УДК 681.142.2

П.Н. Палухин, В.В. Поддубный

ТЕХНОЛОГИЯ ИСПОЛЬЗОВАНИЯ MATLAB-ПРОГРАММ В СРЕДАХ ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ С/С++

Рассматривается технология использования программ, написанных на языке математического пакета MATLAB, в средах визуального программирования C/C++ с целью создания Windows-приложений, требующих привлечения богатого арсенала средств, алгоритмов, процедур и функций современной математики и обладающих развитым интерфейсом, на примере сред программирования MATLAB 6.5 и Borland C++Builder 6.0. В основу технологии положено использование встроенного в систему MATLAB компилятора m-кода в С-код MATLAB Compiler, предназначенного для создания исполнимых модулей программ, написанных на языке MATLAB. Обсуждаются особенности написания m-кода программ на языке MATLAB, пригодного для компиляции в С-код, и применения С-кода, создаваемого компилятором MATLAB Compiler, в визуальных средах программирования Microsoft Visual C/C++ и Borland C++Builder. Изложение иллюстрируется примером.

ВВЕДЕНИЕ И ПОСТАНОВКА ЗАДАЧИ

В практике программирования во многих случаях возникает необходимость создать программное обеспечение, решающее сложную математическую задачу, требующую использования математических процедур, не входящих в набор стандартных средств систем программирования. При решении этой задачи можно пойти разными путями. Либо воспользоваться средствами, предоставляемыми каким-нибудь математическим пакетом, например пакетом MATLAB, и решить математическую задачу, не создавая никакого специального программного обеспечения, написав код программы на языке пакета (MATLAB решает задачи в режиме интерпретации m-кода программы). Либо самому писать весь программный код, включая соответствующие математические процедуры, на каком-либо языке программирования (например, на C/C++). Либо искать (скажем, в Интернете) соответствующую математическую библиотеку, совместимую с данной средой программирования (при этом часто неизвестно, корректно ли она работает, каковы правила ее использования и обладает ли она достаточными возможностями).

Для случая, когда основным требованием является создание автономного приложения с помощью стандартных средств разработки на языке C/C++, для решения исходной задачи имеется еще один путь – воспользоваться дополнительными возможностями, предоставляемыми вышеупомянутым пакетом MATLAB. Этот пакет имеет в своем составе мощную математическую библиотеку MATLAB C/C++ Math Library [1, 2]. Вдобавок, MATLAB является довольно распространенным средством среди специалистов, и алгоритм, который необходимо реализовать в программном комплексе, часто можно построить с помощью имеющихся в системе MATLAB m-файлов или написать такие файлы самому. В этом случае потребуется перевести m-код на язык Си при помощи специальной утилиты пакета – MATLAB Compiler [3]. Полученный С-код, содержащий вызов С-функций математической библиотеки MATLAB, затем используется при создании исполнимого файла решения задачи. Однако для создания полноценного C/C++ приложения этого недостаточно. Система MATLAB создает лишь автономное консольное приложение, пригодное для использования, когда приложению не требуется хорошо организованного ввода и вывода данных. Система MATLAB не имеет развитых средств создания пользовательского интерфейса, какими обладают системы визуального программирования (Microsoft Visual C++, Borland C++Builder и др.). Поэтому создаваемое системой MATLAB автономное приложение не отвечает требованиям профессионального программирования.

Было бы заманчиво объединить широкую гамму возможностей, предоставляемых современными визуальными средствами создания приложений на языке C/C++, с мощными математическими возможностями системы MATLAB. Использование эффективного математического кода MATLAB совместно с возможностями визуальных сред разработки приложений позволило бы программисту-математику решить практически любую прикладную задачу. Пользователи системы MATLAB получили бы инструменты создания полноценного Windows-интерфейса, построения баз данных и т.д. Пользователи визуальных сред программирования Microsoft Visual C/C++ и Borland C++Builder стали бы обладателями математической мощи системы MATLAB, которая профессионально создавалась в течение многих лет (и продолжает развиваться) трудом большого коллектива математиков и программистов фирмы MathWorks, Inc.

Данная статья посвящена рассмотрению некоторых узловых вопросов технологии переноса программ, написанных на языке MATLAB, в визуальную среду программирования C/C++.

Предлагаемую технологию довольно трудно изложить в общем виде, поэтому практическую составляющую технологии целесообразно проиллюстрировать на небольшом конструктивном примере. Пример был выбран с тем расчетом, чтобы как можно проще пояснить технологические этапы использования кода, создаваемого системой MATLAB, не исключая и графические возможности этой системы, поскольку визуальное представление результата работы программы является зачастую неотъемлемой частью создаваемого программного комплекса. Необходимую информацию об особенностях трансляции графических функций MATLAB можно получить из [4]. При этом мы всюду делаем упор на минимальное изменение кода, генерируемого компилятором MATLAB.

Рассмотрим последовательно все основные технологические этапы создания Windows-приложения в визуальной среде программирования C/C++ (конкретно, в среде Borland C++Builder 6.0) с использованием математического обеспечения, предоставляемого средой программирования математического пакета MATLAB 6.5, на примере решения задачи исследования эволюции во времени численности популяций в системе «хищник–жертва», описываемой модифицированной моделью Лотки – Вольтерры [5]. Система

модифицированных нелинейных дифференциальных уравнений Лотки–Вольтерры, учитывающих наличие внутривидовой борьбы в популяции жертв, связанной с нехваткой пищи при превышении численностью этой популяции некоторого критического уровня, имеет вид

$$\begin{cases} \dfrac{dy_1(t)}{dt} = \left(a - by_2(t) - ry_1(t)\right) y_1(t), \\ \dfrac{dy_2(t)}{dt} = \left(-c + dy_1(t)\right) y_2(t), \end{cases} \qquad (1)$$

где $t \in [t_0, t_1]$ – текущее время, изменяющееся на указанном интервале решения задачи; $y_1(t)$, $y_2(t)$ – текущие численности популяций жертв и хищников соответственно; $y(t_0) = y_0$ – заданное начальное условие для вектора-столбца $y(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = [y_1(t), y_2(t)]^T$ численности популяций; T – знак транспонирования. Параметры a, b, c, d, r предполагаются известными постоянными. Смысл их следующий. Параметр a выражает скорость естественного прироста популяции жертв в единицу времени в расчете на одну жертву в отсутствие хищников. Параметр c – скорость естественной гибели (от голода) популяции хищников в единицу времени в расчете на одного хищника в отсутствие жертв. Коэффициенты b и d выражают соответственно влияние на скорости роста – гибели каждой популяции наличия другой популяции. Коэффициент r выражает дополнительное уменьшение скорости роста популяции жертв из-за размножения самих жертв. Величина, обратная этому коэффициенту, выражает количество жертв, которое может прокормить природа в отсутствие хищников.

