

Пакет \mathbf{L}_2 v2 β для Maple V R4*

В. З. Цалюк[†]

1 октября 2009 г.

Работа выполнена при поддержке Российского Фонда Фундаментальных Исследований, грант № 07-01-96060-р-Урал (совместно с администрацией Пермского края).

Аннотация

Настоящий документ содержит описание пакета \mathbf{L}_2 версии v2 β (\mathbf{L}_2 v2 β). Это рабочая версия в процессе развития пакета.

Пакет \mathbf{L}_2 предназначен для производства вычислений с кусочно-полиномиальными функциями в среде символьных вычислений Maple V R4.

Библиотека `l2_lib.m` реализует операции над кусочно-полиномиальными функциями, рассматривая их как элементы гильбертова пространства $\mathbf{L}_2(a, b)$. Кроме этого, она предоставляет возможность производить вычисления с интегральными операторами в этом пространстве, оперируя с кусочно-полиномиальными ядрами этих операторов.

Библиотека `bvp_lib.m` работает с кусочно-полиномиальными функциями, рассматривая их как решения или коэффициенты уравнений или краевых условий линейной краевой задачи. В частности, имеется возможность вычислять значения функций, дифференцировать и интегрировать их. Имеется процедура, вычисляющая функцию Грина для краевой задачи для простейшего дифференциального уравнения вида $\frac{d^n}{dt^n} x = z$.

В составе пакета также имеется архив с примерами применения процедур пакета, в том числе для решения прикладных задач.

Настоящий документ содержит описание основных мотивов, стимулировавших работу над пакетом, и полную документацию для процедур вышеупомянутых библиотек.

*© Copyright В.З. Цалюк, 2006–2009.

© Copyright М. Половинкин, 2009, отдельные процедуры.

[†]Кубанский государственный университет, г. Краснодар

E-mail: vts@math.kubsu.ru

Содержание

Указатель процедур пакета	4
0 Введение. Зачем это?	5
1 Установка пакета	6
1.1 Системные требования	6
1.2 Состав пакета	6
1.3 Установка пакета	6
2 Терминология	7
3 Соглашение об именах	9
4 Основные типы данных	11
4.1 L_2 -функции	11
4.2 Ядра интегральных операторов	12
4.3 Векторно-матричные данные	13
5 Режимы работы пакета	14
6 Библиотека <code>l2_lib</code>	15
6.1 Служебные процедуры	15
6.2 Процедуры для работы с функциями	17
6.2.1 Операции гильбертова пространства	17
6.2.2 Функция + функция \rightarrow функция	18
6.2.3 Функция \rightarrow функция	19
6.2.4 Функция + \dots + функция \rightarrow число (матрица)	19
6.2.5 Функция \rightarrow график	20
6.3 Процедуры для работы с интегральными операторами	21
6.3.1 Ядро + \dots + ядро \rightarrow ядро	21
6.3.2 Ядро + функция \rightarrow функция	22
6.3.3 Ядро + функция \rightarrow ядро	22
6.3.4 Ядро \rightarrow ядро	23
6.3.5 Ядро \rightarrow функция	24
6.3.6 Функция \rightarrow ядро	24
6.3.7 Функция + \dots + функция \rightarrow ядро	25

7	Библиотека <code>bvp_lib</code>	27
7.1	Служебные процедуры	27
7.2	Процедуры для работы с функциями	28
7.3	Функции Грина	30
7.4	Подготовка данных для процедур <code>bvp_FKgreen</code> и <code>bvp_Fpoly</code>	33
8	Библиотека <code>l2m_lib</code>	35
8.1	Служебные процедуры	36
8.2	Процедуры для работы с функциями	37
8.2.1	Операции гильбертова пространства	37
8.2.2	Функция + ... + функция → функция	38
8.2.3	Функция → функция	38
8.2.4	Функция + ... + функция → числовая матрица	38
8.3	Процедуры для работы с интегральными операторами	39
8.3.1	Ядро + ядро → ядро	39
8.3.2	Ядро + функция → функция	39
8.3.3	Функция + ядро + ... + функция + ядро → ядро	39
8.3.4	Ядро → ядро	40
8.3.5	Функция + ... + функция → ядро	40
9	Лицензия	42
10	Переименования	43
	Список литературы	44

Указатель процедур пакета

12_EFcomb, 17
12_EFmult, 17
12_EKcomb, 21
12_EKmult, 22
12_Fcheck, 15
12_Fconjugate, 19
12_FinnerProd, 17
12_FinnerSubst, 18
12_Fint, 19
12_FKdegen, 25
12_FKinnerSubst, 23
12_FKmult, 22
12_FKmult0, 25
12_FKprojector, 26
12_FKresolvent, 25
12_FmatrixGram, 20
12_Fmult, 18
12_Fnorm2, 18
12_Fpiecewise, 19
12_Fplot, 20
12_FplusFunc, 18
12_Fprune, 16
12_FsConvert, 24
12_Fsum, 17
12_FtConvert, 24
12_init, 15
12_Kadjoint, 24
12_Kcheck, 15
12_Kconjugate, 23
12_KFapplyOp, 22
12_KFmult, 22
12_Kmult, 21
12_Kprune, 16
12_Ksum, 21
12_Ksuperpos, 21
12_KtSection, 24
12_setCheck, 16
12_setRelease, 16
bvp_FcombPrepData, 34
bvp_Fcontin, 28
bvp_Fdiff, 28
bvp_FgatherPrepData, 34
bvp_FinSobolev, 29
bvp_FintPrepData, 33
bvp_FKgreen, 30
bvp_Fpoly, 31
bvp_Fprim, 28
bvp_FvalPrepData, 33
bvp_Fvalue, 28
bvp_KC, 30
bvp_KG, 30
bvp_order, 27
12m_col, 36
12m_EFcomb, 37
12m_EFmult, 37
12m_EKcomb, 40
12m_EKmult, 39
12m_Fconjugate, 38
12m_FinnerProd, 37
12m_FKmult0, 40
12m_FKprojector, 40
12m_FmatrixGram, 38
12m_Fmult, 38
12m_Fnorm2, 37
12m_Fsum, 37
12m_Kconjugate, 40
12m_KEmult, 39
12m_KFapplyOp, 39
12m_Kmult, 39
12m_Ksum, 39
12m_Ksuperpos, 39
12m_row, 36
12m_toMatrix, 36

0 Введение. Зачем это?

После выхода в свет книги [2] у нас появилась потребность в вычислениях в пространстве L_2 функций, определенных на интервале (a, b) числовой оси. Все, чем мы обычно располагаем для задания таких функций, это кусочное описание формулами. Но разбиение интервала (a, b) на „куски“ может быть различным для разных функций и может изменяться в результате операций над функциями. Поэтому как разбиение отрезка на „куски“, так и выражения, задающие функцию на этих „кусках“, являются составными частями структуры, содержащей описание кусочно заданной функции.

Интегральные операторы

$$(\mathbf{K}x)(t) = \int_a^b K(t, s)x(s) ds.$$

в пространстве $L_2(a, b)$ задаются их ядрами $K(t, s)$. Ядра операторов также могут быть кусочно заданными. Границы „кусков“, на которые разбивается квадрат $(a, b) \times (a, b)$, могут быть прямыми (отрезками прямых).

В качестве основы пакета была выбрана система символьных вычислений Maple, т.к. это дало возможность решать задачи, избегнув ошибки дискретизации и округления при вычислениях.

Версия **Maple V R4** (Campus Wide Version) была выбрана в силу ее общедоступности: согласно [5, с. 99], она «свободно распространяется на CD-ROM и по сети Internet. Однако она может использоваться пользователями только для целей образования, но не для коммерческих целей.»

Указанная книга явилась для меня единственным, кроме англоязычного help'a самой системы, источником сведений о программировании в **Maple V R4**. Она написана явно наспех и не всюду осмысленно; часто мы встречаем в ней просто не всегда грамотный подстрочный перевод help'a. Тем не менее она оказалась очень полезна для работы, за что я хотел бы выразить ее автору и издательству свою искреннюю признательность.

1 Установка пакета

1.1 Системные требования

...определяются наличием работоспособного пакета Maple V R4. Настоящий пакет тестировался в операционных системах MS Windows 98SE, MS Windows XP.

