# Решение обыкновенных дифференциальных и дифференциально-алгебраических уравнений в SimInTech

Рассматриваются методы численного решения обыкновенных дифференциальных уравнений (ОДУ) и дифференциально-алгебраических уравнений (ДАУ), реализованные в **SimInTech**. Обсуждаются вопросы выбора наиболее подходящего метода и задания его параметров при решении задач различного типа. Приведены результаты решения ряда тестовых задач в сравнении с наиболее эффективными методами решения ОДУ и ДАУ.

Авторы: Козлов Олег Степанович, Скворцов Леонид Маркович, Ходаковский Виктор Владимирович

# Введение

**SimInTech** предоставляет 10 явных и 6 неявных методов численного интегрирования ОДУ. Такой набор реализованных методов позволяет эффективно решать самые различные задачи, но при этом возникает проблема выбора наиболее подходящего метода и правильного задания его параметров. Очень часто пользователь задает только интервал интегрирования и не обращает внимания на другие опции решателя ОДУ, оставляя их заданными по умолчанию. При решении простых задач с умеренной точностью такой подход вполне допустим, однако при решении сложных задач (например, жестких) неудачный выбор метода либо неправильное задание его параметров может привести к неоправданно большим затратам машинного времени либо к невозможности вообще получить правильное решение.

Таким образом, для профессиональной работы с любым моделирующим программным комплексом пользователь должен обладать некоторыми знаниями о реализованных в нем численных методах. В настоящей статье рассматриваются методы, реализованные в **SimInTech**. Статья может быть полезной для пользователей и разработчиков моделирующих программ.

# Общие положения

Зада́ча Коши́ — одна из основных задач теории дифференциальных уравнений (обыкновенных и с частными производными). Задача состоит в нахождении решения (интеграла) дифференциального уравнения, удовлетворяющего так называемым начальным условиям (начальным данным).

## Постановка задачи

Моделирование процессов в непрерывных динамических системах сводится к численному решению задачи Коши

|  |  |
| --- | --- |
|  | (1.1) |

где **x** ­– *n*-мерный вектор переменных состояния, **f** – *n*-мерная вектор-функция правых частей, *t* – независимая переменная, *T* – величина интервала интегрирования. Численное решение задачи (1.1) получаем в виде последовательности векторов , аппроксимирующих истинное решение на сетке , где – величина *i*-го шага интегрирования.

Эффективность численного решения задачи Коши в значительной степени определяется спектром матрицы Якоби системы ОДУ. Сложность задачи можно оценить величиной ρ*T*, где ρ – спектральный радиус матрицы Якоби. При умеренных значениях ρ*T* (нежесткие задачи) интегрирование обычно выполняется традиционными явными методами [1, 2] и требует небольшого объема вычислений. Трудности возникают при больших значениях ρ*T*, когда для получения качественно правильного решения при использовании традиционных методов необходимо выбирать очень малый шаг интегрирования. В зависимости от расположения больших по модулю собственных значений такие задачи подразделяются на жесткие (большие собственные значения находятся в левой полуплоскости), осциллирующие (вблизи мнимой оси) и локально-неустойчивые (в правой полуплоскости).

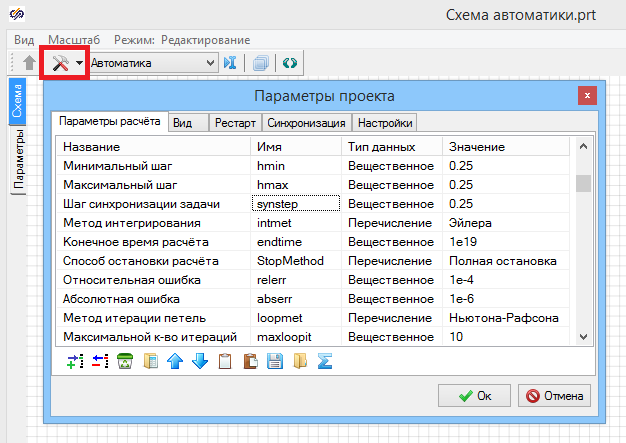
Практика показала, что системы ОДУ, описывающие реальные процессы, скорее всего оказываются жесткими. Достаточно часто в технических приложениях встречаются быстро осциллирующие системы, описывающие высокочастотные колебания, а также локально-неустойчивые системы, в решении которых кратковременные участки с расходящимся процессом чередуются с более продолжительными стабильными участками. Перечисленные типы задач предъявляют совершенно разные требования к методам интегрирования. При интегрировании жестких систем необходимо обеспечить быстрое затухание жестких составляющих, для этого используют неявные A- или L-устойчивые методы [1, 2, 3]. Такие методы подавляют все составляющие решения, соответствующие большим по модулю собственным значениям (если только шаг не выбран очень малым), вследствие чего они плохо приспособлены для решения осциллирующих и локально-неустойчивых систем. При интегрировании осциллирующих систем следует применять специальные методы, обеспечивающие правильный характер огибающей колебательного решения. Специальные методы следует использовать также и для решения жестких локально-неустойчивых систем.

