Вычисление оптимального управления методом динамического программирования в среде Matlab

Тарасов О. В.
АТ-05-1

Вступление

Процесс написания программы в среде Matlab можно условно разделить на две части: создание макета пользовательского интерфейса и написание кода, занимающего непосредственно расчетами.

Для начала рассмотрим основные принципы, на которых основана работа пользовательского интерфейса. Типичный интерфейс состоит из формы, на которой размещаются элементы управления (далее – «контролы»), такие как кнопки, текстовые поля и графики. Каждый тип контрола обладает некоторыми свойствами, которые влияют на его поведение. Некоторые свойства являются общими для всех типов (например, свойство tag, обозначающее имя контрола), в то время как другие свойства могут быть уникальными для данного типа контролов. Каждый из контролов имеет внутренний идентификатор, или хэндл. С помощью хэндла можно получить доступ к свойствам контрола напрямую из программного кода.

Одним из основополагающих понятий является также callback-функция. Эти функции представляют собой механизм, с помощью которого в Matlab происходит обработка событий, сгенерированных контролами. Каждое событие, например, нажатие кнопки, имеет свою callback-функцию, которая вызывается автоматически при наступлении этого события. В каждую callback-функцию передаются три параметра:

- hObejct – хэндл объекта, который вызвал данное событие. Имея хэндл, мы можем легко получить доступ к свойствам данного контрола.
- Eventdata – зарезервированная переменная.
- Handles – список хэндлов всех контролов, расположенных на форме. Эту переменную можно использовать в конструкциях вида handles.{св-во tag нужного контрола} для получения хэндлов других контролов.

Завершая краткий обзор основных принципов программирования в Matlab, стоит сказать об области видимости и объявлении переменных. Matlab не является строго типизированным языком, поэтому любая переменная в нем может хранить любо тип (целое, строку, число с точкой и т. д.). Обычные переменные не требуют специального объявления, то есть их можно начинать использовать по мере необходимости. Однако такие переменные можно использовать только внутри одной функции. Если вы будете использовать в разных функциях переменные с одинаковым именем, вы просто получите две никак не связанные между собой переменные. Однако иногда может потребоваться использовать одну общую переменную в нескольких функциях. В таком случае, в каждой функции, использующей данную переменную, она должна быть объявлена со спецификатором доступа global:

```
global hLog
global I_AKR
global t_reg
global LogCnt
```

Безусловно, вышеизложенный материал описывает лишь малую часть возможностей, предоставляемых средой Matlab, однако его должно хватить для решения поставленной задачи.

Постановка задачи

Пусть система описывается выражением:

$$W(p) = \frac{k}{T \cdot p + 1}$$

$$x(0) = k;$$

$$x(\infty) = 0;$$

Необходимо обеспечить управление системой таким образом, чтобы интеграл потерь вида

