# Первое знакомство с Mathematica
# I Введение.

При запуске программы на экране появляется главное окно, состоящее лишь из строки заголовка и **главного меню** (*File, Edit, Sell, Format,Input,Kernel* и др.). Кроме того, автоматически создаётся новое окно для редактирования пользовательских документов, называемых **блокнотами (notebook)**. Блокнот служит как для ввода команд пользователя, так и для вывода результатов расчётов. Также на экране могут появляется одна или несколько **палитр инструментов**. Палитры служат для визуального ввода команд и различных спецсимволов.

| | |
|---|---|
| Создание, открытие, сохранение блокнота (**Файл *.nb**) | Команды меню *File->new, File->open, File->save* |
| Отображение палитр инструментов* | Команда меню *File->Palettes->название палитры* |

*На начальном этапе для работы нужна лишь палитра **Basic input**.

### 1)Структура блокнота

В блокноте можно выделить 3 типа записей: 1) исполняемые команды, 2) результаты вычислений, 3) заголовки, комментарии и т п.. В соответствии с этим всё содержимое блокнота группируется в **ячейки (Cells)** различного назначения и отображаемые различными **стилями**, что позволяет легко их различать**:

| Стиль(*Format->Style*) | Свойства | Пример |
|---|---|---|
| input | вычисляемые ячейки. Ячейка может содержать несколько команд. | In[45]:= `t = 3 + 2` |
| output | результаты вычислений | Out[45]= 5 |
| title, subtitle, text и.т.п. | комментарии, заголовки и т.п. | ▪ **Вычисление скорости** |

**В принципе, можно изменять соответствия стилей и свойств, т.е., например, сделать ячейку стиля text вычисляемой: Cell->Cell Properties->Cell Evaluatable.

Имеется возможность отображения в верхней части окно для редактирования пользовательских документов стиля текущей ячейки, кнопки Save и некоторых других кнопок. Для этого надо выбрать в меню Edit->Preferences, в открывшемся окне в поле напротив кнопки LookUp ввести: WindowToolbars и нажать кнопку LookUp. Присвоить свойству WindowToolbars значение "EditBar" и нажать Apply.

### 2)Вычисляемые ячейки. Запуск процесса вычисления.

При вводе с клавиатуры по умолчанию создаётся вычисляемая ячейка стиля input. После ввода команды в ячейку для получения результата необходимо одновременно нажать клавиши <Shift> и <Enter>. Для последовательного вычисления всех ячеек в блокноте следует выбрать в главном меню Kernel->Evaluation->Evaluate Notebook.

Результат вычисления **не выводится**, если команда завершается символом “;”.

### 3)Команды и функции инициализации

| Команда | Описание |
|---|---|
| a=b (или Set[a,b]) | присваивание a значения b |
| a:=b (или SetDelayed[a,b]) | присваивание a значения b в невычисленной форме |
| ClearAll[a] | Удаление a |

### 4)Задание собственных функций

In[494]:= `F[x_] = 2x^2;  F[3]`

Out[494]= 18

Пример к пп 3) и 4): сравните 3 варианта

1)
```
In[477]:= ClearAll[F, x];
          x = 4;
          F[x_] = x/2 3 - 1;
          F[8]
          x
Out[480]= 5
Out[481]= 4
```

2)
```
In[467]:= ClearAll[F, x];
          x = 4;
          F[x_] := x/2 3 - 1;
          F[8]
          x
Out[470]= 11
Out[471]= 4
```

3)
```
In[472]:= x = 4;
          F[x_] := x/2 3 - 1;
          ClearAll[F, x];
          F[8]
          x
Out[475]= F[8]
Out[476]= x
```

**5)Ввод операторов с клавиатуры в более „изящной'' форме**

| Оператор | $\frac{\square}{\square}$ | $\square^{\square}$ | $\sqrt{\square}$ | $\square_{\square}$ | $\int F[x]\,dx$ | $\prod$ |
|---|---|---|---|---|---|---|
| Как набрать | [Ctrl + /] | a [ctrl +^] | [ctrl +@] | [ctrl +-] | ESC int ESC F[x] ESC dd ESC x | ESC prod ESC |

**6)Ввод некоторых констант и символов с клавиатуры: ESC <символы> ESC**

| Константа | $e$ =2.7 | $\pi$ =3.14 | $i$ =(-1)^1/2 | $\infty$ | $\alpha$ | $\beta$ | $\gamma$ | $\varphi$ | $\tau$ | $\Theta$ | $\Psi$ | $\psi$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| <символы> | ee | pi | ii | inf | a | **b** | **g** | **j** | **t** | **th** | **Ps** | **ps** |

# II. Массивы и списки

а) прямое задание 1- и 2-мерного массива: **m={1,3,5,r,t}**; **M={{a11,a12} ,{a21,a22}**};
Эквивалентная запись: **m=List[1,3,5,r,t]**; Примечание: запись n={{1,3,5,r,t}}; будет восприниматься как 2-мерный массив с размерностями 1 и 5.