Еще один тип задач, трудных для решения – задачи с разрывами правой части. Для правильного решения таких задач необходимо уменьшать шаг интегрирования в окрестности точки разрыва, что предъявляет повышенные требования к процедуре управления величиной шага.

Таким образом, для эффективного и качественно правильного решения задачи Коши необходимо выявить характер задачи и выбрать наиболее подходящий метод. Современные моделирующие программы, как правило, содержат наборы методов, позволяющих решать задачи разных типов. Однако характер задачи может изменяться в процессе решения или при переходе от одной компоненты вектора переменных к другой. Для решения таких задач могут быть эффективными адаптивные методы [4, 5], расчетные формулы которых настраиваются на решаемую задачу, используя для этого оценки некоторых параметров (например, собственных значений якобиана) системы ОДУ. Особенно перспективны явные адаптивные методы, не требующие при своей реализации вычисления матрицы Якоби и решения алгебраических уравнений. Такие методы есть среди решателей ОДУ **SimInTech**.

## Параметры интегрирования

Выбор метода и установка его параметров в **SimInTech** осуществляется в диалоговом окне «Параметры проекта». Это окно имеет 4 (5 в режиме разработчика) закладок, но только одна из них («Параметры расчёта») имеет отношение к методам интегрирования, а остальные («Вид», «Рестарт» и «Синхронизация»; «Настройка» в режиме разработчика) позволяют устанавливать различные режимы моделирования и отображения схемы проекта, см. рисунок:



**Параметры проекта → Параметры расчёта.**

При открытии окна «Параметры проекта» активной является закладка «Параметры расчёта», где задаются следующие параметры:

**Метод интегрирования** *intmet*. Можно выбрать один из 16 методов (Эйлера, Рунге-Кутты классический, Рунге-Кутты модифицированный, Мерсона классический, Мерсона модифицированный, Адаптивный 1, ..., Адаптивный 5, Адаптивный неявный, Диагонально неявный DIRK2, Гира, Эйлера неявный, DIRK3, DIRK4). По умолчанию установлен метод Эйлера. Метод Эйлера может иметь только фиксированный шаг, а все остальные методы могут использоваться как с фиксированным шагом, так и с автоматическим выбором шага.

**Конечное время расчёта** *endtime*.Длительность интервала модельного времени, на котором выполняется моделирование (по умолчанию время не ограничено и задано равным ­1e19).

**Минимальный шаг** *hmin*. Этот параметр задает ограничение снизу на шаг интегрирования, и по умолчанию устанавливается равным 0.25. Если значение *hmin* оказывается недостаточно малым для расчета с заданной точностью, то выдается сообщение «Заданная точность не обеспечивается». В таких случаях следует уменьшить минимальный шаг (задать его 1e-10, 1e-20, 1e-50 ...) либо снизить требования к точности. Задание очень малого значения *hmin* в большинстве случаев не сказывается заметно на времени счета.

**Максимальный шаг** *hmax*. Ограничение сверху на шаг интегрирования, а для метода Эйлера – величина шага (по умолчанию 0.25). Если задать максимальный шаг равным минимальному, то интегрирование будет выполняться с фиксированным шагом. Задание слишком малого значения *hmax* может привести к неоправданному увеличению времени счета, а задание большого значения *hmax* в некоторых случаях приводит к уменьшению числа точек, выводимых на график. Рекомендуемое значение максимального шага – *endtime/100*.

**Шаг синхронизации задачи** *synstep*. Приблизительное значение шага вывода результатов в графические окна, при этом реальный шаг вывода всегда будет не меньше шага интегрирования. Точное значение шага вывода будет соблюдаться, если установить табуляцию результатов расчета. Рекомендуется задавать *synstep* равным максимальному шагу интегрирования, но не менее *endtime/1000*.

**Относительная ошибка (точность)** *relerr*. Допустимая относительная ошибка интегрирования (по умолчанию 0.001). Не следует задавать это значение больше 0.01 или меньше 1e-10.

Дополнительно устанавливаются следующие параметры:

**Абсолютная ошибка** *abserr*. Допустимая абсолютная ошибка интегрирования (по умолчанию 1e‑10). Оказывает существенное влияние на выбор величины шага только в тех случаях, когда значения некоторых переменных близки к нулю.

