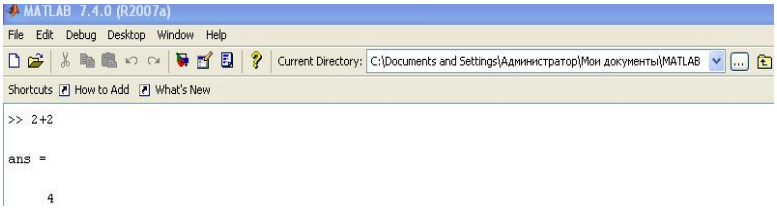
ВВЕДЕНИЕ

Ядро MATLAB состоит из встроенных функций и стандартных m-функций. Он может дополняться специализированными пакетами (тулбоксами). Из них в пособии используется тулбокс **Symbolic math**.



При запуске MATLAB открывается *рабочий стол* (*Desktop*). На нем можно поместить одно или несколько *окон*. Основное (на рисунке) **Command Window**. В его *командной строке* >> набираются команды и возвращаются результаты. В окне **Command History** сохраняются все ранее набранные в командной строке команды. В окне **Workspace** хранится список всех переменных текущего сеанса, их размерность, занимаемый объем памяти и тип данных. Окно **Current Directory** содержит список файлов (их можно открыть двойным щелчком левой кнопки). При этом файлы с расширениями **.fig**, **.mdl** открываются в графическом окне, а с расширением **.m** – в окне редактора отладчика. Настройка вида рабочего стола производится с помощью команд меню **Desktop**. Команды меню **File** позволяют, в частности, создавать новые m-, fig-, mdl-файлы. Команды меню **Window** позволяют осуществлять переходы между окнами и закрывать их.

Перечислим несколько способов получения справки.

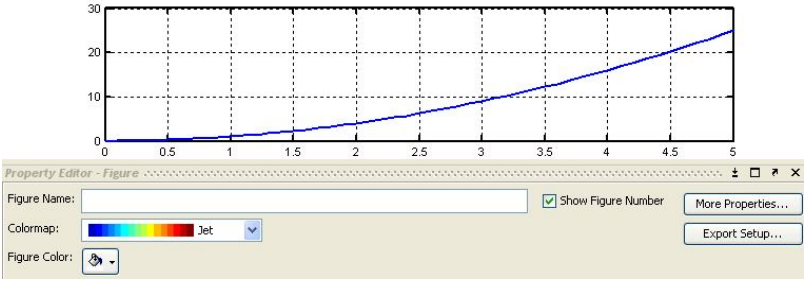
1. Команда >> **help** выводит полный список разделов, сгруппированных в папки.
2. Команда >> **help + имя (**функции, оператора, команды, пакета, папки) возвращает информацию по заданному имени.
3. Команда >> **which fun** (**file**.␣) указывает путь к функции или файлу с соответствующим расширением.
4. Команда >> **doc fun** дает подробные сведения по функции **fun**.
5. Команда >> **type file** выводит в командное окно полный текст М-файла.

6. На рабочем столе кнопка ? или последовательный выбор команд **Help|**
MATLAB Help открывают справочное окно с двумя панелями: левая **Help Navigator** (список справочных пакетов расширения) и правая, в которой отображается запрашиваемая информация.

Команда >> **save** (**load**) создает для хранения переменных рабочей области (загружает в рабочую область) файл с именем **matlab.mat**. Команда **>> save** (**load**) **fname** сохраняет (загружает) переменные с именем **fname.mat**.

К демонстрационным примерам можно добраться либо с помощью команды >> **demo**, либо выбором команд **Help** → **Demos**.

Последовательности команд **Edit|Current Object Properties, Edit| Axes Properties, Edit |Figure Properties**, дают удобные, но несколько разные по целям инструменты редактирования графического окна. Последняя из них и кнопка **Export Setup** позволяют изменять толщину линии.



Возврат к исходному окну осуществляется кнопкой ↗.

К этому же окну приводит кнопка ▣.

Глава 1. MATLAB

§ 1.1. Матрицы, массивы, арифметические операции

Всюду ниже стрелка $\Rightarrow$ – означает нажатие клавиши «enter».

Форматы отображения чисел в командном окне. Тип данных, в которых для представления числа используется формат с плавающей запятой, называется **double**.

Пр. >> 3.5e+12 (означает $3.5 \cdot 10^{12}$); -10.2; 34e-1 $\Rightarrow$ ans = 3.4000.

По умолчанию для вывода вещественных чисел используется **format short** – четыре десятичные цифры после запятой. Для полного представления следует ввести команду **format long** (сохраняет до 15 знаков). Для перехода к представлению чисел в виде обыкновенных дробей используют команду **format rat**.

Пр. >> 3/7 $\Rightarrow$ ans = 0.4286

>> format long >> 3/7 $\Rightarrow$ ans = 0.428571428 571429

>> format rat >> 0.428571428 571429 $\Rightarrow$ ans =3/7

>> sqrt(3) $\Rightarrow$ ans = 1351/780.

Специальные переменные. ans (*answer* – ответ) – обозначение выводи мой непоименованной величины. **Inf** – машинная бесконечность.

NaN – неопределенность. **pi** – число 3,14159. **i, j** – мнимая единица.

Пр. >> **0/0** $\Rightarrow$ ans=NaN >> **pi/0** $\Rightarrow$ ans= Inf >> i^2 $\Rightarrow$ ans= -1

>> **exp**(1) $\Rightarrow$ ans=2.7183 >> **exp**(i) $\Rightarrow$ ans = 0.5403 + 0.8415i.

Регулярный одномерный массив (то есть матрица-строка) задается либо с шагом **h**: x=**[a:h:b]**, либо с числом узлов **n**: x=**linspace**(a,b,n). При этом linspace(a,b):=linspace(a,b,100).

Пр. >> x=[-1:0.5:1] $\Rightarrow$ -1 -0.5 0 0.5 1

Пр. >> x=-1:0.5:1 $\Rightarrow$ -1 -0.5 0 0.5 1

Пр. >> x=linspace(-1,1,5) $\Rightarrow$ -1 -0.5 0 0.5 1

Матрица задается квадратными скобками. Строки отделяются «;» или «enter». ***Пр***. >>A=[1 2; 2 3] $\Rightarrow$ A= $\begin{matrix} 1 & 2 \\ 2 & 3 \end{matrix}$. A=[1, 2; 2, 3] $\Rightarrow$ A= $\begin{matrix} 1 & 2 \\ 2 & 3 \end{matrix}$.

Быстрое формирование матриц. rand (random – случайный), **rand(n)**, **rand(m,n)**, **rand(m,n,p)** – p экземпляров матриц размера

$m \times n$ с элементами из интервала $(0,1)$. **randint** (integer – целое число), **randint(n)**, **randint(m,n)**, **randint (m,n,p)** – матрица размера $m \times n$ с целочисленными элементами из отрезка $[0, p-1]$.

Пр.1 >> rand $\Rightarrow$ ans=0.8147. ***Пр.2*** >> rand int(2) $\Rightarrow$ ans = $\begin{matrix} 0 & 0 \\ 0 & 1 \end{matrix}$.

Пр.3 >> rand(2,1) $\Rightarrow$ ans = $\begin{matrix} 0.3112 \\ 0.5285 \end{matrix}$. randint(2,3,5) $\Rightarrow$ ans = $\begin{matrix} 4 & 1 & 2 \\ 3 & 2 & 0 \end{matrix}$.

Объединение (конкатенация) матриц с одинаковым количеством строк (или столбцов) >> A; B; C=[A B] (D=[A;B]).

Пр. >> A=[1 2;2 3]; B=[1 2 3;3 4 5]; C=[A B] $\Rightarrow$ C = $\begin{matrix} 1 & 2 & 1 & 2 & 3 \\ 2 & 3 & 3 & 4 & 5 \end{matrix}$.

Специальные массивы eye(n), eye(m,n) – единичная матрица, **zeros(n), zeros(m,n)** – нулевая матрица, **ones(n), ones(m,n)** – матрицы из единиц. M=**diag**(v) – диагональная матрица с элементами из массива v, **diag**(M) – диагональ матрицы M.

Пр.1 >> eye(2,3) $\Rightarrow$ ans = $\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{matrix}$.

Пр.2 >> v=[-1 2]; M=diag(v) $\Rightarrow$ M= $\begin{matrix} -1 & 0 \\ 0 & 2 \end{matrix}$; diag(M) $\Rightarrow$ ans = $\begin{matrix} -1 \\ 2 \end{matrix}$.

Арифметические операции над массивами.

Пр. >> C=[0 -2; 3 1];

>> A+C $\Rightarrow$ ans = $\begin{matrix} 1 & 0 \\ 5 & 4 \end{matrix}$; >> A*B $\Rightarrow$ ans = $\begin{matrix} 7 & 10 & 13 \\ 11 & 16 & 21 \end{matrix}$;

>> 3*C $\Rightarrow$ ans = $\begin{matrix} 0 & -6 \\ 9 & 3 \end{matrix}$.

Арифметические операции “.*”, ”./”, “.\”, “.^” – поэлементные мультипликативные операции над массивами.

Пр. >> A=[1 2;2 3]; B=[0 -2; 3 1]; >> A*B $\Rightarrow$ ans = $\begin{matrix} 6 & 0 \\ 9 & -1 \end{matrix}$;

>> A.*B $\Rightarrow$ ans = $\begin{matrix} 0 & -4 \\ 6 & 3 \end{matrix}$; >> A.^2 $\Rightarrow$ ans = $\begin{matrix} 1 & 4 \\ 4 & 9 \end{matrix}$;

>> A^2 $\Rightarrow$ ans = $\begin{matrix} 5 & 8 \\ 8 & 13 \end{matrix}$.

Приоритет выполнения операций. Возведение в степень $\rightarrow$ Умножение и деление $\rightarrow$ Сложение и вычитание. Операции одинакового приоритета выполняются в порядке слева направо. Для изменения порядка нужно использовать скобки. ***Пр***. >> 2-3*5+6/3 $\Rightarrow$ ans = – 11; >> (2-3)*5+6/3 $\Rightarrow$ ans = – 3.

Операции с матрицами. rank(A) – ранг A. **det**(A) – определитель A. **eig**(A) – собственные числа A (*eigen* – собственный). $A.'(A')$ – *транспонированная* (*эрмитова* или *комплексно-сопряженная*) матрица: $A' := \overline{A.'}$ **inv**(A) – обратная к квадратной невырожденной матрице A (inverse –обратный). **pinv**(A) – псевдообратная матрица к матрице A(m,n) (Мура-Пенроуза псевдообращение матрицы A). В случае m<n – это правая обратная R=R(n,m) к матрице A со свойством: $RA = (RA)'$. В случае m>n – это левая обратная L=L(n,m) со свойством: $AL = (AL)'$. **[U,S,V]=svd**(A) – разложение (с помощью сингулярных чисел – singular value decomposition) матрицы A на произведение $A = U * S * V'$ диагональной S и двух *унитарных* матриц U, V (то есть $U * U'$, $V * V'$ - единичные матрицы). **[B,J]=jordan**(A) -матрица перехода и ЖНФ. **expm**(A) – экспонента от матрицы A.

Пр.1 >> A=[1 2;2 3]; det(A) $\Rightarrow$ ans =-1; >> eig(A) $\Rightarrow$ ans = $\begin{matrix} -0.2361 \\ 4.2361 \end{matrix}$;

Пр.2 >> A=[2-i 1; i 2; -i 1]; A.' $\Rightarrow$ ans = $\begin{matrix} 2-i & i & -i \\ 1 & 2 & 1 \end{matrix}$;

A'$\Rightarrow$ ans = $\begin{matrix} 2+i & -i & i \\ 1 & 2 & 1 \end{matrix}$.

Пр.3 >> A=[2 1; 0 2; 3 1];

pinv(A) $\Rightarrow$ ans = $\begin{matrix} 0.1321 & -0.1887 & 0.2453 \\ 0.0566 & 0.4906 & -0.0377 \end{matrix}$.

Проверка: >> pinv(A)*A $\Rightarrow$ ans = $\begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$.

>> A*pinv(A) $\Rightarrow$ ans = $\begin{matrix} 0.3208 & 0.1132 & 0.4528 \\ 0.1132 & 0.9811 & -0.0755 \\ 0.4528 & -0.0755 & 0.6981 \end{matrix}$ – симметричная матрица.

Пр.4 >> [U,S,V]=svd(A) $\Rightarrow$ U = $\begin{matrix} -0.5660 & 0.0199 & -0.8242 \\ -0.2338 & -0.9625 & 0.1374 \\ -0.7905 & 0.2705 & 0.5494 \end{matrix}$

S = $\begin{matrix} 3.9501 & 0 \\ 0 & 1.8430 \\ 0 & 0 \end{matrix}$ V = $\begin{matrix} -0.8870 & 0.4618 \\ -0.4618 & -0.8870 \end{matrix}$.

Проверка: >> U*U' $\Rightarrow$ ans = $\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$. >> V*V' $\Rightarrow$ ans = $\begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$.

Пр.5 >> A=[1 2;2 3];

[B,J]=jordan(A) $\Rightarrow$ B = $\begin{matrix} 0.28, & 0.72 \\ 0.45, & -0.45 \end{matrix}$ J = $\begin{matrix} 4.23, & 0 \\ 0, & -0.23 \end{matrix}$.

>> expm(A) $\Rightarrow$ ans = $\begin{matrix} 19.6800 & 30.5651 \\ 30.5651 & 50.2452 \end{matrix}$.

Решение СЛАУ. Напомним, что система линейных алгебраических уравнений (СЛАУ) A*X=B называется ***квадратной***, если число уравнений m совпадает с числом неизвестных n, ***недопределенной***, если $m < n$ и ***переопределенной***, если $m > n$. СЛАУ называется ***определенной***, если она имеет ровно одно решение, ***неопределенной***, если решений больше одного (а значит согласно теории – бесконечное чис-

ло), ***совместной***, если хотя бы одно решение и ***несовместной***, если решений нет. СЛАУ совместна $\Leftrightarrow$ $\mathrm{rank}\left(\left[\mathrm{A\ B}\right]\right)=\mathrm{rank}\left(\mathrm{A}\right)$.

Если A*X=B, то X=A\B. Способ применим только к совместным системам. При этом для неопределенных систем находится только одно решение.

Пр.1 >> A=[1 2;2 3]; B=[-2 ;1]; rank(A) $\Rightarrow$ ans =2.

>> rank([A B]) $\Rightarrow$ ans = 2. >> X=A\B $\Rightarrow$ ans = $\begin{matrix} 8 \\ -5 \end{matrix}$. >> A*X $\Rightarrow$ ans = $\begin{matrix} -2 \\ 1 \end{matrix}$.

Пр.2 >> A=[4 5 2;4 5 1]; B=[2; 3]; >> rank(A) $\Rightarrow$ ans =2.

>> rank([A B]) $\Rightarrow$ ans = 2. >> X=A\B $\Rightarrow$ X = $\begin{matrix} 0 \\ 0.8 \\ -1 \end{matrix}$. >> A*X $\Rightarrow$ ans = $\begin{matrix} 2 \\ 3 \end{matrix}$.

Пр.3 >> A=[2 1; 0 2; 2 3]; B=[-2 ;1;-1]; >> rank(A) $\Rightarrow$ ans =2.

>> rank([A B]) $\Rightarrow$ ans = 2. >> X=A\B $\Rightarrow$ X = $\begin{matrix} -1.25 \\ 0.5 \end{matrix}$. >>A*X $\Rightarrow$ ans = $\begin{matrix} -2 \\ 1 \\ -1 \end{matrix}$.

Для нахождения общего решения неопределенной системы нужно предварительно *свободные* неизвестные (переменные) объявить *символьными* (смотри ниже).

Пр. >> A=[1 2 3 0;2 3 0 -2]; B=[-2 ;1]; (считаем *базисными* первые две переменные, а тогда оставшиеся две – свободные) >> syms u v $\Rightarrow$
>> A1=[1 2;2 3]; B=[-2-3*u ; 1+2*v]; >> rank(A1) $\Rightarrow$ ans =2.

>> rank(A) $\Rightarrow$ ans =2. >> A1\B $\Rightarrow$ ans = $\begin{matrix} 8+9*\mathrm{u}+4*\mathrm{v} \\ -5-6*\mathrm{u}-2*\mathrm{v} \end{matrix}$ – это есть общее решение СЛАУ $\begin{cases} x_1+2x_2+3x_3=-2 \\ 2x_1+3x_2-2x_4=1 \end{cases}$.

§ 1.2. Программирование в MATLAB

М-файлом называется любой текстовый файл, содержащий произвольный код на М-языке и имеющий расширение m. Различают два типа М-файлов.

М-файлом-функцией (***файлом-функцией***) называется М-файл, содержащий инструкцию на М-языке, которая реализует независимый от Command Windows фрагмент кода с помощью параметров только этого файла.

Пр.1 Программа вычисления векторного произведения двух векторов, заданных своими координатами, выглядит следующим образом:

```
function ros=v_prod(x1,y1,z1,x2,y2,z2)
% векторное произведение
ros=[z2*y1-y2*z1 x2*z1-z2*x1 y2*x1-x2*y1];
```

>> v_prod(1, 2, 3, 4, 5, 6) $\Rightarrow$ ans = -3 6 -3.

Пр.2 Созданная программа `v_prod` может быть использована при создании файла-функции вычисления расстояния между прямыми в пространстве, заданными каноническими уравнениями:

$$\frac{x-x_i}{m_i}=\frac{y-y_i}{n_i}=\frac{z-z_i}{p_i},\ i=1,2:$$

```
function
ros=ro_lines(x1,y1,z1,m1,n1,p1,x2,y2,z2,m2,n2,p2)
A=v_prod(m1,n1,p1,m2,n2,p2);B=[x2-x1 y2-y1 z2-z1];
% расстояние между двумя прямыми
ros=abs(A*B')/sqrt(A*A');
```

Здесь команда `abs` возвращает норму вектора.

Найдем расстояние между прямыми $\frac{x-1}{4}=\frac{y-2}{5}=\frac{z-3}{6}$, $\frac{x-1}{4}=\frac{y-2}{5}=\frac{z-3}{6}$, $\frac{x-6}{9}=\frac{y-7}{1}=\frac{z-8}{2}$.

>> ros=ro_lines(1,2,3,4,5,6,6,7,8,9,1,2) $\Rightarrow$ ros =0.7288.

Пр.3 Следующая файл-функция имеет несколько выходных аргументов и рассчитывает, сколько в данном количестве секунд будет часов и минут:

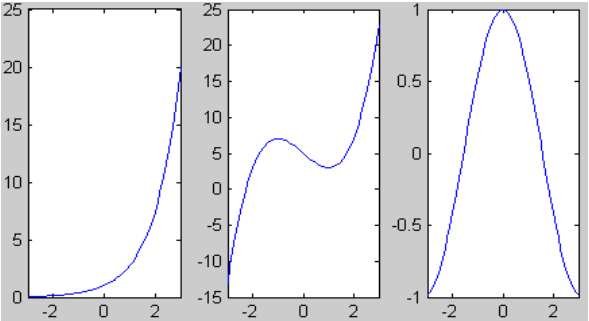
```
% пересчет секунд в часы и минуты
% команда floor(a) – максимальное целое, не превышающее a
function [hour,minute,second]=hms(sec)
hour=floor(sec/3600);
minute=floor((sec-hour*3600)/60);
second=sec-hour*3600-minute*60;
```

>> [h,m,s]=hms(100000) $\Rightarrow$ h =27 m =46 s =40.

М-файлом-сценарием (***файлом-программой***) называется М-файл, содержащий инструкцию на М-языке, подлежащую исполнению в автоматическом режиме. Он обрабатывает все переменные из Workspace.

Пр. Создадим файл-программу рисования графиков трех функций.

```
x=-3:0.1:3;
f=(x).^3-3*x+5;
subplot(1,3,2);
plot(x,f);
g=cos(x);
subplot(1,3,3);
plot(x,g);
r=exp(x);
subplot(1,3,1);
plot(x,r);
```



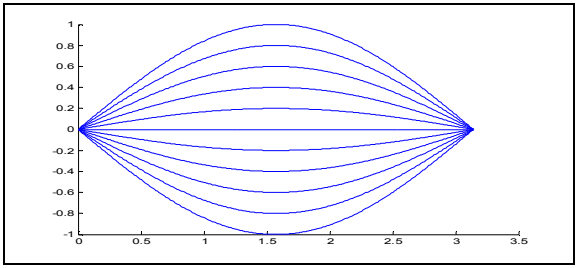
Первые две координаты команды **subplot** дают размер матрицы открывающихся графических окон. Последняя координата определяет порядок расположения графиков функций. Изображение графиков появляется при наборе имени созданного файла-программы.

В приведенных примерах операции в М-фалах выполняются *последовательно*. Рассмотрим конструкции языка программирования Matlab для создания в программах *повторяющихся циклически действий*.

Циклом типа **for…end** называется цикл с *определенным* количеством операций. В качестве примера создадим файл-программу пучка синусоид.

Пр.

```
% создаем графическое окно
figure
% создаем цикл
x=0:0.01:pi;
for a=-1:0.2:1
    y=a*sin(x);
    hold on
    plot(x,y)
end
```



Циклом типа **while…end** называется цикл с *неопределенным* количеством операций. Он выполняется до тех пор, пока выполняется условие цикла.

Пр. Приведем файл-программу с именем **sum1** вычисления суммы ряда $\sum_{n=1}^{\infty}\frac{1}{n^2}$ с ограничением на величину n-го члена.

```
s=0;n=1;a=1;
while a>1e-5;
    s=s+a;
    n=n+1;
    a=1/n^2;
end
n=n-1 % наименьший номер с a>1e-5
s=s-1/n^2 % соответствующая частичная сумма
```

>> sum1 $\Rightarrow$ n = 316 s = 1.6418.

Служебные слова условного оператора **if, elseif, else** продемонстрируем на примере создания файла-функции **сигнум числа:**

```
% М-файл-функция вычисления сигнум числа
function y=signum(x)
if x>0 y=1;
elseif x==0 y=0;
else y=-1;
 end
```

>> signum(-10) $\Rightarrow$ ans =-1.

Команда **switch** осуществляет *ветвление по нескольким условиям.*

Пр . Следующая файл-функция выделяет среди вводимых чисел только единицы и двойки.

```
%выделение единиц и двоек
function y=count(x)
switch x
    case 1
        y='one';
    case 2
        y='two';
        otherwise
        y='many';
end
```

>> y=count(98) $\Rightarrow$ y = many. >> y=count(2) $\Rightarrow$ y = two.

§ 1.3. Векторная алгебра и аналитическая геометрия

Длину вектора можно находить двумя способами.

Пр.1 >> a=[1, 2, 3]; **norm**(a) $\Rightarrow$ ans =3.7417.

Пр.2 >> a0=sqrt(1^2+2^2+3^2) $\Rightarrow$ a0 =3.7417.

Скалярное произведение векторов также вычисляется двумя способами:

Пр.1 >> b=[3,-2,0]; **dot**(a,b) $\Rightarrow$ ans = -1. ***Пр.2*** >> c=a*b' $\Rightarrow$ c = -1.

Векторное произведение вычисляем с помощью встроенной функции **cross.**

Пр. >> c=cross(a,b) $\Rightarrow$ c = 6 9 -8.

Проверка: >> c*a' $\Rightarrow$ ans = 0; >> c*b' $\Rightarrow$ ans = 0, то есть векторное произведение ортогонально векторам a, b.

Смешанное произведение вычисляем в соответствии с определением.

Пр. >> c=[-2 2 -1]; d=dot(cross(a,b),c) $\Rightarrow$ d =14.

Решим несколько задач аналитической геометрии.