$$\int_0^{\infty} (C_1 \cdot x^2 + C_2 \cdot u^2) dt \to \min_u$$

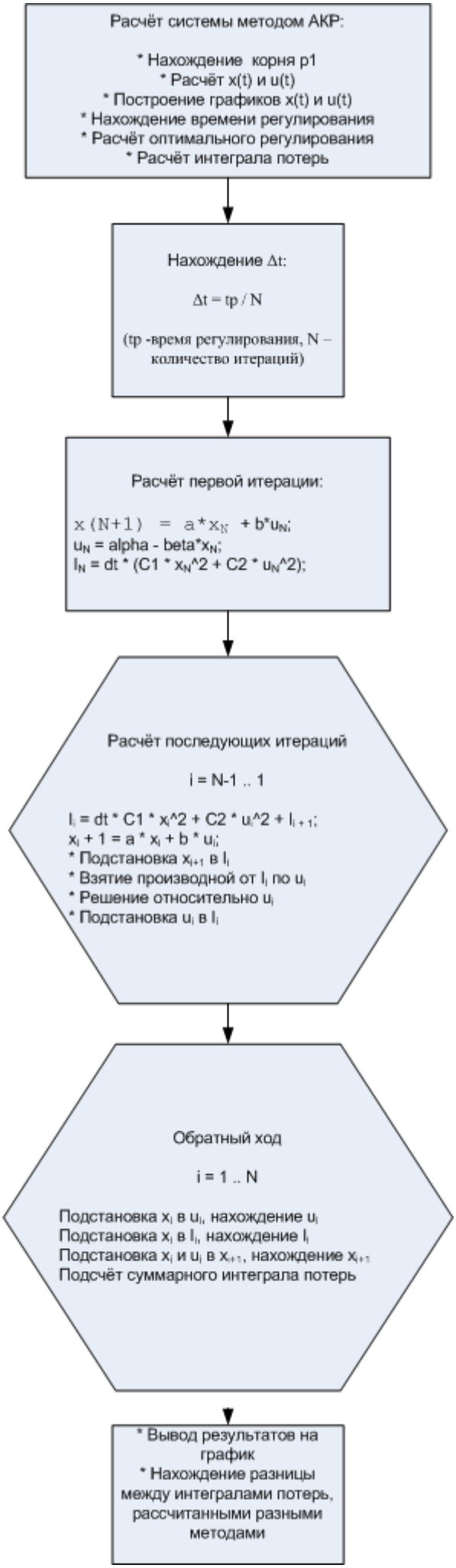
стремился к минимуму по U.

Будем решать задачу последовательно двумя методами: аналитического конструирования регулятора и динамического программирования.

Метод АКР даёт точный результат, однако объекты, полученные с его помощью, зачастую оказываются физически нереализуемы. В таких случаях может оказаться полезным метод динамического программирования, хоть он и вносит определённую погрешность. Данный метод удобен для программной реализации на компьютере, который позволяет быстро произвести большое количество итераций, тем самым значительно снижая погрешность вычислений.

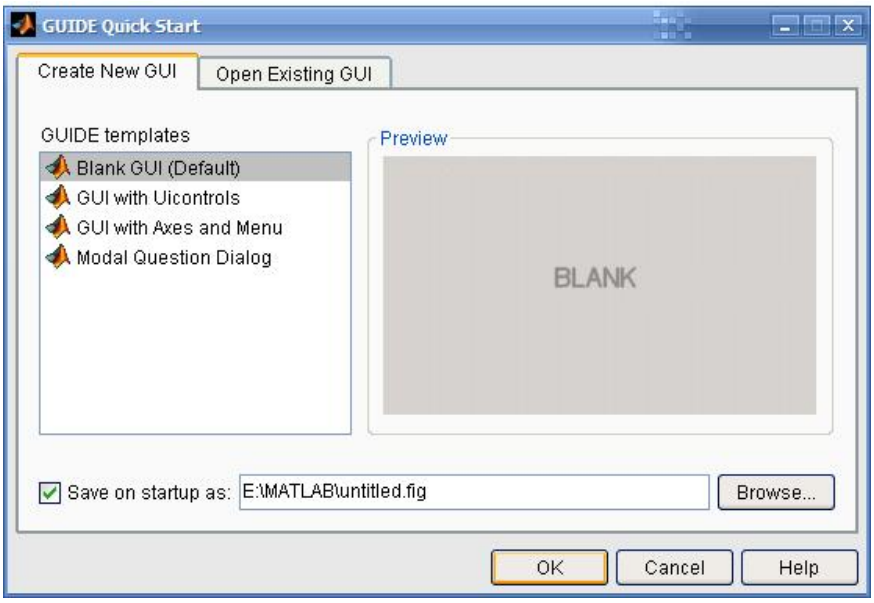
Общий алгоритм работы программы

Прежде, чем приступить к реализации метода динамического программирования, стоит продумать структуру будущей программы. На следующем рисунке приведена упрощённая блок-схема, описывающая процесс расчёта.