**Метод итерации петель** *loopmet***.** Метод решения алгебраических уравнений при наличии в системе алгебраических контуров (Простая итерация – по умолчанию, Ньютона-Рафсона, Бройдена (секущих), Без итераций). Выбранный метод используется для расчета начального состояния алгебраических переменных (независимо от метода интегрирования), а также для расчета алгебраических переменных в процессе интегрирования явным методом. В процессе интегрирования неявным методом дифференциальные и алгебраические переменные решаются совместно, поэтому выбор метода итерирования не имеет значения. Наиболее надежным является метод Ньютона-Рафсона, однако в некоторых случаях и другие методы могут иметь преимущество.

**Максимальной количество итераций.** Максимальное число итераций при решении алгебраических уравнений в одной точке решения (по умолчанию 10). Этот параметр, как и предыдущий, влияет на решение только в тех случаях, когда в системе есть алгебраические контуры.

## Управление шагом интегрирования

Начальный шаг устанавливается равным . В процессе интегрирования с переменным шагом необходимо, кроме решения на очередном шаге, вычислять также оценку ошибки, которая используется для управления величиной шага. Для этого применяют две различные формулы интегрирования, дающие на *m-*м шаге два решения: и . Пусть – вектор переменных, – приращение, – оценка ошибки на последнем шаге. Для управления величиной шага используется нормированная ошибка, вычисляемая по формуле

|  |  |
| --- | --- |
|  | (1.2) |

причем шаг считается удачным, если *err ≤ 1*. В случае удачного шага принимаем значение в качестве нового вектора переменных. В противном случае (при неудачном шаге) шаг отбрасывается, после чего производится пересчет с уменьшенным размером шага. Шаг считается неудачным также и в том случае, когда при вычислении правых частей одна из производных превысила максимально допустимое значение, либо произошло прерывание, вызванное переполнением, делением на ноль, недопустимым значением аргумента и т.п. В таких случаях размер шага уменьшается сразу в 4 раза.

Особенностью **SimInTech** и аналогичных систем моделирования является то, что вычисление правой части системы ОДУ (1.1) осуществляется одновременно с расчетом всей модели. При этом некоторые блоки модели рассчитываются только на заключительной стадии удачного шага. К таким блокам относятся дискретные, ключи, а также некоторые логические блоки. Это сделано с целью исключения внутри шага разрывов правой части, которые могут привести к неоправданному уменьшению шага и возникновению «скользящего режима». Поэтому для получения оценки ошибки на *m-*м шаге нежелательно использовать значение , как это делается, например, в методе Дорманда-Принса [6].

В большинстве методов, реализованных в **SimInTech**, используется стандартная процедура управления величиной шага [3], задаваемая формулой

|  |  |
| --- | --- |
|  | (1.3) |

где *fac = 0.8 … 0.9* – множитель безопасности, *α* – величина, обратная порядку оценки ошибки, *err* – нормированная ошибка (1.2).

# Явные методы

Явные методы представлены в **SimInTech** методом Эйлера, классическими и модифицированными методами Рунге-Кутты и Мерсона, а также пятью адаптивными методами.

## Классические методы

### Метод Эйлера

Является простейшим методом интегрирования ОДУ, он имеет 1-й порядок и задается расчетной формулой

Метод Эйлера можно использовать при интегрировании с постоянным шагом и низкой точностью.

### Классический метод Рунге-Кутты

Задается формулами

|  |  |
| --- | --- |
|  | (2.1) |

Метод имеет 4-й порядок, но применяя экстраполяцию по Ричардсону, можно повысить порядок до 5-го, получив дополнительно оценку ошибки. Окончательные расчетные формулы имеют вид

|  |  |
| --- | --- |
|  | (2.2) |

где – оценка ошибки, а и – численные решения, полученные, соответственно, путем выполнения одного шага размера *h* и двух шагов размера *h/2*. Отметим, что такой метод требует довольно значительных вычислительных затрат, поскольку на каждом шаге функция **f** вычисляется 11 раз.

### Метод Мерсона

Имеет 4-й порядок и задается формулами

|  |  |
| --- | --- |
|  | (2.3) |

По сравнению с классическим методом Рунге-Кутты, в методе Мерсона используется более экономный способ оценивания ошибки. Другое преимущество – в методе Мерсона используется ПИ-управление размером шага, что позволяет обеспечить SC‑устойчивость [3].

При решении умеренно жестких задач явными методами размер шага обычно приближается к границе устойчивости, но в этом случае процедура управления шагом может оказаться неустойчивой. В результате происходят резкие колебания величины шага, приводящие к ухудшению точности и увеличению вычислительных затрат. Устойчивость управления шагом (SC‑устойчивость) может быть обеспечена при использовании процедуры ПИ-управления, задаваемой формулой

|  |  |
| --- | --- |
|  | (2.4) |

где коэффициент *β* выбирается из условия SC-устойчивости (обычно 0 ≤ *β* ≤ 0.1). При *β*=0 получаем стандартную процедуру (1.3). Для метода Мерсона выбраны значения *α*=0.2, *β*=0.08, обеспечивающие SC-устойчивость. Эксперименты показали, что использование ПИ-закона (2.4) позволяет снизить затраты и повысить точность решения умеренно жестких задач по сравнению с использованием стандартной процедуры (2.2).

