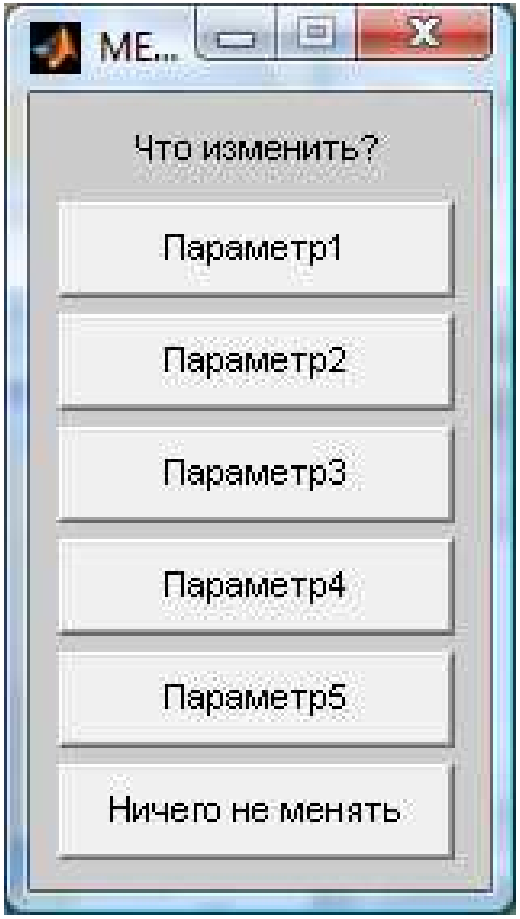
6.4.5. Организация изменения данных в диалоговом режиме

Повторение действий, содержащихся в ядре "Yadro.m", имеет смысл только в том случае, когда в начале этого ядра обеспечено выполнение действий по изменению некоторых из исходных величин.

Организацию диалогового изменения данных рассмотрим на примере некоторых 5 параметров: *Параметр1, Параметр2, … , Параметр5.* Пусть их обозначения как переменных в программе таковы: x1, x2, … , x5. Тогда меню выбора параметра для изменения его значения должно содержать 6 альтернатив: 5 из них предназначены для выбора одного из указанных параметров, а последняя – для выхода из меню, если значение всех параметров установлены.

Поэтому вариант оформления такого меню может быть, например, следующим:

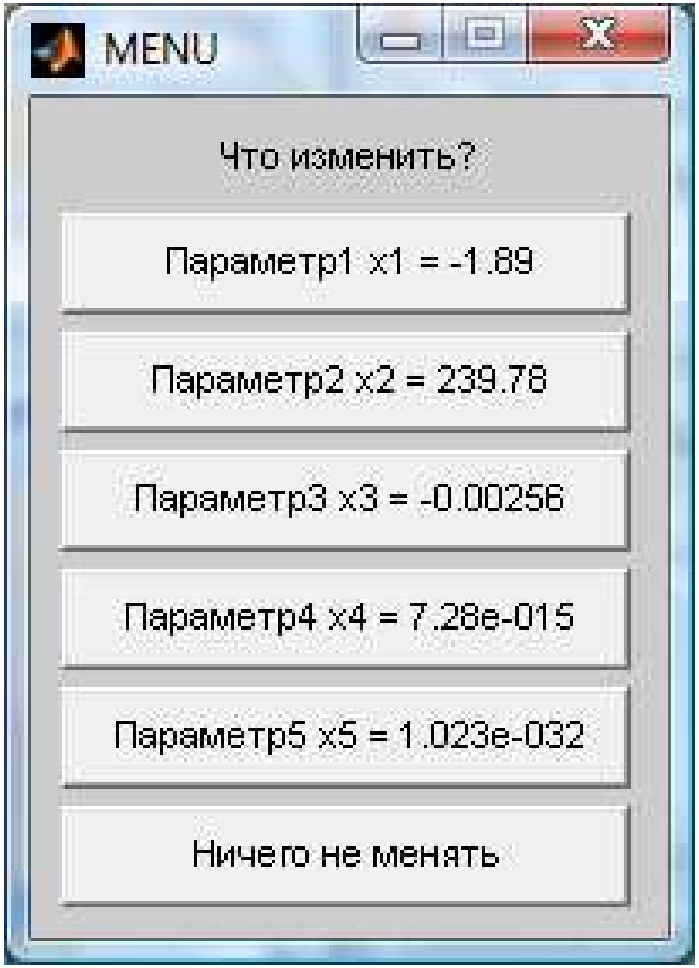
k = menu(' Что изменить? ', 'Параметр1', 'Параметр2', … 'Параметр3 ', ' Параметр4 ', ' Параметр5 ', ' Ничего не менять ')



Недостатком такого оформления окна меню является отсутствие информации о значении самого параметра. Поэтому пользователю сложно принять решение, значение какого именно параметра следует изменить и как.

