Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Факультет: «Специальное машиностроение»

Кафедра: «Робототехнические системы и мехатроника»

Лабораторная работа № 2

по курсу «Теория автоматического управления»

Вариант 5

Выполнил: Садовец Роман
Группа: СМ7-51Б

Проверил(а):

Москва, 2023 г.

I. Осциллятор

А) Рассматриваем осциллятор (рис. 1) ***без учета сухого трения*** с начальными условиями:

```
m = 1; % масса дощечки, кг
alpha = 0.1; % коэф-т вязкого трения, кг/с
k = 10; % жесткость пружины, Н/м
v_0 = 10; % начальная скорость, м/с
x_0 = 1; % начальная координата, м
t_sim = 20; % время симуляции, с
```

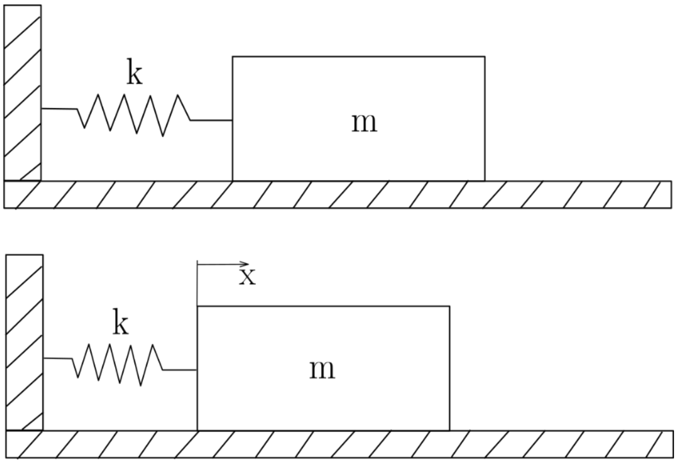


Рис 1. Осциллятор

Запишем дифференциальное уравнение для заданных условий:

$$m\ddot{x} = -\alpha\dot{x} - kx$$

$$\ddot{x} + \frac{\alpha}{m}\dot{x} + \frac{k}{m}x = 0 \qquad (1)$$

Для прорисовки всех необходимых figure (x(t); v(t); слева x(t) и справа v(t); сверху x(t) и снизу v(t)), введём функцию sim_oscillator(…), рисующую необходимые графики (**Важно** – функции вводятся в конце программы)

```
%Вызов прорисовки графиков функций
sim_oscillator(m, alpha, k, x_0, v_0, t_sim); % основная функция

function [] = sim_oscillator(m, alpha, k, x_0, v_0, t_sim)
   %функция, которую необходимо проинтегрировать (лямбда-выражение)
   dzdt = @(t, z) [z(2,1); -k / m * z(1,1) - alpha / m * z(2,1)];

   z0 = [x_0, v_0]; % задание начального состояния
   t0 = 0; % задание начального времени
   [t, z] = ode45( dzdt , [t0, t0 + t_sim], z0); %Запуск вычислений
   plot_oscillator(t, z); % функция, которую надо реализовать
end
```

Средствами MATLAB невозможно решить дифференциальные уравнения высших (в нашем случае второго) порядка, поэтому задачу необходимо свести к системе дифференциальных уравнений первого порядка. Для этого в уравнении (1) обозначим $\dot{x} = v$. Получим:

$$\dot{v} + \frac{\alpha}{m} v + \frac{k}{m} x = 0$$

Или в виде системы:

$$\begin{cases} \dot{x} = v; \\ \dot{v} = -\frac{\alpha}{m} v - \frac{k}{m} x; \end{cases} \quad (2)$$

В этом случае можно воспользоваться решателем ode45(odefun, tspan, y0), где odefun – дифур, tspan – интервал для вычислений, y0 – начальные условия. Он может принимать на вход вектор уравнений и решать их. Эти все действия продемонстрированы в виде кода выше.

plot_oscillator(t, z) – функция прорисовки графиков. Выполняется в отдельном файле (можно ссылаться на функцию из другого файла, если они в одном репозитории).