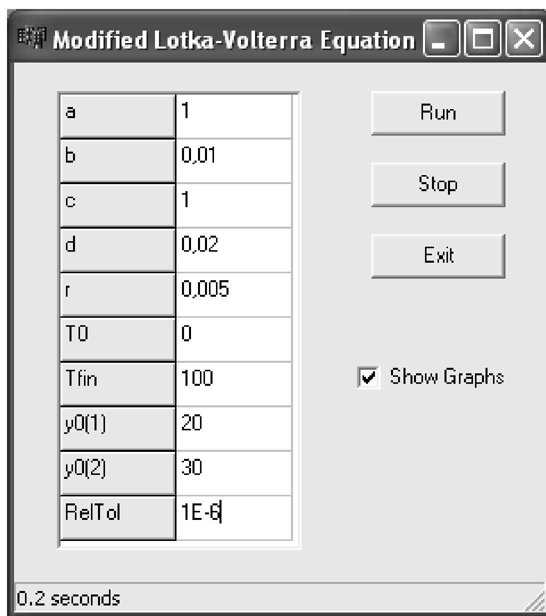

Рис. 1. Общий вид формы примера приложения

Программная реализация рассматриваемого примера приложения должна давать пользователю возможность в интерактивном режиме менять все исходные данные задачи (параметры a, b, c, d, r, начальный $t_0 = T0$ и финальный $t_1 = T\text{fin}$ моменты времени исследования, начальное значение численности популяции жертв $y_1(t_0) = y0(1)$ и хищников $y_2(t_0) = y0(2)$, допустимую относительную погрешность RelTol численного интегрирования системы дифференциальных уравнений). Вид возможной формы интерфейса этого приложения приведен на рис.1.

На рис.2 и 3 приведены графические результаты работы приложения при значениях параметров, заданных на форме рис.1.

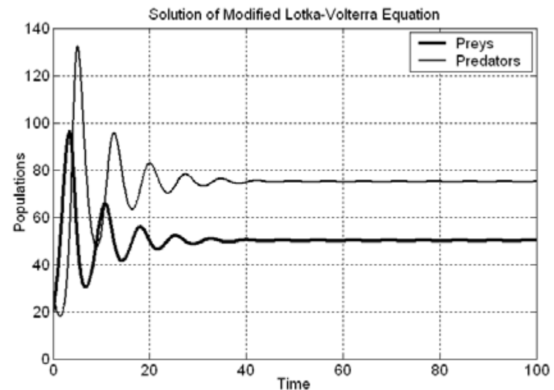

Рис. 2. Интегральные кривые решения

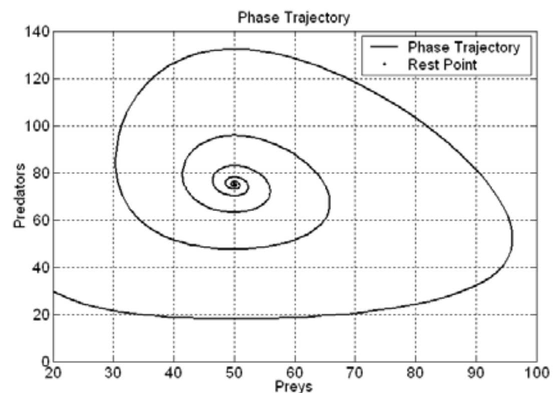

Рис. 3. Фазовая траектория

Для численного интегрирования системы дифференциальных уравнений (1) использован метод Рунге–Кутты 4 – 5 порядка точности. На графиках видно, как численности взаимодействующих популяций жертв и хищников приближаются со временем к равновесным значениям $y_1^* = c/d$ и $y_2^* = \left(a - rc/d\right)/b$ (к точке покоя на фазовой траектории).

ЭТАПЫ СОЗДАНИЯ ИСПОЛНИМОГО ПРИЛОЖЕНИЯ

Рассмотрим основные этапы создания исполнимого модуля рассматриваемого приложения, иллюстрируя тем самым технологию создания приложений, использующих богатые математические возможности пакета MATLAB и средства визуальной среды программирования C/C++.

Этап 1. Создание исходного m-кода

Наличие данного этапа может показаться необязательным, так как исходный код можно писать сразу на языке Си, используя функции математической библиотеки MATLAB [1, 2]. Стоит сразу оговорить,

что данный путь не оптимален по нескольким причинам. Во-первых, создавая приложение таким образом, пользователь теряет возможность контекстной отладки кода. Во-вторых, построенный код может получиться неустойчивым в плане исполнения. В-третьих, даже если создается чисто Си-приложение, содержащее небольшие участки сложных математических вычислений, изначально проще их предварительно написать в MATLAB в виде функций, затем перевести полученный m-код в C-код с помощью компилятора MATLAB, полученный код впоследствии легко подключить в виде модуля к приложению на C\C++.

