О П И С А Н И Е

универсальной Си программы **manzhuk** для решения систем обыкновенных дифференциальных уравнений (ОДУ) и дифференциально-алгебраических уравнений (ДАУ) в расширенном координатном пространстве переменных

Введение

Си программа **manzhuk** предназначена для решения систем ОДУ-ДАУ в расширенном координатном пространстве переменных, в котором в качестве зависимых переменных систем ОДУ-ДАУ включаются дифференциальные переменные, алгебраические переменные, а также производные дифференциальных переменных. Все переменные рассчитываются с гарантированной достоверностью и точностью результатов решения (обеспечиваемом AL-устойчивыми методами интегрирования систем ДАУ) и с одинаковой математической и компьютерной точностью для всех переменных расширенного координатного пространства переменных на каждом шаге интегрирования. Программа ориентирована в первую очередь на решение жестких и сверхжестких систем ОДУ-ДАУ с многопериодным характером решения, которые часто сводятся к решению плохо обусловленных систем линейных алгебраических уравнений (ЛАУ), и может использоваться в качестве математического ядра при математическом и компьютерном (численном) моделировании динамических систем, в первую очередь, для программно-методического комплекса (ПМК) ПА10 [8].

В ОПИСАНИИ используется следующая терминология:

Математическая модель (mathematical model) – система ОДУ-ДАУ, отображающая функционирование моделируемой динамической системы с требуемой достоверностью и точностью.

Математическое моделирование (modeling) – процесс получения математических моделей динамических систем на основе фундаментальных законов физики, химии и т.п.

Компьютерная модель – алгоритмическая и программная реализация математической модели.

Компьютерное (численное) моделирование (simulation) – процесс получения результатов математического моделирования динамических систем с помощью компьютерных моделей.

Жесткие системы ОДУ-ДАУ (stiff ODE-DAE systems (stiff set of ODE-DAE)) - системы ОДУ-ДАУ со степенью жесткости более 10^6.

Плохообусловленные системы ЛАУ (badly conditioned (ill-conditioned) LAE systems (ill-conditioned set of LAE)) - системы ЛАУ с числом обусловленности более 10^6, к решению которых часто сводится решение жестких и сверхжестких систем ОДУ-ДАУ неявными методами интегрирования.

Достоверность компьютерного моделирования (reliability of simulation) – отклонение результатов компьютерного моделирования динамических систем от заведомо точных результатов не более чем на 20 процентов (точные результаты получают аналитически или экспериментально, как правило, оценивая относительную погрешность численного моделирования в процентах).

Точность компьютерного моделирования (accuracy of simulation) – фактическая степень отклонения результатов достоверного численного моделирования от заведомо точных результатов.

Математическая точность (accuracy) решения систем ОДУ-ДАУ – относительная погрешность нормы векторов решения систем ОДУ-ДАУ (параметр eps) по сравнению с заведомо точным аналитическим решением в относительных единицах (в MATLAB по умолчанию eps=0.001).

Компьютерная точность (precision) решения систем ОДУ-ДАУ и ЛАУ – количество верных значащих цифр в каждом элементе векторов решения систем ОДУ-ДАУ и ЛАУ по сравнению с заведомо точным аналитическим (идеальным для систем ЛАУ) решением.

Решение систем ЛАУ с повышенной точностью вычислений (LAE systems solution with extra precision calculations) - применение методов получения решений с повышенной точностью вычислений (extra precision calculations), реализованных в пакетах прикладных матема-

тических программ, а также в библиотеках математических программ и описанных в документации на соответствующие пакеты и библиотеки. Пакеты прикладных математических программ:
Maple (http://maplesoft.com - метод Software Floating Point (SFP метод)),
MATLAB (http://mathworks.com - метод Variable Precision Arithmetic (VPA метод)),
Mathematica (http://wolfram.com - метод Arbitrary Precision Arithmetic (APA метод)),
Библиотека Intel для десятичных вычислений с повышенной точностью (стандарт IEEE 754-2008 Decimal Floating-Point for Intel® Architecture Processors, IEEE Standard for Floating-Point Arithmetic 754-2008, IEEE, 2008:
http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5223319&url [1],
переменные типа Decimal на языке C#, класс java.math.BigDecimal на языке Java,
а также Си (Си++) библиотеки для вычислений с повышенной точностью:
http://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic
http://en.wikipedia.org/wiki/GNU_Multi-Precision_Library
http://en.wikipedia.org/wiki/MPFR
http://en.wikipedia.org/wiki/Class_Library_for_Numbers
http://gmplib.org/
http://www.boost.org/doc/libs/1_53_0/libs/rational/rational.html .
http://speleotrove.com/decimal/dnusers.html#example7
http://www.ttmath.org/

Научно-исследовательские работы по математическому и компьютерному моделированию динамических систем выполняются преподавателями, аспирантами, сотрудниками и студентами кафедры САПР МГТУ имени Н.Э.Баумана с 1974 года (первоначально для математического и компьютерного моделирования электронных схем). При программной реализации численных методов решения систем ОДУ-ДАУ (при разработке математического ядра программ математического и компьютерного моделирования динамических систем) следует выделить два основных типа погрешностей численного решения систем ОДУ – глобальная погрешность (или качественная ошибка) и локальная погрешность (или количественная ошибка). Математики-программисты основное внимание уделяют оценке математической локальной погрешности интегрирования (обычно методом Рунге или методами, основанными на предсказании-коррекции (prediction-correction methods)), хотя главное, - это получение качественно достоверного и корректного численного решения систем ОДУ-ДАУ. В работах [2,3] показано, что для гарантии получения качественно корректного решения разнообразных систем ОДУ при контроле только локальной погрешности интегрирования численный метод решения систем ОДУ должен быть AL-устойчивым, т.е. абсолютно устойчивыми строго в левой полуплоскости комплексной плоскости устойчивости методов численного решения систем ОДУ. Следует отметить, что любые численные методы интегрирования систем ОДУ при очень маленьких шагах интегрирования будут квази-AL-устойчивыми, т.к. области абсолютной устойчивости всех методов интегрирования выходят из точки (0,0) комплексной плоскости устойчивости. В работе [4] на основе Паде аппроксимации экспоненты в конечную дробь и использования многочленов Якоби [5] были получены коэффициенты AL-устойчивых неявных методов интегрирования систем ОДУ 2-го, 4-го и 6-го порядков точности. Программная реализация этих методов сводится к многократному получению матриц Якоби для решаемой системы ДАУ и решению соответствующих систем ЛАУ на каждом шаге интегрирования, что, как правило, приводит к нескольким тысячам и более обращений к программе-решателю систем ЛАУ (LAE solver) на всем заданном отрезке интегрирования системы ДАУ. AL-устойчивые неявные методы интегрирования систем ДАУ 2-го и 4-го порядков точности были реализованы в программе DMAN на языках ФОРТРАН и Си при использовании удвоенной точности вычислений [6]. Конечная математическая и компьютерная точность численного моделирования динамических систем может быть невысокой, т.к. исходные параметры моделируемых систем получают с невысокой математической и компьютерной точностью, поэтому, по умолчанию, математическая точность решения систем ОДУ-ДАУ задается невысокой (eps=0.001 в MATLAB). Решение с данным параметром eps большого количества тестовых и практических задач математического и компьютерного моделирования динамических систем с помощью программы DMAN показало, что для получения качественно достоверного и корректного решения жестких и сверхжестких систем ОДУ-ДАУ необходимо на всех шагах чис-

ленного интегрирования обеспечивать численное решение соответствующих плохо обусловленных систем ЛАУ с повышенной точностью, т.е. с **точностью в 15 верных значащих цифр для всех значений элементов вектора решения систем ЛАУ**.

Математические модели динамических процессов в реальных технических системах и объектах необходимо получать на основе фундаментальных физических законов именно в форме неоднородных систем ДАУ, не разрешенных относительно производных, и решать эти системы без каких-либо эквивалентных математических преобразований и без получения "правой" части для производных в явном, аналитическом виде. При математическом моделировании динамических процессов в реальных технических системах и объектах состояние этих систем и объектов обычно рассматривают в пространстве дифференциальных переменных или в «пространстве переменных состояния» с получением в явном аналитическом виде производных этих переменных по времени, но этого недостаточно для достоверного и точного математического и компьютерного моделирования реальных технических систем и объектов, так как любой дифференциальной переменной состояния в реальном мире обычно соответствует алгебраическая переменная. Например, в электрике и электронике переменным состояния, напряжениям на емкостях и токам через индуктивности, соответствуют алгебраические переменные, токи через емкости и напряжения на индуктивностях, в механике переменным состояния, скоростям тел определенной массы, соответствуют алгебраические переменные, соответствующие силы инерции, и т.д. и т.п., поэтому состояние технических динамических систем и объектов следует рассматривать в пространстве дифференциально-алгебраических переменных. Основным физическим свойством алгебраических переменных является возможность идеального скачка значения этих переменных в бесконечно малый отрезок времени, в то время как для дифференциальных переменных состояния такие скачки не возможны, поэтому общепринятое приведение систем ДАУ к системам ОДУ в нормальной форме Коши с преобразованием алгебраических переменных в дифференциальные переменные состояния физически неверно и полученная таким образом математическая модель динамических процессов в реальных технических системах и объектах может быть физически не достоверна. Следует отметить, что в задачах анализа динамики в небесной механике, химической кинетике, экологии и ряда других задач алгебраические переменные обычно не нужны, но в таких задачах кроме точного значения дифференциальных переменных состояния необходимо на каждом шаге интегрирования получать с такой же точностью производные дифференциальных переменных. Следует также отметить, что многие методы численного решения систем ОДУ разработаны только для однородных систем ОДУ, но известное из математики преобразование неоднородных систем ОДУ в однородные системы ОДУ путем однократного дифференцирования независимой переменной времени и преобразования этой переменной в дифференциальную переменную с добавлением в неоднородную систему ОДУ соответствующего дифференциального уравнения dt/dt=1 также физически не достоверно.

В известных программах-решателях систем ОДУ-ДАУ (ODE-DAE Solvers) используются три классические постановки задач решения систем ОДУ-ДАУ.

1) Нормальная форма Коши в координатном базисе дифференциальных переменных состояния (явная форма представления систем ОДУ-ДАУ):

$$dX / dt = F(X, t) \quad ,$$

где X - вектор координатного базиса дифференциальных переменных состояния размерностью m. F - вектор-функция правых частей системы ОДУ размерностью m, t – независимая переменная (обычно время). Заданы начальные условия $X_0=X(0)$ и отрезок интегрирования $t=[T0,TK]$, $T0$ – заданное время начала интегрирования, TK – заданное время окончания интегрирования.

2) Дифференциально-алгебраическая форма разных индексов в полном координатном базисе дифференциально-алгебраических переменных (полуявная форма):

$$\begin{cases} dX / dt = F(X, Y, t) \\ G(X, Y) = 0 \end{cases} \qquad \textit{или} \qquad M\, dX/dt=F(X,t),$$

где Y - вектор алгебраических переменных полного координатного базиса дифференциально-алгебраических переменных размерностью k. G - вектор-функция размерностью k. M – сингу-

лярная матрица ранга *m* размером $(m+k)^2$. Заданы согласованные начальные условия: $X_0=X(0)$ $Y_0=Y(0)$ и отрезок интегрирования $t=[T0,TK]$.

3) Дифференциально-алгебраическая форма в полном координатном базисе дифференциально-алгебраических переменных (неявная форма):

$$G(X, dX/dt, Y, t) = 0,$$

где *G* - вектор-функция размерностью *m + k*. Заданы согласованные начальные условия и отрезок интегрирования. Задача решения систем ОДУ-ДАУ в данной постановке была поставлена в полном координатном базисе дифференциально-алгебраических переменных Линдой Петзольд и ее научным руководителем, известным математиком Гиром в 1982 г. и была разработана знаменитая программа DASS (Differential Algebraic Systems Solver) на основе метода BDF (Backward Differential Formula), заменой в вышеприведенной формуле вектора производных по формулам Гира $dX/dt = BDF(X,h)$ и решением на каждом шаге интегрирования замкнутой системы нелинейных алгебраических уравнений (НАУ) $G(X,Y)=0$ относительно дифференциально-алгебраических переменных *X* и *Y* [2].

Программа-решатель систем ДАУ (DAE Solver) manzhuk рассматривает системы ДАУ в еще более **расширенном координатном пространстве переменных**, в котором к дифференциально-алгебраическим переменным добавлены производные дифференциальных переменных. Задача решения систем ДАУ в расширенном координатном пространстве переменных была поставлена в статье [3]:

$$G(PX, X, Y, t) = 0, \qquad (1)$$

где $PX=dX/dt$ - вектор производных дифференциальных переменных по времени для расширенного координатного пространства переменных размерностью *m*. *G* - вектор-функция размерностью *m + k*. Заданы начальные условия $X_0=X(0)$ и отрезок интегрирования $t=[T0,TK]$. Алгоритмы для решения системы (1) основаны на совместном решении систем НАУ $G(X,Y,PX)=0$ и систем ЛАУ $H(X,PX)=0$, сформированных на соответствующих стадиях методов интегрирования, относительно дифференциально-алгебраических переменных *X и Y* и производных дифференциальных переменных *PX*. Были разработаны формулы *H(X,PX)* *S*-стадийных неявных методов интегрирования систем ДАУ вида (1) в расширенном координатном пространстве дифференциально-алгебраических переменных и производных дифференциальных переменных *X, Y* и *PX* [4]:

$$H_i(PX_i, X_i, X_{n-1}, PX_{n-1}, h_n) =$$

$$= h_n \sum_{j=1}^{s} d_{ij} PX_j - \sum_{j=1}^{s} a_{ij} X_j - b_i X_{n-1} - h_n c_i PX_{n-1} = 0$$

$$X_n = X_s, PX_n = PX_s, i = 1,...,s, t_n = t_s,$$

s – число стадий ,

$h_n = (t_n - t_{n-1})$ – *n – й шаг интегрирования*,

d_{ij}, a_{ij}, b_i, c_i – параметры метода.

Эти формулы не требуют получения «правой» части решения системы ОДУ-ДАУ (вектор-функции *F* в известных формулах численного решения систем ОДУ). На основе этих формул были разработаны и реализованы три метода интегрирования [4]:

M1 - A-устойчивый первого порядка точности (совпадает по параметрам с неявным методом Эйлера);

M2 – AL-устойчивый неявный метод второго порядка точности (совпадает по параметрам с неявным методом трапеций);

M3 - AL-устойчивый неявный метод четвертого порядка точности (совпадает по параметрам с неявным методом Лобатто IIIA).

Математическая постановка задачи решения систем ДАУ в расширенном координатном базисе с гарантированной достоверностью и точностью

Дана система ДАУ общего вида в расширенном координатном базисе переменных, не разрешенная относительно производных:

$$G(PX,X,Y,t)=0 \qquad (2),$$

где X - вектор дифференциальных переменных системы (2) размерностью m; PX - вектор производных дифференциальных переменных по переменной t размерностью m, т.е. $PX = dX/dt$; Y - вектор алгебраических переменных системы (2) размерностью k; t - независимая переменная (обычно, время); G - вектор-функция системы (2) размерностью n, где $n=(m+k)$. Заданы отрезок интегрирования и начальные условия для вектора дифференциальных переменных: $X0=X(t0)$, где $t0$ - начальный момент времени интегрирования. Начальные значения остальных переменных, т.е. $PX(t0)$ и $Y(t0)$ должны рассчитываться перед началом интегрирования автоматически. Необходимо достоверно и точно вычислить вектора $X(t)$, $PX(t)$, $Y(t)$ как функции времени на заданном отрезке интегрирования с математической точностью eps не менее 0.001.

В качестве примера рассмотрим систему ОДУ осциллятора Дуффинга (англ. Duffing oscillator) - простейшую одномерную нелинейную динамическую систему, моделирующую движение одномерной частицы в потенциальном поле. Особенностью осциллятора Дуффинга является возможность получения хаотической динамики (эффект бифуркации, т.е. удвоения частоты собственных колебаний осциллятора). Уравнение движения для осциллятора Дуффинга имеет вид: $m\,d^2x/dt^2=a\,x+b\,x^3$, где x и m соответственно - координата частицы и её масса, a и b параметры диссипации. В отсутствие диссипации (трения), гармонический (линейный) осциллятор, находящийся под действием внешней периодической силы, испытывает резонанс, если частота этой силы совпадает с собственной частотой колебаний осциллятора. В отличие от линейного осциллятора, осциллятор Дуффинга под действием внешней периодической силы $A_F\,cos(om\,t)$ может испытывать хаотическое бистабильное поведение. Для осциллятора Дуффинга, имеющего стационарный (не хаотический) режим колебаний, система ОДУ второго порядка в нормальной форме Коши может иметь вид:

dx1/dt = x2
*dx2/dt = 0.5*x1 - 0.25*x2 -0.5* x1*x1*x1 +0.3*cos(om*t)*
где x1,x2 - дифференциальные переменные системы ОДУ; *om* - параметр.

Эта система может быть представлена как система ДАУ вида (2) в расширенном координатном пространстве переменных по-разному (в зависимости от методов получения данной математической модели), например:

1-ый вариант:

g(X) = px1 - x2 = 0
g2(X) = px2 -0.5 x1 + 0.25*x2 +0.5* x1*x1*x1 -0.3*cos(om*t) = 0*
где *G = (g1,g2); X = (x1,x2); PX = (px1,px2); px1 = dx1/dt; px2 = dx2/dt; m = 2; k = 0; n = 2.*

2-ой вариант:

g1(X) = px1 - x2 = 0
*g2(X) = px2 - 0.5*x1 + 0.25*x2 + 0.5*y1 –0.3*cos (om*t) = 0*
*g3(X) = x1*x1*x1 - y1 = 0*
где *G = (g1,g2,g3); X = (x1,x2); PX = (px1,px2); px1 = dx1/dt; px2 = dx2/dt; Y = (y1); m = 2;*
k = 1; n = 3.

Отметим, что замена нелинейных функций в исходной системе ОДУ на дополнительные алгебраические переменные превращает нелинейные системы ОДУ в линейные подсистемы ОДУ плюс подсистемы НАУ.

В процессе интегрирования система (2) на каждом *n*-ом шаге интегрирования автоматически дополняется системой:

$$H_i(PX_n,X_n,h_n)=0, \qquad (3)$$

где h_n - *n*-ый шаг интегрирования; H_i - вектор-функция размерностью *m*, которая соответствует выбранному методу интегрирования на *i*-ой стадии метода интегрирования. Например, для одностадийного неявного метода M1 (совпадает по параметрам с неявным методом Эйлера) вектор-функция H_1 имеет вид:

$H_1 = PX_n - (1/h_n) * (X_n – X_{n-1}) = 0$,

где X_{n-1} - значение вектора дифференциальных переменных в момент времени t_{n-1}; t_{n-1} - момент времени в начале шага интегрирования h_n; X_n - значение вектора дифференциальных переменных в текущий момент времени t_n; $h_n=(t_n-t_{n-1})$.

manzhuk - программа-решатель систем ДАУ в расширенном координатном базисе с полными матрицами Якоби:

Описание функции:

```
void manzhuk(double z[],double px[],double z1[],double xp1[],double f[],
      double rj1[],double rj2[],double t, double t0,double tk,
      double h,double hmn,double hmx,double eps,double *tkv,
      int n,int m,int nm,int ncon,int *nbad,int *ier,int ip[],
      void fct(double z[],double px[],double f[],double rj1[],double rj2[],
      int n,int m,double t,double h, int ncon,int *nbad,int ip[]),
      void out(double z[],double px[],int n,int m,double t,double t0,double tk,
      double h,double *tkv,int ncon,int ip[]))
```

Пример обращения к функции:

```
manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,
&ier,ip,fcttask001,outtask001);
```

В программе-решателе систем ДАУ manzhuk реализованы 3 метода интегрирования систем ДАУ [4, 5]:

M1 - A-устойчивый первого порядка точности (совпадает по параметрам с неявным методом Эйлера);

M2 – AL-устойчивый неявный метод второго порядка точности (совпадает по параметрам с неявным методом трапеций);

M3 - AL-устойчивый неявный метод четвертого порядка точности (совпадает по параметрам с неявным методом ЛобаттоIIIA).

Во всех методах на каждом шаге интегрирования по классическим алгоритмам выбора шага интегрирования оценивается относительная, локальная погрешность интегрирования для каждой дифференциальной переменной (по отношению к максимальному абсолютному значению каждой дифференциальной переменной на заданном отрезке интегрирования). Вычисленная погрешность сравнивается с заданной погрешностью eps и если она ее превышает, то шаг интегрирования h уменьшается. Максимальное абсолютное значение каждой дифференциальной переменной определяется в ходе интегрирования автоматически, эти значения можно также задать перед началом интегрирования.

Во всех методах на каждом шаге интегрирования решается система НАУ относительно векторов расширенного пространства переменных *X*, *PX*, *Y* методом Ньютона-Рафсона, для которого необходимо вычислять матрицу Якоби *RJ* для системы (2) и системы (3) с дальнейшим формированием матрицы коэффициентов системы ЛАУ на каждой итерации метода Ньютона-Рафсона. Система ЛАУ должна решаться с повышенной точностью вычислений [6]. Матрица Якоби *RJ1* для системы (3) вычисляется в программе-решателе систем ДАУ manzhuk автоматически. Матрица Якоби *RJ2* для системы (2) должна вычисляться в специ-

альной подпрограмме пользователя fct, наряду с вычислением вектор-функции *G* (массив f[] в решателе manzhuk). Эта матрица состоит из двух подматриц:

RJ = (RJ1, RJ2),

где *RJ* - матрица Якоби размером (*n*nm); nm=n+m. RJ1* - матрица частных производных вектор-функции *G* (массив f[] в решателе manzhuk) по переменным *PX* размером (*n*m),* т.е. матрица *RJ1 = [dG/dPX]*. *RJ2* - матрица частных производных вектор-функции *G* (массив f[] в решателе manzhuk) по переменным Z размером *(n*n),* т.е. матрица *RJ2 = [dG/dZ]*, где *Z* - вектор переменных системы ОДУ размерностью *n*, который включает дифференциальные переменные и алгебраические переменные системы (2), т.е. *Z = (X,Y).*

Для приведенного выше примера (уравнения Дуффинга):

1-ый вариант:

g1(X) = px1 - x2 = 0

g2(X) = px2 -0.5 x1 + 0.25*x2 +0.5* x1*x1*x1 -0.3*cos(om*t) = 0*

2-ой вариант:

g1(X) = px1 - x2 = 0

*g2(X) = px2 - 0.5*x1 + 0.25*x2 + 0.5*y1 –0.3*cos (om*t) = 0*

*g3(X) = x1*x1*x1 - y1 = 0*

матрица Якоби и соответствующие подматрицы имеют вид:

1-ый вариант:

RJ	*RJ1*		*RJ2*	
	1	0		-1
	0	1	-0.5+1.5*x1*x1	0.25

2-ой вариант:

RJ	*RJ1*		*RJ2*		
	1	0	0	-1	0
	0	1	-0.5	0.25	0.5
	0	0	3*x1*x1	0	-1

Начальным приближением для метода Ньютона-Рафсона являются значения переменных в начале шага интегрирования, поэтому, если на некотором шаге нет сходимости итераций, то текущий шаг h уменьшается вплоть до минимального hmin, после чего выдается сообщение об ошибке (обычно из-за ошибок в формулах для вычисления элементов матрицы Якоби в подпрограммах пользователя или из-за расходимости самой системы ДАУ). Для определения наличия таких ошибок при вычислении элементов матрицы *RJ2* следует сначала выполнить расчеты для случая численного определения элементов матрицы *RJ2*, установив параметр ier=-1 (см. задачи task002 и task01 далее).

Рассмотрим примеры решения приведенных выше 2-х вариантов уравнения Дуффинга (задачи task001 и task002). Обращение к функции manzhuk (на примере 2-х вариантов решения уравнения Дуффинга: task001 и task002) имеет вид:

manzhuk(z,px,z1,px1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,&ier,ip,fcttask001, outtask001);

и

manzhuk(z,px,z1,px1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,&ier,ip,fcttask002, outtask002);

Подпрограммы fcttask001 и fcttask002 определения вектор-функции *G* (массив f[]) и матрицы Якоби *RJ* (массивы rj1[] и rj2[], в первом варианте элементы матрицы *RJ2* вычисляются аналитически, во втором варианте численно) и подпрограммы-функции outtask001 и outtask002 задания параметров для выдачи результатов решения соответствующих систем ДАУ на печать и для построения соответствующих графиков имеют вид:

```
//task001 task002 - Duffing equation
```

```
void fcttask001(double z[],double px[],double f[],double rj1[],double rj2[],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
      f[1]=px[1]-z[2];
            rj1[1*n+1]=1e0;
            rj2[1*n+2]=-1e0;
      f[2]=px[2]-0.5*z[1]+0.25e0*z[2]+0.5e0*z[1]*z[1]*z[1]-0.3*cos(om*t);
            rj1[2*n+2]=1e0;
            rj2[2*n+1]=-0.5e0+1.5e0*z[1]*z[1];
            rj2[2*n+2]=0.25e0;
      return;
}
void outtask001(double z[],double px[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
      if(ncon==0) fprintf(f01,"           Duffing equations,t,x1,x2,px1,px2\n");
      if(ncon==0) fprintf(f02,"           Duffing equations,t,x1,x2,px1,px2\n");
// start print of tabulation results
      if(ncon==0)tkp=deltatp;
      if(deltatp==0) goto m20;
      if(t<=tkp) goto m10;
      tkp=tkp+deltatp;
m10:
      if((tkp<*tkv)&&(*tkv>=t)) *tkv=tkp;
      if(t<tstartp) goto m20;
      if(t>tendp) goto m20;
      if(t==*tkv) fprintf(f01,"  %e   %e  %e  %e
%e\n",t,z[1],z[2],px[1],px[2]);
m20:
// end print of tabulation results
      fprintf(f02,"  %e   %e  %e  %e  %e\n",t,z[1],z[2],px[1],px[2]);
      return;
}
void fcttask002(double z[],double px[],double f[],double rj1[],double rj2[],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
      f[1]=px[1]-z[2];
            rj1[1*n+1]=1e0;
      f[2]=px[2]-0.5*z[1]+0.25e0*z[2]+0.5e0*z[3]-0.3*cos(om*t);
            rj1[2*n+2]=1e0;
      f[3]=z[1]*z[1]*z[1]-z[3];
      return;
}
void outtask002(double z[],double px[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
      if(ncon==0) fprintf(f01,"           Duffing equations,t,x1,x2,px1,px2\n");
      if(ncon==0) fprintf(f02,"           Duffing equations,t,x1,x2,px1,px2\n");
// start print tabulation
      if(ncon==0)tkp=deltatp;
      if(deltatp==0) goto m20;
      if(t<=tkp) goto m10;
      tkp=tkp+deltatp;
m10:
      if((tkp<*tkv)&&(*tkv>=t)) *tkv=tkp;
      if(t<tstartp) goto m20;
      if(t>tendp) goto m20;
      if(t==*tkv) fprintf(f01,"  %e   %e  %e  %e
%e\n",t,z[1],z[2],px[1],px[2]);
m20:
// end print tabulation
      fprintf(f02,"  %e   %e  %e  %e  %e\n",t,z[1],z[2],px[1],px[2]);
      return;
      }
```

На печать выдаются значения дифференциальных переменных и их производных в стационарном режиме колебаний, начиная с момента времени начала печати tstartp=240 до

момента времени окончания печати tendp=245 с временным интервалом печати deltatp=1, tp – вспомогательная переменная, соответствующая табулированным моментам времени, которую сначала надо установить, равной шагу печати tp=deltatp (если deltatp=0, то результаты на печать не выводятся), на график выдаются значения этих переменных на всех шагах интегрирования.