Пр.1 Расстояние от точки $M_0(x_0, y_0, z_0)$ до прямой $l: \dfrac{x-x_1}{m_1} = \dfrac{y-y_1}{n_1} = \dfrac{z-z_1}{p_1}$ проходящей через точку $M_1(x_1, y_1, z_1)$ и с направляющим вектором $k_1 = \{m_1, n_1, p_1\}$, вычисляется по формуле $\rho(M_0, l) = \dfrac{\left|\left[\overline{M_0M_1}, \overline{k}_1\right]\right|}{\left|\overline{k}_1\right|}$. Положим $M_0 = (1,2,3)$, $l_1: \dfrac{x-3}{2} = \dfrac{y+1}{0} = \dfrac{z-2}{-4}$, $M_1 = (3,-1,2)$.

>>M0M1=[2 -3 -1]; k1=[2 0 -4]; p=norm(cross(M0M1,k1))/norm(k1) $\Rightarrow$ p= = 3.2863.

Пр.2 *Расстояние между двумя прямыми* l_i $\dfrac{x-x_i}{m_i} = \dfrac{y-y_i}{n_i} = \dfrac{z-z_i}{p_i}$, $i = 1,2$ вычисляем по формуле:

$$\rho(l_1,l_2)=\frac{\left|\left(\overline{M_1M_2},\bar{k}_1,\bar{k}_2\right)\right|}{\left|\left[\bar{k}_1,\bar{k}_2\right]\right|}=\frac{\left|\left(\left[\bar{k}_1,\bar{k}_2\right],\overline{M_1M_2}\right)\right|}{\left|\left[\bar{k}_1,\bar{k}_2\right]\right|}.$$

Найдем с помощью этой формулы расстояние между конкретными прямыми в MATLAB.

```
>> M1=[1 2 3]; k1=[2 0 4]; M2=[3 -1 2]; k2=[-1 -3 5]; M1M2=[2 -3 -1];
p=norm(dot(cross(k1,k2),M1M2))/norm(cross(k1,k2)) ⇒ p =3.7131.
```

Задача. Выведем формулы вычисления координат точки $M_1(x_1,y_1,z_1)$, симметричной к точке $M_0(x_0,y_0,z_0)$ относительно плоскости L: $Ax+By+Cz+D$. Так как вектор $N=(A,B,C)$ перпендикулярен L, то прямая $\begin{cases} x=At+x_0 \\ y=Bt+y_0 \\ z=Ct+z_0 \end{cases}$ проходит через точку $M_0(x_0,y_0,z_0)$ и тоже перпендикулярна L. Подставляя эти уравнения в уравнение плоскости, находим $t_0=-\dfrac{Ax_0+By_0+Cz_0+D}{A^2+B^2+C^2}$ и координаты точки пересечения прямой и плоскости L: $\begin{cases} x'=At_0+x_0 \\ y'=Bt_0+y_0 \\ z'=Ct_0+z_0 \end{cases}$. Так как эта точка является серединой отрезка M_0M_1, то $\begin{cases} 2x'=x_1+x_0 \\ 2y'=y_1+y_0 \\ 2z'=z_1+z_0 \end{cases}$. Из последних двух равенств получаем искомые формулы $\begin{cases} x_1=2At_0-x_0 \\ y_1=2Bt_0-y_0 \\ z_1=2\tilde{N}t_0-z_0 \end{cases}$.

Приведем пример на использование этих формул.

Пр. `>> N=[1 2 3]; D=7; M0=[-5 -3 8]; t0=-(dot(N,M0)+D)/dot(N,N); M1=2*t0*N-M0 ⇒`

`M1 = 2.1429 -2.7143 -16.5714.`

Для следующей задачи нам понадобятся теоретические результаты.

Составим по уравнениям двух прямых $l_i: \dfrac{x-x_i}{m_i}=\dfrac{y-y_i}{n_i}=\dfrac{z-z_i}{p_i}$, $i=1,2$, матрицу $A=\begin{pmatrix} x_1-x_2 & y_1-y_2 & z_1-z_2 \\ m_1 & n_1 & p_1 \\ m_2 & n_2 & p_2 \end{pmatrix}$. Тогда условие их скрещиваемости: $rang(A)=3$.

Условие пересечения этих прямых: $rang(A)=rang\begin{pmatrix} m_1 & n_1 & p_1 \\ m_2 & n_2 & p_2 \end{pmatrix}=2$. Условие параллельности и несовпадения: $rang(A)=2,\ rang\begin{pmatrix} m_1 & n_1 & p_1 \\ m_2 & n_2 & p_2 \end{pmatrix}=1$. Условие совпадения: $rang(A)=1$.

Пр. Исследуем взаиморасположение двух прямых

$l_1: \dfrac{x-3}{2}=\dfrac{y+1}{0}=\dfrac{z-2}{-4}$, $\quad l_2: \dfrac{x-5}{1}=\dfrac{y-2}{3}=\dfrac{z-2}{2}$.

>>A=[-2 -3 0;2 0 -4;1 3 2]; rank(A)$\Rightarrow$ans=2. >>rank([2 0 -4;1 3 2])$\Rightarrow$ans=2.

Следовательно, эти прямые пересекаются.

§ 1.4. Элементарные функции и операции над ними

Встроенные основные элементарные функции. **exp** (exp(x)=e^x), **log** (log(x):=$\ln x$), **log10** (log10(x):= $\lg x$), **log2** (log2(x)= $\log_2 x$), **sqrt** (sqrt(x)=$\sqrt{x}$), **sin** (sin(x)=$\sin x$), **cos** (cos(x)=$\cos x$**),** **tan** (tan(x)=$tg\,x$), **asin** (asin(x)=$\sin x$), **sinh** (sinh(x)=$sh\,x$), **asinh** (asinh(x)=$arcsh\,x$). В тригонометрических функциях используются радианы, а не градусы.

Пр.1 >> pi^exp(1) ⇒ ans =22.4592; >> format long ⇒ pi^exp(1) ans= =22.45 9157718361052.
Пр.2 >> cos(pi) ⇒ ans = -1; >> tan(pi/2) ⇒ 1.6331e+016;
>> tan(sym('pi/2')) ⇒ ans =Inf.
Пр.3 >> log(exp(2))+sqrt(4) ⇒ ans = 4.
Пр.4 >>log2(3) ⇒ ans = 1.5850; >> 2^ans ⇒ ans = 3.

Элементарные функции, задаваемые пользователем (*function handles*)
Пр.1 >> f=@(x) x^2-x; f(5) ⇒ ans =20.
Пр.2 >> f=@(x) x.^2-x; f(1:1:5) ⇒ ans = 0 2 6 12 20.
Пр.3 >> g=@ (x,y) x^2+y^3 ; f(1)+g(1,2) ⇒ ans = 9.
В старых версиях MATLAB используют команду **inline.**
Пр.4 >> f=inline('x^2-x', 'x'); f(5) ⇒ ans = 20.
Пр.5 >> g=inline('x^2+y^3','x','y'); f(1)+g(1,2) ⇒ ans = 9.

Вычисления с рациональными функциями. *Полином* (*многочлен*) *n* -ой степени a(z) задается массивом своих коэффициентов, начиная со старшего a=[a(0) … a(n)]. Функция **polyval** возвращает значения многочлена, заданного таким способом, в точках массива.
Пр. >> a=[1 -5 3 9 0]; **polyval**(a, [0 1]) ⇒ ans = 0 8 – значения многочлена, заданного своими коэффициентами **a,** в точках 0, 1.

Функция **roots** вычисляет нули многочлена (то есть корни соответствующего алгебраического уравнения), заданного массивом коэффициентов **a**. Функция **poly** наоборот возвращает массив коэффициентов многочлена по известным его нулям. Эта же функция генерирует коэффициенты *характеристического многочлена* квадратной матрицы.
Пр.1 >> roots(a) ⇒ ans = 0 3 3 -1.
Пр.2 >> poly([0 3 3 -1]) ⇒ ans = 1 -5 3 9 0.
Пр.3 >> A= [1 2;3 4]; p=poly(A) ⇒ p= 1 -5 -2.
Пр.4 >> A^2-5*A-2*A^0 ⇒ ans = $\begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix}$ – проверка теоремы Кэли.

Рациональная функция $a(z)/b(z)$ задается с помощью функции r=**tf**(a,b) (*transfer function* – передаточная функция) и массивов коэффициентов числителя и знаменателя. *Нули* рациональной функции вычисляются с помощью **zero**(tf(a,b)), а *полюсы* – с помощью **pole**(tf(a,b)).
Пр. >> a=[1 -5 3 9 0]; b=[1 -4 5 -2]; zero(tf(a,b)) ⇒ ans = 0 3 3 -1.
>> pole(tf(a,b)) ⇒ ans = 2 1 1.

Команда [r p k]=**residue**(a,b) – разложение $a(z)/b(z)$ на сумму *простейших дробей*: **r** – числители простейших дробей; **p** – полюсы простейших дробей (нули знаменателя); **k** – вектор коэффициентов целой части дроби.

Пр. >> [r,p,k]=residue(a,b) $\Rightarrow$ r = 6 -12 -8 p = 2 1 1 k = 1 -1.

То есть

$$\frac{z^4-5z^3+3z^2+9z}{z^3-4z^2+5z-2}=z-1+\frac{6}{z-2}-\frac{12}{z-1}-\frac{8}{(z-1)^2}.$$

Вычетами рациональной функции в полюсах **2, 1** являются соответственно коэффициенты **6 , -12**.

Описание символьных объектов производится командой **syms**:

>> syms pi a b x y z .

Пр.1 >> tan(pi/2) $\Rightarrow$ ans = Inf.

Пр.2 >> **expand**((x+a)*(y-b) $\Rightarrow$ ans =x*y-x*b+a*y-a*b (*expand* – раскладывать).

Пр.3 >> f=exp(x); g=sin(y); h=**compose**(f,g) $\Rightarrow$ h =exp(sin(y)).

Пр.4 >> **diff** (x*sin(x)) $\Rightarrow$ ans = sin(x)+x*cos(x) (*differential coefficient* – производная); **diff** (x^3-x^2, 2) $\Rightarrow$ ans = 6*x-2 (второй аргумент указывает порядок производной); **diff**(sin(a*x), a, 2) $\Rightarrow$ ans =-sin(a*x)*x^2 (второй аргумент **a** указывает переменную, по которой берется частная производная, третий аргумент ***2*** указывает порядок этой производной).

Пр.5. >> u=x^2+z; v=x*y*z; J=**jacobian**([u,v], [x y z]) $\Rightarrow$ J = $\begin{matrix}[2*x, & 0, & 1\ ]\\ [y*z, & x*z, & x*y]\end{matrix}$ – *матрица Якоби* отображения с координатными функциями u=u(x,y,z), v=v(x,y,z).

Пр.6 >> w=a*x*y; H=jacobian([diff(w,x), diff(w,y)], [x y]) $\Rightarrow$ H = $\begin{matrix}[0 & a]\\ [a & 0]\end{matrix}$

– *матрица Гессе* функции **w** с переменными **x,y** и параметром **a**.

Пр.7 >> **int**(x^2) $\Rightarrow$ ans =1/3*x^3 – *первообразная* функции x^2.

Пр.8 >> **int**(x+1, 1,3) $\Rightarrow$ ans = 6 – *определенный интеграл* от функции x+1 на отрезке [1,3].

Пр.9 >> syms v u $\Rightarrow$ **int**(**int**(u^2-v^2, 0, u^2), 0, 2) =32/105 – *двойной интеграл* от функции $u^2 - v^2$ на области $G = \left\{(u,v) \in \tilde{\mathrm{R}}^2 :\ 0 \le u \le 2,\ 0 \le v \le u^2\right\}$ выбор порядка переменных интегрирования – *по степени близости к букве* **x**).

Пр.10 >> x=3; z=2; **subs**(u) $\Rightarrow$ ans = 11 – подстановка численных значений в символьное выражение (*substitution* –подстановка).

Пр.11 >> **double**(sqrt(4)) $\Rightarrow$ ans = 2 – преобразование символьного объекта в численный. **clear a** – отмена команды syms для символьного объекта **a**.

Нахождение нулей функции и решение систем нелинейных уравнений.

>> **fzero**(' f(x)', x0) – решение уравнения f(x)=0 с начальным приближением x0 (*zero* – нуль). >> **fzero**(' f(x)', [a b]) – решение уравнения f(x)=0 на отрезке [a b] при условии f(a)*f(b)<0.

Пр.1 >> fzero('x^2-3*x+2', 3) $\Rightarrow$ ans = 2; >> fzero('x^2-3*x+2', 0) $\Rightarrow$ ans =1; >> fzero('x^2-3*x+2', 1.5) $\Rightarrow$ ans = 1.

>> fzero('x^2-3*x+2', [0 3]) $\Rightarrow$ ans = Error, так как f(0)*f(3)=2>0.

Пр.2 >> fzero('cos(x)', [-2 5]) $\Rightarrow$ ans = 4.7124, хотя $\cos x$ имеет на [-2 5] три нуля.

>> **[x,y]=solve**('f(x,y)=0', 'g(x,y)=0') – решение системы уравнений (*solve* – решать).

Пр.1 >> [x]=solve ('cos(x)=0') $\Rightarrow$ ans = 1/2*pi.

Пр.2 >> [x]=solve('x^2/3+48/x^2-10*(x/3-4/x)=0') $\Rightarrow$ x= -2 6 3+21^(1/2) 3-21^(1/2).

Пр.3 >> [x,y]=solve('x^4+y^4-x^2*y^2=13','x^2-y^2+2*x*y=1') $\Rightarrow$ x = 1, -1, 2*i, 2*i; y = 2, -2, i, -i . И еще четыре громоздких ответа.

Пр.4 >> [x]=solve('sin(x)+2=exp(x)') $\Rightarrow$ x =1.0541 – трансцендентное решение.

Аналитическое решение дифференциальных уравнений производится командой **dsolve**.

Пр.1 >> dsolve('x*Dy+1=y^2','x') $\Rightarrow$ ans = -tanh(log(x)+C1) – общее решение уравнения Бернулли $x\,y'_x + 1 = y^2$.

Пр.2 >> [x y z]=dsolve('Dx=2*x+z', 'Dy=3*y', 'Dz=-x+4*z', 'x(0)=1', 'y(0)=-1', 'z(0)=0') $\Rightarrow$ x= exp(3*t)*(1-t) y =-exp(3*t) z =- exp(3*t)*t –

решение задачи Коши *нормальной однородной системы линейных дифференциальных уравнений:*

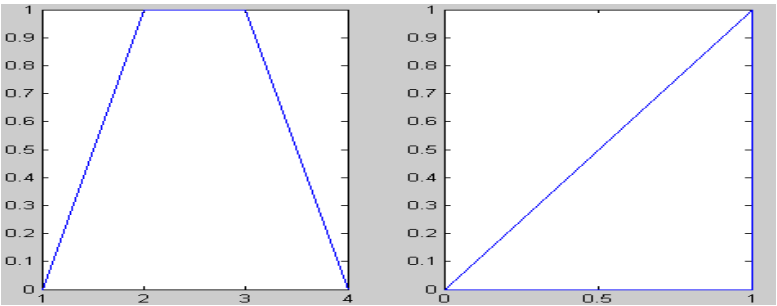
$$\begin{cases} x_t' = 2x + z \\ y_t' = 3y \\ z_t' = -x + 4z \end{cases}, \quad \begin{cases} x(0) = 1 \\ y(0) = -1. \\ z(0) = 0 \end{cases}$$

Другой способ решения задачи Коши для неоднородной системы с помощью *формулы Коши*: >> syms t x >> X0=[1;-1;0]; B=[0;t;0]; A=[2 0 1;0 3 0;-1 0 4]; >> X=expm(t*A)*X0+int((expm(t-x)*A)*B, x, 0, t) $\Rightarrow$ X = exp(3*t)-exp(3*t)*t -exp(3*t)+3*exp(t)*t-3*t -exp(3*t)*t.

§ 1.5. Графики

>> $x = [x_1\ x_2\ \dots\ x_n]$**;** **plot**(x) – ломаная с точками излома $\{(1,x_1),\dots(n,x_n)\}$ (*plot* – график, кривая). >> $X = [x_{11}\ x_{12}\dots x_{1n};\dots;x_{m1}\ x_{m2}\dots x_{mn}]$; **plot**(X) – n штук ломаных $\{(1,x_{1k}),\dots(m,x_{mk})\}$, $k = 1,\dots,n$. >> $y = [y_1\ y_2\ \dots\ y_n]$**;** **plot**(x,y) – ломаная с точками излома $\{(x_1,y_1),\dots(x_n,y_n)\}$.

Пр.1 >> x=[0 1 1 0]; plot(x) – ломаная с точками излома $\{(1,0),(2,1),(3,1),(4,0)\}$. ***Пр.2*** >> y=[0 0 1 0]; plot(x,y) – треугольник с вершинами $(0,0),(1,0),(0,1)$.

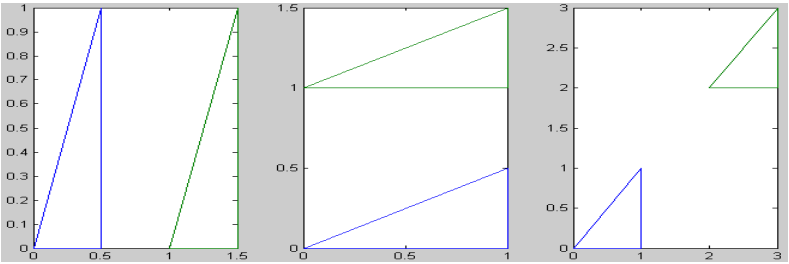


>> **plot**(X,y) – **m** штук «горизонтально» расположенных ломаных $\{(x_{k1},y_1),\dots(x_{kn},y_n)\}$, $k = 1,\dots,m$. >> $Y = [y_{11}\ y_{12}\dots y_{1n};\dots;y_{m1}\ y_{m2}\dots y_{mn}]$;

plot(x,Y) – **m** штук «вертикально» расположенных ломаных $\{(x_1, y_{k1}),...(x_n, y_{kn},)\}$, $k = 1,...,m$.

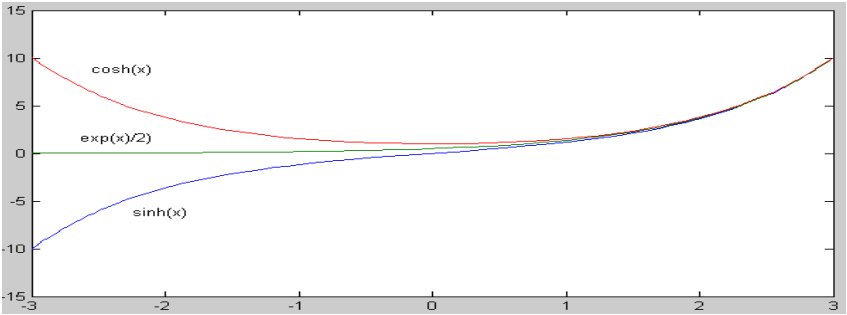
>> **plot**(X,Y) – **n** штук ломаных $\{(x_{1k}, y_{1k}),...(x_{mk}, y_{mk})\}$, $k = 1,...,n$.

Пр.1 >> X=[0 0.5 0.5 0;1 1.5 1.5 1]; **plot**(X,y) – два рядом расположенных треугольника. ***Пр.2*** >> Y=[0 0 0.5 0;1 1 1.5 1]; **plot**(x,Y) – два друг над другом расположенных треугольника. ***Пр.3*** >> X=[0 2;1 3;1 3;0 2]; Y=[0 2;0 2;1 3;0 2]; **plot**(X,Y) – два треугольника на биссектрисе.



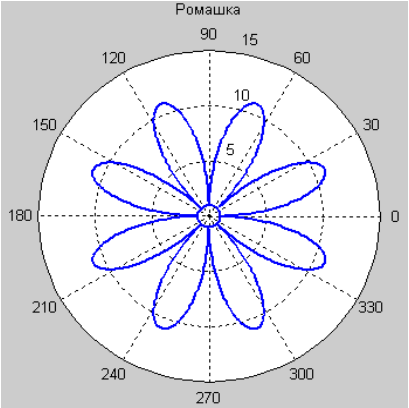
>> **fplot**(’f(x)’,[a,b]), **ezplot**(’f(x)’, [a,b]) – график функции f(x) на отрезке [a,b]. >> **fplot**(’[f(x),g(x)]’,[a,b]) – графики функций f(x), g(x) на [a,b].
Пр. >> **fplot**('[sinh(x),exp(x)/2,cosh(x)]',[-3,3]) – графики трех асимптотически сближающихся функций $sh(x)$, $\frac{1}{2}e^x$, $ch(x)$.

>> **gtext**('sinh(x)'); **gtext**('exp(x)/2)'); **gtext**('cosh(x)') – вставка надписи имени функции в произвольную точку графика при помощи мыши.



>> fi=[a:h:b]; **polar**(fi,r) – график функции r(fi) в *полярных координатах.*

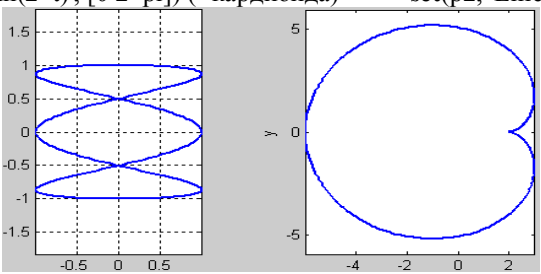
Пр. >> fi=0:0.01:2*pi;
p= **polar**(fi,10***abs**(sin(4*fi))+1);
>> **hold on** >> q=**polar**(fi,0*fi+1);
>> r=[p q]; **set**(p,'LineWidth',1.5)
>> title('Ромашка').
Здесь ***abs***(.) – операция взятия абсолютной величины;
hold on – команда, которая фиксирует текущий график;
set – команда установки ширины линии ***LineWidth***=1.5;
title('·') - заголовок графика.



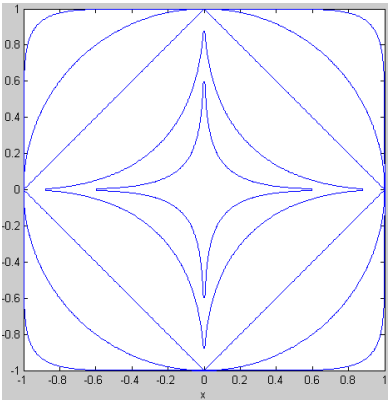
>> **ezplot**(’f(t)’,’g(t)’) или **ezplot**(’f(t)’,’g(t)’,[a,b]) – график параметрически заданной функции x=f(t), y=g(t) (во второй команде – при $t \in$ [a,b]).

Пр.1 >> **syms** t >> **subplot**(1,2,1) >> x=sin(3*t); y=cos(t); p1=**ezplot**(x,y), **grid** (фигура Лиссажу; ***grid*** – команда изображения координатной сетки);
>> **set**(p1,' LineWidth ',1.5)

Пр.2 >> **subplot**(1,2,2) >> p2=**ezplot**('4*cos(t)-2*cos(2*t)', '4*sin(t)-2*sin(2*t)', [0 2*pi]) (- кардиоида) >> **set**(p2,' LineWidth ',1.5)

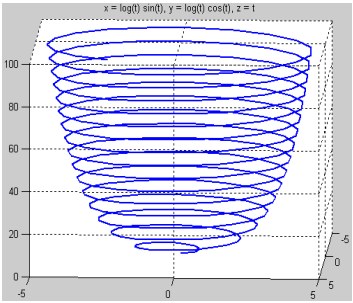


>> **ezplot**(’f(x,y)’), **ezplot**(’f(x,y)’,[a,b,c,d])– график **неявно** заданной функции f(x,y)=0 (во второй команде при $x \in [a,b],\ y \in [c,d]$).



