Итак, небольшой гайд по выполнению 3ей ЛР.

Что у нас в задании?

1. Самостоятельно ознакомиться со справочными сведениями относительно приложения MatLab Simulink.

2. Построить фазовый портрет и графики во временной области непрерывной модели решения дифференциального уравнения.

3. Разработать модель Simulink для решения дифференциального уравнения.

4. Построить графики дискретной (не)линейной модели решения разностного уравнения.

5. Разработать модель Simulink для решения разностного уравнения (системы уравнений).

6. Получить сравнительные графики поведения моделей при разных параметрах дифференциального уравнения, параметра дискретизации и настроек Simulink.

7. Составить и представить преподавателю отчет о работе.

I. Пункты 2-3 - т.к. сначала необходимо построить графики, а затем разработать модель - надо сначала решить диф. уравнение с помощью скрипта Matlab,а потом уже построить схему. Решается диф. уравнение с помощью функции ode45.

Для этого необходимо написать функцию, задающую- диф. уравнение, для этого надо преобразовать наше первое уравнение из варианта таким образом, чтобы в левой части осталась только производная.

Пример Вар№1: y' + 2\*x\*y = x \* exp(-x^2) => y' = x \* exp(-x^2) - 2\*x\*y

после чего объявляем функцию вида diff\_eq = @(x, y) x \* exp(-x^2) - 2\*x\*y;

передаем функцию в метод ode45, получаем решение.

x = linspace(-2, 2, 1000); % интервал для решения

y0 = 2; % Начальное условие y(0) = 2

[t, y] = ode45(diff\_eq, x, y0); % решение

по решению строим графики - фазовый портрет и графики решений.

Фазовый портрет для одного диф. уравнения - это соотношение искомой функции y и её производной y'. Поэтому сначала ищем для каждого [t,y] значение производной, например так:

diffValues = zeros(size(y, 1), 1);

for i = 1:size(y,1)

diffValues(i) = diff\_eq(t(i), y(i));

end

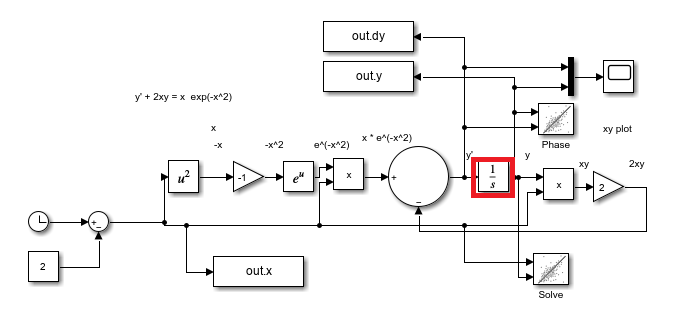
Потом строим графики:

plot(diffValues, y); % фазовый портрет

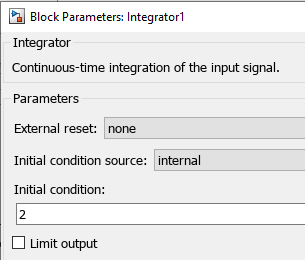
plot(t, y); % график решения ДУ

Далее строим модель.

В наших вариантах вроде бы как везде фигурирует производная первого порядка, поэтому нам понадобится один интегратор, который обозначается как 1/s



На вход к нему приходит производная, выходит первообразная. В настройках интегратора задаем Initial Value начальным значением искомой функции.



Далее собираем нашу правую часть уравнения с помощью сигналов.

Слева в интегратор должна прийти производная. По нашему уравнению она равна:

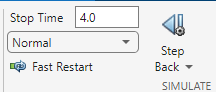
y' = x \* exp(-x^2) - 2\*x\*y

Для вычисления нам нужны значения x и y. y у нас есть, он выходит из интегратора. x – это параметр функции y. В нашем случае его роль играет текущее время, вернее текущее время его задает. Для получения времени используем элемент Clock.

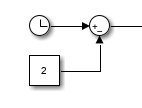
Мы задали какие-то границы интегрирования в нашем скрипте:

x = linspace(-2, 2, 1000); % интервал для решения

x - от -2 до 2, значит выполнять модель будем 4с. устанавливаем конец симуляции:

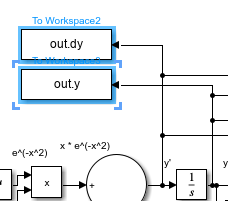
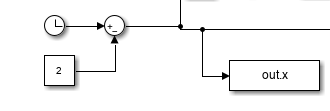


Но время начинается с 0, а нам нужно -2. Значит вычитаем из времени 2 с помощью сумматора:

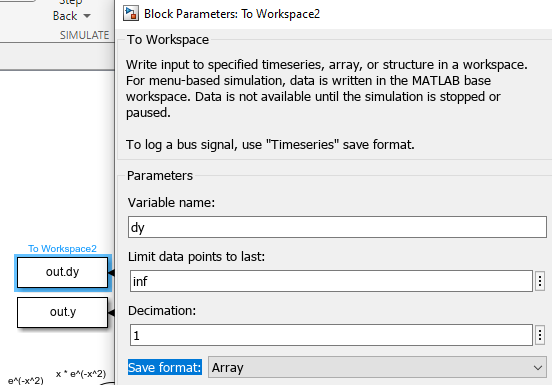


Готово, у нас есть x и y. Составляем из них выражение всякими умножениями, делениями, экспонентами и т.д., как на схеме, и подключаем в левую часть интегратора. На схеме подписано как преобразуется выражение.