Расчёт системы методом АКР:
* Нахождение корня p1
* Расчёт x(t) и u(t)
* Построение графиков x(t) и u(t)
* Нахождение времени регулирования
* Расчёт оптимального регулирования
* Расчёт интеграла потерь
Нахождение Δt:
Δt = tp / N
(tp -время регулирования, N – количество итераций)
Расчёт первой итерации:
x(N+1) = a*x_N + b*u_N;
u_N = alpha - beta*x_N;
l_N = dt * (C1 * x_N^2 + C2 * u_N^2);
Расчёт последующих итераций
i = N-1 .. 1
l_i = dt * C1 * x_i^2 + C2 * u_i^2 + l_i+1;
x_i + 1 = a * x_i + b * u_i;
* Подстановка x_i+1 в l_i
* Взятие производной от l_i по u_i
* Решение относительно u_i
* Подстановка u_i в l_i
Обратный ход
i = 1 .. N
Подстановка x_i в u_i, нахождение u_i
Подстановка x_i в l_i, нахождение l_i
Подстановка x_i и u_i в x_i+1, нахождение x_i+1
Подсчёт суммарного интеграла потерь
* Вывод результатов на график
* Нахождение разницы между интегралами потерь, рассчитанными разными методами

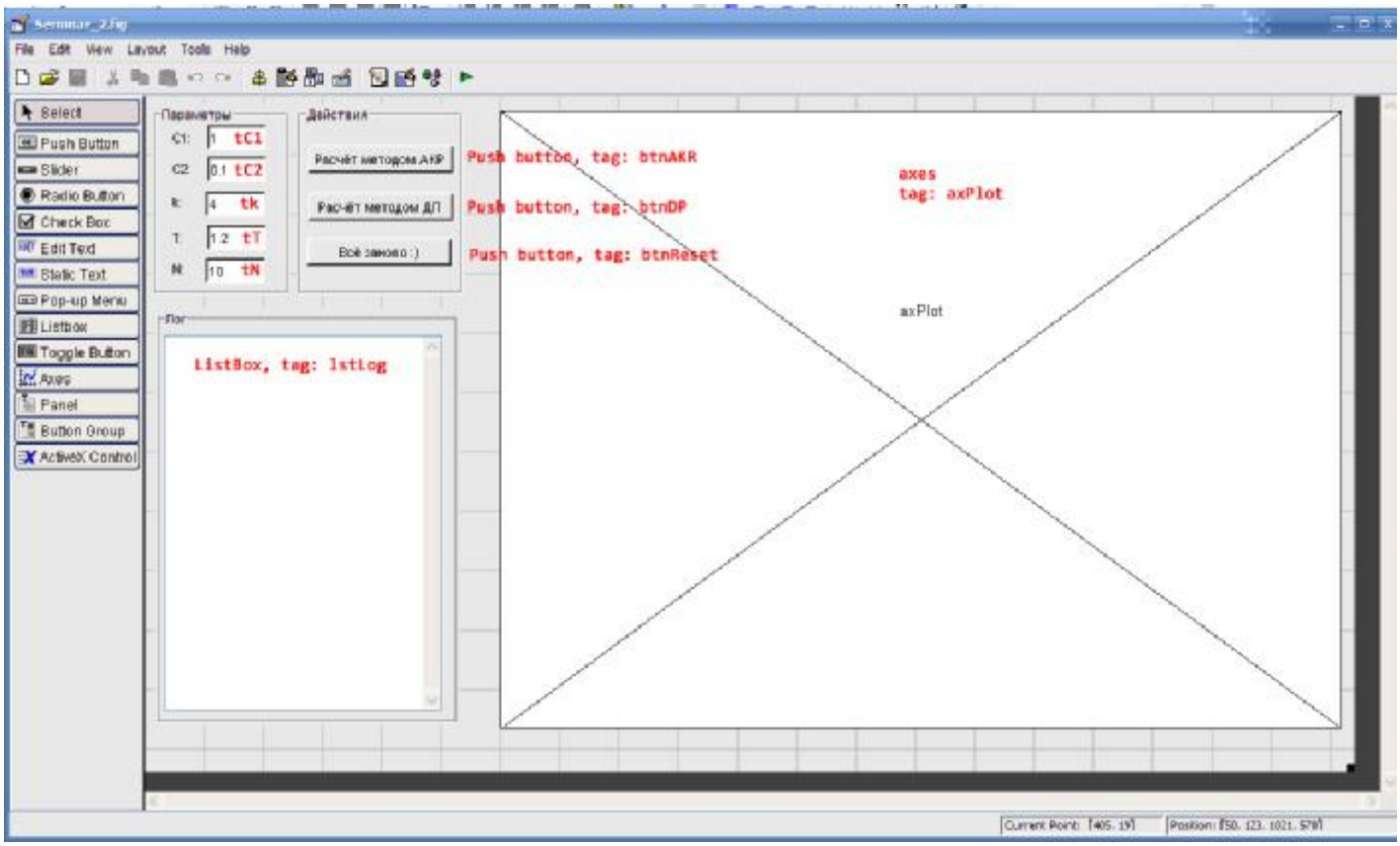
Создание макета пользовательского интерфейса

