

Лекция 11

14. Графики

График явно заданных функций в декартовых координатах (Explicit function)

Для построения графиков есть команды **plot2d()** и **wxplot2d()**.

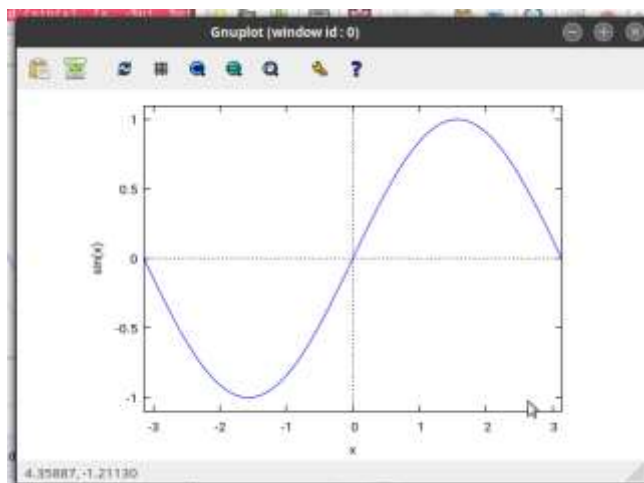
Первая строит график в отдельном окне, вторая – встраивает в лист вычислений.

Построим график функции $y=\sin(x)$ на отрезке $x \in [-\pi, +\pi]$

```
plot2d (sin(x), [x, -%pi, %pi])$
```

Интервал изменения ординаты программа выбирает сама, исходя из минимальных и максимальных значений функции. Этот интервал можно задать и самому.

```
plot2d (sin(x), [x, -%pi, %pi], [y,-1, 1])$
```



Для построения на одном чертеже нескольких графиков исходные функции записывают через запятую в квадратных скобках.

Построим графики функции $y_1=x^2-2$ и $y_2=x$ на отрезке $x \in [-3, +3]$.

(первый вариант)

```
plot2d ([x^2-2, x],[x, -3, 3],[y,-3, 8])$
```

или (второй вариант)

```
y1:x^2-2;
```

```
y2:x;
```

```
plot2d ([y1, y2],[x, -3, 3],[y,-3, 8])$
```

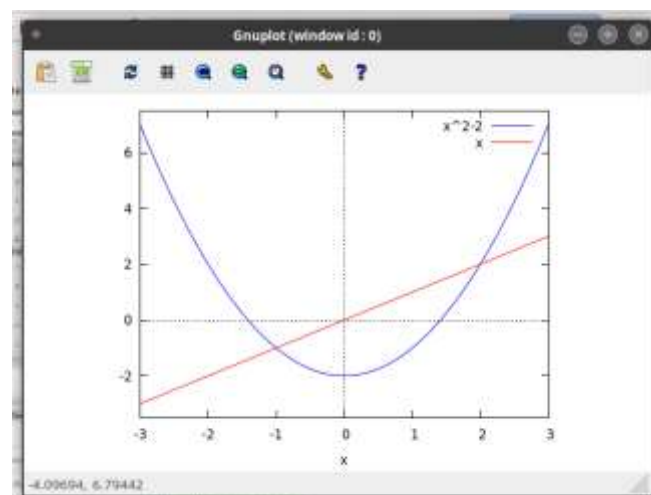
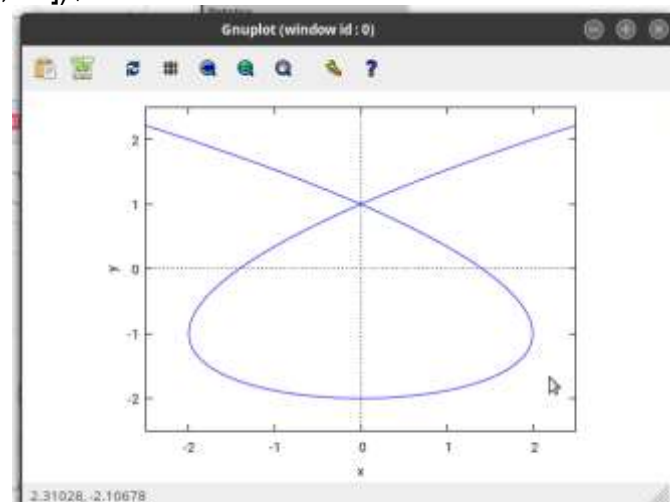


График неявно заданной функции (mplicit function)

Построим график функции $x^2 - y^3 + 3*y = 2$

```
plot2d (x^2-y^3+3*y=2, [x,-2.5,2.5], [y,-2.5,2.5])$
```



Графики параметрически заданных функций (Parametric function).

Если функция задана в параметрическом виде, используется опция `parametric`.

Построим график функции $x(t) = \sin t$ и $y(t) = -2\cos(4t) + e^{\cos(t)}\sin(t/12)^5$ в интервале изменения параметра $t \in [-\pi, +\pi]$:

```
plot2d ([parametric, sin(t)*((-2*cos(4*t))+%e*cos(t)*sin(t/12)^5),[t,-%pi, %pi], [nticks,100]])$
```

Параметр `[nticks]` задает количество точек, по которым строится график. Чем больше это значение, тем более гладким будет построенная кривая, но при этом увеличивается время, необходимое для ее построения.

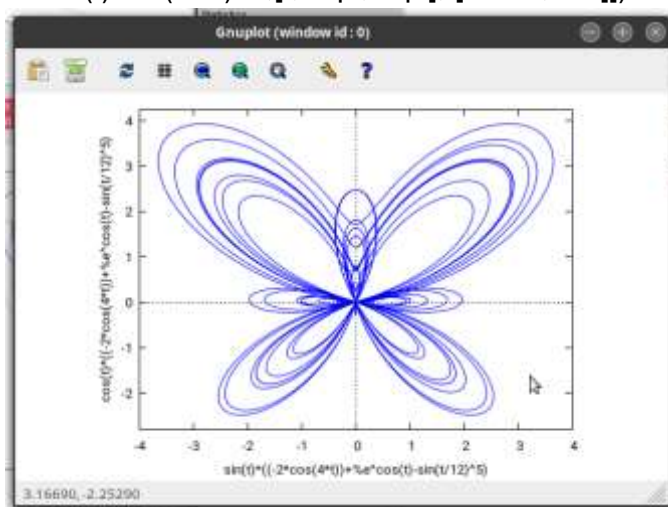


График контурных линий (Contour lines)

```
plot2d ([contour, u^3 + v^2], [u, -4, 4], [v, -4, 4])$
```

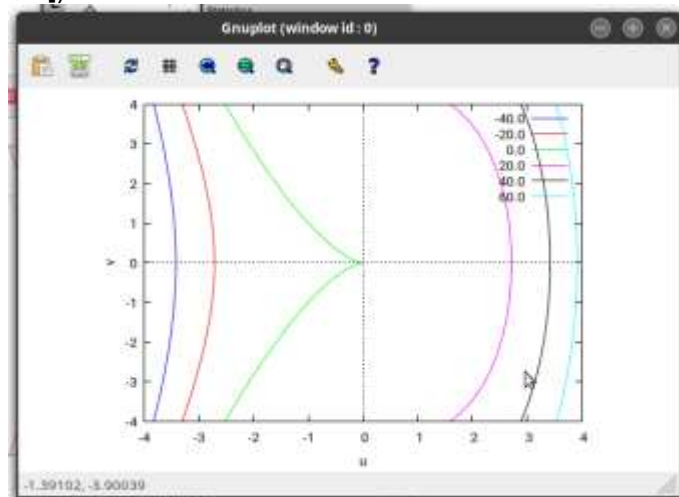
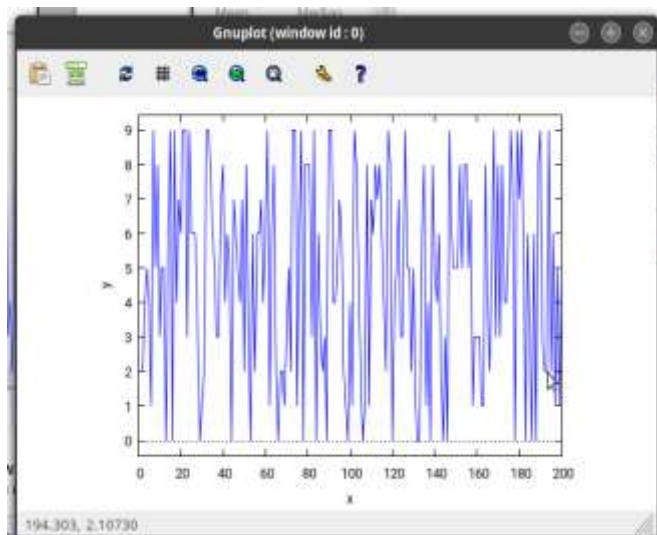


График из 200 случайных чисел от 0 до 9 (A plot of 200 random numbers 0 ... 9)

Графики дискретных множеств:

```
plot2d ([discrete, makelist ( random(10), 200)])$
```

`makelist` – список кандидатов



Построение поверхностей

Построение явно заданных поверхностей. Для построения трехмерной поверхности функции двух аргументов есть команды `plot3d()` и `wxplot3d()`.

Если для построения использовалась функция `plot3d()`, то поверхность можно изучить с разных сторон, вращая его с помощью мышки.

Формат задания функции: Function: **plot3d**

`plot3d(expr,x_range,y_range, ...,options, ...)`

`plot3d([expr_1, ...,expr_n],x_range,y_range, ...,options, ...)`

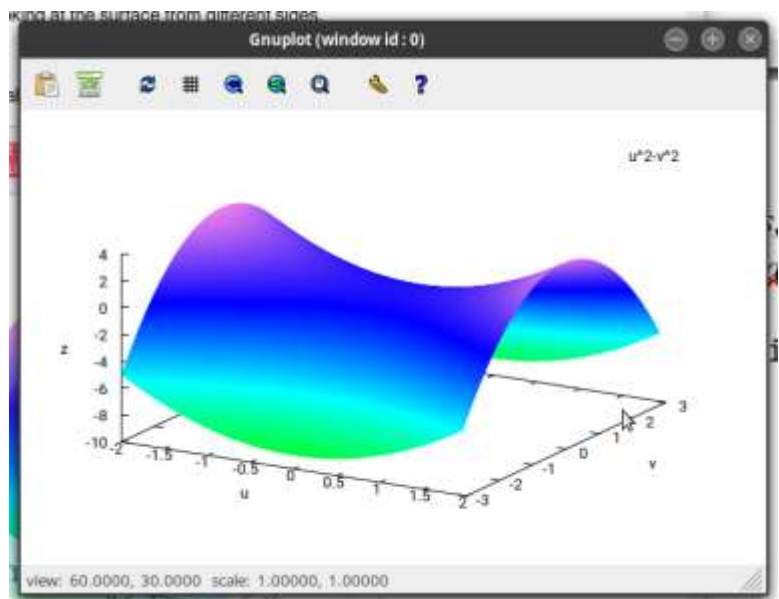
Построим график функции $z = x^2 - y^2$ на прямоугольнике $x \in [-2, +2]$, $y \in [-3, +3]$:

Объемный 3D график

`plot3d (u^2 - v^2, [u, -2, 2], [v, -3, 3], [grid, 100, 100], nomesh_lines)$`

`grid, 100, 100` – сетка

`nomesh_lines` – линии Номеша



Возможны построения графиков с изменением палитры цветов

Построим график функции $\ln(x^2 \cdot y^2)$ с изменением палитры и цветовой полосой, связывающей цвета в квадрате $x \in [-2, +2]$, $y \in [-2, +2]$:

`plot3d (log(x^2*y^2), [x, -2, 2], [y, -2, 2], [grid, 29, 29],
[palette, [gradient, red, orange, yellow, green]],
color_bar, [xtics, 1], [ytics, 1], [ztics, 4], [color_bar_tics, 4])$`

`palette` – палитра

`gradient` – градиентный

`red` – красный

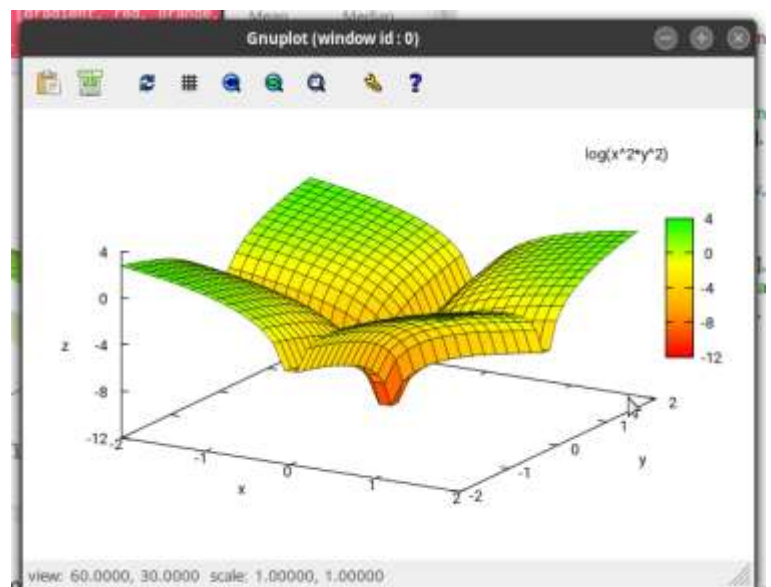
`orange` – оранжевый

`yellow` – желтый

`green` – зеленый

`color_bar` – цвет столбиков

`color_bar_tics` – цветные полосы



Построение сферических фигур

Построим график сферы

```
plot3d ( 5, [theta,0,%pi], [phi,0,2*%pi], same_xyz, nolegend, [transform_xy, spherical_to_xyz],
[mesh_lines_color,blue], [palette,[gradient,"#1b1b4e", "#8c8cf8"]])$
```

plot3d – график 3d

theta – тетта

phi – фи

nolegend – нет легенды

transform – преобразование

spherical – специфический

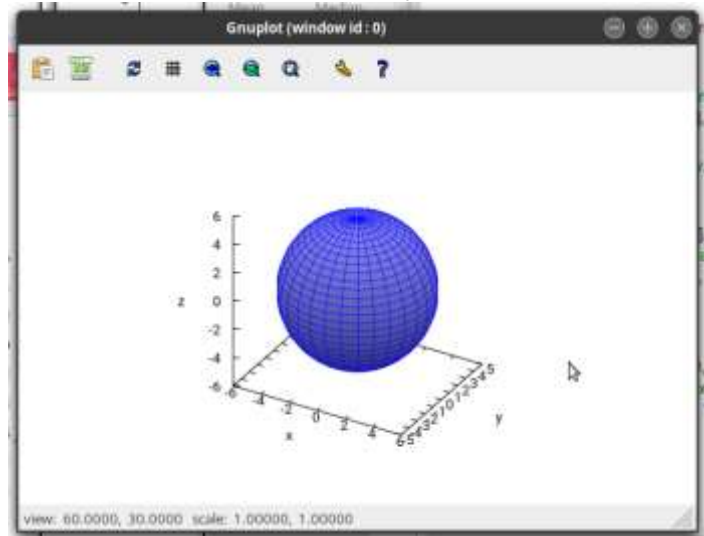
mesh_lines – линии сетки

palette – палитра

gradient – градиентный

#1b1b4e,

#8c8cf8 – оттенки цветов



Построение параметрически заданной поверхности.