Для отображении информации о текущем значении соответствующего параметра можно использовать функцию ***sprintf:***

```
x1=-1.89; x2=239.78; x3=-2.56e-3;
  x4=7.28e-15; x5=1.023e-32;
k = menu( ' Что изменить? ', ...
sprintf (' Параметр1 x1 = %g', x1),...
sprintf (' Параметр2 x2 = %g', x2),...
sprintf (' Параметр3 x3 = %g', x3),...
sprintf (' Параметр4 x4 = %g', x4),...
sprintf (' Параметр5 x5 = %g', x5),...
' Ничего не менять ')
```



Меню позволяет выбрать параметр, который нужно изменить, однако не обеспечивает самого изменения выбранного параметра.

Это изменение должно быть осуществлено с помощью ввода нового значения с клавиатуры:

x = ***input***([***sprintf*** *('Текущее значение x =%g',x), 'Новое значение x = ']).*

Если ввести команды

```
» x = 3.02e-2;
» x=input( [sprintf('Текущее значение x = %g
  ', x), ' Новое значение x = '])
```

то в командном окне появится надпись:

Текущее значение x = 0. 0302 Новое значение x =

Теперь нужно набрать на клавиатуре новое значение x и нажать <Enter>, например:

```
Текущее значение x=0.0302 Новое значение
  x=0.073
x = 0.0730
```

Чтобы предотвратить повторный вывод на экран введенного значения, необходимо строку с функцией **input** завершить символом “ ; ”.

В нашем случае нужно организовать выбор разных видов операторов в соответствии с отдельными выбранными параметрами. Для этого можно использовать оператор условного перехода:

```
if k==1,
x1 = input( [sprintf( 'Текущее значение x1 = %g', x1) ...
                                ' Новое значение x1= ']);
elseif k==2,
x2 = input( [sprintf('Текущее значение x2 = %g', x2) ...
                               ' Новое значение x2= ']);
elseif k==3,
x3 = input( [sprintf('Текущее значение x3 = %g', x3) ...
                               ' Новое значение x3= ']);
elseif k==4
x4 = input( [sprintf('Текущее значение x4 = %g', x4) ...
                               ' Новое значение x4= ']);
elseif k==5
x5 = input( [sprintf('Текущее значение x5 = %g', x5) ...
                               ' Новое значение x5= ']);
end
```

Чтобы можно было проконтролировать правильность ввода новых значений, обеспечить возможность их корректировки и последовательного изменения всех желаемых параметров, нужно, чтобы после ввода нового значения любого параметра на экране снова возникало то же меню, но уже со скорректированными значениями. При этом конец работы с меню должен наступить только при условии выбора последней альтернативы меню "Ничего не менять", соответствующей значению **k**, равному 6.

Поэтому предыдущие операторы следует заключить в цикл:

```
k=1;
while k<6
k = menu( ' Что менять? ', ...
sprintf (' Параметр1 x1 = %g', x1),...
sprintf (' Параметр2 x2 = %g', x2),...
sprintf (' Параметр3 x3 = %g', x3),...
sprintf (' Параметр4 x4 = %g', x4),...
sprintf (' Параметр5 x5 = %g', x5), ' Ничего не менять ');
if k==1,
x1 = input( [sprintf('Текущее значение x1 = %g', x1) ...
' Новое значение x1= ']);
elseif k==2,
x2 = input( [sprintf('Текущее значение x2 = %g', x2) ...
' Новое значение x2= ']);
elseif k==3,
x3 = input( [sprintf('Текущее значение x3 = %g', x3) ...
' Новое значение x3= ']);
elseif k==4
x4 = input( [sprintf('Текущее значение x4 = %g', x4) ...
' Новое значение x4= ']);
elseif k==5
x5 = input( [sprintf('Текущее значение x5 = %g', x5) ...
' Новое значение x5= ']);
end
end
```

6.4.6. Типовая структура и оформление **Script-файла**

1. Весь процесс диалогового изменения параметров удобно оформлять в виде отдельного Script-файла, к примеру, с именем "ScrFil_Menu", где под сокращением "ScrFil" понимается имя основного (собирательного) Script-файла.
2. Перед главным циклом программы, обеспечивающим возвращение к началу вычислений, необходимо поместить часть программы, которая задает первоначальные значения всех параметров.
3. В начале работы программы удобно вывести на экран краткую информацию о назначении программы, исследуемой математической модели с указанием исходных параметров. Это желательно также оформить в виде отдельного Script-файла, например, с именем "ScrFil_Zastavka".
4. При завершении работы программы желательно очистить рабочее пространство от введенных глобальных переменных, закрыть открытые программой графические окна (фигуры) и т.д. Эту завершающую часть тоже можно оформить как отдельный Script-файл, например, назвав его "ScrFil_Kin".