Классические методы рекомендуется использовать для решения нежестких задач.

Результаты тестирования классических методов приведены в разделах 4.1, 4.4, 4.5.

## Модифицированные классические методы

Классические методы неэффективны при решении жестких задач, однако несложная модификация расчетных схем может значительно расширить область их применения, позволяя эффективно решать как нежесткие, так и умеренно жесткие задачи [4]. В модифицированных методах на основе предварительных стадий вычисляются покомпонентные оценки наибольшего собственного значения матрицы Якоби, которые используются для стабилизации расчетной схемы. В приведенных ниже формулах все действия с векторами выполняются покомпонентно.

### Модифицированный метод Рунге-Кутты

Отличается от классического способом вычисления :

Здесь **z**₁ – вектор покомпонентных оценок наибольшего по модулю собственного значения матрицы *h***J**, где **J** – матрица Якоби. Для нежестких компонент (**z**₁ ≥ -2) используется классическая формула, а для жестких компонент – формула, полученная из условия стабилизации метода в полученной точке жесткого спектра. Отметим, что на самом деле компоненты вектора ***z***₁ не вычисляются (чтобы исключить переполнение или деление на 0), а вычисляются только компоненты вектора , удовлетворяющие условию , т.е. соответствующие жестким компонентам.

### Модифицированный метод Мерсона

Также отличается от классического способом вычисления :

Модифицированные классические методы могут использоваться для решения умеренно жестких задач с повышенной точностью.

## Адаптивные методы

Модифицированные методы можно рассматривать как адаптивные методы Рунге-Кутты. Значительно более эффективными при решении жестких задач оказались адаптивные одношаговые и многошаговые методы [4, 5], основанные на получении оценок наибольших по модулю собственных значений и последующей стабилизации расчетной схемы в полученных точках жесткого спектра. В **SimInTech** реализовано 5 таких методов.

Одношаговые адаптивные методы строятся на основе стадий Рунге-Кутты, которые выполняются по формулам

|  |  |
| --- | --- |
| ; | (2.5) |

где *s* - число стадий, *β* и *α* – параметры метода (в общем случае самонастраиваемые, оптимальная настройка этих параметров рассматривалась в [4, 5]). Далее вычисляются векторы

|  |  |
| --- | --- |
|  | (2.6) |

которые используются в заключительной формуле шага интегрирования, а также для получения оценок собственных значений.

Вектор покомпонентных оценок одного собственного значения находится в виде

|  |  |
| --- | --- |
|  | (2.7) |

На основе полученных оценок вычисляется вектор настраиваемых параметров , который используется в формуле шага интегрирования:

|  |  |
| --- | --- |
|  | (2.8) |

На основе формул (2.5)–(2.8) построены одношаговые методы Адаптивный 1, Адаптивный 2, Адаптивный 3 и Адаптивный 5 (т.е. все «Адаптивные» кроме 4-го). Рассмотрим их подробнее.

### Адаптивный 1

Трехстадийный метод (число стадий *s* = 3), имеет 1-й порядок для жестких и 2-й для нежестких задач. В нем параметр *β* = 1, тогда вектор (2.7) получим в виде

Компоненты вектора настраиваемых параметров вычисляются по формуле

|  |  |
| --- | --- |
|  |  |

а формула (2.8) принимает вид Для оценивания ошибки используется двухшаговая формула

Где – отношение текущего шага к предыдущему, , *i* = 1, …, *n*. Для очень жестких компонент выполняются одна или две дополнительные стадии, расширяющие область устойчивости в окрестности точек жесткого спектра.

### Адаптивный 2

Расчетные формулы такие же, как у метода Адаптивный 1, но дополнительные стадии не используются, а значение параметра *β* настраивается на основе оценки жесткости задачи (изменяется от 2/3 до 1), благодаря чему порядок метода для нежестких задач повышается до 3-го. Ошибка оценивается по правилу Рунге, т.е. с использованием одного шага размером *h* и двух шагов размером *h*/2.

### Адаптивный 3

Четырехстадийный метод (число стадий *s* = 4), а вектор настраиваемых параметров вычисляется по формуле

|  |  |
| --- | --- |
|  |  |

Для оценивания ошибки используется вложенная формула. Порядок метода – от 2-го до 3-го.

### Адаптивный 5

Пятистадийный метод (*s* = 4) 2-го порядка для жестких и 3-го для нежестких задач, шаг которого выполняется по формуле

Векторы настраиваемых коэффициентов вычисляются с использованием покомпонентных оценок двух наибольших собственных значений, которые могут быть как вещественными, так и комплексными. Благодаря этому метод эффективен также и при решении осциллирующих задач.