Если поверхность задана параметрически (от двух параметров x и y), зависимости $\text{expr}_1(x, y)$, $\text{expr}_2(x, y)$ и $\text{expr}_3(x, y)$ следует писать в квадратных скобках через запятую.

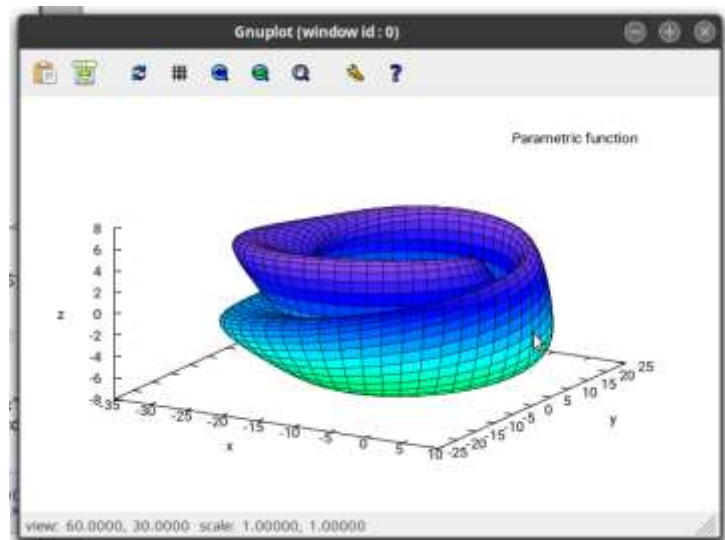
Построим параметрический график трех уравнений в квадрате $x \in [-35, +10]$, $y \in [-25, +25]$:

```
expr_1: 5*cos(x)*(cos(x/2)*cos(y)+sin(x/2)*sin(2*y)+3)-10$
```

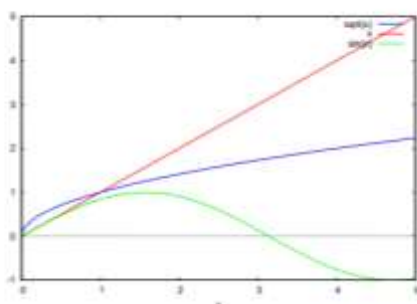
```
expr_2: -5*sin(x)*(cos(x/2)*cos(y)+sin(x/2)*sin(2*y)+3)$
```

```
expr_3: 5*(sin(x/2)*cos(y)+cos(x/2)*sin(2*y))$
```

```
plot3d ([expr_1, expr_2, expr_3], [x, -%pi, %pi],[y, -%pi, %pi], [grid, 50, 50])$
```



```
--> wxplot2d([sqrt(x), x, sin(x)], [x, 0, 5], [y, -1, 5])$
```



Построение нескольких графиков в одном окне.

Y1: sqrt(x);

Y2:x;

Y3:sin(x);

```
wxplot2d ([sqrt(x),x,sin(x)], [x, 0, 5], [y,-1, 5])$
```

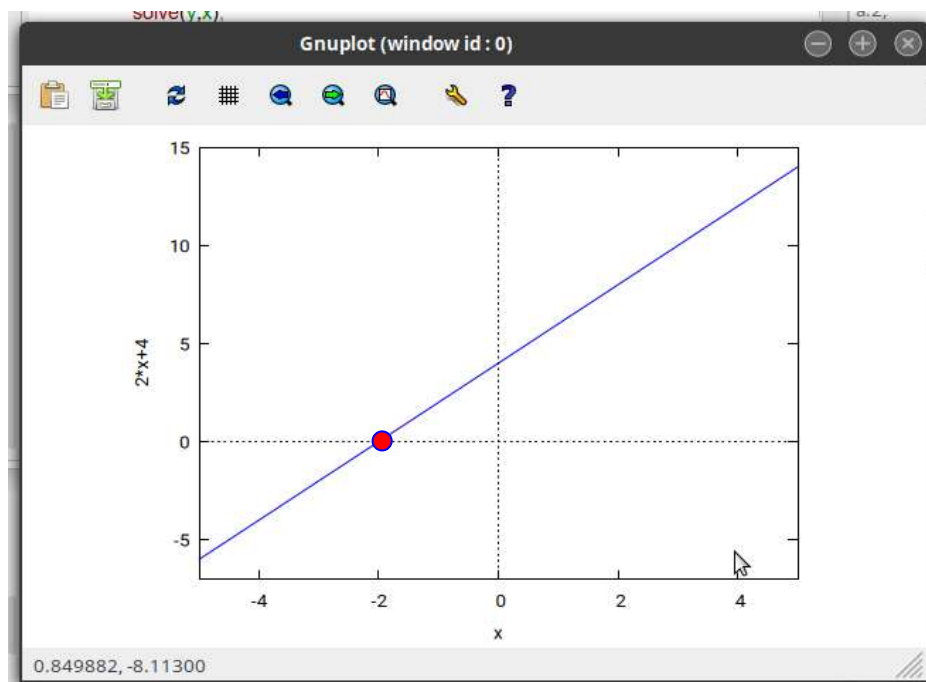
```
или plot2d ([y1,y2,y3], [x, 0, 5], [y,-1, 5])$
```

15. Решение Уравнений в Maxima

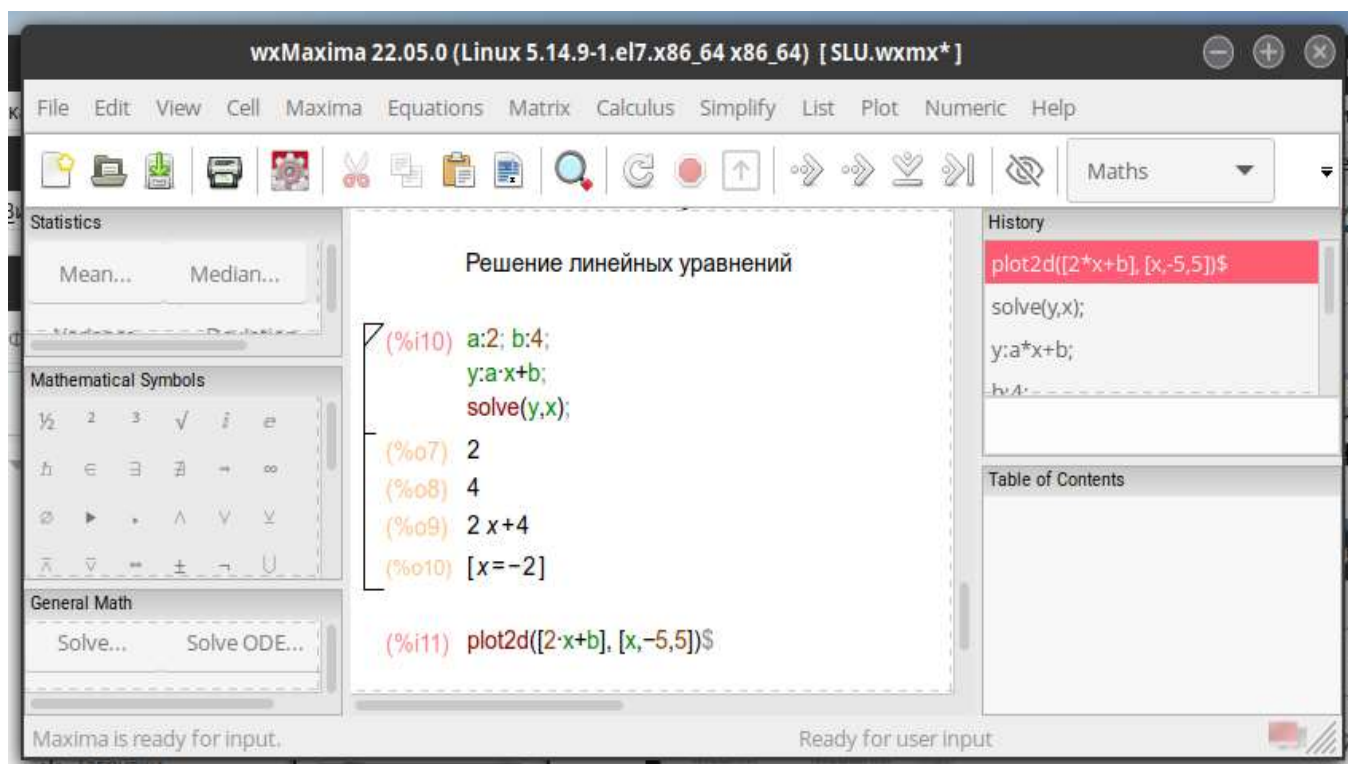
15.1 Решение линейных уравнений (ЛУ)

Уравнение вида $a \cdot x + b = 0$ является линейным уравнением. И при заданных параметрах a и b его решение сводится к нахождению корня по формуле $x = -b/a$.

Для нахождения корня уравнения достаточно построить график уравнения и на нем определить корень путем пересечения линии графика с осью x .



В системе Maxima уравнение решается с помощью функции `solve(y,x)`



15.2 Решение систем линейных уравнений (СЛУ)

Векторные и матричные операторы и функции системы Maxima позволяют решать задачи линейной алгебры.

Система линейных алгебраических уравнений (СЛАУ) в общем случае имеет вид:

$$y = \begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n \end{cases}$$

где $a_{11} a_{12} \dots a_{nn}$ – коэффициенты уравнений, а b_1, \dots, b_n – свободные члены уравнений и x_1, x_2, \dots, x_n , корни, которые требуется найти.

Система может иметь:

- Одно решение,
- Иметь много решений,
- Не иметь решения

Существуют разные способы решения СЛУ. В системе Maxima:

- метод Крамера, с нахождением определителей системы (главного и определителей, в которых заменены последовательно столбцы свободными членами);
- метод Гаусса, основанный на эквивалентных преобразованиях системы (поменять местами строки системы или к строчке прибавить или вычесть другую, умноженную на число);
- с помощью функции `linsolve()`.

Рассмотрим решение СЛАУ на примере трех уравнений с тремя неизвестными вида:

$$y = \begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases}$$

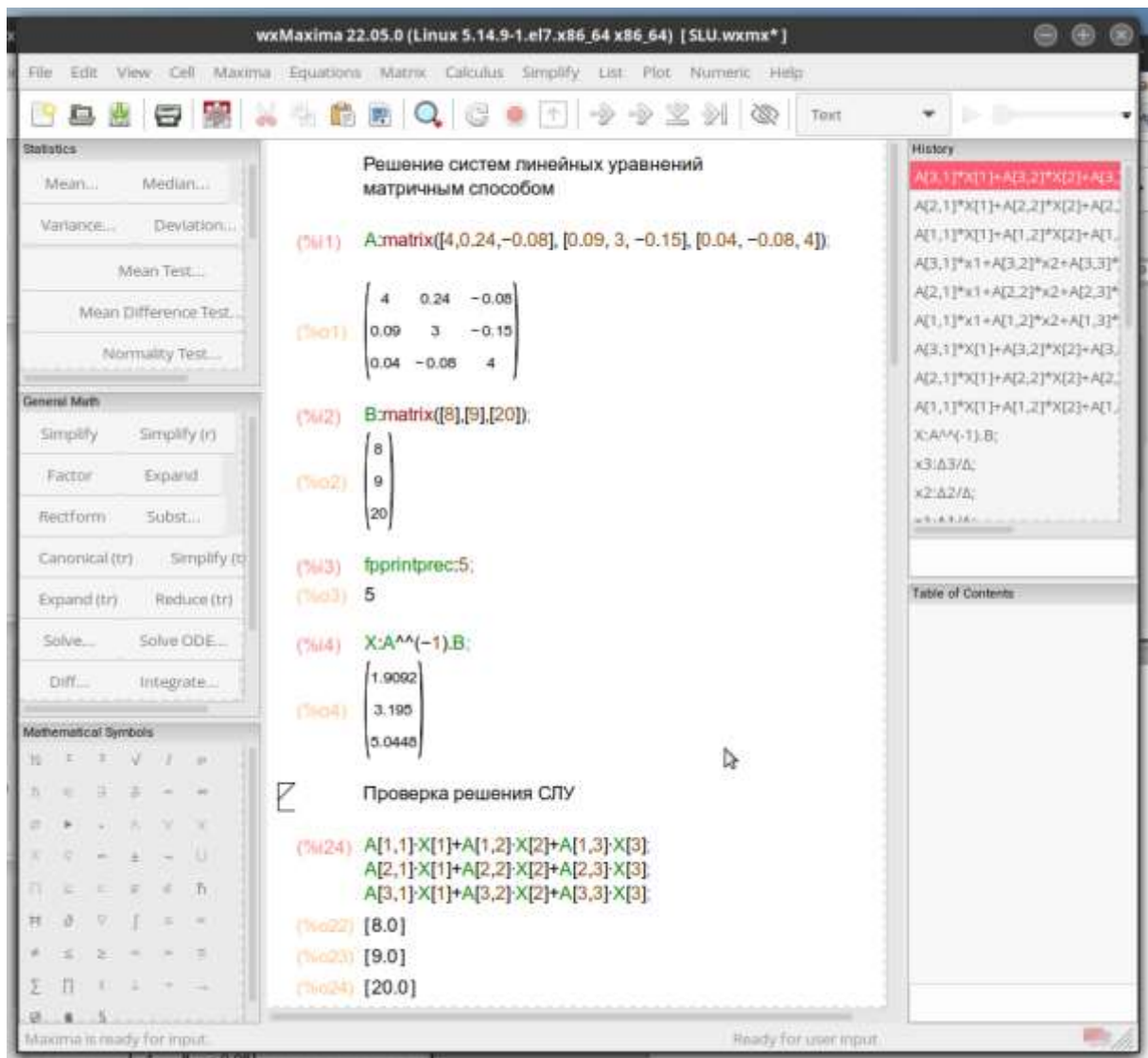
Метод Гаусса

Для решения системы линейных уравнений методом Гаусса необходимо выполнить следующие действия:

1. Задать матрицу A коэффициентов системы;
2. Задать вектор B свободных членов системы;
3. Вывести вектор X с помощью обратной матрицы $X = A^{(-1)} \cdot B$;
4. Сделать проверку решения СЛУ.