Для запуска редактора интерфейса нужно ввести команду `guide`. На экране появится окно, в котором можно создать новый, или открыть уже существующий интерфейс. Мы будем создавать интерфейс с нуля, поэтому в списке шаблонов выбираем “Blank GUI”. Ставим галочку “Save on startup as”, чтобы Matlab автоматически сохранил нашу работу. В текстовом поле нужно указать путь для нового файла, а затем нажать OK.



После этого появится окно с редактором интерфейса. В левой части окна расположена панель контролов, из которых может состоять интерфейс. Чтобы поместить элемент на форму, достаточно выбрать его из списка и щёлкнуть в том месте формы, где вы бы хотели поместить контрол. Дважды щёлкнув на элементе, вы можете вызвать редактор свойств.

Итак, создадим макет нашей будущей программы. Поместим на форму элементы и зададим их свойства так, как показано на скриншоте:



Напишем теперь функцию для расчета объекта по методу аналитического конструирования регулятора.

```
function btnAKR_Callback(hObject, eventdata, handles)
% Declare some global vars that we'll need in the future
global hLog
global I_AKR
global t_reg
global LogCnt

hLog = handles.lstLog;
LogCnt = 0;

set(hObject, 'Enable', 'off');
drawnow();
```

Сначала мы объявляем несколько глобальных переменных:

- hLog – Этой переменной мы присваиваем хэндл списка, в который выводятся сообщения по ходу расчета.
- I_AKR – значение интеграла потерь. Нам нужно сохранить это значение, чтобы потом сравнить результат с методом динамического программирования.
- T_reg – время регулирования, будет использовано позднее в расчете по методу динамического программирования.
- LogCnt – переменная, в которой хранится количество выведенных сообщений. Так как функция btnAKR_Callback вызывается самой первой, это самое подходящее место для того, чтобы инициализировать LogCnt.

В последних двух строках мы делаем кнопку неактивной и принудительно перерисовываем интерфейс.

Далее мы должны перенести значения, введенные пользователем, во внутренние переменные. Для этого используются функции get, которая возвращает указанное свойство контрола и str2double, которая преобразует строку в число. В этом блоке также инициализируется переменная t, обозначающая время (расчет будет идти от 0 до 5 секунд с шагом 0.01).

```
% Set basic variables
t = [0:0.01:5];
T = str2double(get(handles.tT, 'String'));
C1 = str2double(get(handles.tC1, 'String'));
C2 = str2double(get(handles.tC2, 'String'));
k = str2double(get(handles.tk, 'String'));
```

Как известно, выражения x(t) и u(t) зависят от корня p1, который рассчитывается по формуле:

$$p_1 = -\frac{1}{T} \cdot \sqrt{1 + \frac{C_1}{C_2} \cdot k^2}$$

В следующем блоке мы находим значение корня p1 и выводим сообщение с результатом.

```
% Compute p1 root
p1 = -(1/T) .* sqrt(1 + ((C1 / C2) .* k.^2));
szText = 'Нашли корень p1=';
szText = strcat(szText, num2str(p1));
AppendLog(szText);
```

Для преобразования численного значения p1 в строку используется функция num2str. С помощью функции strcat производится конкатенация двух строк.