Ниже представлен программный код для вывода графиков x(t) и v(t) внутри функции plot_oscillator. На рис.2 – полученные figure

```
%%%%%%%%%%%%Coordinates%%%%%%%%%%%%%%
figure('Name','x(t) plot');
plot(t, z(:,1),"DisplayName","x(t)");
xlabel("t, m");
ylabel("x, m");
title("Coordinates of oscillator");
legend;
grid on;
%%%%%%%%%%%%%%Velocity%%%%%%%%%%%%%%%
figure('Name','v(t) plot');
plot(t, z(:,2),"DisplayName","v(t)");
xlabel("t, m");
ylabel("v, m");
title("Velocity of oscillator");
legend;
grid on;
```

figure() – создание окна;

plot() – рисование графика (кривой) по точкам;

xlabel(), ylabel() – наименование осей;

title() – заголовок окна;

legend – включить легенду графика;

grid on – включить сетку графика;

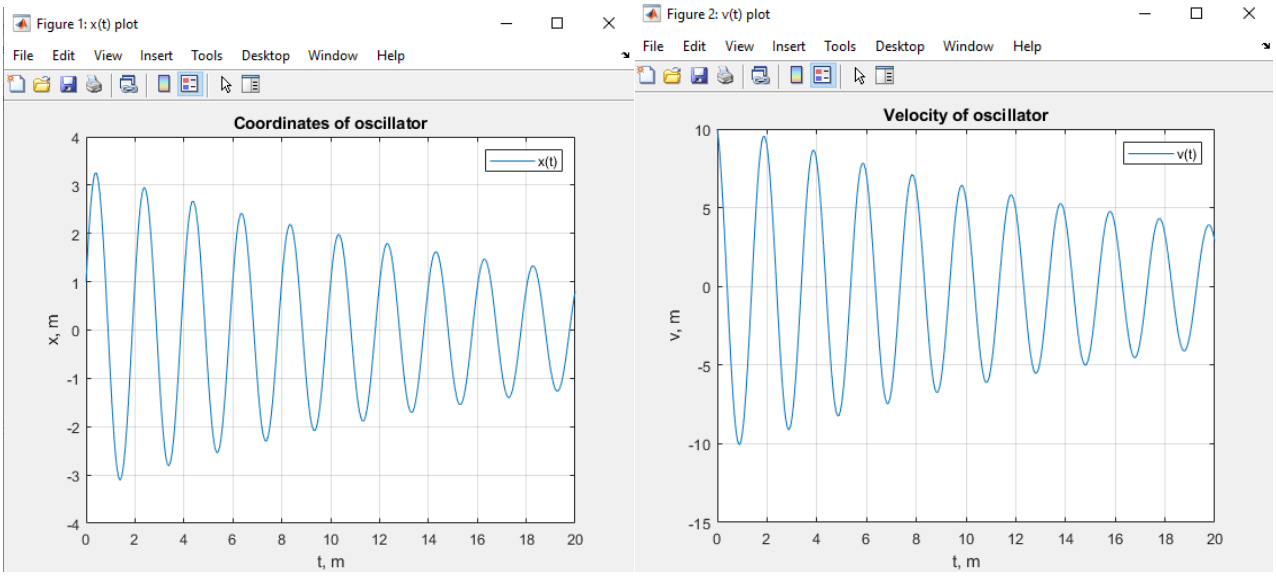


Рис. 2. Графики x(t) и v(t)

Далее представлен программный код для прорисовки x(t) слева и v(t) справа на одной figure:

```
figure('Name','Coordinates and velocities');
subplot(1,2,1); %Создание в окне сетки 1:2. Выбор активным окна 1
hold on; %Сохранение графика
plot(t, z(:,1),"DisplayName","x(t)","LineStyle","-.");
xlabel("time, s");
ylabel("meters");
title("Coordinates");
grid on;
legend;

subplot(1,2,2); %Выбор активным окна 2 в сетке
hold on;
plot(t, z(:,2),"DisplayName","v(t)","Color","black");
xlabel("time, s");
ylabel("m/s");
title("Velocities");
grid on;
legend;
hold off;
```

Полученный результат представлен на рис. 3.

Для реализации последнего графика, необходимо в предыдущем коде сменить сетку на 2:1 (subplot(2,1,1)). Графики представлены на рис. 4.