По данным симуляции надо построить графики. Можем использовать элемент To Workspace, который выведет наши значения в массивы размером по кол-ву итераций работы системы:

В настройках надо выбрать Array:



Вывести надо y (искомую функцию), dy (производную), x (аргумент). По ним строим графики, вместе с теми что построили раньше из скрипта.

% Построение фазового портрета

figure

subplot(1, 2, 1);

hold on;

plot(diffValues, y);

plot(out.dy, out.y);

hold off;

xlabel('dy/dx');

ylabel('y');

title('Фазовый портрет');

% Построение графика решения

subplot(1, 2, 2);

hold on;

plot(t, y);

plot(out.x, out.y);

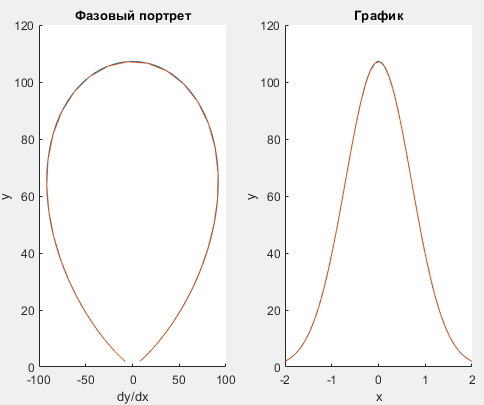
hold off;

xlabel('x');

ylabel('y');

title('График');

Получаем графики:



Чтобы в структуру out попали наши значения из модели, перед построением графика либо запускаем модель руками, либо с помощью out = sim("lab3\_1.slx");

II.Пункты 4-5-6-7

Вообще в задании не сказано о том, что надо разрабатывать модель для решения системы ДУ. Однако, я это понял после того, как сделал модель) но все равно в методичке написано, что надо сравнить реальное решение и решение дискретизацией, поэтому, наверное, хотя бы скриптом систему ДУ решить надо. Системы у нас простые, делаем все так же как с одним уравнением:

dxdt = @(t, x) 4\*x(1) - x(2);

dydt = @(t, x) x(1) + 2\*x(2);

%здесь x(1) – x, x(2) – y. Сам по себе x это вектор.

% Начальные условия

x0 = -1;

y0 = 0;

tspan = [0 1]; % Диапазон времени для решения

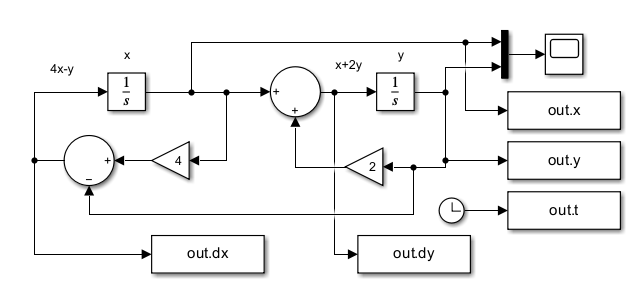
% Решение системы дифференциальных уравнений

[t, sol] = ode45(@(t, x) [dxdt(t, x); dydt(t, x)], tspan, [x0; y0]);

Фазовый портрет строим по полученным значениям x,y, а не как раньше, по производной и искомой функцией. Но в примере и такие тоже строятся) кажется, они не обязательны

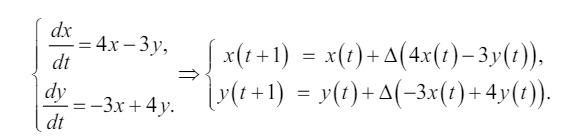
plot(sol(:, 1), sol(:, 2), "b", "LineWidth", 3);

Схема имеет следующий вид:



Все также, строим графики по снятым значениям, проверяем совпадение с результатом скрипта. Только теперь тут два интегратора, которые друг другу скармливают значения искомых функций. Не забываем в параметрах задать начальные значения для интеграторов. Возможно, что эту схему вообще строить не надо) а надо только для дискретизации. но это не точно, из задания сложно понять

Теперь дискретизация системы ДУ методом Эйлера. Сама дискретизация вроде бы несложна:



Функции в скрипте имеют вид:

nextX = @(xt, yt, delta) xt + delta .\* (4 .\* xt - yt);

nextY = @(xt, yt, delta) yt + delta .\* (xt + 2 .\* yt);

Принимают предыдушие значения функции, преобразуют по формуле.