1.2 Состав пакета

В состав пакета входят следующие файлы:

Имя файла	Описание	
l2_lib.m	Основная библиотека пакета	
bvp_lib.m	„Прикладная“ библиотека, посвященная краевым задачам	
l2m_lib.m	Библиотека для работы с векторно-матричными данными	
l2doc.pdf	Настоящий файл документации	
adjoint.zip	Архивы с примерами использования: Действие сопряженного оператора на функционал	
delay.zip		Вычисление критических значений параметров для вариационной задачи „с отклоняющимся аргументом“
green.zip		Вычисление и проверка функции Грина краевых задач для уравнения $x^{(n)} = z$
iters.zip		Эксперименты о сходимости алгоритма вычисления спектрального радиуса самосопряженного оператора
pillar.zip		Вычисление критической продольной нагрузки для вертикальной стойки переменного сечения
plot.zip		Обсуждение проблемы построения графика L_2 -функции и способов ее решения
examples.zip		Разнообразные примеры

Просмотреть (краткую) документацию по пакету в формате .htm и скачать последнюю версию можно по адресам
<http://vts.math.kubsu.ru/l2/l2.htm>,
<http://l2.pstu.ru/l2.htm>.

1.3 Установка пакета

Файлы библиотек *_lib.m скопируйте в папку, обозначенную системной переменной **libname** установленного пакета Maple V R4.

2 Терминология

Я полагаю, что искушенные пользователи Maple имеют представление о разнице между выражением и функцией, но на всякий случай считаю нужным обсудить это здесь.

Большинство процедур Maple использует и выдает в качестве значений не функции, а выражения. Функции, в действительности, используются в Maple редко. К сожалению, в help'е пакета авторы, с присущей школьникам (и алгебраистам) беззаботностью, часто называют выражения функциями, что затрудняет понимание.

Эту разницу, при необходимости, можно уяснить, разобрав предлагаемые здесь вопросы и ответы на них.

Q. Maple-функция `piecewise` создает выражение или функцию?

A. Выражение типа `piecewise`.

Q. `x(t)` — это функция или выражение?

A. Это выражение, значение которого является значением функции `x` при значении аргумента, равного `t`. Однако, значение аргумента может зависеть от какого-то другого аргумента.

Q. Что в Maple подвергается интегрированию и интегральным преобразованиям Лапласа, Фурье и т.д.?

A. Нет, не функции, а выражения, содержащие некоторую переменную, по которой и совершается интегрирование.

Q. Если `x` — функция одного аргумента, то `int(x(s), s = 0..t)` представляет собой выражение или функцию?

A. Выражение, содержащее переменную `t`. Если мы хотели получить функцию, нам следовало написать

```
func := t -> int(x(s), s = 0..t);
```

Q. Как получить значение функции `x` в точке `1/2`?

A. `x(1/2)`;

Q. Как подставить значение `t = 1/2` в выражение `x`?

A. `subs(t = 1/2, x)`

Q. `x(t)` и `x(s)` — это одно и то же или нет?

A. Нет, это разные выражения. Однако, они равны, если переменным `t` и `s` было присвоено одно и то же значение.

A функция одна и та же в любом случае!

Q. Что будет выведено на экран в результате выполнения следующих двух последовательностей команд? Они довольно часто встречаются в практике программирования итерационных процессов.

Для выражений:

Для функций:

```
f := t:
f_old := f:
f := t^2:
f - f_old;

f := t -> t:
f_old := f:
f := t -> t^2:
f(t) - f_old(t);
```

A. В версии **Maple V R4** в первом случае получается $t^2 - t$, а во втором — 0. Что наводит на очень глубокие мысли о возможных источниках ошибок.

У Вас другая, более совершенная версия? — Тогда проверьте сами.

Q. Пусть X — выражение и

```
x := t -> X:
```

Если в выражение $x(t)$ подставить s вместо t :

```
subs(t=s, x(t)):
```

то получится ли то же, что и $x(s)$? А что получится, если в $x(t)$ подставить t вместо s :

```
subs(s=t, x(t)): ?
```

A. Это для вашего самостоятельного размышления. Главное, чтобы был понят вопрос!

3 Соглашение об именах

Выражения, моделирующие собой функции пространства, мы называем L_2 -функциями, не в честь пространства, а по имени пакета.

Выражения, моделирующие интегральные операторы, действующие из $L_2(a, b)$ в $L_2(a, b)$, а точнее, их ядра, мы называем ядрами, не создавая терминологических конфликтов.

Все процедуры и глобальные переменные пакета имеют имена, начинающиеся с символов `l2_`, `bvp_` или `l2m_`. Использование этих префиксов в каких-либо других целях — это, гм. . . , не самая умная идея.

После `l2_`, `bvp_` или `l2m_` имена процедур имеют, как правило, прописные буквы **K**, **F** и/или **E**, указывающие на тип(ы) операндов, участвующих в операции, реализуемой процедурой:

K — для ядер (т.е. интегральных операторов),

F — для L_2 -функций,

E — для выражений, тип которых никак не регламентируется. В частности, так могут появляться постоянные множители. Пользователю надлежит самостоятельно позаботиться о том, чтобы соответствующая операция с таким значением была осмысленной.

Затем со строчной буквы следует обозначение реализуемой операции (в сокращенном виде).

Например, процедура для вычисления скалярного произведения двух L_2 -функций имеет имя `l2_FinnerProd`. Если бы была написана процедура для вычисления скалярного произведения ядер (в пространстве $L_2((a, b) \times (a, b))$), то она бы имела имя `l2_KinnerProd`.

Пакет содержит 5 специальных глобальных переменных.

Границы интервала (a, b) устанавливаются и хранятся в переменных `l2_a` и `l2_b` соответственно. Переменная `l2_complex` логического типа указывает, над каким числовым полем: \mathbb{R} или \mathbb{C} — мы производим вычисления. Как известно, от этого зависят применяемые расчетные формулы. Эти переменные создаются и принимают свои значения при вызове какой-либо из процедур `*_init`.

Глобальная переменная `bvp_n` хранит в себе порядок краевой задачи, с которой работает (в текущий момент) библиотека `bvp_lib`. Ее значение устанавливается процедурой `bvp_order`.

Глобальная переменная `l2_check` логического типа управляет проверками параметров процедур на правильность (допустимость). Если `l2_check = true` (так называемый режим **check**), то производятся все

предусмотренные проверки. При `l2_check = false` (режим **release**) почти все проверки пропускаются. Последний режим рекомендуется тогда, когда программа отлажена и исходные данные проверены.

Две переменные, **t** и **s**, имеют сакральное значение для этого пакета, и не должны использоваться как-то иначе. Первая из них является универсальным обозначением аргумента всех рассматриваемых функций: $x(t)$ — а вторая используется вместе с первой для описания ядер интегральных операторов

$$(\mathbf{K}x)(t) = \int_a^b K(t, s)x(s) ds.$$

Дело в том, что в связи с описанными в параграфе 2 *особенностями* пакета Maple мы предпочитаем функции и ядра представлять не функциями, а выражениями. Содержащиеся в этих выражениях переменные **t** и **s** изображают аргумент t функций и аргументы (t, s) ядер.

Запрещено ограничивать переменные **t** и **s** предложением `assume(...)`, т.к. Maple V R4 не умеет **subs** переменные в выражениях, если они **assumed**.

4 Основные типы данных

4.1 L_2 -функции

Функции, являющиеся элементами пространства $L_2(a, b)$, описываются в нашем пакете кусочно-заданными выражениями. Они представляются в виде списка `list`

`x := [V_0, E_1, V_1, E_2, V_2, ..., E_n, V_n]:`

где V_i — точки разбиения отрезка $[a, b]$ на части, называемые здесь и далее *границами*. E_i — *выражения*, по которым вычисляются значения функции $x(t)$ для $t \in (V_{i-1}, V_i)$.

Границы V_i должны составлять упорядоченную последовательность чисел (`constant`), причем должны выполняться неравенства

$$l2_a \leq V_0 < V_n \leq l2_b.$$

Эти условия можно проверить процедурой `l2_Fcheck`.