Рассмотрим пример решения системы линейных уравнений в Maxima:

$$\begin{cases} 4x_1 + 0.24x_2 - 0.08x_3 = 8 \\ 0.09x_1 + 3x_2 - 0.15x_3 = 9 \\ 0.04x_1 - 0.08x_2 + 4x_3 = 20 \end{cases}$$



Метод Крамера

1. Задается матрица коэффициентов и вычисляется главный определитель системы:

$$\Delta = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} - a_{13}a_{22}a_{31}$$

2. Вычисляются определители Δ_1 , Δ_2 и Δ_3 , составленные путем замены элементов соответствующего столбца матрицы A – свободными членами:

$$\Delta_1 = \begin{vmatrix} b_1 & a_{12} & a_{13} \\ b_2 & a_{22} & a_{23} \\ b_3 & a_{32} & a_{33} \end{vmatrix} \quad \Delta_2 = \begin{vmatrix} a_{11} & b_1 & a_{13} \\ a_{21} & b_2 & a_{23} \\ a_{31} & b_3 & a_{33} \end{vmatrix} \quad \Delta_3 = \begin{vmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ a_{31} & a_{32} & b_3 \end{vmatrix}$$

3. Если главный определитель $\Delta \neq 0$, рассчитываются корни по формулам:

$$x_1 = \frac{\Delta_1}{\Delta} \quad x_2 = \frac{\Delta_2}{\Delta} \quad x_3 = \frac{\Delta_3}{\Delta}$$

4. Сделать проверку решения СЛУ.

Рассмотрим пример решения системы линейных уравнений в Maxima:

$$\begin{cases} 4x_1 + 0.24x_2 - 0.08x_3 = 8 \\ 0.09x_1 + 3x_2 - 0.15x_3 = 9 \\ 0.04x_1 - 0.08x_2 + 4x_3 = 20 \end{cases}$$

wxMaxima 22.05.0 (Linux 5.14.9-1.el7.x86_64 x86_64) [SLU.wxmx*]

File Edit View Cell Maxima Equations Matrix Calculus Simplify List Plot Numeric Help

Maths

Statistics

Mean... Median...

Variance... Deviation...

Mean Test...

Mean Difference Test...

Normality Test...

Linear Regression...

General Math

Simplify Simplify (r)

Factor Expand

Rectform Subst...

Canonical (tr) Simplify (t)

Expand (tr) Reduce (tr)

Solve... Solve ODE...

Diff... Integrate...

Limit... Series...

Mathematical Symbols

Statistics

Mean... Median...

General Math

Simplify Simplify (r)

Mathematical Symbols

Method of Cramer

(%i5) $\Delta := \text{determinant}(A), \text{numer};$

(%o5) 47.874

(%i7) $\Delta x := \text{matrix}([8, 0.24, -0.08], [9, 3, -0.15], [20, -0.08, 4]);$

(%o7) $\begin{pmatrix} 8 & 0.24 & -0.08 \\ 9 & 3 & -0.15 \\ 20 & -0.08 & 4 \end{pmatrix}$

(%i8) $\Delta y := \text{matrix}([4, 8, -0.08], [0.09, 9, -0.15], [0.04, 20, 4]);$

(%o8) $\begin{pmatrix} 4 & 8 & -0.08 \\ 0.09 & 9 & -0.15 \\ 0.04 & 20 & 4 \end{pmatrix}$

(%i6) $\Delta z := \text{matrix}([4, 0.24, 8], [0.09, 3, 9], [0.04, -0.08, 20]);$

(%o6) $\begin{pmatrix} 4 & 0.24 & 8 \\ 0.09 & 3 & 9 \\ 0.04 & -0.08 & 20 \end{pmatrix}$

(%i9) $\Delta 1 := \text{determinant}(\Delta x), \text{numer};$

(%o9) 91.402

(%i10) $\Delta 2 := \text{determinant}(\Delta y), \text{numer};$

(%o10) 152.96

(%i11) $\Delta 3 := \text{determinant}(\Delta z), \text{numer};$

(%o11) 241.52

(%i12) $x1 := \Delta 1 / \Delta;$

(%o12) 1.9092

(%i13) $x2 := \Delta 2 / \Delta;$

(%o13) 3.195

(%i14) $x3 := \Delta 3 / \Delta;$

(%o14) 5.0448

Проверка решения СЛУ

(%i21) $A[1,1] \cdot x1 + A[1,2] \cdot x2 + A[1,3] \cdot x3;$

(%o21) 8.0

(%i20) $A[2,1] \cdot x1 + A[2,2] \cdot x2 + A[2,3] \cdot x3;$

(%o20) 9.0

(%i21) $A[3,1] \cdot x1 + A[3,2] \cdot x2 + A[3,3] \cdot x3;$

(%o21) 20.0

History

$A[3,1] \cdot x[1] + A[3,2] \cdot x[2] + A[3,3] \cdot x[3];$

$A[2,1] \cdot x[1] + A[2,2] \cdot x[2] + A[2,3] \cdot x[3];$

$A[1,1] \cdot x[1] + A[1,2] \cdot x[2] + A[1,3] \cdot x[3];$

$A[3,1] \cdot x1 + A[3,2] \cdot x2 + A[3,3] \cdot x3;$

$A[2,1] \cdot x1 + A[2,2] \cdot x2 + A[2,3] \cdot x3;$

$A[1,1] \cdot x1 + A[1,2] \cdot x2 + A[1,3] \cdot x3;$

$A[3,1] \cdot x[1] + A[3,2] \cdot x[2] + A[3,3] \cdot x[3];$

$A[2,1] \cdot x[1] + A[2,2] \cdot x[2] + A[2,3] \cdot x[3];$

$A[1,1] \cdot x[1] + A[1,2] \cdot x[2] + A[1,3] \cdot x[3];$

$X := A^{-1} \cdot B;$

$x3 := \Delta 3 / \Delta;$

$x2 := \Delta 2 / \Delta;$

$x1 := \Delta 1 / \Delta;$

$\Delta 3 := \text{determinant}(\Delta z), \text{numer};$

$\Delta 2 := \text{determinant}(\Delta y), \text{numer};$

Table of Contents

History

$A[3,1] \cdot x[1] + A[3,2] \cdot x[2] + A[3,3] \cdot x[3];$

$A[2,1] \cdot x[1] + A[2,2] \cdot x[2] + A[2,3] \cdot x[3];$

Table of Contents

Maxima is ready for input.

Ready for user input

Использование функции `linsolve()`

Решение СЛУ в Maxima реализуется и с помощью функции `linsolve()`. Она имеет особенность, заключающуюся в том, что аргументы функции должны быть представлены в виде списка данных, разделенных запятой и помещенных в квадратные скобки []. Синтаксис функции:

$$\text{linsolve}([u1, u2, u3], [x1, x2, x3]),$$

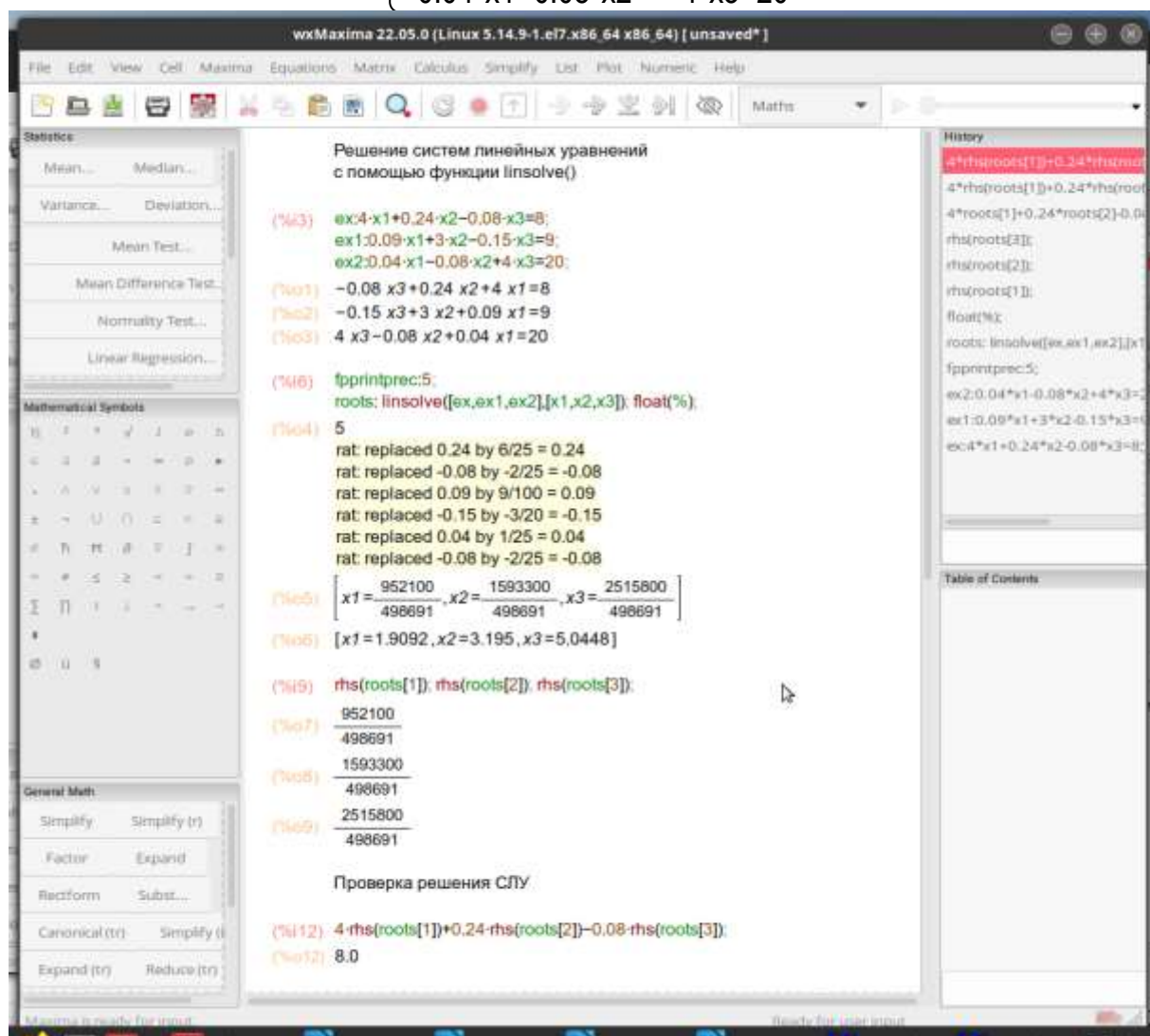
где $u1, u2, u3$ – заданные уравнения, $x1, x2, x3$ – неизвестные, которые надо найти.

Для решения СЛУ с помощью функции `linsolve()` необходимо:

1. Задать три линейных уравнения $u1, u2, u3$ с тремя неизвестными, с заданными коэффициентами и свободными членами;
2. С помощью функции `linsolve([u1, u2, u3], [x1, x2, x3])` вычислить корни;
3. Сделать проверку решения СЛУ путем подстановки корней в уравнение.

Рассмотрим пример решения системы линейных уравнений в Maxima:

$$\begin{cases} 4x_1 + 0.24x_2 - 0.08x_3 = 8 \\ 0.09x_1 + 3x_2 - 0.15x_3 = 9 \\ 0.04x_1 - 0.08x_2 + 4x_3 = 20 \end{cases}$$



Корни выдаются в виде списка. Числовое значение одного из корней может быть получено при использовании функции `rhs(roots[1])` и ссылкой на порядковый номер корня. Функция `rhs()` – возвращает правую часть (то есть второй аргумент) выражения.

15.3. Решение нелинейных уравнений

На практике возникает задача нахождения корней нелинейного уравнения (НУ) вида

$$F(x)=0$$

или системы нелинейных уравнений (СНУ) с одним неизвестным x .

$$\begin{cases} F1(x) = 0, \\ F2(x) = 0. \end{cases}$$

Очевидно, что корни системы можно найти из условия $F1(x)=F2(x)$.

Или последнее выражение можно свести к виду $F1(x) - F2(x)=0$.

Таким образом, задача решения СНУ сводится к нахождению значений аргумента x функции $F(x)=F1(x) - F2(x)$ одной переменной, при котором значение функции $F(x) = 0$.