б) Генерация массивов:

**Table[F[x,x2…,xn],{x, *xmin,*xmax, *step*}, {x2,…},{xn,…}]** – создаёт n-мерный массив, заполняемый значениями функции F.

Range[*xmin,x*max,*step*]-генерация списка элементов от xmin до xmax с шагом step.

в) Обращение к элементу массива: **m[[2]]**; **M[[1,2]]**; **n[[1,2]**]. Если набрать M[[1]], то из массива M будет выдана 1-я строка {a11,a12}.

г) функции работы с массивами: **Append[m, elem]**; **Prepend[m, elem]**; **Insert[m, elem,pos];**

добавляют в список m элемент elem соответственно в конец, в начало и в позицию pos. **Delete[m,pos**] удаляет элемент из позиции pos.

**Take[m,{nmin,nmax}]** – создаёт новый список из элементов m с nmin по nmax.

Все перечисленные функции работают и с многомерными массивами.

д) определение размерности массива: **Dimensions[m]**;

е) работа с квадратными матрицами: **DiagonalMatrix[{1,2,3}]** – создаёт матрицу $\begin{pmatrix}1&0&0\\0&2&0\\0&0&3\end{pmatrix}$;

**IdentityMatrix[n]** создаёт единичную nxn матрицу. **Transpose[M**]-возвращает транспонированную по отношению к M матрицу. **Inverse[M]** –вычисляет обратную матрицу. **Det[M]**-вычисляет определитель. **Tr[M]**-вычисляет след.

ж)Умножение матриц: **Dot[M1,M2]**, либо **M1. M2**

з) **Eigenvectors[d], Eigenvalues[d], Eigensystem[d]**- функции для нахождения собственных значений и собственных векторов матричного уравнения **d.x=mx** (m-число, m-вектор-столбец).

**LinearSolve[d, y]**-решает систему линейных уравнений **d.x=y.**

ж) Способы управления выводом массивов на экран: в виде матрицы: **MatrixForm[M]** (либо **M// MatrixForm**); в виде таблицы **TableForm[M]** (либо **M// TableForm**).

# III.Основные функции для обработки символьных выражений

1)Упрощение символьных выражений

**Simplify[expr]** и **FullSimplify[expr]** –приводят выражение expr к наиболее простому виду

**Expand[expr]** – раскрывает все скобки в выражении expr.

**Collect[expr, {x1,x2..}]** – преобразует выражение, группируя слагаемые с одинаковыми степенями x1,x2….

**TrigExpand[expr]** и **TrigReduce[expr]** проводят действия, аналогичные Expand и Collect применительно к тригонометрическим выражениям.

**TrigToExp[expr] –**преобразует в expr все тригонометрические ф-и в экспоненты

**ExpToTrig[expr] –** выполняет действия, обратные TrigToExp[expr].

2)Изменение структуры символьных выражений

**ReplaceAll[F,{x->expr1,y->expr2…}]**- в соответствии с заданными правилами замещает в выражении F x на expr1 y на expr2 и.т.д.. Другая форма записи:

**F/.{x->expr1,y->expr2}.**

**ReplaceRepeated[F,{x->expr1,y->expr2…}]** – итерационно выполняет операцию ReplaceAll до тех пор, пока выражение F не перестаёт меняться. Другая форма записи: **F//.{x->expr1,y->expr2}.**

# IV.Дифференцирование, интегрирование, суммирование

**D[f[x], {x,*n*}]**- находит частную производную n-го порядка от функции f(x): $\frac{\partial^n f(x)}{\partial x^n}$

**Derivative[n1,n2,n3][f][a,b,c]**-вычисляет производную $\frac{\partial^{n1+n2+n3} f}{\partial x1^{n1}\,\partial x2^{n2}\,\partial x^{n3}}$ функции f(x1,x2,x3) в точке {a,b,c}.