Типовая схема оформления **Script-файла** отдельной программы:

```
% <Обозначение Script-файла (ScrFil.m)>
% <Текст комментария с описанием
  назначения программы>
< Пустая строка >
% Автор < Фамилия И. О., дата создания,
  организация>
ScrFil_Zastavka
k = menu('Что делать? ','Продолжить
  работу', 'Закончить работу');
if k==1,
while k==1
ScrFile_Yadro
k = menu(' Что делать ? ',' Продолжить
  работу ', ' Закончить работу ');
end
end
ScrFil_Kin
```

6.5. Создание функций от функций

Существуют функции (процедуры) алгоритмы определения которых требуют вычисления других функций. К таким процедурам принадлежат:

- вычислений *интеграла* от функции, которые требуют указания имени М-файла, содержащего вычисления значения подынтегральной функции;
- *численного интегрирования* дифференциальных уравнений, использование которых требует указания имени М-файла, в котором вычисляются правые части уравнений в форме Коши;
- алгоритмов численного *вычисления корней* нелинейных алгебраических уравнений (нулей функций), которые нуждаются в указания файла-функции, нуль которого отыскивается;
- алгоритмов *поиска минимума* функции, которую, в свою очередь, надо задавать соответствующим М-файлом и т.п.

6.5.1. Процедура ***feval***

Чтобы первый, более общий алгоритм был приспособлен для любой функции, нужно, чтобы имя этой функции было некоторой переменной, принимающей определенное значение только при обращении к основному алгоритму.

В MatLAB любая функция (процедура), например, с именем FUN1, может быть выполнена не только с помощью обычного обращения:

[y1,y2,... ,yk] = **FUN1**(x1,x2,... ,xn),

а и при помощи специальной процедуры ***feval:***

[y1,y2,... ,yk] = ***feval****('FUN1',x1,x2,... ,xn),*

где имя функции FUN1 является уже одной из входных переменных (текстовой - и поэтому помещается в апострофы).

Преимуществом второй формы является возможность унифицировать обращение ко всем функциям имеющим одинаковое число входных и выходных параметров. При этом имя функции может быть произвольным и изменяться при повторных обращениях.

6.5.2. Пример создания процедуры от функции

Процедура численного интегрирования ОДУ методом Рунге-Кутта 4-го порядка

Пусть задана система ОДУ в форме Коши:

$$\frac{dy}{dt} = Z(y,t)$$

где ***y*** - вектор переменных состояния системы; ***t*** - аргумент (время); ***Z*** – вектор заданных функций.

Если значение вектора ***y*** в момент времени ***t*** известно, то общая формула, по которой может быть найден вектор ***yout*** значений переменных состояния системы в момент времени ***tout*** = *t* + *h* (где *h* - шаг интегрирования), имеет вид:

yout = y + h*F(y,t).

Функция ***F(y,t***) связана с вектором ***Z*** и может приобретать разный вид в зависимости от выбранного метода численного интегрирования.

Для метода Рунге-Кутта 4-го порядка выберем такую ее форму:

$F=(k_1+3\cdot k_2+3\cdot k_3+k_4)/8$

где

$$k_1 = Z(y,t)$$

$$k_2 = Z\left(y + h\cdot\frac{k_1}{3},\ \ t+\frac{h}{3}\right)$$

$$k_3 = Z\left(y + h\cdot k_2 - h\cdot\frac{k_1}{3},\ \ t+\frac{2h}{3}\right)$$

$$k_4 = Z\left(y + h\cdot k_3 - h\cdot k_2 - h\cdot k_1,\ \ t+h\right)$$

Создадим М-файл процедуры, которая осуществляет эти вычисления, назвав его "rko43":

```
function [tout, yout] = rko43(Zfun,h,t,y)
% Расчет промежуточных значений производных
k1 = feval('Zfun', t, y);
k2 = feval('Zfun', t+h/3, y+h/3*k1);
k3 = feval('Zfun', t+2*h/3, y+h*k2-h/3*k1);
k4 = feval('Zfun', t+h, y+h*(k3+k1-k2));
% Расчет новых значений вектора переменных состояния
tout = t + h;
yout = y + h*(k1 + 3*k2 + 3*k3 + k4)/8;
```