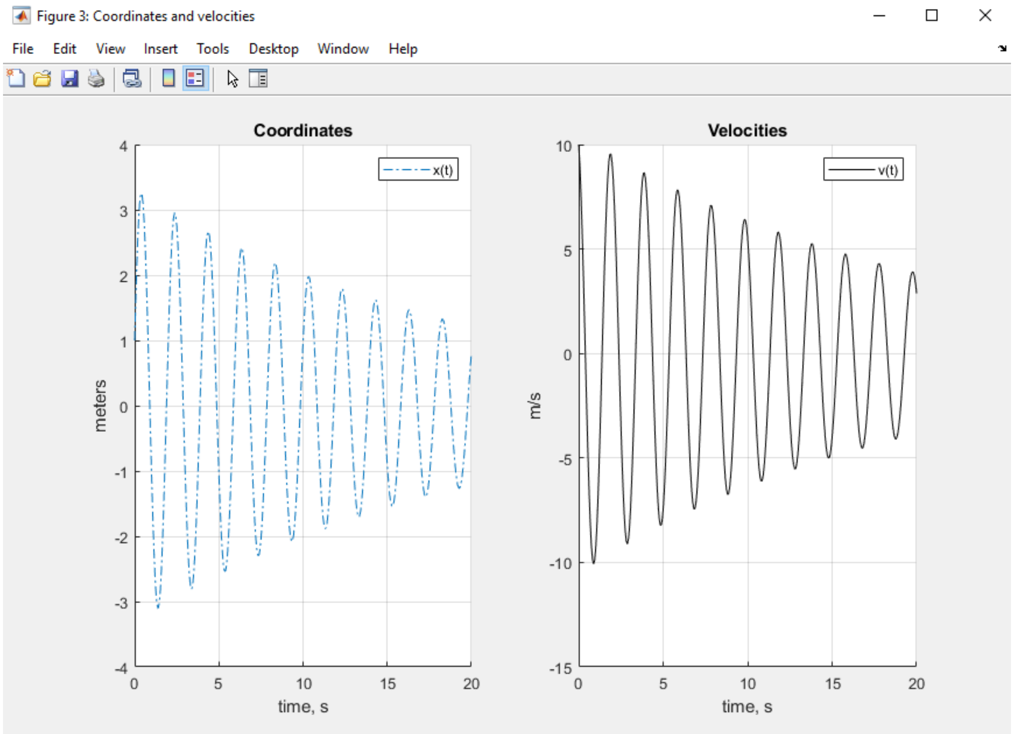


Рис. 3. Слева x(t) и справа v(t)

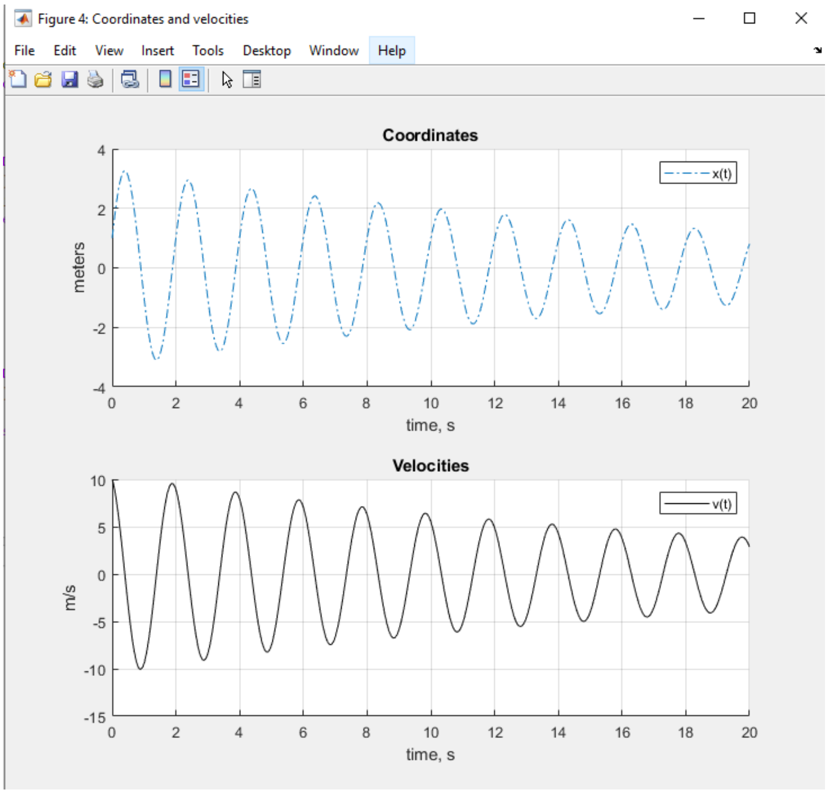


Рис. 4. Сверху x(t) и снизу v(t)