**Integrate[f[x],x]** –ищет в символьном виде первообразная ф-и f(x).

**Integrate[f[x],{x,x1,x2}]** – вычисляет интеграл $\int_{x1}^{x2} f(x)\,dx$

**NIntegrate[f[x],{x,x1,p1,p2,…x2}]** – также вычисляет интеграл $\int_{x1}^{x2} f(x)\,dx$, но все операции производит численно; p1,p2…-особые точки f(x).

**Sum[f[i], {i, imin, imax}]** и **NSum[f[i], {i, imin, imax}]** – вычисляют сумму $\sum_{i=imin}^{imax} f(i)$ соответственно в символьном и в численном виде.

**Product[F[i], {i, imin, imax}]** и **NProduct[F[i], {i, imin, imax}]** – выч. произв-е $\prod_{i=imin}^{imax} F(i)$ соответственно в символьном и в численном виде.

# V.Решение уравнений

**Solve[{eqn1,eqn2,…}, {var1,var2,…}]** – решает в символьном виде с-му ур-й eqn1,eqn2,…относительно переменных var1,var2,… Выражения eqn имеют вид leftside = = rightside.

Пример: Solve[X^2+2X==0,X] Результат: {{X→-2},{X→0}}.

**NSolve[{eqn1,eqn2,…}, {var1,var2,…}]** – делает то же, что и Solve, но численно.

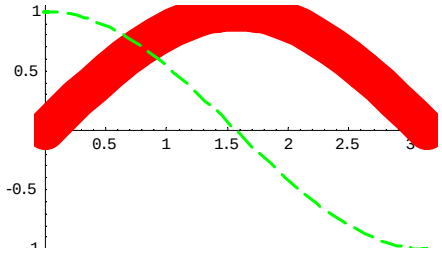
Пример: NSolve[X^2+2X==0,X] Результат: {{X→-2.},{X→0.}}.

**DSolve[{eqn1,eqn2,…}, {F1[var1],F2[var2],…}{var1,var2,…}]** – решает в символьном виде с-му дифференциальных ур-й eqn1,eqn2,…относительно функций F1[var1],F2[var2],… от переменных var1,var2,… Выражения eqn имеют вид leftside==rightside. Необходимо помнить, что результат может содержать 1 или несколько констант интегрирования (обычно они обозначаются C1,C2… или C[1],C[2]…).

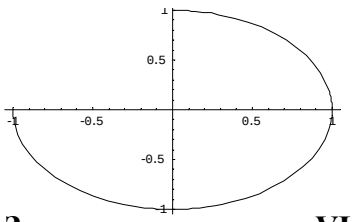
**NDSolve[{eqn1,eqn2,…}, {F1[var1],F2[var2],…}{var1,var2,…}]** – делает то же, что и DSolve, но численно. Необходимо помнить, что результат решения вообще говоря зависит от начальных условий, поэтому выражения eqn должны полностью определять начальные (граничные) условия.

# VI.Построение графиков (примеры)

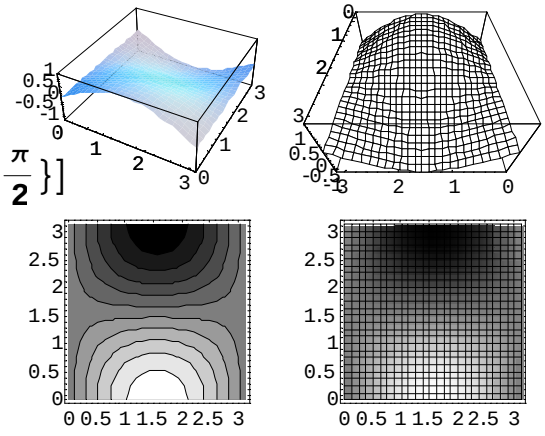
```
In[1]:= Plot[{Sin[x], Cos[x]}, {x, 0, π},
  PlotStyle→{{Hue[0], Thickness[0.1]},
    {RGBColor[0, 1, 0], Thickness[0.01],
     Dashing[{0.05, 0.025}]}}]
```



```
In[2]:= ParametricPlot[{Sin[t], Cos[t]}, {t, 0, 3 π/2}]
```



```
In[26]:= p = Plot3D[Sin[x] Cos[y], {x, 0, π}, {y, 0, π}];
  p1 = ContourPlot[Sin[x] Cos[y], {x, 0, π}, {y, 0, π}];
  p2 = DensityPlot[Sin[x] Cos[y], {x, 0, π}, {y, 0, π}];
In[32]:= Show[GraphicsArray[{{Show[p, Mesh→False], Show[p,
    Mesh→True, Shading→False, ViewPoint→{0, 1, 1},
    BoxRatios→{200, 200, 100}]}, {p1, p2}}]]
```