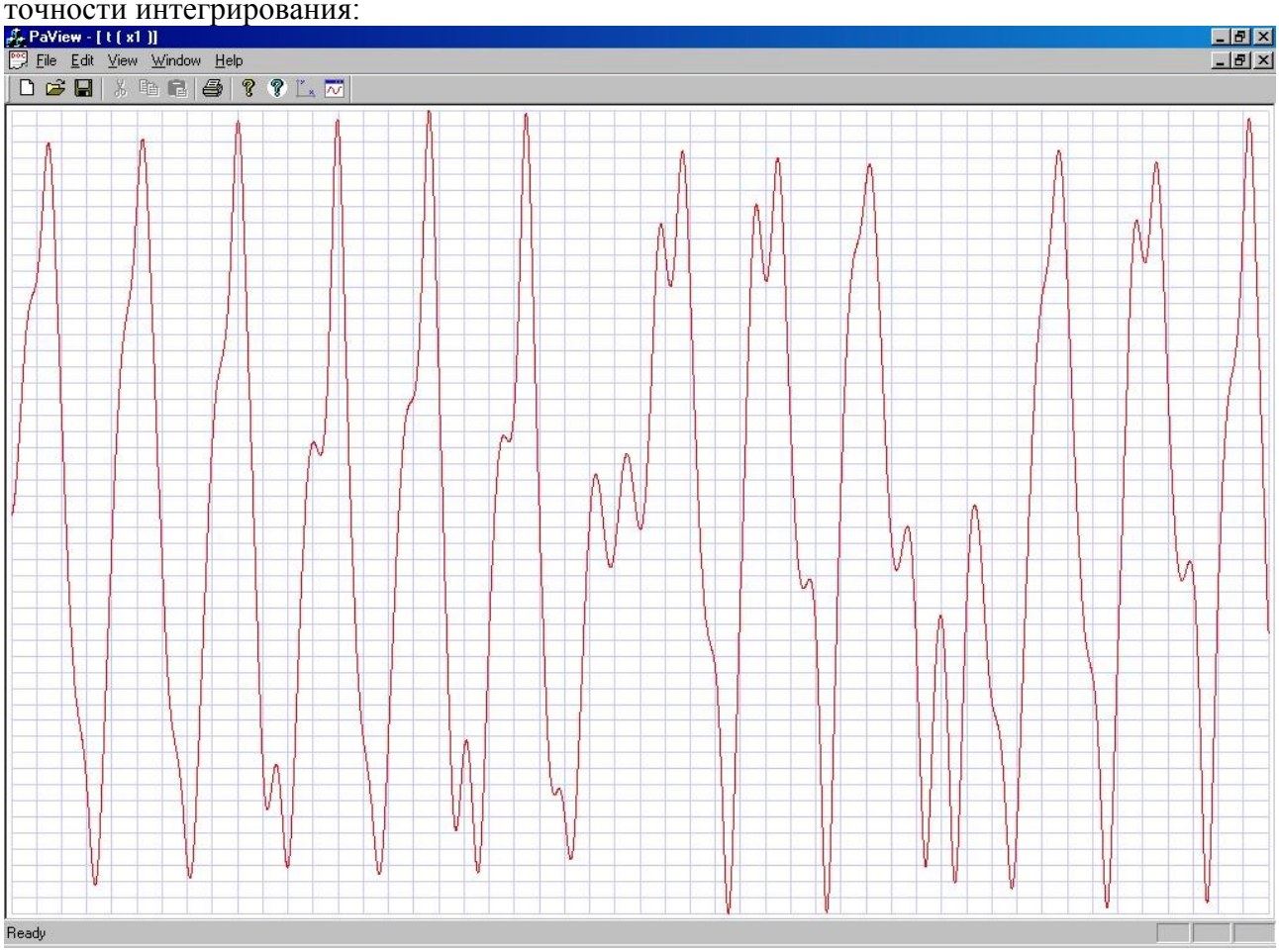
Основные программы-функции для этой задачи имеют вид:

```
//task001  - Duffing equations
      f01=fopen("task001.rez","wt");
      f02=fopen("grtask001.rez","wt");
            om=1e0;
            pi4=atan(1.0);
            eps=1e-3;
            fprintf(f01,"        relative tolerance - eps=%e pi4=%e\n",eps,pi4);
            nm=1;
            fprintf(f01,"        number of method - nm=%d\n",nm);
            t0=0e0;
            tk=250e0;
            tstartp=240e0;//time for stasrt printing
            tendp=245e0;//time for end printing
            deltatp=1e0;//step for printing
            fprintf(f01," tstartp=%e deltatp=%e
tendp=%e\n",tstartp,deltatp,tendp);
            hmn=1e-10;
            hmx=tk/10e0;
            n=2;
            m=2;
            z[1]=0e0;
            z[2]=0e0;
            z1[1]=fabs(z[1]);
            z1[2]=fabs(z[2]);
      manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad
,&ier,ip,fcttask001,outtask001);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
            fprintf(f01,"\n");
            for(i=1;i<=16;i++)printf("%d ",ip[i]);
            printf("\n");
            fclose(f01);
            fclose(f02);

// task002 - Duffing equations
      f01=fopen("task002.rez","wt");
      f02=fopen("grtask002.rez","wt");
            om=1e0;
            pi4=atan(1.0);
            eps=1e-3;
            fprintf(f01,"        relative tolerance - eps=%e pi4=%e\n",eps,pi4);
            nm=3;
            fprintf(f01,"        number of method - nm=%d\n",nm);
            t0=0e0;
            tk=250e0;
            tstartp=240e0;//time for start printing
            tendp=245e0;//time for end printing
            deltatp=1e0;//step for printing
      fprintf(f01," tstartp=%e deltatp=%e tendp=%e\n",tstartp,deltatp,tendp);
            hmn=1e-10;
            hmx=tk/10e0;
            n=3;
            m=2;
            z[1]=0e0;
            z[2]=0e0;
            z1[1]=fabs(z[1]);
            z1[2]=fabs(z[2]);
ier=-1;
```

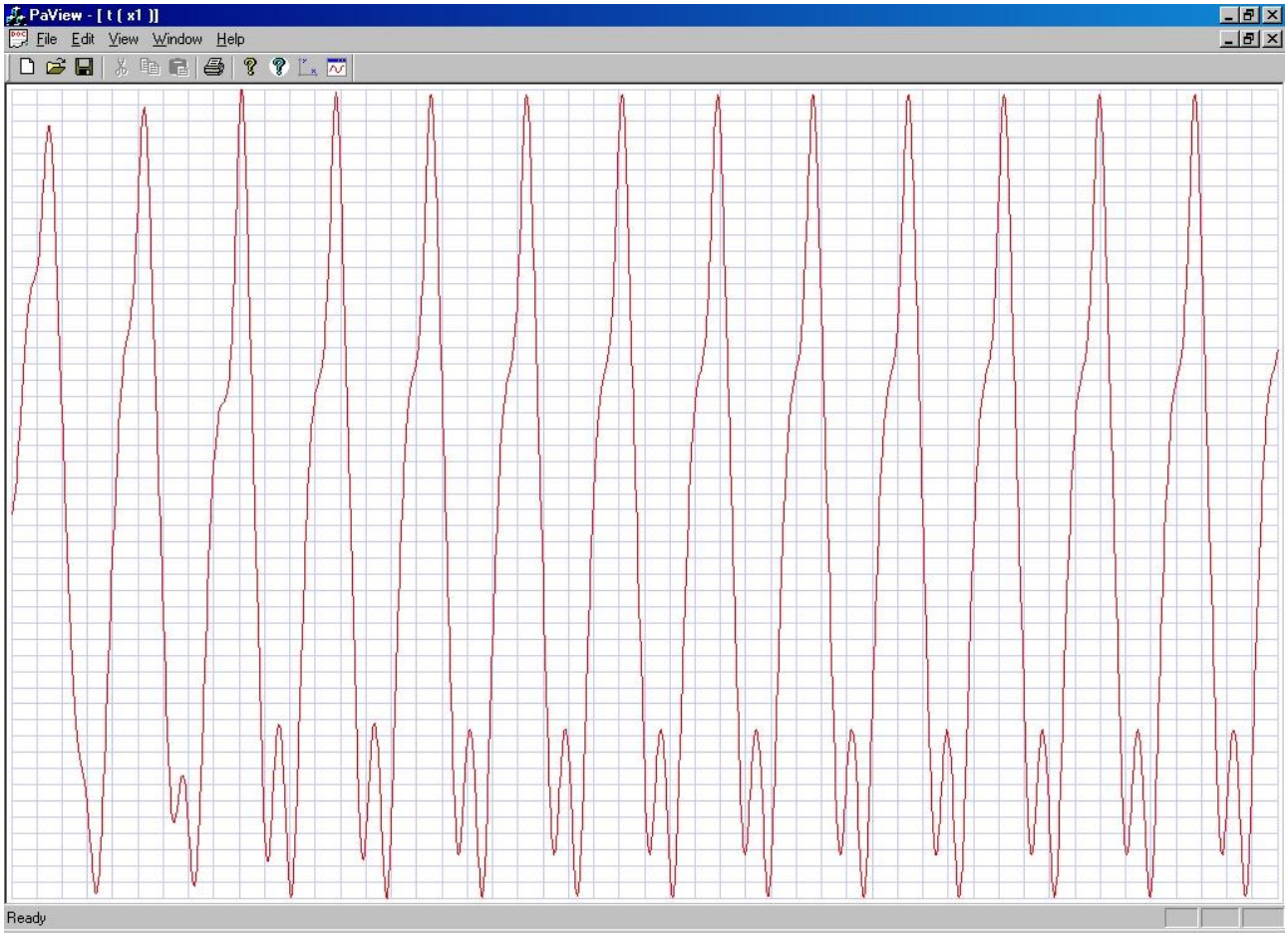
```
      manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad
,&ier,ip,fcttask002,outtask002);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d  ",ip[i]);
            fprintf(f01,"\n");
            for(i=1;i<=16;i++)printf("%d ",ip[i]);
            printf("\n");
            fclose(f01);
            fclose(f02);
```

Цель решения данных уравнений Дуффинга – расчет стационарного режима колебаний и времени установления этих стационарных колебаний от нулевых начальных условий. В первом варианте решения данных уравнений Дуффинга используется A-устойчивый неявный метод интегрирования метод M1 (параметры совпадают с параметрами неявного метода Эйлера), во втором варианте AL-устойчивый неявный метод 4-го порядка точности (метод M3) при одинаковой заданной точности интегрирования eps=0.001, tstartp=240 сек. – время начала печати стационарного режима колебаний, tendp=245 сек. – время окончания печати стационарного режима колебаний, deltatp=1 сек. – шаг выдачи на печать стационарного режима колебаний.

Метод M1 не смог получить достоверное решение данных уравнений при заданной точности интегрирования:



Достоверное решение было получено методом M3:



Результаты расчета стационарного режима для метода M3:

```
        relative tolerance - eps=1.000000e-003 pi4=7.853982e-001

        number of method - nm=3
 tstartp=2.400000e+002 deltatp=1.000000e+000 tendp=2.450000e+002
          Duffing equations,t,x1,x2,px1,px2
  2.400000e+002   -1.047072e+000  3.134470e-001  3.134470e-001  6.982013e-002
  2.410000e+002   -7.911507e-001  1.257812e-001  1.257812e-001  -3.652856e-001
  2.420000e+002   -8.633323e-001  -2.667927e-001  -2.667927e-001  -3.418079e-001
  2.430000e+002   -1.226397e+000  -3.572472e-001  -3.572472e-001  2.616109e-001
  2.440000e+002   -1.340264e+000  1.962548e-001  1.962548e-001  6.353381e-001
  2.450000e+002   -9.135161e-001  5.594846e-001  5.594846e-001  8.424740e-002
ier= 0
4  7213  1727  38  43  0  0  0  7205  1727  0  0  0  0  0  0
```

Следует отметить, что при увеличении заданной точности на 3 порядка удалось получить достоверное решение данных уравнений методом M1, но при этом время решения было на несколько порядков больше, чем для метода M3 (285126 шагов интегрирования по сравнению с 1727 шагами для метода M3).

По результатам решения уравнения Дуффинга можно сделать следующий вывод:

Для AL-устойчивых методов интегрирования высокого порядка точности (метод M3) для задач с колебательным характером решений можно сразу получить достоверное решение при невысокой заданной точности интегрирования (eps=0.001). Для не AL-устойчивых методов интегрирования необходимо как минимум два расчета с увеличенными параметрами точности, и в случае не совпадающих результатов, следует увеличивать заданную точность интегрирования вплоть до получения достоверного решения. Поскольку при очень маленьком шаге интегрирования любые методы интегрирования (даже неявный метод Эйлера) будут квази-AL-устойчивыми, то практически получить достоверное решение для данного класса задач можно любым известным методом интегрирования, но время расчета может быть неприемлемым.

z - массив переменных системы ДАУ размерности n, в первых m элементах этого массива размещаются дифференциальные переменные *X* системы ДАУ, в остальных элементах размещаются алгебраические переменные, поэтому всегда должно выполняться усло-

вие (n >= m). Поскольку нумерация массива начинается с 1, то отводимый размер массива следует увеличить как минимум до (n+1). Перед началом интегрирования в первых m элементах этого массива размещаются начальные значения дифференциальных переменных – *X0*.

px - массив производных дифференциальных переменных по времени размерностью m, т.е. px=dx/dt. Поскольку нумерация массива начинается с 1, то отводимый размер массива следует увеличить как минимум до (m+1).

z1 - рабочий массив размерностью n, перед началом интегрирования в первых m элементах этого массива размещаются максимальные абсолютные значения дифференциальных переменных. Если они известны, то их считывают из файла и присваивают соответствующим значениям массива z1. Если они неизвестны, то в этот массив пересылаются абсолютные величины начальных значений дифференциальных переменных *X0*, как в вышеприведенном примере. Поскольку нумерация массива начинается с 1, то отводимый размер массива следует увеличить как минимум до (n+1).

px1 - рабочий массив размерностью m.

f - массив размерностью n для вычисления вектор-функции *G* (массив f[]) системы ДАУ в подпрограмме fct. Поскольку нумерация массива начинается с 1, то отводимый размер массива следует увеличить как минимум до (n+1).

rj1 - матрица размером (n*m) для вычисления частных производных вектор-функции *G* (массив f[]) по переменным px в подпрограмме fct, т.е. *rj1(1,1) = dg(1)/dpx(1), rj1(2,2) = dg(2)/dpx(2)* и т.д., хранится по строкам, т.е. rj1(1,1) == rj1[1*n+1], rj(2,2) == rj1[2*n+2] и т.д. Поскольку нумерация массива начинается с 1, то отводимый размер массива следует увеличить как минимум до ((n+1)*(m+1)).

rj2 - матрица размером (n*n) для вычисления частных производных вектор-функции *G* (массив f[]) по переменным z в подпрограмме fct, т.е. *rj2(1,1) = dg(1)/dz(1), rj2(1,2) = dg(1)/dz(2*) и т. д., хранится по строкам, т.е. rj2(1,1) == rj2[1*n+1], rj2(1,2) == rj1[1*n+2] и т.д. Поскольку нумерация массива начинается с 1, то отводимый размер массива следует увеличить как минимум до ((n+1)*(n+1)). В программе предусмотрена возможность численного вычисления элементов этой матрицы. Если перед обращением к программе установить параметр ier=-1, то все элементы матрицы rj2 будут вычисляться автоматически методом приращений так же, как во всех пакетах и библиотеках математических программ. Однако этот метод вычисления частных производных следует применять только на этапе отладки задачи решения систем ОДУ-ДАУ. Общее правило – все, что можно вычислить аналитически следует вычислять аналитически (частные производные для линейных функций следует вычислять только аналитически!). Для сложных функциональных зависимостей следует использовать численное вычисление частных производных покомпонентно. В предыдущей задаче task002 был приведен пример применения параметра ier=-1, далее будет приведен еще один пример применения параметра ier=-1 – расчет методом М3 уравнений орбиты Луны, взятых из книги [7] - MOON ORBIT EQUATIONS (задача task01):

```
//task01 - moon orbit equation
	double mu0,mu1;
void fcttask01(double z[],double px[],double f[],double rj1[],double rj2[],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
	double d1,d2;
	d1=sqrt(((z[1]+mu0)*(z[1]+mu0)+z[2]*z[2])*((z[1]+mu0)*(z[1]+mu0)+z[2]*z[2]
)*((z[1]+mu0)*(z[1]+mu0)+z[2]*z[2]));
	d2=sqrt(((z[1]-mu1)*(z[1]-mu1)+z[2]*z[2])*((z[1]-mu1)*(z[1]-
mu1)+z[2]*z[2])*((z[1]-mu1)*(z[1]-mu1)+z[2]*z[2]));
	f[1]=px[1]-z[3];
		rj1[1*n+1]=1e0;
	f[2]=px[2]-z[4];
		rj1[2*n+2]=1e0;
	f[3]=px[3]-z[1]-2.0*z[4]+mu1*(z[1]+mu0)/d1+mu0*(z[1]-mu1)/d2;
		rj1[3*n+3]=1e0;
	f[4]=px[4]-z[2]+2.0*z[3]+mu1*z[2]/d1+mu0*z[2]/d2;
		rj1[4*n+4]=1e0;
```

```
      return;
}
void outtask01(double z[],double px[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
      if(ncon==0) fprintf(f02,"           moon orbit equation,t,x1,x2,x3,x4\n");
      fprintf(f02,"  %e   %e  %e  %e  %e\n",t,z[1],z[2],z[3],z[4]);
      return;
}
```

Основная программа имеет вид:

```
      f01=fopen("task01.rez","wt");
      f02=fopen("grtask01.rez","wt");
            mu0=0.012277471e0;
            mu1=1e0-mu0;
            eps=1e-3;
            fprintf(f01,"        relative tolerance - eps=%e\n",eps);
            nm=3;
            fprintf(f01,"        number of method - nm=%d\n",nm);
            t0=0e0;
            tk=17.06521656015796e0;
            hmn=1e-6;
            hmx=tk/10e0;
            n=4;
            m=4;
            z[1]=0.994e0;
            z[2]=0e0;
            z[3]=0e0;
            z[4]=-2.00158510637908e0;
            z1[1]=0.994e0;
            z1[2]=0e0;
            z1[3]=0e0;
            z1[4]=2.00158510637908e0;
ier=-1;
      manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad
      ,&ier,ip,fcttask01,outtask01);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
            fprintf(f01,"\n");
            for(i=1;i<=16;i++)printf("%d ",ip[i]);
            printf("\n");
            fclose(f01);
            fclose(f02);
```

Методом М3 было получено достоверное решение этой задачи, т.к. значения переменных в конечный момент времени совпали с начальными условиями в пределах заданной точности интегрирования:

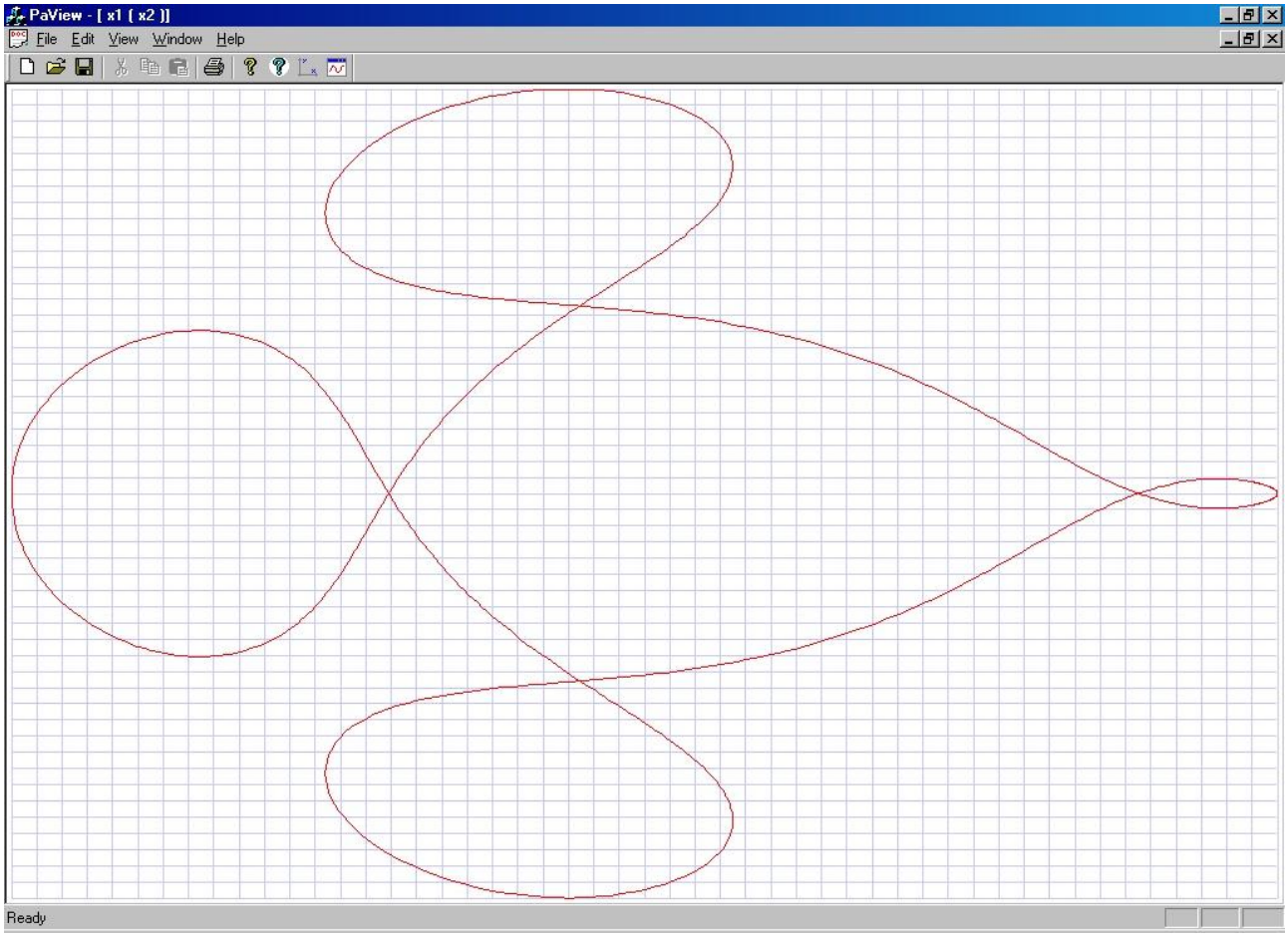
```
        relative tolerance - eps=0.001

        number of method - nm=3
ier= 0
4 1342 315 13 17 0 0 0 1336 315 0 0 0 0 0 0
```

0.000000e+000 9.940000e-001 0.000000e+000 0.000000e+000 -2.001585e+000

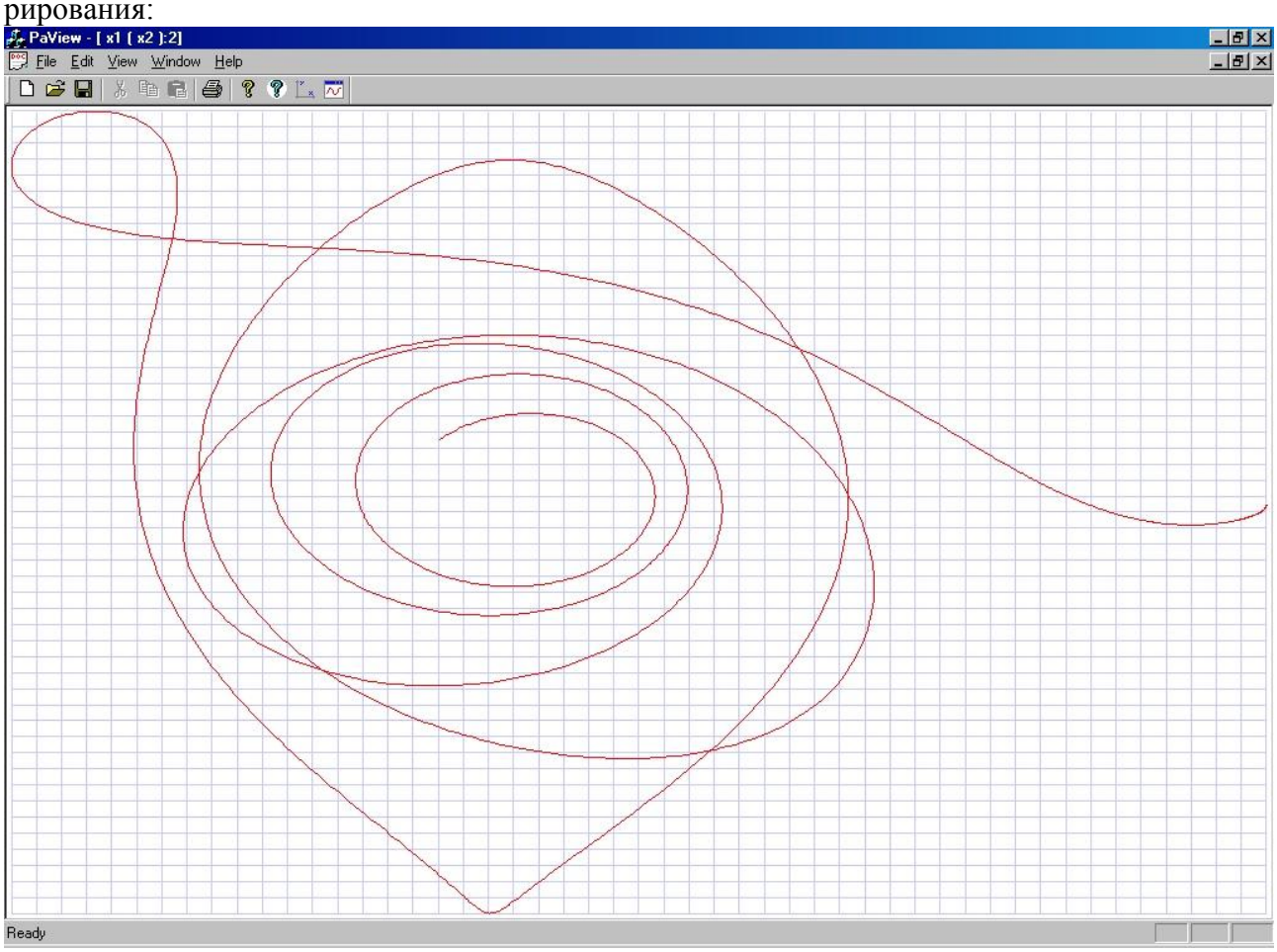
…

1.706522e+001 9.942272e-001 1.192969e-003 1.790359e-001 -1.943118e+000

В результате график траектории Луны, т.е. зависимости x2(x1) получился замкнутым:



Метод M1 не смог получить достоверное решение этой задачи при заданной точности интегрирования:



По результатам решения данных уравнений можно сделать вывод, аналогичный вышеприведенному выводу по результатам решения уравнения Дуффинга.

t - текущее время интегрирования.
t0 - заданное время начала интегрирования.
tk - заданное время окончания интегрирования.
h - текущий шаг интегрирования.
hmn - заданный минимальный шаг интегрирования.
hmx - заданный максимальный шаг интегрирования.
eps - заданная математическая точность интегрирования, одинаковая для всех переменных расширенного координатного пространства переменных (математическая точность решения системы ДАУ обычно задается равной 0.001, – именно такая математическая точность интегрирования систем ОДУ задается по умолчанию в пакете математических программ MATLAB).
tkv - переменная, с помощью которой можно точно табулировать задаваемые пользователем моменты времени в ходе интегрирования. Эта переменная устанавливается в подпрограмме пользователя out и обычно используется для табуляции результатов. Выше в задачах task001 и task002 был приведен пример табуляции результатов расчета, ниже приведен пример получения значений переменных в целочисленные моменты времени 0, 1, 2, для этого в подпрограмму out вставлены соответствующие операторы (задача task02):

```
//task02 - IMSL test - using tkv parameter for RESULTS TABULATION as in IMSL
void fcttask02(double z[],double px[],double f[],double rj1[],double rj2[],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
      f[1]=px[1]-2e0*z[1]+2e0*z[1]*z[2];
       rj1[1*n+1]=1e0;
       rj2[1*n+1]=-2e0+2e0*z[2];
       rj2[1*n+2]=2e0*z[1];
      f[2]=px[2]-z[2]*z[1]+z[2];
       rj1[2*n+2]=1e0;
       rj2[2*n+1]=-z[2];
       rj2[2*n+2]=-z[1]+1e0;
      return;
}
void outtask02(double z[],double px[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
//results for graphics
       if(ncon==0) fprintf(f02,"           tabulation test from
IMSL,t,x1(t),px1,x2,px2\n");
       if(ncon==0) fprintf(f01,"           tabulation test from
IMSL,t,x1(t),px1,x2,px2\n");
       fprintf(f02,"  %e   %e  %e  %e  %e\n",t,z[1],px[1],z[2],px[2]);
// start print of tabulation results
      if(ncon==0)tkp=deltatp;
      if(deltatp==0) goto m20;
      if(t<=tkp) goto m10;
      tkp=tkp+deltatp;
m10:
      if((tkp<*tkv)&&(*tkv>=t)) *tkv=tkp;
      if(t<tstartp) goto m20;
      if(t>tendp) goto m20;
if(t==*tkv) fprintf(f01,"  %e   %e  %e  %e  %e\n",t,z[1],z[2],px[1],px[2]);
m20:
// end print of tabulation results
      return;
}
```

На печать выдаются значения дифференциальных переменных и их производных, начиная с момента времени начала печати tstartp=t0 до момента времени окончания печати tendp=tk с временным интервалом печати deltatp=1, tp – вспомогательная переменная, соответствующая табулированным моментам времени, которую сначала надо установить, равной шагу печати tp=deltatp (если deltatp=0, то результаты на печать не выводятся).