Пр.3
>>**ezplot**('(x^2)^(1/6)+(y^2)^(1/6)=1', [-1,1]); >> **hold on**
>> **ezplot**('(x^2)^(1/4)+(y^2)^(1/4)=1', [-1,1])
>> **ezplot**('(x^2)^(1/2)+(y^2)^(1/2)=1', [-1,1])
>> **ezplot**('(x^2+(y^2)=1',[-1,1])
>> **ezplot**('(x^2)^4+(y^2)^4=1',[-1,1])
Команда $ezplot(\cdot)$ не работает в цикле, и семейство циклоид приходится строить последовательно.

Если $X=[x_{11}\ldots x_{1n};\ldots;x_{m1}\ldots x_{mn}]$, $Y=[y_{11}\ldots y_{1n};\ldots;y_{m1}\ldots y_{mn}]$, $Z=[z_{11}\ldots z_{1n};\ldots;z_{m1}\ldots z_{mn}]$, то команда >> **plot3**(X,Y,Z) изображает **n**



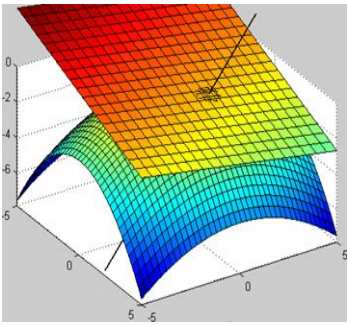
штук пространственных ломаных $\{(x_{1k},y_{1k},z_{1k}),\ldots(x_{mk},y_{mk},z_{mk})\}$, $k=1,..,n$.
Команда >> **ezplot3**('x(t)','y(t)','z(t)','t',[a,b]) изображает график пространственной кривой для $t\in[a,b]$.
Пр. p=**ezplot3**('log(t)*sin(t)', 'log(t)*cos(t)','t',[1,100])
>> **set**(p,'LineWidth',1.5).

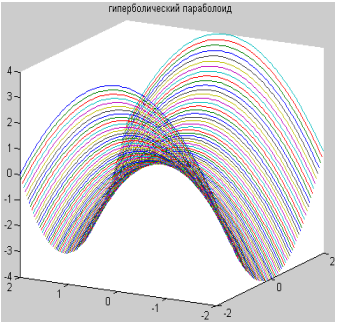
При построении поверхности z=f(x,y) область определения задается командой [X,Y]= **meshgrid**(x,y). Она представляет собой прямоугольную координатную сеть точек (*meshgrid*) с координатами из регулярных одномерных массивов $x=[a:h_1:b]$, $y=[c:h_2:d]$. Третья координата задается командой **Z=f(X.,Y.)**. Команда **surf**(X,Y,Z) представляет поверхность в виде вписанного цветного многогранника. Команда **plot3**(X,Y,Z) представляет эту же поверхность в виде семейства разноцветных параллельных кривых. Команда **ezsurf**(z(x,y), [a,b,c,d],n) стро-

ит график символьной функции в виде покрывающей сети густотой **n** над прямоугольной областью $a \le x \le b$, $c \le y \le d$.

Пр.1 Изобразим поверхность с уравнением $z = -0.2x^2 - 0.1y^2$ в точке $(x_0, y_0) = (1,1)$, касательную плоскость в соответствующей точке и нормаль к поверхности.

>> syms x y t >> z=-x^2-y^2/2; >> **ezsurf**(z,[-10,10],60) >> grid hold on
% строим касательную плоскость.
>> x0=5;y0=-1; z0=subs(z,{x,y},{x0,y0}); dzdx=diff(z,x);dzdy=diff(z,y);
dzdx0=subs(dzdx,{x,y},{x0,y0}); dzdy0=subs(dzdy,{x,y},{x0,y0});
f=z0+(x-x0)*dzdx0+(y-y0)*dzdy0 **;** **ezsurf**(f,[-10,10],40]
% строим нормаль к поверхности.
>> t=-10:0.1:10; dzdx0=double(dzdx0); dzdy0=double(dzdy0);
xt=x0+dzdx0*t;yt=y0+dzdy0*t; zt=zo-t; **plot3**(xt,yt,zt,'k','LineWidth',1.5).

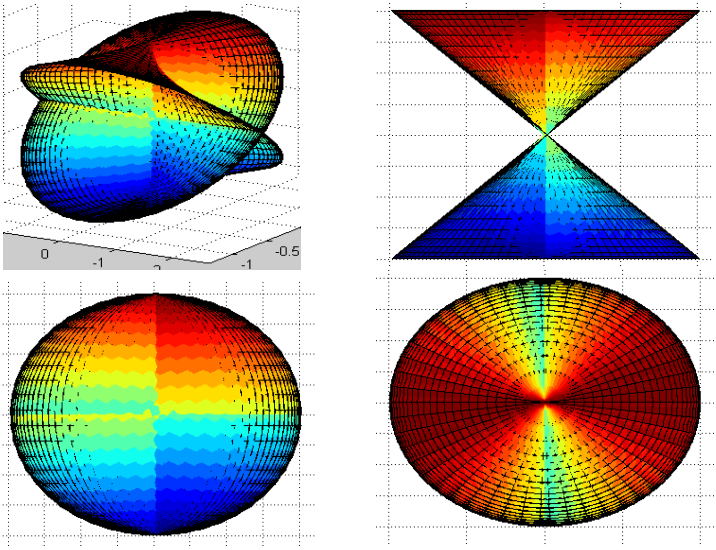




Пр.2 Последовательность команд >> x=[-2:0.05:2]; y=[-2:0.05:2]; [X,Y]=**meshgrid**(x,y) ; Z=X.^2-Y.^2 ; **plot3**(X,Y,Z) дает фрагмент гиперболического параболоида. >>**title**(‘гиперболический параболоид’).

Команды **ezsurf, ezmesh** выполняют построение поверхности без предварительного вычисления элементов массива аргументов.

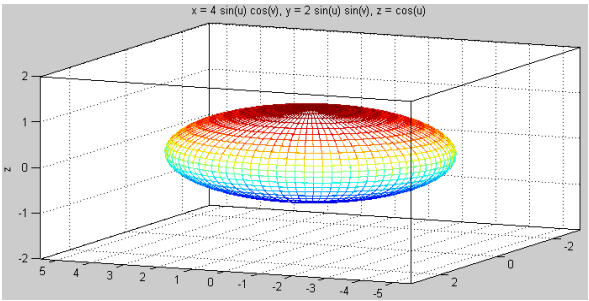
Пр.3 >> **ezsurf**('sin(u)*cos(v)','2*sin(u)*sin(v)','2*cos(v)') – параметрически заданная поверхность и ее проекции соответственно на координатные плоскости XOZ, YOZ, XOY.



Дополнительная команда **axis**([a b c d e f]) изображает фрагмент пространства $G = \{(x, y, z) : a \le x \le b,\ c \le y \le d,\ e \le z \le f\}$ с графиком в нем.

>>**xlabel**(‘txt’), >>**ylabel**(‘txt’) – тексты txt для осей (*label* – метка, знак). Команда >> **box on** (**off**) наносит (удаляет) рамку вокруг текущего изображения. >> **box** переключает режим нанесения-удаления.

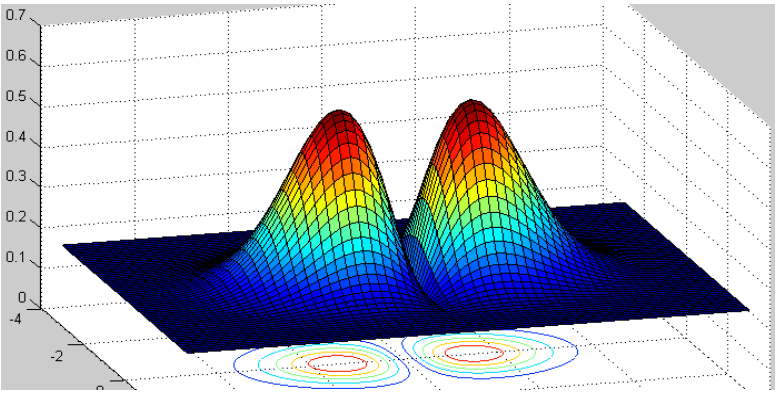
Пр.4 >> **ezmesh**('2*sin(u)*cos(v)','2*sin(u)*sin(v)','cos(v)') – параметрически заданная поверхность эллипсоида. >> **axis**([-3 3 -3 3 -2 2]) >> **box on**



Команда **surfc**(X,Y,Z) дает дополнительно изображение линий уровня.

Последовательность команд $x=[a:h_1:b]; y=[c:h_2:d]$; [X,Y]=**meshgrid**(x,y); Z=f(X.,Y.); **contour**(x,y,Z) изображает линии уровня функции z=f(x,y) в плоскости XOY. При этом команда **contour**(x,y,Z,n) изображает **n** линий уровня; команда **contour**(x,y,Z,**v**) изображает линии уровня f(x,y)=v со значения из заданного массива **v**; команда **contour**(x,y,Z,[v v]) изображает одну линию уровня f(x,y)=v.

Пр. >> x=-3:0.1:3; y=-3:0.1:3; [X,Y]=**meshgrid**(x,y);
Z=abs(X).*exp(-X.^2 – Y.^2)+0.2; **surf**(X,Y,Z)
>> hold on >> **contour**(x,y,Z,5)



§ 1.6. Численные методы

Нахождение точек экстремума функции. Для поиска точек минимума функции одной переменной используется команда **fminbnd** (‘f(x)’, a, b), где **a, b** – концы отрезка, на котором ищется минимум. Для более развернутого вывода, содержащего, значение функции в точке минимума, название численного метода и число итераций, используют четыре возвращаемых параметра.

Пр.1 >> [x, val, flag, output]=**fminbnd**('x^3-5*x^2+7*x-10', 0, 5) $\Rightarrow$ x = 2.3333 val = -8.1852 flag(- признак) = 1 output = iterations: 10 funcCount: 11 algorithm: 'golden section search, parabolic interpolation'.

Для поиска точек минимума функции нескольких переменных используется команда **fminsearch** (‘f(x,y)’, [a b]), где [a b] – начальное приближение.

Пр.2 >>**fminsearch**('x(1)^4-5*x(1)^2*x(2)+7*x(1)+4*x(2)^4-2*x(2)-10', [0 5])
$\Rightarrow$ ans = -1.9019 1.0788

Аппроксимация и интерполяция массивов. Зададим два числовых массива

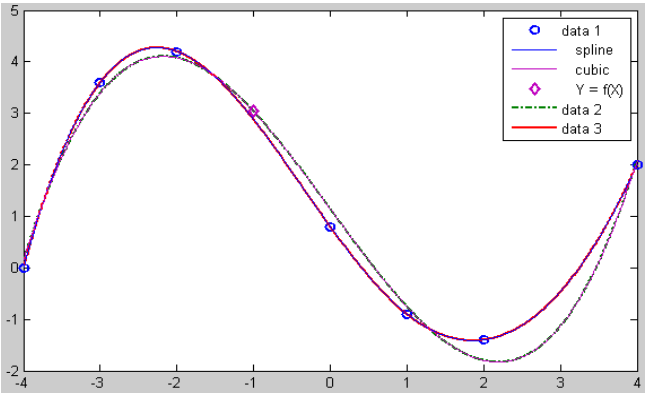
>> x=[-4 -3 -2 0 1 2 4]; y=[0 3.6 4.2 0.8 -0.9 -1.4 2];

k=**polyfit**(x,y,3) $\Rightarrow$ k= 0.1806 0.1406 -2.1171 0.8442 – коэффициенты полинома степени n=3, аппроксимирующего таблицу данных x,y по *методу наименьших квадратов* (МНК).

>> t=-4:0.01:4; r=**polyval**(k,t); – массив значений аппроксимирующего полинома на последовательности **t**.

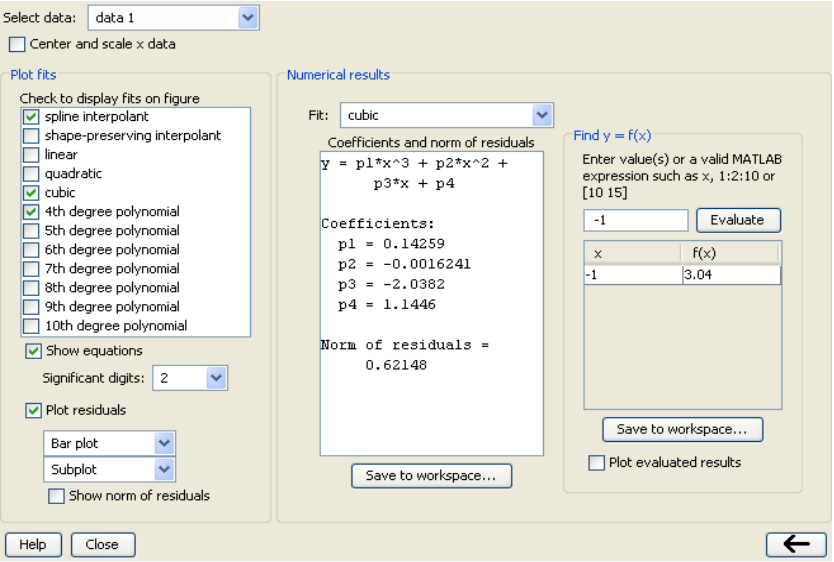
>> s=**spline**(x,y,t); – массив значений, построенного по данным **x**,**y** естественного *кубического сплайна*, вычисленных на последовательности **t**.

>> **plot**(x,y,'o', t,r,'-.', t,s,'-') – графики исходного массива **x**,**y**, (x,y,'o' отмечены кружочками), аппроксимирующего полинома (t,r,'-.' – отмечен штрихпунктирной линией) и интерполирующего сплайна (t,s,'-' – отмечен сплошной линией).



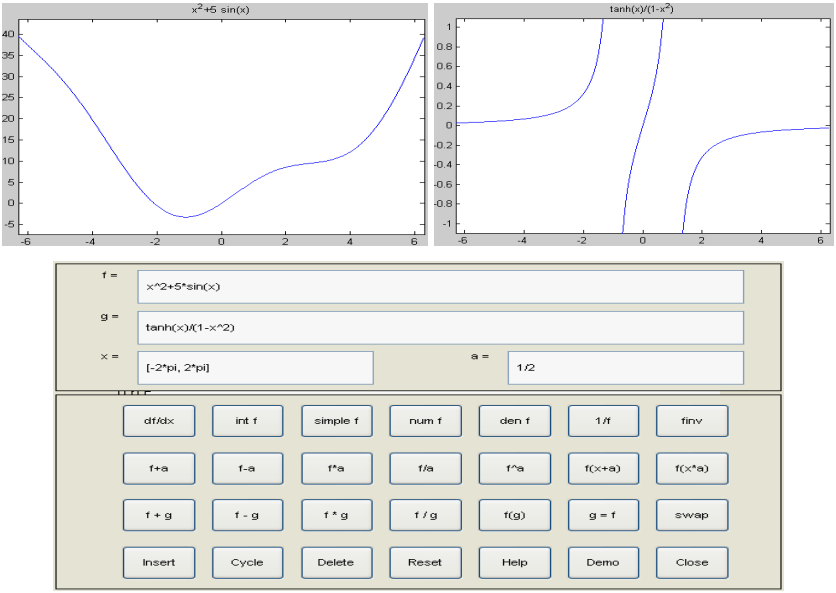
Графическое окно обработки данных **Basic Fitting** (*fitting* – подгонка, сглаживание) позволяет быстро и наглядно производить аппроксимацию и интерполяцию с помощью указанных выше и иных процедур; определять погрешности аппроксимации; численно исследо-

вать результаты аппроксимации. В него входят из графического окна через меню **Tools**.



Окно разворачивается на три панели: 1) на **Plot fits** предлагается меню способов аппроксимации (в окне выбрана сплайн-интерполяция и МНК многочленами третьей и четвертой степени); установка флажка **Plot residuals** (диаграмма разностей) создает отдельное окно с диаграммой разностей между значениями аппроксимирующей кривой и табличными значениями в узлах; установка флажка **Show equations** выводит на экран уравнения выделенных кривых; 2) на панель **Numerical results** выводится уравнение выделенной аппроксимирующей кривой и **Norm of residuals** (корень квадратный из суммы квадратов соответствующих разностей); 3) на панели **Find** $y = f(x)$ можно вычислять значения аппроксимирующей функции в задаваемой точке или на массиве.

Функциональный калькулятор вызывается командой **funtool**. При этом появляются два графических окна и одно окно набора данных. Он позволяет производить различные символьные вычисления с парой аналитически задаваемых функций. Результаты представляются в виде графиков.



Калькулятор степенных вычислений производит аппроксимацию функций с помощью многочленов Тейлора. Он вызывается командой **Taylortool**.

Формула Тейлора для $n+1$ раз непрерывно дифференцируемой функции $f(x)$ в окрестности точки x_0 имеет вид:

$$f(x) := f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \ldots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1},$$

где $R_n(x) = \dfrac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1}$ есть остаточный член *в форме Лагранжа* и число ξ находится между x_0 и x. Для $h > 0$ положим $M_k(h) := \max\limits_{|x - x_0| \le h} |f^{(k)}(x)|$. Тогда $\forall\, x \in [x_0 - h, x_0 + h]$

$$|f(x) - p_n(x)| = |R_n(x)| \le \frac{1}{(n+1)!} M_{n+1}(h) \cdot h^{n+1} =: \omega(n, h).$$

Формула Тейлора позволяет решать следующие три задачи:

1) по заданным степени n многочлена и длине отрезка $2h$ оценить в терминах $\omega(n,h)$ величину отклонения $p_n(x)$ от $f(x)$ на $[x_0-h, x_0+h]$;

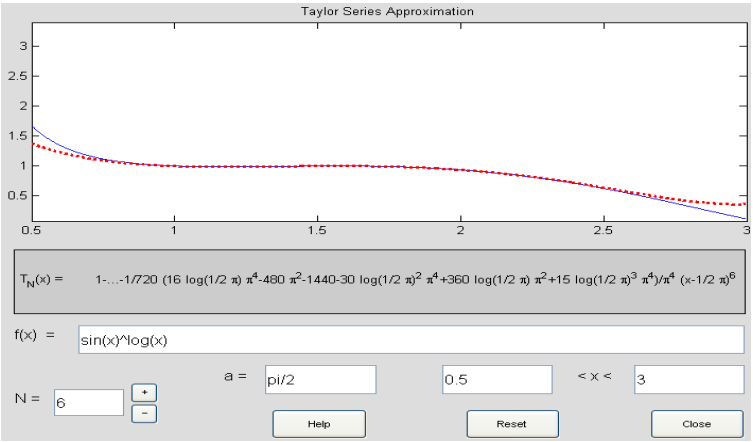
2) при длине отрезка $2h$ и точности ω определить степень n многочлена $p_n(x)$, отклонение которого от $f(x)$ на $[x_0-h, x_0+h]$ не превышает ω;

3) по заданным степени многочлена n и точности ω определить максимальную длину $2h$ отрезка $[x_0-h, x_0+h]$, на котором $p_n(x)$ отклоняется от $f(x)$ не более, чем на ω.

Пр. Зададимся функцией $f(x)=(\sin x)^{\ln x}$, точностью $\omega=0{,}1$, и многочленом Тейлора степени $n=6$. Подбирая грубые пределы изменения **x**, по графикам функции и многочлена видим, что требуемая точность достигается на отрезке $[0.7, 2.7]$.

Многочлен Тейлора функции вычисляется с помощью команды **taylor**.

Пр. >>**syms** x >> **taylor**(sqrt(x+2), 3, 4)$\Rightarrow$ans=6^(1/2)+1/12*6^(1/2)*(x-4)--1/288*6^(1/2)* (x-4)^2 – многочлен Тейлора степени **3**-1=2 для функции $\sqrt{x+2}$ в окрестности точки x=**4**.



Численное интегрирование. Функции численного интегрирования: **trapz** – метод трапеций; **quad** – метод Симпсона; **quad8** (*quadratura*) – метод Ньютона-Котеса; **dblquad** – вычисление двойного интеграла.

Пр.1 >> f=**inline**('sin(x)./x','x'); [I,cnt]=**quad**(f,0,pi/2,0.001) $\Rightarrow$ I = 1.3708

cnt =14 – вычисление интеграла $\int\limits_0^{\pi/2} \frac{\sin x}{x} dx$ с точностью 0,001; cnt =14 – число точек, в которых пришлось вычислять подынтегральную функцию.

Пр.2 >> x=0:0.1:pi/2; y=sin(x)./x; **trapz**(x,y) $\Rightarrow$ ans = NaN.

Однако >> x=0.001:0.01:pi/2; y=sin(x)./x; **trapz**(x,y) $\Rightarrow$ ans = 1.3635.

Команда **dblquad**(fun,x1,x2,y1,y2) (*double integral*) возвращает двойной интеграл по прямоугольной области.

Пр.3>> f=**Inline**('x.^2*log(y)-cos(x+y)','x','y');

dblquad(f,-1,1,2,3) $\Rightarrow$ ans = 1.8992.

Эта же команда позволяет вычислять двойной интеграл и по криволинейной области.

Пр.4 >> **dblquad**(@(x,y) sqrt(max(4-(x.^2+y.^2), 0)), -2, 2, -2, 2) $\Rightarrow$ ans = 16.7552.

Численное решение дифференциальных уравнений. При выборе численного метода решения *нормальной системы дифференциальных уравнений* (НСОДУ) $\begin{cases} x_1' = f_1(t, x_1(t), ..., x_n(t)) \\ \quad . \quad . \quad . \\ x_n' = f_n(t, x_1(t), ..., x_n(t)) \end{cases}$ (или в матричной форме $X_t' = F(t, x)$) важно определиться с жесткостью системы и выбором явного или неявного метода. Если вычислить собственные числа матрицы Якоби $\mathrm{Re}\lambda_1 \le ... \le \mathrm{Re}\lambda_N < 0$, а решение искать на отрезке $[0, T]$, то при больших значения $\frac{\mathrm{Re}\lambda_1}{\mathrm{Re}\lambda_N} T$ система считается *жесткой* (величина жесткости в случае нестационарных систем, вообще говоря, является переменной). В явных численных методах оценка сверху на величину шага интегрирования более ограничительная, но формула вычисления сеточной функции более простая.

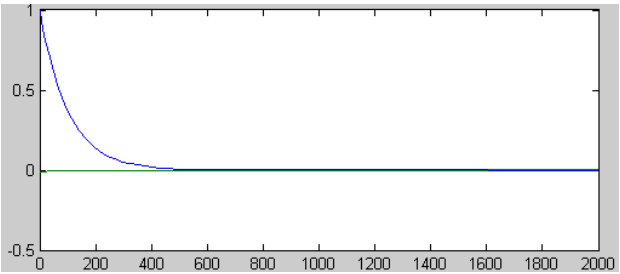
В MATLAB для решения жестких НСОДУ используются решатели (*solver* – решающее устройство) ode15s, ode23s, ode23t, ode23tb, а для решения нежестких – ode45, ode23, ode113.

Пр. Решим задачу Коши для НСОДУ $\begin{cases} x_1' = x_2 \\ x_2' = -x_1 - 100.01x_2 \end{cases}$, $\begin{cases} x_1(0) = 1 \\ x_2(0) = 0 \end{cases}$, на отрезке [0 2000]. Так как собственные числа равны $\lambda_1 = -100$, $\lambda_2 = -0.01$, то $\frac{\lambda_1}{\lambda_2} T = 2 \cdot 10^7$, то есть система жесткая. Сначала создается файл-функция

```
function  xdot=func(t,x)
xdot=[x(2); -x(1)-1000.001*x(2)];
```

Результат выполнения команд

>> x0=[1 0]; [t,x]=**ode15s**('func', [0 2000], x0); plot(t,x) виден на рисунке.