Если между `l2_a` и V_0 имеется строгое неравенство $l2_a < V_0$, это означает, что $x(t) = 0$ для $t \in [a, V_0)$. Аналогичное соглашение имеет место и для отрезка $(V_n, b]$.

Список, состоящий ровно из одного элемента, как, например, `[E]`, воспринимается, как функция, равная E на всем отрезке $[a, b]$.

Если список `x` пуст или состоит из 2-х элементов, то это воспринимается как функция, тождественно равная 0.

Таким образом, для пространства $L_2(0, 3)$ выражения `[]`, `[0]`, `[0, 0, 3]`, `[1, 0, 2]`, `[1, t^2]` означают одно и то же — тождественный ноль.

Предложенная структура похожа на Maple-структуру `pwlist`. Отличие в том, что у нас крайние элементы списка являются точками разбиения, а не выражениями, имеющими место до бесконечности.

Пример: функцию

$$y(x) = |x - 1| - 1$$

из пространства $L_2(0, 2)$ можно задать в виде

`y := [0, abs(t-1) - 1, 2]:`

или в виде

`y := [0, t, 1, 2-t, 2]:`

Второй из способов предпочтительнее, т.к. меньше загружает Maple вычислительной работой.

Обратите внимание, что вместо аргумента x мы использовали специальную переменную `t`!

4.2 Ядра интегральных операторов

выражаются в виде списка `list`

$$K := [V_0, D_1, V_1, D_2, V_2, \dots, D_n, V_n]:$$

где V_i — точки разбиения отрезка $[a, b]$ на части, D_i — списки, похожие на L_2 -функции. Разница в том, что в них в качестве аргумента используется переменная s вместо t , а t может возникать как параметр, от которого могут зависеть как границы, так и выражения списка D_i .

Границы V_i должны составлять упорядоченную последовательность чисел (`constant`), причем должны выполняться неравенства $V_0 \geq l2_a$ и $V_n \leq l2_b$.

Если между $l2_a$ и V_0 имеется строгое неравенство $l2_a < V_0$, это означает, что $K(t, s) = 0$ для $t \in [a, V_0]$ и всех s . Аналогичное соглашение имеет место и для отрезка $(V_n, b]$.

Если ядро выражено списком, состоящим ровно из одного элемента (списка), как, например, $K := [D]$, то оно определяется списком D для всех $t \in [a, b]$.

Если список K пуст или состоит из 2-х элементов, то это воспринимается как ядро, тождественно равное 0.

Границы списка D_i должны составлять последовательность линейных выражений вида $c_1 * t + c_0$ (где c_0, c_1 — числа (`constant`)), строго возрастающую при каждом $t \in (V_{i-1}, V_i)$.

Эти условия можно проверить процедурой `l2_Kcheck`.

Примеры: 1) оператор

$$(Ky)(x) = \int_0^{1-x} sy(s) ds$$

в пространстве $L_2(0, 1)$ задается ядром

$$K := [0, [0, s, 1-t, 0, 1], 1]:$$

или, проще, в виде

$$K := [[0, s, 1-t]]:$$

2) Оператор

$$(Ky)(x) = \begin{cases} \int_0^x sy(s) ds, & 0 < x < 1, \\ 0, & 1 < x < 2, \\ x \int_0^3 y(s) ds, & 2 < x < 3, \end{cases}$$

в пространстве $L_2(0, 3)$ задается ядром

```
К := [0, [0, s, t], 1, [], 2, [t], 3]:
```

Обратите внимание, что вместо аргумента x мы использовали специальную переменную t !

4.3 Векторно-матричные данные

Для работы с вектор-функциями в составе пакета имеется библиотека `l2m_lib.m`. Такие данные надо заносить в стандартную конструкцию Maple

```
К := matrix(n, m):
```

или, что то же,

```
К := array(1..n, 1..m):
```

После чего каждому элементу матрицы следует присвоить в качестве значения L_2 -функцию или ядро, например,

```
К[1, 2] := ...
```

Если какой-либо из элементов не был заполнен, то в режиме **check** (с. 9), при обращении к процедуре библиотеки `l2m_lib.m` Вы получите сообщение об ошибке с указанием индексов этого элемента.

Использование типа **vector** пакетом L_2 не предусмотрено. Вместо этого предлагается создавать матрицы, состоящие из одного столбца или одной строки, в зависимости от ситуации. Недостатком такого подхода является то, что к элементу вектора приходится обращаться, используя двойной индекс. Преимущество — в универсальности процедур библиотеки `l2m_lib.m`.

Для создания таких данных можно использовать вспомогательные процедуры, описанные ниже в п. 8.1.

5 Режимы работы пакета

Процедуры пакета L_2 могут работать в одном из двух режимов. Эти режимы можно условно обозначить как **check** и **release**. По умолчанию, после инициализации пакета одной из процедур `*_init`, действует режим **check**. Установка одного из этих режимов производится процедурами `l2_setCheck` или `l2_setRelease`.

Режим **check** означает, что все исходные данные подвергаются наиболее придирчивой проверке по количеству параметров, по их типу, по соответствию размерностей операндов, по допустимости их значений и т.д. Неполные данные, которые могут быть интерпретированы по правилам, приведенным на с. 11–12, пополняются. Так, например, L_2 -функция `[t+2]` заменяется на `[l2_a, t+2, l2_b]`.

В режиме **release** почти все проверки пропускаются. Предполагается, что все исходные данные вводятся в полной форме: не `[]`, не `[0]`, а `[l2_a, 0, l2_b]`.

Если Вы создаете документ **Maple** только для однократного расчета, то режим **check** Вам не мешает. Если же расчет предполагается повторять многократно, то имеет смысл подумать о том, не включить ли в какой-то момент режим **release** для ускорения вычислительной работы.

6 Библиотека `l2_lib`

6.1 Служебные процедуры

`l2_init := proc(a::numeric,b::numeric)`

или

`l2_init := proc(a::numeric,b::numeric, complex::boolean)`

или

`l2_init := proc(a::numeric,b::numeric, complex::name)`

Не возвращает значения. Устанавливает внутренние переменные пакета, содержащие левую и правую границы интервала (a, b) . Порождает ошибку при $a \geq b$.

Значение аргумента `complex = true` означает, что все вычисления будут производиться над полем \mathbb{C} комплексных чисел. Можно вместо `true` написать просто `complex`.

Во всех остальных случаях, включая отсутствие этого аргумента, будут применяться более простые формулы, предназначенные для вычислений над полем \mathbb{R} действительных чисел.

`l2_Fcheck := proc(x::list)`

Проверяет корректность задания L_2 -функции `x`. Возвращает `true`, если проверка не нашла ошибок. При обнаружении ошибок (отклонений от описанных выше правил) возвращает `false` и выводит в окно Maple сообщение с диагностикой ошибки. После исправления диагностированных ошибок повторная проверка может выявить новые ошибки!

Любое четное количество элементов списка считается ошибкой.

Тип результата — логическая константа.

`l2_Kcheck := proc(K::list)`

Проверяет корректность задания ядра `K`. Возвращает `true`, если проверка не нашла ошибок. При обнаружении ошибок (отклонений от описанных выше правил) возвращает `false` и выводит в окно Maple сообщение с диагностикой ошибки. После исправления диагностированных ошибок повторная проверка может выявить новые ошибки!

Любое четное количество элементов списка считается ошибкой.

Тип результата — логическая константа.

Проверка исходных данных двумя последними процедурами может быть полезной в случае появления каких-либо странных, необъяснимых на первый взгляд, сообщений об ошибках.

```
l2_Fprune := proc(x::list)
```

Осуществляет „обрезку“ L_2 -функции, т.е. удаляет крайние выражения-элементы E_i , если они равны 0, и устраняет повторы, когда два соседних выражения E_i и E_{i+1} совпадают. Например, если

```
x := [0, 0, 1, 2*t, 2, 2*t, 3, 0, 4]:
```

то

```
l2_Fprune(x) = [1, 2*t, 3].
```

Тип результата — L_2 -функция.