Итак, создадим m-файл функционального типа:

```
% Файл "lve.m"
function main,
% Программа решения системы
% дифференциальных уравнений
% Лотки–Вольтерры
global a b c d r; % Глобальные переменные
tic, % Фиксация времени начала решения
a = 1; b = 0.01; c = 1; d = 0.02; r = 0.005;
% Задание параметров системы
T0 = 0; Tfin = 100; % Границы интервала
y0 = [20; 30]; % Начальная численность
                    % популяций
yp = [c/d; (a–r*c/d)/b]; % Точка покоя
opt = odeset('RelTol', 1e–6); % Установка
% относительной точности решения
tt = []; yy = []; % Объявление массивов
[tt, yy] = ode45(@lv, [T0, Tfin], y0, opt);
tt = tt'; yy = yy'; % Интегрирование системы
% дифференциальных уравнений Лотки–
% Вольтерры методом Рунге–Кутты 4–5
% порядка точности с заданной точностью
time = toc; disp(time) % Фиксация времени
% конца решения задачи и вывод времени счета
% на экран
figure(1) % Открытие 1-го окна графиков
plot(tt, yy(1, :), 'g', tt, yy(2, :), 'r'), grid on,
% Изображение графиков и координатной сетки
title('Solution of Modified Lotka–Volterra Equation'),
% Заголовок графика
legend('Preys', 'Predators'), % Легенды графиков
xlabel('Time'), ylabel('Populations'),
% Обозначение осей координат
figure(2) % Открытие 2-го окна графиков
plot(yy(1,:), yy(2,:), 'b', yp(1), yp(2), 'r.'), grid on,
% Изображение графиков и координатной сетки
title('Phase Trajectory'),
% Заголовок графика
legend('Phase Trajectory', 'Rest Point'),
% Легенды графиков
xlabel('Preys'), ylabel('Predators')
% Обозначение осей координат
% ---------------------------------------------------------
function z = lv(t, y)
% Вектор-функция, выражающая правую часть
% системы дифференциальных уравнений
% Лотки–Вольтерры
global a b c d r; % Глобальные переменные
z=[(a–b*y(2)–r*y(1))*y(1); (–c+d*y(1))*y(2)];
% Возвращаемый функцией вектор-столбец
```

Эта программа с помощью функции ode45 (метода Рунге – Кутты 4 – 5 порядка) численно решает модифицированную систему уравнений Лотки–Вольтерры (1), которая описывает поведение популяций жертв (Preys) и хищников (Predators). Затем с помощью функции plot строятся графики решения этих уравнений в зависимости от времени (интегральные кривые решения) и фазовая траектория решения (зависимость численности популяции хищников от численности популяции жертв). На графики наносятся координатные сетки командой grid, выводятся заголовки с помощью функции title, а также метки кривых при помощи функции legend. Оси графиков подписываются с помощью функций xlabel и ylabel.

После создания исходного m-кода можно перейти к следующему этапу.

Этап 2. Перевод m-кода в C-код с помощью компилятора MATLAB Compiler

Технология перевода программ из среды MATLAB в C-код с созданием исполняемого модуля консольного приложения полностью изложена в [3]. При этом m-код программы должен быть определенным образом подготовлен (модифицирован).

В первую очередь необходимо преобразовать m-файлы сценарного типа (script-файлы) в m-файлы функционального типа (function-файлы). Как это делается, описано в [3].

Далее следует преобразовать циклические операции с массивами из формы, представленной слева на рис.4, в форму, представленную справа:

```
for  i = 1 : N,                    for  i = 1 : N,
   M = [M, <значение>];               M(i) = <значение>;
end                                end
```

Рис.4. Преобразование операций с массивами

Такую же операцию необходимо применить и к многомерным массивам. Размер кода во втором случае, возможно, немного и увеличится, но скорость выполнения цикла может увеличиться во много раз.

Для проверки эффективности такой замены был сконструирован простейший тест. Было создано два файла-функции, содержащие вышеописанные циклы, в теле которых находилась процедура вывода прогресса счета, и на выходе строился график зависимости времени выполнения программ от числа итераций цикла. Оказалось, что цикл второго («правого») типа гораздо более эффективен, чем цикл первого («левого»). Рост времени выполнения операций первого цикла имеет экспоненциальный характер в силу постоянного динамического увеличения памяти, отводимой под массив, тогда как время работы второго цикла растет линейно и очень медленно (например, для числа отсчетов 10000 имеем соотношение по времени тип1/тип2 как 0,02сек/6,72сек в тестируемой конфигурации).

На следующем шаге рекомендуется удалить все комментарии MATLAB на русском языке, так как некоторые из них вызывают ошибку компиляции. Это проблема русификации системы MATLAB, которая проявляется в некоторых версиях системы при некоторых версиях операционной системы. Например,

русская буква "я", встречающаяся как в комментариях, так и в функциях текстового вывода, может восприниматься в качестве символа конца строки. «Разорванная» строка, в свою очередь, вызывает ошибку компиляции. Данную проблему можно разрешить уже после перевода m-кода на язык Си (см. этап 4).

Рекомендуется также, во избежание ошибок, привести к единому регистру имена файлов, названия функций и наименования функций-параметров.

Не следует использовать переменную с именем "j" в алгоритмах, которые переводятся на C++, так как у класса MathWorks есть метод MathWorks::j(), и это вызывает ошибку компиляции.

Еще одно важное замечание. Команды title, xlabel, ylabel не являются библиотечными функциями MATLAB и хранятся лишь в виде m-файлов. Автоматически компилятор MATLAB не транслирует функции и процедуры, записанные во вспомогательные файлы, поэтому их тексты придется самим добавлять в файл "lve.m". Найти их можно в каталоге "$MATLAB\toolbox\matlab\graph2d", либо вызвать на экран, просто набрав команду

```
type title.m
```

в командном окне MATLAB.

После проделанных манипуляций m-файл "lve.m" готов к компиляции. Перед компиляцией командой

```
mbuild –setup
```

из командной строки устанавливается требуемый компилятор с языка C/C++. Для компиляции использовался следующий синтаксис команды

```
mcc –m –B sgl lve.m
```

из командной строки.

Получающийся в результате исполняемый модуль располагается в той же директории, где находится m-файл программы lve.m, и имеет имя lve.exe, т.е. имя m-файла, использованного для компиляции.

Этап 3. Присоединение С-кода, созданного компилятором MATLAB, к проекту в визуальной среде программирования

В качестве визуальной среды программирования был выбран Borland C++ Builder версии 6.0, который помимо удобного пользовательского интерфейса предоставляет мощный компилятор кода на C/C++.