Основная программа-функция имеет вид:

```
//task02 - IMSL test - using tkv parameter for RESULTS TABULATION as in IMSL
```

```
      f01=fopen("task02.rez","wt");
      f02=fopen("grtask02.rez","wt");
            eps=1e-3;
            fprintf(f01,"        relative tolerance - eps=%e\n",eps);
            nm=3;
            fprintf(f01,"        number of method - nm=%d\n",nm);
            t0=0e0;
            tk=10e0;
            tstartp=t0;//time for stasrt print
            tendp=tk;//time for end print
            deltatp=1e0;//step for printing
      fprintf(f01," tstartp=%e deltatp=%e tendp=%e\n",tstartp,deltatp,tendp);
            hmn=1e-7;
            hmx=5e0;
            n=2;
            m=2;
            z[1]=1e0;
            z[2]=3e0;
            z1[1]=z[1];
            z1[2]=z[2];
manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,&ier,
ip,fcttask02,outtask02);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
            fprintf(f01,"\n");
            for(i=1;i<=16;i++)printf("%d ",ip[i]);
            printf("\n");
            fclose(f01);
            fclose(f02);
```

В результате решения были выданы табулированные значения переменных в пределах заданной точности интегрирования:

```
        relative tolerance - eps=0.001

        number of method - nm=3
 tstartp=0.000000e+000 deltatp=1.000000e+000 tendp=1.000000e+001
           tabulation test from IMSL,t,x1(t),px1,x2,px2
  1.000000e+000   7.734449e-002  1.464448e+000  -7.184493e-002  -1.351181e+000
  2.000000e+000   8.497812e-002  5.779528e-001  7.172956e-002  -5.288394e-001
  3.000000e+000   2.908859e-001  2.492525e-001  4.367638e-001  -1.767484e-001
  4.000000e+000   1.446575e+000  1.872161e-001  2.351506e+000  8.360612e-002
  5.000000e+000   4.051529e+000  1.439454e+000  -3.560922e+000  4.392536e+000
  6.000000e+000   1.756134e-001  2.258617e+000  -4.420601e-001  -1.861974e+000
  7.000000e+000   6.530776e-002  9.088042e-001  1.191158e-002  -8.494522e-001
  8.000000e+000   1.472118e-001  3.667170e-001  1.864534e-001  -3.127319e-001
  9.000000e+000   6.504210e-001  1.875576e-001  1.056859e+000  -6.556622e-002
  1.000000e+001   3.143842e+000  3.486691e-001  4.095363e+000  7.474913e-001
ier= 0
4 515 125 5 5 0 0 0 511 125 0 0 0 0 0 0
```

ar - рабочий массив размером не менее, чем (3*(n+1)*(m+1) + 5*(n+1)*(n+1) + 9*(n+1) + 25*(m+1) + 3), в первых трех элементах этого массива размещаются максимальные (среди всех дифференциальных переменных) текущие значения относительных погрешностей интегрирования для методов M1, M2 и M3 соответственно. В последующих m элементах этого массива размещаются максимальные абсолютные значения дифференциальных переменных.

n - общее число уравнений в системе ДАУ.

m - количество дифференциальных переменных в системе ДАУ.

nm - заданный номер метода интегрирования:

nm = 1 - метод M1;

nm = 2 - метод M2;

nm = 3 - метод M3;

ncon - ключ расчета начальных значений переменных:

ncon = 0 - выполняется расчет начальных значений переменных px0 и y0 при заданном x0 для момента времени t0 от нулевых значений px0 и y0. Значения x0 соответствуют

исходным значениям первых m элементов массива z. До начала интегрирования с этим значением ключа ncon выполняется одно обращение к подпрограммам fct и out для выполнения однократных вычислений параметров, которые в ходе интегрирования не изменяются. После выполнения однократных вычислений значение ncon=1 устанавливается автоматически.

ncon = 1 - расчет начальных значений переменных px0 и y0 не производится. Этот признак следует использовать в случаях повторных обращений к программе-решателю manzhuk, если надо продолжить интегрирование решаемой системы ДАУ. До этого должно быть хотя бы одно обращение к решателю manzhuk с ncon=0 или с ncon=2.

При нормальном выходе из решателя manzhuk устанавливается ncon=1.

ncon = 2 - выполняется расчет начальных значений переменных px0 и y0 при заданном x0 для момента времени t0 от вводимых начальных значений px0 и y0. Эти значения либо вводятся из заранее сформированного файла исходных данных, либо присваиваются вычисленным заранее значениям этих переменных. Значения x0 соответствуют исходным значениям первых m элементов массива z.

nbad – (принципиально новый, важнейший параметр) - ключ уменьшения шага интегрирования, устанавливаемый в подпрограммах fct вычислений элементов вектор-функции *G* (массив f[]) и матрицы Якоби *RJ* (массивы rj1[] и rj2[]) системы ДАУ. Перед обращением к подпрограммам fct устанавливается nbad=0 на каждой итерации. Если отдельные переменные в подпрограммах fct принимают значения, которые могут привести к останову вычислений (корень из отрицательного числа, превышение допустимого порядка в степенных функциях, то необходимо установить nbad=1, тогда итерации будут прерваны, произойдет уменьшение шага интегрирования до тех пор, пока можно будет продолжать вычисления, либо до значения минимального шага с сообщением о не сходимости итераций. Если в моделях установить nbad=2, то произойдет уменьшение шага до учетверенного минимального с дальнейшим продолжением интегрирования с этим шагом (фактически расчет с новыми начальными условиями). nbad=2 можно установить только в том случае, если nbad не равно 1. nbad=2 следует использовать для идентификации точек разрыва производных в кусочно-нелинейных функциях.

Рассмотрим пример использования параметра nbad=1 для программирования функций с ограниченной областью определения аргумента. Вычисление любой вектор-функции *G* (массив f[]) системы ДАУ можно свести к вычислению совокупности некоторых функций одного аргумента, типа f(x). Рассмотрим функцию f(x) с ограниченной областью определения x, например f(x)=sqrt(x), определена только для x>=0. Поэтому при каждом обращении к этой функции необходимо проверить значение аргумента x и, если аргумент выходит из области определения функции, то следует установить параметр nbad=1, не вычисляя значения функции. Фрагмент программы для функции sqrt(x) будет:

```
        if (x < 0) goto m1000;
        f=sqrt(x);
        …
m1000   nbad=1;
        return;
```

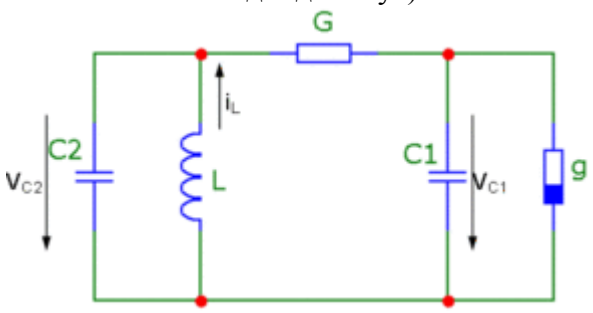
Рассмотрим еще пример использования параметра nbad=1. Некоторые функции, теоретически определенные для любых значений аргументов, практически могут принимать значения, выходящие за пределы разрядной сетки процессора, или иметь не реальные значения аргумента или функции. Например, функция f(x)=exp(x) при x>40 обычно выходит за допустимые значения в реальных динамических системах. В этом случае также следует использовать параметр nbad=1. Фрагмент программы для функции exp(x) будет:

```
        if (x>40.0) goto m1000;
        f=exp(x);
        …
m1000  nbad=1;
        return;
```

Рассмотрим пример использования параметра nbad=2 для программирования кусочно-линейной функции.

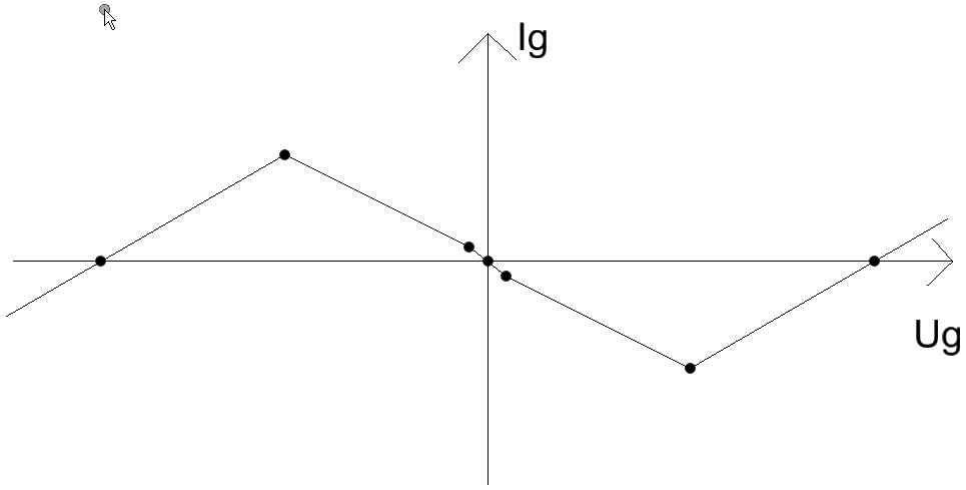
Важнейшей практической функцией f(x) является кусочно-линейная функция определенная на экспериментально снятых точках. В точках излома эта функция имеет разрыв всех производных и для прохода по этим точкам следует использовать параметр nbad=2. Далее приведен текст подпрограммы для вычисления любой кусочно-линейной функции y(x), заданной массивами значений точек излома xu(n) и yu(n). Задается также точность прохождения точек разрыва по x и по y - epsx и epsy и частная производная dy/dx – rjyx для матрицы rj, injr – признак численного расчета частной производной, indp – глобальная переменная, в которой хранится номер текущего отрезка кусочно-линейной функции.

Ниже приведен пример моделирование электронной схемы Чуа. Эта простейшая электрическая цепь, демонстрирующая режимы хаотических колебаний, была предложена профессором Калифорнийского университета Леоном Чуа в 1983 году. Цепь состоит из двух ёмкостей, одной индуктивности, линейного сопротивления и нелинейного сопротивления (обычно называемого диодом Чуа):



G,L,C1,C2-пассивные элементы, g-диод Чуа. В классическом варианте предлагаются следующие значения параметров элементов: L=1/7Гн; G=0.7См; C1=1/9Ф; C2=1Ф.

Модель диода Чуа в схеме - это кусочно-линейная вольт-амперная характеристика с участком отрицательного сопротивления – тока через диод *Ig* от напряжения на диоде, равного напряжению на емкости C1 (*Uc1*).



Теоретически фазовые портреты (зависимости переменных от их производных, например, *Uc1(dUc1/dt)*) имеют два аттрактора. Система ОДУ для схемы Чуа имеет вид:

$C1(dUc1/dt) = G(Uc2-Uc1)-Ig(Uc1)$

$C2(dUc2/dt) = G(Uc1-Uc2)+I_L$

$L(dI_{L/dt}) = -Uc2$

Функция funkl для реализации любых кусочно-линейных зависимостей и интегрирования функций с такими зависимостями использует параметры nbad=1 и nbab =2 и не может быть реализована в известных программах-решателях систем ОДУ-ДАУ, реализующих неявные методы интегрирования этих систем (задача task03):

```
//task03 - chua circuit
      double r,xu[6],yu[6],ikl,epx,epy;
      int indp,inj;
void funkl(double x,double *y,int n,double epsx,double epsy,double *rjyx,int
inrj,int *nbad1)
{
      // injr - если равно 1, то это признак численного расчета частной
производной rjyx
static      int ind,ivoz,iii;
static      double dx,a,b,epx,epm,epy;
      ind=1;
      for (iii=1;iii<=n;iii++) {
            if((xu[iii+1])>x) goto m10;
            ind=ind+1;
      }
m10:
      if(ip[1]!=0) goto m5;
      if(inrj!=0) goto m5;
      indp=ind;
m5:
      dx=xu[indp]-xu[indp+1];
      if(fabs(dx)<1e-200) goto m1000;
      a=(yu[indp]-yu[indp+1])/dx;
      b=(yu[indp]*xu[indp+1]-yu[indp+1]*xu[indp])/(-dx);
      *y=a*x+b;
      *rjyx=a;
      epx=fabs(xu[indp+1]-x);
      epm=fabs(xu[indp+1]);
      if(ind<indp) epx=fabs(xu[indp]-x);
      if(ind<indp) epm=fabs(xu[indp]);
      if(epm>1e-200) epx=epx/epm;
      epy=fabs(yu[indp+1]-*y);
      epm=fabs(yu[indp+1]);
      if(ind<indp) epy=fabs(yu[indp]-*y);
      if(ind<indp) epm=fabs(yu[indp]);
      if(epm>1e-200) epy=epy/epm;
      if(inrj!=0) return;
      ivoz=0;
      if((ind!=indp)&&(epx>epsx)&&(epy>epsy)) ivoz=2;
      if(abs(ind-indp)>1) ivoz=2;
      if((*nbad1!=1)&&(ivoz==2)) *nbad1=2;
      return;
m1000:    *y=0e0;
      *rjyx=0e0;
      *nbad1=1;
}
```

Программы-функции fct и out:

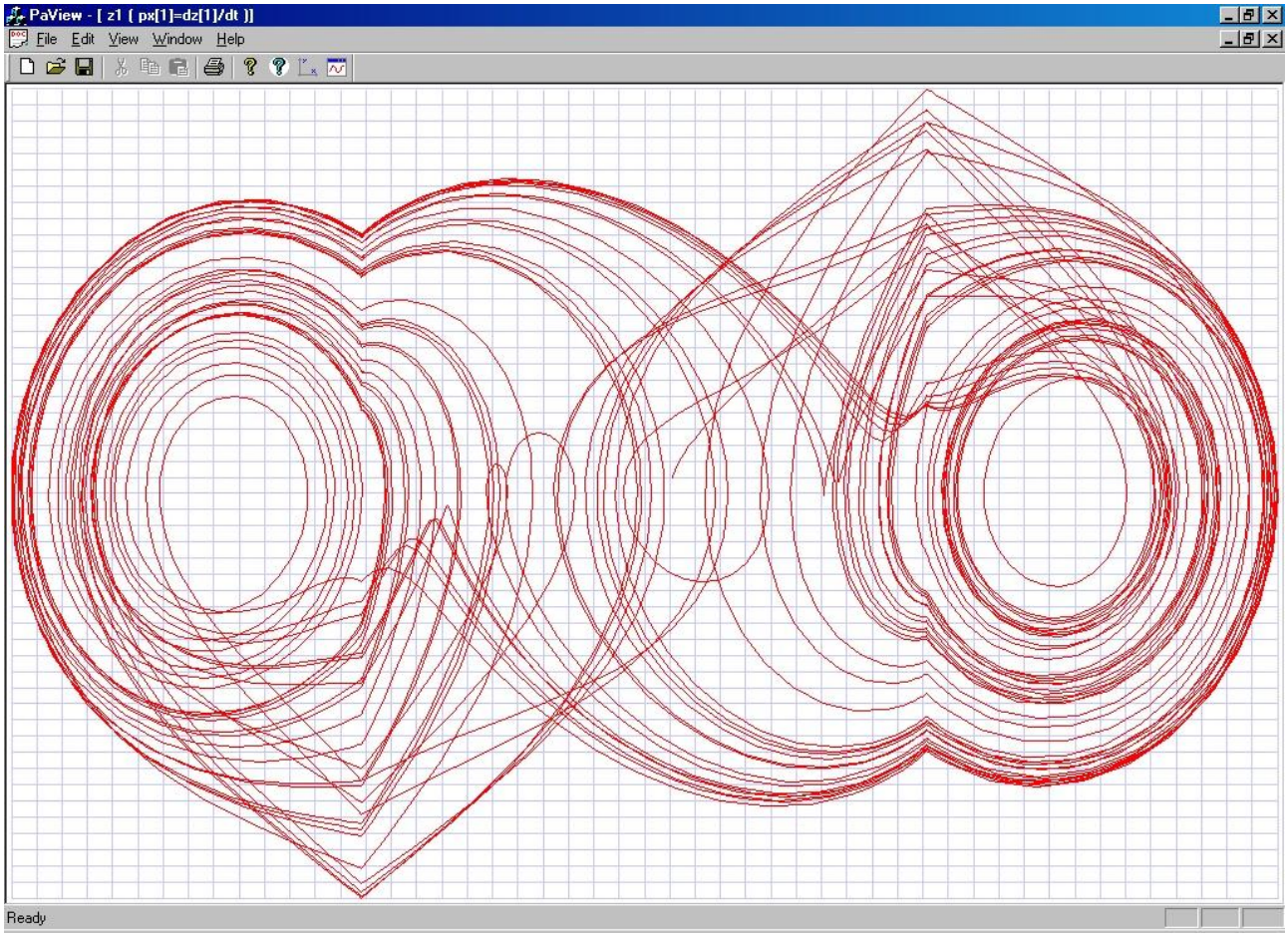
```
void fcttask03(double z[],double px[],double f[],double rj1[],double rj2[],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
double rjv,ikl1;
int nbad1;
nbad1=*nbad;
funkl(z[1],&ikl1,n,epx,epy,&rjv,inj,&nbad1);
*nbad=nbad1;
            ikl=ikl1;
      f[1]=c1*px[1]-z[2]/r+z[1]/r+ikl;
            rj2[1*n+2]=-1e0/r;
            rj2[1*n+1]=1e0/r+rjv;
      f[2]=c2*px[2]-z[1]/r+z[2]/r-z[3];
            rj2[2*n+1]=-1e0/r;
            rj2[2*n+2]=1e0/r;
```

```
            rj2[2*n+3]=-1e0;
      f[3]=l*px[3]+z[2];
            rj2[3*n+2]=1e0;
            rj1[1*n+1]=c1;
            rj1[2*n+2]=c2;
            rj1[3*n+3]=l;
}
void outtsk03(double z[],double px[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
      if(ncon==0) fprintf(f03,"chua-circuit,t-time,z1,z2,z3,px[1]=dz[1]/dt\n");
       fprintf(f03,"  %e   %e  %e  %e  %e\n",t,z[1],z[2],z[3],px[1]);
}
```

Основная программа-функция имеет вид:

```
//task03 - chua circuit - chaos test with piecewise-linear approximation of base
function.
      f01=fopen("task03.rez","wt");
      f02=fopen("grtask03.rez","wt");
            r=10e0/7e0;
            c1=1e0/9e0;
            c2=1e0;
            l=1e0/7e0;
            xu[1]=-21e0;
            xu[2]=-11e0;
            xu[3]=-1e0;
            xu[4]=1e0;
            xu[5]=11e0;
            xu[6]=21e0;
            yu[1]=0e0;
            yu[2]=5.8e0;
            yu[3]=0.8e0;
            yu[4]=-0.8e0;
            yu[5]=-5.8e0;
            yu[6]=0e0;
            epx=1e-3;
            epy=1e-3;
inj=0;//аналитический расчет частной производной в функции funkl для Якобиана
            nm=3;
            eps=1e-3;
            fprintf(f01,"        relative tolerance - eps=%e\n",eps);
            fprintf(f01,"        number of method - nm=%d\n",nm);
            t0=0e0;
            tk=261.2e0;
            hmn=1e-10;
            hmx=tk;
            n=3;
            m=3;
            z[1]=1e-1;
            z[2]=0e0;
            z[3]=0e0;
            z1[1]=fabs(z[1]);
            z1[2]=fabs(z[2]);
            z1[3]=fabs(z[3]);
manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,&ier,
ip,fcttask03,outtask03);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
            fprintf(f01,"\n");
            for(i=1;i<=16;i++)printf("%d ",ip[i]);
            printf("\n");
            fclose(f01);
            fclose(f02);
```

Было получено достоверное решение этой задачи методом М3, т.к. график зависимости z[1](px[1]) показал два аттрактора как по теории:



Метод M1 не смог получить достоверное решение этой задачи при точности интегрирования eps=0.001. По результатам решения данных уравнений можно сделать вывод, аналогичный вышеприведенному выводу по результатам решения уравнения Дуффинга. Кроме того известные программы-решатели систем ОДУ-ДАУ, реализующие неявные методы интегрирования, не имеют средств идентификации точек разрыва частных производных в матрице Якоби системы ДАУ, поэтому данная задача может быть решена только известными программами-решателями систем ОДУ-ДАУ, реализующими явные методы интегрирования, поскольку эта задача с не жесткой системой ОДУ (аналогичное решение было получено явным методом Рунге-Кутты-Мерсона).

ier - код ошибки:

ier = 0 - нет ошибок;
ier = 1 - m отрицательно или больше n;
ier = 2 - hmn отрицательно или больше hmx;
ier = 3 – t0 больше tk;
ier = 4 - eps меньше 1e-12;
ier = 5 - nm меньше 0, или больше 3;
ier = 6 - ncon меньше 0, или больше 2;
ier = 7 - нет сходимости итераций при решении нелинейных алгебраических уравнений на минимальном шаге интегрирования hmin. Следует проверить правильность формул для вычисления элементов матриц rj1 и rj2, уменьшить заданный минимальный шаг интегрирования или проверить является ли устойчивой исходная система ДАУ.
ier = 8 - Вырожденность системы ЛАУ при выполнении Ньютоновских итераций.
ier=-1 – Это значение параметра используется для установки ключа, обеспечивающего численное вычисление элементов матрицы rj2.

ip - рабочий массив размером не менее, чем (20+2*n+6*m), в первых 20-ти элементах этого массива размещаются специальные счетчики и ключи для методов интегрирования:
ip(1) - номер итерации (начиная с 0) при решении системы нелинейных алгебраических уравнений на текущем шаге интегрирования (не более 10-ти).
ip (2) - суммарное количество итераций при решении нелинейных алгебраических уравнений на всех выполненных и аннулированных шагах интегрирования.

ip (3) - суммарное количество выполненных шагов интегрирования.
ip (4) - суммарное количество делений шага интегрирования с отменой выполненного предыдущего шага, т.е. суммарное количество фактически аннулированных шагов.

Условно аннулированными называются шаги h = hmin, которые выполняются несмотря на то, что текущая погрешность превышает заданную при nbad=2.

ip (5) - суммарное количество аннулированных шагов из-за превышения заданной погрешности eps, включая условно аннулированные.
ip (6) - суммарное количество аннулированных шагов из-за не сходимости итераций при решении алгебраических уравнений на каждом шаге интегрирования, включая условно аннулированные.
ip (7) - суммарное количество выполненных итераций по методу M2.
ip (8) - суммарное количество выполненных шагов интегрирования по методу M2.
ip (9) - суммарное количество выполненных итераций по методу M3.
ip (10) - суммарное количество выполненных шагов интегрирования по методу M3.
ip (11) – плохая обусловленность системы..
ip (12) – количество коррекций производных для методов 2 и 4 порядка точности.
ip (13) – уменьшение шага из-за prwar и acr.
ip (14) – уменьшение шага из-за nbad=1.
ip (15) - уменьшение шага из-за превышения числа итераций больше 10.
ip (16) -уменьшение шага из-за того, что невязка и приращения не уменьшаются.
ip (17)}
... } - резерв счетчиков и ключей.
ip (20)}
ip (21) }
... } - рабочий массив для подпрограммы решения
... } системы линейных алгебраических уравнений.
ip (20+2*n+6*m)}

fct - имя внешней подпрограммы пользователя. В этой подпрограмме пользователь вычисляет вектор-функцию *G* (массив f[]) решаемой системы ДАУ и матрицу *RJ* (*RJ1=dG/dXP, RJ2=dG/dZ*: массивы rj1[] и rj2[] частных производных df/dpx и df/dz соответственно). Нулевые значения элементов матриц rj1 и rj2 в подпрограмме fct присваивать не нужно. К подпрограмме fct происходит обращение на каждой итерации. При первом обращении к решателю (ключ ncon=0 или ncon=2) организовано одно обращение к подпрограмме fct при t=t0 и h=hmin для выполнения однократных вычислений параметров, не изменяющихся в ходе интегрирования. После выполнения однократных вычислений устанавливается ключ ncon=1. При каждом обращении в массив f заносится рекомендуемый вектор приращений dz для переменных z с целью численного вычисления частных производных df(i)/dz(j) = (fi(zj+dzj)-fi(dzj))/dzj; Эту возможность следует использовать только при невозможности аналитического вычисления частных производных. Далее приведен пример численного вычисления одного из элементов матрицы Якоби rj2 для нелинейной системы ДАУ с известным аналитическим решением - NONLINEAR EQUATIONS WITH ANALITIC SOLUTION (task04). В функции fct элементы матрицы Якоби для 1,3 и 4 уравнений системы ДАУ вычисляются аналитически, а элементы матрицы Якоби для 2-го уравнения вычисляются численно, методом приращений. Т.к. циклы выполняются, начиная с 1, то размеры массивов должны быть, как минимум на n больше (5 – в данной задаче), ri – рабочая переменная для какой-либо нелинейной функции (в данной задаче функции заданы в явном виде, но возможны и неявные функции, вычисляемые путем обращения к другим не рекурсивным функциям), `rp` – рабочая переменная для вычисления соответствующей функции с учетом приращения соответствующнего аргумента, rizi – вычисленная частная производная нелинейной функции ri по соответствующему аргументу zi.