Использование „обрезки“ может быть полезно, т.к. функции пакета, вообще говоря, не оптимизируют получаемые с их помощью результаты.

```
l2_Kprune := proc(K::list)
```

Осуществляет „обрезку“ ядра, т.е. удаляет крайние выражения-элементы D_i , если они равны 0 (представлены пустыми списками). Осуществляет „обрезку“ каждого выражения D_i . Устраняет повторы, когда два соседних выражения D_i и D_{i+1} совпадают.

Тип результата — ядро.

Использование „обрезки“ может быть полезно, т.к. функции пакета, вообще говоря, не оптимизируют получаемые с их помощью результаты.

```
l2_setCheck := proc()
```

или

```
l2_setCheck := proc(toCheck::boolean)
```

Устанавливает режим **check** (см. § 5), если параметр **toCheck=true** или отсутствует, или имеет неверный тип. Устанавливает режим **release**, если **toCheck=false**.

```
l2_setRelease := proc()
```

или

```
l2_setRelease := proc(toRelease::boolean)
```

Устанавливает режим **release** (см. § 5), если параметр **toRelease=true** или отсутствует, или имеет неверный тип. Устанавливает режим **check**, если **toRelease=false**.

6.2 Процедуры для работы с функциями

6.2.1 Операции гильбертова пространства

`l2_Fsum := proc(x::list, y::list)`

Вычисляет сумму двух L_2 -функций x и y :

$$x(t) + y(t).$$

Тип результата — L_2 -функция.

Примеры см. в файле `zip://green.zip/green.mws`.

См. также `l2_FplusFunc`.

`l2_EFmult := proc(f, x::list)`

Умножает L_2 -функцию x на ограниченное выражение f :

$$f(t) \cdot x(t).$$

Если это выражение — константа, то мы имеем операцию умножения вектора на скаляр.

Тип результата — L_2 -функция.

Примеры см. в файлах

`zip://green.zip/green.mws`,

`zip://pillar.zip/round.mws`.

`l2_EFcomb := proc(c_1,x_1::list, c_2,x_2::list,...
 c_n,x_n::list)`

Вычисляет линейную комбинацию L_2 -функций x_1, x_2, \dots, x_n с коэффициентами c_1, c_2, \dots, c_n соответственно:

$$\sum_{i=1}^n c_i \cdot x_i(t).$$

Тип результата — L_2 -функция.

Коэффициенты могут быть любыми выражениями — на ответственность пользователя. Количество пар параметров произвольно.

`l2_FinnerProd := proc(x::list, y::list)`

Вычисляет скалярное произведение L_2 -функций x и y :

$$\langle x, y \rangle = \int_a^b x(t) \overline{y(t)} dt.$$

Если `l2_complex = false`, то, конечно, операция комплексного сопряжения не производится.

Тип результата — константа.

Пример см. в файле `zip://examples.zip/adjoint.mws`.

`l2_Fnorm2 := proc(x::list)`

Вычисляет квадрат нормы L_2 -функции x в пространстве $L_2(a, b)$:

$$\|x\|^2 = \int_a^b x^2(t) dt.$$

Тип результата — константа.

Пример в файле `zip://pillar.zip/round.mws`.

6.2.2 Функция + функция \rightarrow функция

`l2_FplusFunc := proc(x::list, f)`

Прибавляет к значениям L_2 -функции x значения выражения f :

$$x(t) + f(t).$$

Тип результата — L_2 -функция.

`l2_Fmult := proc(x::list, y::list)`

Вычисляет произведение L_2 -функций x и y :

$$x(t) \cdot y(t).$$

(Эта операция возможна в пространстве $L_2(a, b)$, так как кусочно многочленные функции ограничены на $[a, b]$.)

Тип результата — L_2 -функция.

Пример в файле `zip://pillar.zip/round.mws`.

`l2_FinnerSubst := proc(x::list, h::list)`

Вычисляет значение оператора внутренней суперпозиции

$$(S_h x)(t) = \begin{cases} x(h(t)), & \text{если } h(t) \in [a, b], \\ 0, & \text{если } h(t) \notin [a, b] \end{cases} \quad (1)$$

для L_2 -функций x и h .

Тип результата — L_2 -функция.

Пользователя просят самому позаботиться о том, чтобы оператор S_h был корректно определен в том пространстве, элементом которого он считает L_2 -функцию x . Условия действия оператора S_h в пространствах суммируемых функций см. [6], [3, прилож. В.1].

Ограничение, накладываемое пакетом, состоит в том, чтобы решения уравнений $h(t) = b_i$, где b_i — (рациональные) границы L_2 -функции x , если эти решения лежат в $[a, b]$, были рациональны.

Примеры см. в файлах

```
zip://examples.zip/innerSubst.mws,
zip://iters.zip/iters3.mws,
zip://iters.zip/iters4.mws,
zip://iters.zip/iters5.mws.
```

6.2.3 Функция \rightarrow функция

```
l2_Fpiecewise := proc(x::list)
```

Превращает L_2 -функцию в выражение типа `piecewise`.

Тип результата — `piecewise`-выражение.

Эту процедуру удобно использовать для построения графиков L_2 -функций. Пример см. в файле

```
zip://pillar.zip/round.mws.
```

```
l2_Fconjugate := proc(x::list)
```

Для L_2 -функции x вычисляет комплексно сопряженную L_2 -функцию.

Если, конечно, `l2_complex = true`. Иначе возвращает ту же функцию x .

Тип результата — L_2 -функция.

6.2.4 Функция $+ \dots +$ функция \rightarrow число (матрица)

```
l2_Fint := proc(x::list)
```

Вычисляет интеграл от L_2 -функции x :

$$\int_a^b x(t) dt.$$

Тип результата — константа.

Можно вместо `l2_Fint(x)` писать `int(l2_Fpiecewise(x), t=l2_a..l2_b)`; но это будет считаться раза в 4 дольше.

```
l2_FmatrixGram := proc(phi_1::list,
                       phi_2::list,
                       ...,
                       phi_n::list)
```

Вычисляет матрицу Грама L_2 -функций `phi_1`, `phi_2`, ..., `phi_n`:

$$G = (\langle \varphi_i, \varphi_j \rangle)_{i,j=1}^n.$$

Тип результата — квадратная матрица.

6.2.5 Функция → график

```
l2_Fplot := proc(x::list, интервал аргументов, интервал значений,
                 color=цвет, ...опции);
```

Создает и (в случае точки с запятой) изображает график одной L_2 -функции `x`. *Интервал аргументов* и *интервал значений* задаются в виде (например) `0..1.3` и необязательны. Опция `color=цвет` обязательна! Остальные опции стандартной команды `plot` могут использоваться, но, например, опция `discont=false` не действует.

Изображение нескольких графиков на одном рисунке этой процедурой невозможно. Списки значений опций типа `color=[red, blue]` не рекомендуются, т.к. приводят к странным результатам.

Тип результата — график (рисунок).

Пример см. в файле `zip://plot.zip/plot.mws`. В этом же архиве — документ, описывающий проблемы, вызвавшие появление этой процедуры в пакете, и способы их решения.

6.3 Процедуры для работы с интегральными операторами

6.3.1 Ядро + ... + ядро → ядро

`l2_Ksum := proc(K_1::list, K_2::list)`

Вычисляет сумму ядер `K_1` и `K_2`:

$$K(t, s) = K_1(t, s) + K_2(t, s).$$

Тип результата — ядро.

Примеры см. в файлах

`zip://delay.zip/kernel.mws`,

`zip://green.zip/green.mws`.

`l2_Kmult := proc(K_1::list, K_2::list)`

Вычисляет произведение ядер `K_1` и `K_2`:

$$K_1(t, s) \cdot K_2(t, s).$$

Тип результата — ядро.

Пример см. в файле `zip://examples.zip/Kmult.mws`.

`l2_Ksuperpos := proc(K1::list, K2::list)`

Вычисляет ядро оператора $K = K_1 \cdot K_2$:

$$K(t, s) = \int_a^b K_1(t, \tau) K_2(\tau, s) d\tau.$$

Тип результата — ядро.

Примеры см. в файлах

`zip://examples.zip/adjoint.mws`,

`zip://delay.zip/kernel.mws`,

`zip://examples.zip/super.mws`.