После нахождения корня p1 мы можем рассчитать и вывести графики функций x(t) и u(t):

$$x(t) = k \cdot e^{p_1 \cdot t}$$

$$u(t) = (1 - \sqrt{1 + \frac{C_1}{C_2} \cdot k^2}) \cdot e^{p_1 \cdot t}$$

```
% Compute and draw x(t)
x = k .* exp(p1 * t);
AppendLog('Строим график x(t)...');
axes(handles.axPlot);
grid on;
hx = plot(t, x, 'b');
line([0 5], [0.05*x(1) 0.05*x(1)], 'LineWidth', 2, 'LineStyle', '--',
'Color', 'g');
set(hx, 'LineWidth', 2);

% Compute and draw u(t)
u = (1 - sqrt(1 + (C1 / C2) .* k .^ 2)) .* exp(p1 .* t);
AppendLog('Строим график u(t)...');
hu = plot(t, u, 'r');
set(hu, 'LineWidth', 2);
line([0 5], [0.05*u(1) 0.05*u(1)], 'LineWidth', 2, 'LineStyle', '--',
'Color', 'g');
```

С помощью функции axes мы выбрали оси для рисования текущего графика, затем вывели график x(t) и провели пунктирную линию, обозначающую пятипроцентную трубку точности. Те же манипуляции провели и с u(t).

Теперь необходимо найти время регулирования.

```
% Find regulation time
t_reg = t(find(x <= (0.05 * x(1)), 1));
szText = 'Время регулирования tp=';
szText = strcat(szText, num2str(t_reg));
AppendLog(szText);
axis([0 t_reg*1.3 u(1) x(1)]);
```

Используя функцию find, мы нашли индекс первого значения в векторе x(t), которое входит в пятипроцентную трубку точности. Очевидно, что время регулирования равно значению элемента вектора t, располагающегося по этому же индексу. После нахождения времени регулирования мы использовали функцию axis для установки масштаба графика.

После этого нам осталось лишь посчитать интеграл потерь и W*(p) по известным формулам:

$$W^* = \frac{1}{k} \cdot (\sqrt{1 + \frac{C_1}{C_2} \cdot k^2} - 1)$$

$$\int_0^\infty (C_1 \cdot x^2 + C_2 \cdot u^2) dt$$

```
% Compute the integral
AppendLog('Считаем интеграл потерь...');
tt = sym('tt');
I = C1 * ((k * exp(p1 * tt)).^2) + C2 * (((1 - sqrt(1 + (C1/C2)*k.^2)) *
exp(p1*tt)).^2);
I_AKR = int(I, tt, 0, Inf);
I_AKR = double(I_AKR);
szText = '--> Интеграл потерь I=';
```

```
szText = strcat(szText, num2str(I_AKR));
AppendLog(szText);

% Compute optimal W(p)
W_akr = (1 ./ k) .* (sqrt(1 + (C1 / C2) .* k.^2 - 1) - 1);
szText = '==> Расчёт завершён, W*(p)=';
szText = strcat(szText, num2str(W_akr));
AppendLog(szText);
set(handles.btnDP, 'Enable', 'on');
```

При расчете интеграла потерь мы впервые столкнулись с применением символьной математики в Matlab. С помощью функции sym мы создали символьную переменную tt (обычное t уже занято), которую затем использовали для записи и расчета интеграла.

Теперь перейдем к написанию функции расчета по методу динамического программирования.