В отличие от решения СЛУ использование прямых вычислений далеко не всегда возможно и решение находится зачастую с использованием итерационных методов, т.е. находится приближенное решение \hat{x} , удовлетворяющее при заданной погрешности вычисления $\delta > 0$ условию $\hat{x} - x \vee < \delta$.

Многие уравнения, такие как трансцендентные, и системы нелинейных уравнений СНУ не имеют аналитических решений. Они могут решаться численными методами с заданной погрешностью ($\ll 0.0001$).

Поиск корня с помощью функции solve()

Для решения уравнения вида $F(x)=0$ предназначена функция:

$$\text{solve}(F, x)$$

$$\text{solve}(F1, F2, \dots, x1, x2, \dots).$$

где в качестве аргументов используются: одно уравнение F (НУ) или список уравнений $F1, F2$ (СНУ), а также один корень x (НУ) или список неизвестных $x1, x2, \dots$ (СНУ), если это система уравнений.

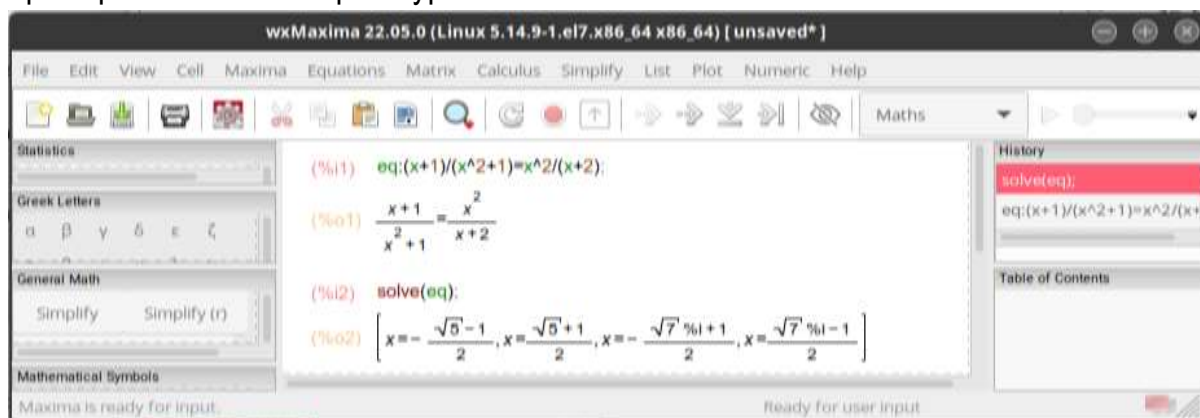
Функция $\text{solve}()$ ищет один корень в окрестности его существования, которое должно быть задано и имеет аналитическое решение.

Функция $\text{solve}()$ может решать как одно уравнение, так и систему уравнений. Если корней много, то функция возвращает список, состоящий из всех корней заданного уравнения. Если, аналитического решения нет, то надо использовать другие подходы для вычисления корней – численные методы.

Для нахождения окрестности существования корней можно построить график.

Причем функция $\text{solve}()$ находит не только действительные корни, но и все комплексные корни и записывает их в виде списка элементов.

Пример вычисление корней уравнения:



Поиск корня с помощью функции *allroots()*

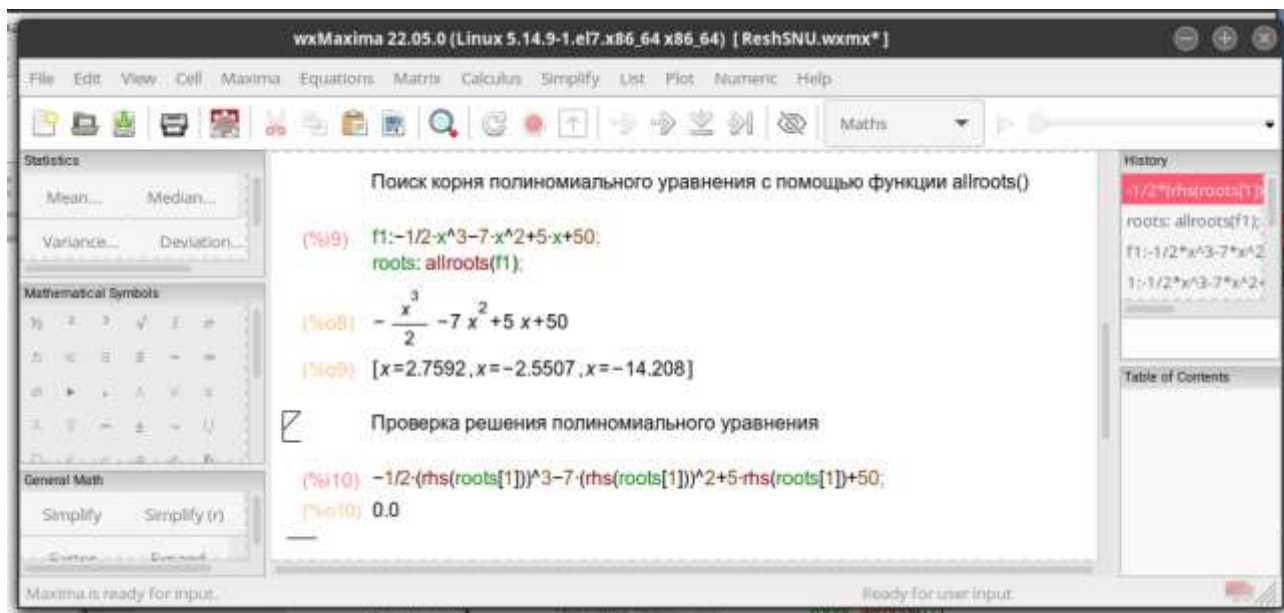
Для уравнений вида полином $F = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \dots + a_1 \cdot x^1 + a_0$ не всегда возможно найти все окрестности корней по графику т.к. возможны комплексные корни, которые на графике не отражаются, поэтому для поиска корней полинома степени n предназначена функция:

$$allroots(F).$$

Корни полинома F могут быть как вещественными, так и комплексными числами.

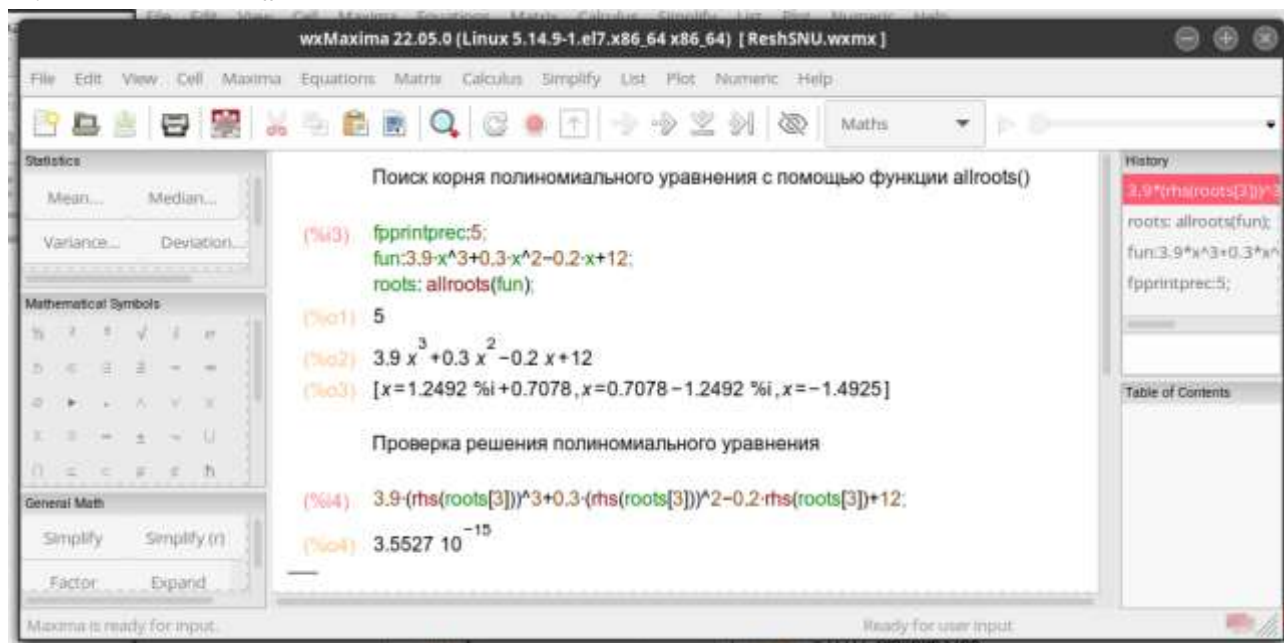
Функция *allroots()* ищет сразу все корни полинома.

Рассмотрим пример нахождения корней полинома третьей степени:



Как видно все корни вещественные и при подстановке корней обращают функцию $F=0$.

Кроме того, рассмотрим пример решения полиномиального уравнения, у которого функция *allroots()* находит мнимые корни.



Проверка решения уравнения относительно 3-го корня дает с учетом погрешности $< 3.55 \cdot 10^{-15}$ весьма точный результат, т.е. найдено решение при $F \rightarrow 0$.

Поиск корня с помощью функции *find_root()*

Имеется множество нелинейных уравнений произвольной формы и СНУ, которые не имеют аналитических решений методом прямых вычислений. Тогда они могут решаться только численными методами с заданной погрешностью.

В этом случае, для нахождения численного значения произвольной функции F , применяется функция:

$$\text{find_root}(F, x, x_0, x_n),$$

Этой функции надо указать отрезок $x_0..x_n$, на котором расположен корень уравнения x . Если на отрезке $x_0 - x_n$ несколько корней, то все окрестности существования корней можно выявить, построив график функции $F(x)$ и убедиться, что на задаваемом отрезке только один корень.

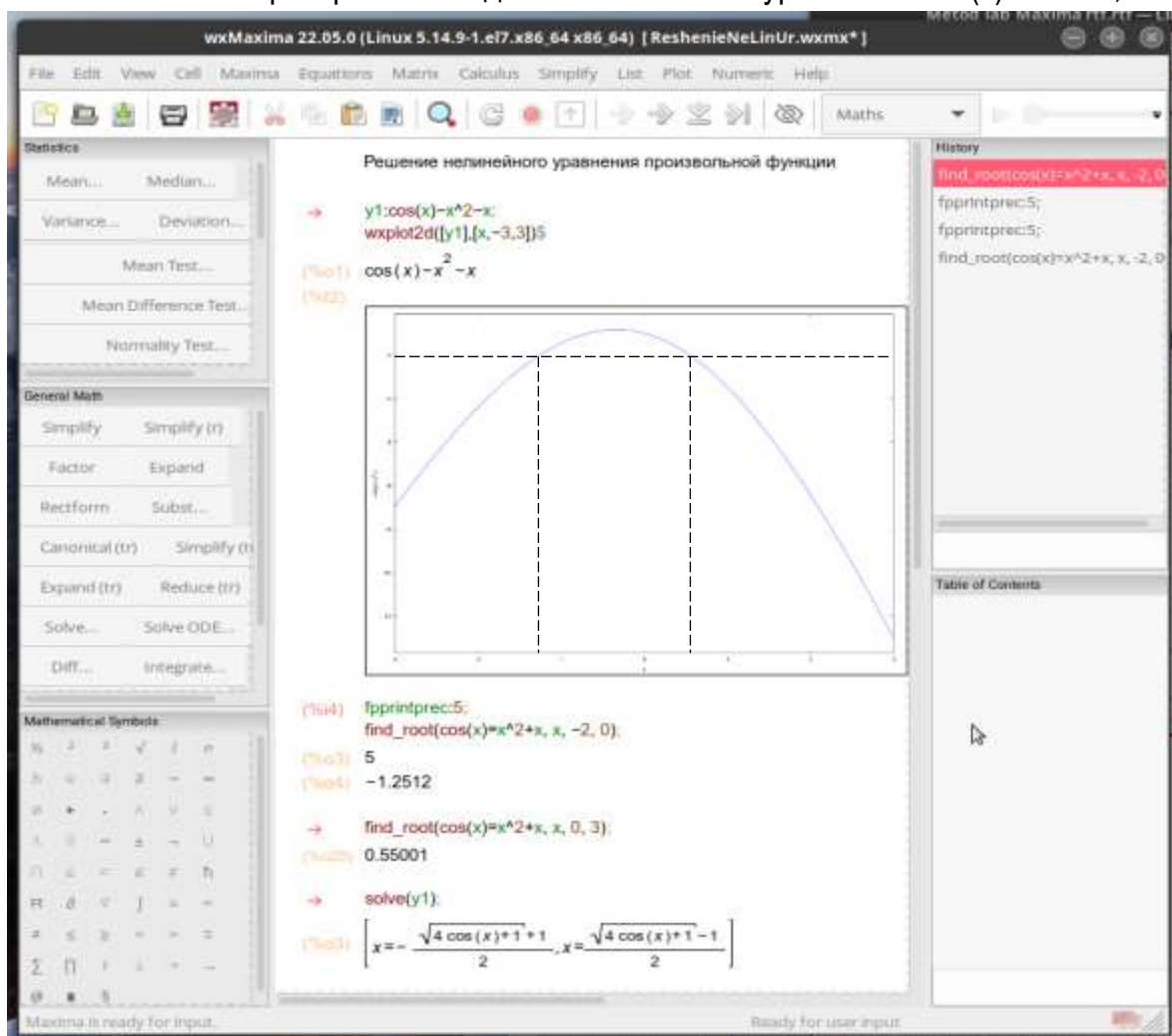
Для решения НУ необходимо выполнить следующие действия:

1. задать функцию $f1$,
2. построить график этой функций;
3. вывести решение с помощью функции $\text{find_root}(f1, [x, x_0, x_n])$.