**Замечания к разделу VI:**

**1)** В некоторых случаях при построении графиков с помощью функции **Evaluate** необходимо указать Mathematica, что необходимо обработать функции, графики которых надо построить, до начала построения графиков. В нижеследующем примере вначале генерируется список функций и только затем вычисляются их значения в точках построения графика. Без оператора Evaluate будет ошибка.

Plot[Evaluate[Table[BesselJ[n, x], {n, 1, 4}]], {x, 0, 10}]

**2)** В случае, если значения функции F являются комплексными, попытка построения её графика приведёт к ошибке. Для визуализации в этом случае можно выделить отдельно действительную и мнимую части функции F, либо её модуль с помощью соответственно функций **Re[F]**, **Im[F]** и **Abs[F]**.

**3)** Для построения графиков по точкам следует использовать функции **ListPlot[{{$x_1$,$y_1$},{$x_2$,$y_2$},…}], ListPlot3D[{{$z_{11}$,$z_{12}$,..},{$z_{21}$,$z_{22}$,..},..}], ListDensityPlot[ {{$z_{11}$,$z_{12}$,..}, {$z_{21}$,$z_{22}$,..},..}], ListContourPlot[{{$z_{11}$,$z_{12}$,..},{$z_{21}$,$z_{22}$,..},..}]**, где $y_i$ –значения отображаемой функции в точках $x_i$ .

# VII.Дополнительные функции для решения уравнений

**Eliminate[{eqn1,eqn2,…}, {var1,var2,…}]** -исключает переменные var1,var2,… из системы уравнений eqn1,eqn2,….

**FindRoot[{eqn1,eqn2,…},, {{var1,value1},{var2,value2}…}] –** ищет численное решение с-мы уравнений, начиная с точки var1=value1, var2=value2,…, где value1, value2,…- некоторые числа.

# VIII.Поиск экстремальных значений

**Min[list]** , **Max[list]** –возвращают соответственно значение минимального либо максимального элемента в списке list.

**FindMinimum[F[x,y,…],{x,x0},{y,y0},…] –** производит поиск локального минимума функции F[x,y,…] в окрестности точки x=x0,y=y0,…,где x0, y0,… - некоторые числа.

Замечание : хотя с помощью функции **ConstrainedMin** возможен поиск глобального минимума, однако данная функция корректно работает лишь в ограниченном числе случаев, поэтому её использование не рекомендуется. В более поздней версии Mathematica 5 для поиска глобальных экстремумов появляются более совершенные функции **Minimize,Maximize,NMinimize,NMaximize**.

# IX.Работа с файлами

1) Простейшие функции:

Запись в файл:

а) **Put[expr1,expr2,expr3,… "file.ext"]** (упрощённый вариант **expr>>"filename"**); – создаёт , либо перезаписывает файл file.ext помещая в него выражения expr1,expr2….

б) **PutAppend[expr1,expr2,expr3,… "file.ext"]** (упрощённый вариант **expr1>>>"filename"**); **Save["filename",exr2,expr3,…]** –добавляют expr1,expr2… в конец файла file.ext (при отсутствии файла он создаётся). Различие между этими функциями поясняет пример:

```
In[1]:= a[b_, c_] = 2 b c; M = {1, 2}; M >> "G:\1.dat"; a >>> "G:\1.dat"; Save["G:\2.dat", M, a];
```

| Имя файла | G:\1.dat | G:\2.dat |
|---|---|---|
| Содержимое файла | {1, 2}<br>a | M = {1, 2}<br>a[b_, c_] = 2*b*c |

Чтение из файла:

в) **expr=Get["file.ext"]** (упрощённый вариант **expr=<<"filename"**)-считывает последнее записанное в файл выражение

г) **ReadList["file.ext",*{type1,type2,…},n*]** – считывает содержимое файла и возвращает в виде списка. Указав необязательные параметры ***,{type1,type2,…}*** и *n*, можно считывать только n объектов, причём соответствующих указанным типам.