Б) Рассматриваем осциллятор **с *учетом сухого трения (сухое трение скольжения и сухое трение покоя)*** с начальными условиями:

```
m = 1; % масса дощечки, кг
alpha = 0.1; % коэф-т вязкого трения, кг/с
k = 7; % жесткость пружины, Н/м
mu = 0.2; % Коэф-т сухого трения скольжения стали по стали
g = 9.81; % Ускорение свободного падения, м/с^2
v_0 = 10; % начальная скорость, м/с
x_0 = 1; % начальная координата, м
t_sim = 20; % время симуляции, с
```

Тогда дифференциальное уравнение **в случае движения в направлении оси Ox** примет вид:

$$m\ddot{x} = -\alpha\dot{x} - kx - \mu mg$$

$$\ddot{x} + \frac{\alpha}{m}\dot{x} + \frac{k}{m}x = -\mu mg \quad (3)$$

Или в виде системы:

$$\begin{cases} \dot{x} = v; \\ \dot{v} = -\frac{\alpha}{m}v - \frac{k}{m}x - \mu mg; \end{cases} \quad (4)$$

При движении в отрицательном направлении:

$$\ddot{x} + \frac{\alpha}{m}\dot{x} + \frac{k}{m}x = \mu mg \quad (5)$$

Для учета времени воспользуемся конструкцией **if** – будем проверять направление движения бруска ($v > 0$ или $v < 0$). Для реализации подобной схемы вынесем выбор необходимого диффура в отдельную функцию. Это можно реализовать с помощью анонимной функции, которая указывает на отдельную функцию программы. Не стоит забывать и про сухое трение покоя: в случае остановки/смены направления движения мы должны проверять, преодолевает ли сила упругости пружины силу сухого трения покоя:

$$\left|\frac{k}{m}x\right| > \mu mg \quad (6)$$

Тогда программный код для чередования формул (3) и (5), а также учета (6) в MATLAB будет выглядеть следующим образом:

```
function [] = sim_oscillator(k,m,alpha,mu,g,x_0,v_0,t_simulation)
    z0 = [x_0, v_0]; % Задание начального состояния
    t0 = 0; % Задание начального времени
    z = []; %Общий вектор состояния осциллятора
    t = []; %Общее время движения осциллятор
```

```
    % Назначение рабочего диффура в зависимости от направления движения
    dzdt = @(t, z) differential(t, z, k, m, alpha, mu, g);
    % Запуск вычислений
    [t, z] = ode45( dzdt , [t0, t0 + t_simulation], z0);
    plot_oscillator(t, z); % Функция отрисовки графиков
end

function dzdt = differential(t, z, k, m, alpha, mu, g)
   if ( abs( k/m * z(1) ) < mu * m * g) & ( z(2) == 0 )
   % Остановка в случае, если пружина не преодолевает силу сухого трения покоя
      dzdt = [0; 0];
   else
      if ( z(2) > 0 )
         %Движение по направлению оси Ох
         dzdt = [z(2); -k / m * z(1) - alpha / m * z(2) - mu * m * g];
      else
         %Движение против направления оси Ох
         dzdt = [z(2); -k / m * z(1) - alpha / m * z(2) + mu * m * g ];
      end
   end
end
```

Также в качестве дополнения и лучшего понимания перемещения, выведем фазовый портрет v(x) движения осциллятора. Полученные графики для движения с силой трения покоя представлены на рис. 5 и рис. 6.