Если надо найти корни у нескольких уравнений, например, двух, то для решения НУ в командной строке необходимо:

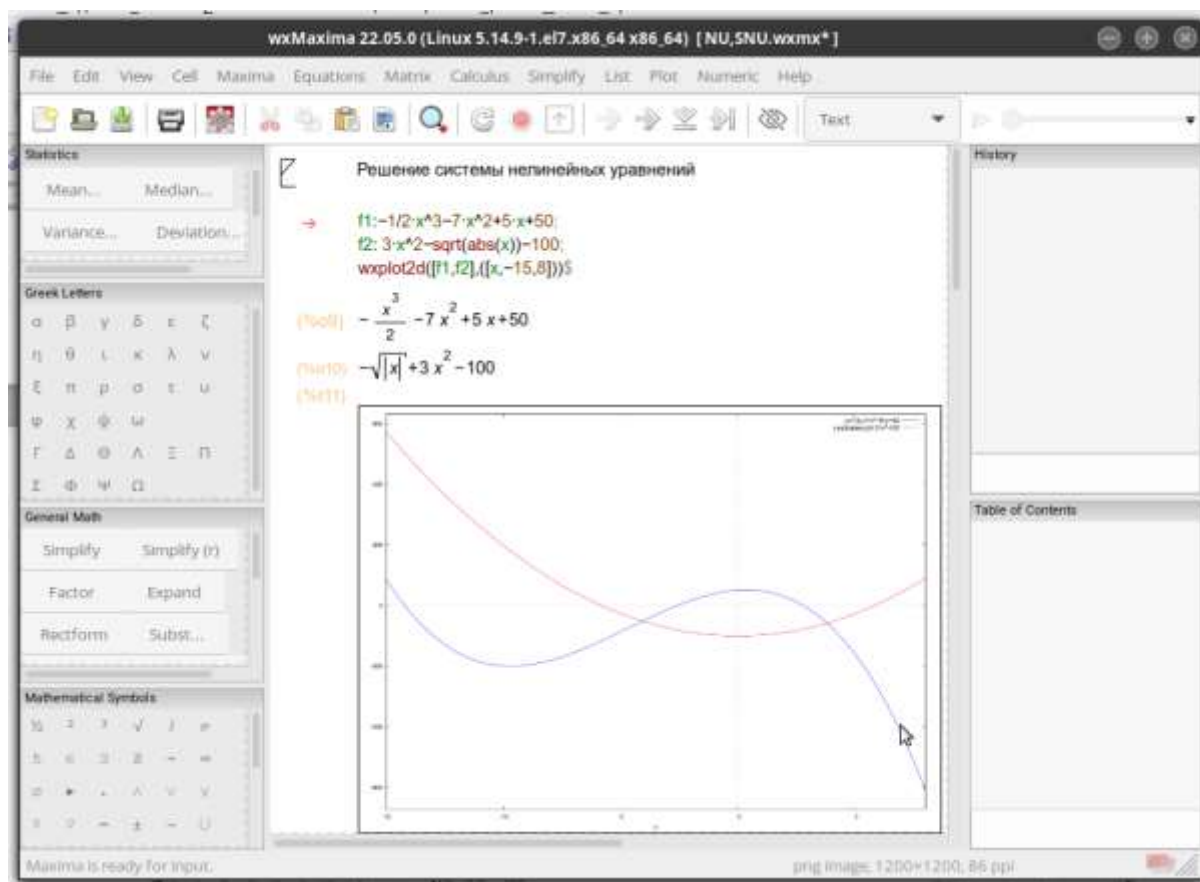
1. задать функции $f1$ и $f2$,
2. построить графики этих функций;
3. вывести решение с помощью функции $\text{find_root}(f1, [x, x_0, x_n])$, $\text{find_root}(f2, [x, x_0, x_n])$.

I. Рассмотрим решения одного нелинейного уравнения $\cos(x) - x^2 - x = 0$;



После нахождения решения рекомендуется сделать проверку подстановкой найденных корней в уравнения. Если в результате решения левая часть не равна 0, а например, 10^{-9} ..., то это значение и есть погрешность вычисления.

II. Решение системы НУ – f_1 и f_2 .



Система нелинейных уравнений СНУ можно решить с помощью той же функции `solve(f1-f2, x)`, у которой аргументом функции является разница функций $f_1 - f_2 = 0$.

Но если система *Maxima* напишет комментарий, что аналитического решения нет, то для поиска корней произвольной функции используется функция `find_root()`. Но этой функции надо указать отрезок, на котором расположен корень уравнения. Если на этом отрезке несколько корней, то необходимо выявить все отрезки на которых располагается лишь один корень и поочередно вычислить точное значение (возможно с погрешностью).

Проанализируем графики функций, приведенных на рисунке.

- Первая функция f_1 пересекает ось x – три раза в окрестностях точек -14, -3, +3.
- Вторая функция f_2 пересекает ось x – в окрестности точек -5 и +5.
- А система функций имеет корни в окрестности точек -4 и +4.

Осталось их только уточнить с помощью функции `find_root()`.

Для получения уточненных значений корней необходимо:

1. задать сами функции f_1 и f_2 ,
2. построить их графики,
3. выявить по графикам окрестности существования корней,
4. использовать функции `find_root()` для вывода численных значения корней в выявленных окрестностях.
5. Сделать проверку решения путем подстановки корней в уравнение или СНУ.

Примеры результатов расчета корней НУ и СЛУ с помощью функции *find_root()*

wxMaxima 22.05.0 (Linux 5.14.9-1.el7.x86_64 x86_64) [SNU.wxmx*]

File Edit View Cell Maxima Equations Matrix Calculus Simplify List Plot Numeric Help

Statistics

Mean... Med...
Variance... De...
Mean Te...
Mean Differ...
Normality

Greek Letters

α β γ δ ϵ
 ζ η θ ι κ
 λ ν ξ π ρ
 σ τ υ φ χ
 ψ ω
 Γ Δ Θ Λ Ξ
 Π Σ Φ Ψ Ω

General Math

Simplify Simp...
Factor Exp...
Rectform Sub...
Canonical (tr)

Численное решение нелинейных уравнений с помощью функции *find_root()*

Нахождение корней первого уравнения $f_1=0$. Согласно рисунку первый корень находится в диапазоне -15..-10, второй - в диапазоне -5..0, третий - в диапазоне 0..5

```
(%i9) fpprintprec:5; find_root(f1,x,-15,-5);
```

```
(%o8) 5
```

```
(%o9) -14.208
```

```
(%i10) find_root(f1,x,-5,0);
```

```
(%o10) -2.5507
```

```
(%i11) find_root(f1,x,0,5);
```

```
(%o11) 2.7592
```

Нахождение корней второго уравнения $f_2=0$. Согласно рисунку первый корень находится в диапазоне -10..0, второй - в диапазоне 0..5.

```
(%i12) find_root(f2,x,-10,0);
```

```
(%o12) -5.8429
```

```
(%i13) find_root(f2,x,0,10);
```

```
(%o13) 5.8429
```

Нахождение корней системы уравнений $f_1-f_2=0$. Согласно рисунку первый корень СЛУ находится в диапазоне -7..-2, второй - в диапазоне 2..6

```
(%i14) find_root(f1-f2,x,-7,-2);
```

```
(%o14) -4.0656
```

```
(%i15) find_root(f1-f2,x,2,6);
```

```
(%o15) 3.7904
```

History

```
find_root(f1-f2,x,2,6);
```

```
find_root(f1-f2,x,-7,-2);
```

```
find_root(f2,x,0,10);
```

```
find_root(f2,x,-10,0);
```

```
find_root(f1,x,0,5);
```

```
find_root(f1,x,-5,0);
```

```
find_root(f1,x,-15,-5);
```

```
fpprintprec:5;
```

```
find_root(f1,x,-5,0);
```

```
fpprintprec:5; find_root(f1,
```

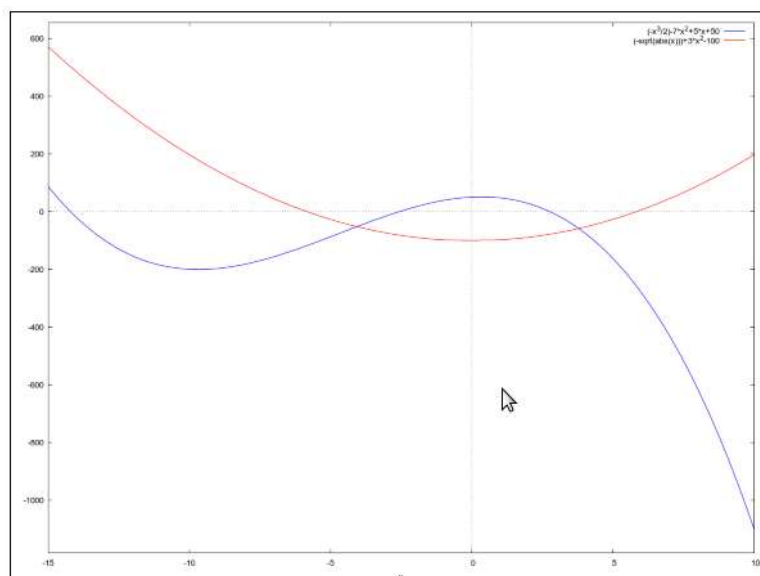
```
fpprintprec:5; find_root(f1,
```

```
fpprintprec:5; find_root(f1,
```

```
wxplot2d([f1,f2],[x,-15,10])
```

Table of Contents

Maxima is ready for input. Ready for user input



Проверка решения НУ и СНУ

The screenshot displays the wxMaxima 22.05.0 interface. The main window shows the solution of a system of nonlinear equations (НУ) and a system of nonlinear equations (СНУ). The left sidebar contains various toolbars for statistics, mathematical symbols, and general math operations. The right sidebar shows the history and table of contents.

Решение НУ $f_1=0$. Оно имеет 3 корня R1, R2, R3

```
(%i8) R1: find_root(f1,x,-15,-5);
(%o8) -14.20846316093738

Проверка 1 корня  $f_1=0$ 

(%i34) -1/2 (R1)^3-7 (R1)^2+5 (R1)+50;
(%o34) 4.263256414560601 10-14

(%i10) R2: find_root(f1,x,-5,0);
(%o10) -2.550749225446046

Проверка 2 корня  $f_1=0$ 

(%i22) -1/2 (R2)^3-7 (R2)^2+5 (R2)+50;
(%o22) 0.0

(%i12) R3: find_root(f1,x,0,5);
(%o12) 2.759212386383427

Проверка 3 корня  $f_1=0$ 

(%i35) -1/2 (R3)^3-7 (R3)^2+5 (R3)+50;
(%o35) 0.0
```

Решение НУ $f_2=0$. Оно имеет 2 корня R4, R5

```
(%i17) R4: find_root(f2,x,-10,-5);
(%o17) -5.842864646203678

Проверка 1 корня  $f_2=0$ 

(%i36) 3 (R4)^2-sqrt(abs(R4))-100;
(%o36) 1.4210854715202 10-14

(%i19) R5: find_root(f2,x,5,10);
(%o19) 5.842864646203678

Проверка 2 корня НУ  $f_2=0$ 

(%i37) 3 (R5)^2-sqrt(abs(R5))-100;
(%o37) 1.4210854715202 10-14
```

Решение СНУ $f_1-f_2=0$. Она имеет 2 корня R6, R7

```
(%i23) R6: find_root(f1-f2,x,-10,0);
(%o23) -4.065561825108673

Проверка 1 корня СНУ  $f_1-f_2=0$ 

(%i41) -1/2 (R6)^3-7 (R6)^2+5 (R6)+50-3 (R6)^2+sqrt(abs(R6))+100;
(%o41) 1.4210854715202 10-14

(%i25) R7: find_root(f1-f2,x,0,5);
(%o25) 3.790388974082077

Проверка 2 корня СНУ  $f_1-f_2=0$ 

(%i42) -1/2 (R7)^3-7 (R7)^2+5 (R7)+50-3 (R7)^2+sqrt(abs(R7))+100;
(%o42) 1.4210854715202 10-14
```

Как видно из результатов расчета численными методами проверка решения корней НУ и СНУ методом подстановки дает хороший результат. Максимальная погрешность расчета составляет порядок 10^{-14} , что близко к 0 и удовлетворяет условию решения $F=0$.

16. Программирование в Maxima

Maxima – интерпретатор. Вызывать интерпретатор Maxima из консоли не очень удобно поэтому на Maxima устанавливается оболочка wxMaxima (open-source)

Программирование в Maxima выполняется на встроенном макроязыке. Для решения задачи надо создать несколько вычисляемых областей.

Сложные алгоритмы обработки можно создавать с помощью функций пользователя, в которых можно использовать средства программирования. Весь алгоритм можно оформить как одну функцию, внутри которой применяются средства программирования.

Для создания программ можно использовать операторы, имеющиеся в системе:

| | |
|-----------------|--|
| if | Оператор условия. |
| then | Оператор то. |
| else | Оператор иначе. |
| for | Оператор цикла регулярного (для). |
| thru | Оператор достижения конечного значения (через). |
| while | Оператор пока выполняется условие. |
| unless | Оператор пока не будет достигнуто условие (если только). |
| do | Оператор делать (перечисление действий в теле цикла). |
| step | Оператор шаг. |
| next | Оператор следующий |
| break | Оператор останова. |
| continue | Оператор продолжения. |
| return | Оператор возврата произвольного значения. |

Ветвление. Оператор создания условных выражений **if**

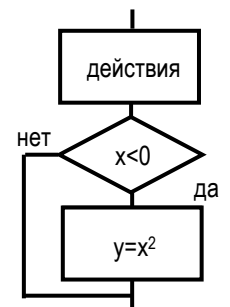
Позволяет выбрать один из двух возможных вариантов, и предназначен для организации разветвления в программе.

Задается в виде:

if <условие> **then** <выражение>

Если условие выполняется, то возвращается значение выражения.

Например, **if** $x < 0$ **then** x^2
возвращается x^2 , если $x < 0$



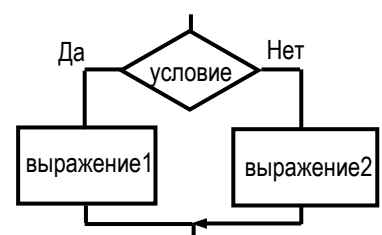
Совместно с этим оператором часто используются операторы иного выбора **else** и прерывания **break**.