Наверное есть встроенная в матлаб функция для решения дискретизированных функций, но я написал свою, куда надо передать функции для получения следующих значений, интервал, начальные значения, параметр дискретизации:

% решение системы двух дифференциальных уравнений методом дискретизации

% эйлера

function [t, x, y] = eiler(nextX, nextY, startX, startY, fromT, toT, delta)

% кол-во шагов алгоритма для заданного параметра дискретизации

steps = int32((toT - fromT) ./ delta) + 1;

t = zeros(steps, 1);

x = zeros(steps, 1);

y = zeros(steps, 1);

prevX = startX;

prevY = startY;

% первые элементы - начальные значения

x(1) = startX;

y(1) = startY;

t(1) = fromT;

for i = 2:steps

x(i) = nextX(prevX, prevY, delta);

y(i) = nextY(prevX, prevY, delta);

localT = delta \* double(i - 1);

t(i) = localT;

prevX = x(i);

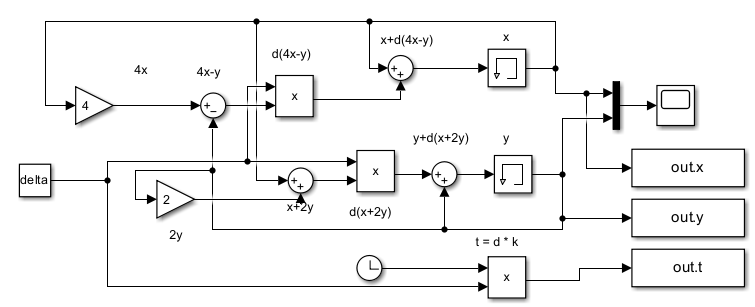
prevY = y(i);

end

end

Собственно просто итерируемся по интервалу с шагом delta, выдает результат.

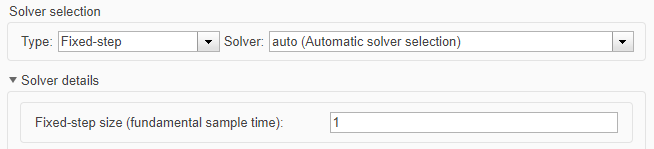
Также строим модель для решения дискретизацией:



Теперь интегрирование нам не нужно, нам нужно только чтобы значение сохранялось между итерациями. Поэтому вместо интеграторов используем блок Memory, в котором задаем начальные значения, так же как в интеграторах. Остальное все аналогично, блоками составляем выражение, приводим в Memory на вход.

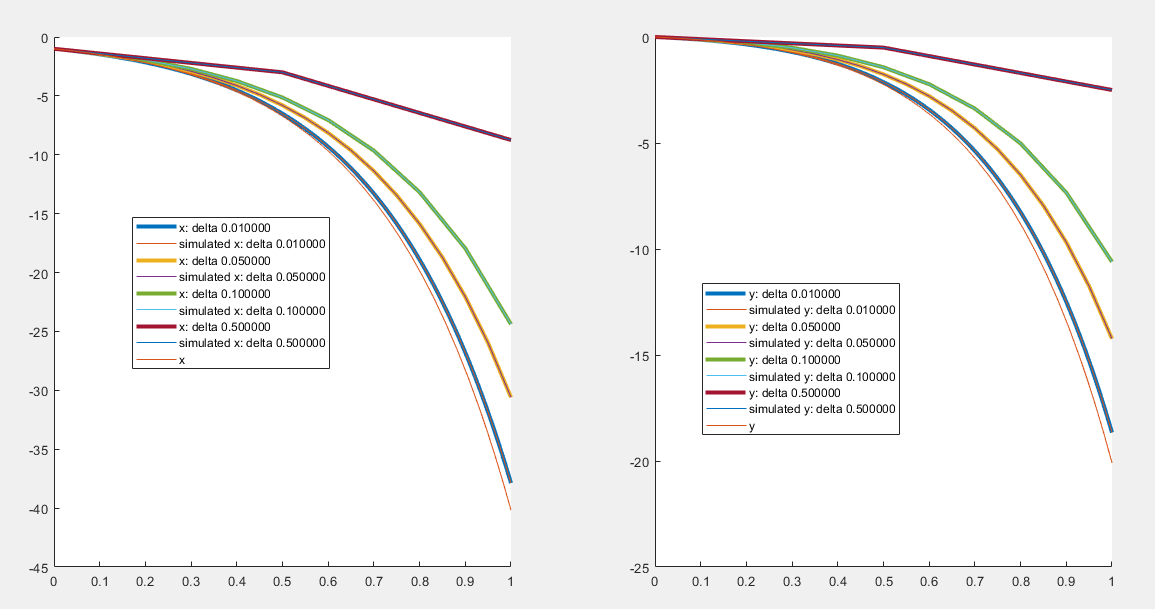
За delta у нас отвечает переменная из workspace, которую мы устанавливаем в цикле (см lab3\_56.m).

ВАЖНО! Установить в параметрах модели такие настройки:



Чтобы у нас время было целым и имело значение текущей итерации. t вычисляем по формуле t = d \* k, оно нам нужно для построения графиков.

Строим графики, сравниваем с реальными графиками из скрипта (предыдущий скрипт):



Проверяем: графики совпадают (модельные и скриптовые), ближе всего к реальным функциям модели с наименьшей delta.

Как оно работает подробнее смотрим в скриптах.