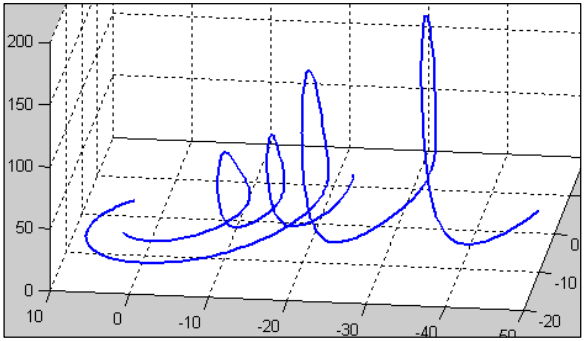
Для этой жесткой системы функция **ode15s** дает низкую скорость интегрирования, в чем нетрудно убедиться.

Приведем схему рисование нескольких траекторий, например, системы уравнений Росслера. Сначала создаем файл-функцию с записью правых частей системы и запоминаем ее:

```
>> function dx = ode(t,x)
dx=zeros(3,1);
dx(1)  = -x(2)-x(3);
dx(2)  = x(1)+0.5*x(2);
dx(3)  = 0.5+x(3)*(x(1)-3).
```

Затем в командном окне набираем программу рисования двух траекторий и запускаем ее на выполнение клавишей enter.

```
>> x0=4;
for k=1:2
x0=(-1)^k*dx0;
[t,x]=ode45('ode',[0 7],[x0 4 0]);
plot3(x(:,1),x(:,2),x(:,3),'LineWidth',1.5)
hold on
end
>> grid
```



§ 1.7. Обработка экспериментальных данных

Случайная выборка нормального распределения формируется командой **A=m+s*randn(n,1)**, где **m** – *среднее значение*; **s** – *стандартное отклонение*; **n** – *объем выборки*. С выборкой **A** можно осуществлять следующие операции: **min(A)** – найти минимальное значение, **max(A)** – найти максимум, **mean(A)** – среднее, **median(A)** – медиана, **prod(A)** – произведение, **sum(A)** – сумма элементов, **cumsum(A)** – *кумулятивная сумма*, **std(A)** – стандартное отклонение, **sort(A)** – упорядочивание по возрастанию; **-sort(A)** – упорядочивание по убыванию; **ecdf**(A) – график *эмпирической функции распределения*.

Пр.1 >> A=5+1*randn(15,1) ⇒ A = 4.5674 3.3344 5.1253 5.2877 3.8535 6.1909 6.1892 4.9624 5.3273 5.1746 4.8133 5.7258 4.4117 7.1832 4.8636

>> cumsum(A) ans = 4.5674 7.9019 13.0272 18.3149 22.1684 28.3593 34.5485 39.5108 44.8381 50.0128 54.8261 60.5518 64.9635 72.1467 77.0103.

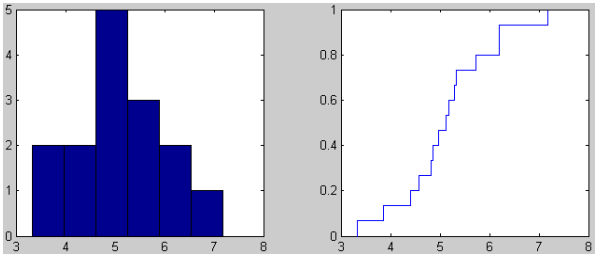
Гистограмма выборки строится по команде **hist(A)**. При этом длина отрезка равна размаху выборки и по умолчанию делится на 10 равных частей. Гистограмма с *m* отрезками строится по команде **hist(A,m)**. Массивы высот прямоугольников гистограммы и абсцисс середин прямоугольников вычисляются по команде **[y,x]=hist(A,m)**.

Пр.2 [y,x]=**hist**(A,6) $\Rightarrow$ y = 2 2 5 3 2 1

x = 3.6551 4.2966 4.9381 5.5795 6.2210 6.8625

>> **hist**(A,6) – выводит изображение гистограммы.

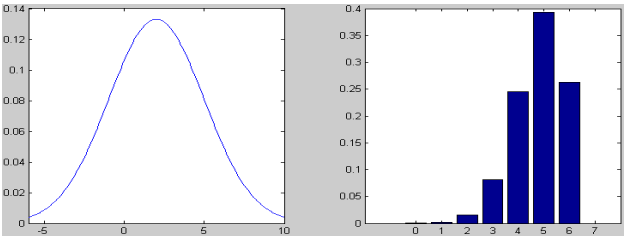
>> **ecdf**(A) – выводит график эмпирической функции распределения.



MATLAB позволяет вычислять плотности распределения и функции распределения различных случайных величин. Так команда **normpdf**(x, a, σ) (*normal probability density function*) вычисляет значения плотности *нормального распределения* с параметрами a, σ на массиве **x**. Команда **binopdf**(x,N,P) (*binomial probability density function*) вычисляет значения *биномиального распределения* с параметрами N,P на массиве **x**.

Пр.3 >> subplot(1,2,1); x=-6:0.1:10; plot(x, **normpdf**(x,2,3)) – плотность с матожиданием =2 и стандартным отклонением =3

>> subplot(1,2,2); x=0:1:7; **bar**(x, **binopdf**(0:7, 6, 0.8)) (*bar* – прямоугольник; команда построения столбцового графика) – график биномиального закона распределения с параметрами N=6, P=0.8.



Доверительные интервалы матожидания и дисперсии нормально распределенной генеральной совокупности вычисляются по формулам

$$\left(Mx - Sx \cdot t_{\frac{1+\gamma}{2},n-1} / \sqrt{n}, Mx + Sx \cdot t_{\frac{1+\gamma}{2},n-1} / \sqrt{n}\right), \quad \left(Dx \cdot n / \chi^2_{\frac{1+\gamma}{2}}(n-1), Dx \cdot n / \chi^2_{\frac{1-\gamma}{2}}(n-1)\right)$$

Найдем их для случайной выборки и трех доверительных вероятностей γ .

Пр.4 >>x=**randn**(1,200); – *случайная выборка* объема 200 >> n=**lenth**(x); Mx=mean(x)$\Rightarrow$Mx =-0.0440 – *выборочное среднее*. >> Sx=**std**(x) $\Rightarrow$ Sx = 0.9757 – *стандартное отклонение*. >> Dx=**Sx^2** $\Rightarrow$ Dx = 0.9519 – *несмещенная выборочная дисперсия*.

>> h=[0.9;0.95;0.99]; – *доверительные вероятности*.

>> t=**tinv**((1+h)/2,n-1) $\Rightarrow$t = 1.6525; 1.9720; 2.6008; – *квантиль **t**-распределения* с *n-1* степенями свободы.

>> chi2l=**chi2inv**((1+h)/2,n-1) $\Rightarrow$ chi2l = 232.9118; 239.9597; 254.1352

>> chi2r=**chi2inv**((1-h)/2,n-1) $\Rightarrow$ chi2r =167.3610; 161.8262; 151.3699; – *квантили* χ^2 – *распределения* с *n-1* степенями свободы.

Остается вычислить доверительные интервалы матожидания и дисперсии.

Mxd=[h,Mx–Sx*t/sqrt(n), Mx+Sx*t/sqrt(n)] $\Rightarrow$ Mxd = $\begin{matrix} 0.90 & -0.1580 & 0.0700 \\ 0.95 & -0.1801 & 0.0920 \\ 0.99 & -0.2234 & 0.1354 \end{matrix}$

Dxd = [h, n*Dx./chi2l, n*Dx./chi2r] $\Rightarrow$ Dxd = $\begin{matrix} 0.90 & 0.8174 & 1.1376 \\ 0.95 & 0.7934 & 1.1765 \\ 0.99 & 0.7492 & 1.2578 \end{matrix}$.

Первые столбцы – доверительная вероятность, второй и третий – концы интервалов.

Проверим гипотезу о распределении генеральной совокупности с помощью критерия согласия χ^2 и уровнями значимости $\alpha = 0.1,\ 0.01$. Произведем произвольную выборку объема n=100 и вычислим ее среднее значение и стандартное отклонение.

>> Mx=mean(A) $\Rightarrow$ Mx = 5.2799 >> sigma=std(A) $\Rightarrow$ sigma =2.9702.

Образуем по выборке статистический ряд из m=8 классов и вычислим частоты и середины интервалов:

>> A=10*rand(100,1); [ni,x]=hist(A,8) $\Rightarrow$ ni =12 11 12 10 10 12 17 16; x = 0.7182 1.9166 3.1149 4.3133 5.5117 6.7100 7.9084 9.1067.

Вычислим массив концов этих интервалов:
>> k=1:1:9; xi=x(1)+(k-3/2)*(x(8)-x(1))/7 $\Rightarrow$ xi = 0.1190 1.3174 2.5157 3.7141 4.9125 6.1108 7.3092 8.5076 9.7059.
Проверим гипотезу о нормальном распределении, для чего посчитаем массив теоретических частот с помощью функции нормального распределения normcdf.
>> k=1:1:8; p=normcdf(xi(k+1),Mx,sigma) – normcdf(xi(k),Mx,sigma) $\Rightarrow$ p = 0.0499 0.0849 0.1230 0.1517 0.1594 0.1426 0.1086 0.0705.

Вычислим *наблюдаемое значение* по формуле $\chi^2_{набл} = \frac{1}{n}\sum_{k=1}^{m}\frac{n_k^2}{p_k} - n$.

>> hi=sum((ni.^2)./p)/100-100 $\Rightarrow$ hi = 40.6627.

С помощью функции chi2inv распределения χ^2 для числа степеней свободы $m - l - 1 = 5$, где $l = 2$ - число параметров нормального распределения, вычислим *теоретическое значение* при двух уровнях значимости.
>> alpha=0.1; chi2inv(1-alpha,5) $\Rightarrow$ ans = 9.2364.
>> alpha=0.01; chi2inv(1-alpha,5) $\Rightarrow$ ans = 15.0863.
Так как в обоих случаях оно меньше наблюдаемого, то гипотезу отвергаем.

Проверим гипотезу о равномерном распределении. Для этого с помощью метода моментов найдем его параметры a, b.

>>a=Mx-sqrt(3)*sigma $\Rightarrow$ a= 0.1355 >>b=Mx+sqrt(3)*sigma $\Rightarrow$ b=10.4244.
>> k=1:1:8; p=unifcdf(xi(k+1),a,b) – unifcdf(xi(k),a,b) $\Rightarrow$
p = 0.1149 0.1165 0.1165 0.1165 0.1165 0.1165 0.1165 0.1165.
>> hi=sum((ni.^2)./p)/100-100 $\Rightarrow$ hi = 11.6163.

Так как для второго уровня значимости теоретическое значение больше наблюдаемого, то гипотезу о равномерном распределении генеральной совокупности принимаем с уровнем значимости $\alpha = 0.01$.

Глава 2. SIMULINK

«С момента появления пакета Simulink, универсальные, не ориентированные на конкретные прикладные области, пакеты для моделирования и исследования динамических систем в широком понимании этого термина, включая дискретные, непрерывные и гибридные модели, стали повседневной реальностью.»
CNews 17.07.03

§ 2.1. Элементы теории динамических систем

Определение. ***Динамической системой*** (***ДС***) называется восьмерка объектов $\Sigma := \{T, Y, U, \Omega, X, \Gamma, \delta, \lambda\}$, где T – ***область определения*** рассматриваемых в системе функций (упорядоченное множество, часто – множество моментов времени); U – множество значений ***функций входа системы*** из $\Omega = \{u : T \to U\}$; Y - множество значений ***функций выхода системы*** из $\Gamma := \{y : T \to Y\}$; X - *множество состояний системы*; $\delta : T \times X \to Y$ - ***функция перехода*** (***переходная функция***) ***состояний***, которая определяет по заданной функции входа $u \in U$ и *начальному состоянию системы* $x = x(\tau) \in X$ состояние для $\forall t > \tau,\ t \in T$ по правилу $x(t) := \delta(t; \tau, x, u)$; $\lambda : T \times X \to Y$ – ***выходное отображение системы***, определяющее ***функцию выхода*** $y(t) := \lambda\left(t, \delta(t; \tau, x(t), u(\cdot))\right)$.

Замечание. Это формальное определение. Менее формально ***динамическая система*** – это математический объект, соответствующий реальным физическим, химическим, биологическим и другим системам, эволюция которых во времени на бесконечном интервале времени однозначно определяется начальным состоянием. Динамические системы – образы соответствующих реальных систем. Они также изменяются во времени.

Определение. Сужения функции входа и выхода $u_{(t_1,t_2]} := u\big|_{(t_1,t_2]}$, $y_{(t_1,t_2]} := y\big|_{(t_1,t_2]}$ называются соответственно ***отрезками функций входа и выхода***.

Замечание. Функция перехода должна удовлетворять условиям:
1) (*согласованность*) $\forall u \in U \;\; \forall x \in X \;\; \forall t \in T \;\; \delta(t,t,x,u) = x$;
2) (*полугрупповое свойство*)
$\forall t_1 < t_2 < t \;\; \forall u \in U \;\; \forall x \in X \;\; \delta(t,t_1,x,u) \equiv \delta\left(t,t_2,\delta\left(t_2,t_1,x,u\right),u\right)$;
3) (*причинность*) Если $u',u'' \in \Omega$ и $u'_{(t_1,t]} = u''_{(t_1,t]}$, то $\delta(t;t_1,x,u') = \delta(t;t_1,x,u'')$.

Замечание. Пару (t,x), где $t \in T$, $x \in X$ называют также ***событием*** (***фазой***) ***системы*** Σ, а пару (T,X) – ***пространством событий*** (***фазовым пространством***) этой системы. Функция перехода $x(t) := \delta(t;\tau,x,u)$ называется ***траекторией*** (***движением***, ***орбитой***, ***потоком***, ***решением***). Говорят, что ***управление*** u ***переносит (переводит, изменяет, преобразует) состояние*** x ***системы в состояние*** $x(t)(=\delta(t;\tau,x,u))$.

Определение. Динамическая система Σ называется ***стационарной***, если:
1) T есть аддитивная группа (определена операция сложения);
2) множество допустимых функций входа Ω замкнуто относительно операции сдвига аргумента: если $u(\cdot) \in \Omega$, то $\forall \tau \in T \;\; u(\cdot+\tau) \in \Omega$;
3) $\forall s \in T \;\; \delta(t+s;\tau+s,x,u(\cdot+s)) \equiv \delta(t;\tau,x,u)$;
4) $\forall t \in T \;\; \lambda(t,\cdot) \equiv \lambda(\cdot) : X \to Y$.

Определение. ***ДС*** называется ***системой с непрерывным временем***, если $T = \mathrm{R}$, и ***системой с дискретным временем***, если $T = \mathrm{Z}$.

Определение. ***ДС*** называется ***конечной***, если множество состояний конечно: $card(X) < \infty$, и ***конечномерной***, если X является конечномерным векторным пространством: $\dim(X) < \infty$.

Определение. ***ДС*** Σ называется ***линейной***, если:
1) множества X,U,Ω,Y,Γ являются векторными пространствами;
2) отображения $\delta(t;\tau,\cdot,\cdot) : X \times \Omega \to X$, $\eta(t,\cdot) : X \to Y$ являются линейными.

Определение. ***ДС*** Σ называется ***гладкой***, если:
1) $T = \mathrm{R}$;
2) X,Ω топологические пространства;
3) $\forall \tau, x \;\; x(t) = \delta(t;\tau,x,u(\cdot))$ гладкая функция каждый раз, когда $u(\cdot)$ есть функция непрерывная.

Определение ***Конечным автоматом Мили*** называется множество из пяти объектов $\mathrm{A} = (S, A, B, \delta, \lambda)$, в котором:

S – конечное непустое множество (***пространство***) ***состояний***;

A – конечное непустое множество ***входных сигналов*** (***входной алфавит***);

B – конечное непустое множество ***выходных сигналов*** (***выходной алфавит***);

$\delta : S \times A \to S$ – ***функция переходов***;

$\lambda : S \times A \to B$ – ***функция выходов***.

Определение. Автомат Мили $\mathrm{A} = (S, A, B, \delta, \lambda)$, у которого функция выходов задается в виде $\lambda(q, a) = h(\delta(q, a))$, где $h : S \to B$ – отображение (называемое *определяющим*), называется ***автоматом Мура***. То есть функция выходов автомата Мура определяется только состоянием автомата, но не q, как у автомата Мили, а состоянием $\delta(q, a)$, в которое он переходит при подаче буквы a.

Обозначение: $\mathrm{A} = (S, A, B, \delta, h)$.

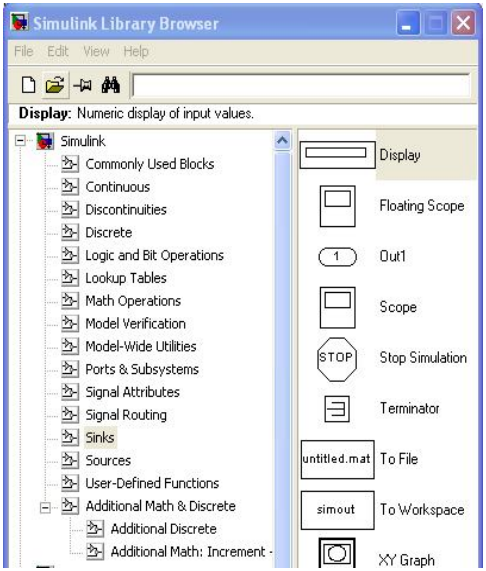
§ 2.2. Библиотека блоков. Окно модели.

Программа **Simulink** является приложением к пакету **MATLAB**. При моделировании с использованием **Simulink** реализуется *принцип визуального программирования*, в соответствии с которым, пользователь 1) создает на экране *модель устройства* из стандартных блоков библиотеки и 2) осуществляет расчеты. При этом, в отличие от классических способов моделирования, пользователю не нужно *досконально* изучать язык программирования и численные методы математики, а достаточно общих знаний, требующихся при работе на компьютере и, естественно, знаний той предметной области, в которой он работает.

Существует два способа запуска **SIMULINK:**

1. Нажать кнопку на панели инструментов командного окна **MATLAB**.
2. В строке окна **Command Window** набрать **Simulink** и нажать **Enter**. Открывающееся окно **Simulink Library Browser (**обозреватель библиотеки блоков) содержит следующие элементы (в порядке сверху

вниз): **1)** название окна; **2)** меню: **File**, **Edit**, **View**, **Help**; **3)** слева – панель инструментов, с ярлыками наиболее часто используемых команд.
4. Справа – окно комментария для вывода поясняющего сообщения о выбранном блоке.
5. Строка состояния, содержащая подсказку по выполняемому действию (на картинке – пояснение блока **Display**).
6. Окно слева – список разделов библиотеки.
7. Окно справа – блоки из активизированного раздела.



Библиотека **Simulink** содержит следующие (основные для начального знакомства) разделы:

1. Commonly Used Blocs–общеполезные блоки (повторяются в специализированных разделах).

2. Continuous – непрерывные блоки. Приведем некоторые из них. Блоки дифференцирования du/dt, интегрирования $\frac{1}{s}$, задержки сигнала; блок $\dot{x} = Ax+Bu$, $y = Cx+Du$ реализует MIMO (многомерные вход и вы-

ход) – непрерывную линейную стационарную систему, задаваемую *уравнениями перехода* и *выхода* $\begin{cases} x_t' = A \cdot x + B \cdot u \\ y = C \cdot x + D \cdot u \end{cases}$, где $x - n$-мерный *вектор состояния системы*, $u - m$-мерный *вектор входа*, $y - r$-мерный *вектор выхода*, а A, B, C, D – постоянные матрицы соответствующих размеров; блоки $\frac{1}{s+1}$, $\frac{(s-1)}{s(s+1)}$ реализуют SISO-(скалярные вход и выход) и SIMO (скалярный вход и векторный выход) – линейные непрерывные стационарные системы, задаваемые передаточной функцией (Transfer Fcn) соответственно по известным коэффициентам или нулям-полюсам.

3. Discontinuities. Приведем некоторые блоки. Блок ограничивает входной сигнал задаваемыми верхней и нижней величинами. Блок «мертвая зона» с задаваемыми границами $\tau < T$ возвращает выходной сигнал по формуле $y(t) = \begin{cases} x(t) - \tau, \ x(t) \le \tau \\ 0, \quad \tau < x(t) < T \\ x(t) - T, \ x(t) \ge T \end{cases}$.

4. Discrete – дискретные блоки.

5. Logic and Bit Operations – различные логические операции и детекторы.

6. Math Operations содержит более 30 блоков математических операций, включая скалярное перемножение векторов ; матричное умножение ; деление ; округление floor; функция сигнум ; блок **Gain** (коэффициент усиления); блок суммирования **Sum** (сумматор) (можно менять число входов и знаки слагаемых); блок взятия абсолютной величины |u|; блок некоторых элементарных функций и других операций e^u; блок вычисления минимума или максимума входного сигнала min; блок sin тригонометрических и гиперболических функций.

7. Ports &Subsystems. Приведем некоторые блоки. Введение блока **In** в *подсистему* или *модель* создает для них *внешний входной порт*; аналогично **Out** создает *внешний выходной порт*; блок-заготовка для создания *подсистем* (**Subsystem**); блок **Enable Subsystem** (*разрешенная подсистема*) разрешает работать подсистеме только при положительном значении *управляющего сигнала*.

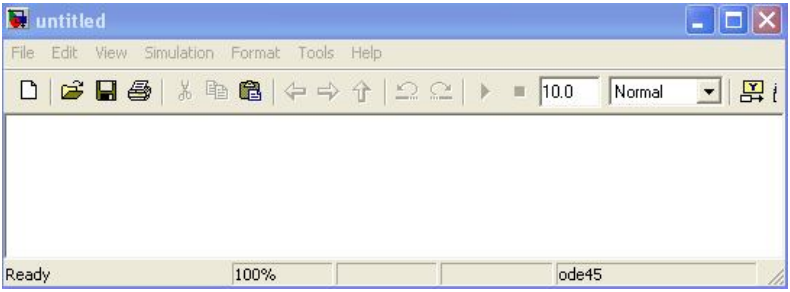
8. Sinks – регистрирующие устройства.

Цифровой дисплей для входных величин с возможность выбора их формата; блок Scope (осциллограф, дисплей) строит графики входных функций на одном или нескольких экранах, аргументом которых является время; графопостроитель (**XY Graph**) строит графики параметрически заданных функций, аргументом которых является время; блок **Terminator** (заглушка) блокирует не востребованные в системе выходы блоков.

9. Sources — источники сигналов разного типа. Блок **Clock** имеет выходной переменной текущее время моделирования; блок **Digital Clock** дает текущее дискретное время моделирования с задаваемым периодом; у блока выходом является постоянная скалярная или векторная величина; блок **Signal Builder** является конструктором сигналов, имеющих вид ломаных кривых; блок Signal Generator генерирует сигналы разного рода; блок присоединяется к незадействованным в модели входным портам блоков.

10. User-Defined Functions – функции, определяемые пользователем. Блок **Fcn** генерирует выходной сигнал по аналитической формуле, набранной в диалоговом окне блока.

Окно модели открывается несколькими способами: нажатием пиктограммы ; комбинация клавиш Ctrl+N; последовательностью вложенных закладок File ⇒ New ⇒ Model.



Окно содержит следующие элементы:

1) заголовок, с названием окна;
2) меню с командами **File** и т.д.;
3) панель инструментов;
4) окно для создания схемы модели;
5) строка состояния, содержащая информацию о текущем состоянии модели.

Меню окна:

File (*Файл***)** – работа с файлами моделей.
Edit (*Редактирование***)** – изменение модели и поиск блоков.
View (*Вид***)** – управление показом элементов интерфейса.
Simulation (*Моделирование***) –** задание настроек для моделирования и управление процессом расчета.
Format (*Форматирование***)** – изменение внешнего вида блоков и модели в целом.
Tools (*Инструменты***)** – отладчик, линейный анализ и другие средств для работы с моделью.
Help (*Справка***) –** вывод окон справочной системы.