Оператор **else** (“иначе”)

Обычно используется с оператором **if** в следующей конструкции:

if <условие> **then** <выражение1> **else** <выражение2>

Если условие выполняется (true – истина), то возвращается значение выражения1, а иначе значение выражения2 (условие false – ложно).

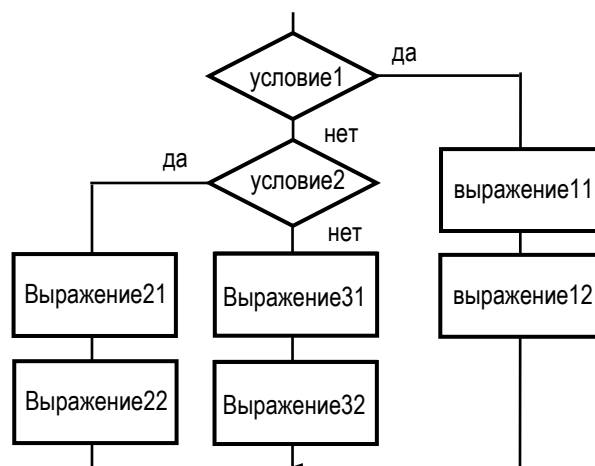


Система **Maxima** позволяет использовать различные формы оператора **if**,
 Например, **if** <условие1> **then** <выражение1> **elseif** <условие2> **then** <выражение2>
else <выражение3>

Если выполняется <условие1>, то выполняется <выражение1>, иначе – проверяется <условие2>, и, если оно истинно — выполняется <выражение2>, а иначе – выполняется <выражение3>.

```

if <условие> then
(
    Блок инструкций 1;
)
elseif <условие2> then
(
    Блок инструкций 2;
)
else
{
    Блок инструкций 3;
}
    
```



В общем случае Блок инструкций 1 = выражение11, выражение 12...

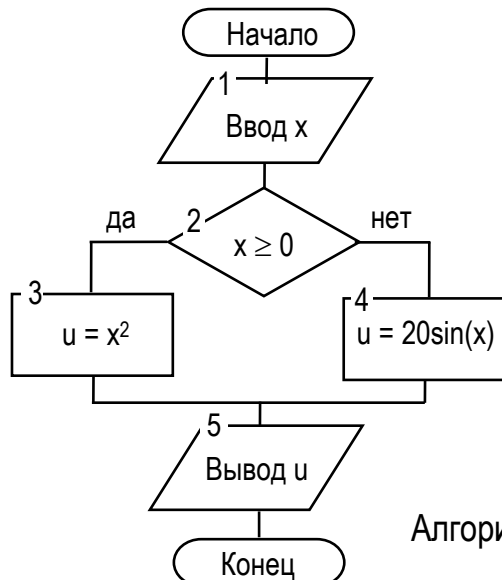
В случае, если в какой-либо ветви необходимо выполнить одно выражение, то блочная инструкция в виде круглых скобок () в этой ветви не нужна.

Степень вложенности может быть сколь угодно большой:

Например, **if** <условие1> **then** <выражение1> **elseif** <условие2> **then** <выражение2>
elseif ... **else** <выражение0>

Если выполняется <условие1>, то выполняется <выражение1>, иначе – проверяется <условие2>, и если оно истинно – выполняется <выражение2>, и т.д.
 Если ни одно из условий не является истинным – выполняется <выражение0>.

Задача1: Вычислить заданную функцию, $u(x) = \begin{cases} x^2, & \text{при } x \geq 0 \\ 20 * \sin(x), & \text{при } x < 0 \end{cases}$



Алгоритмическая структура решения задачи

Построим график функции $u(x)$

`wxplot2d([if x≥0 then x^2 else 20*sin(x)], [x,-5,5])$`

Plot 2D

Expression(s): Special

Variable: From: To: ☐ logscale

Variable: From: To: ☐ logscale

Ticks:

Format:

Options:

File:

wxMaxima 22.05.0 (Linux 5.14.9-1.el7.x86_64 x86_64) [Prg wxmx*]

File Edit View Cell Maxima Equations Matrix Calculus Simplify List Plot Numeric Help

Statistics

- Mean... Median...
- Variance... Deviation...
- Mean Test...
- Mean Difference Test...
- Normality Test...
- Linear Regression...

Mathematical Symbols

General Math

- Expand (tr) Reduce
- Solve... Solve ODE
- Diff... Integrate
- Limit... Series...
- Plot 2D... Plot 3D...

Операторы условных выражений

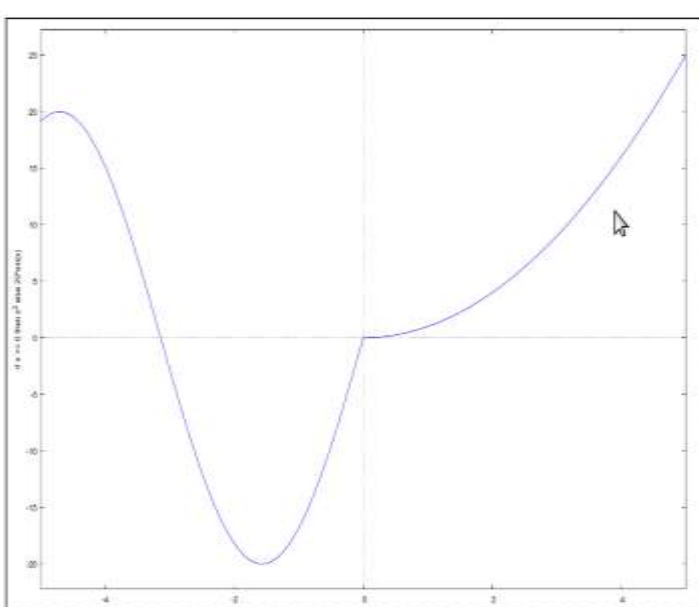
Проверка ветки "Да"

```
(%i3) x:3$
fx: if x>0 then x^2 else 20*sin(x);
(%o3) 9
```

Проверка ветки "Нет"

```
(%i6) x:-5$
fx: if x>0 then x^2 else 20*sin(x);
fx, number;
(%o5) -20 sin(5)
(%o6) 19.17848549326277
```

```
(%i1) wxplot2d([if x>0 then x^2 else 20*sin(x)], [x,-5,5])$
(%t1)
```



History

```
fx, number;
fx: if x>0 then x^2
x:-5$
fx: if x>0 then x^2
x:3$
wxplot2d([if x>0 t
```

Table of Contents

Maxima is ready for input. Ready for user input.

Операторы цикла

Алгоритмы решения многих задач являются циклическими, т.е. для достижения результата определенная последовательность действий должна быть выполнена несколько раз. Циклом называется группа инструкций, повторяющихся многократно с разными данными. Например, программа контроля знаний. Для циклов применяются операторы: **for - thru**, **while**, **unless**, **do**.

Оператор **for** с параметром

Служит для организации циклов с заданным числом повторения (используется для описания регулярных циклических процессов). В Maxima используется инкрементный вариант.

В цикле задаются – начальное значение индекса (инициализация), шаг (**step**), конечное значение индекса и тело цикла.

Для выполнения итераций используется оператор **do**. При этом могут использоваться три варианта его вызова, отличающиеся **условием окончания** цикла:

1. **for** <переменная>:< начальное значение, шаг> **thru** <конечное значение >
 do <операторы тела цикла>
2. **for** <переменная>:<начальное значение, шаг> **while** <условие> **do**
 <операторы тела цикла>
3. **for** <переменная>:<начальное значение, шаг> **unless** <условие> **do**
 <операторы тела цикла>

Шаг по умолчанию равен 1

Ключевые слова **thru**, **while**, **unless** указывают на способ завершения цикла:

1. по достижении переменной цикла значения конечное (до предела);
2. пока выполняется <условие>;
3. пока не будет достигнуто <условие>.

Все параметры могут быть произвольными выражениями. Контрольная переменная по завершении цикла предполагается положительной (при этом начальное значение может быть и отрицательным). Выражения вычисляются на каждом шаге цикла и проверяются условия завершения, поэтому их сложность влияет на время выполнения цикла.

При нормальном завершении цикла возвращаемая величина **done**.

Принудительный выход из цикла осуществляется при помощи оператора **return**, который может возвращать произвольное значение.

Контрольная переменная цикла — локальная внутри цикла, поэтому её изменение в цикле не влияет на контекст (даже при наличии вне цикла переменной с тем же именем).

for i:io thru ik step 2 do <тело цикла>;

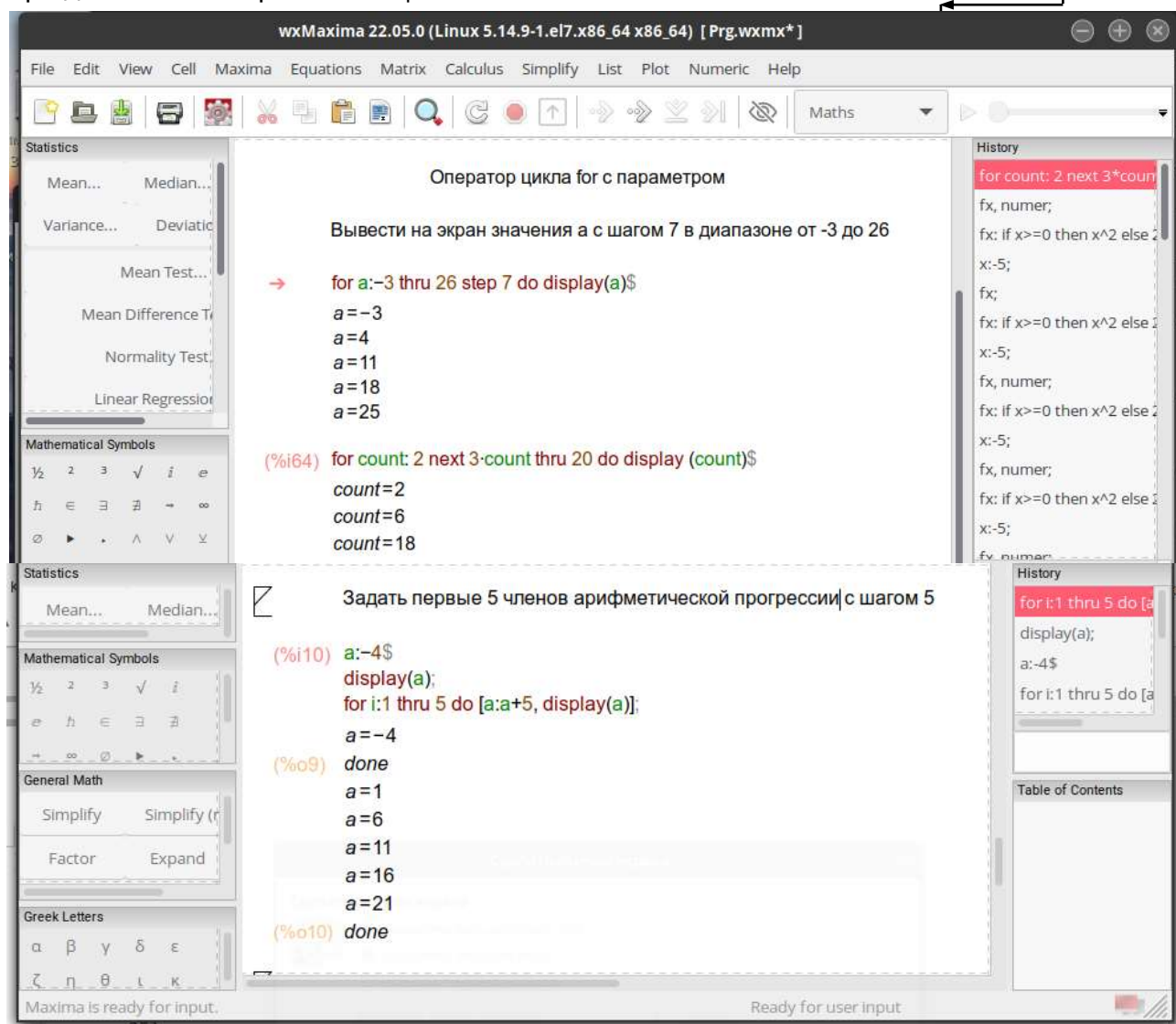
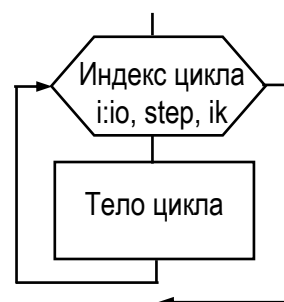
i – переменная цикла,

io – начальное значение,

ik – конечное значение,

step – шаг (по умолчанию равен 1).

thru указывает, что завершение цикла происходит при достижении переменной цикла значения ik



Если цикл предназначен для повторного выполнения программного блока (несколько действий в теле цикла), то его формат:

for i:io step 2 thru ik do

(

<Инструкции тела цикла>,

)

Примеры работы операторов цикла

for i:1 thru 10 do

(block[i], i:3, return(i));

return(10);

При нормальном завершении цикла на экран возвращается **done**. Принудительный выход из цикла осуществляется при помощи оператора **return**, который может возвращать произвольное значение.