Создадим проект в этой среде, содержащий форму, изображеную на рис.1, на которую поместим компонент StringGrid для ввода и редактирования исходных данных, кнопки пуска, остановки и выхода, а также чек-бокс вывода графики и строку статуса, выводящую процент выполнения задачи. Наша задача состоит в том, чтобы написать обработчики событий, связанных с этими органами управления, и использовать полученный на предыдущем этапе С-код. Но прежде чем присоединять к проекту этот С-код, необходимо настроить данную среду на совместное использование с библиотеками MATLAB C Math Library и MATLAB C Graphics Library. Исчерпывающую информацию по этому вопросу можно получить из файла
MATLAB$\extern\examples\cppmath\borland\readme.txt.

Следует заметить, что после указания путей к стандартным математическим библиотекам и подключаемым дополнительно c/cpp-файлам в опциях проекта, а также добавления к проекту соответствующих математических библиотек может оказаться, что проект не компонуется. Это означает, что не все необходимые библиотеки были подключены к проекту. Чтобы узнать, в каком из библиотечных файлов хранится «сбойная» функция, надо ознакомиться с содержимым файлов с расширением ".def", которые размещены в папке "MATLAB$\extern\include\".

Добавление С-кода в проект

На данном этапе постараемся произвести минимальную модификацию созданного MATLAB Compiler С-кода, который состоит из трех файлов: "lve.c", "lve.h", "lve_mainhg.c". Поскольку файл "lve_mainhg.c" отвечает за создание автономного консольного приложения, то он более не потребуется, однако содержащийся в нем код, за исключением функции main, нужно перенести в создаваемый проект, для удобства в ".h"-файл, дополнительно подключаемый к ".cpp"-файлу, реализующему методы формы (см. несколько ниже).

Теперь подключим к проекту файл "lve.c", произведя следующую небольшую модификацию кода.

```
// Файл "lve.c
#include "mhelper.h" /* Содержит (пока) лишь прототип функции WinFlush() */
< .. >
static void Mlve(void)
{ < .. >
mlfDisp(mclVv(time, "time"));
WinFlush(); /* Вручную добавленный вызов функции */
mclPrintAns(&ans, mlfNFigure(0, _mxarray0_, NULL));
< .. > }
< .. >
```

Хорошо видно, что изменения кода минимальны. Был лишь подключен файл, содержащий объявление функции Winflush, использовавшейся после mlfDisp для одновременного вывода времени работы программы на форму, полученное от Print Handler.

Print Handler – это пользовательский обработчик печати. Для используемого примера его можно сконструировать таким образом, чтобы данный обработчик было удобно использовать при выводе матриц и других составных объектов печати. Для полноценной же работы ему потребуется функция, которая осуществляла бы вывод накопленного текста на форму и вызывалась следом за выводящей библиотечной функцией (например, mlfPrintMatix, mlfDisp, mlfPrintf и т.д.). Способы создания, регистрация и использование пользовательского обработчика печати описываются в [3], дополнительную информацию по использованию можно найти в [2].

Доработка исходного С-кода

Понятно, что функция main, содержащая, в свою очередь, функцию mclMainhg, удалена (проект должен содержать либо main для консольных приложений, либо WinMain для Windows-приложений). Но вспомогательная функция компилятора MATLAB

mclMainhg несла большую смысловую нагрузку, которую как раз можно компенсировать вызовом функций компилятора MATLAB из модуля "Unit1.h" (см. ниже). Функция mclMainhg делает следующее:

- Инициализирует рабочие таблицы: таблицу глобальных переменных, глобальную и локальные таблицы файлов-функций, рабочие переменные модулей и специализированные библиотеки. Также удаляет из памяти рабочие переменные и завершает работу специализированных библиотек. Данное действие компенсируется "ручным" вызовом функции

```
mclLibInitCommon(&_main_info);
```

при нажатии кнопки "Run", где &_main_info – адрес структуры, содержащей все вышеперечисленные таблицы.

- Выполняет инициализацию графической библиотеки MATLAB. Данное действие можно осуществить самостоятельно, единожды вызвав библиотечную функцию mlfHGInitialize, что и было сделано при создании формы приложения.
- Вызывает через feval-интерфейс рабочий код (через функцию mlxLve). Аналогичное действие можно выполнить при нажатии "Run" только через стандартный интерфейс (функцию mlfLve).
- Завершает работу графической библиотеки MATLAB. Соответствующее действие с помощью функции mlfHGTerminate было осуществлено в файле "Unit1.cpp" в момент закрытия приложения.

```
// Файл "cpphelper.c"
< .. > /* Содержимое "Lve_mainhg.c", за исключением функции main. */
//----- Print Handler -----

// Файл "Unit1.cpp"
#include "cpphelper.h" /* Дополнительно подключаемый файл (см. выше) */
TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner): TForm(Owner)
{
 const char *argv = *_argv;
 mlfHGInitialize(&_argc, &argv); /* Инициализируем MATLAB C Graphics Library */
}
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 mlfSetPrintHandler(WinPrint); /* Регистрируем обработчик печати */
 mclLibInitCommon(&_main_info ); /* Осуществляем инициализацию таблиц компилятора */
 mlfLve(); //Имя исходной m-функции
}
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
 mlfHGTerminate(); /* Прерываем использование графической библиотеки */
}
```

Здесь по нажатию кнопки на форме производится запуск транслированного кода с помощью функций, полностью подменяющих функцию компилятора MATLAB mclMainhg. Следует заметить, что замена функции mclMainhg – рекомендуемое, но не обязательное действие. Вызов

```
mclMainhg(&_argc, &argv, mlxLve, 0, &_main_info);
```

произойдет вполне успешно, однако при повторном нажатии на кнопку приложение выдаст ошибку. Это связано с тем, что работа данной функции рассчитана на одиночный запуск из консольного приложения. При ее использовании также теряется удобная возможность отладки подключенного кода, что довольно неприятно. Поэтому настоятельно рекомендуем использовать описанный нами способ запуска C-кода.