Кнопки панели инструментов:

New Model – открыть окно для новой модели.
Open Model – открыть список **mdl**-файлов.
Save Model – сохранить модель в виде **mdl**-файла.
Print Model – вывести на печать блок-диаграмму модели.
Cut – вырезать выделенную часть модели в буфер.
Copy – копировать выделенную часть модели в буфер.

Paste – вставить в окно модели содержимое буфера.
Undo – отменить предыдущую операцию редактирования.
Redo – восстановить результат отмененной операции редактирования.
Library Browser – открыть окно обозревателя библиотек.
Toggle Model Browser – открыть окно обозревателя модели.
Go to parent system – перейти из подсистемы в “родительскую систему”.
Debug – запустить отладчик модели – окно **Simulink debagger**.
Start/Pause/Continue Simulation – запустить модель на исполнение (команда **Start**); после запуска модели этой же кнопкой приостанавливается моделирование (команда **Pause)**; после приостановки этой же кнопкой продолжается моделирование (команда **Continue)**.
Stop – закончить моделирование ■.
Окно **Normal/Accelerator** – обычный или ускоренный режим расчета. Доступно, если установлено приложение **Performance Tool**. Accelerator

Строка состояния:

1) отображаются краткие комментарии к кнопкам панели инструментов, а также к пунктам меню, когда указатель мыши находится над соответствующим элементом; это же поле используется для индикации состояния **Simulink**: Ready (Готов) или **Running** (Выполнение);
2) поле *масштаб отображения блок-диаграммы* 100%;
3) поле *сообщение о степени завершенности сеанса моделирования*;
4) поле *текущее значение модельного времени*;
5) поле *используемый алгоритм расчета состояний модели* (*метод решения*) ode45.

Блоки представляют собой элементарные динамические системы с входами, выходами и пространством состояний. При этом состояние может задаваться априори и не меняться в процессе симуляции, а может меняться в зависимости от входного сигнала (*блоки с памятью*).

Динамическая система создается в виде *структурной схемы.* В открытое окно модели из библиотек перетаскиваются необходимые блоки нажатием левой кнопки мыши. Затем их соединяют в нужном порядке *линиями связи.* При этом указатель мыши с нажатой левой кнопкой помещают на выход одного блока (должен появиться крестик), перемещают его на вход другого блока (должен появиться двойной крестик) и отпускают. Если соединяемые блоки находятся не на одном

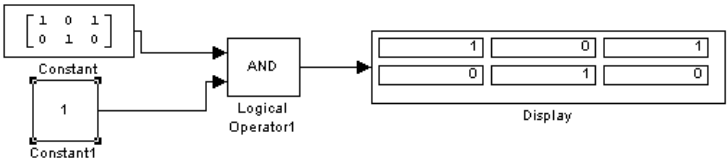
уровне, то линия связи будет с изломом. *Отрезки излома* можно *перемещать* с помощью нажатой левой копки. Аналогично *перемещаются точки излома*. Для *удаления линии связи* нужно ее активизировать щелчком мыши, и в открывшемся меню выбрать **Delete**. *Ответвление от* какой-либо точки *линии связи* создается правой кнопкой мыши либо одновременным нажатием левой кнопкой + клавиша **Ctrl**. *Излом линии связи* в какой-либо точке создается в этой точке левой кнопкой + клавиша **Shift**. Для *поименования линии связи* нужно рядом с ней два раза щелкнуть левой кнопкой мыши. Аналогично создается необходимая надпись в любой точке окна. Редактирование шрифтов (fonts) надписей осуществляется выбором последовательности команд Format $\Rightarrow$ Font.

§ 2.3. Логические блоки и конечные автоматы

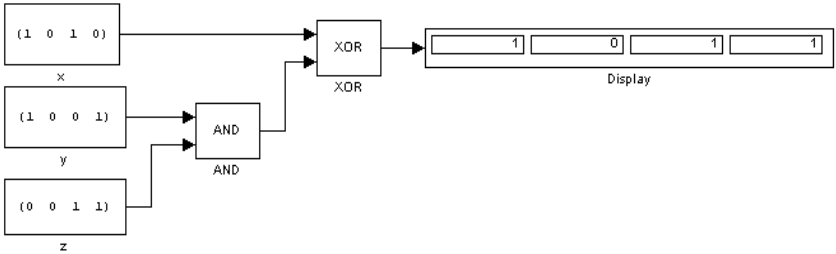
Для моделирования логических устройств, реализующих логические формулы, используется блок **Logical Operator** AND библиотеки **Logical and Bit Operations**. Первый параметр блока **Operator** реализует логические операции AND (конъюнкция), OR (дизъюнкция), NAND (И-НЕ, шрих Шеффера), NOR (ИЛИ-НЕ, стрелка Пирса), XOR (сложение по модулю 2), NOT (НЕ). Второй параметр **Number of input ports** задает количество входных портов (по числу булевых переменных в логической формуле). Входные сигналы допускаются скалярного, векторного и матричного типов.

Пр.1 Поэлементное логическое умножение матрицы размера 2×3 на 1. Блоки **Constant** по умолчанию имеют выходной тип данных **double**, а блок **Logical Operator –** тип данных **Boolean**. Последний нужно установить во всех блоках.

После окончания моделирования в правом нижнем углу дисплея появится треугольник, что означает наличие нескольких окон. Растягивая дисплей с помощью левой кнопки за один из концов, получим 6 окон (см. рисунок).

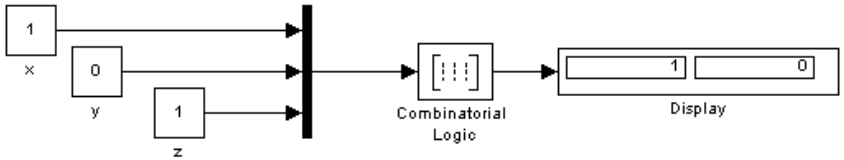


Пр.2 Вычисление логической формулы $x \oplus y \wedge z$ в четырех точках.



Для моделирования логических устройств, реализующих функциональные преобразователи и булевы функции, используется блок **Combinatorial Logic** из той же библиотеки. Его параметром является матрица функционального преобразователя. Именно, если последний $F = (f_1, ..., f_m) : A^n \to A^m$, $A = \{0,1\}$, зависит от n булевых переменных и определяется m логическими функциями f_j, то эта матрица имеет размер $2^n \times m$ и ее столбцами являются 2^n-мерные векторы, задающие соответствующие координатные функции f_j. Поскольку блок имеет один вход, то перед ним необходимо поставить блок **Mux** (*multiplex* – коммутировать, уплотнять каналы передачи) с n входами.

Пр.3 Пусть функциональный преобразователь определяется двумя координатными булевыми функциями от трех переменных $F = (f_1, f_2)$, где $f_1 = \{0,0,0,1,0,1,1,1\}$, $f_2 = \{0,1,1,0,1,0,0,1\}$. В текстовом окне **Truth table** (таблица истинности) *диалогового окна блока* набираем соответствующую матрицу размера 8×2.



Задача. Спроектировать логическое устройство «Электронный регистратор пропущенных ударов в боксерском поединке с тремя судьями. Очко противнику засчитывается, если удар признали не менее двух судей». Схему создать в элементном базисе Буля и в базисе И-НЕ.

Сначала создаем булеву функцию с тремя булевыми переменными по количеству судей. Переменная принимает значение 1, если судья засчитал удар. В противном случае значение 0. На основании словесного описания таблица истинности имеет следующий вид:

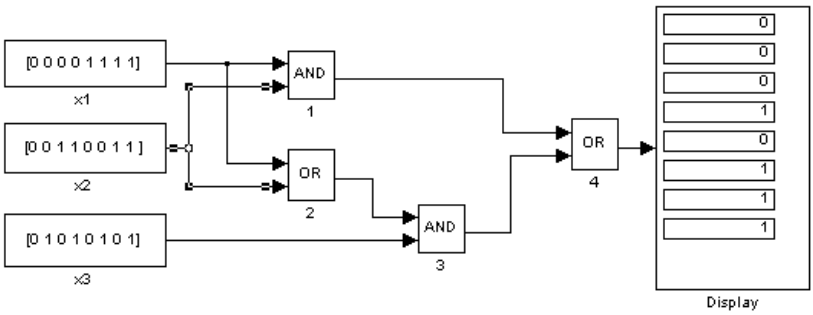
x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Минимизируя соответствующую СДНФ, получаем такую логическую формулу:

$$f(x_1, x_2, x_3) = x_1x_2 \vee x_1x_3 \vee x_2x_3 = x_1x_2 \vee (x_1 \vee x_2)x_3 .$$

По ней проектируем логическое устройство в *булевом базисе*.

На схеме показаны результаты принятия решения по 8 нанесенным ударам.



Поскольку $p \vee q = (p \mid p) \mid (q \mid q)$, $p \wedge q = (p \mid q) \mid (p \mid q)$, то логическая формула в базисе И-НЕ принимает вид:

$$f(x_1, x_2, x_3) = \Big(\big((x_1 \mid x_2) \mid (x_1 \mid x_2)\big) \mid \big((x_1 \mid x_2) \mid (x_1 \mid x_2)\big)\Big) \mid$$

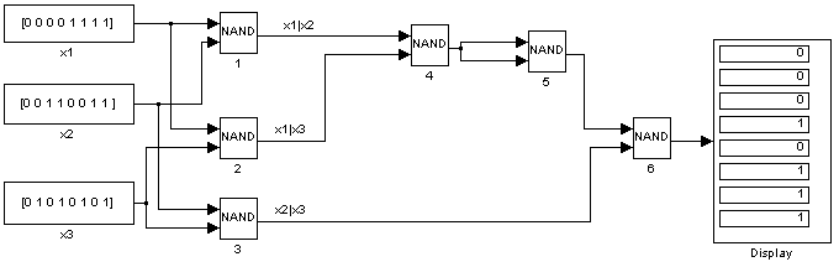
$$\mid \left(\begin{array}{l} \Big(\big(\big((x_1 \mid x_1) \mid (x_2 \mid x_2)\big) \mid x_3\big) \mid \big(\big((x_1 \mid x_1) \mid (x_2 \mid x_2)\big) \mid x_3\big)\Big) \mid \\ \mid \Big(\big(\big((x_1 \mid x_1) \mid (x_2 \mid x_2)\big) \mid x_3\big) \mid \big(\big((x_1 \mid x_1) \mid (x_2 \mid x_2)\big) \mid x_3\big)\Big) \end{array} \right).$$

Если же воспользоваться определением $\overline{p \wedge q} = p \mid q$ и законом двойного отрицания, получим более простую формулу:

$$f(x_1, x_2, x_3) = \overline{\overline{x_1 x_2 \vee x_1 x_3} \wedge \overline{x_2 x_3}} = \overline{x_1 x_2 \vee x_1 x_3} \mid (x_2 \mid x_3) = \overline{x_1 x_2} \wedge \overline{x_1 x_3} \mid (x_2 \mid x_3) =$$

$$= \overline{\overline{(x_1 \mid x_2) \wedge (x_2 \mid x_3)}} \mid (x_2 \mid x_3) = \overline{(x_1 \mid x_2) \mid (x_2 \mid x_3)} \mid (x_2 \mid x_3).$$

По ней и строим логическую схему в базисе И-НЕ.



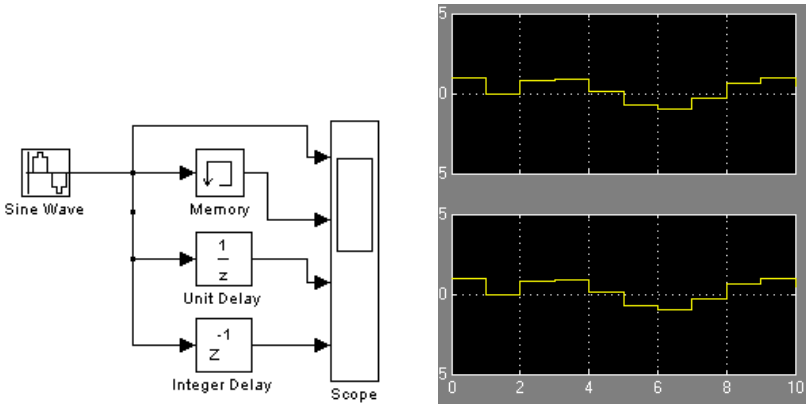
Рассмотрим еще три блока из библиотеки **Logical and Bit Operations**.

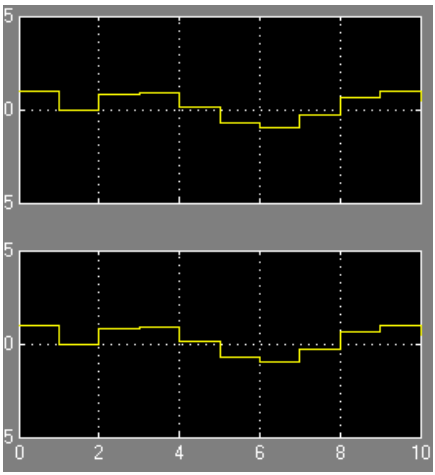
В блоке **Memory** значение выходной переменной совпадает со значением входной переменной на предыдущем шаге. Для функций, реализующих интегрирование с переменным шагом, время задержки изменяется в процессе интегрирования.

Блок **Unit Delay** для процессов с постоянным шагом производит задержку входного сигнала ровно на один шаг.

Блок **Integer Delay** для те же процессов производит задержку на заданное число шагов.

Эти блоки при соответствующей настройке могут работать одинаково, как показывает пример.





Блок **Memory** можно рассматривать как простейший автомат Мура ***D- триггер*** (*элемент задержки*). Поэтому с его помощью можно проектировать конечные автоматы.

Задача. Спроектировать автомат «Мудрый отец» по следующему словесному описанию. Пусть входной алфавит обозначает оценки, приносимые из школы сыном: $A = \{a_1 = 2,\ a_2 = 3,\ a_3 = 4,\ a_4 = 5\}$; выходной алфавит – внешнюю реакцию отца: $B = \{b_1 = \text{íàêàçàòü},$ $b_2 = \text{ïîðóãàòü},\ b_3 = \text{óñïîêîèòü},\ b_4 = \text{âûðàçèòüíàäåæäó},\ b_5 = \text{ïîðàäîâàòüñÿ},$ $b_6 = \text{ïîõâàëèòü}\}$; пространство состояний – возможные психические состояния отца:
$S = \{q_1 = \text{крайнее раздражение},\ q_2 = \text{напряженность},\ q_3 = \text{миролюбие},$ $q_4 = \text{удовлетворение}\}$. По этим данным, руководствуясь здравым смыслом, составляем таблицы функций перехода и выхода:

δ:

$a \setminus q$	q_1	q_2	q_3	q_4
a_1	q_1	q_1	q_2	q_2
a_2	q_2	q_2	q_3	q_3
a_3	q_2	q_3	q_3	q_4
a_4	q_3	q_3	q_4	q_4

λ:

$a \setminus q$	q_1	q_2	q_3	q_4
a_1	b_1	b_2	b_3	b_3
a_2	b_2	b_3	b_3	b_4
a_3	b_3	b_4	b_5	b_5
a_4	b_4	b_5	b_5	b_6

С помощью *кодировщиков* и *декодировщиков*
для $K_A : A \to \{0,1\}^2$

A	a_1	a_2	a_3	a_4
$X = \{0,1\}^2$	00	01	10	11

для $D_S : \{0,1\}^2 \to S$

$Z = \{0,1\}^2$	00	01	10	11
S	q_1	q_2	q_3	q_4

для $D_B : \{0,1\}^3 \to B$

$Y = \{0,1\}^3$	000	001	010	011	100	101	110	111
E	b_1	b_2	b_3	b_4	b_5	b_6	-	-

преобразуем данный автомат в эквивалентный *цифровой* с входным алфавитом $X = \{0,1\}^2$, пространством состояний $Z = \{0,1\}^2$ и выходным алфавитом $Y = \{0,1\}^3$:

Z	X	$Z \times X$	G:	F:
x_1x_2	x_3x_4	$x_1x_2x_3x_4$	$x_1^+x_2^+$	$y_1y_2y_3$
00	00	0000	00	000
00	01	0001	01	001
00	10	0010	01	010
00	11	0011	10	011
01	00	0100	00	001
01	01	0101	01	001
01	10	0110	10	011
01	11	0111	10	100
10	00	1000	01	010
10	01	1001	10	010
10	10	1010	10	100
10	11	1011	11	100
11	00	1100	01	010
11	01	1101	10	011
11	10	1110	11	100
11	11	1111	11	101

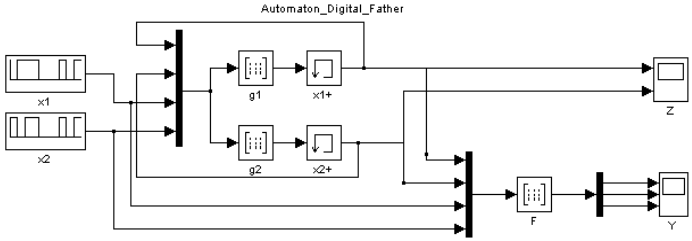
Здесь функция перехода

$$G : \{0,1\}^2 \times \{0,1\}^2 = \{0,1\}^4 \to \{0,1\}^2, \quad (x_1^+, x_2^+) := G(x_1, x_2, x_3, x_4),$$

а функция выхода

$$F : \{0,1\}^2 \times \{0,1\}^2 = \{0,1\}^4 \to \{0,1\}^3, \quad (y_1, y_2, y_3) := F(x_1, x_2, x_3, x_4).$$

По этим данным синтезируем конечный автомат. Для входных слов (-серии оценок) используем блоки **Repeating Sequence Stair**; в верхний вносим последовательность старших разрядов массива двухразрядных чисел, а в нижний – последовательность младших разрядов. В этих блоках выбирается выходной тип данных **Inherit via back propagation**, а период квантования **Simple time** равен 1. Начальные состояния блоков **Memory** это состояние отца перед поступлением первой оценки. В двух блоках **Combinatorial Logic** запишем соответственно старший и младший разряды функции перехода G, а еще в одном блоке – массив трехразрядных значений функции выхода F.



Приведем результаты эксперимента на этом автомате с начальным состоянием отца q_4 при соответствующей серии оценок у сына.

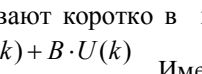
Оценки	Состояние	Реакция
3	q_4 – удовлетворение	выразить надежду
4	q_3 – миролюбие	порадоваться
5	q_3 – миролюбие	порадоваться
5	q_4 – удовлетворение	похвалить
2	q_4 – удовлетворение	успокоить
2	q_2 – напряженность	поругать
2	q_1 – крайнее раздражение	наказать
5	q_1 – крайнее раздражение	выразить надежду
2	q_3 – миролюбие	успокоить
5	q_2 – напряженность	порадоваться

§ 2.4. Дискретные системы

Математической моделью системы с дискретным временем является пара, составленная из функции перехода, задаваемой *нормальной системой разностных уравнений n-го порядка* $\begin{cases} x_1(k+1) = f_1(k, x_1(k), ..., x_n(k), u_1(k), ..., u_m(k)) \\ \quad . \quad . \quad . \\ x_n(k+1) = f_n(k, x_1(k), ..., x_n(k), u_1(k), ..., u_m(k)) \end{cases}$, $k = 0, \ldots$, и функции выхода $\begin{cases} y_1(k) = g_1(k, x_1(k), ..., x_n(k), u_1(k), ..., u_m(k)) \\ \quad . \quad . \quad . \\ y_r(k) = g_r(k, x_1(k), ..., x_n(k), u_1(k), ..., u_m(k)) \end{cases}$, $k = 0, \ldots$.

Для линейной стационарной дискретной системы имеем соответственно $\begin{cases} x_1(k+1) = a_{11}x_1(k) + \ldots + a_{1n}x_n(k) + b_{11}u_1(k) + \ldots + b_{1m}u_m(k) \\ \quad . \quad . \quad . \\ x_n(k+1) = a_{n1}x_1(k) + \ldots + a_{nn}x_n(k) + b_{n1}u_1(k) + \ldots + b_{nm}u_m(k) \end{cases}$ и

$$\begin{cases} y_1(k) = c_{11}x_1(k) + \ldots + c_{1n}x_n(k) + d_{11}u_1(k) + \ldots + d_{1m}u_m(k) \\ \quad . \quad . \quad . \\ y_r(k) = c_{r1}x_1(k) + \ldots + c_{rn}x_n(k) + d_{n1}u_1(k) + \ldots + d_{nm}u_m(k) \end{cases}.$$

Эти последние системы записывают коротко в виде системы матричных уравнений $\begin{cases} X(k+1) = A \cdot X(k) + B \cdot U(k) \\ Y(k) = C \cdot X(k) + D \cdot U(k) \end{cases}$. Именно так обозначается блок **Discrete State-Space** 

библиотеки **Discrete**.

Инструментом решения таких систем является Z-преобразование, которое ставит в соответствие последовательности чисел (*оригиналу*) $\{x(k)\}$ функцию комплексного переменного (*изображение*) $X(z) = \sum_{k=0}^{\infty} \frac{x(k)}{z^k}$. Оно преобразует линейные разностные уравнения и системы соответственно в линейные алгебраические уравнения и системы. В частности, уравнение перехода для SISO – линейной стационарной дискретной системы задается в виде разностного уравнения $b_0 y(k+n) + \ldots + b_n y(k) = a_0 x(k+n) + \ldots + a_m x(k), \quad k = 0, \ldots$. Если применить к нему Z-преобразование при нулевых начальных условиях, то получим алгебраическое уравнение:

$$Y(z) = \frac{a_0 z^m + \ldots + a_m}{b_0 z^n + \ldots + b_n} X(z) =: T(z) X(z) \cdot$$

Рациональная функция $T(z)$ является характеристикой уравнения и называется *передаточной функцией системы* (*transfer function*). В библиотеке **Discrete** имеется несколько блоков со специальными передаточными функциями, которые по известному входному сигналу $x(k)$ возвращают решение $y(k)$ разностного уравнения этой передаточной функции. Блок

определяется передаточной функцией

$T(z)=\dfrac{1}{z}$, и потому решает разностное уравнение вида $x(k+1)=u(k),\ x(0)=0$. Так как он задерживает входной сигнал на один такт, то называется **Unit Delay** (задержка). Уравнение с произвольной передаточной функцией и нулевыми начальными условиями решается блоком **Discrete Transfer Fcn** $\boxed{\frac{1}{z+0.5}}$.

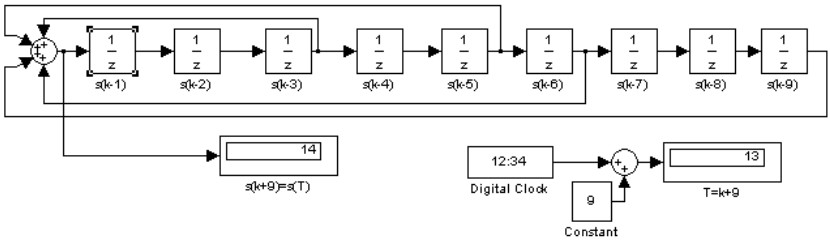
Задача. Пусть сообщение передается с помощью m сигналов длительностью соответственно $t_1<\ldots<t_m$ единиц времени. Известно, что число различных сообщений, которые можно передать с помощью этих сигналов за $\boldsymbol{t}$ единиц времени, вычисляется по рекуррентной формуле $s(t)=s(t-t_1)+\ldots+s(t-t_m)$. Считаются только максимальные сообщения, то есть сообщения, к которым нельзя присоединить ни одного сигнала, не превысив при этом $\boldsymbol{t}$. Спроектируем устройство, вычисляющее количество различных сообщений, которые можно передать за произвольный промежуток времени, при четырех базовых сигналах длительностью соответственно $t_1=3,\ t_2=5,\ t_3=6,\ t_4=9$.