Одношаговые адаптивные методы имеют невысокую точность, поэтому в **SimInTech** реализован также многошаговый метод. На многих жестких задачах этот метод оказался наиболее эффективным среди всех адаптивных методов. Поскольку он до сих пор нигде не описан, рассмотрим его более подробно.

### Адаптивный 4

Многошаговый адаптивный метод переменного порядка (от 1-го до 5-го для жестких задач и от 2-го до 6-го для нежестких задач),основанный на использовании разделенных разностей, определяемых рекуррентно согласно формулам

(аналогично определяются ).Введем также коэффициенты , зависящие от размеров текущего и предыдущих шагов и задаваемые формулами

|  |  |
| --- | --- |
|  | (2.9) |

В этом случае *k*-шаговые формулы прогноза и коррекции при интегрировании с переменным шагом запишутся в виде

|  |  |
| --- | --- |
|  | (2.10a) |
|  | (2.10б) |

где – разделенная разность с использованием вместо . Формула прогноза имеет порядок *k*, а порядок формулы коррекции на 1 больше.

На основе формул вида (2.10) обычно строят многошаговые методы переменного порядка и шага, при этом принимается , а оценка ошибки получается как разность между прогнозом и скорректированным решением. Для адаптивного метода эти формулы являются предварительными, на их основе можно получить вектор оценок наибольшего собственного значения:

Чтобы предотвратить деление на ноль, вместо **z** вычисляем вектор

|  |  |
| --- | --- |
|  | (2.11) |

причем только для жестких и неустойчивых компонент, удовлетворяющих условию , где *μ* – константа, определяемая областью устойчивости и зависящая от *k*. Формула интегрирования для таких компонент имеет вид

|  |  |
| --- | --- |
|  | (2.12) |

где векторные коэффициенты вычисляются по формулам

|  |  |
| --- | --- |
|  | (2.13) |

(**e** – *n*-мерный вектор с единичными компонентами).

Формула (2.12) построена таким образом, что она имеет порядок *k* и обеспечивает затухание за один шаг при решении уравнения

Формулы (2.9)–(2.13) задают многошаговый адаптивный метод с переменным шагом, при этом для нежестких компонент ограничиваемся формулами (2.10), и дальнейшие вычисления не производим. Оценку ошибки получаем как приращение в формуле коррекции (2.10б) для нежестких компонент или последний член в формуле (2.12) для жестких компонент. Приведенные формулы позволяют также легко изменять порядок метода, который определяется значением *k*, т.е. числом членов в формулах (2.10), (2.12). Если на предыдущем шаге было , то на очередном шаге значение *k* может быть от 1 до . Новое значение *k* определяется на этапе прогноза исходя из условия, чтобы прогноз был наиболее точным. Формулу прогноза (2.10а) можно записать в виде

Новое значение *k* определяется как максимальное число, для которого выполняется условие

где норма ошибки вычисляется в соответствии с (1.2).

# Неявные методы

При решении многих жестких задач явные адаптивные методы показали результаты, сравнимые с неявными методами, однако наиболее универсальным средством решения жестких задач все-таки остаются неявные методы.

## Метод Гира

Метод Гира является многошаговым методом переменного (от 1-го до 6-го) порядка, построенным на основе формул дифференцирования назад (ФДН) [1-3], которые можно записать в виде

|  |  |
| --- | --- |
|  | (3.1) |

где коэффициенты зависят от порядка формулы *k*. В методе Гира эти формулы реализованы с помощью вектора Нордсика [1,2], позволяющего использовать эффективный алгоритм автоматического изменения порядка и величины шага. Последние компоненты вектора Нордсика позволяют получить оценку ошибки.

При *k* = 1 (тогда ) получим **неявный метод Эйлера** – метод 1-го порядка, также реализованный в **SimInTech**.

## Адаптивный неявный метод

Адаптивный неявный метод построен на основе метода трапеций, формула которого имеет вид

Выполняя один шаг величиной *h*, получим , а выполняя два шага величиной h/2, получим . Далее вычисляется вектор покомпонентных оценок наибольшего собственного значения

который используется в заключительной расчетной формуле

Погрешность оценивается по формуле . Порядок метода изменяется от 2‑го до 4‑го, в зависимости от жесткости задачи.

## Диагонально неявные методы Рунге-Кутты

В общем случае метод Рунге-Кутты задается формулами

|  |  |
| --- | --- |
|  | (3.2) |

и может быть представлен в виде таблицы Бутчера

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | **c** | **A** | = |  |  | |  |  |  |  | | (3.3) |

Часто приводят также формулу

которая используется для получения оценки погрешности численного решения . В этом случае в таблицу (3.3) добавляется строка коэффициентов .