Необходимо сделать еще одно важное замечание. В случае, если нет надобности использовать в приложении графику MATLAB, часть кода, касающуюся инициализации графической библиотеки, можно удалить из проекта.

Этап 4. Внесение в приложение новых функциональных возможностей

Название данного этапа довольно неопределенно, так как включает большое число технических приемов, касающихся взаимодействия между рабочими средами. В данный этап можно поместить непосредственную разработку интерфейса пользователя, который уже естественно привязан к назначению создаваемого приложения. В контексте данного этапа можно произвести расширение и переработку рассматриваемого примера, чтобы «пробросить мостик» между созданием иллюстративного примера и разработкой реального приложения.

Увеличим временной интервал вычислений в исходном m-коде, а внутри функции lv предусмотрим вывод на экран процента от общего времени вычислений в алгоритме. Расширив код подобным образом и проделав заново все вышеперечисленные технологические этапы, получим следующее: пока будут производиться вычисления, форма примера как бы «заснет». Она не станет отвечать ни на какие внешние или внутренние воздействия и лишь по истечении времени работы цикла опять «проснется», когда будет выведено, что вычисления произведены на 100%.

Причина данного явления состоит в том, что математические вычисления осуществляются в контексте главного потока VCL, а не независимо, как это должно быть. Данное явление не критично, если вычисления не занимают много времени, в противном случае необходимо знать прогресс вычислений и, возможно, досрочно их завершить.

Обращаем особое внимание, что при создании рассматриваемого примера использовалась следующая комбинация средств: MATLAB Compiler v3.0 и Borland C++Builder v6.0 (содержит компилятор Borland C++ v5.6).

Добавим на форму еще одну кнопку "Stop" и панель статуса (Status Bar), на которую будем отображать процент прогресса и общее время выполнения вычислений. По нажатию на кнопку "Run" будет создаваться новый поток класса TNewThread, реализующий в отдельном модуле абстрактный класс TThread, внутри которого уже будут производиться заданные вычисления. При использовании данного стандартного приема программирования возникает две особен-

ности, связанные с использованием функций математической библиотеки MATLAB:

1. Инициализация графической библиотеки MATLAB и глобальной таблицы функций должна осуществляться в модуле главной формы, в противном случае возникает ошибка. К сожалению, при использовании различных версий MATLAB Compiler и Borland C++Builder возникают существенные различия при инициализации и использовании графических функций внутри пользовательского потока и потока VCL. В данной статье описывается такое использование совместно лишь с последними версиями вышеупомянутых программных продуктов.

2. Вызов методов созданного потока нельзя осуществлять напрямую из C-модуля, так как объекты можно использовать лишь в C++. Однако вызов методов потока можно произвести с помощью дополнительных C-функций, которые объявляются с модификатором extern "C" в модуле, описывающем методы потока, где также и реализуются. Данная дополнительная функция вызывает метод потока, который, в свою очередь, вызывает с помощью метода Synchronize еще один метод, содержащий защищаемый код (пример см. ниже в файле "Unit2.cpp").

3. Настоятельно рекомендуется использовать концепцию «неизменяемого» m-кода при построении приложения.

«НЕИЗМЕНЯЕМЫЙ» m-КОД

«Неизменяемым» можно назвать такой m-код, который после трансляции на язык C/C++ нет необходимости редактировать и дополнять вызовом каких-либо C-функций. Определенный, стандартизованный набор C-функций просто напрямую вызывается из m-кода. Использование данной концепции, на самом деле, должно превалировать при создании автономных приложений в визуальной среде программирования.

Это связано с тем, что исходный m-код зачастую подвергается небольшой переработке на третьем технологическом этапе (где производится доработка C-кода, полученного трансляцией из m-кода). И после повторного прохождения этапов технологии необходимо заново осуществить доработку C-кода, что из довольно интересного процесса в первый момент превращается в неприятную, рутинную и, возможно, для кого-то нетривиальную операцию.

Разработчики MATLAB предусмотрели возможность сопряжения m-кода с C/C++-кодом. Начиная с версии 2.1, компилятор MATLAB поддерживает вызов произвольной C/C++-функции из m-кода. Достаточно просто предоставить m-файлу функцию-заглушку, определяющую, каким образом данный код будет работать в m-коде, а затем реализовать тело функции в C/C++.

С другой стороны, связь между вычислительным кодом и кодом, реализующим пользовательскую часть приложения, совершенно необходима, причем она выражается, как правило, определенным, в какой-то мере стандартным набором функциональных особенностей:

- присвоение значения конкретной C-переменной, конкретного поля объекта и т.п. переменной, используемой функциями библиотеки MATLAB;
- вывод текста (скорее всего «накопленного» обработчиком печати) на рабочую форму;
- досрочное прекращение вычислений.

Это лишь неполный перечень действий, который можно взять за основу и осуществлять вызов соответствующих этим действиям C-функций прямо из m-кода, первоначально не думая об их реализации, которая уже непосредственно привязана к разработке пользовательской части, хотя здесь возможно использование каких-либо шаблонов, облегчающих создание приложения для неискушенных пользователей данной технологии.