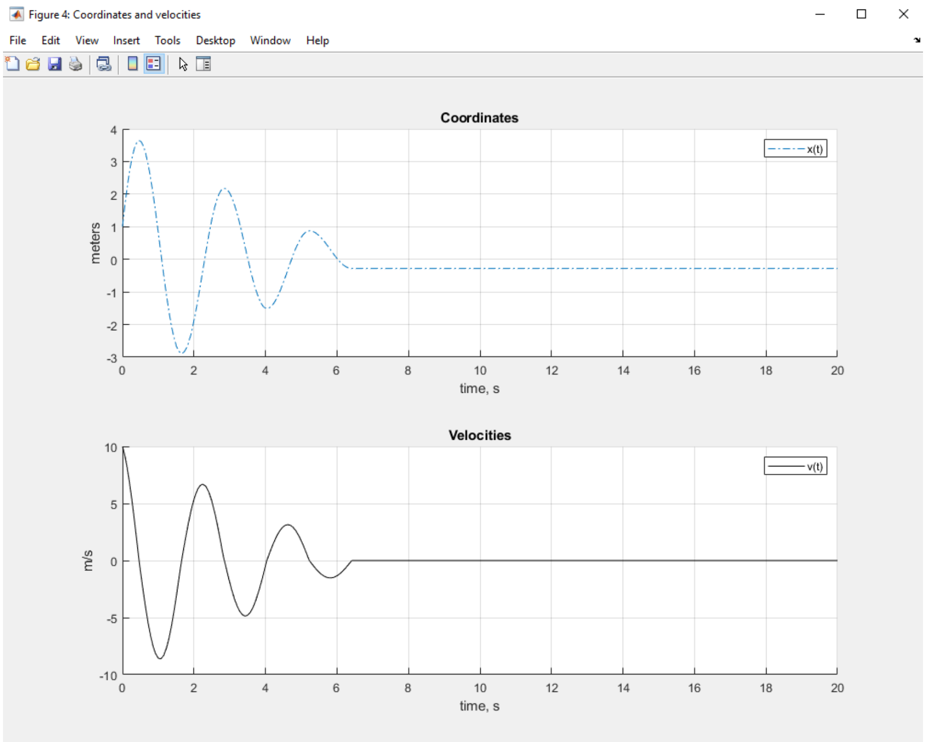


Рис. 5. Графики x(t) и v(t) при наличии сухого трения

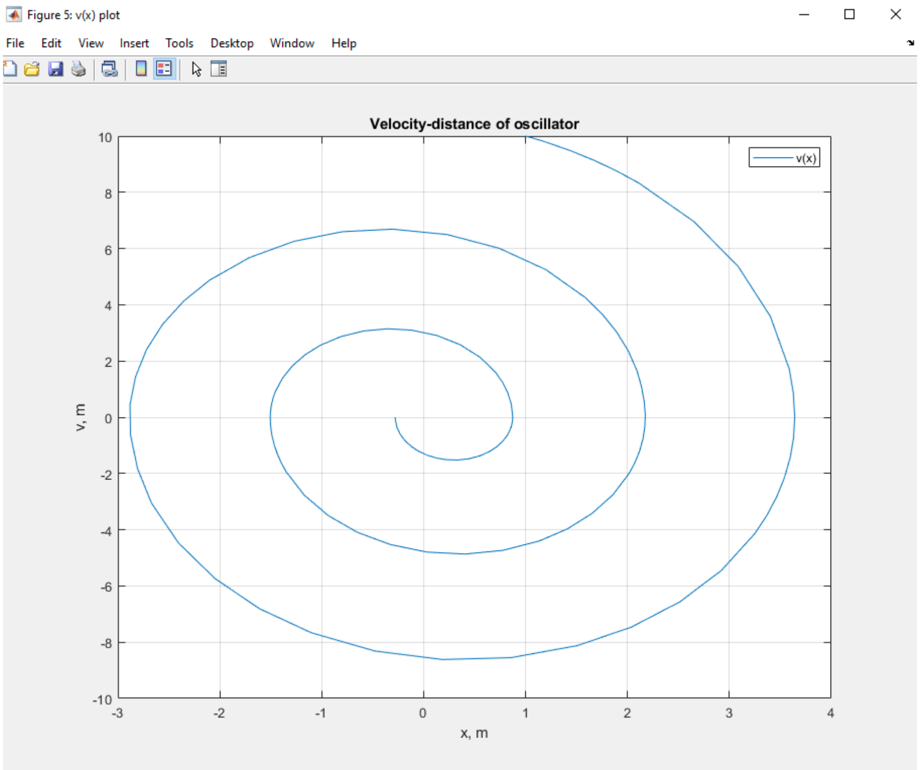


Рис. 6. Фазовый портрет v(x) при наличии сухого трения

II. Нахождение локального минимума функции с применением градиентного спуска

Для 5-го варианта функция и начальные условия представлены в табл. 1

Вариант	Функция	Начальная точка	Отрезок
5	$x^2 - 2x + e^{-x}$	1.8	[0.5, 2]

Табл. 1

Для ввода функции и её первой производной воспользуемся Анонимной функцией (лямбда-функцией). Для упрощения поиска производной, можно воспользоваться функцией diff(). Стоит учитывать, что diff() на выходе даёт функцию в символьном виде (type syms), что не позволяет её использовать для расчётов. Для решения проблемы, необходимо используемые переменные перевести в символьный вид, и записать производную в символьном виде:

```
y = @(x) x.^2 - 2.*x + exp(-x);

syms x;
g = diff(y(x));
dydx = @(x) vpa(subs(g, x), 5);
```

Анонимная функция y = y(x) работает корректно. Однако переменная g = diff(y(x)) – символьный тип, и для работы с ним необходимо ввести переменную x в символьном виде и падать её на вход. С помощью subs() производим замену символьной переменной x на число. vpa() необходимо для округления числа до 5 знака после запятой.