Программы-функции fct и out:

```
//task04 - NONLINEAR EQUATIONS WITH ANALITIC SOLUTION
```

```
void fcttask04(double z[],double px[],double f[],double rj1[],double rj2[],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
int i;
double dz[5],zp[5],r1,rp,r1z1,r1z4,r2,r2z1,r2z4;
      for(i=1;i<=4;i++){
            dz[i]=f[i];
            zp[i]=z[i]+dz[i];
      }
            r1=2e0*t*z[4]*z[1];
            r1z1=2e0*t*z[4];
            r1z4=2e0*t*z[1];
      f[1]=px[1]-r1;
            rj1[1*n+1]=1e0;
            rj2[1*n+1]=-r1z1;
            rj2[1*n+4]=-r1z4;
            r2=10e0*t*z[4]*(z[1]*z[1]*z[1]*z[1]*z[1]);
            rp=10e0*t*z[4]*(zp[1]*zp[1]*zp[1]*zp[1]*zp[1]);
            r2z1=(rp-r2)/dz[1];
            rp=10e0*t*zp[4]*(z[1]*z[1]*z[1]*z[1]*z[1]);
            r2z4=(rp-r2)/dz[4];
      f[2]=px[2]-r2;
            rj1[2*n+2]=1e0;
            rj2[2*n+1]=-r2z1;
            rj2[2*n+4]=-r2z4;
      f[3]=px[3]-2e0*t*z[4];
            rj1[3*n+3]=1e0;
            rj2[3*n+4]=-2e0*t;
      f[4]=px[4]+2e0*t*(z[3]-1e0);
            rj1[4*n+4]=1e0;
            rj2[4*n+3]=2e0*t;
      return;
}
void outtask04(double z[],double px[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
int i;
double za[5],ep[5],em[5],d;
      za[1]=exp(sin(t*t));
      za[2]=exp(5e0*(sin(t*t)));
      za[3]=sin(t*t)+1e0;
      za[4]=cos(t*t);
for(i=1;i<=4;i++){
      d=1e0;
      ep[i]=abs(za[i]-z[i])/d;
      if(em[i]>ep[i]) goto m3;
      em[i]=ep[i];
m3:;
}
if(t==tk) fprintf(f01,"  t= %e   z1= %e  z2= %e  z3= %e  z4=
%e\n",t,z[1],z[2],z[3],z[4]);
if(t==tk) fprintf(f01,"  t= %e   em1= %e  em2= %e  em3= %e  em4=
%e\n",t,em[1],em[2],em[3],em[4]);
if(ncon==0) fprintf(f02,"       NONLINEAR EQUATIONS WITH ANALITIC SOLUTION,t-
time,z1,z2,z3,z4\n");
fprintf(f02,"  %e   %e  %e  %e  %e\n",t,z[1],z[2],z[3],z[4]);
      return;
}
```

Основная программа-функция имеет вид:

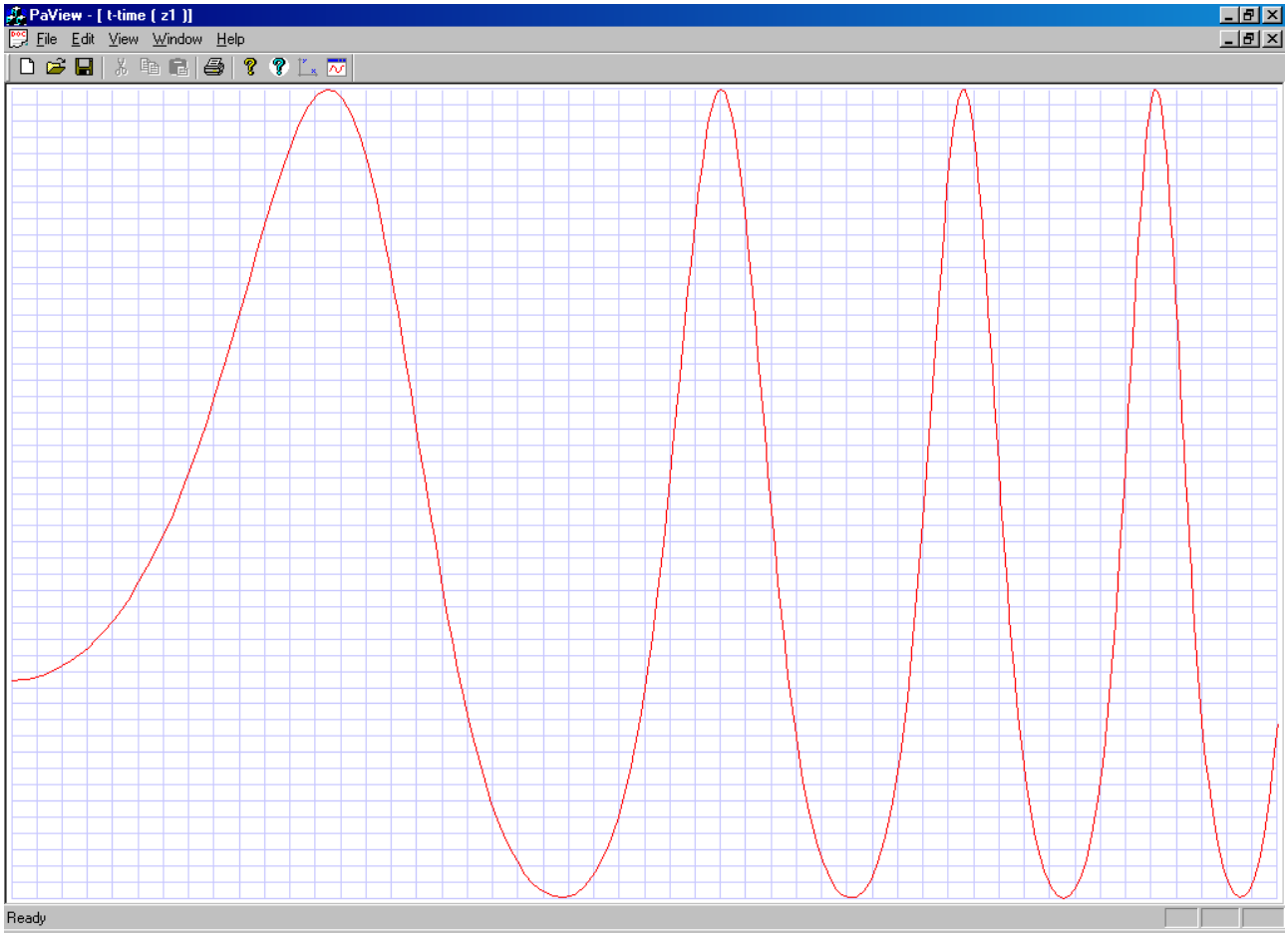
```
//task04 - NONLINEAR EQUATIONS WITH ANALITIC SOLUTION
      f01=fopen("task04.rez","wt");
      f02=fopen("grtask04.rez","wt");
            eps=1e-3;
            fprintf(f01,"        relative tolerance - eps=%e\n",eps);
            nm=3;
            fprintf(f01,"        number of method - nm=%d\n",nm);
            t0=0e0;
```

```
            tk=5e0;
            hmn=1e-9;
            hmx=tk/10e0;
            n=4;
            m=4;
            z[1]=1e0;
            z[2]=1e0;
            z[3]=1e0;
            z[4]=1e0;
            z1[1]=abs(z[1]);
            z1[2]=abs(z[2]);
            z1[3]=abs(z[3]);
            z1[4]=abs(z[4]);
      manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad
,&ier,ip,fcttask04,outtask04);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
            fprintf(f01,"\n");
            for(i=1;i<=16;i++)printf("%d ",ip[i]);
            printf("\n");
            fclose(f01);
            fclose(f02);
```

Было получено достоверное решение с заданной точностью (em[i] – значение максимальной погрешности интегрирования для каждой переменной на заданном отрезке интегрирования), Количество шагов интегрирования в данном случае оказалось таким же, как при аналитическом вычислении матрицы Якоби и при автоматическом численном вычислении всех элементиов матрицы Якоби с параметром ier=-1:

```
        relative tolerance - eps=0.001

        number of method - nm=3
  t= 5.000000e+000   z1= 8.760108e-001  z2= 5.176173e-001  z3= 8.676297e-001
z4= 9.911955e-001
  t= 5.000000e+000   em1= 3.624009e-005  em2= 5.683132e-003  em3= 1.858172e-005
em4= 1.778772e-005
ier= 0
4 1160 285 9 9 0 0 0 1160 285 0 0 0 0 0 0
```

График зависимости первой переменной от времени имеет вид:



Метод M1 не смог получить достоверное решение этой задачи при заданной точности интегрирования eps=0.001. По результатам решения данных уравнений можно сделать вывод, аналогичный вышеприведенному выводу по результатам решения уравнений Дуффинга, орбиты Луны и схемы Чуа. Кроме того известные стандартные программы-решатели систем ОДУ-ДАУ из библиотек математических программ на языке Си не предлагают средств для численного вычисления отдельных элементов матрицы Якоби.

Общий заголовок функции fct:

void fct(double z[],double px[],double f[],double rj1[],double rj2[],int n,int m,double t,double h,int ncon,int *nbad,int ip[]);

out - имя внешней подпрограммы пользователя для вывода и обработки результатов, в том числе и для табуляции результатов интегрирования (см. соответствующие примеры выше). Обращение к этой подпрограмме происходит:

а) после расчета начальных значений переменных px0 и y0 для момента времени t0. При этом перед обращением к подпрограмме out устанавливается ключ ncon=0. Это можно использовать для однократных вычислений параметров подпрограммы out, которые не изменяются в ходе интегрирования.

б) после каждого успешно выполненного шага интегрирования;

в) в случае ошибки (ier не равно 0).

бщий заголовок функции out:

void out(double z[],double px[],int n,int m,double t,double t0,double tk,double h,double tkv,double ar[],int ncon,int ip[]);

Задаваемые параметры интегрирования:

Перед обращением к программе должны быть заданы значения следующих параметров, ключей и переменных:

n - общее число уравнений в системе ДАУ;

m - количество дифференциальных переменных в системе ДАУ;
t0 - заданное время начала интегрирования;
tk - заданное время окончания интегрирования;
hmn - заданный минимальный шаг интегрирования;
hmx - заданный максимальный шаг интегрирования;
eps - заданная относительная погрешность интегрирования,
одинаковая для всех дифференцируемых переменных;
nm - заданный номер метода интегрирования;
ncon - ключ расчета начальных значений переменных;
ier = -1 – ключ для установки автоматического численного вычисления
элементов матрицы rj2.
z(1)...z(m) - начальные значения дифференциальных переменных.
z1(1)...z1(m) - либо максимальные абсолютные значения дифференциальных
переменных, либо z1(i)=abs(z(i)).

Тестирование программ-решателей систем ДАУ

В ДАУ-решателе (DAE Solver) manzhuk реализованы принципиально новые методы и алгоритмы, изложенные в работе [5]. Результаты тестирования предыдущей версии dae-solver-01 программы-решателя систем ДАУ (реализованной в рамках Си библиотеки SADEL) и сравнение с лучшими программами-решателями для решения жестких систем ОДУ из пакетов и Си библиотек математических программ были приведены в работе [8].

Рассмотрим тестовые задачи подробнее. Тестовые задачи должны быть достоверно и точно решены программами-решателями систем ОДУ-ДАУ при заданной точности интегрирования eps=0.001. Все решения ниже были получены в программе manzhuk.cxx с другими параметрами интегрирования, заданными по умолчанию (все тестовые задачи были решены за несколько секунд на Intel i5, 2Gb, Win7-x64&VS2008). После написания отдельного программного продукта разработчику необходимо провести его тестирование для выявления ошибок и определения, удовлетворяет ли продукт поставленным требованиям. Тестирование программ моделирования динамических систем проводится в двух направлениях: тестирование на конкретных известных практических задачах (электрические, механические, гидравлические динамические системы и т.д.) и тестирование математического ядра (программы-решателя систем ОДУ-ДАУ). Поскольку основные характеристики программы-решателя (достоверность результатов, точность результатов и машинное время, затраченное на получение решения) зависят от математического ядра, его тестирование должно показывать надежность и эффективность программ моделирования динамических систем. В настоящее время первое направление развито достаточно широко, а второе применяется крайне редко, несмотря на то, что оно необходимо для дальнейшего усовершенствования существующих программ моделирования динамических систем и разработки новых. Математическое ядро программ моделирования динамических систем представляет собой блок решения систем ОДУ-ДАУ (ODE-DAE-solver). Все задачи для тестирования математического ядра программ моделирования динамических систем условно можно разделить на «простые» и «трудные». С «простыми» задачами большинство программ моделирования динамических систем справляются без особых затруднений. К ним, например, относятся большинство не жестких систем ОДУ, имеющих решениями функции с плавно меняющимися свойствами. К «трудным» относятся задачи, вызывающие определенные сложности при их решении программами моделирования динамических систем. В данном описании практически всегда применялись «трудные» задачи, поскольку «простые» для тестирования представляют малый интерес. В качестве тестовых задач по возможности выбирались системы ОДУ-ДАУ, имеющие аналитическое решение.

Графики различных зависимостей для рассчитанных переменных были получены с помощью программы pa10_view.exe. Эта программа разработана только для построения графиков функций, полученных решателем manzhuk:

Формат файла результатов для этой программы:

1-я строка файла результатов - заголовок задачи, (запятая) время, (запятая) и далее через запятую имена ровно 4-х переменных.

Следующие строки - это результаты решения ДАУ на каждом шаге интегрирования.

Примеры вывода результатов расчета переменных ДАУ в этом формате приведены в соответствующих тестах: файлы grtest01, grtest02 …

Тест00. Линейная система ОДУ с постоянными коэффициентами слабо (a=0.001) и сильно (a=0.999) связанными компонентами (test00) [9]. Эта задача относится к классу «простых».

Исходная система:

$$x'_1(t) = \frac{1}{1-a}(\lambda_1 - a\lambda_2)x_1 + \gamma x_2 + \frac{1}{1-a}(\beta + a\gamma - \lambda_1)x_3$$

$$x'_2(t) = \frac{a}{1-a}(\lambda_1 - \lambda_2)x_1 + \beta x_2 + \frac{1}{1-a}(\beta + a\beta - a\lambda_1)x_3$$

$$x'_3(t) = \frac{a}{1-a}(\lambda_1 - \lambda_2)x_1 + \gamma x_2 + \frac{1}{1-a}(\beta + a\gamma - a\lambda_1)x_3$$

Аналитическое решение:

$$x_1(t) = c_1 e^{\lambda_1 t} + c_2 e^{\lambda_2 t} + c_3 e^{\lambda_3 t}$$

$$x_2(t) = c_1 a e^{\lambda_1 t} + c_2 e^{\lambda_2 t} - c_3 e^{\lambda_3 t}$$

$$x_3(t) = c_1 a e^{\lambda_1 t} + c_2 e^{\lambda_2 t} + c_3 e^{\lambda_3 t}$$

Интервал интегрирования: [0,10]

Параметры:

$$\lambda_1 = -10^5; \quad \lambda_2 = -1; \quad \lambda_3 = -10^2; \quad \beta = 0.5(\lambda_2 + \lambda_3); \quad \gamma = 0.5(\lambda_2 - \lambda_3)$$

$$c_1 = c_3 = 1; \quad c_2 = 1.5;$$

Программы-функции fct и out:

```
//test00 Linear equations with analitic solutions
double c01,c02,c03,l01,l02,l03,a;
double b,g,a1,a11,a12,a13,a21,a22,a23,a31,a32,a33;
void fcttest00(double z[],double px[],double f[],double rj1[],double rj2[],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
      f[1]=px[1]-a11*z[1]-a12*z[2]-a13*z[3];
      f[2]=px[2]-a21*z[1]-a22*z[2]-a23*z[3];
      f[3]=px[3]-a31*z[1]-a32*z[2]-a33*z[3];
            rj1[1*n+1]=1;
            rj1[2*n+2]=1;
            rj1[3*n+3]=1;
            rj2[1*n+1]=-a11;
            rj2[1*n+2]=-a12;
            rj2[1*n+3]=-a13;
            rj2[2*n+1]=-a21;
            rj2[2*n+2]=-a22;
            rj2[2*n+3]=-a23;
            rj2[3*n+1]=-a31;
            rj2[3*n+2]=-a32;
            rj2[3*n+3]=-a33;
return;
}
void outtest00(double z[],double px[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
double em[4],ea[4],pa[4];
double zr1,zr2,zr3;
int io;
```

```
            if(ncon!=0) goto m1;
      fprintf(f01,"time \nx(1) eps x(1) maxeps x(1) \nx(2) eps x(2) maxeps x(2)
\nx(3) eps x(3) maxeps x(3)\n");
      for(io=1;io<=3;io++) em[io]=0.;
m1:
            zr1=exp(l01*t);
            zr2=exp(l02*t);
            zr3=exp(l03*t);
            if((l01*t)<-70) zr1=0.;
            if((l02*t)<-70) zr2=0.;
            if((l03*t)<-70) zr3=0.;
            ea[1]=c01*zr1+c02*zr2+c03*zr3;
            ea[2]=c01*a*zr1+c02*zr2-c03*zr3;
            ea[3]=c01*a*zr1+c02*zr2+c03*zr3;
      for (io=1;io<=n;io++){
            pa[io]=fabs(z[io]-ea[io]);
            if(em[io]>pa[io]) goto m3;
            em[io]=pa[io];
m3:;
      }
      if(t==tk){
            fprintf(f01,"\nt=%14.5e\n",t);
            for(io=1;io<=n;io++)fprintf(f01,"z[i]=%14.5e ea[i]=%14.5e
em[i]=%14.5e\n",z[io],pa[io],em[io]);
      }
       return;
```

Основная программа:

```
//test00 - linear ODE with analitic solution
int ii,iii;
      f01=fopen("test00.rez","wt");
      for (ii=1;ii<=2;ii++) {
            if(ii==1) a=0.001;
            if(ii==2) a=0.999;
            c01=1.;
            c02=1.5;
            c03=1.;
            l01=-1e5;
            l02=-1.;
            l03=-1e2;
      fprintf(f01,"\nii= %d\n",ii);
      fprintf(f01,"a=%14.5e\nc1=%14.5e c2=%14.5e c3=%14.5e\n l1=%14.5e l2=%14.5e
l3= %14.5e\n",a,c01,c02,c03,l01,l02,l03);
            b=0.5*(l02+l03);
            g=0.5*(l02-l03);
            a1=1./(1.-a);
            a11=a1*(l01-a*l02);
            a12=g;
            a13=a1*(b+a*g-l01);
            a21=a*a1*(l01-l02);
            a22=b;
            a23=a1*(g+a*b-a*l01);
            a31=a*a1*(l01-l02);
            a32=g;
            a33=a1*(b+a*g-a*l01);
            eps=1e-3;
      fprintf(f01,"        relative tolerance - eps=%e\n",eps);
            nm=3;
      fprintf(f01,"        number of method - nm=%d\n",nm);
            t0=0.;
            tk=10.;
            hmn=1e-10;
            hmx=tk;
            n=3;
            m=3;
            z[1]=c01+c02+c03;
            z[2]=c01*a+c02-c03;
            z[3]=c01*a+c02+c03;
```

```
            z1[1]=c01+c02+c03;
            z1[2]=c01*a+c02-c03;
            z1[3]=c01*a+c02+c03;
manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,&ier,
ip,fcttest00,outtest00);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
            fprintf(f01,"\n");
            for(i=1;i<=16;i++)printf("%d ",ip[i]);
            printf("\n");
}
            fclose(f01);
```

Результаты решения методом М3 дали намного более точные результаты (переменная em), чем заданная точность интегрирования (eps=0.001):

```
ii= 1
a=  1.00000e-003
c1=  1.00000e+000 c2=  1.50000e+000 c3=  1.00000e+000
 l1= -1.00000e+005 l2= -1.00000e+000 l3=  -1.00000e+002
        relative tolerance - eps=0.001

        number of method - nm=3
time
x(1) eps x(1) maxeps x(1)
x(2) eps x(2) maxeps x(2)
x(3) eps x(3) maxeps x(3)

t=  1.00000e+001
z[i]=  6.81373e-005 ea[i]=  3.73806e-008 em[i]=  1.13611e-006
z[i]=  6.81373e-005 ea[i]=  3.73802e-008 em[i]=  6.65607e-007
z[i]=  6.81373e-005 ea[i]=  3.73802e-008 em[i]=  6.65607e-007
ier= 0
2 260 126 0 0 0 0 0 256 126 0 0 0 0 0 0

ii= 2
a=  9.99000e-001
c1=  1.00000e+000 c2=  1.50000e+000 c3=  1.00000e+000
 l1= -1.00000e+005 l2= -1.00000e+000 l3=  -1.00000e+002
        relative tolerance - eps=0.001

        number of method - nm=3
time
x(1) eps x(1) maxeps x(1)
x(2) eps x(2) maxeps x(2)
x(3) eps x(3) maxeps x(3)

t=  1.00000e+001
z[i]=  6.81630e-005 ea[i]=  6.30860e-008 em[i]=  1.84925e-006
z[i]=  6.81630e-005 ea[i]=  6.30860e-008 em[i]=  1.84929e-006
z[i]=  6.81630e-005 ea[i]=  6.30860e-008 em[i]=  1.84925e-006
ier= 0
4 539 130 0 0 0 0 0 535 130 0 0 0 0 0 0
```

Тест01. Пример расчета жесткой системы ОДУ 2-го порядка (MU – параметр жесткости) – тест Ван дер Поля [10].

$$dx_1/dt = x_2$$

$$dx_2/dt = -x_1 + MU\cdot(1-x^2_1)\cdot x_2$$

$$x_1(0) = -1, x_2(0) = 1,$$

$$t \in (0, 4.2\cdot MU)$$

Сравнение программ в [7] было проведено для часто встречающихся на практике значений жесткости MU=10^6 и MU=10^9. Решение ниже получено с параметром жесткости MU=10^{20}.

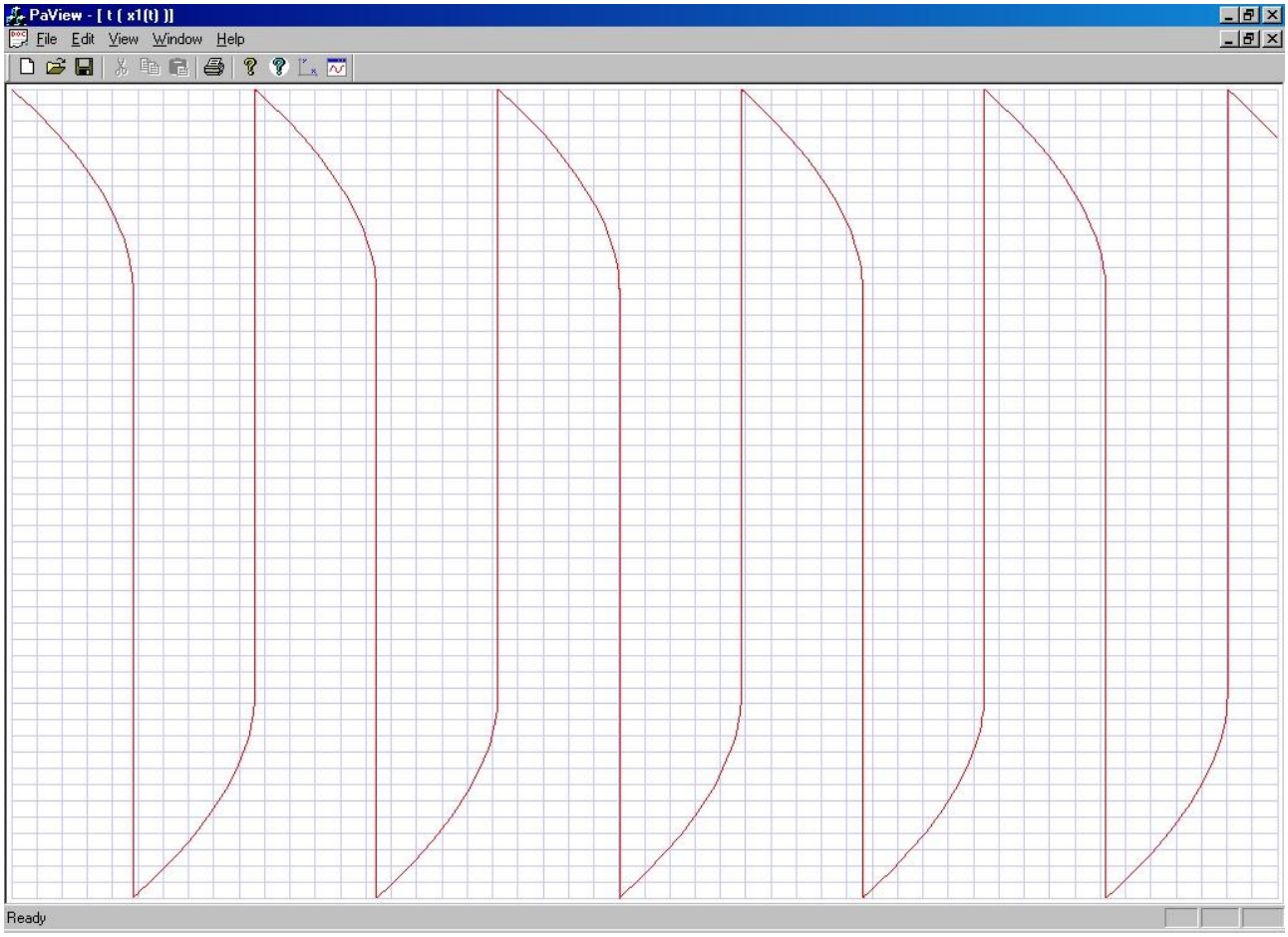
Программы rct и out:

```
//test01 - Van der Pol equations
  double mu;
void fct01(double z[],double px[],double f[],double rj1[],double rj2[],int n,int
m,double t,double h,int ncon,int *nbad,int ip[])
{
f[1]=px[1]-z[2];
rj1[1*n+1]=1e0;
rj2[1*n+2]=-1e0;
f[2]=px[2]-mu*(1e0-z[1]*z[1])*z[2]+z[1];
rj1[2*n+2]=1e0;
rj2[2*n+1]=-mu*z[2]*(-2e0*z[1])+1e0;
rj2[2*n+2]=-mu*(1e0-z[1]*z[1]);
      return;
}
void out01(double z[],double px[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
if(ncon==0) fprintf(f02,"           Van-der-pol
equations,t,x1(t),px1,x2,px2\n");
fprintf(f02,"  %e   %e  %e  %e  %e\n",t,z[1],px[1],z[2],px[2]);
      return;
}
```

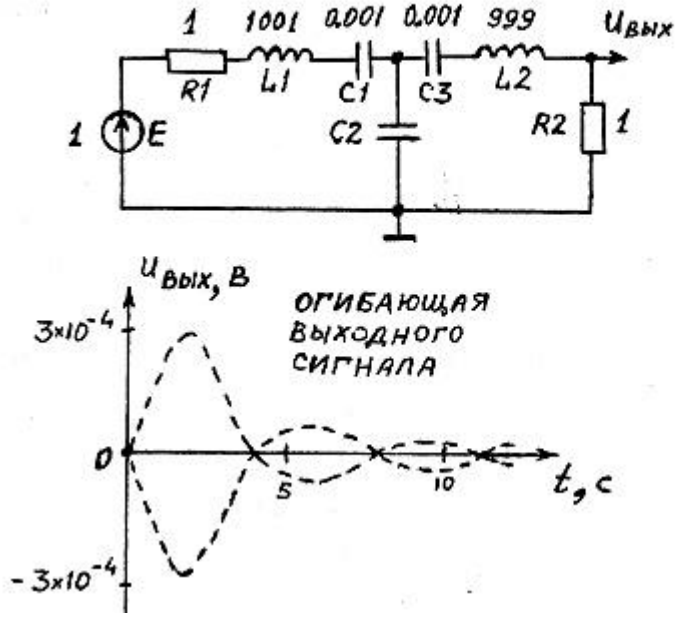
Основная программа

```
//test01 - Van der Pol equations
      f01=fopen("test01.rez","wt");
      f02=fopen("grtest01.rez","wt");
            eps=1e-3;
      fprintf(f01,"        relative tolerance - eps=%e\n",eps);
            mu=1e20;
      fprintf(f01,"        stiffness parameter - MU=%e\n",mu);
            nm=3;
      fprintf(f01,"        number of method - nm=%d\n",nm);
            t0=0e0;
            tk=8.4*mu;
            hmn=1e-12/mu;
            hmx=tk;
            n=2;
            m=2;
            z[1]=2e0;
            z1[1]=2e0;
            z[2]=0e0;
            z1[2]=2e0;
manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,&ier,
ip,fct01,out01);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
            fprintf(f01,"\n");
            for(i=1;i<=16;i++)printf("%d ",ip[i]);
            printf("\n");
            fclose(f01);
            fclose(f02);
```

Результаты решения методом M3:



Тест02. Пример моделирования электрической схемы с многопериодным решением (система ОДУ 5-го порядка) – тест Маничева [11].