Чтобы не быть голословными, рассмотрим конкретный пример создания m-файла на основе описанной выше технологии. Рассмотрим ключевые фрагменты кода, описывающего использованные технические приемы.

```
% Файл "func.m"
function main
global a b c d r mMyExit Tfin oldpers,
tic;
% Блок инициализации
mShowGraph = AssignValue(0);
% Отображать ли график
mMyExit = 0; oldpers = 0;
a = 1; b = 0.01; c = 1; d = 0.02; r = 0.005;
T0 = 0; Tfin = 100;
y0 = [20; 30]; yp = [c/d; (a–r*c/d)/b];
opt = odeset('RelTol', 1e-6); tt = []; yy = [];
% Блок вычислений
% (сюда же можно отнести функцию lv)
[tt, yy] = ode45(@lv, [T0, Tfin], y0, opt);
tt = tt'; yy = yy';
time = toc;
% Блок вывода
if (mShowGraph)
% Если 1 – выводим график, 0 – пропускаем вывод
  figure(1)
  < .. >
end
fprintf(1, '%3.1f seconds', time);
SyncVCL(0); % Осуществить вывод на форму
                % в то же место
% Блок описания внешних C-функций
function CloseThread;
% Завершить выполнение потока
%#external
%-----------------------------------------------------------
function SyncVCL(MethodNumber);
% Синхронизировать с VCL вывод текста
% Здесь MethodNumber – определенный способ
%вывода текста
%#external
%-----------------------------------------------------------
function OutVar = AssignValue(VarNumber);
% Присвоить конкретное значение
% Здесь VarNumber – определенный способ
% присвоения значения
% Здесь OutNumber – выходное значение,
% приведенное к библиотечному типу mxArray
%#external
%-----------------------------------------------------------
```

```
function z = lv(t, y)
global a b c d r mMyExit Tfin oldpers,
mMyExit = AssignValue(1);
% Осуществить ли досрочное завершение
% вычислений
if (mMyExit)
  CloseThread; % Здесь работа потока прерывается
end;
pers = floor((t / Tfin) * 100);
% Вычисляем процент прогресса вычислений
if (pers > oldpers)
  fprintf(1, '%4.0f%% Complete', pers);
  SyncVCL(0); % Осуществить вывод на форму
  oldpers = pers;
end
z = [(a – b*y(2) – r*y(1))*y(1); (–c + d*y(1))*y(2)];
%------------------------------------------------------------
function hh = title(string, varargin)
```

Приведенный код мы рекомендуем в качестве шаблона, используемого для создания «быстро переводимых» m-файлов.

Все m-файлы будущего приложения следует объединить в один единственный, который можно назвать, скажем, "func.m", а основную часть, из которой вызываются вспомогательные функции, назвать, например, main.

Затем выделить в программе «блок инициализации», в котором необходимо присвоить первоначальные значения используемым переменным. Переменные, значения которых должен ввести пользователь, инициализировать с помощью функции AssignValue(), «заранее знающей», каким образом осуществить означенное присвоение.

В «блоке вычислений» следует произвести требуемые манипуляции, где, используя функцию SyncVCL(), можно выводить на форму текущие значения переменных, либо при помощи функции CloseThread досрочно прекратить процесс вычислений.

Завершать программу должен «блок вывода» результатов проделанных вычислений в графической или текстовой форме, где можно протестировать специальные переменные на предмет необходимости вывода конкретных составляющих.

И уже в самом конце нужно разместить описание внешних С-функций, внутри которых должна быть обязательно указана директива

%#external,

предписывающая компилятору использовать внешнюю реализующую версию соответствующей m-функции.

Далее применим те же технологические приемы для создания демонстрационного приложения на базе «неизменяемого» m-кода.

Уже на втором технологическом этапе, т.е. на этапе трансляции, возникают некоторые отличия, связанные с присутствием внешних нереализованных С-функций. Во-первых, компилятор создаст дополнительный заголовочный файл (будет называться "func_external.h"), где разместит прототипы реализующих версий внешних m-функций, а интерфейсная часть будет содержаться в главном файле ("func.c"). Во-вторых, компилятор не сможет создать исполняемый файл ввиду отсутствия реализующей части, что, в общем-то, и не нужно, так как «файл-оболочка» ("func_mainhg.c") все-таки будет создан. Однако не будет создана директория "bin" с содержимым, но это тоже не помеха – ее можно взять у любого другого приложения либо создать вручную, скопировав в нее файлы "FigureMenuBar.fig" и "FigureToolBar.fig", находящиеся в директории "$(MATLAB)\extern\include". К тому же данная директория нужна, если вызываются графические функции MATLAB, но даже и без нее программа будет нормально работать, просто не будет функционировать панель и меню графического окна MATLAB, а также библиотека будет выдавать предупреждение об отсутствии файлов ".fig".

К форме разбираемого примера добавим также CheckBox, регулирующий вывод графика после вычислений. Учитывая вышеописанные изменения, рассмотрим соответствующие изменения в проектных файлах.