Задача2(s,p). Пример программного вычисления заданной конечной суммы s тригонометрического ряда

$$s = \sum_{n=1}^5 a \cdot \cos(n \cdot x)$$

Накопление суммы происходит постепенно:

$s_0 = 0$ // начальное значение

$s_1 = s_0 + a \cdot \cos(1 \cdot x)$

$s_2 = s_1 + a \cdot \cos(2 \cdot x)$

$s_3 = s_2 + a \cdot \cos(3 \cdot x)$

$s_4 = s_3 + a \cdot \cos(4 \cdot x)$

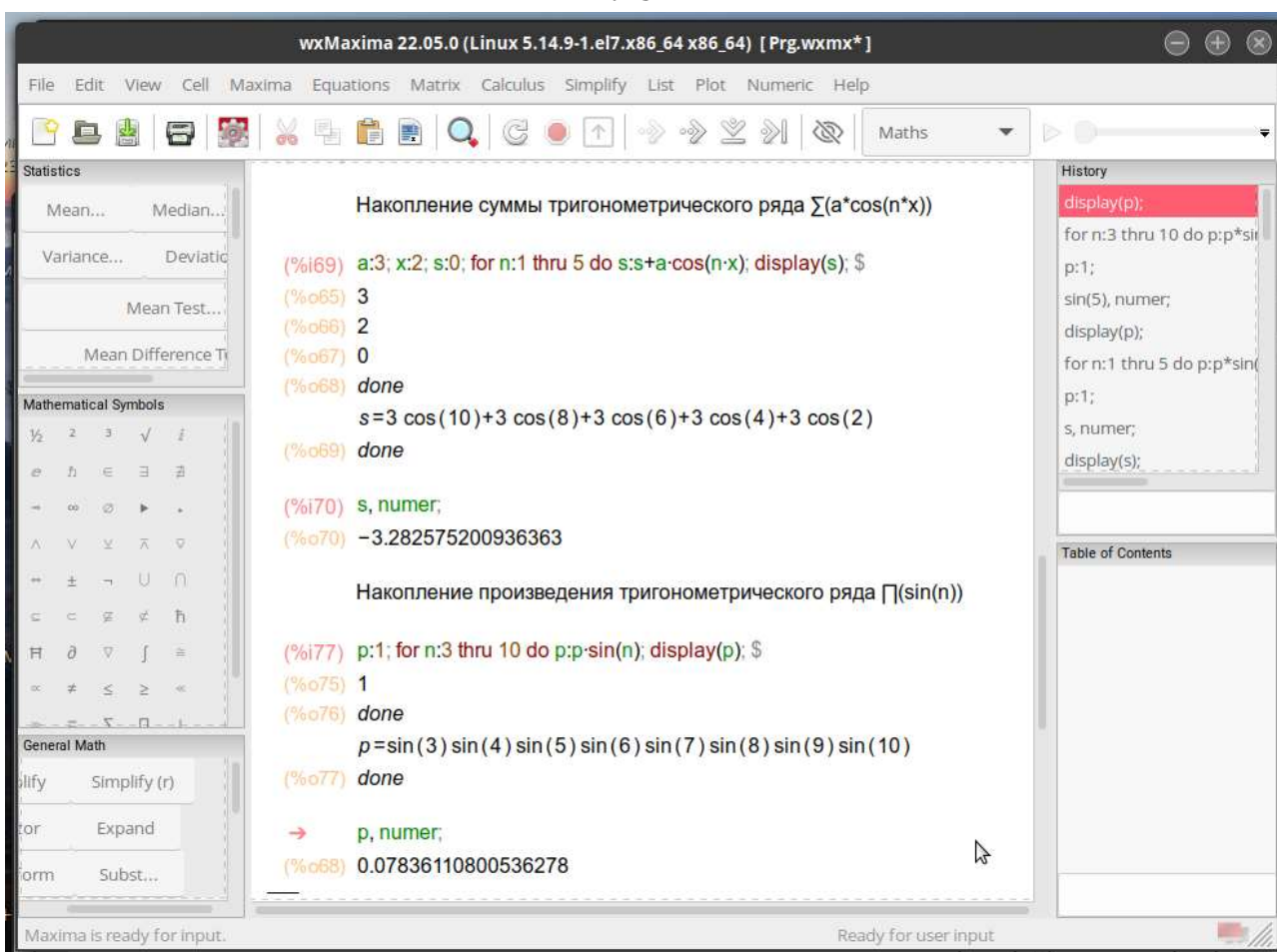
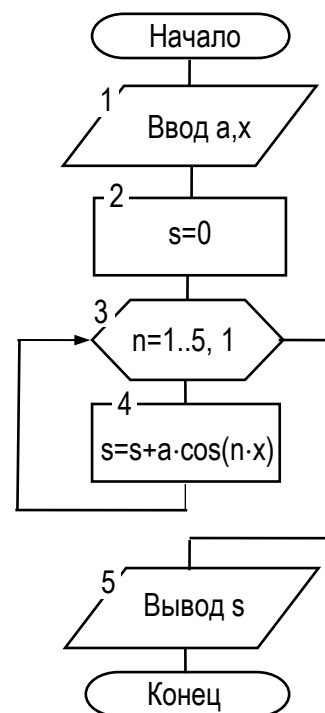
$s_5 = s_4 + a \cdot \cos(5 \cdot x)$

$s = s_5$ // результат накопления

Аналогично накапливается произведение членов ряда.

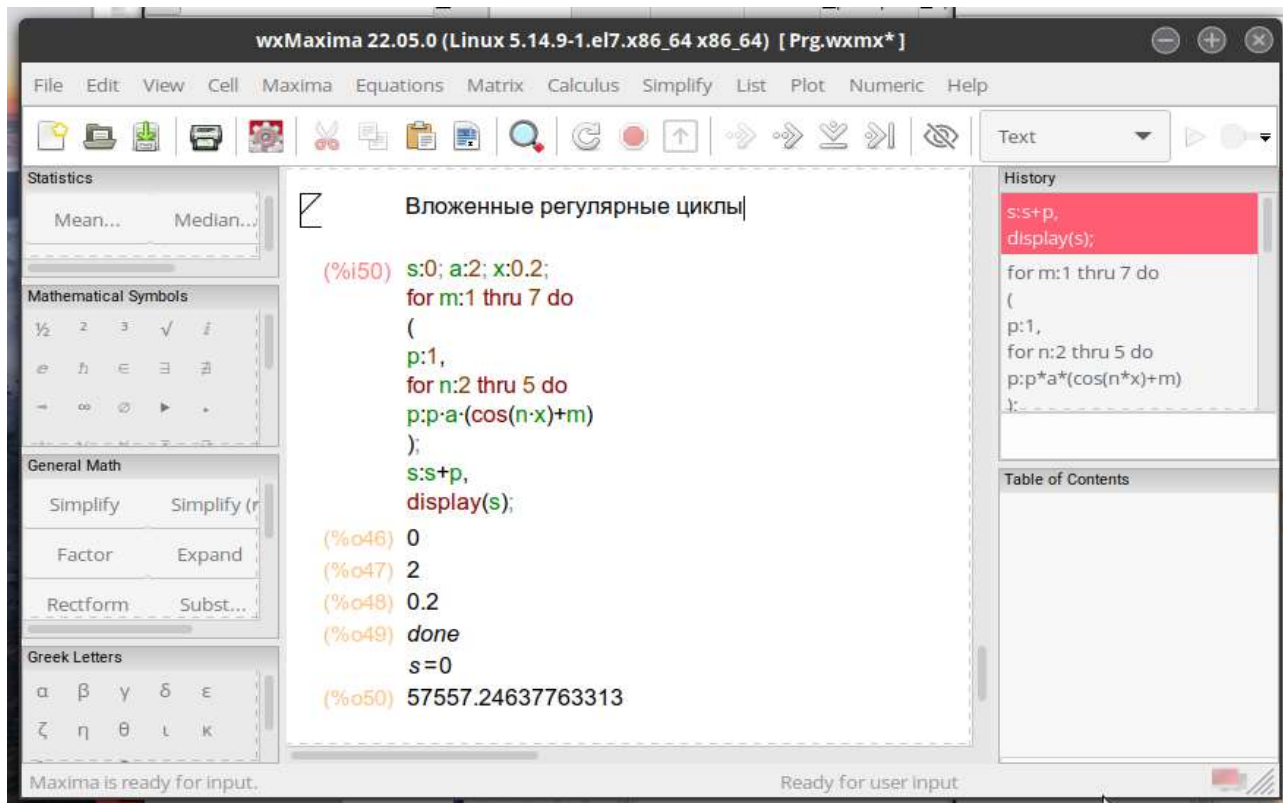
При этом начальное значение не должно влиять на результат вычисления, поэтому оно в этом случае принимается равным 1 ($p=1$).

$$p = \prod_{n=3}^{10} \sin(n)$$



Задача3. Пример вычисления функции, в состав которой входят и сумма, и произведение элементов тригонометрического ряда. Такие регулярные циклы называются вложенными и при их вычислении используют задание одного тела цикла (внутреннего) внутри другого (внешнего).

$$s = \sum_{m=1}^7 \prod_{n=2}^5 a \cdot \cos(n \cdot x) + m$$



Процесс вычисления с накоплением суммы и произведения

$s_0 = 0$ и $p_0 = 1$ // начальные значения

$$s_1 = s_0 + [(\cos(2 \cdot x) + 1) \cdot (\cos(3 \cdot x) + 1) \cdot (\cos(4 \cdot x) + 1) \cdot (\cos(5 \cdot x) + 1)]$$

$$s_2 = s_1 + [(\cos(2 \cdot x) + 2) \cdot (\cos(3 \cdot x) + 2) \cdot (\cos(4 \cdot x) + 1) \cdot (\cos(5 \cdot x) + 2)]$$

$$s_3 = s_2 + [(\cos(2 \cdot x) + 3) \cdot (\cos(3 \cdot x) + 3) \cdot (\cos(4 \cdot x) + 1) \cdot (\cos(5 \cdot x) + 3)]$$

$$s_4 = s_3 + [(\cos(2 \cdot x) + 4) \cdot (\cos(3 \cdot x) + 4) \cdot (\cos(4 \cdot x) + 1) \cdot (\cos(5 \cdot x) + 4)]$$

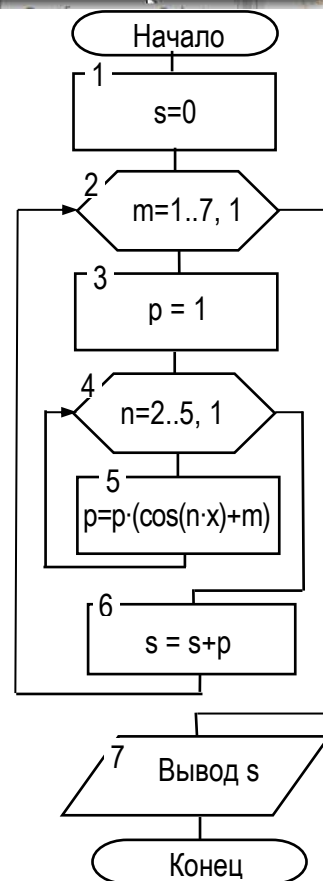
$$s_5 = s_4 + [(\cos(2 \cdot x) + 5) \cdot (\cos(3 \cdot x) + 5) \cdot (\cos(4 \cdot x) + 1) \cdot (\cos(5 \cdot x) + 5)]$$

$$s_6 = s_5 + [(\cos(2 \cdot x) + 6) \cdot (\cos(3 \cdot x) + 6) \cdot (\cos(4 \cdot x) + 1) \cdot (\cos(5 \cdot x) + 6)]$$

$$s_7 = s_6 + [(\cos(2 \cdot x) + 7) \cdot (\cos(3 \cdot x) + 7) \cdot (\cos(4 \cdot x) + 1) \cdot (\cos(5 \cdot x) + 7)]$$

$s = s_7$ // результат накопления

В этом примере присутствуют два регулярных цикла: один – внешний по параметру m , другой цикл – внутренний – по параметру n . В состав тела внешнего цикла входят блоки 3, 4, 5, 6, а в состав тела внутреннего цикла входит блок 5.



Задача4. Пример накопления суммы элементов 3-й строки и сумму элементов 2-го столбца матрицы $D[3 \times 3]$. Используя регулярные циклы можно производить разного рода операции с массивами.

Для заданной матрицы D накопить и вывести сумму элементов 3-й строки и сумму элементов 2-го столбца

```
(%i78) D:matrix([1,3,5],[2,4,6],[9,8,7]);
```

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \\ 9 & 8 & 7 \end{pmatrix}$$

```
(%o78)
```

```
(%i87) sr:0; for n:1 thru 3 do sr:sr+D[3,n]; display(sr);
```

```
(%o85) 0
```

```
(%o86) done
```

```
sr=24
```

```
(%o87) done
```

```
(%i90) sc:0; for n:1 thru 3 do sc:sc+D[n,2]; display(sc);
```

```
(%o88) 0
```

```
(%o89) done
```

```
sc=15
```

```
(%o90) done
```

Выводить четные элементы матрицы и элементы с условием.

```
(%i107) for n:1 thru 3 do
  for m:1 thru 3 do
    s: if mod(D[n,m],2)=0 then display(D[n,m]);
```

```
D2,1 =2
```

```
D2,2 =4
```

```
D2,3 =6
```

```
D3,2 =8
```

```
(%o107) done
```

```
(%i108) for n:1 thru 3 do
  for m:1 thru 3 do
    s: if D[n,m]>5 then display(D[n,m]);
```

```
D2,3 =6
```

```
D3,1 =9
```

```
D3,2 =8
```

```
D3,3 =7
```

```
(%o108) done
```


Цикл с предусловием

Цикл с оператором **while...** используется в том случае, если последовательность действий надо выполнить несколько раз, причем количество повторений заранее не известно и может быть определено, только во время работы программы.

Процессы, в которых количество повторений заранее не определено, называются итерационными. В итерационных циклах на каждом шаге вычислений происходят последовательное приближение и проверка условия достижения искомого результата. Выход из итерационного цикла осуществляется в случае выполнения заданного условия.