Рекуррентная формула принимает вид $s(t)=s(t-3)+s(t-5)+s(t-6)+s(t-9)$. Это разностное уравнение девятого порядка, и поэтому надо знать 9 начальных данных. Несложный подсчет показывает, что

$$s(0)=0,\ s(1)=0,\ s(2)=0,\ s(3)=1,\ s(4)=1,$$
$$s(5)=2,\ s(6)=3,\ s(7)=3,\ s(8)=4.$$

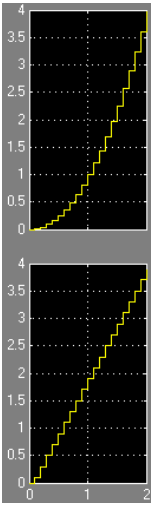
Напомним, что блок **Unit Delay** преобразует входной сигнал в $x(k-1)$. Расположим последовательно 9 экземпляров этого блока и свяжем их. Если вход первого блока равен $x(k)$, то выход последнего будет $x(k-9)$. Из рекуррентной формулы следует, что вход первого блока совпадает с суммой выходов третьего, пятого, шестого и последнего девятого блоков. Настроим блок **Sum** на 4 входа и соединяем их с этими выходами. Выход сумматора соединим с входом первого из блоков, а также выведем на дисплей. В диалоговых окнах блоков задержки зададим начальные условия последовательно 4,3,3,2,1,1,0,0,0.

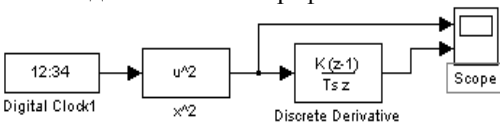
Число, посылаемое на вход устройства (длительность времени моделирования) отличается от интересующего нас времени $\boldsymbol{t}$ на 9. Поэтому создадим еще один дисплей, выводящий истинное значение $\boldsymbol{t}$.



Тестирование устройства с параллельным подсчетом вручную дает совпадение результатов. Задавая, например, время моделирования *к=4*, получаем *s(13)=14*.

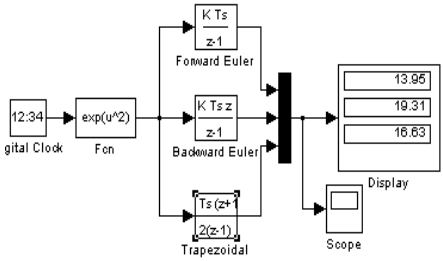
Блок **Discrete Derivative** (*разделенная разность*) работает с сеточными функциями, заданными на сетке с постоянным шагом (*sample time* – эталонное время). Шаг задается в блоке **Digital Clock**. В примере на вход этого блока подается функция x^2 с шагом $h = 0{,}1$. На выходе получается сеточная функция, составленная из разделенных разностей. Ожидаемая ее линейность видна на нижнем графике.

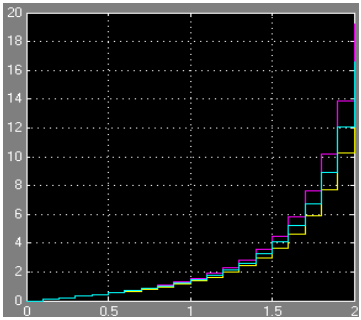




Настройки блока **Discrete-Time Integrator** включает три квадратурные формулы: прямой метод Эйлера **ForwardEuler** (левых прямоугольников), обратный метод Эйлера **BackwardEuler** (правых прямоугольников), метод трапеций (*Trapezoidal*) и шаг интегрирования **Sample time**. Следующий пример демонстрирует применение всех трех методов к вычислению $\int_0^2 e^{x^2} dx$ с шагом $h = 0{,}1$.

Пр. quad(@(x) exp(x.^2),0,2) $\Rightarrow$ ans = 16.4526 – вычисление в MATLAB.





Задача (мажордома). За круглый стол необходимо усадить ***k*** супружеских пар так, чтобы никакие два супруга не сидели рядом. Обозначим *A(k)* число способов, которым это можно сделать; *B(k)* (*C(k)*) – число способов, при которых рядом сидит ровно одна пара (ровно две пары) супругов. Тогда рекуррентная формула задается в виде такой системы разностных уравнений:

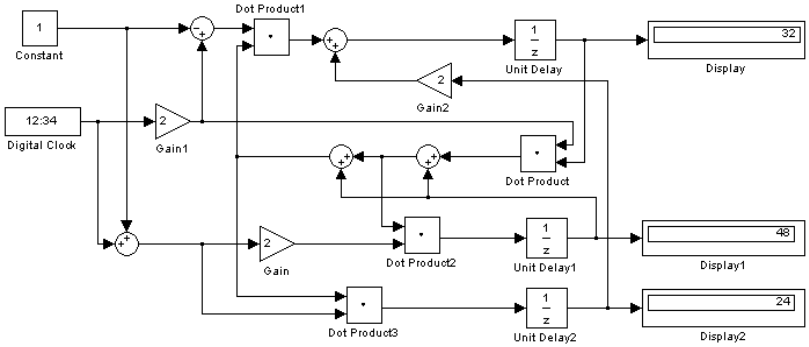
$$\begin{cases} A(k+1) = 2k(2k-1)A(k) + 2(2k-1)B(k) + 2C(k) \\ B(k+1) = 4k(k+1)A(k) + 2(k+1)B(k) \\ C(k+1) = 2k(k+1)A(k) + 2(k+1)B(k). \end{cases}.$$

Для упрощения схемы сгруппируем правые части.

$$\begin{cases} A(k+1) = (2k-1)\big(2k\,A(k) + 2B(k)\big) + 2C(k) \\ B(k+1) = 2(k+1)\big(2k\,A(k) + B(k)\big) \\ C(k+1) = (k+1)\big(2k\,A(k) + 2B(k)\big). \end{cases}.$$

Несложно вычислить *A(2)=2, B(2)=0, C(2)=4*. Решение существует, начиная с k=2. Поэтому в окне модели **Simulation** | **Configuration Parameters** установим начальное время **Start Time** равным 2.

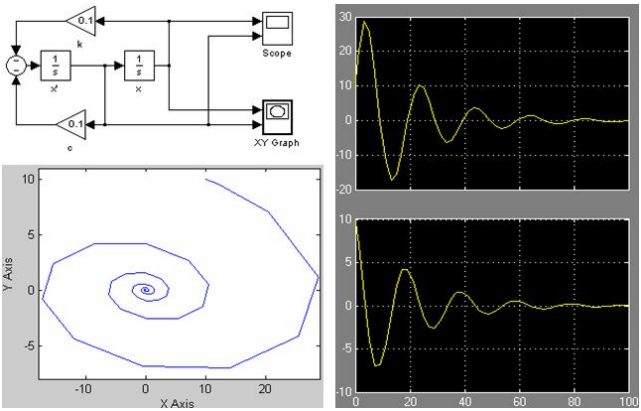
1) На блоках задержки задаем начальные условия последовательно 2, 0, 4.
2) Присоединяем к ним слева сумматоры и в соответствии с уравнениями формируем число входов и их знаки.
3) Образуем сомножители. Устанавливаем блоки **Dot Product**.
4) К выходам блоков задержки присоединяем дисплеи.



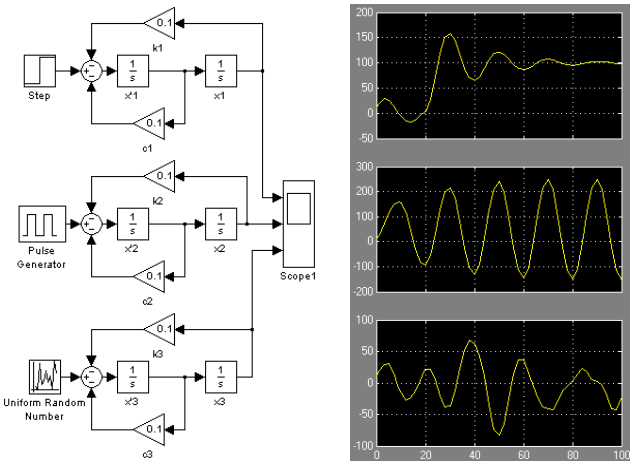
Тестирование при *k=3* подтверждает правильность работы устройства.

§ 2.5. Непрерывные системы

Создадим имитационную модель материальной точки единичной массы, колеблющейся прямолинейно относительно начала координат под действием силы трения с коэффициентом трения ***c*** и силы, пропорциональной расстоянию до начала координат с коэффициентом жесткости ***к***. Уравнение свободных колебаний имеет вид $x'' + cx' + kx = 0$, где $x(t)$ – координата точки в момент времени ***t***. Согласно критерию Рауса-Гурвица эта система (асимптотически) устойчива (то есть колебания с любым начальным условием затухают) тогда и только тогда, когда $c, k > 0$. Создаем структурную схему, как показано на рисунке. Параметры системы (коэффициенты усиления в блоках **Gain**) выберем достаточно малыми: $c = k = 0{,}1$, а начальные скорость и ускорение точки (= параметры Initial condition в блоках **Integrator**) достаточно большими: $x(0) = x'(0) = 10$. На графиках видно, что колебания и скорость затухают с течением времени. На графопостроителе по оси X откладывается $x(t)$, а по оси Y – $x'(t)$. Траектория наматывается на начало координат, которое является *устойчивым фокусом* рассматриваемой системы. Для того чтобы на экране графопостроителя вместилась вся траектория, нам пришлось подбирать параметры этого блока.



В следующем эксперименте рассмотрим вынужденные колебания точки под действием сил разного характера. Из множества блоков выходных сигналов выберем блок **Step** (ступенька, ступенька Хевисайда $10 \cdot \chi(t-20)$, то есть постоянная сила величиной 10 начинает действовать на 20-й секунде), блок **Pulse Generator** (генератор импульсов с амплитудой 10, периодом 20 и шириной импульса 10) и блок **Uniform Random Number** (равномерно распределенный случайный сигнал). Структурная схема и графики законов движения точки показаны ниже.



———

Создадим устройство, имитирующее движение автомобиля с автопилотом, поддерживающим постоянную скорость v_0 при учете аэродинамической силы (лобовое сопротивление) и профиля пути. Автопилот представляет собой *пропорциональный регулятор*: сила тяги-торможения $F_т$ пропорциональна рассогласованию $v_0 - s'(t)$ между задаваемой и линейной скоростью. Естественно потребовать ограниченность этой силы, скажем, $-1000 \le F_т(t) \le 100$. Коэффициент пропорциональности (*обратной связи*) обозначим k.

Пусть профиль дороги синусообразный $y(x) = 10h \cdot \sin(10^{-2} f \cdot x)$, то есть вершины холмов достигают высоты $10h$ и повторяются с периодом $200\pi / f$. Так как тангенс угла наклона профиля к горизонтали равен $tg(\theta(x)) = \dfrac{dy}{dx} = 10^{-1} h f \cos(10^{-2} f \cdot x)$, то проекция веса автомобиля на касательную к профилю равна

$$F_д(x) = mg \sin(\theta(x)) = \frac{mgtg(\theta(x))}{\sqrt{1+tg^2(\theta(x)}} = \frac{mgy'_x}{\sqrt{1+y'^2_x}} = \frac{mg\,10^{-1} h f \cos(10^{-2} f \cdot x)}{\sqrt{1+10^{-2} h^2 f^2 \cos^2(10^{-2} f \cdot x)}}.$$

Автомобиль не отрывается от дороги, то есть предполагается уравнение связи:

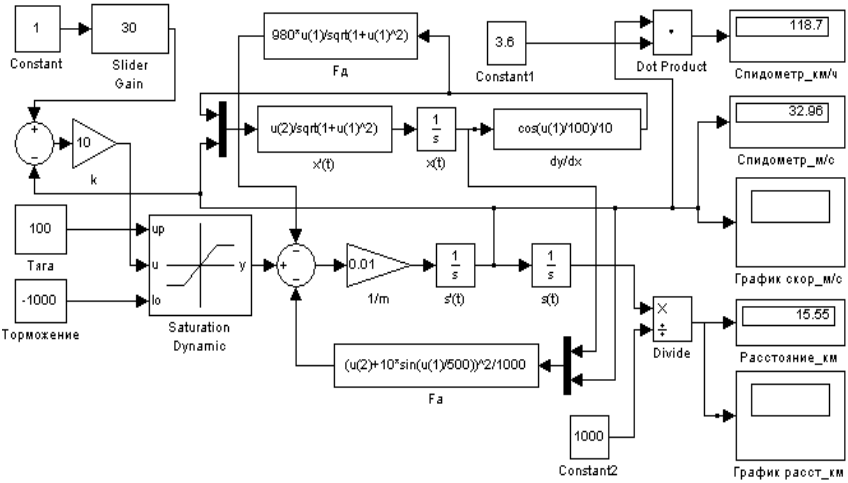
$$y(t) = 10h \cdot \sin(10^{-2} f \cdot x(t)).$$

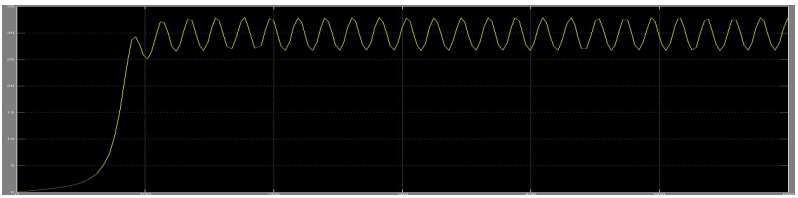
Тогда пройденный путь равен $s(t) = \int_0^t \sqrt{x'^2_t + y'^2_t}\,\mathrm{sgn}\, x'_t dt$, а линейная скорость $s'(t) = \sqrt{1+y'^2_x}\, x'_t$.

Предположим, что ветер дует параллельно профилю дороги по закону $10w\sin(x/500)$, то его скорость зависит только от точки дороги, но не зависит от времени. Аэродинамическая сила пропорциональна квадрату суммы скорости автомобиля и скорости ветра. Положим $F_а(t) = 0.001 \cdot \left(s'(t) + 10w\sin(x(t)/500)\right)^2$.

По третьему закону Ньютона имеем $m \cdot s''(t) = F_т(t) - F_а(t) - F_д(x(t))$. Это нелинейное дифференциальное уравнение второго порядка.

Создадим структурную схему, полагая $m = 100кг$, $g = 9,8$, скорость ветра $10w = 10м/с$, высота холмов $10h = 10м$, расстояние между холмами $628f = 628м$. Для ограничения силы тяги-торможения используем блок **Saturation Dynamic** (динамическое насыщение), который ограничивает сверху величину второго входа значением первого входа, а снизу – значением третьего входа. Аэродинамическая сила и неровности дороги задаются с помощью блока **Fcn**.

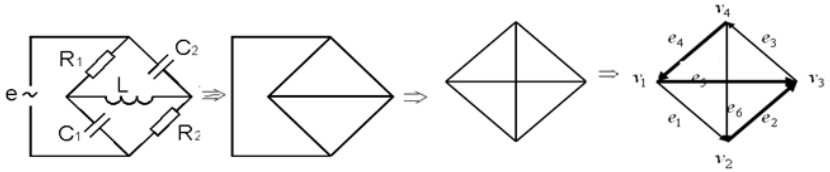




Эксперимент с моделью проводился при рабочей скорости, $V_0 = 30м/с = 108км/ч$. На дисплеях указана скорость машины через $t = 600с = 10мин$. На осциллографе – график изменения скорости в течение этого времени. Колебание $30 \pm 2,5м/с$ согласуется с выбранным профилем дороги.

Задача. По заданной электрической схеме построить ее имитационную модель. Создать устройство, рассчитывающее падение напряжений на всех двухполюсниках.

В соответствии с теорией графов переходим от схемы к орграфу.



Направление дуг задается произвольным образом. Соответствие между элементами и дугами следующее:

$$C_1 \leftrightarrow e_1 \quad R_2 \leftrightarrow e_2 \quad C_2 \leftrightarrow e_3 \quad R_1 \leftrightarrow e_4 \quad L \leftrightarrow e_5 \quad E \leftrightarrow e_6 .$$

С помощью *матрицы инцидентности* орграфа составляем уравнения Кирхгофа для токов $\begin{cases} -i_1 + i_4 - i_5 = 0 \\ i_1 - i_2 + i_6 = 0 \\ i_2 - i_3 + i_5 = 0 \\ i_3 - i_4 - i_6 = 0 \end{cases}$.

Исключаем уравнения, содержащие i_6. В оставшихся уравнениях переходим от токов на участках к соответствующим падениям напряжений по формулам: $i = \frac{1}{R}u \quad i(t) = \frac{1}{L}\int_0^t u(\tau)dt + i(0) \quad i(t) = Cu'_t$.

С помощью *цикломатической матрицы* составляем уравнения Кирхгофа для напряжений $\begin{cases} u_1 + u_2 - u_5 = 0 \\ u_3 + u_4 + u_5 = 0 \\ u_2 - u_4 - u_5 + u_6 = 0 \end{cases}$. Требуемая система уравнений принимает вид

$$\begin{cases} u_1 + u_2 - u_5 = 0 \\ u_3 + u_4 + u_5 = 0 \\ u_2 - u_4 - u_5 + E = 0 \\ \tilde{N}_1 u'_1 - \frac{1}{R_1}u_4 + \frac{1}{L}\int_0^t u_5(\tau)d\tau + i_5(0) = 0 \\ \frac{1}{R_2}u_2 - \tilde{N}_2 u'_3 + \frac{1}{L}\int_0^t u_5(\tau)d\tau + i_5(0) = 0 \end{cases}$$

. Это интегродифференциальная система, которую можно решить с помощью преобразования Лапласа.

Порядок создания структурной схемы следующий:

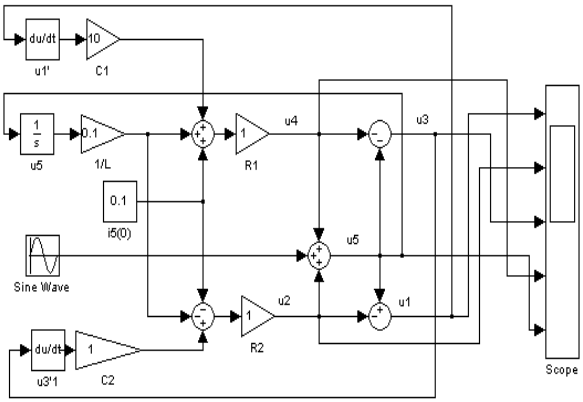
Сначала создаем схемы двух последних уравнений, выражая в них соответственно u_4 и u_2.

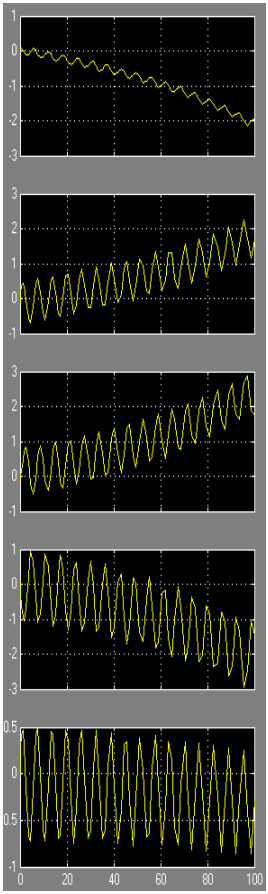
Затем создаем схему третьего уравнения, выражая в нем u_5 через остальные слагаемые.

С помощью полученных выходов создаем схемы первого и второго уравнений, выражая в них соответственно u_1 и u_3.

Выходы u_4 и u_2 соединяем со входами блоков дифференцирования **Derivative**, а выход u_5 – с блоком интегрирования **Integrator**.

Остается выбрать какой-либо блок с управляющим напряжением E и соединить все выходы с осциллографом.





На осциллографе показаны изменения напряжений в ветвях за время $t = 100$ ск при $E = \sin t$. Первый график – на C_1, второй – на R_2, третий – на C_2, четвертый – на R_1, пятый – на L.

§ 2.6. Подсистемы и маскирование

Определение. ***Блочно-иерархическим подходом*** к процессу моделирования называется расчленение системы на иерархические уровни, при котором на каждом уровне формируются задачи, поддающиеся пониманию и решению.

АЛГОРИТМ (создания подсистемы).

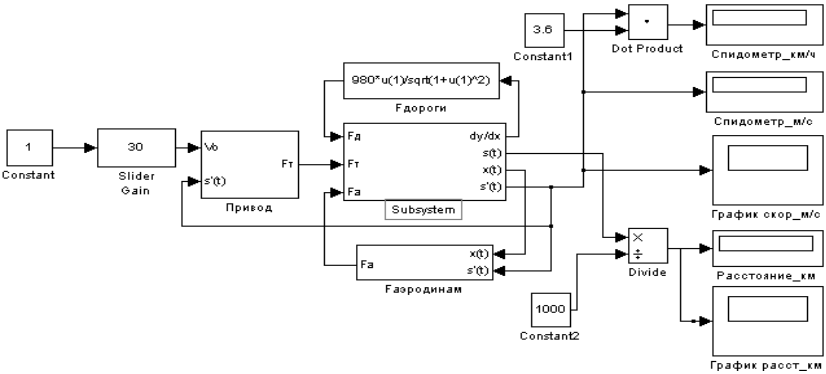
1) С помощью левой кнопки мыши выделяются все блоки и линии связи, которые войдут в подсистему.

2) Выбирается команда **Edit | Create Subsystem** (создать подсистему), которая на место выделенных блоков помещает блок **Subsystem**. Последний имеет столько входов **In** и выходов **Out**, сколько их имела выделенная подсистема.

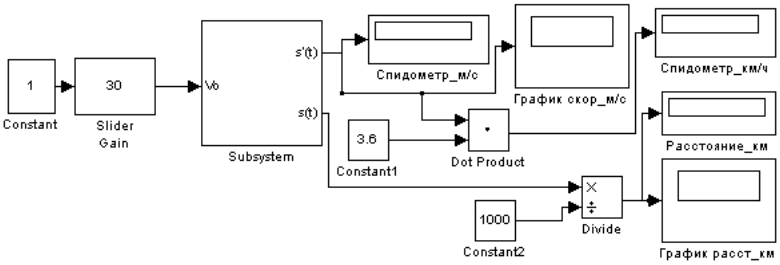
3) Для просмотра и редактирования содержимого подсистемы нужно на блоке два раза нажать левую кнопку мыши. В открывающемся окне входы автоматически соединены с блоками **In**, а выходы с блоками **Out**. Изменение их имен повлечет изменение имен соответствующих входов-выходов на изображении блока.

4. Команда **Create Subsystem** не может быть отменена, но копировать содержимое подсистемы можно.

Покажем процесс создания иерархической системы «Автомобиль с автопилотом» из структурной схемы предыдущего параграфа. Совокупности блоков, связанных с действующими на автомобиль силами, являются автономными в схеме, и потому преобразуем их в подсистемы, назвав соответственно "Привод", $F_{\text{дороги}}$, $F_{\text{аэродин}}$. Блоки, связанные с решением дифференциального уравнения, тоже соединяем в одну подсистему "Диффер. уравнение". Переименовываем входы и выходы подсистем.

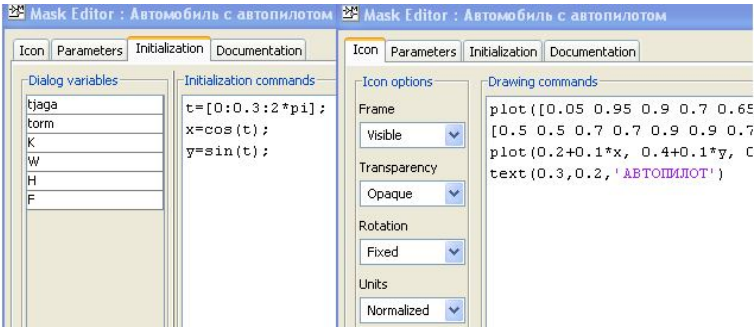


На следующем шаге создаем иерархию второго уровня, объединив все 4 подсистемы в одну с именем «Автомобиль с автопилотом».