Файл "func.c" просто подключаем к проекту, никоим образом не дополняя и не изменяя его. Затем, если вдруг понадобится внести какие-либо изменения в m-коде, достаточно лишь заново произвести трансляцию на язык Си и заменить старый "func.c" на его обновленную версию.

```
// Файл "Unit1.cpp"
#include "Unit1.h" /* Содержит описание класса TForm1 и внешнее объявление Form1 */
#include "Unit2.h" /* Содержит описание класса TNewThread и внешнее объявление MyThread – объекта, инкапсулирующего поток WinAPI */
#include "libmatlb.h" /* Содержит прототипы функций инициализации графики */
/* Глобальные переменные, управляющие выводом графика и досрочным завершением */
double MyCheck = 1, MyExit; /* дополнительного процесса */
//------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner): TForm(Owner)
{
const char *argv = *_argv;
/* При создании формы инициализируем графическую библиотеку MATLAB */
mlfHGInitialize(&_argc, &argv);
}
void __fastcall TForm1::Button1Click(TObject *Sender)
{
MyExit = 0;
Button1->Enabled = false; CheckBox1->Enabled = false;
// Создаем новый поток и сразу запускаем его
MyThread = new TNewThread(false);
}
void __fastcall TForm1::Button2Click(TObject *Sender)
{
MyExit = 1;
}
void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
MyCheck = CheckBox1->Checked;
}
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
```

```
{ /* Во время завершения работы приложения выгружаем графическую библиотеку */
mlfHGTerminate();
}

// Файл "Unit2.cpp"
#include "cpph.h" /* Включает содержимое "func_mainhg.h" до функции main и реализацию функции PrintHandler */
#include "func_external.h" /* Включает прототипы специальных С-функций */
extern double MyCheck, MyExit;
//---------------------------------------------------------------
__fastcall TNewThread::TnewThread (bool
CreateSuspended): TThread(CreateSuspended)
{ /* Устанавливаем, чтобы поток автоматически уничтожился после завершения */
FreeOnTerminate = true;
}
void __fastcall TNewThread::Execute() /* "Тело" потока */
{
mclLibInitCommon(&_main_info ); /* Инициализируем таблицы компилятора */
mlfSetPrintHandler(WinPrint); /* Регистрируем обработчик печати */
mlfFunc(); /* Вызываем основную функцию, находящуюся в "func.c" */
/* Ожидаем, пока пользователь не закроет все графические окна, это является необходимым действием, а иначе завершающийся поток их закроет сам. Функция pause не помогает в данной ситуации */
mlfHGWaitForFiguresToDie();
/* Разрешаем использование ранее отключенных кнопки "Run" и CheckBox */
UpdateMethod(1);
}
void __fastcall TNewThread::UpdateMethod(int mode)
{ /* Вызов методов потока, содержащих код, защищаемый от конфликтов доступа к компонентам VCL (thread-safe code) */
switch (mode)
{
case 0: Synchronize(UpdateForm); break;
case 1: Synchronize(EnableButton);
}
}
void __fastcall TNewThread::UpdateForm()
{ /* Выводим содержимое буфера Print Handler в строку статуса */
Form1–>StatusBar1->SimpleText = OutputBuffer;
FreeOutput();
}
void __fastcall TNewThread::EnableButton()
{ // Разрешаем использование кнопки и CheckBox
Form1 –> Button1 –> Enabled = true;
Form1 –> CheckBox1 –> Enabled = true;
}
void Mfunc_CloseThread(void)
{ /* "Тело" реализующей версии m-функции CloseThread */
MyThread –> UpdateMethod(1);
EndThread(0);
}
void Mfunc_SyncVCL(mxArray * MethodNumber)
{ /* "Тело" реализующей версии m-функции SyncVCL() */
MyThread –> UpdateMethod(*mxGetPr(MethodNumber));
}
extern mxArray * Mfunc_AssignValue(int nargout_, mxArray * VarNumber)
{/* "Тело" реализующей версии m-функции AssignValue */
mxArray *ReturnNumber = NULL;
int VarNum;
VarNum = (int)*mxGetPr(VarNumber); /* Получаем номер способа присваивания */
switch (VarNum)
{
case 0: ReturnNumber = mlfScalar(MyCheck); break;
case 1: ReturnNumber = mlfScalar(MyExit);
} /* Возвращаем значение типа mxArray, "понятного" функциям MATLAB C Math Lib */
return ReturnNumber;
}
```

Организацию ввода и редактирования исходных данных можно реализовать следующим образом. Добавим массив Mass чисел с плавающей запятой, в который будем записывать введенные пользователем параметры динамической системы. Затем этот массив передадим с помощью функции-заглушки AssignValue в аналогичный массив в m-коде. Фунция AssignValue реализуется таким образом, чтобы по селектору значение переменной определенного типа данных Си преобразовывалось в соответствующее значение переменной MATLAB, но было бы уже представлено типом, который понимает математическая библиотека MATLAB. В MATLAB Math Library имеются необходимые для этого функции: mlfScalar – для скалярных значений, mlfDoubleMatrix – для матриц, значениями которых являются числа с плавающей запятой.

Завершение работы приложения можно произвести кнопкой "Exit".

Соответствующие обработчики, созданные вручную, представлены в приводимом ниже файле Unit1.cpp.

```
// Файл "Unit1.cpp"
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include "Unit2.h"
#include "libmatlb.h"
//---------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
double MyCheck = 1, MyExit, Mass[10];
TNewThread *MyThread;
TForm1 *Form1;
//---------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
      : TForm(Owner)
{
  const char *argv = *_argv;
  mlfHGInitialize(&_argc,&argv);
}
```

```
//---------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 for (int i = 0; i < 10; i++)
   Mass[i] = StringGrid1->Cells[1][i].ToDouble();
//Переписываем значения параметров системы в
//в дополнительный массив
 MyExit = 0;
 Button1->Enabled = false;
 CheckBox1->Enabled = false;
 MyThread = new TNewThread(false);
}
//---------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
 MyExit = 1;
}
//---------------------------------------------------------------
void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
 MyCheck = CheckBox1->Checked;
}
//---------------------------------------------------------------
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
  mlfHGTerminate();
}
//---------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
 StringGrid1–>Cells[0][0]="a";
 StringGrid1–>Cells[0][1]="b";
 StringGrid1–>Cells[0][2]="c";
 StringGrid1–>Cells[0][3]="d";
 StringGrid1–>Cells[0][4]="r";
 StringGrid1–>Cells[0][5]="T0";
 StringGrid1–>Cells[0][6]="Tfin";
 StringGrid1–>Cells[0][7]="y0(1)";
 StringGrid1–>Cells[0][8]="y0(2)";
 StringGrid1–>Cells[0][9]="RelTol";
 StringGrid1–>Cells[1][0]=1;
 StringGrid1–>Cells[1][1]=0.01;
 StringGrid1–>Cells[1][2]=1;
 StringGrid1–>Cells[1][3]=0.02;
 StringGrid1–>Cells[1][4]=0.005;
 StringGrid1–>Cells[1][5]=0;
 StringGrid1–>Cells[1][6]=100;
 StringGrid1–>Cells[1][7]=20;
 StringGrid1–>Cells[1][8]=30;
 StringGrid1–>Cells[1][9]=1E-6;
}
//---------------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)
{
Close();
}
```

Небольшое дополнение в файле Unit2.cpp представлено ниже.

```
// Файл "Unit2.cpp"
< .. >
extern double MyCheck, MyExit, Mass[10];
< .. >
extern mxArray * Mfunc_AssignValue(int nargout_, mxArray * VarNumber)
{
  mxArray *ReturnNumber = NULL;
  int VarNum;
  VarNum = (int)*mxGetPr(VarNumber);
  switch (VarNum)
  {
   case 0:  ReturnNumber = mlfDoubleMatrix(1, 10, Mass, NULL); break;
   case 1:  ReturnNumber = mlfScalar(MyCheck); break;
   case 2:  ReturnNumber = mlfScalar(MyExit);
  }
  return ReturnNumber;
}
```