В явном методе при *j ≥ i*, тогда формулы (3.2) задают расчетный алгоритм, который может быть непосредственно реализован. В противном случае метод является неявным и требует при своей реализации решения системы алгебраических уравнений. Среди неявных методов Рунге‑Кутты наиболее просто реализуются диагонально неявные (DIRK – Diagonally Implicit Runge‑Kutta), у которых матрица **A** имеет нижнюю треугольную форму.

В **SimInTech** реализованы три метода DIRK, имеющие таблицу Бутчера вида

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | |  |  | | (3.4) |

Эти методы имеют явную первую стадию и *r* = *s* – 1 неявных стадий с одинаковыми диагональными элементами матрицы **A**, поэтому их обычно называют ESDIRK (Explicit first stage Singly DIRK). Явная стадия не требует вычислений, поскольку результат ее выполнения совпадает с последней стадией предыдущего шага. Такие методы называют иногда также FSAL-DIRK (First Same As Last). Преимущество методов вида (3.4) по сравнению с другими методами DIRK состоит в том, что они являются жестко точными и могут иметь второй стадийный порядок [7, 8].

### Диагонально неявный (DIRK2)

Метод, задаваемый таблицей Бутчера

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |

имеет 2-й порядок. Его можно интерпретировать как последовательное применение правила трапеций и формулы дифференцирования назад второго порядка, поэтому он получил название TR-BDF2. Этот метод реализован также и в системе MATLAB/Simulink под названием Ode23tb. Отметим, что реализации этого метода в **SimInTech** и в MATLAB различаются, вследствие чего могут немного различаться и результаты его применения в этих системах.

### DIRK3

Метод 3-го порядка с 3-мя неявными стадиями и коэффициентами

### DIRK4

Метод 4-го порядка с 4-мя неявными стадиями.

Эти методы предложены в [7], там же приведены их коэффициенты и детали реализации.

# Решение тестовых задач (примеры)

Решатели ОДУ **SimInTech** были испытаны на множестве разнообразных задач, среди которых – задачи из тестовых наборов, приведенных в [2, 3, 9]. Результаты тестовых испытаний в сравнении с решателями MATLAB представлены в [10]. Среди задач были нежесткие, жесткие, локально-неустойчивые, осциллирующие, а также задачи с разрывами. Оценивались точность решения и вычислительные затраты. Нежесткие задачи оказались самыми легкими и были успешно решены всеми методами с автоматическим выбором шага. Поэтому остановимся на жестких задачах, численное решение которых часто встречает затруднения.

## Нежесткие задачи

## Жесткие задачи

Посмотрим, как влияет жесткость задачи на точность ее решения и вычислительные затраты.

### Задача Капса

Будем решать задачу Капса

которая имеет плавное решение *x*₁*(t) = exp(-*2*t), x*₂*(t) = exp(-t)*, не зависящее от параметра жесткости *μ* (собственные значения якобиана при больших *μ* примерно равны –*μ*, –1). В качестве меры вычислительной работы будем использовать число вычислений правой части *Nf* (включая также и вычисления, выполняемые при расчете матрицы Якоби), а точность численного решения будем оценивать по формуле

где *error* – максимальная относительная ошибка на всем интервале интегрирования. Таким образом, *scd* – минимальное число правильных значащих цифр численного решения (от англ. significant correct digits). Результаты для явных методов и трех лучших неявных методов при *hmin* = 1e-6, *hmax* = 1, *relerr* = 1e-3, *abserr* = 1e-10 приведены в таблице 1.

Таблица 1. Точность и вычислительные затраты при решении задачи Капса

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | *scd Nf* | *scd Nf* | *scd Nf* | *scd Nf* | *scd Nf* |
| Рунге-Кутты кл.  Рунге-Кутты мод. Мерсона кл. Мерсона мод. | 3.7   110 3.7   110 4.9   100 4.9   100 | 4.1   264 3.6   198 4.5   250 4.5   250 | 0.4   2639 3.3   528 3.8   1419 4.4   835 | -0.7   25770 3.1   605 3.9   14098 3.9   4133 | -1.9   255720 3.4   1210 4.1   140902 3.1   19715 |
| Адаптивный 1 Адаптивный 2 Адаптивный 3 Адаптивный 4 Адаптивный 5 | 4.0   87 3.8   120 3.7   75 5.0   42 4.3   120 | 3.6   143 3.7   208 3.5   125 5.6   48 3.6   217 | 2.7   237 2.2   248 3.4   150 5.6   51 3.9   260 | 2.1   304 2.0   256 3.4   165 5.5   60 3.9   270 | 2.0   319 1.8   240 3.6   195 5.5   60 3.9   270 |
| Гира DIRK3 DIRK4 | 2.9   39 2.6   36 4.0   55 | 3.1   40 3.0   49 4.0   53 | 3.1   40 3.3   34 4.2   47 | 3.0   40 3.3   34 3.6   41 | 3.0   40 3.3   34 3.6   41 |