Система ОДУ для этого теста имеет вид:

$$kr = ku / ki, kc = kt \cdot ki / ku, kl = kt \cdot ku / ki$$

$$dx_1 / dt = x_4 / 0.001 \cdot kc$$

$$dx_2 / dt = x_5 / 0.001 \cdot kc$$

$$dx_3 / dt = (x_4 - x_5) / kc$$

$$dx_4 / dt = (ku - x_1 - x_3 - kr \cdot x_4) \cdot / 1001 \cdot kl$$

$$dx_5 / dt = (-x_2 + x_3 - kr \cdot x_5) / 999 \cdot kl$$

$$x_1(0) = 0, x_2(0) = 0, x_3(0) = 0, x_4(0) = 0, x_5(0) = 0$$

$$t \in (0,12560 \cdot kt)$$

Программы rct и out:

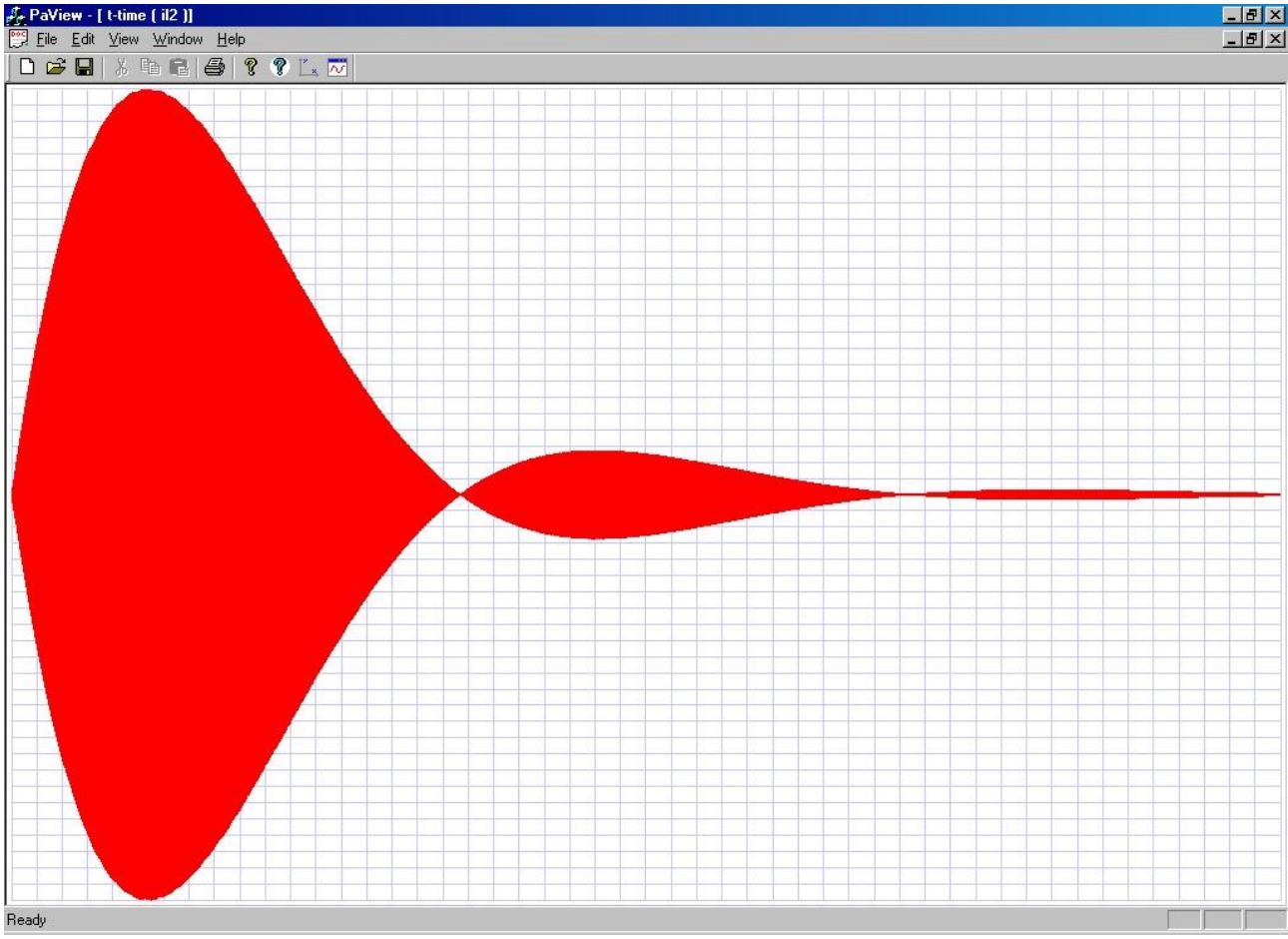
```
//test02 - High Q filter
double c3,l1,l2,l3,r1,r2,e;
double kr,kc,kl,kt,ku,ki;
void fcttest02(double z[],double px[],double f[],double rj1[],double rj2[],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
      f[1]=c1*px[1]-z[4];
            rj1[1*n+1]=c1;
            rj2[1*n+4]=-1e0;
      f[2]=c2*px[2]-z[5];
            rj1[2*n+2]=c2;
            rj2[2*n+5]=-1e0;
      f[3]=c3*px[3]-z[4]+z[5];
            rj1[3*n+3]=c3;
            rj2[3*n+4]=-1e0;
            rj2[3*n+5]=1e0;
      f[4]=l1*px[4]-e+z[1]+z[3]+r1*z[4];
            rj1[4*n+4]=l1;
            rj2[4*n+1]=1e0;
            rj2[4*n+3]=1e0;
            rj2[4*n+4]=r1;
      f[5]=l2*px[5]+z[2]-z[3]+r2*z[5];
            rj1[5*n+5]=l2;
            rj2[5*n+2]=1e0;
            rj2[5*n+3]=-1e0;
            rj2[5*n+5]=r2;
      return;
}
void outtest02(double z[],double px[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
      if(ncon==0) fprintf(f02,"           T11-HIGH Q FILTER,t-
time,il2,uc3,il1,uc1\n");
      fprintf(f02,"  %e   %e  %e  %e  %e\n",t,z[5],z[3],z[4],z[1]);
      return;
}
```

Основная программа

```
//test02 - High Q filter
      f01=fopen("test02.rez","wt");
      f02=fopen("grtest02.rez","wt");
            kt=1e0;
            ku=1e-2;
            ki=1e0;
            pi4=atan(1.0);
      fprintf(f01,"        KT=%e   KU=%e   KI=%e\n",kt,ku,ki);
            kr=ku/ki;
            kc=kt*ki/ku;
```

```
        kl=kt*ku/ki;
    fprintf(f01,"        KR=%e   KC=%e   KL=%e\n",kr,kc,kl);
        e=1e0*ku;
        r1=1e0*kr;
        l1=1001e0*kl;
        c1=0.001e0*kc;
        c2=0.001e0*kc;
        c3=1e0*kc;
        l2=999e0*kl;
        r2=1e0*kr;
    fprintf(f01,"    e= %e    r1= %e   l1= %e    c1= %e\n",e,r1,l1,c1);
    fprintf(f01,"    c2= %e    c3= %e   l2= %e    r2= %e\n",c2,c3,l2,r2);
        eps=1e-3;
    fprintf(f01,"        relative tolerance - eps=%e\n",eps);
        nm=3;
    fprintf(f01,"        number of method - nm=%d\n",nm);
        t0=0e0*kt;
        tk=12560*kt;
        hmn=1e-6*kt;
        hmx=tk;
        n=5;
        m=5;
        for(i=1;i<=5;i++) z[i]=0;
        for(i=1;i<=5;i++) z1[i]=z[i];
manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,&ier,
ip,fcttest02,outtest02);
        fprintf(f01,"ier= %d \n",ier);
        printf("ier= %d \n",ier);
        for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
        fprintf(f01,"\n");
        for(i=1;i<=16;i++)printf("%d ",ip[i]);
        printf("\n");
        fclose(f01);
        fclose(f02);
```

Решение получено для встречающихся на практике параметров масштабных коэффициентов по времени, току и напряжению: kt=1, ki=1, ku=10^{-2}, а также для значений параметров kt=10^{-104}, ki=1, ku=1, которые дают трудный тест для современных решателей систем ОДУ=ДАУ.



Тест03. Нелинейная жесткая система ОДУ, имеющая локально-неустойчивое решение – тест Скворцова [12].

$$dx_1 / dt = x_2$$

$$dx_2 / dt = MU \cdot (1 - x_1^2) \cdot (x_1 + x_2)$$

$$x_1(0) = 2, x_2(0) = 0, t \in (0,3)$$

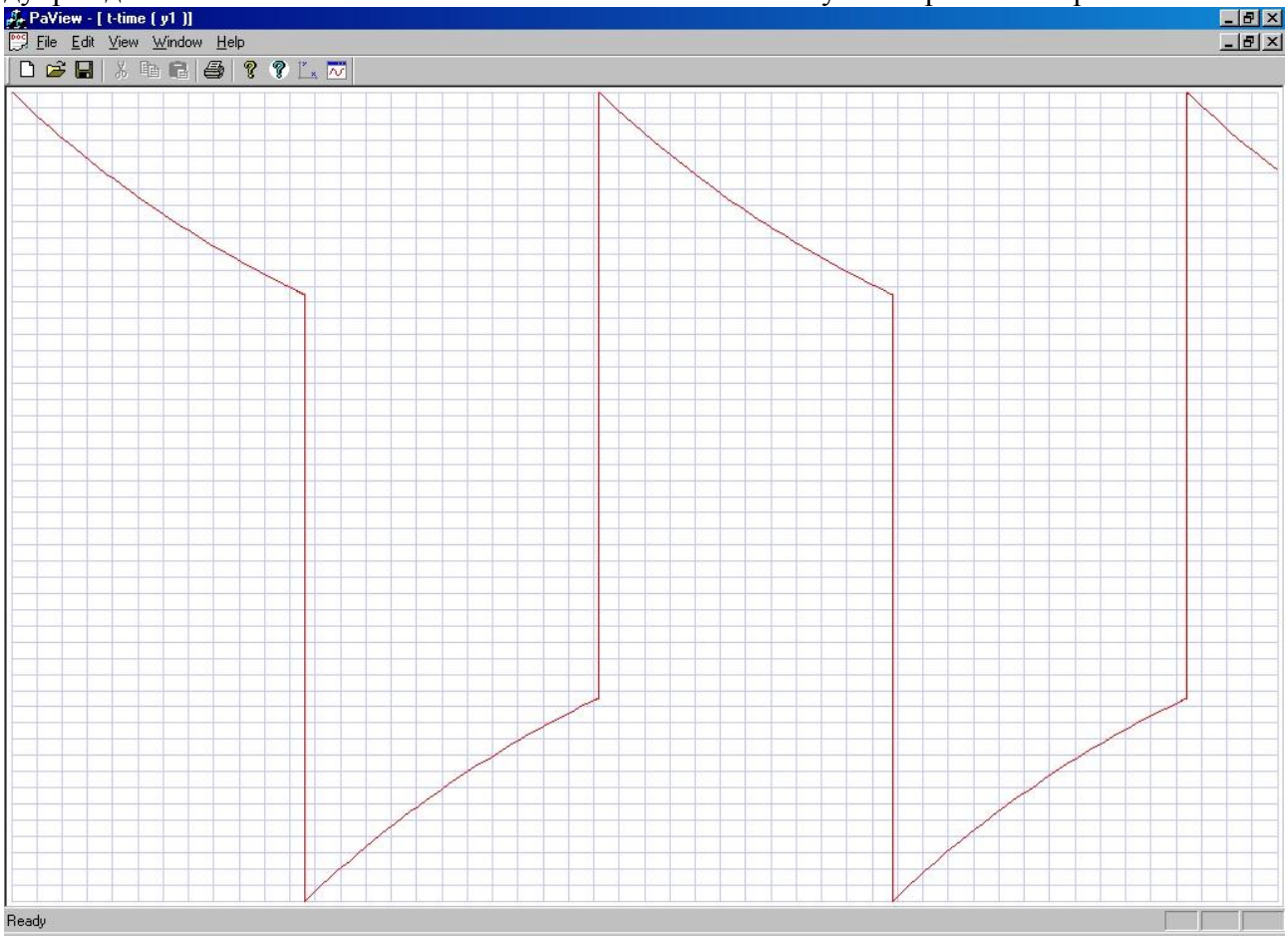
Программы rct и out:

```
//test03 Skvorcov test
void fcttest03(double z[],double px[],double f[],double rj1[],double rj2[5],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
      f[1]=px[1]-z[2];
            rj1[1*n+1]=1e0;
            rj2[1*n+2]=-1e0;
      f[2]=px[2]-1e06*(z[1]+z[2]-z[1]*z[1]*z[1]-z[1]*z[1]*z[2]);
            rj1[2*n+2]=1e0;
            rj2[2*n+1]=-1e6*(1e0-3.0*z[1]*z[1]-2e0*z[2]*z[1]);
            rj2[2*n+2]=-1e6*(1e0-z[1]*z[1]);
      return;
}
void outtest03(double z[],double px[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
      if(ncon==0) fprintf(f02,"           Skvorcov-test,t-
time,y1,xp1,y1,xp1\n");
      fprintf(f02,"  %e   %e   %e   %e   %e\n",t,z[1],px[1],z[1],px[1]);
      return;
}
```

Основная программа

```
//test03 - test Skvorcova
      f01=fopen("test03.rez","wt");
      f02=fopen("grtest03.rez","wt");
            pi4=atan(1.0);
            eps=1e-7;
      fprintf(f01,"        relative tolerance - eps=%e\n",eps);
            nm=3;
      fprintf(f01,"        number of method - nm=%d\n",nm);
            t0=0e0;
            tp=0e0;
            tk=3e0;
            hmn=1e-16;
            hmx=tk/10.;
            n=2;
            m=2;
            z[1]=2e0;
            z1[1]=1e0;
            z[2]=0e0;
            z1[2]=2e0;
manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,&ier,
ip,fcttest03,outtest03);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
            fprintf(f01,"\n");
            for(i=1;i<=16;i++)printf("%d ",ip[i]);
            printf("\n");
            fclose(f01);
            fclose(f02);
```

Сравнение программ было проведено для часто встречающегося на практике значения жесткости MU=10^6. Программы-решатели жестких систем ОДУ-ДАУ из пакета MATLAB дали неверное решение этого теста при параметрах точности по умолчанию (0.001) без предупреждения пользователя об ошибке. В manzhuk было получено правильное решение:



Следует отметить, что после соответствующего выбора и настройки методов интегрирования было получено верное решение этого теста в MATLAB. Таким образом, с помощью

пакетов математических программ (MATLAB, Maple, Mathcad, Mathematica) можно правильно решить практически любую систему ОДУ-ДАУ (после настройки методов и параметров интегрирования, но при этом время расчета может быть очень большим). Главный недостаток пакетов математических программ – эти программы могут выдать неверное решение при стандартной заданной точности интегрирования без предупреждения пользователя. Другой недостаток - пользователь должен быть математиком высокой квалификации, знающим математический английский (MATLAB не локализуется). В ПМК ПА10 системы ОДУ-ДАУ формируются автоматически и пользователи не должны быть математиками высокой квалификации, поэтому программы интегрирования должны выдавать правильное решение при стандартной заданной точности интегрирования, либо предупреждение о возможном неверном решении. В первую очередь это требование должно выполняться для всех известных тестовых задач.

Тест04. Нелинейная жесткая система ОДУ для математического моделирования процессов реального лазера – тест Евстифеева [13].

$$dx_1 / dt = -x_1 \cdot (\alpha \cdot x_2 + \beta) + \gamma$$

$$dx_2 / dt = x_2 \cdot (p \cdot x_1 - \sigma) + \tau \cdot (1 + x_1)$$

$$x_1(0) = -1, x_2(0) = 0, t \in (0, 10^6),$$

$$\alpha = 1.5 \cdot 10^{-18}, \beta = 2.5 \cdot 10^{-6}, \gamma = 2.1 \cdot 10^{-6},$$

$$p = 0.6, \sigma = 0.18, \tau = 0.016$$

Программы fct и out:

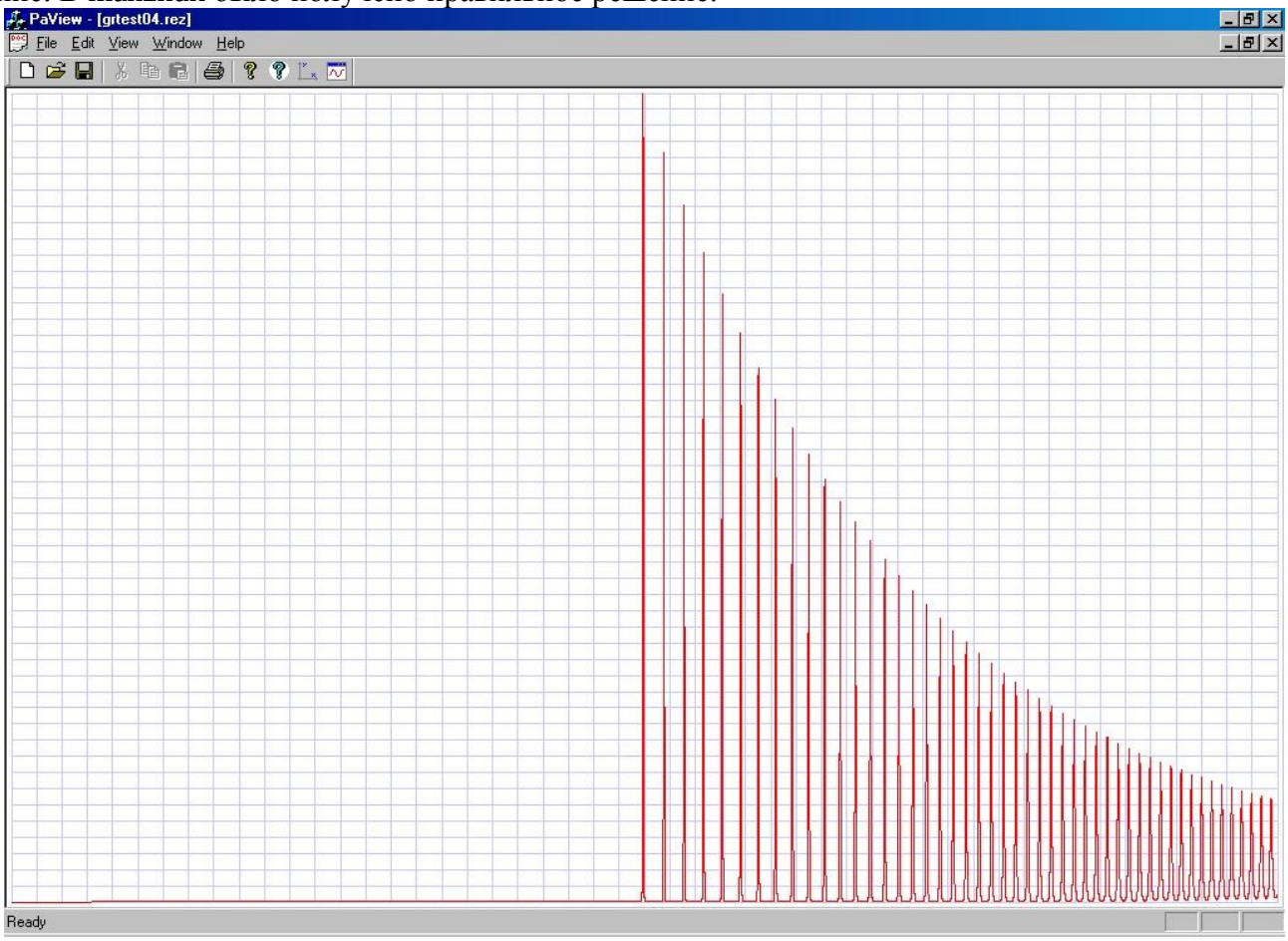
```
//test04 - Evstifeev lazer test
double al,be,ga,p,si,ta;
void fcttest04(double z[],double px[],double f[],double rj1[],double rj2[],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
      f[1]=px[1]+z[1]*(al*z[2]+be)-ga;
            rj1[1*n+1]=1e0;
            rj2[1*n+1]=(al*z[2]+be);
            rj2[1*n+2]=al*z[1];
      f[2]=px[2]-z[2]*(p*z[1]-si)-ta*(1e0+z[1]);
            rj1[2*n+2]=1e0;
            rj2[2*n+1]=-z[2]*p-ta;
            rj2[2*n+2]=-(p*z[1]-si);
      return;
}
void outtest04(double z[],double xp[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
      if(ncon==0) fprintf(f02,"           Evstifeev lazer test,t-
time,y2,y1,yp1,yp2\n");
      fprintf(f02,"  %e   %e  %e  %e  %e\n",t,z[2],z[1],xp[1],xp[2]);
      return;
}
```

Основная программа-функция:

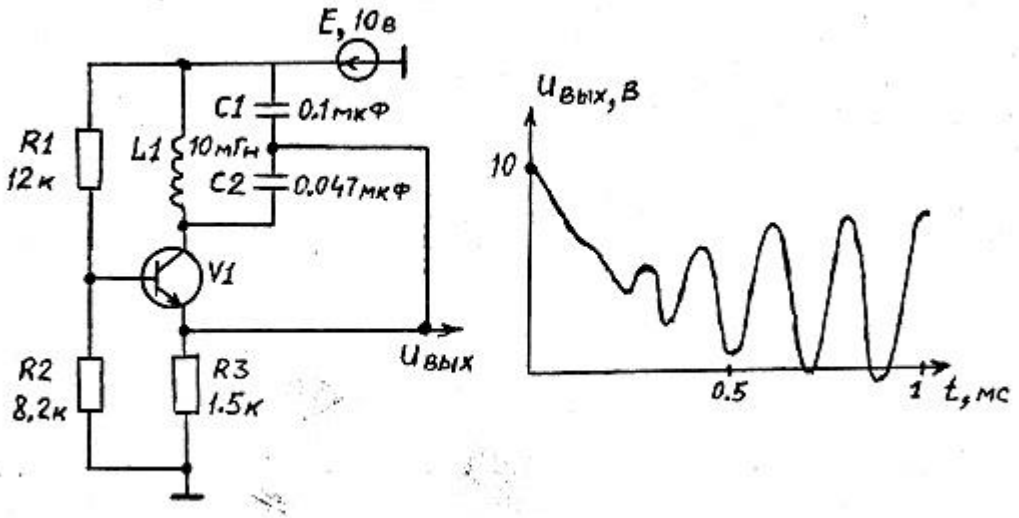
```
//test04 - Evstifeev lazer test
      f01=fopen("test04.rez","wt");
      f02=fopen("grtest04.rez","wt");
            al=1.5e-18;
            be=2.5e-6;
            ga=2.1e-6;
            p=0.6;
            si=0.18;
            ta=0.016;
      fprintf(f01,"al=%e be=%e ga=%e p=%e si=%e ta=%e\n",al,be,ga,p,si,ta);
            eps=1e-10;
      fprintf(f01,"        relative tolerance - eps=%e\n",eps);
```

```
            nm=3;
        fprintf(f01,"        number of method - nm=%d\n",nm);
            t0=0e0;
            tk=1e6;
            hmn=1e-16;
            hmx=tk;
            n=2;
            m=2;
            z[1]=-1e0;
            z1[1]=1e0;
            z[2]=0e0;
            z1[2]=1e0;
manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,&ier,
ip,fcttest04,outtest04);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
            fprintf(f01,"\n");
            for(i=1;i<=16;i++)printf("%d ",ip[i]);
            printf("\n");
            fclose(f01);
            fclose(f02);
```

Сравнение программ-решателей систем ОДУ-ДАУ было проведено для параметров реального работающего лазера. Программа-решатель из пакета Maple не дала правильное решение. В manzhuk было получено правильное решение:

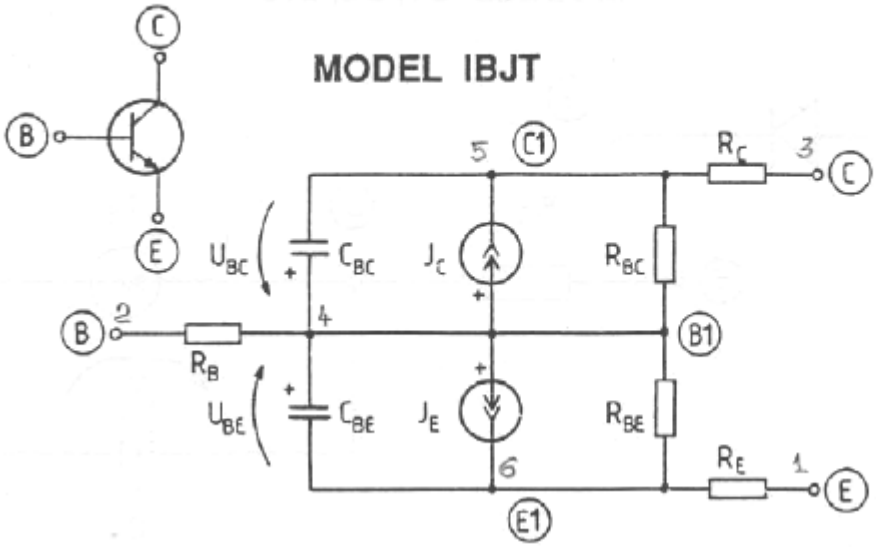


Тест05. Генератор Колпитца (test05) [14].



Данная схема и дифференциальные уравнения модели транзистора были взяты из работы [14]. Система ОДУ-ДАУ для всей схемы была получена по методу узловых потенциалов в пространстве дифференциально-алгебраических переменных. Модель транзистора из [14] имеет вид:

2.3 BIPOLAR JUNCTION TRANSISTOR MODELS



Модель транзистора оформлена в виде программы-функции ibjt1. Этот тест также является классическим примером использования параметра nbad=1 и аналитического вычисления элементов матрицы Якоби в программе-функции модели транзистора ibjt1.