MATLAB-программа принимает следующий вид:

```
% Файл "func.m"
function main
global a b c d r mMyExit Tfin oldpers,
tic
mMyExit = 0;
oldpers = 0;
mass = AssignValue(0);
a = mass(1); b = mass(2); c = mass(3); d = mass(4);
r = mass(5); T0 = mass(6); Tfin = mass(7);
y0(1) = mass(8); y0(2) = mass(9);
reltol = mass(10);
mShowGraph = AssignValue(1);
yp = [c/d; (a–r*c/d)/b];
opt = odeset('RelTol', reltol);
tt = []; yy = [];
[tt, yy] = ode45(@lv, [T0, Tfin], y0, opt);
tt = tt'; yy = yy';
time = toc;
if (mShowGraph),
  figure(1),
  plot(tt, yy(1, :), 'g', tt, yy(2, :), 'r'), grid on,
  title('Solution of Modified Lotka-Volterra Equation'),
  legend('Preys','Predators'),
  xlabel('Time'), ylabel('Populations'),
  figure(2),
  plot(yy(1,:), yy(2,:), 'b', yp(1), yp(2), 'r.'), grid on,
  title('Phase Trajectory'),
  legend('Phase Trajectory', 'Rest Point'),
  xlabel('Preys'), ylabel('Predators')
end,
fprintf(1, '%3.1f seconds', time);
SyncVCL(0);
function CloseThread;
%#external
%-------------------------------------------------------------
function SyncVCL(MethodNumber);
%#external
%-------------------------------------------------------------
function OutVar = AssignValue(VarNumber);
%#external
```

```
%-------------------------------------------------------------
function z = lv(t, y)
global a b c d r mMyExit Tfin oldpers,
mMyExit = AssignValue(2);
if (mMyExit)
  CloseThread;
end;
pers = floor((t / Tfin) * 100);
if (pers > oldpers)
   fprintf(1, '%4.0f%% Complete', pers);
   SyncVCL(0);
   oldpers = pers;
end
z = [(a–b*y(2)–r*y(1))*y(1); (–c+d*y(1))*y(2)];
%-------------------------------------------------------------
function hh = title(string, varargin)
if nargin > 1 & (nargin–1)/2–fix((nargin–1)/2),
  error('Incorrect number of input arguments')
end
ax = gca;
if isappdata(ax,'MWBYPASS_title'),
   h = mwbypass(ax,'MWBYPASS_title',…
         string,varargin{:});
else
   h = get(ax,'title');
   set(h, 'FontAngle', get(ax, 'FontAngle'), ...
         'FontName', get(ax, 'FontName'), ...
         'FontSize', get(ax, 'FontSize'), ...
         'FontWeight', get(ax, 'FontWeight'), ...
         'Rotation', 0, ...
         'string', string, varargin{:});
end
if nargout > 0
  hh = h;
end
%-------------------------------------------------------------
function hh = xlabel(string, varargin)
if nargin > 1 & (nargin–1)/2–fix((nargin–1)/2),
  error('Incorrect number of input arguments')
end
ax = gca;
if isappdata(ax,'MWBYPASS_xlabel')
   h = mwbypass(ax,'MWBYPASS_xlabel',…
         string,varargin{:});
else
   h = get(ax,'xlabel');
   set(h, 'FontAngle', get(ax, 'FontAngle'), ...
         'FontName', get(ax, 'FontName'), ...
         'FontSize', get(ax, 'FontSize'), ...
         'FontWeight', get(ax, 'FontWeight'), ...
         'string', string,varargin{:});
end
if nargout > 0
  hh = h;
end
%-------------------------------------------------------------
function hh = ylabel(string, varargin)
if nargin > 1 & (nargin–1)/2–fix((nargin–1)/2),
  error('Incorrect number of input arguments')
end
ax = gca;
if isappdata(ax, 'MWBYPASS_ylabel')
   h = mwbypass(ax, 'MWBYPASS_ylabel',…
         string, varargin{:});
else
   h = get(ax, 'ylabel');
   set(h, 'FontAngle', get(ax, 'FontAngle'), ...
         'FontName', get(ax, 'FontName'), ...
         'FontSize', get(ax, 'FontSize'), ...
         'FontWeight', get(ax, 'FontWeight'), ...
         'string', string, varargin{:});
end
if nargout > 0
  hh = h;
end
%-------------------------------------------------------------
function hh = mwbypass(h,id, varargin)
fcn = getappdata(h, id);
if nargout > 0
   if ~iscell(fcn)
      hh = feval(fcn, varargin{:});
   else
      hh = feval(fcn{:}, varargin{:});
   end
else
   if ~iscell(fcn)
      feval(fcn, varargin{:});
   else
      feval(fcn{:}, varargin{:});
   end
end
```

В заключение заметим, что несмотря на очевидные удобства предлагаемой в данной работе технологии использования MATLAB-программ в средах визуального программирования C/C++, в ней имеется по крайней мере один отрицательный момент: вызов интерфейсной функции (например, присваивания) будет осуществляться гораздо медленнее, чем та же операция в Си. Так, из-за постоянного внешнего тестирования переменной MyExit цикл работает в полтора-два раза дольше в используемой конфигурации системы. При возникновении подобной ситуации можно на этапе выпуска финального релиза программного продукта осуществить замену соответствующей операции, если это возможно, на С-эквивалент.

ЛИТЕРАТУРА

1. MATLAB C Math Library User's Guide. Revised for Version 2.2 (Release 12.1). 2001. 432 c.
2. MATLAB C++ Math Library Reference. Revised for Version 2.2 (Release 12.1). 2001. 429 c.
3. MATLAB Compiler User's Guide. Sixth printing. Revised for Version 3.0 (Release 13). 2002. 274 c.
4. MATLAB C/C++ Graphics Library User's Guide. Fifth printing. Revised for Version 2.1 (Release 12). 2000. 52 c.
5. *Эрроусмит Д., Плейс К.* Обыкновенные дифференциальные уравнения. Качественная теория с приложениями. М.: Мир, 1986. 244 с.

Статья представлена факультетом информатики Томского государственного университета, поступила в научную редакцию 30 апреля 2003 г.