Вычислительные затраты классических явных методов возрастают практически линейно при увеличении жесткости, что вызвано необходимостью уменьшать шаг интегрирования для обеспечения устойчивости численного решения. Значительное снижение точности классического метода Рунге‑Кутты объясняется отсутствием SC‑устойчивости. Модифицированные методы эффективнее классических, но заметно уступают адаптивным и неявным методам. Адаптивные методы демонстрируют небольшое увеличение объема вычислений при повышении жесткости (а первые два из них также и снижение точности), что вызвано снижением порядка при решении жестких задач. Точность и вычислительные затраты неявных методов практически не зависят от жесткости. Результаты решения задачи Капса подтвердили эффективность метода Гира, который традиционно считается одним из лучших для жестких задач, а также показали перспективность новых методов (адаптивных, DIRK3, DIRK4).

### Задачи из тестового набора

Для более полного тестирования жестких решателей **SimInTech** и MATLAB были выбраны первые шесть задач из тестового набора, приведенного в [Test Set for Initial Value Problem Solvers. Release 2.2. August 2003], а именно: VDPOL, ROBER, OREGO, HIRES, E5, PLATE. Результаты приведены в [12] и в Приложении. С приемлемыми затратами и точностью решили все шесть задач четыре решателя **SimInTech** (Диагонально неявный, Гира, DIRK3, DIRK4) и два решателя системы MATLAB (ode15s и ode23tb). В [12] приведен график зависимости суммарных вычислительных затрат от усредненной по всем задачам точности для этих решателей. Лучшие результаты – у методов Гира, DIRK3, DIRK4 и ode15s, при этом методы Гира и ode15s имеют небольшое преимущество при расчетах с высокой точностью, а решатели DIRK3 и DIRK4 при той же задаваемой точности обеспечивают, как правило, более высокую точность. Отметим, что явный решатель Адаптивный 4 оказался лучшим среди всех методов при решении жесткого уравнения Ван-дер-Поля (задача VDPOL).

Приведем также результаты сравнения решателей **SimInTech** с решателем RADAU, реализующим методы Radau IIa порядков 5, 9 и 13 [6, 11]. По результатам тестирования наиболее известных программ этот решатель был лучшим для большинства задач из [11]. Мы выбрали задачи VDPOL и OREGO, которые решали при *relerr* = *abserr* = 1e-4 согласно условиям, приведенным в [11]. Из этой же работы заимствованы результаты для RADAU (tables II.8.4, II.9.3 в [11]). Полученные результаты представлены в таблице 2. Вычислительные затраты оценивались следующими показателями: *Nf* – число вычислений функции (сюда не включены вычисления, выполняемые при расчете якобиана); *NJac* – число вычислений якобиана; *NLU* – число LU‑разложений. При вычислении *scd* по формуле (4.1) использовалась максимальная относительная ошибка в конце интервала интегрирования. Отметим высокую эффективность явного адаптивного метода, который показывает превосходные результаты также и при решении других жестких задач (например, задачи CUSP из [6], имеющей 96‑й порядок и 32 жестких собственных значения). Таким образом, явные адаптивные методы могут успешно конкурировать с лучшими неявными методами при решении многих жестких задач.

Таблица 2. Точность и вычислительные затраты при решении двух жестких задач

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Задача | Метод | *scd* | *Nf* | *NJac* | *NLU* |
| VDPOL | Адаптивный 4 Гира DIRK3 DIRK4 RADAU | 4.11 3.39 3.99 4.08 4.44 | 1756 1361 2221 2711 2214 | 0 151 75 94 165 | 0 265 278 242 231 |
| OREGO | Адаптивный 4 Гира DIRK3 DIRK4 RADAU | 3.50 1.78 2.36 3.66 3.12 | 2931 1410 2396 3160 3416 | 0 236 149 157 200 | 0 356 317 350 267 |