`l2_EKcomb := proc(c_1, K_1::list, c_2, K_2::list, ...
 c_n, K_n::list)`

Вычисляет линейную комбинацию ядер `K_1`, `K_2`, ..., `K_n` с коэффициентами `c_1`, `c_2`, ..., `c_n` соответственно:

$$\sum_{i=1}^n c_i \cdot K_i(t, s).$$

Тип результата — ядро.

Коэффициенты могут быть любыми выражениями — на ответственность пользователя. Количество пар параметров произвольно.

6.3.2 Ядро + функция → функция

`l2_KFapplyOp := proc(K::list, x::list)`

Вычисляет значение интегрального оператора \mathbf{K} на L_2 -функции \mathbf{x} :

$$(\mathbf{K}x)(t) = \int_a^b K(t, s)x(s) ds.$$

Тип результата — L_2 -функция.

Примеры — в файлах

`zip://examples.zip/adjoint.mws,`

`zip://examples.zip/super.mws,`

`zip://delay.zip/K1s5.mws,`

`zip://green.zip/green.mws.`

6.3.3 Ядро + функция → ядро

`l2_FKmult := proc(x::list, K::list)`

Вычисляет произведение L_2 -функции \mathbf{x} (аргумента t) и ядра \mathbf{K} :

$$x(t) \cdot K(t, s).$$

Тип результата — ядро.

Эта процедура вычисляет ядро произведения \mathbf{xK} , где \mathbf{x} — оператор умножения на функцию $x(\cdot)$, \mathbf{K} — интегральный оператор с ядром $K(t, s)$.

Пример см. в файле `zip://examples.zip/Kmult.mws`.

`l2_KFmult := proc(K::list, x::list)`

Вычисляет произведение ядра \mathbf{K} и L_2 -функции \mathbf{x} (аргумента s):

$$K(t, s) \cdot x(s).$$

Заменять \mathbf{t} на \mathbf{s} в аргументе \mathbf{x} не следует — это делается в самой функции

`l2_KFmult`.

Тип результата — ядро.

Эта процедура вычисляет ядро произведения \mathbf{Kx} .

Пример см. в файле `zip://examples.zip/Kmult.mws`.

`l2_EKmult := proc(f, K::list)`

Умножает ядро \mathbf{K} на ограниченное выражение \mathbf{f} :

$$f(t, s) \cdot K(t, s).$$

Тип результата — ядро.

Пример см. в файле

`zip://delay.zip/kernel.mws.`

```
l2_FKinnerSubst := proc(K::list, h::list)
```

Для ядра \mathbf{K} и \mathbf{L}_2 -функции \mathbf{h} вычисляет ядро интегрального оператора $S_h\mathbf{K}$, где S_h — определенный согласно (1) оператор внутренней суперпозиции, а \mathbf{K} — интегральный оператор, порожденный ядром \mathbf{K} .

Тип результата — ядро.

Как и ранее, оператор S_h должен быть корректно определен в том пространстве, элементом которого он считает \mathbf{L}_2 -функцию \mathbf{x} (см. [6], [3, прилож. В.1]).

Ограничения, накладываемое пакетом, состоят в том, чтобы

\mathbf{L}_2 -функция \mathbf{h} была строго монотонна в промежутках между соседними границами. Это условие не проверяется, но при его нарушении правильность результата не гарантирована;

решения уравнений $h(t) = b_i$, где b_i — границы \mathbf{L}_2 -функции \mathbf{x} , если эти решения лежат в $[a, b]$, были рациональны.

Примеры см. в файлах

```
zip://delay.zip/kernel.mws,
```

```
zip://examples.zip/innerSubst.mws.
```

6.3.4 Ядро \rightarrow ядро

```
l2_Kconjugate := proc(K::list)
```

Для ядра \mathbf{K} вычисляет комплексно сопряженное ядро. Если, конечно, `l2_complex = true`. Иначе возвращает то же ядро \mathbf{K} . (Транспонирование матрицы не производится!).

Тип результата — ядро.

`l2_Kadjoint := proc(K::list)`

Вычисляет ядро оператора, сопряженного к оператору \mathbf{K} :

$$K^*(t, s) = \overline{K(s, t)},$$

приведя его к стандартному представлению. Если `l2_complex = false`, то, конечно, операция комплексного сопряжения не производится.

Тип результата — ядро.

Примеры см. в файлах

`zip://delay.zip/kernel.mws,`

`zip://examples.zip/adjoint.mws,`

`zip://examples.zip/super.mws.`

6.3.5 Ядро \rightarrow функция

`l2_KtSection := proc(K::list, c::rational)`

Строит сечение ядра по заданному значению $t = c$:

$$k_c(s) = K(c, s).$$

Тип результата — \mathbf{L}_2 -функция (конечно же, аргумента t).

Если c совпадет с точкой разрыва, соблюдается правило „непрерывности справа“. Если $c \notin [a, b]$, результат будет нулевой.

Пример в файле `zip://examples.zip/section.mws`.

6.3.6 Функция \rightarrow ядро

`l2_FtConvert := proc(x::list)`

Преобразует \mathbf{L}_2 -функцию \mathbf{x} в ядро $K(t, s) = x(t)$.

Тип результата — ядро.

`l2_FsConvert := proc(x::list)`

Преобразует \mathbf{L}_2 -функцию \mathbf{x} в ядро $K(t, s) = x(s)$.

Тип результата — ядро.

6.3.7 Функция + ... + функция → ядро

```
12_FKmult0 := proc(phi::list,psi::list)
```

Вычисляет ядро

$$K(t, s) = \varphi(t)\psi(s),$$

порожденное \mathbf{L}_2 -функциями **phi** и **psi**.

Тип результата — ядро.

Это вспомогательная процедура для ряда следующих...

```
12_FKdegen := proc(phi_1::list,psi_1::list,  
                  phi_2::list,psi_2::list,  
                  ...,  
                  phi_n::list,psi_n::list)
```

Вычисляет (вырожденное) ядро

$$K(t, s) = \sum_{i=1}^n \varphi_i(t) \cdot \psi_i(s)$$

конечномерного оператора

$$\mathbf{K}x = \sum_{i=1}^n \varphi_i \langle x, \bar{\psi}_i \rangle,$$

порожденного \mathbf{L}_2 -функциями **phi_i**, **psi_i**, $i = 1, \dots, n$.

Тип результата — ядро.

Количество пар параметров произвольно.

Пример см. в файле `zip://examples.zip/resolv.mws`.

```
12_FKresolvent := proc(phi_1::list,psi_1::list,  
                      phi_2::list,psi_2::list,  
                      ...,  
                      phi_n::list,psi_n::list)
```

Вычисляет резольвенту $R(t, s)$ вырожденного ядра $K(t, s)$, указанного в описании предыдущей функции **12_FKdegen**. Точнее, вычисляет ядро интегрального оператора \mathbf{R} , такого, что

$$I + \mathbf{R} = (I - \mathbf{K})^{-1}.$$

Именно,

$$R(t, s) = \sum_{i=1}^n \sum_{j=1}^n g_{ij} \varphi_i(t) \psi_j(s),$$

где матрица $(g_{ij})_{i,j=1}^n$ — обратная к матрице $E - (\langle \varphi_j, \overline{\psi_i} \rangle)_{i,j=1}^n$.

Если оператор $I - \mathbf{K}$ необратим, генерируется ошибка.

Список аргументов — такой же, как у функции `l2_FKdegen`. Количество пар параметров произвольно.

Тип результата — ядро.

Пример см. в файле `zip://examples.zip/resolv.mws`.

```
l2_FKprojector := proc(phi_1::list,  
                       phi_2::list,  
                       . . . ,  
                       phi_n::list)
```

Вычисляет ядро $P(t, s)$ интегрального оператора — ортогонального проектора на линейную оболочку линейно независимой системы \mathbf{L}_2 -функций `phi_1, phi_2, . . . , phi_n`:

$$P(t, s) = \sum_{i=1}^n \sum_{j=1}^n \overline{\gamma_{ij}} \varphi_i(t) \overline{\varphi_j(s)},$$

где матрица $(\gamma_{ij})_{i,j=1}^n$ — обратная к матрице Грама $(\langle \varphi_i, \varphi_j \rangle)_{i,j=1}^n$.