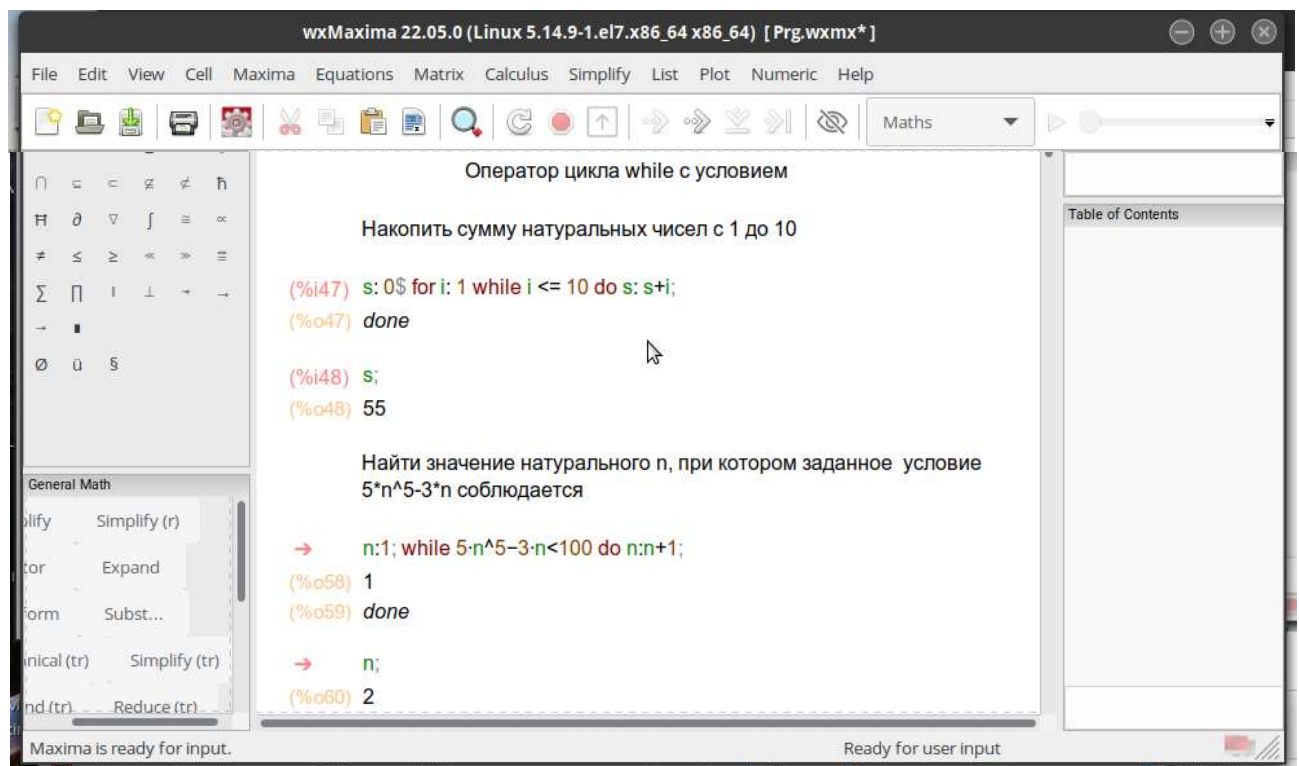
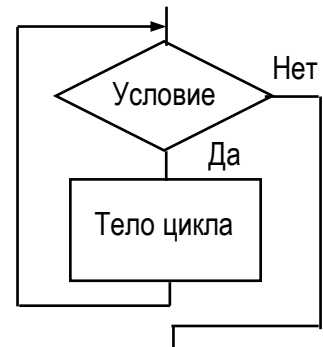
В этом «пока» цикла условие стоит перед телом цикла, поэтому этот цикл называют с предусловием. Тело цикла повторяется, **пока выполняется условие**. Тело цикла первый раз выполняется с проверкой условия.

Форма цикла while с одиночной инструкцией:

```
while <условие> do <тело цикла>;
```

В случае повторного выполнения программного блока формат цикла выглядит следующим образом:

```
while <условие> do
(
  <Инструкции тела цикла,
  .....>
)
```



Оператор **while** указывает, что цикл выполняется, пока выполняется условие.

Цикл с постусловием

Цикл с операторами **unless** ... **do** используется в том случае, если последовательность действий надо выполнить несколько раз, причем количество повторений заранее не известно и может быть определено во время работы программы.

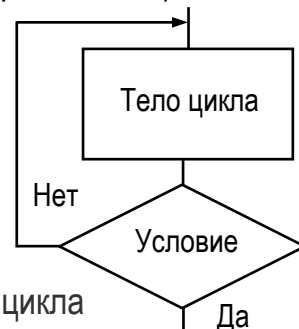
В этом виде цикла условие стоит после тела цикла, поэтому этот цикл называют с предусловием. Тело цикла повторяется, пока выполняется условие. Тело цикла первый раз выполняется без проверки условия. Выход из цикла при **невыполнении условия**.

Эта циклическая инструкция работает по принципу: «Повторить тело цикла пока не будет выполняться некоторое условие».

Ее синтаксис выглядит следующим образом:

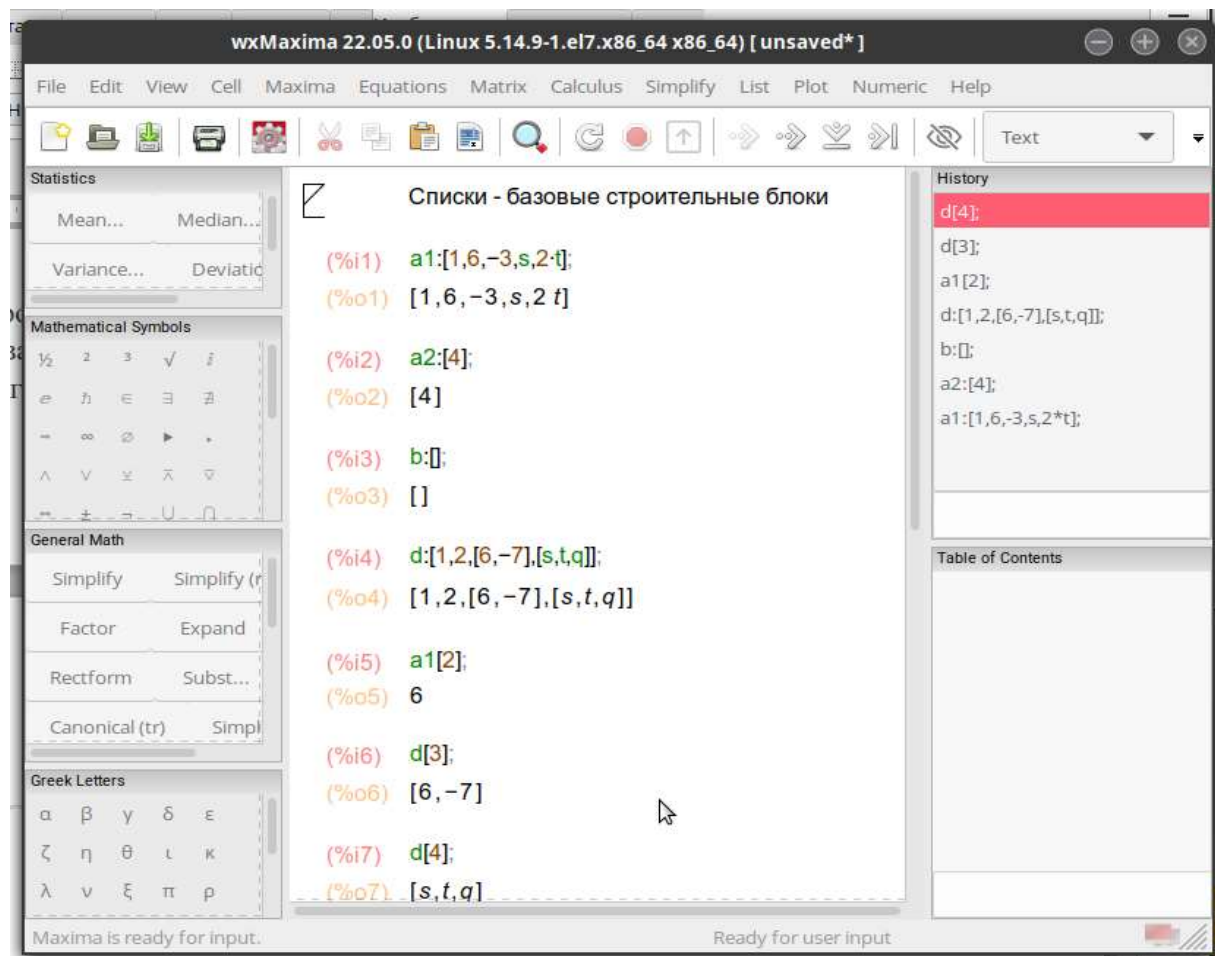
```
for i: 1 step 2 unless <условие> do <тело цикла>
```

Оператор **unless** указывает на то, что цикл выполняется, пока не будет достигнуто условие. При нормальном завершении цикла на экран возвращается **done**. Принудительный выход из цикла осуществляется при помощи оператора **return**, который может возвращать произвольное значение.



Списки

Списки - базовые строительные блоки для Maxima. Чтобы создать список необходимо в квадратных скобках записать все его элементы через запятую. Список может быть пустым или состоять из одного элемента. Задание списков и вывод элементы списка.



Задача5. Рассчитать функцию e^{-x} путем разложения ее в ряд с заданной точностью.

Ярким примером итерационного циклического процесса является вычисление функции, с заданной точностью, представленной с помощью ряда Тейлора.

Как известно, функцию $f(x)$ можно разложить в сходящийся ряд с бесконечным число членов, и вычислить путем сложения некоторого количества членов ряда

$$F(x) = a_0 + a_1 + a_2 + a_3 + \dots + a_{n-1} + a_n + a_{n+1} + \dots$$

Ряд считается сходящимся, если последовательность его частичных сумм имеет конечный предел. Этот предел называется суммой сходящегося ряда.

Зная первый член ряда и формулу n -го члена ряда можно вычислить члены этого ряда, сумму членов сходящегося ряда и, в итоге саму функцию с заданной точностью.

Если обозначить через q_n отношение $n+1$ члена ряда к n -му члену ряда

$$q_n = a_{n+1}/a_n, \quad (2)$$

то ряд можно представить как $F(x) = a_0 + a_0 \cdot q_0 + a_1 \cdot q_1 + a_2 \cdot q_2 + \dots + a_n \cdot q_n + a_{n+1} \cdot q_{n+1} + \dots$

Рассчитаем функцию e^{-x} с заданной точностью, например 0,001

Разложим функцию $\exp(-x)$ в ряд Тейлора. Ряд бесконечный, но сходящийся. Выведем несколько первых членов (например, 6) и проанализируем 0-й член ряда. Он равен 1 ($a_0=1$). Ряд образует геометрическую прогрессию. Рассчитаем знаменатель прогрессии $q(n)=a(n+1)/a(n)$. Зная первый член ряда $a(0)$ и знаменатель $q(n)$ можно рассчитать все последующие члены ряда.

(%i29) `taylor(exp(-x), x, 0, 6);`

(%o29) $1 - x + \frac{x^2}{2} - \frac{x^3}{6} + \frac{x^4}{24} - \frac{x^5}{120} + \frac{x^6}{720} + \dots$

n -й член ряда определяется по формуле $(-1)^n \cdot x^n / n!$

(%i30) `(-1)^n * (x)^n / (n)!;`

(%o30) $\frac{(-1)^n x^n}{n!}$

$(n+1)$ -й член ряда определяется по формуле $(-1)^{n+1} \cdot x^{n+1} / (n+1)!$

(%i31) `(-1)^(n+1) * (x)^(n+1) / (n+1)!;`

(%o31) $\frac{(-1)^{n+1} x^{n+1}}{(n+1)!}$

Рассчитаем знаменатель прогрессии $q(n)=a(n+1)/a(n)$

(%i32) `((-1)^(n+1) * x^(n+1) / (n+1)!) / ((-1)^n * (x)^n / (n)!);`

(%o32) $-\frac{n! x}{(n+1)!}$

Или после упрощения $q(n) = -x/(n+1)$. Т.к. $(n)!$ сократился.

При расчете каждого члена ряда необходимо его сравнивать с заданной точностью вычисления E , и, если выполняется неравенство

$$|a_n| < E,$$

то n -й член ряда суммируется с предыдущими членами и таким образом происходит накопление суммы. Процесс накопления завершается при невыполнении неравенства. Полученная сумма и есть функция $F(x)$.

Составим программу для решения задачи

The screenshot shows the wxMaxima 22.05.0 interface. The main window contains the following Maxima code and output:

```

Итерационные циклы. Работа со списками данных

(%i134) x:0.8; E:0.001;
n:0$ a:1$ s:a$
while abs(a)>E do
(
  L:list[a,s,n], /*Объявление списка данных*/
  q:(-x)/(n+1),
  a:a*q,
  s:s+a,
  n:n+1,
  display(L)
);
(%o129) 0.8
(%o130) 0.001
L=list
1, 1, 0
L=list
-0.8, 0.2, 1
L=list
0.3200000000000001, 0.52, 2
L=list
-0.08533333333333334, 0.43466666666666666, 3
L=list
0.01706666666666667, 0.4517333333333333, 4
L=list
-0.002730666666666667, 0.4490026666666667, 5
(%i134) done

(%i101) x:0.8$ f:exp(-x); /*Вычисление с использованием функции e^(-x)*/
(%o101) 0.4493289641172216

(%i99) abs(0.4490026666666667-0.4493289641172216);
(%o99) 3.262974505549021 10^-4

→ 3.262974505549021·10^-4<<0.001 (E) /*погрешность вычисления*/

Как видно из вычисления членов ряда a(n) при заданной
точности E=0,001 достаточно оставить в разложении 6 членов
ряда (n=0, 1, ... 5) и по их сумме вычислить саму функцию
exp(-x)=a(0)+a(1)+a(2)+a(3)+a(4)+a(5).
  
```

The left sidebar shows various mathematical symbols and functions. The right sidebar shows the History panel with the executed commands.

Для проверки верности вычисления вычисляется заданная функция

Выводы: как видно разница между вычисленным значением с помощью встроенных функций и разложением в ряд оказалось меньше заданной точности вычисления (0,001). Что подтверждает правильность рассуждений и полученных результатов.