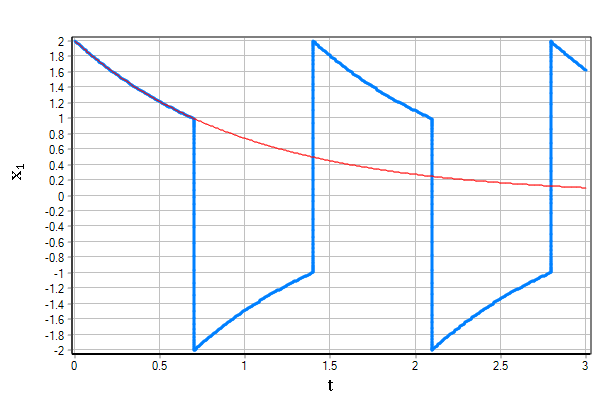
## Локально-неустойчивые задачи

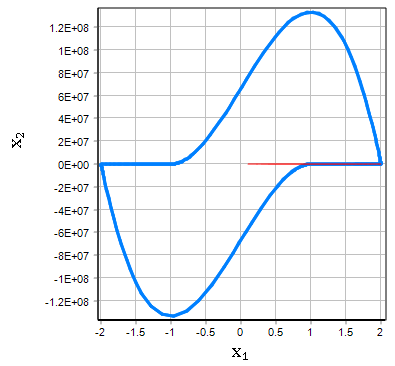
Рассмотрим один тип задач, решение которых неявными методами может привести к качественно неверному результату. Если решение гладкое, то шаг интегрирования устанавливается большим, но тогда неявные методы подавляют все составляющие решения, соответствующие большим по модулю собственным значениям якобиана, независимо от знака действительной части. Поэтому если одно из собственных значений быстро перемещается в правую полуплоскость и становится очень большим, то неявный решатель может этого не заметить, и неустойчивый процесс будет моделироваться как устойчивый.

### Осциллятор Ван-дер-Поля с более резким переходом

Рассмотрим задачу

построенную аналогично осциллятору Ван-дер-Поля, но имеющую более резкий переход от устойчивого состояния к неустойчивому. При больших значениях *μ* задача является жесткой. Правильное решение этой задачи при *μ* = 1e8 показано на рисунке голубой линией. Однако неявные методы обычно дают неправильное решение, показанное на рисунке красной линией, которое практически не изменяется при изменении задаваемой точности в широких пределах. Этот факт может привести к ошибочному мнению, что данное решение является правильным. На приведенном примере видно, что при моделировании процессов, имеющих быстро нарастающий, катастрофический характер, неявные методы могут давать совершенно неверный результат, соответствующий устойчивому процессу. Явные адаптивные методы позволяют качественно правильно решать подобные задачи. При решении рассмотренной задачи все адаптивные методы давали правильное решение при умеренных требованиях к точности, при этом лучшими были методы Адаптивный 2, 3, 5.





## Осциллирующие задачи

## Задачи с разрывами

# Решение дифференциально-алгебраических уравнений (ДАУ)

Часто уравнения математической модели представлены не в нормальной форме Коши (1.1), а в виде системы ДАУ

## Решение ДАУ явными методами

## Решение ДАУ неявными методами

## Системы ДАУ высших индексов

Согласно определению Гира и др., индекс дифференцирования системы (5.1) есть наименьшее число аналитических дифференцирований, требующихся для того, чтобы из уравнений (5.1) можно было бы получить систему ОДУ в форме Коши. При этом каждое дифференцирование понижает индекс на 1.

# Заключение

# Литература и Интернет

1. Современные численные методы решения обыкновенных дифференциальных уравнений / Под ред. Дж. Холла и Дж. Уатта. М.: Мир, 1979.
2. Хайрер Э., Нёрсетт С., Ваннер Г. Решение обыкновенных дифференциальных уравнений. Нежесткие задачи. М.: Мир, 1990.
3. Хайрер Э., Ваннер Г. Решение обыкновенных дифференциальных уравнений. Жесткие и дифференциально-алгебраические задачи. М.: Мир, 1999.
4. Скворцов Л. М. Адаптивные методы численного интегрирования в задачах моделирования динамических систем // Изв. РАН. Теория и системы управления. 1999. № 4. С. 72–78.
5. Скворцов Л. М. Явные адаптивные методы численного решения жестких систем // Математическое моделирование. 2000. № 12. С. 97-107.
6. Dormand, J. R.; Prince, P. J. (1980), "A family of embedded Runge-Kutta formulae", Journal of Computational and Applied Mathematics 6 (1): 19–26, doi:10.1016/0771-050X(80)90013-3. ( [www.sciencedirect.com/science/article/pii/0771050X80900133](http://www.sciencedirect.com/science/article/pii/0771050X80900133) )
7. Скворцов Л. М. Диагонально неявные FSAL-методы Рунге-Кутты для жестких и дифференциально-алгебраических систем // Математическое моделирование. 2002. Т. 14. № 2. С. 3–17.
8. Скворцов Л. М. Точность методов Рунге Кутты при решении жестких задач // Ж. вычисл. матем. и матем. физ. 2003. Т. 43. № 9. С. 1374–1384.
9. Test Set for Initial Value Problem Solvers. Release 2.2. August 2003.
10. Козлов О. С., Скворцов Л. М. Тестовое сравнение решателей ОДУ системы MATLAB // Всероссийская научная конференция «Проектирование научных и инженерных приложений в среде MATLAB». М.: Изд-во ИПУ РАН, 2002. С. 53–60.