Если система линейно зависима, генерируется ошибка.

Тип результата — ядро.

Пример см. в файле `zip://examples.zip/projector.mws`.

7 Библиотека `bvp_lib`

Эта библиотека состоит из процедур, не имеющих непосредственного отношения к пространству \mathbf{L}_2 . Они решают задачи, возникшие при рассмотрении краевых задач для обыкновенных дифференциальных уравнений на отрезке $[a, b]$.

Таким образом, эта библиотека рассматривает \mathbf{L}_2 -функции как элементы таких пространств, как соболевские пространства \mathbf{W}_p^n ($p \geq 1$)¹, или пространства $\mathbf{K}\mathbf{C}^n$ (в обозначениях [4]).

Для решений краевых задач естественно, например, вычислять значение в точке $c \in [a, b]$, что совершенно не характерно для элементов пространства \mathbf{L}_2 .

Глобальная переменная `bvp_n` хранит в себе порядок n краевой задачи, с которой работает (в текущий момент) библиотека `bvp_lib`.

Для инициализации задачи следует воспользоваться процедурой `l2_init(...)`:

```
l2_lib[l2_init](...):
```

или

```
with(l2_lib, [l2_init]):  
l2_init(...):
```

7.1 Служебные процедуры

```
bvp_order := proc(n::posint)
```

Не возвращает значения. Устанавливает значение глобальной переменной `bvp_n := n` — порядок краевой задачи.



¹ Так же, как \mathbf{L}_p вместо \mathbf{L}_2 , здесь вместо $\mathbf{H}^n = \mathbf{W}_2^n$ можно иметь в виду \mathbf{W}_p^n .

7.2 Процедуры для работы с функциями

bvp_Fvalue := proc(x::list, p::realcons)

Вычисляет значение L_2 -функции x в точке p :

$$x(p).$$

Тип результата — число.

Значения в точках разбиения при возможности вычисляются „по непрерывности справа“.

Примеры см. в файле `zip://green.zip/green.mws`.

bvp_Fdiff := proc(x::list, v_1, ..., v_n)

Вычисляет производную L_2 -функции x по переменным v_1, v_2, \dots, v_n . Если на какой-то стадии происходит дифференцирование разрывной функции, то производная вычисляется кусочно, без возникновения дельта-функций, но на экран выводится предупреждение:

WARNING: derivation of discontinuous function.

Тип результата — L_2 -функция.

Примеры см. в файле `zip://green.zip/green.mws`.

bvp_Fprim := proc(x::list, k::posint)

Вычисляет первообразную z k -того порядка от L_2 -функции x , удовлетворяющую начальным условиям $z(a) = \dots = z^{(k-1)}(a) = 0$.

Второй параметр необязателен, в этом случае производится вычисление однократной первообразной ($k = 1$).

Тип результата — L_2 -функция.

Примеры см. в файле `zip://examples.zip/sobolev.mws`.

bvp_Fcontin := proc(x::list)

Проверяет L_2 -функцию x на непрерывность. Возвращает **true**, если функция непрерывна (в понятном, надеюсь, смысле) на отрезке $[a, b]$, и **false** в противном случае.

Тип результата — логический.

Примеры см. в файле `zip://examples.zip/sobolev.mws`.

```
bvp_FinSobolev := proc(x::list, n::nonegint)
```

При положительном n проверяет L_2 -функцию x на принадлежность Соболевскому пространству n раз дифференцируемых функций. Возвращает `true` в случае успешного прохождения теста функцией x и `false` в противном случае. При этом L_2 -функция вида `[l2_a, выражение, l2_b]` считается бесконечно непрерывно дифференцируемой.

В случае отсутствия второго аргумента порядок пространства Соболева берется из глобальной переменной `bvp_n`, значение которой установлено процедурой `bvp_order`.

Тип результата — логический.

При $n = 0$ процедура возвращает наибольшее возможное значение n , при котором L_2 -функция x принадлежит Соболевскому пространству порядка n , или ∞ .

Тип результата — неотрицательное целое или `infinity`.

Примеры см. в файле `zip://examples.zip/sobolev.mws`.



7.3 Функции Грина

bvp_KG := proc()

Возвращает функцию Грина

$$G(t, s) = \frac{1}{b-a} \cdot \begin{cases} (b-t)(s-a), & a < s < t < b, \\ (b-s)(t-a), & a < t < s < b, \end{cases}$$

стандартной двухточечной краевой задачи

$$\begin{aligned} \ddot{x} &= z, \\ x(a) &= x(b) = 0. \end{aligned}$$

Тип результата — ядро.

bvp_KC := proc()

Возвращает функцию Грина (Коши)

$$C(t, s) = \begin{cases} \frac{(t-s)^{n-1}}{(n-1)!}, & a \leq s \leq t \leq b, \\ 0, & a \leq t < s \leq b, \end{cases}$$

стандартной начальной задачи

$$\begin{aligned} x^{(n)} &= z, \\ x(a) &= \dots = x^{(n-1)}(a) = 0. \end{aligned}$$

Порядок n устанавливается процедурой **bvp_order**.

Тип результата — ядро.

bvp_FKgreen := proc(A::matrix, Psi::list)

Вычисляет функцию Грина для краевой задачи

$$\begin{aligned} x^{(n)} &= z, & z &\in \mathbf{L}_2(a, b), \\ \ell^i x &= 0, & i &= 1, 2, \dots, n. \end{aligned}$$

Порядок n устанавливается процедурой **bvp_order**.

Эта процедура и следующий, связанный с ней, подпараграф обязаны своим появлением идее Е. И. Бравого. Использованные расчетные формулы опубликованы в доказательстве леммы 1 в статье [1].

Подготовка исходных данных для этой процедуры:

1) Линейные функционалы $\ell^i : \mathbf{H}^n[a, b] \rightarrow \mathbb{R}$ записать в виде

$$\ell^i x = \sum_{j=1}^n a_j^i x^{(j-1)}(a) + \int_a^b \psi^i(t) x^{(n)}(t) dt.$$

2) После чего следует коэффициенты a_j^i записать в квадратную матрицу **A** (i — номер строки, j — номер столбца), а из \mathbf{L}_2 -функций ψ^i составить список **Psi**.

В следующем подпараграфе описаны процедуры, помогающие в создании матрицы **A** и списка **Psi**.

Если краевая задача не является однозначно разрешимой, процедура генерирует ошибку, с соответствующим сообщением о ней.

Тип результата — ядро.

Исходный текст этой процедуры и примеры использования см. в файле `zip://green.zip/green.mws`.

Примечание. Пусть

G := `bvp_FKgreen(A, Psi)`:

т.е. мы получили функцию Грина **G**. Тогда командой

x := `l2_KFapplyOp(G, f)`:

мы находим решение рассмотренной краевой задачи.

bvp_Fpoly := `proc(A::matrix, b::vector)`

Вычисляет многочлен степени $n - 1$, являющийся решением краевой задачи

$$\begin{aligned} x^{(n)} &= 0, \\ \ell^i x &= \beta^i, \quad i = 1, 2, \dots, n. \end{aligned} \tag{2}$$

Порядок n краевой задачи устанавливается процедурой `bvp_order`.

Параметры этой процедуры:

квадратная матрица **A** — та же, что и для предыдущей процедуры `bvp_FKgreen`;

вектор **b** составляется из правых частей краевых условий β^i , $i = 1, 2, \dots, n$.

Если задача не является однозначно разрешимой, процедура генерирует ошибку, с соответствующим сообщением о ней.

Тип результата — \mathbf{L}_2 -функция.

Исходный текст этой процедуры и примеры использования см. в файле `zip://green.zip/green.mws`.

