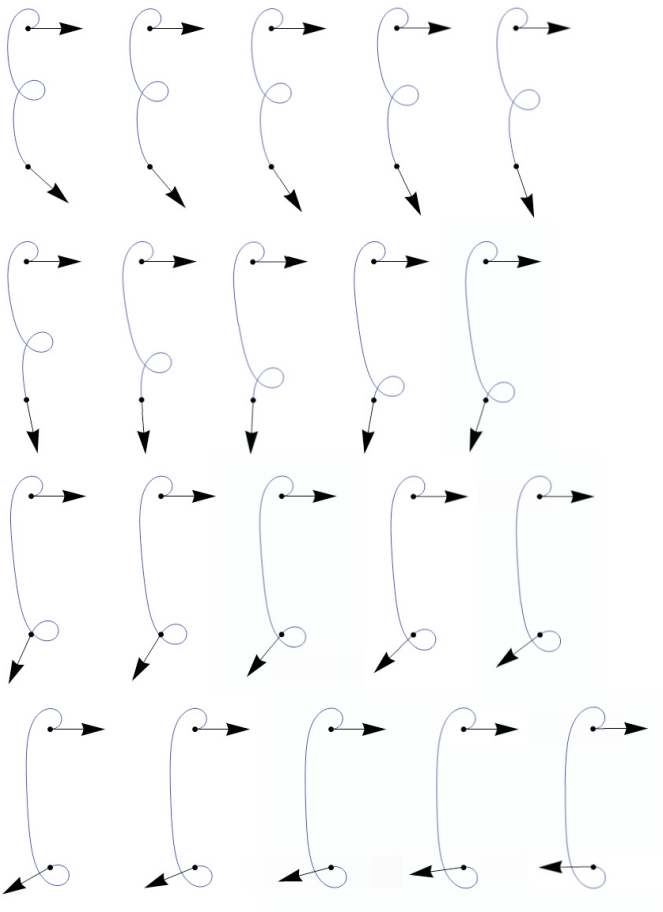

Рис. 6. Кадры фильма с локально оптимальной эластикой

3.4. Классификация проблемных точек

Рассмотрим точки, в которых необходимо изменить формулы, по которым вычисляется эластика. Первый тип таких точек был описан ранее, такие точки задаются уравнением $P = 0$, в них область M_+ переходит в область M_- и наоборот. Для данной области была

создана функция, успешно обрабатывающая соответствующие переходы между областями M_+ и M_- лишь для эластик с небольшой величиной упругой энергии.

Второй тип точек, в которых меняется тип формул, задается уравнением $E = 1$, иными словами это область C_3, граница разделяющая области C_1 и C_2. Если до этого мы находились в области C_j, то следующие точки на кривой будут располагаться в области C_{3-j}. Также как и для первого типа точек изучены переходы для эластик с сравнительно небольшой упругой энергией.

В областях, где эластика заведомо не является глобально оптимальной, проблемные точки не были до конца изучены. Фильмы, описывающие прохождение через эти точки зачастую вычисляются некорректно. Также стоит отметить, что для стержня, у которого расстояние между концами близко к 1, корень уравнения ищется достаточно не устойчиво.

Этот алгоритм не представляется возможным эффективно распараллелить, так как для вычисления кадра на кривой, необходимо знать выходные данные, полученные для предыдущей точки. Зато для успешной обработки проблемных точек необходимо производить довольно большое количество тестирований, которые можно производить параллельно.

4. Выводы

Получены следующие результаты:

- разработаны параллельные и последовательные алгоритмы приближенного решения систем алгебраических уравнений, возникающих при решении задач оптимального управления;
- созданы программы построения анимаций и вычисления оптимального синтеза в задаче Эйлера об эластиках.

В дальнейшем описанные алгоритмы будут использоваться в разработке программ решения нильпотентной субримановой задачи с вектором роста (2,3,5).

Список литературы

[1] Эйлер Л. *Метод нахождения кривых линий, обладающих свойствами максимума или минимума, или решение изопериметрической задачи, взятой в самом широком смысле* // Приложение I, «Об упругих кривых». — Москва-Ленинград: ГТТИ, 1934, с. 447–572. ↑1

[2] Born M. *Stabilität der elastischen Linie in Ebene und Raum* // Preisschrift Und Dissertation. — T. **1**. — Göttingen: Dieterichsche Universitäts-Buchdruckerei, 1906, с. 5–101. ↑1

[3] Ардентов А. А. *Множество разреза в задаче Эйлера об эластиках* // Материалы XII научной студенческой конференции университета города Переславля им. А.К. Айламазяна. — г. Переславль-Залесский: Издательство УГП, 2008. ↑1, 1.1, 2.2, 3.1

[4] Сачков Ю. Л. *Дискретные симметрии в обобщенной задаче Дидоны* // Мат. Сборник. — Т. **197,2**, 2006, с. 95–116. ↑1.3

[5] Сачков Ю. Л. *Множество Максвелла в обобщенной задаче Дидоны* // Мат. Сборник. — Т. **197,4**, 2006, с. 123–150. ↑

[6] Сачков Ю. Л. *Полное описание стратов Максвелла в обобщенной задаче Дидоны* // Мат. Сборник. — Т. **197,6**, 2006, с. 111–160. ↑1.3

[7] http://wolfram.com/products/gridmathematica/. ↑1.5

[8] http://wolfram.com/products/mathematica. ↑1.5

[9] http://documents.wolfram.com/applications/parallel/. ↑1.5

[10] http://www.botik.ru/PSI/CPRC/sachkov/GROUP/rotates.avi. ↑2.5

[11] http://www.botik.ru/PSI/CPRC/sachkov/GROUP/pi.avi. ↑3.3

[12] http://www.botik.ru/PSI/CPRC/sachkov/GROUP/2pi.avi. ↑3.3

A. A. Ardentov. *Sequential and parallel algorithms and programs for approximate solving systems of algebraic equations in control theory problems* // Proceedings of Junior research and development conference of Ailamazyan Pereslavl university. — Pereslavl, 2009. — p. 9–22. (*in Russian*).

Abstract. The article describes development of sequential and parallel algorithms to obtain an approximate solution of systems of algebraic equations. These equations appear in solving control theory problems. Programs which construct animations and compute optimal synthesis for Euler's problem was created on basis of algorithms. The algorithms will be used to develop programs to solve nilpotent sub Riemannian problem with growth vector (2,3,5).