Программы-функции ibjt1, fct и out:

```
//test05 - Kolpitz oscillator
double r3,re,rb,e1;
double ise,isc,cde,cdc,ce,cc;
void ibjt1(double ubc,double ube,double *jc,double *je,double *cbc,double
*cbe,double ce,double cc,int *ivoz,double isc,double ise,double a,double
cdc,double cde,double *r11,double *r12,double *r21,double *r22,double
*rcbc,double *rcbe)
{
double ra1,ra2,ic,ie,aln,ali;
            *ivoz=0;
            ra1=(ubc*(a));
            ra2=(ube*(a));
            if(ra1>40e0||ra2>40e0) *ivoz=1;
            if(*ivoz==1) goto m1000;
            if(ra1<-160e0) ra1=-160e0;
```

```
            if(ra2<-160e0) ra2=-160e0;
            ra1=exp(ra1);
            ra2=exp(ra2);
            ic=(isc)*(ra1-1e0);
            ie=(ise)*(ra2-1e0);
            aln=0.981e0;
            ali=0.8711e0;
            *jc=ic-aln*ie;
            *r11=(isc)*ra1*(a);
            *r12=-aln*(ise)*ra2*(a);
            *je=ie-ali*ic;
            *r21=-ali*(isc)*ra1*(a);
            *r22=(ise)*ra2*(a);
            *cbc=(cdc)*ra1+(cc);
            *rcbc=(cdc)*ra1*(a);
            *cbe=(cde)*ra2+(ce);
            *rcbe=(cde)*ra2*(a);
m1000:;
      return;
}
void fcttest05(double z[],double px[],double f[],double rj1[],double rj2[],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
int ivoz;
double ur1,ur2,ur3,ure;
double jc,je,cbc,cbe,r11,r12,r21,r22,rcbc,rcbe;
      ur1=-z[1]+z[3]+z[4]-z[6];
      ur2=z[1]-z[3]-z[4]+z[6]+e1;
      ur3=-z[3]+e1;
      ure=z[1]-z[2]-z[4];
ibjt1(z[1],z[2],&jc,&je,&cbc,&cbe,ce,cc,&ivoz,isc,ise,a,cdc,cde,&r11,&r12,&r21,&
r22,&rcbc,&rcbe);
      if(ivoz==1) goto m1000;
      f[1]=cbc*px[1]-ur1/r1+ur2/r2+ure/re+jc;
            rj2[1*n+1]=1e0/r1+1e0/r2+1e0/re+r11+px[1]*rcbc;
            rj2[1*n+2]=-1e0/re+r12;
            rj2[1*n+3]=-1e0/r1-1e0/r2;
            rj2[1*n+4]=-1e0/r1-1e0/r2-1e0/re;
            rj2[1*n+6]=1e0/r1+1e0/r2;
      f[2]=cbe*px[2]-ure/re+je;
            rj2[2*n+1]=-1e0/re+r21;
            rj2[2*n+2]=1e0/re+r22+px[2]*rcbe;
            rj2[2*n+4]=1e0/re;
      f[3]=c1*px[3]+z[5]+ur1/r1-ur2/r2-ur3/r3;
            rj2[3*n+1]=-1e0/r1-1e0/r2;
            rj2[3*n+3]=1e0/r1+1e0/r2+1e0/r3;
            rj2[3*n+4]=1e0/r1+1e0/r2;
            rj2[3*n+5]=1e0;
            rj2[3*n+6]=-1e0/r1-1e0/r2;
      f[4]=c2*px[4]+z[5]+ur1/r1-ur2/r2-ure/re;
            rj2[4*n+1]=-1e0/r1-1e0/r2-1e0/re;
            rj2[4*n+2]=1e0/re;
            rj2[4*n+3]=1e0/r1+1e0/r2;
            rj2[4*n+4]=1e0/r1+1e0/r2+1e0/re;
            rj2[4*n+5]=1e0;
            rj2[4*n+6]=-1e0/r1-1e0/r2;
      f[5]=l1*px[5]-z[3]-z[4];
            rj2[5*n+3]=-1e0;
            rj2[5*n+4]=-1e0;
      f[6]=z[6]/rb-ur1/r1+ur2/r2;
            rj2[6*n+1]=1e0/r1+1e0/r2;
            rj2[6*n+3]=-1e0/r1-1e0/r2;
            rj2[6*n+4]=-1e0/r1-1e0/r2;
            rj2[6*n+6]=1e0/rb+1e0/r1+1e0/r2;
      rj1[1*n+1]=cbc;
      rj1[2*n+2]=cbe;
      rj1[3*n+3]=c1;
```

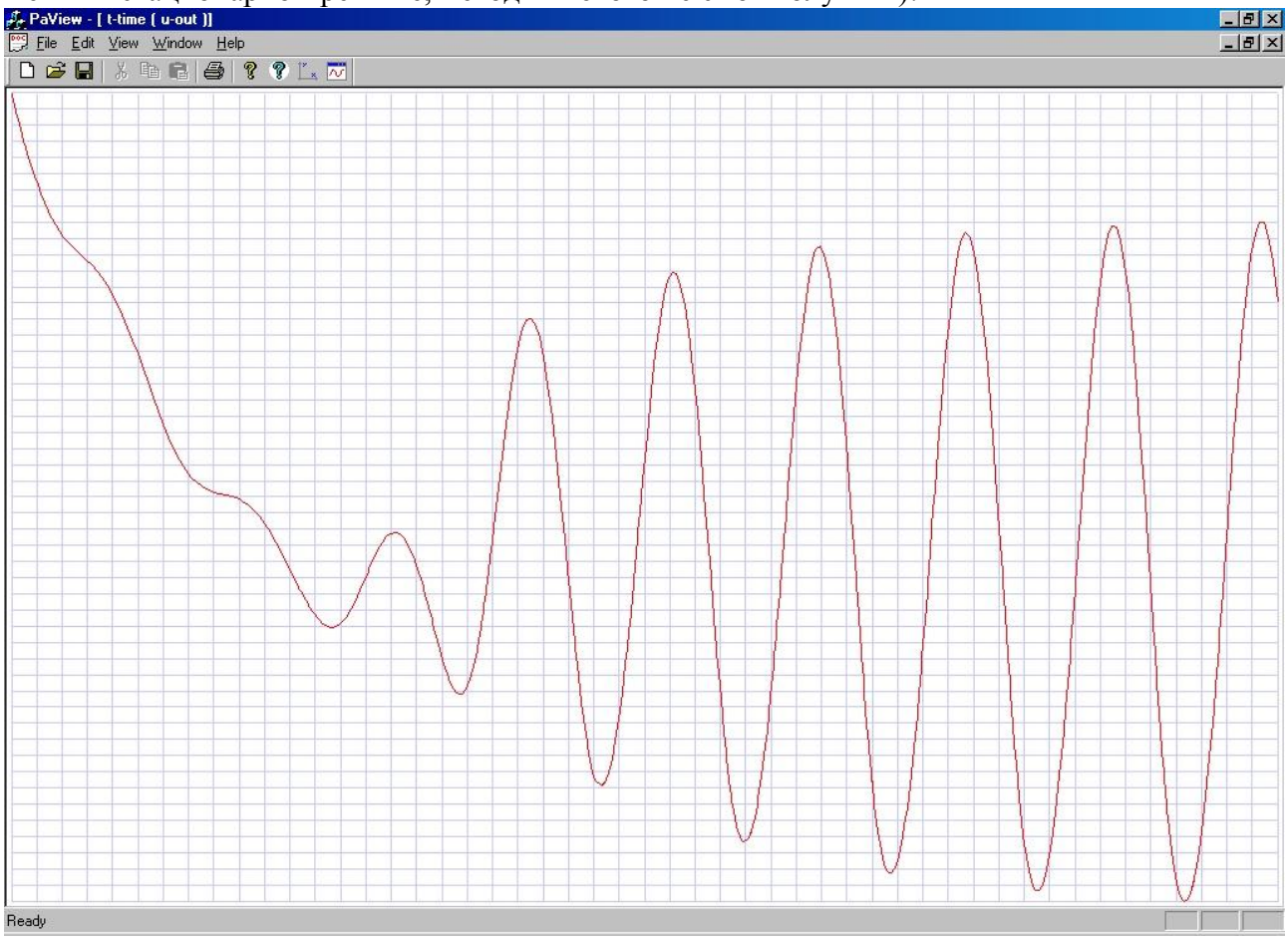
```
      rj1[4*n+4]=c2;
      rj1[5*n+5]=l1;
      goto m1100;
m1000:    (*nbad)=1;
m1100:    return;
}
void outtest05(double z[],double px[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
double vix;
            vix=-z[3]+e1;
      if(ncon==0) fprintf(f02,"          Kolpitz-oscillator,t-time,u-out,u-
kollec,u-emiter,il1\n");
      fprintf(f02,"  %e   %e  %e  %e  %e\n",t,vix,z[1],z[2],z[5]);
      return;
}
```

Основная программа-функция:

```
//test06 - using nbad=1 parameter, kolpitz-oscillator with ibjt1 model for
transistor
      f01=fopen("test05.rez","wt");
      f02=fopen("grtest05.rez","wt");
            kt=1.;
            ku=1.;
            ki=1.;
      fprintf(f01,"        KT=%e   KU=%e   KI=%e\n",kt,ku,ki);
            kr=ku/ki;
            kc=kt*ki/ku;
            kl=kt*ku/ki;
            r1=12e3*kr;
            r2=8.2e3*kr;
            r3=1.5e3*kr;
            re=0.2e0*kr;
            rb=1.8e0*kr;
            c1=100e-9*kc;
            c2=47e-9*kc;
            l1=10e-3*kl;
            e1=10e0*ku;
            ise=0.429e-13*ki;
            isc=0.578e-13*ki;
            a=38.3e0/ku;
            cde=23.6e-21*kc;
            cdc=2.1e-20*kc;
            ce=16.8e-12*kc;
            cc=8.65e-12*kc;
      fprintf(f01,"      %e    %e    %e    %e\n%e    %e    %e    %e\n%e    %e
%e    %e\n%e    %e    %e
%e\n\n",r1,r2,r3,re,rb,c1,c2,l1,e1,ise,isc,a,cde,cdc,ce,cc);
            eps=1e-3;
      fprintf(f01,"        relative tolerance - eps=%e\n",eps);
            nm=3;
      fprintf(f01,"        number of method - nm=%d\n",nm);
            t0=0e0*kt;
            tk=1e-3*kt;
            hmn=tk/1e9;
            hmx=tk;
            n=6;
            m=5;
            for(i=1;i<=6;i++){
            z[i]=0e0;
            z1[i]=0e0;
            };
manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,&ier,
ip,fcttest05,outtest05);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
            fprintf(f01,"\n");
```

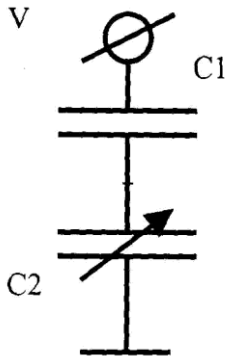
```
for(i=1;i<=16;i++)printf("%d ",ip[i]);
printf("\n");
fclose(f01);
fclose(f02);
```

Правильный результат решения этого теста был получен только AL устойчивыми методами интегрирования M2 и M3 (должны быть отрицательные значения выходного напряжения в стационарном режиме, метод M1 этого не смог получить):

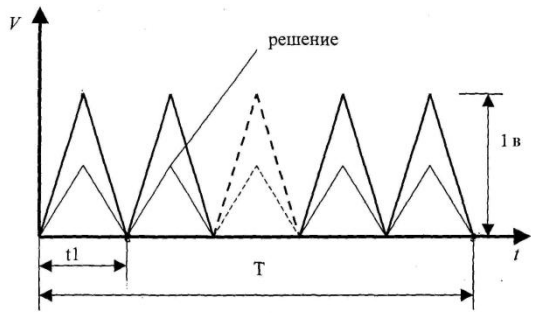


Тест06. Тест Кокина С.А. (test06).

Решение большого количества практических задач показало, что неверные решения часто бывают при интегрировании функций, имеющих разрывы производных этих функций по времени (в основном при наличии кусочно-линейных функций, зависящих от времени). Тестовая задача Кокина Сергея Александровича (одного из разработчиков САПР Avocad) является классическим примером такой задачи: численное интегрирование дифференциальных уравнений емкостного делителя, когда одна из емкостей является функцией напряжения на обкладках соответствующего конденсатора. На рисунке C1 - постоянная емкость, равная 1 (не зависит от напряжения на обкладках), C2 - выражена удобной для получения аналитического решения функцией.



Для удобства выбран пилообразный входной сигнал от 0В до 1В (толстые линии на рисунке), подаваемый на источник напряжения V с периодом t1=2 сек.



Математической моделью этой тестовой задачи будет система ДАУ вида (1) из трех уравнений:

$$\begin{cases} C1(dU_{C1}(t)/dt) - i(t) = 0 \\ C1(0.5 - U_{C2}(t)) \times (dU_{C2}(t)/dt) - i(t) = 0 \\ U_{C1}(t) + U_{C2}(t) - V(t) = 0 \end{cases}$$

относительно трех переменных $U_{C1}(t)$, $U_{C2}(t)$, $i(t) = i_{C1}(t) = i_{C2}(t)$. Аналитическое решение этой задачи (в модели все переменные рассматриваются безразмерными) для напряжения на емкости C2 при начальных условиях $U_{C1}(0) = 0$ и $U_{C2}(0) = 0$ имеет вид:

для напряжения:

$$U_{C2}(t) = C1 + C2(0) - \sqrt{(C1 + C2(0))^2 - 2 \cdot C1 \cdot V(t)},$$

для тока:

$$i_{C2}(t) = \frac{C1 \cdot (0.5 - U_{C2}(t))}{\sqrt{(C1 + C2(0))^2 - 2 \cdot C1 \cdot V(t)}}.$$

Необходимо получить численное решение для напряжения и тока на емкости C2, которое должно совпадать с аналитическим решением.

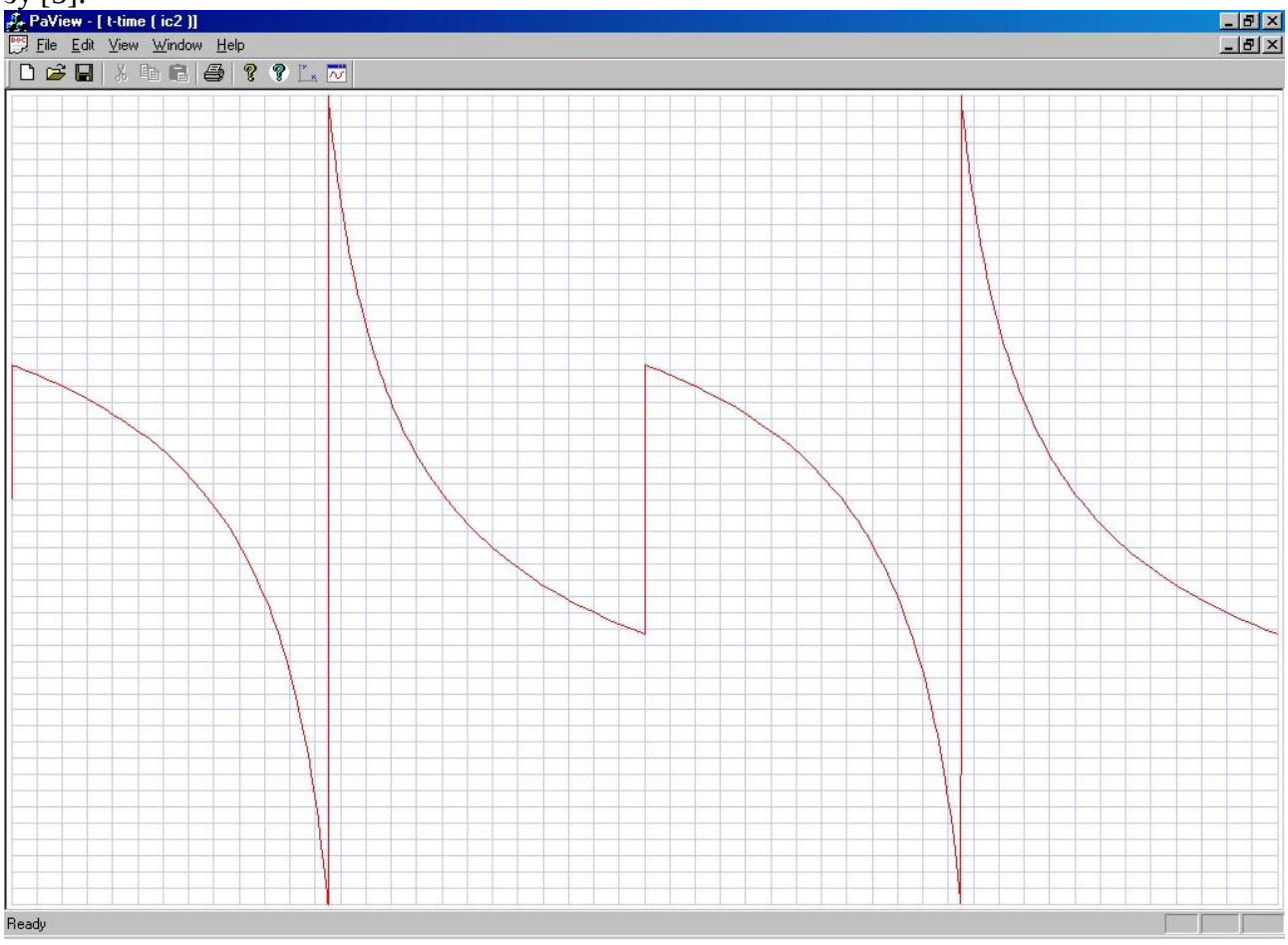
Программы fct и out:

```
//test06 - тест Кокина С.А. одного из разработчиков САПР Avocad
double T1,E,EA,ER;
void fcttest06(double z[],double px[],double f[],double rj1[],double rj2[],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
int N,N1;
            N=(int) t;
            N1=N % 2;
      if(N1==0) E=t-(double) N;
      if(N1 != 0) E=-(t-(double) N)+1.0;
//    не учитывается dC/dt
      f[1]=c1*px[1]-z[3];
            rj1[1*n+1]=c1;
            rj2[1*n+3]=-1e0;
            c2=0.5-z[2];
      f[2]=(c2)*px[2]-z[3];
            rj1[2*n+2]=c2;
            rj2[2*n+2]=-px[2];
            rj2[2*n+3]=-1e0;
            f[3]=z[1]+z[2]-E;
            rj2[3*n+1]=1e0;
            rj2[3*n+2]=1e0;
}
void outtest06(double z[],double px[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
double vvv;
            EA=(1+0.5)-sqrt((1+0.5)*(1+0.5)-2.0*1*E);
            vvv=(2.25-2.0*E);
            ER=1.0*(0.5-EA)/sqrt(vvv);
      if(ncon==0)fprintf(f02,"C&nonlenear2C,t-time,ic2,uc2,EA,ER\n");
      fprintf(f02,"  %e   %e  %e  %e  %e\n",t,z[3],z[2],EA,ER);
}
```

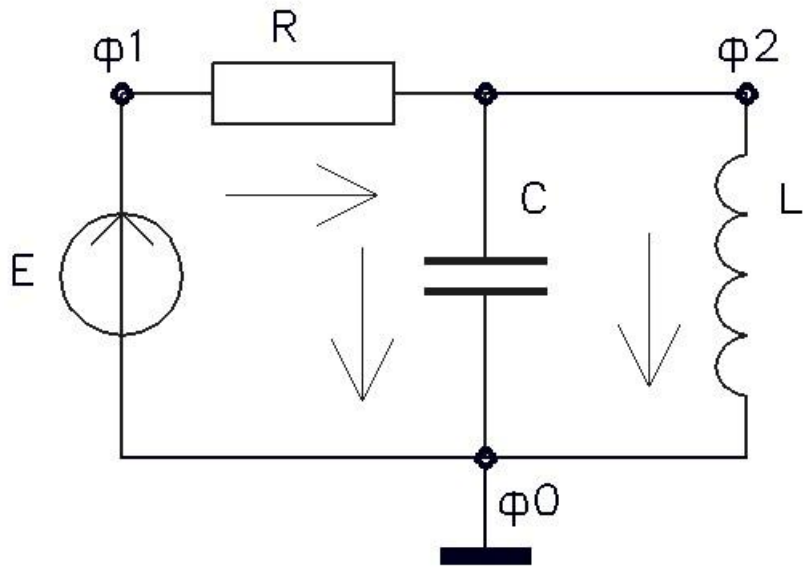
Основная программа

```
//test06 - тест Кокина, это одновременно и теоретический и практический тест
      f01=fopen("test06.rez","wt");
      f02=fopen("grtest06.rez","wt");
            c1=1e0;
            c2=0.5e0;
      fprintf(f01," c1= %e   c2= %e\n",c1,c2);
            eps=1e-5;
      fprintf(f01,"        relative tolerance - eps=%e\n",eps);
            nm=3;
      fprintf(f01,"        number of method - nm=%d\n",nm);
            t0=0.0;
            T1=1.0;
            tk=T1*4.0;
            hmn=1e-16;
            hmx=tk;
            n=3;
            m=2;
            z[1]=0e0;
            z1[1]=0e0;
            z[2]=0e0;
            z1[2]=0e0;
manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,&ier,
ip,fcttest06,outtest06);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
            fprintf(f01,"\n");
            for(i=1;i<=16;i++)printf("%d ",ip[i]);
            printf("\n");
            fclose(f01);
            fclose(f02);
```

Результаты правильного решения теста Кокина для тока $i_{C2}(t)$ были получены не сразу [5]:



Перейдем к рассмотрению примеров получения математических моделей для реальных динамических систем: двух электрических схем, одной линейной и одной нелинейной. Тест 07. Сначала рассмотрим пример получения математической модели для линейной электрической схемы, в расширенном координатном пространстве переменных.



Вектор базисных переменных для этой схемы будет иметь размерность 12 и включает в себя следующие переменные:

Vфс = (uE,iE,uR,iR,uC,iC,uL,iL,ф1,ф2,pxC,pxL).

Система ДАУ для этой схемы будет иметь размерность n=10, m=2 и включает в себя следующие уравнения:

uE=f(t) или uE-f(t)=0

uE=ф1 или uE-ф1=0

uR=R×iR или uR-R×iR=0

uR=ф1-ф2 или uR-ф1+ф2=0

iC=C×xpC или iC-C×xpC=0

uC=ф2 или uC-ф2=0

uL=L×xpL или uL=L×xpL=0

uL=ф2 или uL-ф2=0

iE-iR=0

iR-iC-iL=0.

Отметим, что в базисе узловых потенциалов размерность системы ДАУ будет равна 2 (в пять раз меньше!), но при этом возможности моделирования отдельных элементов схемы будут ограничены (например, нельзя моделировать идеальные источники напряжения).

Для этой задачи вектор *X*=(uC,iL), вектор *PX*=(pxC,pxL), а вектор *Y*=(uE,iE,uR,iR,iC,uL,ф1,ф2). Вектор-функция *F* включает в себя 10 вышеприведенных уравнений.

Программы-функции fct и out:

```
//test07 - rlc circuit in new basis
double c;
```

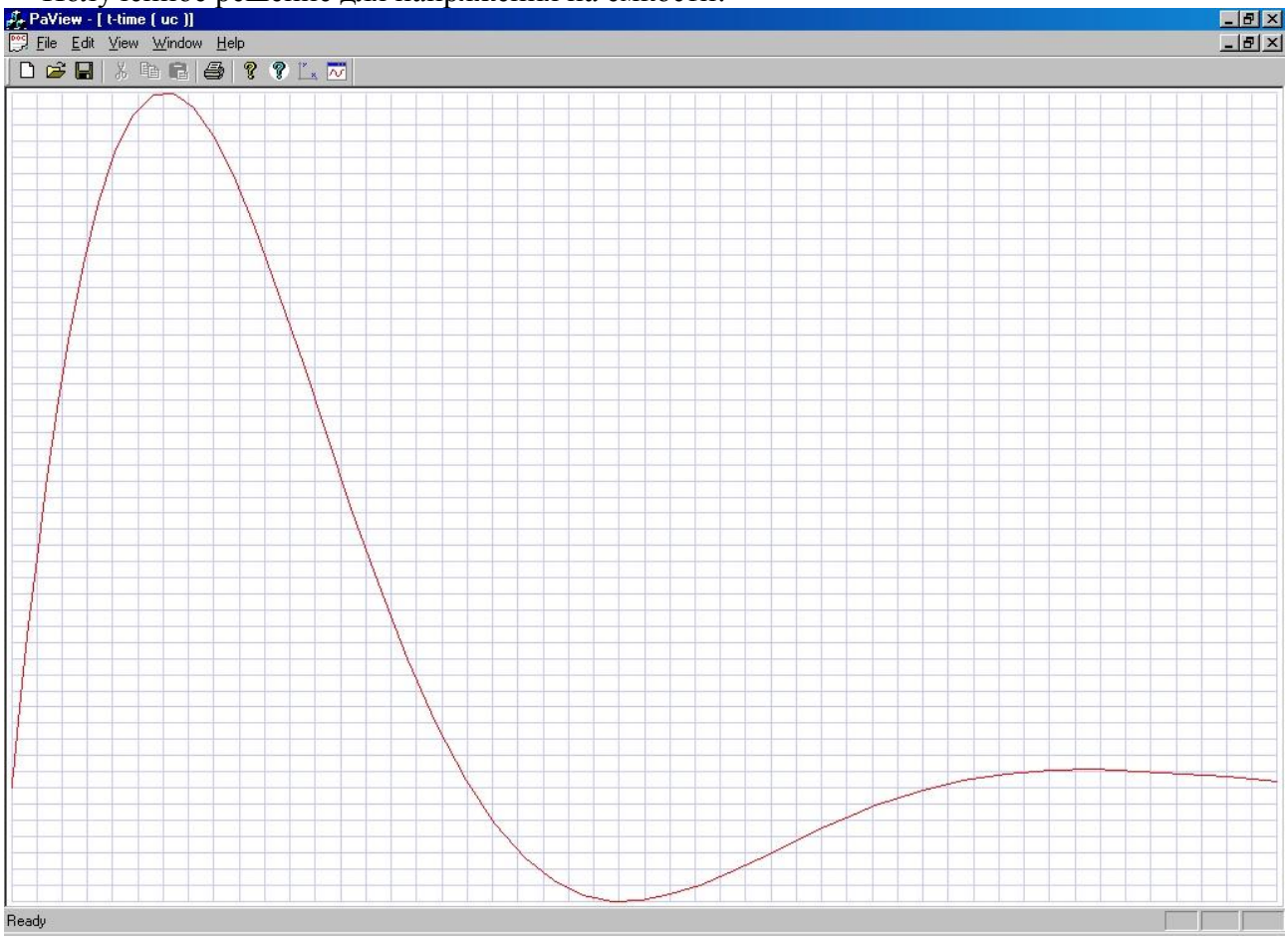
```
void fcttest07(double z[],double px[],double f[],double rj1[],double rj2[],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
      f[1]=z[3]-1e0;
       rj2[1*n+3]=1e0;
      f[2]=z[3]-z[9];
       rj2[2*n+3]=1e0;
       rj2[2*n+9]=-1e0;
      f[3]=z[5]-r*z[6];
       rj2[3*n+5]=1e0;
       rj2[3*n+6]=-r;
      f[4]=z[5]-z[9]+z[10];
       rj2[4*n+5]=1e0;
       rj2[4*n+9]=-1e0;
       rj2[4*n+10]=1e0;
      f[5]=z[7]-c*px[1];
       rj1[5*n+1]=-c;
       rj2[5*n+7]=1e0;
      f[6]=z[1]-z[10];
       rj2[6*n+1]=1e0;
       rj2[6*n+10]=-1e0;
      f[7]=z[8]-l*px[2];
       rj1[7*n+2]=-l;
       rj2[7*n+8]=1e0;
      f[8]=z[8]-z[10];
       rj2[8*n+8]=1e0;
       rj2[8*n+10]=-1e0;
      f[9]=z[4]-z[6];
       rj2[9*n+4]=1e0;
       rj2[9*n+6]=-1e0;
      f[10]=z[6]-z[7]-z[2];
       rj2[10*n+6]=1e0;
       rj2[10*n+7]=-1e0;
       rj2[10*n+2]=-1e0;
}
void outtest07(double z[],double px[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
long j;
if(ncon==0) fprintf(f01,"rlcnew,t-time,uc,ic,ul,il\n");
if(ncon==0) fprintf(f02,"rlcnew,t-time,uc,ic,ul,il\n");
fprintf(f02,"  %e    %e   %e   %e   %e\n",t,z[1],z[7],z[8],z[2]);
}
```

Основная программа-функция:

```
//test07 = rlc circuit in new basis
f01=fopen("test07.rez","wt");
f02=fopen("grtest07.rez","wt");
            kt=1.;
            ku=1.;
            ki=1.;
      fprintf(f01,"kt=%le ku=%le ki=%le\n",kt,ku,ki);
            kr=ku/ki;
            kc=kt*ki/ku;
            kl=kt*ku/ki;
            r=1e0;
            c=1e0;
            l=1e0;
      fprintf(f01," r= %e    l= %e     c= %e\n",r,l,c);
            eps=1e-3;
      fprintf(f01,"        relative tolerance - eps=%e\n",eps);
            nm=3;
      fprintf(f01,"        number of method - nm=%d\n",nm);
            t0=0e0*kt;
            tk=10e0*kt;
            hmn=tk/1e12;
            hmx=tk;
            n=10;
```

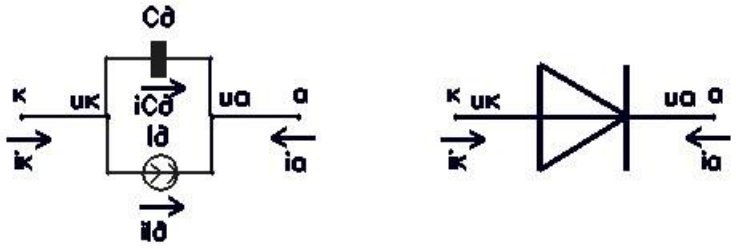
```
            m=2;
            z[1]=0e0;
            z1[1]=0e0;
            z[2]=0e0;
            z1[2]=0e0;
manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,&ier,
ip,fcttest07,outtest07);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
            fprintf(f01,"\n");
            for(i=1;i<=16;i++)printf("%d ",ip[i]);
            printf("\n");
            fclose(f01);
            fclose(f02);
```

Полученное решение для напряжения на емкости:



Полученное решение сравнивалось с аналитическим, при этом были получены численно более устойчивые решения при изменении коэффициентов этой системы ДАУ, по сравнению с системой ДАУ, полученной в базисе узловых потенциалов.