Примечание. Пусть L_2 -функция x найдена в предыдущем примечании, а

$y := \text{bvp_Fpoly}(A, b):$

т.е. мы получили решение y рассмотренной краевой задачи. Тогда командой

$x := \text{l2_Fsum}(x, y):$

мы находим решение полной краевой задачи

$$\begin{aligned}x^{(n)} &= z, & z &\in L_2(a, b), \\ \ell^i x &= \beta^i, & i &= 1, 2, \dots, n.\end{aligned}$$



7.4 Подготовка данных для процедур `bvp_FKgreen` и `bvp_Fpoly`

Первый шаг подготовки данных состоит в записи линейного непрерывного функционала $\ell : \mathbf{H}^n[a, b] \rightarrow \mathbb{R}$ в виде

$$\ell x = \sum_{j=1}^n a_j x^{(j-1)}(a) + \int_a^b \psi(t) x^{(n)}(t) dt.$$

Каждая из функций `bvp_FvalPrepData` и `bvp_FintPrepData` возвращает пару, состоящую из списка $\mathbf{a} = [a_1, \dots, a_n]$ и \mathbf{L}_2 -функции `psi`, для одного из двух примитивных функционалов.

Линейные комбинации примитивных функционалов вычисляются функцией `bvp_FcombPrepData` — это второй шаг подготовки данных.

Третий шаг заключается в формировании матрицы \mathbf{A} и списка `Psi` из данных, подготовленных для каждого из функционалов, с помощью функции `bvp_FgatherPrepData`.

```
bvp_FvalPrepData := proc(k::nonnegint, c::rational)
```

Подготовка данных для функционала

$$\ell x = x^{(k)}(c),$$

где $c \in [a, b]$, $0 \leq k < n$. В случае $k \geq n$ генерируется ошибка.

Значение — пара `[a, psi]`.

Примеры см. в файле `zip://green.zip/green.mws`.

```
bvp_FintPrepData := proc(k::nonnegint, phi::list)
```

Подготовка данных для функционала

$$\ell x = \int_a^b \varphi(t) x^{(k)}(t) dt,$$

где φ — \mathbf{L}_2 -функция, $0 \leq k \leq n$. В случае $k > n$ генерируется ошибка.

Значение — пара `[a, psi]`.

Пример см. в файле `zip://green.zip/green.mws`.

```
bvp_FcombPrepData := proc(c_1,p_1::list, c_2,p_2::list,...  
                          c_m,p_m::list)
```

Вычисляет линейную комбинацию пар p_1, p_2, \dots, p_m вида $[a, \psi]$ с коэффициентами c_1, c_2, \dots, c_m соответственно.

Значение — пара $[a, \psi]$.

Коэффициенты могут быть любыми выражениями — на ответственность пользователя. Количество пар параметров произвольно.

```
bvp_FgatherPrepData := proc(p_1::list, p_2::list,... p_n::list)
```

Составляет из пар p_1, p_2, \dots, p_n вида $[a, \psi]$ матрицу коэффициентов A и список L_2 -функций Ψ .

Значение — пара $[A, \Psi]$.

Количество параметров должно совпадать с порядком системы, установленным функцией `bvp_order`. В противном случае генерируется ошибка.

Примеры — в файле `zip://green.zip/green.mws`.



8 Библиотека `l2m_lib`

Эта библиотека реализует те же операции, что и библиотека `l2_lib`. Буква `m` в имени означает, что исходные данные и результаты работы процедур представляют собой матрицы, элементы которых являются L_2 -функции или ядра.

Таким образом, эта библиотека позволяет работать в пространствах вектор-функций.

Библиотека `l2m_lib` не предполагает возможности работы с данными типа `vector`. Вместо этого предлагается создавать и использовать векторы-столбцы:

```
col := matrix(n, 1):
```

или, что то же,

```
col := l2m_col(n):
```

— и векторы-строки:

```
row := matrix(1, m):
```

или, что то же,

```
row := l2m_row(m):
```

Также скалярные данные при использовании библиотеки следует представлять в виде 1×1 -матрицы:

```
scal := matrix(1, 1):  
scal[1, 1] := x:
```

или, что то же,

```
scal := l2m_toMatrix(x):
```

Это позволяет, например, использовать процедуру `l2m_KFApplyOp(K, x)`

- для $n \times t$ -матрицы `K` и t -вектора-столбца `x`;
- для $n \times t$ -матрицы `K` и $t \times k$ -матрицы `x`;
- для t -вектора-строки `K` и $t \times k$ -матрицы `x`;
- для скаляра `K` и t -вектора-столбца `x`;
- и т.д.

Для инициализации задачи следует воспользоваться процедурой `l2_init(...)`:

```
l2_lib[l2_init](...):
```

или

```
with(l2_lib, [l2_init]):  
l2_init(...):
```

Я предложил своему студенту М. Половинкину написать несколько процедур для этой библиотеки. Те из его процедур, которые без существенных изменений были действительно внесены в состав библиотеки, помечены его © в сносках.

8.1 Служебные процедуры

```
l2m_col := proc(n:: nonnegint)
```

Создает незаполненный вектор-столбец высотой **n** (и шириной 1).

Тип значения — `matrix`.

Примеры — в файле `zip://examples.zip/mprojector.mws`.

```
l2m_row := proc(m:: nonnegint)
```

Создает незаполненную вектор-строку длиной **m** (и высотой 1).

Тип значения — `matrix`.

```
l2m_toMatrix := proc(x)
```

Преобразует скалярное данное **x** в 1×1 -матрицу. Заполняет ее единственный элемент значением **x**.

Тип значения — `matrix`.

Примеры — в файле `zip://examples.zip/mprojector.mws`.



8.2 Процедуры для работы с функциями

8.2.1 Операции гильбертова пространства

`l2m_Fsum := proc(x::matrix, y::matrix)`

Вычисляет сумму двух матриц L_2 -функций x и y .

Тип результата — `matrix` L_2 -функций.

`l2m_EFmult := proc(f::matrix, x::matrix)`

Вычисляет произведение матрицы из выражений f и матрицы L_2 -функций x .

Если матрица f имеет размерность 1×1 и состоит из константы, то мы имеем операцию умножения вектора на скаляр.

Тип результата — `matrix` L_2 -функций.

`l2m_EFcomb := proc(c_1::matrix, x_1::matrix,
 c_2::matrix, x_2::matrix, ...
 c_n::matrix, x_n::matrix)`

Вычисляет линейную комбинацию матричных L_2 -функций x_1, x_2, \dots, x_n с матричными коэффициентами c_1, c_2, \dots, c_n соответственно:

$$\sum_{i=1}^n c_i(t) \cdot x_i(t).$$

Тип результата — `matrix` L_2 -функций.

Элементы матриц c_i могут быть любыми выражениями — на ответственность пользователя. Количество пар параметров произвольно.

`l2m_FinnerProd := proc(x::matrix, y::matrix)`

Вычисляет скалярное произведение двух матриц L_2 -функций x и y :

$$\sum_{i=1}^n \sum_{j=1}^m \int_a^b x_{ij}(t) \bar{y}_{ij}(t) dt.$$

Тип результата — число.

Примеры — в файле `zip://examples.zip/mprojector.mws`.

`l2m_Fnorm2 := proc(x::matrix)`

Вычисляет квадрат нормы матрицы L_2 -функций x :

$$\sum_{i=1}^n \sum_{j=1}^m \int_a^b |x_{ij}(t)|^2 dt.$$

Тип результата — число.

8.2.2 Функция + ... + функция → функция

```
l2m_Fmult := proc(x::matrix, y::matrix)
```

Вычисляет произведение матриц L_2 -функций x и y .

Тип результата — `matrix` L_2 -функций.

8.2.3 Функция → функция

```
l2m_Fconjugate := proc(x::matrix)2
```

Для матричной L_2 -функции x вычисляет комплексно сопряженную матричную L_2 -функцию. Если, конечно, `l2_complex = true`. Иначе возвращает ту же матрицу-функцию x .

Тип результата — `matrix` L_2 -функций.

8.2.4 Функция + ... + функция → числовая матрица

```
l2m_FmatrixGram := proc(phi_1::matrix,  
                        phi_2::matrix,  
                        ...,  
                        phi_n::matrix)2
```

Вычисляет матрицу Грама матричных L_2 -функций `phi_1`, `phi_2`, ..., `phi_n`:

$$G = (\langle \varphi_i, \varphi_j \rangle)_{i,j=1}^n.$$

Тип результата — квадратная матрица.

Примеры — в файле `zip://examples.zip/mprojector.mws`.



²© 2009, М. Половинкин

8.3 Процедуры для работы с интегральными операторами

8.3.1 Ядро + ядро \rightarrow ядро

`l2m_Ksum := proc(K_1::matrix, K_2::matrix)`

Вычисляет сумму матриц ядер K_1 и K_2 .