2) Функции Export и Import

**Export["file.ext", expr] -** сохраняет expr в файл file.ext в формате, соответствующем расширению ext.

**Export["file.ext ", expr, "format"] -** сохраняет expr в файл file.ext в формате "format".

**expr =Import["file.ext"] -** загружает expr из файла file.ext, считая что формат файла соответствует расширению ext.

**expr =Import["file.ext "format"] -** загружает expr из файла file.ext, считая что формат файла "format".

"format"-одна из специальных строк, соответствующая одному из многочисленных поддерживаемых Mathematica форматов ввода-вывода текста, графики, звука.

Пример: In[28]:= `P = Plot[Sin[x], {x, 0, 6}]; Export["G:\1.jpg", P]`

В результате график сохраняется в jpg-файле.

3) Функции ToString и ToExpression

**ToString[expr] –** возвращает текстовую строку, соответствующую значению expr;

**ToExpession[text] –** интерпретирует текстовую строку как строку ввода;

Пример использования:

Пусть Dcm=3. тогда команда data>>"G:\MyFile[Dcm="<>ToString[Dcm]<>"].dat"; сохранит значение data в файл с именем "MyFile[Dcm=3].dat" , а команда ToExpression["data1=<<G:\MyFile[Dcm="<>ToString[Dcm]<>"].dat" ]; присвоит переменной data1 значение, загруженное из этого файла.

# X. Основы программирования

**1)** Использование структуры Module.

**Module[{var1, var2,…},expr1; expr2;…. ;exprn]** – выполняет последовательность действий expr1; expr2;…, считая переменные var1, var2,… локальными, т.е. все изменения этих переменных в процессе вычисления expr1; expr2;…. ;exprn; отменяются после выполнения всей последовательности действий. Если после exprn не стоит ";", то Module возвращает результат выполнения exprn, если же стоит, то ничего не возвращает.

In[29]:= `a = 1; b = 2; Module[{a}, a = b + 1; b = a;] ; {a, b}`

Out[29]= {1, 3}

(значение b изменяется, а значение a –нет)

( см также пример в п.п.3) )

**2)** Циклы Do.

**Do[expr, {imax}]** – вычисляет expr imax раз. Пример

In[2]:= `z = 1 ; Do[z = F[z], {5}] ; z`

Out[2]= $F(F(F(F(F(1)))))$

**Do[expr, {i, imin, imax, *step*}]** – вычисляет expr с переменной i, последовательно принимающей значения от imin до imax с шагом step (параметр step можно не указывать, тогда шаг будет равен1).

In[12]:= `k = 1; z = {0, 0, 0, 0, 0}; Do[z[[k]] = i; k++, {i, 2, 3, 0.3}] ; z`

Out[12]= {2, 2.3, 2.6, 2.9, 0}

**3)** Циклы For

**For[start, test, incr, body] –** сначала вычисляется выр-е start, затем – body и incr до тех пор, пока справедливо условие test. Пример: функция, вычисляющая факториал

In[1]:= `MyFactorial[n_] := Module[{i}, For[i = 1; k = 1, i < n + 1, i++, k = k * i]; k]`

In[2]:= `MyFactorial[4]` Out[2]= 24

start test incr body

**4)** Условные переходы.

**If[condition, t]** или **If[condition, t, f, u]** – проверяет условие condition (имеет вид a>b,a<b, либо a = = b), если оно истинно(true), то выполняется последовательность действий **t**, если ложно (false), то **f**, если же результат проверки condition отличен от true и false, то выполняется **u**.

Замечание: используя структуру If можно организовать условное прерывание выполнения циклов Do и For при помощи команды **Return[expr]** (expr – возвращаемое значение). Также для этой цели можно использовать команду Goto

**5)** Безусловные переходы. Организуются с помощью команды **Goto[tag];** и метки **Label[tag];.** Пример: In[8]:= `i = 3; Goto[3 + 2]; i = 2; Label[1 + 4]; i` Out[1]= 3

**6)** Функции-переключатели.

**Which[condition$_1$, expr$_1$, condition$_2$, expr$_2$,…]** - выполняет команды expr$_i$, первому в порядке следования истинному условию condition$_i$.

**Switch[expr, $expr_1$, $todo_1$, $expr_2$, $todo_2$,… ]** – последовательно сравнивает значение выражения expr с $expr_1$,$expr_2$,…до первого равного true результата сравнения, при этом выполняется соответствующая последовательность операций $todo_i$.