Определение. Маскирование – процедура SIMULINK, которая позволяет отделить существенные поведенческие черты и свойства блоков (подсистем, систем) от особенностей их конкретной реализации. Маскированный блок может иметь свое собственное графическое изображение и свое диалоговое окно.

Покажем процесс маскирования подсистемы на предыдущем примере. Для создания диалогового окна блока необходимо его активизировать левой кнопкой мыши и выполнить команду **Edit** $\Rightarrow$ **Mask Subsystem** (маскировать подсистему).



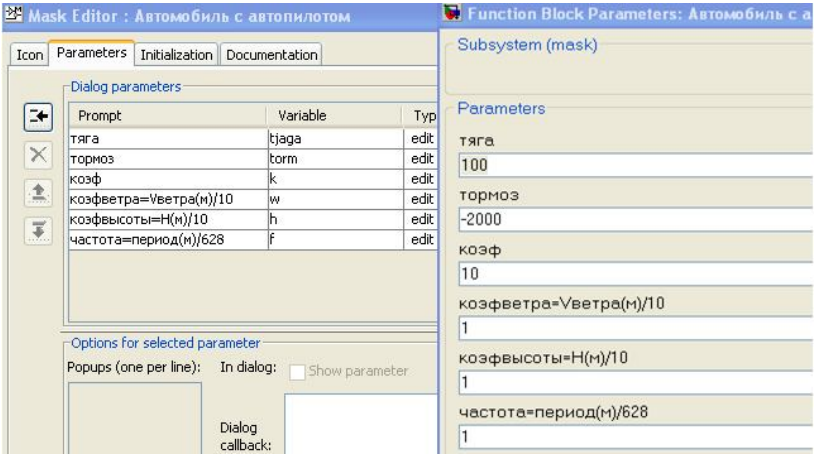
Появившееся диалоговое окно имеет 4 закладки.

1. **Initialization** (*задание начальных условий*). Левое окно – список переменных, используемых для настройки блока; правое окно – операторы для вкладки **Icon**.

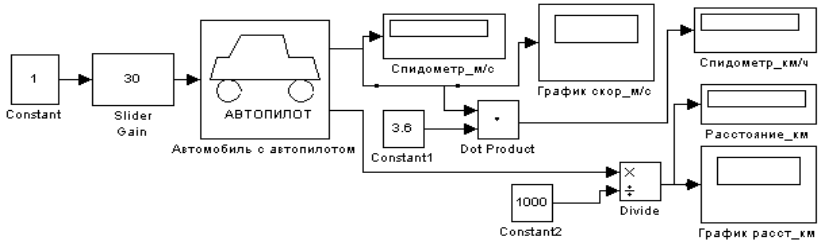
Они должны заканчиваться точкой с запятой. Мы создаем переменные ***x***, ***y***, которые будут использоваться для формирования изображения колес.

2. **Icon** (*пиктограмма*) формирует графическое изображение маскированного блока. В примере первые две строки задают контур автомобиля, третья строка – контуры колес, последняя строка – текстовая часть графического изображения. При этом координаты точек маски берутся в диапазоне от 0 до 1. Опции окна: **Frame** (рамка блока видима или нет); **Transparency** (настройка прозрачности изображения блока); **Rotation** (согласовывает вращение изображения с вращением блока); **Units** (определяет масштаб, используемый при выводе графики и текста на изображение блока).

3. **Parameters**. В нем создаются, редактируются и удаляются поля диалогового окна маскированного блока. **Prompt** (подсказка смысла параметра по его имени). **Variable** (обозначение параметра). Разработчик формирует список параметров исходя из целей работы с моделью. В примере выбраны 6 параметров (окно справа). После формирования списка он будет появляться в диалоговом окне маскированного блока (окно справа). Параметры можно будет редактировать в этом окне.

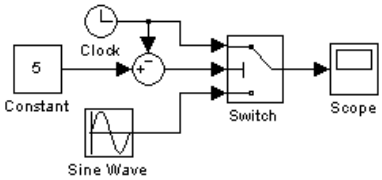


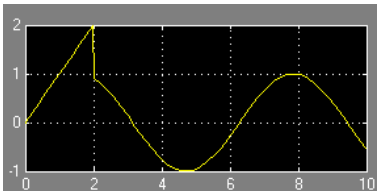
В результате получили не только изображение на блоке, но и быстрый доступ (через диалоговое окно) к изменению параметров блока.



Блок **Switch** (переключатель, коммутатор) библиотеки **Signal Routing** (маршрутизация сигналов) имеет три входных порта, второй из которых управляет переключением первого на третий. В диалоговом окне предусмотрены выбор порога h и 3 условия переключения: $u_2 \ge h,\ u_2 > h,\ u_2 \ne 0$.

Пр. На первый вход подается $u_1 = t$, на второй $u_2 = 5 - t$, на третий $u_3 = \sin t$, порог выбран $h = 3$, а условие переключения $u_2 \ge h$.



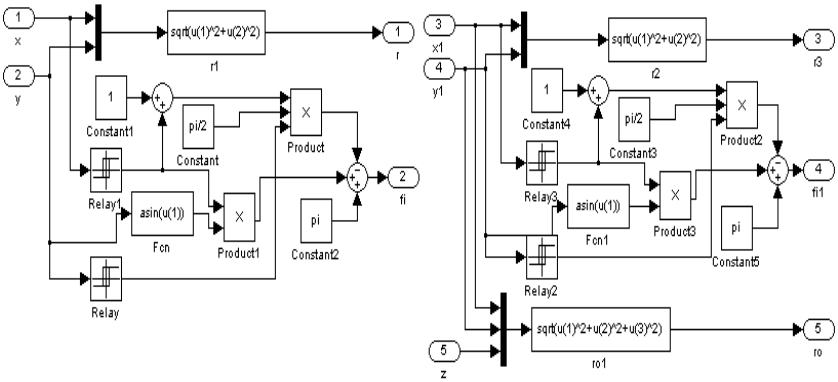


Для создания моделей с различными режимами функционирования (*с переменной структурой*) в SIMULINK используются блок **Configurable Subsys tem**. Технологию создания покажем на примере устройства, преобразующего декартовы координаты точки в ее полярные или цилиндрические координаты.

Сначала создаем структурную схему (левая схема на рисунке), вычисляющую по **x, y** расстояние до начала координат **r** и угол **fi** по формулам:

$$r = \sqrt{x^2 + y^2},\quad fi = sign(x)\arcsin\left(\frac{y}{\sqrt{x^2 + y^2}}\right) + \pi - \frac{\pi}{2}\left(1 + sign(x)\right)sign(y),$$

$$fi \in [0, 2\pi).$$



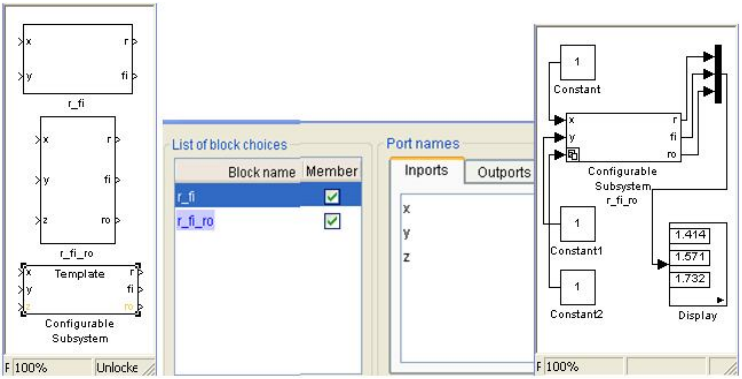
Для **fi** нельзя воспользоваться одним блоком **Fcn**, так как **sign** неэлементарная функция. Ее создадим с помощью блока **Relay** (*переключатель, реле*), полагая в диалогом окне этого блока **Switch on Point= Switch off Point=0**.

Затем создаем структурную схему, вычисляющую по **x**, **y**, **z** те же самые **r**, **fi** и расстояние до начала координат $ro = \sqrt{x^2 + y^2 + z^2}$ (правая схема).

Далее помещаем схемы в блоки **Subsystem** с именами **r**, **fi** и **r**, **fi**, **ro** (рисунок слева). Полученные подсистемы сохраняем в ***библиотечном блоке***: открываем окно **File| New| Library**; переносим их в это окно и сохраняем последнее.

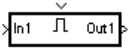
В сохраненное окно вставляем блок **Configurable Subsystem** из библиотеки **Subsystems** (левый рисунок). Открываем диалоговое окно последнего блока (средний рисунок) и в левом из окон напротив имен подсистем ставим флажки (галочки), чтобы включить эти подсистемы в систему с переменной структурой. При этом в правом окне появляются входные и выходные переменные.

Наконец, для создания *системы с переменой структурой* нужно открыть новое окно и перенести в него из только что созданного *библиотечного блока* блок **Configurable Subsystem**. В левом нижнем углу полученной копии появится значок (правый рисунок). Остается присоединить соответствующие источники сигналов ко входам и приемники к выходам этого блока.



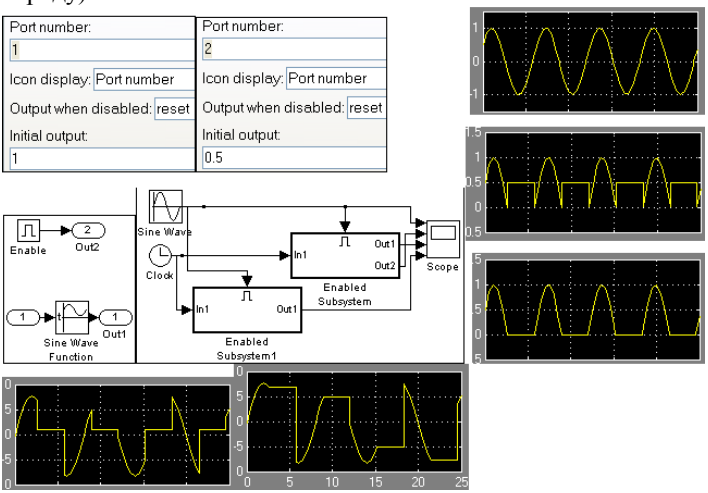
В полученной системе для смены структуры, то есть перехода от подсистемы **r_fi** к подсистеме **r_fi_ro** или наоборот, нужно в меню **Edit** окна модели выбрать команду **Block choice** (*выбор блока*), и в раскрывающемся списке выбрать имя соответствующей подсистемы.

В библиотеке **Subsystems** три условно выполняемых подсистемы.

Блок **Enabled Subsystem** (*разрешать*)  разрешает подсистеме вступать в работу, когда сигнал на управляющем порте положителен. В противном случае при выборе в раскрывающемся списке диалогового окна **Output** (верхний рисунок слева) пункта **Reset** выходная переменная блока принимает значение, указанное в текстовом поле **Initial output**. При выборе пункта **Held** значение выходной переменной будет равно последнему ее значению до того, как работа блока была запрещена.

Покажем возможности блока для подсистемы с сигналом $8\sin(\pi t/4)-0{,}3$ и управляющим сигналом $\sin(t+\pi/6)$ (верхний график справа) на временном отрезке $[0,25]$. В новом окне раскрываем блок **Enabled Subsystem** и создаем требуемую подсистему (второй рисунок слева). Раскрываем диалоговое окно выхода **out1** и устанавливаем **Reset** и **Initial output=1**. Создаем порт **out2** и в его диалоговом окне (верхний рисунок второго ряда) так же выбираем **Reset** и **Initial output=0.5**. На этот порт поступают значения внутренней переменной разрешающей подсистемы (второй график справа). На первом снизу графике показаны значения внешней переменной **out1**. Скопируем в этом же окне

созданный блок и в копии **Enabled Subsystem1** (нижний рисунок во втором ряду) в диалоговом окне выходного порта **out1** заменим **Reset** на **Held** (фиксация). На втором рисунке снизу приведен график выходного сигнала этого блока. Если в диалоговых окнах блока **Enable** обоих подсистем заменить позицию **Reset** на **Held**, то графики выходных портов не изменятся, но изменится график внутренней переменной (третий в правом ряду).



Триггерная подсистема (**Triggered Subsystem**) переходит в рабочее состояние при получении специального сигнала (прохождение управляющим сигналом нулевого значения.

§ 2.7. Качественный анализ стационарных систем (математическая модель «Караси и щуки»)

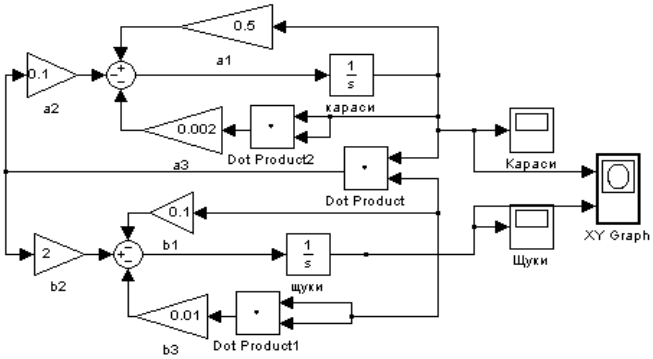
Рассмотрим математическую модель, которая является видоизменением математической модели Лотка-Вольтерра борьбы за существование и задается нелинейной *автономной* (правая часть не зависит от t) системой:

$$\begin{cases} x_t' = a_1 x - a_2 xy - a_3 x^2 \\ y_t' = -b_1 y + b_2 xy - b_3 y^2 \end{cases}.$$

Здесь $x(t)$ – количество карасей в пруду в момент времени *t*; $y(t)$ – количество щук; a_1 – естественный прирост (в отсутствии щук), создаваемый одним карасем в единицу времени, $a_1x(t)$ – естественный прирост, обеспечиваемый всеми карасями; a_2 – доля одного среднестатистического карася, съеденная в единицу времени при его контакте с одной щукой, a_2xy – количество съеденных карасей в единицу времени, когда в водоеме находится *x* карасей и *y* щук; a_3 – коэффициент, характеризующий конкуренцию карасей за пищу. Учитывая эти три фактора, получаем первое из уравнений системы.

b_1 – естественная смерть в единицу времени, приходящаяся на одну щуку при отсутствии карасей; b_1y – естественная смерть в единицу времени *y* щук; b_2 – коэффициент прироста в единицу времени, приходящегося на одну щуку при наличии одного карася; b_2xy – количество родившихся щук в единицу времени, если в водоеме находится *x* карасей и *y* щук; b_3 – коэффициент, характеризующий конкуренцию щук за пищу. Учитывая эти факторы, получаем второе уравнение. По смыслу параметров $b_1, a_2 \in (0,1)$.

Создаем структурную схему **Model_Karasi**.



Фазовое пространство автономной системы распадается на «бассейны» начальных состояний, для которых соответствующие траек-

тории ведут себя качественно одинаково с ростом ***t***. Для получения фазового портрета системы необходимо знать ее *положения равновесия* и их характер.

Для нахождения положений равновесия системы, приравняем к нулю правые части и решим соответствующую систему алгебраических уравнений:

```
>> [x y]=solve('a1*x-a2*x*y-a3*x^2','b1*y-b2*x*y+b3*y^2') ⇒
x =   0        0          a1/a3        (a1*b3+ a2*b1)/(b2*a2+b3*a3)
y =   0      -b1/b3          0          (a1*b2 -a3*b1)/(b2*a2+b3*a3)
```

То есть система имеет четыре положения равновесия. Для определения их характера необходимо линеаризовать правую часть в окрестности положения равновесия и вычислить собственные числа полученной матрицы. Так как функция линеаризации **linmod2** не работает с символьными данными, то выберем, например, параметры a1=0.5; a2=0.1; a3=0.002; b1=0.1; b2=2; b3=0.01.

[A,B,C,D]=**linmod2**('Model_Karasi',[0;0]) $\Rightarrow$ A = $\begin{matrix} 0.5 & 0 \\ 0 & -0.1 \end{matrix}$. **eig**(A) $\Rightarrow$ ans = -0.1 0.5.

То есть положение равновесия (0, 0) является седлом.

[A,B,C,D]=**linmod2**('Model_Karasi',[0;-10]) $\Rightarrow$ A = $\begin{matrix} 5.5 & 0 \\ -20 & 0.1 \end{matrix}$. **eig**(A) $\Rightarrow$ ans = 0.1 5.5.

Второе положение равновесии (0, -10) является неустойчивым узлом.

[A,B,C,D]=**linmod2**('Model_Karasi', [250;0]) $\Rightarrow$ A = $\begin{matrix} -0.5 & -125 \\ 0 & 499.9 \end{matrix}$. **eig**(A) $\Rightarrow$ ans = -0.5 499.9. Третье положение равновесия (250, 0) – седло.

По вышеприведенным формулам вычисляем четвертое положение равновесия. **subs**(a1*b3+ a2*b1)/(b2*a2+a3*b3) $\Rightarrow$ ans = 0.0750. **subs**(b2*a1-a3*b1)/(b2*a2+a3*b3) $\Rightarrow$ ans = 4.9985.

[A,B,C,D]=**linmod2**('Model_Karasi',[0.0750;4.9985]) $\Rightarrow$ A= $\begin{matrix} -1.9996 & -0.0375 \\ 9.9970 & -0.0500 \end{matrix}$.

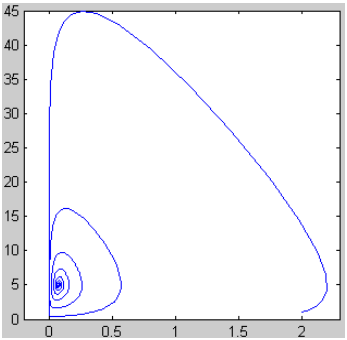
eig(A) $\Rightarrow$ ans = -1.7833 -0.2663.

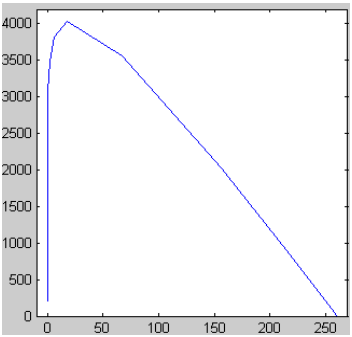
Четвертое положение равновесия (0.0750, 4.9985) – устойчивый узел.

На первом фазовом портрете начальные данные: два карася и одна щука. Фазовая траектория рассчитана для времени моделирования

t=1000. Она приближается к устойчивому состоянию 0.0750 карася и 4.9985 щуки.

На втором фазовом портрете начальные данные: 260 карасей и 10 щук, выбраны вблизи седла (250, 0). Очевидно, популяция карасей неуклонно снижается, а популяция щук в какой-то момент (t=0.018) достигает 4000, но уже при t=0.6 ввиду истребления карасей их становится меньше на 200 шт. Траектория приближается к положению равновесия (0.0750, 4.9985).

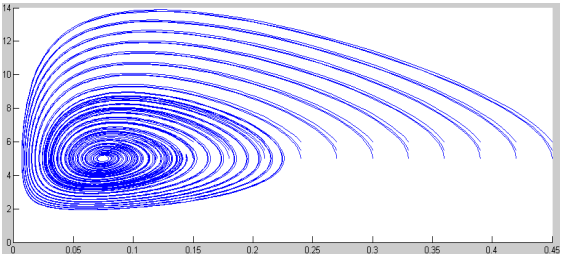




На левом графике кривая более гладкая, так как мы выбрали максимальный шаг интегрирования равным 0,1 (окно модели $\Rightarrow$ Simulation| Solver Options| Max step size).

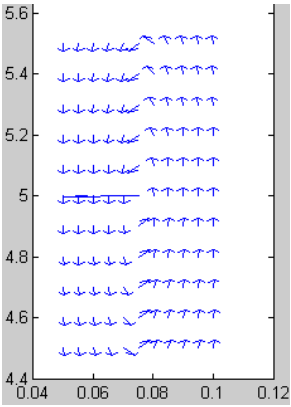
Построение *фазового портрета системы* в окрестности положения равновесия можно произвести с помощью следующей программы, которую можно набрать и запустить на исполнение в рабочем окне MATLAB. Для этого понадобится функция **simset**, которая будет работать в цикле вычислений. Она формирует структуру с необходимыми именами и значениями соответствующих полей. Например, opts =**simset**(‘Solver’,’ode45’, ”***FixStep***’,0.01, ***InitialState***’,[2,1]) задает численный метод **ode45** с постоянным шагом =0,01 и начальное условие (2;1). Функция **sim** запускает моделирование. Например, [t,x]=**sim** (‘***Karasi***’, ***T-Final*** ,***opts***) запускает моделирование модели ‘Karasi’ длительностью T-Final с параметрами opts и возвращает массивы расчетных моментов времени ***t*** и соответствующих значений решения ***x***.

```
>>  for x1=0:0.03:0.45;  for x2=5:0.5:6;
opts=simset('InitialState',[x1,x2]); [t,x,y]=sim('Karasi',50,opts);
hold on; plot(x(:,1),x(:,2));  end  end.
```



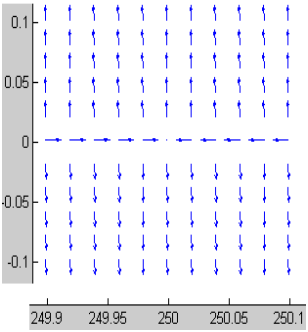
Если положения равновесия вычислить не удается, то сначала строят *поле направлений* автономной системы и по нему находят приближенные значения положений равновесия. Затем с помощью команды **trim** (*trimming* – подгонка) эти значения уточняют. Продемонстрируем это на предыдущем примере.

Следующая программа позволяет выделять фрагменты поля направлений автономной системы. На рисунках эти поля в окрестности четвертого и третьего положений равновесия. Функция **quiver(x,y,** Δ **x,** Δ **y)** (колчан) изображает вектор с началом в точке (x, y) и концом в точке (x+ Δ x, y+ Δ y). Команда **size**(A) возвращает размер массива A.




```
>> h=0.01;
opts=simset('Solver','ode5','FixedStep',h);
x1=249.9:0.02:250.1;
x2=-0.1:0.02:0.1;
[nr,nc]=size(x1);
x1m=zeros(nc,nc);
x2m=x1m;
for nx1=1:nc
for nx2=1:nc
opts=simset(opts,'InitialState',
[x1(nx1), x2(nx2)]);
[t,x,y]=sim('Model_Karasi',h,opts);
dx1=x(2,1)-x1(nx1);
```

```
 dx2=x(2,2)-x2(nx2);
l=sqrt(dx1^2+dx2^2)*80;
if l>1.e-10
x1m(nx2,nx1)=dx1/l;
x2m(nx2,nx1)=dx2/l;
end
end
end
quiver(x1,x2,x1m,x2m,0).
```



На первом фазовом портрете видно, что положение равновесия приближенно равно (0.078, 5). Находим точное значение:

$$>> [x]=\mathbf{trim}('Model_Karasi',[0.078;5]) \Rightarrow x=\begin{pmatrix}0.0750\\4.9985\end{pmatrix}.$$

Линеаризуем нашу систему в окрестности полученной точки, то есть найдем однородную НСЛДУ с постоянными коэффициентами для приращений решения. Ее матрица совпадает с матрицей Якоби правой части автономной системы, вычисленной в точке равновесия. Эта матрица находится с помощью команды **linmod**.

$>>[A,B,C,D]=\mathbf{linmod}('Model_Karasi',[0.0750,4.9985]) \Rightarrow$

$A=\begin{pmatrix}-0.0002 & -0.0075\\ 9.9970 & -0.0500\end{pmatrix}$ B = Empty matrix: 2-by-0 C= Empty matrix: 0-by-2

D = []

Находим собственные числа матрицы *A*.