Начальные условия задачи:

```
x_interval = 0.5:0.05:2;
x_k = 1.8; % Начальная точка отсчёта
y_k = y(x_k); % Значение функции в начальной точке
```

Отрисовка функции:

```
figure('Name','Function');
xlabel("x");
ylabel("y(x)");
title("Gradient descent");
grid on; % Включение сетки на графике
hold on; % Сохранение графика

% Прорисовка функции на заданном диапазоне
plot(x_interval, y(x_interval),"DisplayName","y(x)", "Color","black");
```

Пропишем начальные условия для градиентного спуска:

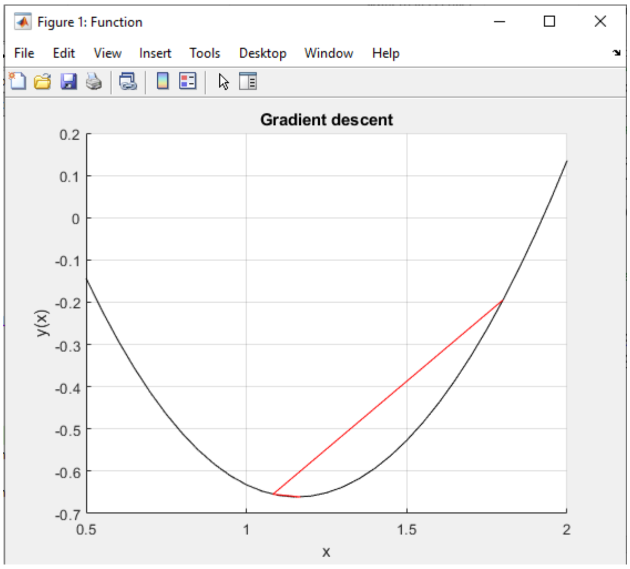
```
%Gradient descent
alpha = 0.5; %Коэф-т приближения
Epsilon = 0.01; %Ошибка, до которой работает градиентный спуск
i = 0; %Ограничитель по количество итераций
```

Остается осуществить прорисовку функции и вывести конечные значения для полученного значения локального экстремума (локального минимума). Для этого воспользуемся циклом while с прорисовкой каждой кривой одна за другой:

```
while ( abs( dydx( x_k ) ) > Epsilon ) && ( i < 100 )
  %Цикл будет работать, пока |f'(x_k)| не станет меньше некоторого ε, или
  %пока не будет достигнуто ограничение по количеству итераций i
  h = -alpha * dydx(x_k); %Шаг градиентного спуска
  %Рисуем прямую, соединяющую старую точку градиентного спуска и новую
  plot([x_k, x_k+h],[y_k, y(x_k + h)],"r")
  x_k = x_k + h; %Поиск x_{k+1} = x_k + h
  y_k = y(x_k);
  i = i+1; %Считаем итерации
  pause(1); %Задержка между отрисовкой
end

%Вывод конечных значений (локального минимума и количества пройденных
% итераций i), с округлением до 5 знаков после запятой
iterations = i
x_last = round(x_k, 5)
y_last = round(y_k, 5)
```

Получим график функции и вывод в командную строку (рис. 7.)



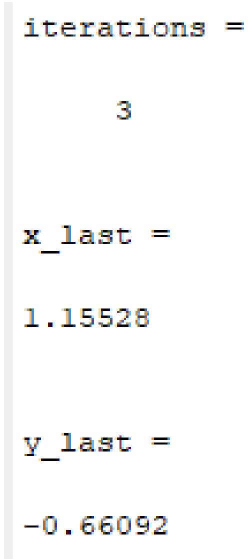


Рис. 7. Полученный график с градиентным спуском и значения полученного локального экстремума (лок. минимума функции) с количеством произведенных итераций

Приложение

1. Публичный репозиторий для лабораторных работ по ТАУ // GitHub URL: https://github.com/RiXenGC/Control-Theory.git