6.5.3. Методы численного интегрирования систем ОДУ (решатели) установленные в Matlab

1) ***с фиксированным шагом:***

- *discrete (no continuous states)* - дискретный (не непрерывные состояния);
- *ode5 – метод Дормана-Пренса (пятого порядка);*
- *ode4 – метод Рунге-Кутта (четвертого порядка);*
- *ode3 – метод Богацкого-Шампена (третьего порядка);*
- *ode2 – метод Хойне (второго порядка);*
- *ode1 – метод Ейлера (первого порядка).*

2) ***с переменным шагом*** :

ode45 *– явные методы Рунге-Кутта 4-го и 5-го порядка.* Это классический метод, рекомендуемый для начальной пробы решения. Во многих случаях он дает хорошие результаты;

ode23 *– явные методы Рунге-Кутта 2-го и 4-го порядка.* При умеренной жесткости системы ОДУ и низких требованиях к точности этот метод может дать выигрыш в скорости решения;

ode113 *– метод Адамса-Башворта-Мултона переменного порядка.* Это адаптивный метод, который может обеспечить высокую точность решения;

ode15s *– метод переменного порядка (от 1 до 5, по умолчанию 5),* использующий формулы численного дифференцирования. Это адаптивный метод, его стоит применять, если решатель ode45 не обеспечивает решения;

ode23s *– метод, использующий модифицированную формулу Розенброка 2-го порядка.* Может обеспечить высокую скорость вычислений при низкой точности решения жесткой системы дифференциальных уравнений;

ode23t *– метод трапеций с интерполяцией.* Этот метод дает хорошие результаты при решении задач, описывающих колебательные системы с почти гармоническим выходным сигналом;

ode23tb *– неявный метод Рунге-Кутта в начале решения и метод, использующий формулы обратного дифференцирования 2-го порядка в последующем.* Несмотря на сравнительно низкую точность, этот метод может оказаться более эффективным, чем *ode15s*.

6.5.4. Использование решателей систем ОДУ MatLab

Функция для решения систем дифференциальных уравнений описывается следующим образом:

[T,Y] = solver(@F, tspan, y0, options) – где вместо solver подставляем имя конкретного решателя – интегрирует систему дифференциальных уравнений вида $y'=F(t,y)$ на интервале tspan с начальными условиями y0.

@F – дескриптор ODE-функции.

Каждая строка в массиве решений Y соответствует значению времени, возвращаемому в векторе-столбце T;

options – аргумент, создаваемый функцией odeset позволяет изменить параметры, установленные по умолчанию. Обычно используемые параметры включают допустимое значение относительной погрешности RelTol (по умолчанию 1e-3) и вектор допустимых значений абсолютной погрешности AbsTol (все компоненты по умолчанию равны 1e-6);

Пример задания параметров:

```
options=odeset('RelTol',1e-3, 'AbsTol',1e-6,
   'MaxStep',1e-4, 'InitialStep',1e-4);
```

[T,Y] = solver(@F,tspan,y0,options,p1,p2...) – дает решение, подобное описанному выше, передавая дополнительные параметры p1, p2,... в m-файл F всякий раз, когда он вызывается.

Алгоритм решения ОДУ рассмотрим на примере решения уравнения Ван-дер-Поля, записанного в виде системы из двух дифференциальных уравнений:

$$\begin{cases} \dfrac{dy_1}{dt} = y_2 \\ \dfrac{dy_2}{dt} = 100 \cdot \left(1 - y_2\right)^2 \cdot y_2 - y_1 \end{cases}$$

При начальных условиях

$$y_1\left(0\right) = 0,$$

$$y_2\left(0\right) = 1.$$

Перед решением нужно записать систему дифференциальных уравнений в виде ode-функции. Для этого составим m-File с названием vdp100.m:

```
function dydt = vdp100(t,y)
dydt = zeros(2,1); % вектор-столбец из нулевых значений, число которых равно числу уравнений
dydt(1) = y(2);
dydt(2) = 100*(1-y(2))^2)*y(2) -y(1);
```

Теперь численно решим эту систему с помощью решателя ode15s

```
» [T,Y]=ode15s(@vdp100,[0 2],[0 1])
```

Построим график полученного решения:

```
» plot(T,Y)
```