$>> \mathbf{eig}(A) \Rightarrow ans = \begin{matrix}-0.0251 + 0.2727i\\ -0.0251 - 0.2727i\end{matrix}$. Так как они лежат в левой полуплоскости комплексной плоскости, то по теореме Ляпунова данное положение равновесия устойчиво (устойчивый фокус).

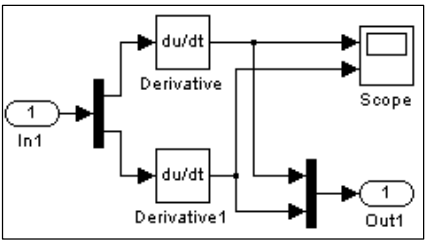
Из второго фазового портрета видно, что второе положение равновесия приближенно равно (250,0). Применяя последовательно команды trim, linmod, eig, установим, что это положение равновесия является неустойчивым (седло): $>> eig(A) \Rightarrow ans = \begin{matrix}-0.5000\\ 499.9000\end{matrix}$.

§2.8 Взаимодействие MATLAB и SIMULINK

Блок **to File** untitled.mat записывает значения величины, поданной на его вход, в М-файл данных для дальнейшего использования в других моделях SIMULINK (S-моделях). При этом первую строку выходной переменной блока образуют моменты времени; вторая строка содержит значения первой координаты входного вектора, третья - значения второй координаты и т.д. В окне настройки блока параметр *Filename* задает имя файла (по умолчанию – имя *untitled.mat*); параметр *Variable name* (*имя переменной*), применяется для обращения к данным, записанным в файле (по умолчанию используется переменная *ans*).

Блок **From Workspace** simin служит для включения сигналов, предварительно полученных в результате вычисления в среде MATLAB, в процесс моделирования. В поле ввода *Data* окна настройки указывается вектор [T,D], где T – массив значений аргумента, для которых определен процесс, D – массив соответствующих значений процесса. Вместо последнего блока можно подключить к блок-схеме S-модели блока **In** 1. Затем открыть окно с помощью последовательности команд Simulation| Simulation parameters| Workspace I/O. В области *Load from Workspace* этого окна установить флажок *Input*, и в поле справа ввести имя, состоящее из имени вектора значений аргумента и вектора значений входного сигнала. Затем запустить моделирование.

Блок **to Workspace** simout сохраняет данные в рабочем пространстве MATLAB. При этом значения модельного времени не записывается, а значения координат входного вектора записываются в виде столбцов выходной матрицы. В окне настройки блока параметр *Variable name* задает имя, под которым данные сохраняются в Workspace; в списке *Save format* нужно выбрать один из трех вариантов записи данных: *Array* (матрица, массив), Structure (структура), Structure with time.



Пр. Наберем в рабочем пространстве массивы аргументов и значений входных сигналов, например, >> t=[0:0.1:3.1]'; u=[sin(t), cos(t)]. Создаем блок-схему и в области *Load from Workspace* установим флажок *In-*

put, а в поле справа – имя [t,u]. В области *Save to Workspace* установим флажки Time с именем, например, *tout* и Output с именем *yout.* Запустим моделирование с t=3.1. На графике получаются волны синусоиды и косинусоиды. В рабочем пространстве с помощью команд *tout* и *yout* возвращаем массивы соответственно аргумента и производных входных сигналов.

———

Определение. ***S-функция*** – относительно самостоятельная программа, написанная на языке MATLAB или C, с помощью которой:

1) создаются новые блоки, дополняющие библиотеку пакета SIMULINK;

2) моделируемая система описывается в виде системы математических уравнений;

3) ранее созданные на языке MATLAB или C программы включаются в S-модели.

Программный код S-функции имеет структуру, которая приведена в файле SfunTMPL.m папки TOOLBOX/SIMULINK/BLOCS. Текст состоит из основной программы и текстов вспомогательных подпрограмм (*процедур*), которые она вызывает.

Входные аргументы S-функции: **t –** текущее значение времени моделирования; **x -** текущее значение *n* переменных состояния непрерывной части и *m* переменных состояния дискретной части; **u –** текущее значение вектора входа; **flag** – целочисленный параметр; устанавливается системой SIMULINK каждый раз, когда вызывается S-функция; последняя выполняет процедуру, соответствующую значению этого параметра; <**параметры**> - список необязательных параметров, совпадающий со списком параметров в текстовом поле *S-function parameters*.

Выходные аргументы S-функции: **sys**– многоцелевой аргумент; значения вектора **sys** зависит от значения параметра **flag**; **x0 –** вектор начальных значений переменных состояния; формируется при значении **flag**, равном 0; **str** – символьная переменная состояния; требуется, только когда значение параметра **flag** равно 0; **ts** – матрица квантования с двумя столбцами; в первом - значение периода квантования, а во втором – значение времени смещения; формируется при значении **flag**, равном 0.

Значения параметра **flag** :

0 вызывает процедуру **mdlInitializeSizes,** которая формирует структуру **sizes** и записывает значения ее полей в вектор **sys**; задает вектор **x0** и матрицу **ts**; определяет пустую матрицу **str** =[ ];

1 вызывает процедуру **mdlDerivatives**, которая вычисляет значения производных переменных состояния непрерывной части и записывает их в вектор **sys**;

2 вызывает процедуру **mdlUpdate**, которая вычисляет значения переменных состояния дискретной части и записывает их в вектор **sys**;

3 вызывает процедуру **outputs**, которая вычисляет вектор выхода и записывает его в вектор **sys**;

4 вызывает процедуру **mdlGetTimeOfNextVarHit**, которая вычисляет следующий момент времени квантования и записывает его в вектор **sys**;

9 вызывает процедуру **mdlTerminate**, которая прекращает работу блока S-функции; выходные аргументы отсутствуют.

Порядок создания блока S-функции:

1) на основании файла-шаблона SfunTMPL.m и с учетом заданных уравнений написать текст S-функции в виде M-файла;
2) перетащить в окно стандартный блок **S-Function** <system>;
3) открыть диалоговое окно и в поле *S-function name* ввести имя S-функции, а в поле *S-function parameters* – имена или значения параметров блока, указанные в списке <**параметры**>;
4) на вход полученного S-блока поступает векторный сигнал **u**; выходом станет векторный сигнал **y**, который сформируется S-функцией.

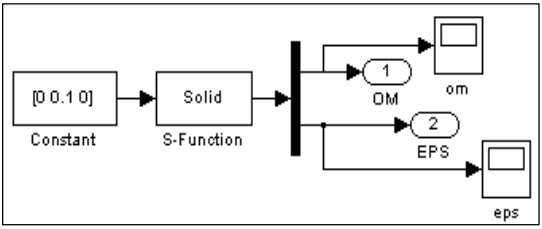
Задача Создать S-модель, которая описывает динамику твердого тела, вращающегося вокруг неподвижной точки *O*.

Математическая модель задается *динамическим уравнением Эйлера* $J\dfrac{d\omega}{dt}+[\omega, J\omega]=M$, где $J=\begin{pmatrix} Jx & -Jxy & -Jxz \\ -Jxy & Jy & -Jyz \\ -Jxz & -Jyz & Jz \end{pmatrix}$ - матрица осевых и центробежных моментов инерции в системе координат с началом в O, которая жестко связана с телом, $\omega=\begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}$ - матрица-столбец проекций вектора угловой скорости на соответствующие оси, M - матрица-столбец проекции момента внешних сил на оси, $[\omega, J\omega]$ - векторное произведение.

Сначала создадим S-функцию Solid.m. M будет входом S-модели; выходом будет вектор $\boldsymbol{y}$ с шестью компонентами: три проекции

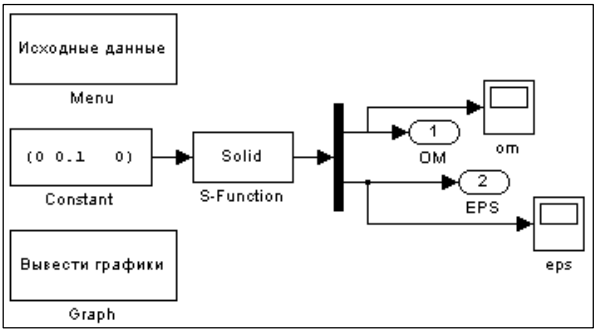
угловой скорости *omx, omy omz* (сокр. от ω) и углового ускорения *epsx, epsy, epsz* (сокр. От $\varepsilon = \omega'$); входными параметрами будут матрица J и вектор начальных значений угловой скорости тела $om0 = \{om0(1), om0(2), om0(3)\}$.

```
function [sys,x0,str,ts] = Solid(t,x,M,flag,J,om0)
switch flag,
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(om0);
   case 1,
    sys=mdlDerivatives(t,x,M,J);
   case 3,
    sys=mdlOutputs(x,M,J);
   case 9,
    sys=[];
 end
%=========================================================
function [sys,x0,str,ts]=mdlInitializeSizes(om0)
sizes = simsizes;
sizes.NumContStates  = 3;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 6;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 1; % проверка на наличие алгебраи-
ческого
контура в модели
sizes.NumSampleTimes = 1; % указать хотя бы один период
квантования
sys = simsizes(sizes);
x0=om0;
str=[];
ts=[0 0];
%=========================================================
function z=mdlDerivatives(t,x,M,J)
J=[J(1,1) -J(1,2) -J(1,3);-J(1,2) J(2,2) -J(2,3);-J(1,3)
-J(2,3) J(3,3)];
om=x(1:3);
z=J\(M-cross(om,J*om));
%=========================================================
function y=mdlOutputs(x,M,J)
y(1:3)=x;
om=x(1:3);
zom=J\(M-cross(om,J*om));
y(4:6)=zom;
```



Теперь организуем S-блок **Solid_M** такого вида. Здесь вход ной сигнал M={0; 0,1; 0}. Задавая в командном окне ***J*** и ***om0***, можно проверить его работу.

Так как графики в SIMULINK получаются на черном фоне, то напишем программу Solid_M_Graph.m их изображения средствами MATLAB. Для создания диалогового режима работы напишем программу Solid_M_Menu.m. Для этого в окно блок-схемы **Solid_ M** поместим два блока **Subsystem**; назовем их соответственно Graph и Menu; раскроем их и удалим содержимое; создадим маски с именами «Вывести графики» и «Исходные данные». В результате окно примет вид:




```
%Solid_M_Graph.m Формирование данных и вывод графиков
options=simset('Solver','ode45','RelTol',1e-6);
sim('Solid_M',[T],options);
tt=tout;
omx=yout(:,1); omy=yout(:,2); omz=yout(:,3);
epsx=yout(:,4); epsy=yout(:,5); epsz=yout(:,6);
StrJ=[sprintf('Jx= %g;',J(1,1)),sprintf('Jy=
%g;',J(2,2)),sprintf('Jz= %g;',J(3,3))]
StrJJ=[sprintf('Jxy= %g;',J(1,2)),sprintf('Jxz=
%g;',J(1,3)),sprintf('Jyz= %g;',J(2,3))]
StrU=[sprintf('omx= %g;',om0(1)),sprintf('omy=
%g;',om0(2)),sprintf('omz= %g;',om0(3))]
```

```
subplot(2,1,1)
h=plot(tt,omx,tt,omy,('.'),tt,omz,'--'); grid
set(h,'LineWidth',2);
set(gca,'LineWidth',2)
set(gca,'FontAngle','italic','Fontsize',16)
title('Проекция угловых скоростей')
set(gca,'Fontsize',14)
ylabel('радиан/с')
legend('omx','omy','omz',0)
xlabel([StrJ,'',StrJJ,'',StrU])
subplot(2,1,2)
h=plot(tt,epsx,tt,epsy,('.'),tt,epsz,'--');grid
set(h,'LineWidth',2);
set(gca,'LineWidth',2)
set(gca,'FontAngle','italic','Fontsize',16)
title('Проекция угловых ускорений')
set(gca,'Fontsize',14)
ylabel('радиан/с^2')
xlabel('Время (с)')
set(gcf,'color','Yellow')
legend('epsx','epsy','epsz',0)
%Solid_M_Menu.m Программа изменения исходных данных
k=1;
while k<11
k=menu('Параметры модели
Solid_M',sprintf('Jx=%g',J(1,1)),...
sprintf('Jy=%g',J(2,2)),sprintf('Jz=%g',J(3,3)),...
sprintf('Jxy=%g',J(1,2)),sprintf('Jxz=%g',J(1,3)),...
sprintf('Jyz=%g',J(2,3)),sprintf('omx=%g',om0(1)),...
sprintf('omy=%g',om0(2)),sprintf('omz=%g',om0(3)),...
sprintf('T=%g',T),'Ничего не менять');
if k==1
    J(1,1)=input([sprintf('Текущее значение
J(1,1)=%g;',J(1,1)),...
        'Установить новое значение J(1,1)= ']);
end
if k==2
    J(2,2)=input([sprintf('Текущее значение
J(2,2)=%g;',J(2,2)),...
        'Установить новое значение J(2,2)= ']);
end
if k==3
```

```
    J(3,3)=input([sprintf('Текущее значение
J(3,3)=%g;',J(3,3)),...
        'Установить новое значение J(3,3)= ']);
end
if k==4
    J(1,2)=input([sprintf('Текущее значение
J(1,2)=%g;',J(1,2)),...
        'Установить новое значение J(1,2)= ']);
end
if k==5
    J(1,3)=input([sprintf('Текущее значение
J(1,3)=%g;',J(1,3)),...
        'Установить новое значение J(1,3)= ']);
end
if k==6
    J(2,3)=input([sprintf('Текущее значение
J(2,3)=%g;',J(2,3)),...
        'Установить новое значение J(2,3)= ']);
end
if k==7
    om0(1)=input([sprintf('Текущее значение
omx=%g;',om0(1)),...
        'Установить новое значение OMx(0)= ']);
end
if k==8
    om0(2)=input([sprintf('Текущее значение
omy=%g;',om0(2)),...
        'Установить новое значение OMy(0)= ']);
end
if k==9
    om0(3)=input([sprintf('Текущее значение
omz=%g;',om0(3)),...
        'Установить новое значение OMz(0)= ']);
end
if k==10
    T=input([sprintf('Текущее значение T=%g;',T), ...
    'Установить новое значение T= ']);
end
end
```

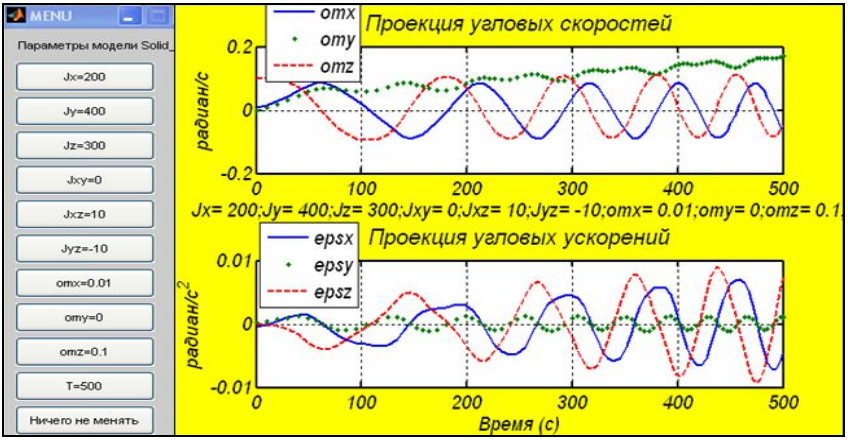
Можно, но не обязательно, создать программу, устанавливающие начальные значения параметров при запуске ***Solid.m***. Например,

```
% Solid_M_Pre.m
J=[200 0 0;0 400 0;0 0 300];
om0=[0.01 0 0.1];
T=500;
```

В рабочем пространстве свяжем эти три программы с S-моделью:

```
>> set_param('Solid_M/Graph','OpenFcn','Solid_M_Graph')
>> set_param('Solid_M/Menu','OpenFcn','Solid_M_Menu')
>> set_param('Solid_M','PredLoadFcn','Solid_M_Pre')
```

Проведем эксперимент со временем t=500 и начальными данными из приводимого ниже меню.



Для копирования блок-схемы в Word необходимо ее выделить левой кнопкой мыши и Edit|Copy Model to Clipboard. Текст М-файла копируется в Word и командное окно через буфер.

Словарь основных английских терминов

A

Accept – допуск; принимать
Access – доступ, подход
Array – числовой массив
Assignment – присвоение

B

Background – цвет фона
Bitwise – поразрядный
Bias – смещение, системная ошибка
Break(point) – нарушать, прерывать (останов)
Bus – шина, магистраль
Browse(r) – просматривать (программа просмотра)

C

Callback – звать обратно (отзывать)
Char – символ ,знак
Chart – карта; outlines – контурная карта
Check(box) – проверка (флажок)
Clipboard – ячейка для информационного обмена
Circuit – схема, цепь, контур, окружность
Configurable – конфигурируемый
Consecutive – последовательный
Constrain(ed) – вынуждать (вынужденный)
Constraint – ограничение, принуждение, связь
Conversion – преобразование

D

Debug – отлаживать, устранять (неполадки)
Decimation – прореживание
Default – по умолчанию, недосмотр
Delay – задержка
Disable – отключить, вывести из строя
Desire – желание; желать
Dock – уменьшить

E

Else – в противном случае, иначе
Enable – разрешённый; включить
Environment – окружение, среда
Exceed – превышать
Expect – ожидать, надеяться
Explorer – исследователь, зонд

F

Feature – черта, особенность, признак
Feed – подача, питание; подающий механизм
Foreground – приоритетный
Frame – кадр, система отсчета, рамка; группа данных, единица представления знаний, описывающих понятие;

G

Gain – коэффициент усиления; получать
Grid – сетка
Guess – догадка, приблизительная оценка

H

Handle – обрабатывать, оперировать, манипул.
Hide – скрывать (информацию)
Hit – попадание, удар
Hold (held) – фиксатор, захват, экстраполятор; хранить, удерживать, приостанавливать, удовлетворять.

I

Inherit – наследовать
Implement – выполнять, внедрить
Initialization – присвоение начального значения
Instantaneous – мгновенный
Invoke – вызывать, активировать

L

Load – загрузка, груз
Layout – размещение, компоновка
Log – регистрировать событие (в файл)
Loop – петля, цикл

M

Match – сравнивать, отождествлять
Merge – поглощать, соединять
Mode – режим

O

Offset – смещение, сдвиг
Opaque – непрозрачный, тупой
Option – выбор, возможность
Overlap – перекрывать, совмещать
Overload – перегрузка

P

Pane – клетка (в таблице)
Pars(ing) – синтаксический анализ
Paste – вставить
Plant – объект (управления), агрегат
Plot – график
Popup – раскрывающийся список
Profiler – обрабатывать (по шаблону)
Prompt – подсказка
Propagate – распространять, проходить
Provide – доставлять, давать

R

Ramp – пилообразный сигнал; (быстрое) изменение по линейному закону
Randle – обрабатывать, оперировать
Rate – степень, скорость, оценка
Reduce – понижать, уменьшать, сокращать
Relational (operator) – сравнения…
Relay – реле, переключатель
Remove – удалить
Reset – сброс
Residual – разность, остаток
Restore – восстановить
Routing – маршрутизация

S

Sample time – период квантования
Saturation – насыщение
Scale – масштаб

Seed – источник, начало
Set – устанавливать, присваивать значение
Setup – установка, схема, настройка
Simulate – моделировать, имитировать
Sink – сток, приемник
Scope – экран, рамка
Slide – скользить; движок (потенциометра)
Slope – наклон, склон
Span – диапазон, интервал, измерение
Specify – точно определять, уточнять
Split – щель, трещина, расщепление
Stair – ступенька, лестница
Steady – статический, установившийся
Stem – ствол, стебель
Stiff – жесткая (модель)
Storage (store) – хранение, хранилище (хранить)
String – строка, цепочка, последовательность
character~ – символьная строка
Subtract – вычитание
Sweep – развертка
Switch – переключатель, коммутатор

T

Target – адресат, цель, задание; объектный
Template – шаблон; маска
Threshold – порог, начало
Thrust – толчок, опора, тяга, нагрузка
Tick – делать отметку, отмечать
Tighten – сжимать, натягивать
Toggle – тумблер
Tolerance – допуск, терпимость
Transfer – пересылка, передача
Transition – переход, перемещение
Transparent – прозрачный, скрытый
Transport – перенос
Trim – порядок, состояние готовности
Tunable – легко настраиваемый

U

Undo – отменить, уничтожить
Update – обновить (диаграмму), корректировать

V

Vary – изменять, разниться
Violation – нарушение, противоречие

W

Warning – предупреждение
Width – ширина, длительность
Wise – образ, способ; мудрый
Wizard – ас, волшебник, фокусник

Библиографический список

1. Арнольд В.И. «Жесткие» и «мягкие» математические модели / В.И. Арнольд. – М.: МНЦМО, 2000. – 32 с.

2. Братищев А.В. Математические функции пакета MATLAB: метод. указания / А.В. Братищев. – Ростов н/Д: Издательский центр ДГТУ, 2009. – 14 с.

3. Виленкин Н.Я. Комбинаторика / Н.Я. Виленкин. – М.: Наука, 1969. – 328 с.

4. Голубева О.В. Теоретическая механика / О.В. Голубева. – М.: Высшая школа, 1976. – 352 с.

5. Дэбни Дж. Simulink 4. Секреты мастерства / Дж. Дэбни, Т. Харман. – М: Бином, 2003. – 403 с.

6. Иглин С.П. Обработка массивов данных [Электрон. ресурс]. – Режим доступа: http://www.exponenta.ru.

7. Карпов Ю.Г. Теория автоматов / Ю.Г. Карпов. – СПб.: Питер, 2002. – 224 с.

8. Крылов В.В. Основы теории цепей / В.В. Крылов, С.Я. Корсаков. – М.: Высшая школа, 1990. – 224 с.

9. Лазарев Б. Моделирование процессов и систем в MATLAB / Ю. Лазарев. – СПб.: Питер; Киев, 2005. – 512 с.

10. Мещеряков В.В. Задачи по математике с MATLAB и SIMULINK / В.В. Мещеряков. – М.: ДИАЛОГ-МИФИ, 2007. – 528 с.

11. Мороз А.И. Курс теории систем / А.И. Мороз. – М: Высшая школа, 1987. – 304 с.

12. Нефедов В.Н. Курс дискретной математики / В.Н. Нефедов, В.А. Осипова. – М.: Изд-во МАИ, 1992. – 264 с.

13. Новгородцев А. Расчет электрических цепей в MATLAB. Учебный курс / А. Новгородцев. – СПб.: Питер, 2004. – 250 с.

Оглавление

Введение…… 3

Глава 1. MATLAB……... 5
§ 1.1. Массивы, матрицы, математические операции……. 5
§ 1.2. Программирование в MATLAB…….. 10
§ 1.3. Векторная алгебра и аналитическая геометрия……. 13
§ 1.4. Элементарные функции и операции над ними……... 15
§ 1.5. Графики…….. 19
§ 1.6. Численные методы…… 25
§ 1.7. Обработка экспериментальных данных…… 32

Глава 2. SIMULINK…… 36
§ 2.1. Элементы теории динамических систем…… 36
§ 2.2. Библиотека блоков. Окно модели……... 38
§ 2.3. Логические блоки и конечные автоматы…… 44
§ 2.4. Дискретные системы…… 51
§ 2.5. Непрерывные системы…… 56
§ 2.6. Подсистемы и маскирование……... 62
§ 2.7. Качественный анализ стационарных систем……. 68
§ 2.8. Взаимодействие MATLAB и SIMULINK……. 74

Словарь основных английских терминов……. 82

Литература…… 85