Тест08. Рассмотрим сначала пример получения математической модели для нелинейной модели диода. На рисунке показаны эквивалентная схема диода из двухполюсных элементов и его представление как функционального многополюсника.



Вектор переменных для этой схемной модели диода будет иметь размерность 9 (из них первые 5 переменных являются внутренними) и включает в себя следующие переменные (u или ф – напряжения или потенциалы, i – токи, px – производные соответствующих переменных по времени для данной схемы):

Vдиод = (uСд, iСд, pxuСд, uIд, iIд, uк, iк, ua, ia)

Система ДАУ для этой схемной модели диода (математическая модель диода в расширенном пространстве переменных) будет иметь размерность 6 и включает в себя следующие уравнения (первые 4 уравнения – для 2-х внутренних двухполюсников схемной модели диода):

iСд-Сд×pxuСд=0

uСд-ua+uк=0

iIд-(iт×$e^{(uIд/фт)}$+1)=0

uIд-ua+uк=0

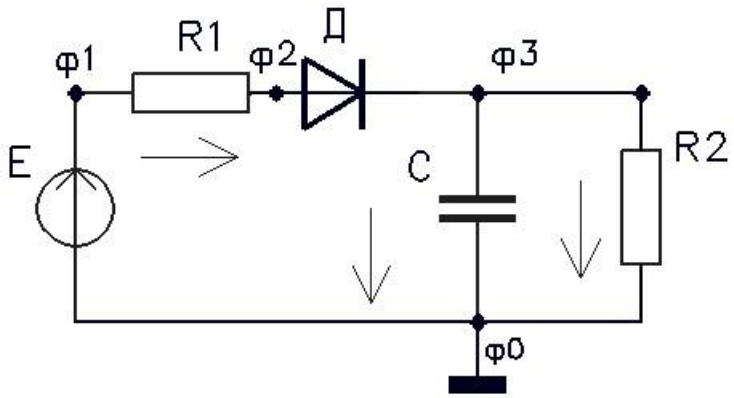
ia-iСд-iIд=0

iк+iСд+iIд=0

В этих уравнениях iт, Сд, и фт – это параметры данной математической модели диода.

Такие уравнения будем включать в библиотеку моделей электронных схем для ПМК ПА10 и использовать при получении моделей более сложных схем, содержащих диоды.

Рассмотрим пример получения математической модели соответствующей схемы с диодом, показанной на рисунке ниже, в расширенном пространстве переменных с использованием полученной модели диода.



Вектор переменных Vсхемы включает в себя в первую очередь переменные диода, для которых узловые потенциалы заменяются на потенциалы узлов подключения диода:

Vдиод = (uСд, iСд, xpСд, uIд, iIд, ф3, iк, ф2, ia)∈Vсхемы

Затем в вектор переменных Vсхемы добавляются остальные переменные, в результате вектор математической модели схемы на рисунке будет иметь размерность 19 и включает в себя следующие переменные:

Vсхемы=(uСд, iСд, uIд, iIд, ф3, iк, ф2, ia uE, iE, uR1, iR1, uC, iC, uR2, iR2, ф1, xpСд, xpC).

Система ДАУ для этой схемы будет иметь размерность 17 и включает в себя следующие уравнения:

Сначала в эту систему добавляются 6 уравнений диода, в которых узловые потенциалы заменяются на узлы подключения диода:

iСд-Сд×xpСд=0

uСд-ф2+ф3=0

iIд-(iт×$e^{(uIд/фт)}$+1)=0

uIд-ф2+ф3=0

ia-iСд-iIд=0

iк+iСд+iIд=0

Затем добавляются 2 уравнения для суммы токов в узлах подключения диода:

iR1-ia=0

iк+iC1+iR2=0

Затем добавляются 8 уравнений для 4-х двухполюсников:

uE-f(t)=0

uE-ф1=0

uR1-R1×iR1=0

uR1-ф1+ф2=0

iC-C×xpC=0

uC1-ф3=0

uR2-R2×iR2=0

uR2-ф3=0

И в завершении формирования замкнутой системы ДАУ добавляется 1-но уравнение для суммы токов в узле ф1:

iE-iR1=0

Для этой задачи вектор *X*=(uСд,uC), вектор *PX*=(xpСд,xpC), а вектор *Y*= (iСд, uIд, iIд, ф3, iк, ф2, ia uE, iE, uR1, iR1, iC, uR2, iR2, ф1). Вектор-функция *F* включает в себя 17 вышеприведенных уравнений.

Программы-функции fct и out:

```
//test08 - circuit with one diod - ac to dc converter (new basis)
double am,id;
double it,cb,mft,tau;
double ue,cd,ide,ra1;
void fcttest08(double z[],double px[],double f[],double rj1[],double rj2[],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
            ue=am*sin(om*t);
            ra1=(z[4]/mft);
      if(ra1>=70.0) *nbad=1;
      if(*nbad==1) return;
      if(ra1<-70.0) ra1=-70.0;
            ra1=exp(ra1);
            ide=it*(ra1-1.0);
            cd=cb+(it+ide)*(tau/mft);
      f[1]=z[3]-cd*px[1];
            rj1[1*n+1]=-cd;
            rj2[1*n+3]=1.0;
            rj2[1*n+4]=-px[1]*(tau/mft)*it*(1.0/mft)*ra1;
      f[2]=z[1]-z[6]+z[7];
            rj2[2*n+1]=1.0;
            rj2[2*n+6]=-1.0;
            rj2[2*n+7]=1.0;
      f[3]=z[5]-ide;
            rj2[3*n+5]=1.0;
```

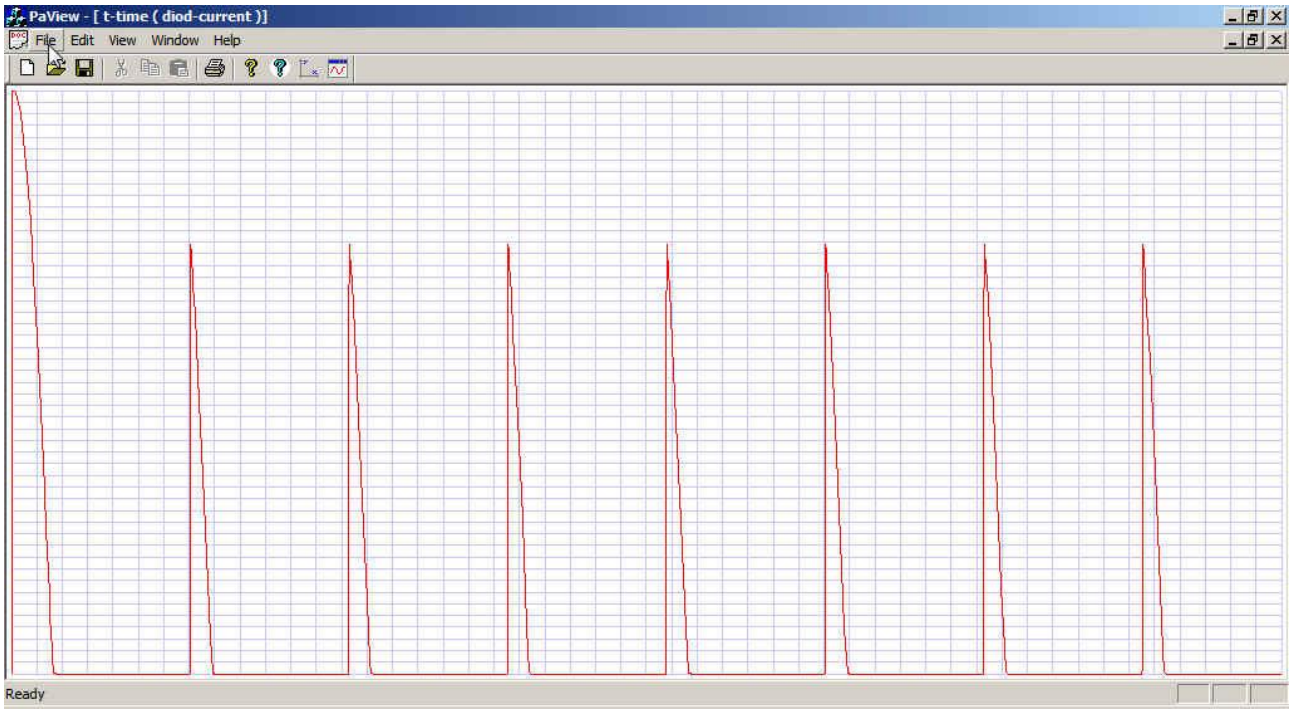
```
            rj2[3*n+4]=-it*(1.0/mft)*ra1;
      f[4]=z[4]-z[6]+z[7];
            rj2[4*n+4]=1.0;
            rj2[4*n+6]=-1.0;
            rj2[4*n+7]=1.0;
      f[5]=z[8]-z[3]-z[5];
            rj2[5*n+8]=1.0;
            rj2[5*n+3]=-1.0;
            rj2[5*n+5]=-1.0;
      f[6]=z[9]+z[3]+z[5];
            rj2[6*n+9]=1.0;
            rj2[6*n+3]=1.0;
            rj2[6*n+5]=1.0;
      f[7]=z[11]-z[8];
            rj2[7*n+11]=1.0;
            rj2[7*n+8]=-1.0;
      f[8]=-z[9]-z[14]-z[15];
            rj2[8*n+9]=-1.0;
            rj2[8*n+14]=-1.0;
            rj2[8*n+15]=-1.0;
      f[9]=z[13]-ue;
            rj2[9*n+13]=1.0;
      f[10]=z[13]-z[17];
            rj2[10*n+13]=1.0;
            rj2[10*n+17]=-1.0;
      f[11]=z[11]-r1*z[10];
            rj2[11*n+11]=1.0;
            rj2[11*n+10]=-r1;
      f[12]=z[11]-z[17]+z[6];
            rj2[12*n+11]=1.0;
            rj2[12*n+17]=-1.0;
            rj2[12*n+6]=1.0;
      f[13]=z[14]-c1*px[2];
            rj1[13*n+2]=-c1;
            rj2[13*n+14]=1.0;
      f[14]=z[2]-z[7];
            rj2[14*n+2]=1.0;
            rj2[14*n+7]=-1.0;
      f[15]=z[16]-r2*z[15];
            rj2[15*n+16]=1.0;
            rj2[15*n+15]=-r2;
      f[16]=z[16]-z[7];
            rj2[16*n+16]=1.0;
            rj2[16*n+7]=-1.0;
      f[17]=z[12]-z[10];
            rj2[17*n+12]=1.0;
            rj2[17*n+10]=-1.0;
      id=z[5];
}
void outtest08(double z[],double px[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
      if(t==t0) fprintf(f02,"ac-to-dc-converter,t-time,diod-current,diod-
voltage,ic,uc\n");
      fprintf(f02," %e   %e  %e  %e  %e\n",t,id,z[1],z[14],z[2]);
}
```

Основная программа-функция:

```
//test08 - circuit with one diod - ac to dc converter (new basis)
f01=fopen("test08.rez","wt");
f02=fopen("grtest08.rez","wt");
            kt=1.0;
            ku=1.0;
            ki=1.0;
fprintf(f01,"        kt=%e   ku=%e   ki=%e\n",kt,ku,ki);
            kr=ku/ki;
            kc=kt*ki/ku;
            kl=kt*ku/ki;
```

```
            pi4=atan(1.0);
            am=100.0*ku;
            mft=0.026*ku;
            r1=1.0*kr;
            r2=5e3*kr;
            c1=1e-5*kc;
            om=100.0*4.0*pi4/kt;
            it=1e-7*ki;
            cb=1e-10*kc;
            tau=1e-6*kt;
            tk=0.16*kt;
fprintf(f01,"  r1=%e r2=%e c1=%e am=%e om=%e\n  it=%e cb=%e mft=%e tau=%e
tk=%e\n\n",
            r1,r2,c1,am,om,it,cb,mft,tau,tk);
eps=1e-3;
fprintf(f01,"        relative tolerance - eps=%e\n",eps);
nm=3;
fprintf(f01,"        number of method - nm=%d\n",nm);
t0=0.0*kt;
hmn=tk/1e12;
hmx=tk/20.0;
n=17;
m=2;
z[1]=0.0;
z[2]=0.0;
z1[1]=0.0;
z1[2]=0.0;
manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,&ier,
ip,fcttest08,outtest08);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
            fprintf(f01,"\n");
            for(i=1;i<=16;i++)printf("%d ",ip[i]);
            printf("\n");
            fclose(f01);
            fclose(f02);
```

Полученное решение для тока через диод:



Полученное решение сравнивалось с решением, полученным ранее для метода узловых потенциалов в программе ПА7, при этом были получены численно более устойчивые решения при изменении коэффициентов этой системы ДАУ, по сравнению с системой ДАУ, полученной в базисе узловых потенциалов.

Рассмотрим тестовые задачи из области химической кинетики и экологии.

Тест09. Задача определения концентрации озона в атмосфере описывается системой ОДУ 2-го порядка [15].

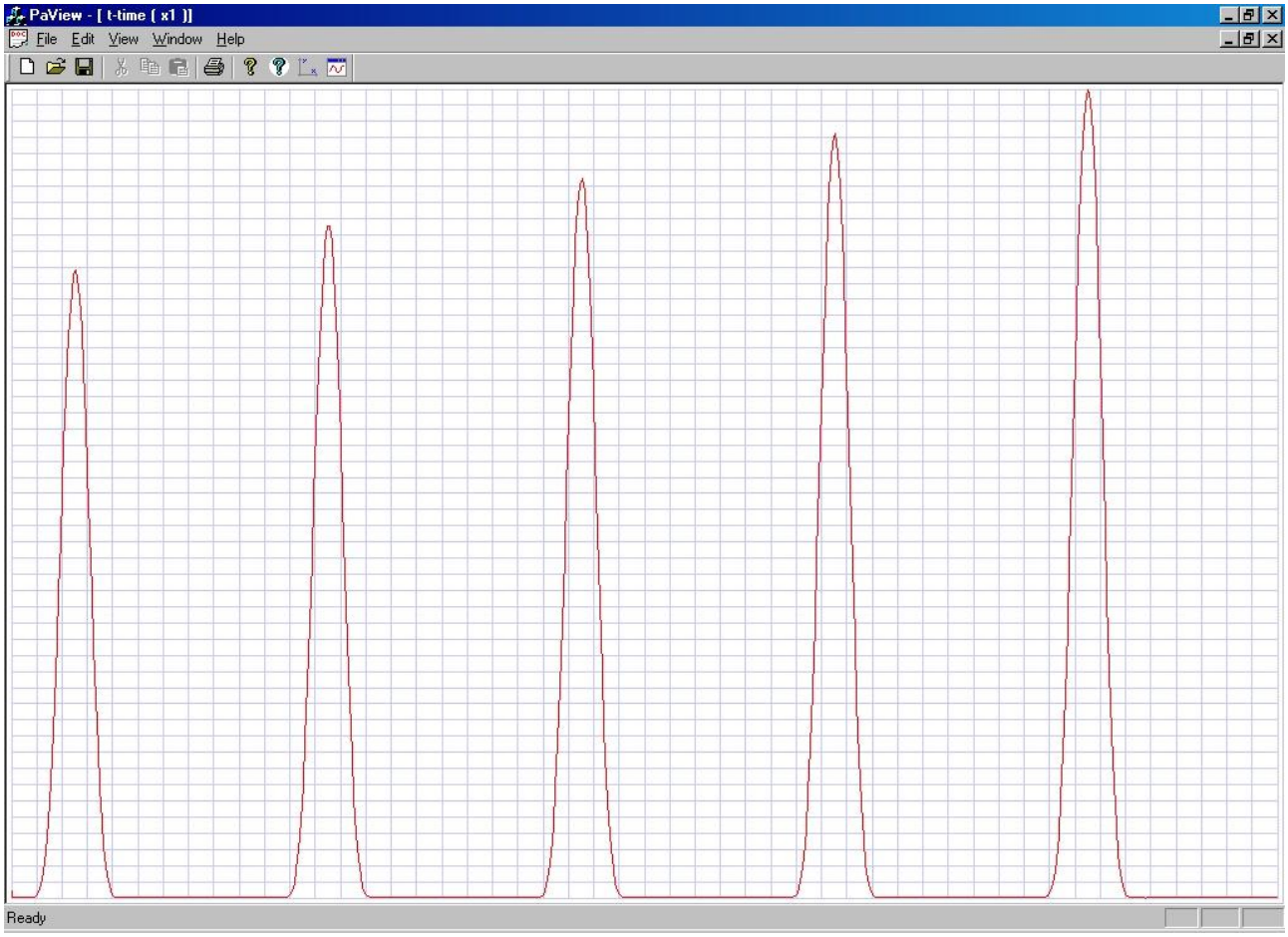
Программы-функции fct и out:

```
//test09 - Ozone model
double k1,k2,k5,c4,omt,k3,k4;
void fcttest09(double z[],double px[],double f[],double rj1[],double rj2[5],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
      omt=(atan(1.0)/10800.0)*t;
      if(sin(omt) <= 0.0) k3=0.0;
      if(sin(omt) > 0.0) k3=exp(-c3/sin(omt));
      if(sin(omt) <= 0.0) k4=0.0;
      if(sin(omt) > 0.0) k4=exp(-c4/sin(omt));
      f[1]=px[1]+k1*k5*z[1]-k4*z[2]+k2*z[1]*z[2]-2.0*k3*k5;
            rj1[1*n+1]=1e0;
            rj2[1*n+1]=k1*k5+k2*z[2];
            rj2[1*n+2]=-k4+k2*z[1];
      f[2]=px[2]-k1*k5*z[1]+k4*z[2]+k2*z[1]*z[2];
            rj1[2*n+2]=1e0;
            rj2[2*n+1]=-k1*k5+k2*z[2];
            rj2[2*n+2]=k4+k2*z[1];
      return;
}
void outtest09(double z[],double px[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
      if(ncon==0) fprintf(f02,"           Ozone model,t-time,x1,x2,k3,k4\n");
      fprintf(f02,"  %e   %e  %e  %e  %e\n",t,z[1],z[2],k3,k4);
      return;
}
```

Основная программа-функция:

```
//test09 - Ozone model
      f01=fopen("test09.rez","wt");
      f02=fopen("grtest09.rez","wt");
            k1=1.63e-16;
            k2=4.66e-16;
            k5=3.7e16;
            c3=22.62;
            c4=7.601;
      fprintf(f01,"  k1= %e  k2= %e   k5= %e  c3= %e  c4= %e\n",k1,k2,k5,c3,c4);
            eps=1e-3;
      fprintf(f01,"        relative tolerance - eps=%e\n",eps);
            nm=3;
      fprintf(f01,"        number of method - nm=%d\n",nm);
            t0=0e0;
            tk=3600.0*120.0;
            hmn=1e-12;
            hmx=tk;
            n=2;
            m=2;
            z[1]=1e6;
            z[2]=1e12;
            for(i=1;i<=2;i++) z1[i]=abs(z[i]);
manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,&ier,
ip,fcttest09,outtest09);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
            fprintf(f01,"\n");
            for(i=1;i<=16;i++)printf("%d ",ip[i]);
            printf("\n");
            fclose(f01);
            fclose(f02);
```

Полученное решение для первой переменной является достоверным и точным:



Тест10. Задача определения фазовых траекторий динамической нелинейной системы Рикитаки описывается системой ОДУ 4-го порядка [16].

Программы-функции fct и out:

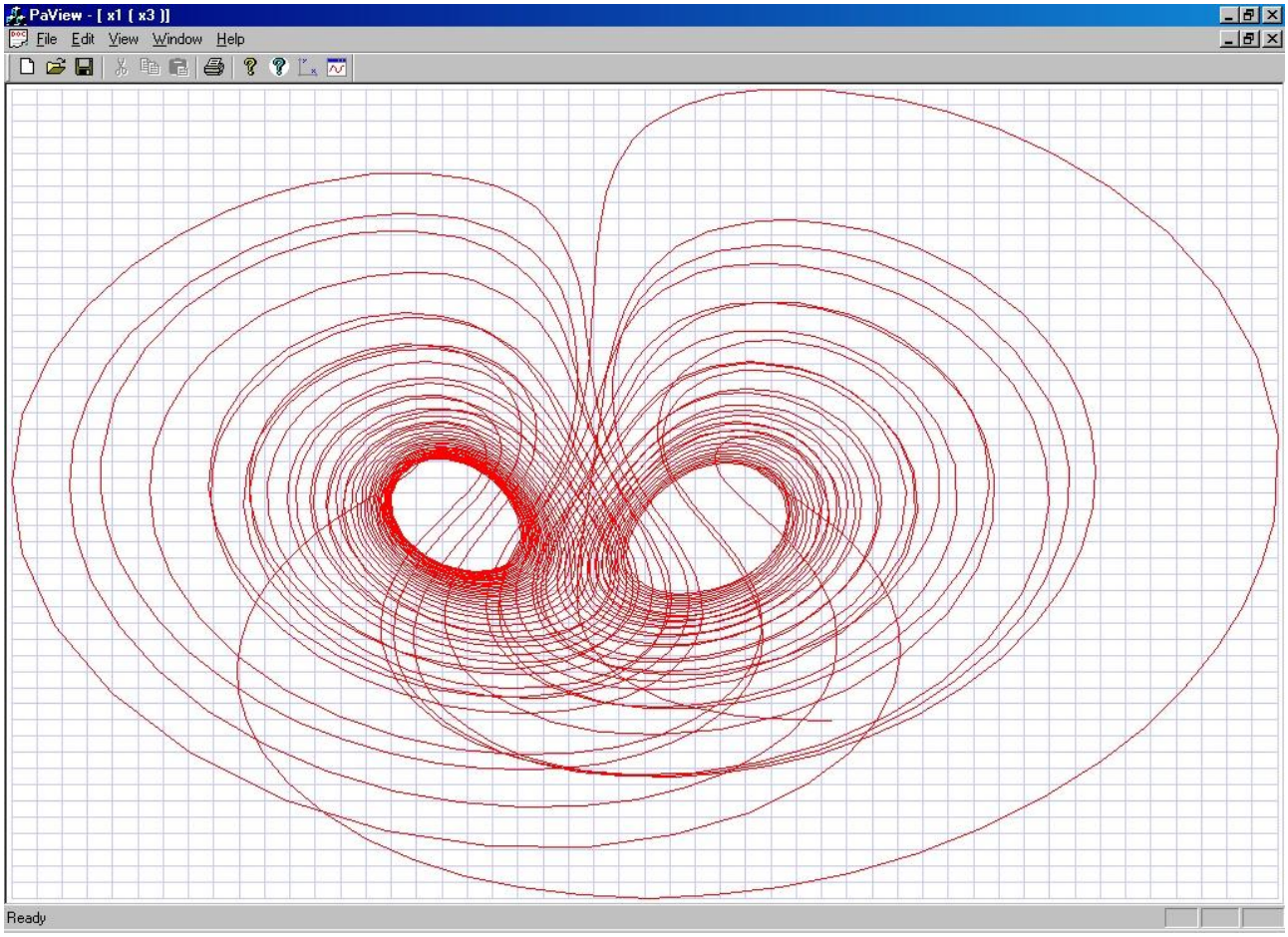
```
//test10 - Rikitaki nonlinear dynamics test
double v1,v2;
void fcttest10(double z[],double px[],double f[],double rj1[],double rj2[5],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
      f[1]=px[1]+mu*z[1]-z[2]*z[3];
            rj1[1*n+1]=1e0;
            rj2[1*n+1]=mu;
            rj2[1*n+2]=-z[3];
            rj2[1*n+3]=-z[2];
      f[2]=px[2]+mu*z[2]-z[1]*z[4];
            rj1[2*n+2]=1e0;
            rj2[2*n+2]=mu;
            rj2[2*n+1]=-z[4];
            rj2[2*n+4]=-z[1];
      f[3]=px[3]-1.0+v1*z[3]+z[1]*z[2];
            rj1[3*n+3]=1e0;
            rj2[3*n+3]=v1;
            rj2[3*n+1]=z[2];
            rj2[3*n+2]=z[1];
      f[4]=px[4]-1.0-v2*z[4]+z[1]*z[2];
            rj1[4*n+4]=1e0;
            rj2[4*n+4]=-v2;
            rj2[4*n+1]=z[2];
            rj2[4*n+2]=z[1];
      return;
}
void outtest10(double z[],double px[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
```

```
      if(ncon==0) fprintf(f02,"           Rikitaki nonlinear dynamics test,t-
time,x1,x2,x3,x4\n");
      fprintf(f02,"  %e   %e  %e  %e  %e\n",t,z[1],z[2],z[3],z[4]);
//          for(i=1;i<=16;i++)printf("%d ",ip[i]);
//          printf("\n");
      return;
}
```

Основная программа-функция:

```
//test10 - Rikitaki nonlinear dynamics test
      f01=fopen("test10.rez","wt");
      f02=fopen("grtest10.rez","wt");
            mu=1.1;
            v1=0.0022;
            v2=0.002;
      fprintf(f01,"  mu= %e  v1= %e   v2= %e\n",mu,v1,v2);
            eps=1e-3;
      fprintf(f01,"        relative tolerance - eps=%e\n",eps);
            nm=3;
      fprintf(f01,"        number of method - nm=%d\n",nm);
            t0=0e0;
            tk=240.0;
            hmn=1e-12;
            hmx=tk;
            n=4;
            m=4;
            z[1]=1.0;
            z[2]=1.0;
            z[3]=-1.0;
            z[4]=2.0;
            for(i=1;i<=4;i++) z1[i]=abs(z[i]);
manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,&ier,
ip,fcttest10,outtest10);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
            fprintf(f01,"\n");
            for(i=1;i<=16;i++)printf("%d ",ip[i]);
            printf("\n");
            fclose(f01);
            fclose(f02);
```

Полученное решение фазовой траектории, т.е. зависимости третьей переменной от первой, соответствует результатам работы [16]:



Тест11. Известная задача из области экологии – задача хищник-жертва (модель Лоттка-Волтера) описывается системой ОДУ 2-го порядка.