Тип результата — `matrix` ядер.

`l2m_Kmult := proc(K_1::matrix, K_2::matrix)`

Вычисляет произведение матриц ядер K_1 и K_2 .

Тип результата — `matrix` ядер.

`l2m_Ksuperpos := proc(K_1::matrix, K_2::matrix)`

Вычисляет матричное ядро оператора $K = K_1 \cdot K_2$, где операторы K_1 и K_2 порождены матричными ядрами K_1 и K_2 .

Тип результата — `matrix` ядер.

8.3.2 Ядро + функция \rightarrow функция

`l2m_KFapplyOp := proc(K::matrix, x::matrix)`

Вычисляет значение матричного интегрального оператора K на матричной L_2 -функции x .

Тип результата — `matrix` L_2 -функций.

Примеры — в файле `zip://examples.zip/mprojector.mws`.

8.3.3 Функция + ядро + ... + функция + ядро \rightarrow ядро

`l2m_EKmult := proc(f::matrix, K::matrix)3`

Вычисляет произведение матрицы из выражений f и матрицы-ядра K .

Тип результата — `matrix` ядер.

`l2m_KEmult := proc(f::matrix, K::matrix)`

Вычисляет произведение матрицы-ядра K и матрицы из выражений f (в указанном порядке).

Тип результата — `matrix` ядер.

³ © 2009, М. Половинкин

```
l2m_EKcomb := proc(c_1::matrix,K_1::matrix,
                  c_2::matrix,K_2::matrix,...
                  c_n::matrix,K_n::matrix)3
```

Вычисляет линейную комбинацию матриц-ядер K_1, K_2, \dots, K_n с матричными коэффициентами c_1, c_2, \dots, c_n соответственно:

$$\sum_{i=1}^n c_i(t, s) \cdot K_i(t, s).$$

Тип результата — **matrix** ядер.

Элементы матриц c_i могут быть любыми выражениями — на ответственность пользователя. Количество пар параметров произвольно.

Примеры — в файле `zip://examples.zip/mprojector.mws`.

8.3.4 Ядро → ядро

```
l2m_Kconjugate := proc(K::matrix)4
```

Для матрицы ядер K вычисляет комплексно сопряженную матрицу ядер. Если, конечно, `l2_complex = true`. Иначе возвращает ту же матрицу K .

Тип результата — **matrix** ядер.

8.3.5 Функция + ... + функция → ядро

```
l2m_FKmult0 := proc(phi::list,psi::list)
```

Вычисляет ядро

$$K(t, s) = \varphi(t)\psi(s),$$

порожденное матрицами L_2 -функций **phi** и **psi**.

Тип результата — **matrix** ядер.

Это вспомогательная процедура.

```
l2m_FKprojector := proc(phi_1::matrix,
                       phi_2::matrix,
                       ...,
                       phi_n::matrix)
```

Вычисляет матричное ядро $P(t, s)$ интегрального оператора — ортогонального проектора на линейную оболочку линейно независимой системы векторных L_2 -функций **phi_1, phi_2, ..., phi_n**:

$$P(t, s) = \sum_{i=1}^n \sum_{j=1}^n \gamma_{ij} \varphi_i(t) \overline{\varphi_j(s)}^T,$$

⁴ © 2009, М. Половинкин

где матрица $(\gamma_{ij})_{i,j=1}^n$ — обратная к матрице Грама $(\langle \varphi_i, \varphi_j \rangle)_{i,j=1}^n$, операция T превращает вектор-столбец в строку.

Вектор-функции **phi_i** представляются матрицами с одинаковым числом строк и 1 столбцом.

Если система линейно зависима, генерируется ошибка.

Тип результата — **matrix** ядер.

Примеры — в файле `zip://examples.zip/mprojector.mws`.

Contents ▲

9 Лицензия

Настоящая лицензия является соглашением между автором пакета **L₂** (далее „пакет“) и Вами, его „Пользователем“. Если Вас не устраивают условия этого соглашения, просто прекратите пользоваться этим пакетом и уничтожьте все имеющиеся в Вашем распоряжении копии файлов пакета.

Авторские права (Copyright) на пакет **L₂** принадлежат его автору, В.З. Цалюку, aka VTs.

Права Пользователя. Любому физическому или юридическому лицу, получившему пакет законным способом, бесплатно предоставляется неисключительное право использования, неограниченного копирования и распространения пакета при соблюдении следующих условий:

- Вы признаете приоритет автора пакета, распространяющийся на научные результаты, имеющие научную новизну, и тексты программ, имеющиеся в составе архивов примеров, входящих в состав пакета, если относительно них там нет указаний на авторство других лиц;
- при публикации любым способом (включая составление отчетов) результатов научных и/или проектных исследований, полученных с использованием пакета или его частей, факт использования пакета и авторские права на него упоминаются в публикации с указанием источника (сайта пакета);
- передача копий пакета другим лицам производится бесплатно и в полном составе, указанном в п. 1.2 настоящего документа, включая указание авторских прав и настоящую лицензию. При этом получатель (копии) пакета принимает на себя все права и обязанности Пользователя, обусловленные настоящим соглашением;
- Вы не имеете права на вскрытие технологии, декомпиляцию и деассемблирование пакета.

Ограничение ответственности. Пакет передается Пользователю „как есть“ („as is“), без каких-либо гарантий, выраженных явно или подразумеваемых, включая (но не ограничиваясь ими) подразумеваемые гарантии удовлетворительного качества и применимости для конкретной цели.

Автор пакета ни в коем случае не несет ответственности (материальной, уголовной или любой другой) за какие-либо убытки и/или ущерб (в том числе, убытки в связи с недополученной коммерческой прибылью, прерыванием коммерческой или производственной деятельности, утратой деловой информации, судебные издержки и иной имущественный ущерб), возникающие в связи с использованием или невозможностью использования пакета.

10 Переименования

Если Вы ранее решали задачи с использованием пакета и хотите, чтобы Ваши программы работали с новой версией пакета, то Вам следует проверить, использовались ли процедуры, список которых приведен ниже, и в случае необходимости исправить их имена.

Имя процедуры	заменить на
12_Fcomb	12_EFcomb
12_Kcomb	12_EKcomb
12_FmultFunc	12_EFmult
12_KmultFunc	12_EKmult
12m_Fcomb	12m_EFcomb
12m_Kcomb	12m_EKcomb
12m_FmultFunc	12m_EFmult
12m_KmultFunc	12m_EKmult

Список литературы

- [1] **Azbelev, N. V., Tsalyuk, V. Z.** *Application of Green's operator to quadratic variational problems.* OPUSCULA MATHEMATICA, 2006. V. 26, No 2. Pp. 243–256.
- [2] **Азбелев Н. В., Култышев С. Ю., Цалюк В. З.** ФУНКЦИОНАЛЬНО-ДИФФЕРЕНЦИАЛЬНЫЕ УРАВНЕНИЯ И ВАРИАЦИОННЫЕ ЗАДАЧИ. Москва–Ижевск: НИЦ «Регулярная и хаотическая динамика». 2006. 122 с. (<http://www.rcd.ru>).
- [3] **Азбелев Н. В., Максимов В. П., Рахматуллина Л. Ф.** ЭЛЕМЕНТЫ СОВРЕМЕННОЙ ТЕОРИИ ФУНКЦИОНАЛЬНО-ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ. МЕТОДЫ И ПРИЛОЖЕНИЯ. М.: Институт компьютерных исследований, 2002. 384 с.
- [4] **Алексеев В. М., Тихомиров В. М., Фомин С. В.** ОПТИМАЛЬНОЕ УПРАВЛЕНИЕ. М.: Наука, 1979. 432 с.
- [5] **Дьяконов В. П.** МАТЕМАТИЧЕСКАЯ СИСТЕМА MAPLE V R3/R4/R5. М.: „СОЛОН“. 1998. 400 с.
- [6] **Данфорд Н., Шварц Дж. Т.** ЛИНЕЙНЫЕ ОПЕРАТОРЫ. ЧАСТЬ 1. ОБЩАЯ ТЕОРИЯ.