Программы-функции fct и out:

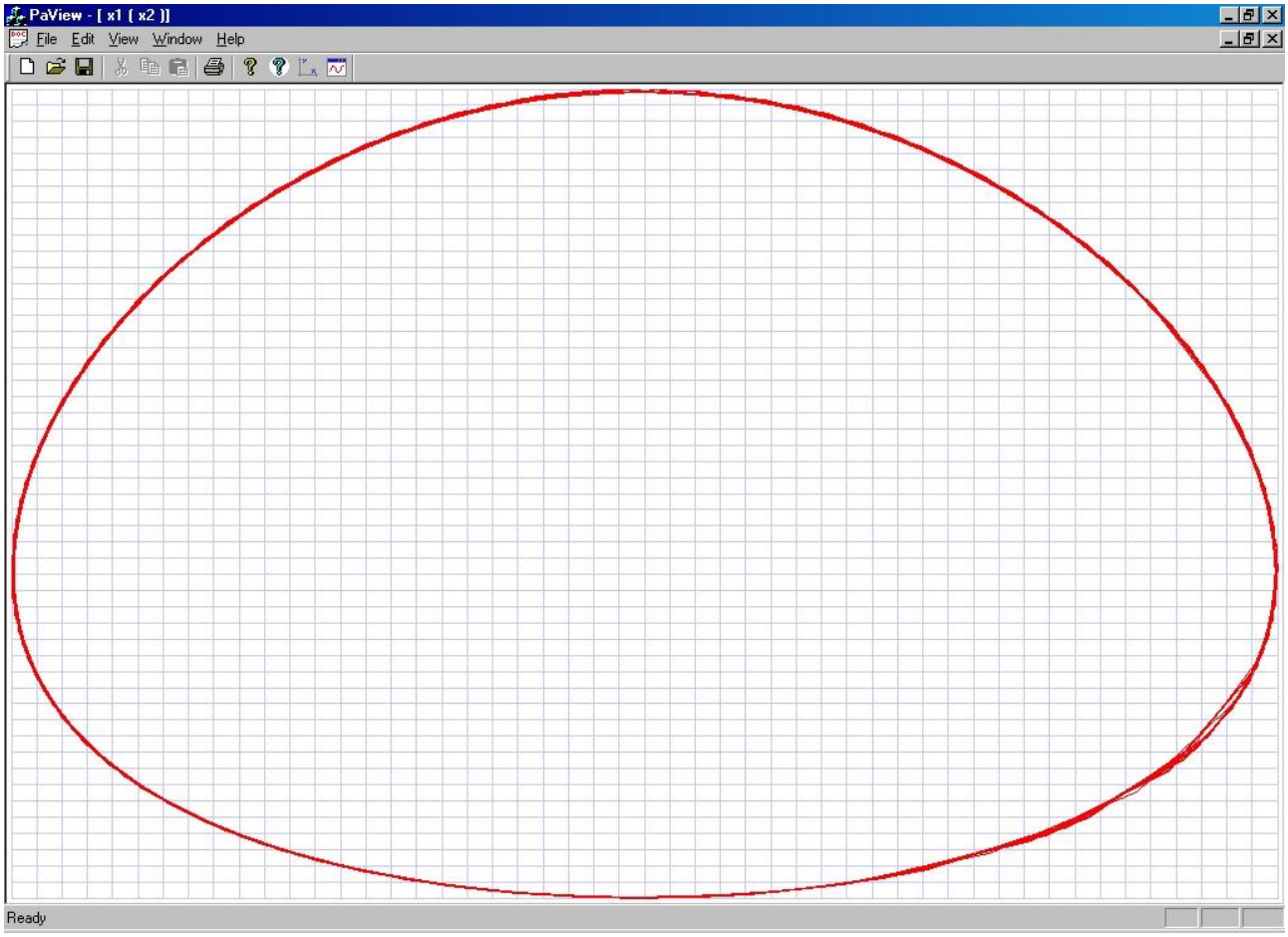
```
//test11 - Lottka-Voltera test
double b1,b2;
void fcttest11(double z[],double px[],double f[],double rj1[],double rj2[5],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
      f[1]=px[1]-b1*z[1]+a12*z[1]*z[2];
            rj1[1*n+1]=1e0;
            rj2[1*n+1]=-b1+a12*z[2];
            rj2[1*n+2]=a12*z[1];
      f[2]=px[2]+b2*z[2]-a21*z[1]*z[2];
            rj1[2*n+2]=1e0;
            rj2[2*n+1]=-a21*z[2];
            rj2[2*n+2]=b2-a21*z[1];
      return;
}
void outtest11(double z[],double px[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
      if(ncon==0) fprintf(f02,"           Lottka-Voltera test,t-
time,x1,x2,px1,px2\n");
      fprintf(f02,"  %e   %e  %e  %e  %e\n",t,z[1],z[2],px[1],px[2]);
      return;
}
```

Основная программа-функция:

```
//test11 - Lottka-Voltera test
      f01=fopen("test11.rez","wt");
      f02=fopen("grtest11.rez","wt");
            b1=1.0;
            b2=1000.0;
            a12=0.1;
            a21=100;
```

```
      fprintf(f01,"  b1= %e  b2= %e   a12= %e  a21= %e\n",b1,b2,a12,a21);
            eps=1e-3;
      fprintf(f01,"        relative tolerance - eps=%e\n",eps);
            nm=3;
      fprintf(f01,"        number of method - nm=%d\n",nm);
            t0=0e0;
            tk=30.0;
            hmn=1e-12;
            hmx=tk;
            n=2;
            m=2;
            z[1]=10.0;
            z[2]=17.0;
            for(i=1;i<=2;i++) z1[i]=abs(z[i]);
//ier=-1;
manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,&ier,
ip,fcttest11,outtest11);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
            fprintf(f01,"\n");
            for(i=1;i<=16;i++)printf("%d ",ip[i]);
            printf("\n");
            fclose(f01);
            fclose(f02);
```

Полученное решение для зависимости второй переменной от первой является достоверным и точным:



Тест12. Рассмотрим задачу химической кинетики. Система ОДУ для частных реакций гидроалюминирования олефинов с диизобутилалюминийгидридом (реакция ДИБАГ, значения параметров соответствуют T = -65◦C) имеет вид:

$$\begin{cases} dx_1/dt = -0{,}2x_1 + 0{,}7x_2^2 - 0{,}17x_1x_5 \\ dx_2/dt = 0{,}4x_1 - 1{,}4x_2^2 + 0{,}17x_1x_5 - 3{,}5x_2x_5 \\ dx_5/dt = -0{,}17x_1x_5 - 3{,}5x_2x_5 \\ dx_8/dt = 0{,}17x_1x_5 + 3{,}5x_2x_5 \end{cases}$$

При t=0: x_1=0,044 ммоль; x_5=0,467 ммоль; x_8=0,0056 ммоль.
При t=0 (в долях): x_1=0,086; x_5=0,903; x_8=0,011.
Остальные нули.
Балансные соотношения по закону сохранения массы имеют вид:
b1 = $60x_1+30x_2+19x_5+49x_8$ = 22,86;
b2 = $36x_1+18x_2+8x_5+26x_8$= 10,6;
b3 = $2x_1+x_2+x_5+2x_8$= 1,1;
b4 = $2x_1+x_2+x_8$= 0,18;

Программы-функции fct и out:

```
//test12 - химическая кинетика - реакция с диизобутилалюминийгидридом (ДИБАГ)
double b3,b4,km1;
void fcttest12(double z[],double xp[],double f[],double rj1[],double rj2[],int
n,int m,double t,double h,int ncon,int *nbad,int ip[])
{
      f[1]=xp[1]+k1*z[1]-km1*z[2]*z[2]+k2*z[1]*z[3];
            rj1[1*n+1]=1.0;
      f[2]=xp[2]-2.0*k1*z[1]+2.0*km1*z[2]*z[2]-k2*z[1]*z[3]+k3*z[2]*z[3];
            rj1[2*n+2]=1.0;
      f[3]=xp[3]+k2*z[1]*z[3]+k3*z[2]*z[3];
            rj1[3*n+3]=1.0;
      f[4]=xp[4]-k2*z[1]*z[3]-k3*z[2]*z[3];
            rj1[4*n+4]=1.0;
return;
}
void outtest12(double z[],double xp[],int n,int m,double t,double t0,double
tk,double h,double *tkv,int ncon,int ip[])
{
      if(t==t0) fprintf(f02,"chemical kinetics dibag reaction,t-
time,x1,x2,x3,x4\n");
            b1=60.0*z[1]+30.0*z[2]+19.0*z[3]+49.0*z[4];
            b2=36.0*z[1]+18.0*z[2]+8.0*z[3]+26.0*z[4];
            b3=2.0*z[1]+1.0*z[2]+1.0*z[3]+2.0*z[4];
            b4=2.0*z[1]+1.0*z[2]+1.0*z[4];
      if(t==t0) fprintf(f01,"chemical kinetics dibag reaction,t-
time,x1,x3,b1,b2\n");
// start print of tabulation results
      if(ncon==0)tkp=deltatp;
      if(deltatp==0) goto m20;
      if(t<=tkp) goto m10;
      tkp=tkp+deltatp;
m10:
      if((tkp<*tkv)&&(*tkv>=t)) *tkv=tkp;
      if(t<tstartp) goto m20;
      if(t>tendp) goto m20;
      if(t==*tkv) fprintf(f01," %e %e %e %e %e\n",t,z[1],z[3],b1,b2);
m20:
// end print of tabulation results
      fprintf(f02," %e   %e  %e  %e  %e\n",t,z[1],z[2],z[3],z[4]);
}
```

Основная программа-функция:

```
//test12 - химическая кинетика - реакция с диизобутилалюминийгидридом (ДИБАГ)
      f01=fopen("test12.rez","wt");
      f02=fopen("grtest12.rez","wt");
```

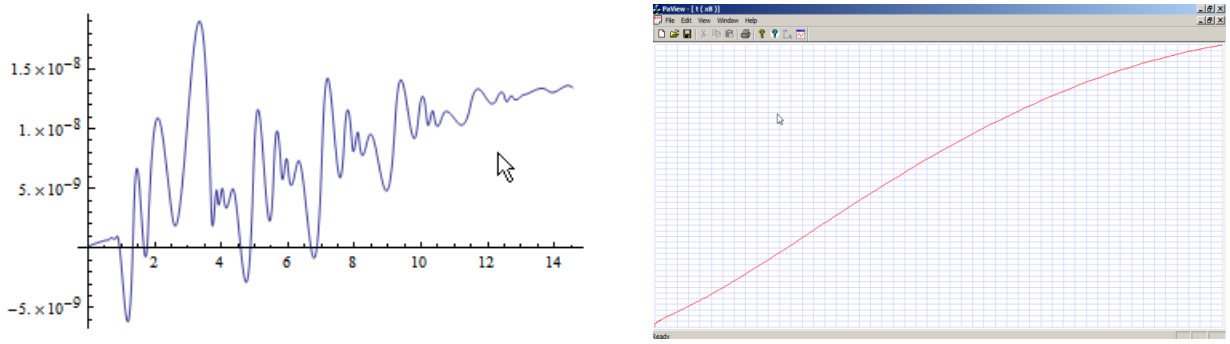
```
            k1=0.2;
            km1=0.7;
            k2=0.17;
            k3=3.48;
fprintf(f01,"chemical kinetics - DIBAG reaction, parameters:\n");
fprintf(f01,"    k1=%e    km1=%e   k2=%e   k3=%e\n",k1,km1,k2,k3);
            eps=1e-3;
fprintf(f01,"        relative tolerance - eps=%e\n",eps);
            nm=3;
fprintf(f01,"        number of method - nm=%d\n",nm);
            t0=0.0;
            tk=3.55;
            tstartp=2e0;//time for stasrt printing
            tendp=3.5;//time for end printing
            deltatp=0.1;//step for printing
      fprintf(f01," tstartp=%e deltatp=%e tendp=%e\n",tstartp,deltatp,tendp);
            hmn=1e-10;
            hmx=tk;
            n=4;
            m=4;
            z[1]=0.086;
            z[2]=0.0;
            z[3]=0.903;
            z[4]=0.011;
            z1[1]=0.086;
            z1[2]=0.0;
            z1[3]=0.903;
            z1[4]=0.011;
            px[1]=-k1*z[1]+km1*z[2]*z[2]-k2*z[1]*z[3];
            px[2]=2.0*k1*z[1]-2.0*km1*z[2]*z[2]+k2*z[1]*z[3]-k3*z[2]*z[3];
            px[3]=-k2*z[1]*z[3]-k3*z[2]*z[3];
            px[4]=k2*z[1]*z[3]+k3*z[2]*z[3];
            ncon=2;
            ier=-1;
manzhuk(z,px,z1,xp1,f,rj1,rj2,t,t0,tk,h,hmn,hmx,eps,&tkv,n,m,nm,ncon,&nbad,&ier,
ip,fcttest12,outtest12);
            fprintf(f01,"ier= %d \n",ier);
            printf("ier= %d \n",ier);
            for(i=1;i<=16;i++)fprintf(f01,"%d ",ip[i]);
            fprintf(f01,"\n");
            for(i=1;i<=16;i++)printf("%d ",ip[i]);
            printf("\n");
            fclose(f01);
            fclose(f02);
```

Полученное решение для зависимости первой и третьей переменной от времени, а также балансные соотношения (b1, b2) на каждом шаге интегрирования, которые подтверждают достоверность и точность результатов решения этой тестовой задачи имеет вид:

```
chemical kinetics - DIBAG reaction, parameters:
    k1=2.000000e-001    km1=7.000000e-001   k2=1.700000e-001   k3=3.480000e+000
        relative tolerance - eps=1.000000e-003
        number of method - nm=3
 tstartp=2.000000e+000 deltatp=1.000000e-001 tendp=3.500000e+000
chemical kinetics dibag reaction,t-time,x1,x3,b1,b2
 2.000000e+000 4.306834e-002 8.261066e-001 2.285600e+001 1.060600e+001
 2.100000e+000 4.163330e-002 8.229766e-001 2.285600e+001 1.060600e+001
 2.200000e+000 4.024805e-002 8.199515e-001 2.285600e+001 1.060600e+001
 2.300000e+000 3.891073e-002 8.170280e-001 2.285600e+001 1.060600e+001
 2.400000e+000 3.761957e-002 8.142028e-001 2.285600e+001 1.060600e+001
 2.500000e+000 3.637285e-002 8.114726e-001 2.285600e+001 1.060600e+001
 2.600000e+000 3.516895e-002 8.088342e-001 2.285600e+001 1.060600e+001
 2.700000e+000 3.400629e-002 8.062844e-001 2.285600e+001 1.060600e+001
 2.800000e+000 3.288337e-002 8.038202e-001 2.285600e+001 1.060600e+001
 2.900000e+000 3.179875e-002 8.014386e-001 2.285600e+001 1.060600e+001
 3.000000e+000 3.075104e-002 7.991368e-001 2.285600e+001 1.060600e+001
 3.100000e+000 2.973891e-002 7.969120e-001 2.285600e+001 1.060600e+001
```

```
 3.200000e+000 2.876109e-002 7.947614e-001 2.285600e+001 1.060600e+001
 3.300000e+000 2.781634e-002 7.926827e-001 2.285600e+001 1.060600e+001
 3.400000e+000 2.690349e-002 7.906731e-001 2.285600e+001 1.060600e+001
 3.500000e+000 2.602140e-002 7.887304e-001 2.285600e+001 1.060600e+001
ier= 0
3 305 103 0 4 0 0 0 303 103 0 0 0 0 0 0
```

Ниже приведен пример решения практической задачи химической кинетики [17]. Данная задача является жесткой за счет большого разброса порядков в константах скоростей реакции и собственных значений матрицы Якоби, коэффициент жесткости более 10^6. Для решения этой задачи сначала был выбран пакет математических программ Mathematica 8.0. Для проверки достоверности и точности полученного решения была выбрана Си библиотека SADEL.



а **б**

Рис. 2. График зависимости дифференциальной переменной x8 системы ОДУ от времени при TOL = 10^{-6}, а - Mathematica 8.0; б – SADEL.

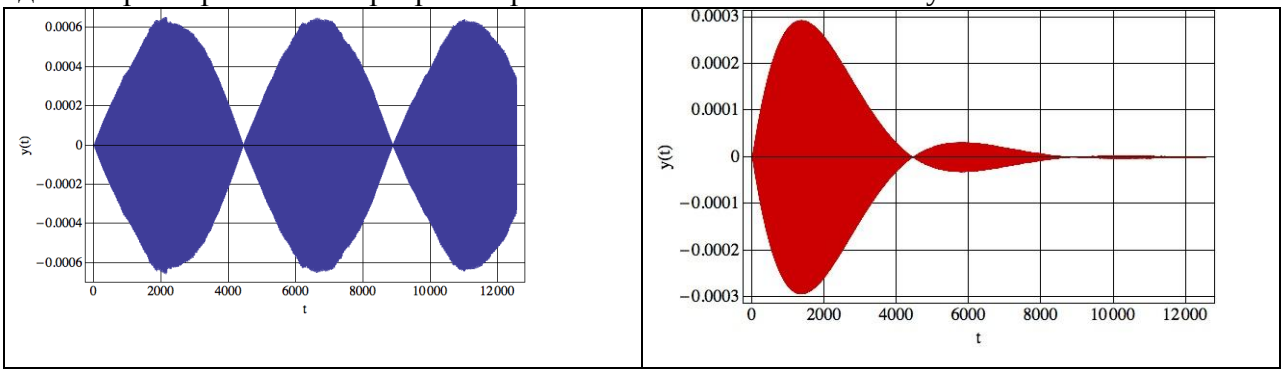
Как видно из рис. 2а, метод, выбранный в пакете Mathematica 8.0 с параметрами по умолчанию выдал колебательный график решения. Неявный метод М3, реализованный в библиотеке SADEL, выдал правильный график решения – рис.2б. Следует отметить, что после настройки параметров методов интегрирования, реализованных в пакете Mathematica 8.0, был также получен правильный график решения, однако большинство пользователей математических программ не являются профессиональными специалистами в численных методах и программах для решения систем ОДУ-ДАУ, поэтому достоверность и требуемая точность решения систем ОДУ-ДАУ должна быть обеспечена для параметров программ-решателей этих уравнений, рекомендуемых для этих программ-решателей по умолчанию, либо должно быть выдано соответствующее предупреждающее сообщение.

Приведем в заключение результаты сравнительного тестирования различных программ решателей жестких систем ОДУ [8].

Таблица 1. Сравнение программ-решателей систем ОДУ-ДАУ для жестких систем ОДУ с многопериодным решением (ТЕСТЫ 1-4)

Тесты\Программы-решатели систем ОДУ-ДАУ	*manzhuk.c 2012 Методы M2,M3*	*MATLAB 2007 Метод Ode15s*	*Maple 2007 Метод Rosenbrock*	*Mathematica 8 2011 Метод BDF*	*NAG C-Library 2012 Метод BDF*	*IMSL C-Library 2012 Метод BDF*
ТЕСТ 1. Уравнения Ван дер Поля $MU=10^6$	+	+	-	-	+	+
ТЕСТ 2. Уравнения Ван дер Поля $MU=10^9$	+	-	-	-	-	-
ТЕСТ 3. Высокодобротный фильтр *kt=1, ki=1, ku=0.01*	+	-	+	+	-	-
ТЕСТ 4. Высокодобротный фильтр $kt=10^{-104}$, *ku=1, ki=1*	+	-	-	+	-	-
ТЕСТ 5. Локально неустойчивая система ОДУ $MU=10^6$	+	-	-	+	-	-
ТЕСТ 6. Моделирование свечения лазера	+	+	-	+	+	-

В таблице 1 сравнивались только методы интегрирования программ-решателей систем ОДУ-ДАУ, рекомендуемые в соответствующих пакетах и библиотеках математических программ для решения жестких систем ОДУ-ДАУ. Знак минус означает невозможность получения решения или (в большинстве случаев) качественно неверный результат без всякого предупреждения о возможных ошибках. Например, на рисунке 1 представлен результат качественно неверного решения теста 4 с помощью программы-решателя систем ОДУ-ДАУ неявным методом BDF из библиотеки C-Library NAG (а), а также правильное решение этого теста с помощью методов M2 и M3 из библиотеки SADEL (б). Предупреждений о возможном недостоверном решении в программе-решателе библиотеки C-Library NAG не было.



а **б**

Рис. 1. Решение задачи ТЕСТ 4 при TOL = 0.001, а - метод BDF из библиотеки C-Library NAG, б - метод M3 из Си библиотеки SADEL.

Выводы

Таким образом, модели, полученные в новом расширенном координатном пространстве переменных, значительно превышают по размерности решаемых уравнений модели, полученные в классических координатных пространствах переменных, но при этом значительно увеличивается скорость сходимости и точность решения систем НАУ на каждом шаге интегрирования. Хотя требования к затратам оперативной памяти при этом резко возрастают, это не является проблемой для современной вычислительной техники. Достоверность и точность результатов решения систем ОДУ-ДАУ намного важнее. Все вышеприведенные задачи и тесты были решены достоверно и точно только одним методом интегрирования (М3) и изменением только одного параметра интегрирования (задаваемая точность интегрирования eps).

С помощью пакетов математических программ (MATLAB, Maple, Mathcad, Mathematica) можно правильно решить практически любую систему ОДУ-ДАУ (после настройки методов и параметров интегрирования, но при этом время расчета может быть очень большим). Главный недостаток известных пакетов и математических программ – **они могут выдать неверное решение при стандартной заданной точности интегрирования eps=0.001 без предупреждения пользователя.** Другой недостаток - пользователь должен быть математиком высокой квалификации, знающим математический английский (например, MATLAB не локализуется). В ПМК ПА10 системы ОДУ-ДАУ формируются автоматически и пользователи не должны быть математиками высокой квалификации, поэтому программы интегрирования систем ОДУ-ДАУ для математического ядра ПМК ПА10 должны выдавать правильное решение при заданной точности интегрирования **eps=0.001**, либо предупреждение о возможном неверном решении. В первую очередь это требование должно выполняться для всех известных тестовых задач.

В данной версии программы-решателя систем ОДУ-ДАУ manzhuk реализован классический, устаревший алгоритм выбора шага интегрирования, поэтому заданная точность не гарантируется за однократный расчет. Для гарантии достоверности и точности полученных решений надо выполнять многократные расчеты (желательно и с решателями из других библиотек) с разными параметрами точности eps (вплоть до максимально допустимой точности $1e^{-11}$) разными методами интегрирования и добиваться совпадения результатов расчета. Пока используется стандартная программа решения систем ЛАУ с удвоенной точностью вычислений на языке Си (тип double), поэтому все вышеприведенные тестовые системы ОДУ из области математического и компьютерного моделирования динамических систем Хайрера-Ванера-Маничева-Скворцова-Евстифеева-Кокина AL-устойчивыми неявными методами интегрирования систем ОДУ решаются достоверно, очень точно и быстро за несколько секунд, но пока с использование классических методов оценки достоверности и точности решения систем ОДУ без использования повышенной точности вычислений при решении систем ЛАУ. Новые алгоритмы и программы решения систем ЛАУ с использованием повышенной точности вычислений и соответствующего эффективного и оптимального выбора шагов интегрирования будут реализованы позднее.

Список литературы

1. IEEE Standard for Floating-Point Arithmetic 754-2008, IEEE, 2008 // http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5223319&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5223319 (дата просмотра сайта январь 2010 г.).
2. Guiyou Mao, Linda R. Petzold. Efficient integration over discontinuities for differential-algebraic systems, Computers & Mathematics with Applications, Volume 43, Issues 1–2, January 2002, Pages 65-79.
3. Норенков И.П., Трудоношин В.А., Федорук В.Г. Метод формирования математических моделей для адаптируемых программных комплексов анализа радиоэлектронных схем // Радиотехника, - 1986, № 9. С.67-72.
4. Маничев В. Б., Глазкова В. Н. Методы интегрирования систем ОДУ для адаптируемых программных комплексов анализа РЭС // Радиотехника.- 1988, №4. С. 88-91.
5. Д.М. Жук, В.Б. Маничев, А.О. Ильницкий Методы и алгоритмы решения дифференциально-алгебраических уравнений для моделирования систем и объектов во временной области. // Информационные технологии. – 2010, №7, часть1. С. 16-24.
6. В.Б.Маничев, В.В.Глазкова, Д.Ю.Кожевников, Д.А.Кирьянов, М.К.Сахаров Решение систем линейных алгебраических уравнений с удвоенной точностью вычислений на языке Си. //Вестник МГТУ, сер. Приборостроение. 2011. - Вып. 4. С. 25-36.
7. Хайрер Э., Нерсетт С., Ваннер Г. Решение обыкновенных дифференциальных уравнений. Нежесткие задачи: Пер. с англ.-М.:Мир, 1990.- 512 с
8. Жук Д.М., Маничев В.Б., Сахаров М.К. SADEL – библиотека «сверхточных» решателей для программного комплекса ПА10 (SADEL-PA10). В сб. научных трудов МЭС - 2012 - М.: ИППМ РАН, 2012. С. 147-153.
9. Заворин А.Н. Тестирование программ решения жестких систем обыкновенных дифференциальных уравнений. Материал информационного фонда РФАП Латвии. Инв.No ИМ0020, ВЦ ЛГУ им. П.Стучки, Рига, 1984, 42 с.
10. Хайрер Э., Ваннер Г. Решение обыкновенных дифференциальных уравнений. Жесткие задачи: Пер. с англ. -М.: Мир, 2001. - 498 с.
11. Маничев В.Б. Новые алгоритмы для программ анализа динамики технических систем // Вестник МГТУ, сер. Приборостроение. - 1996. - Вып. 1. - С. 48-56.
12. Скворцов Л.М. Явный многошаговый метод численного решения жестких дифференциальных уравнений // Журнал вычислительной математики и математической физики. 2007. Т. 47. № 6. С. 959-967.
13. Системы автоматизированного проектирования. Учеб. пособие для вузов: в 9 кн., кн. 8 // Под ред. И.П.Норенкова. М.: Высшая школа, 1986.
14. O. Vityaz, V.Porra. Testing of Time Domain Simulators for Nonlinear Electronic Circuits. Helsinki University of Technology, Faculty of Electrical Engineering, Electronic Circuit Design Laboratory, Report 4, Finland, July 1988. 65 pages.
15. Alan C. Hindmarsh, Linda R. Petzold. Algorithms and Software for Ordinary Differential Equations and Differential / Algebraic Equations. Numerical Mathematics Group Center for Computational Sciences@ Engineering, UCRL-JC-116619, April 19 1994, 38 pages.
16. В.И.Потапов Визуализация фазовых траекторий динамической системы Рикитаки / НЕЛИНЕЙНАЯ ДИНАМИКА, 2010, Т. 6, №2, с. 255–265.
17. Valerii N. Snytnikov, Tatyana I. Mishchenko, Vladimir N. Snytnikov, Sergei E. Malykhin,Vasilii I. Avdeev, Valentin N. Parmon. “Autocatalytic gas-phase dehydrogenation of ethane”. // Research on Chemical Intermediates, March 2012, Volume 38, Issue 3-5, pp 1133-1147.