Снова начнем с объявления глобальных переменных и переноса значений, введенных пользователем.

```
function btnDP_Callback(hObject, eventdata, handles)
% hObject    handle to btnDP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Declare some global vars that we'll need in the future
global hLog
global I_AKR
global t_reg
hLog = handles.lstLog;

set(hObject, 'Enable', 'off');
drawnow();

% Set basic variables
T = str2double(get(handles.tT, 'String'));
C1 = str2double(get(handles.tC1, 'String'));
C2 = str2double(get(handles.tC2, 'String'));
k = str2double(get(handles.tk, 'String'));
N = str2double(get(handles.tN, 'String'));
```

Поле этого рассчитаем некоторые вспомогательные переменные.

```
dt = t_reg / N;
t = [0:dt:(N-1)*dt];
a = 1 - (dt / T);
b = k * dt / T;
alpha = 0.05 * k / b;
beta = a / b;
```

Теперь создадим массивы символьных переменных x1..xn и u1..un.

```
AppendLog(' ');
AppendLog('=== Считаем методом ДП ===');

for i = 1:N+1
    tmpStr = strcat('x', num2str(i));
    sx(i) = sym(tmpStr);
    tmpStr = strcat('u', num2str(i));
    su(i) = sym(tmpStr);
end
```

Произведем расчет первой итерации:

```
AppendLog('Итерация №1');
% First iteration
x(N+1) = a*sx(N) + b*su(N);
u(N) = alpha - beta*sx(N);
I(N) = dt * (sx(N)^2 + u(N)^2);
sI(N) = dt * (C1*sx(N)^2 + C2*u(N)^2);
```

В массив sI будут записываться суммы интегралов потерь на каждой итерации, а в массив I – сами интегралы. Далее выполним все оставшиеся итерации.

```
% Straigt iterations
for i = N-1:-1:1
    szText = 'Итерация №';
    szText = strcat(szText, num2str(N+1-i));
    AppendLog(szText);
    sI(i) = dt*(C1*sx(i)^2 + C2*su(i)^2) + sI(i + 1);
    x(i + 1) = a*sx(i) + b*su(i);
    sI(i) = subs(sI(i), sx(i + 1), x(i + 1));
    dI = diff(sI(i), su(i));
    u(i) = solve(dI, su(i));
    sI(i) = subs(sI(i), su(i), u(i));
    I(i) = dt*(C1*sx(i)^2 + C2*u(i)^2);
end
x(1) = k;
```

В данном блоке используется цикл for со счетчиком i, который уменьшается от N-1 до 1. Внутри цикла формируется интеграл, в котором затем производятся подстановка x, рассчитанного на предыдущей итерации. Затем вычисляется производная и из полученного выражения находится u для текущей итерации.

Поле завершения всех итераций производится так называемый «обратный ход».

```
AppendLog('Обратный ход...');
% Backwards iterations
I_DP = 0;
for i = 1:N
    szText = 'Итерация №';
    szText = strcat(szText, num2str(i));
    AppendLog(szText);
    u(i) = double(subs(u(i), sx(i), x(i)));
    I(i) = double(subs(I(i), sx(i), x(i)));
    x(i+1) = subs(x(i+1), sx(i), x(i));
    x(i+1) = subs(x(i+1), su(i), u(i));
    x(i+1) = double(x(i+1));
    I_DP = I_DP + I(i);
end
```

На основе известного x(1) рассчитывается u и I для текущей итерации, а также x для следующей. После завершения цикла мы получаем векторы значений x и u. Также в цикле происходит расчет суммарного интеграла потерь. По завершение расчета на экран выводятся графики и сообщения с результатами сравнения интегралов потерь.

```
AppendLog('Расчёт завершён.');
% Plot the result
axes(handles.axPlot);
hx = stem(t, x(1:N), 'fill', 'Color', 'm');
hu = stem(t, u, 'fill', 'Color', 'm');
set(hx, 'MarkerFaceColor', 'm');
set(hx, 'LineWidth', 2);
set(hx, 'LineWidth', 2);
set(hu, 'MarkerFaceColor', 'm');
set(hu, 'LineWidth', 2);
set(hu, 'LineWidth', 2);
```

```
% Print integral
I_DP = double(I_DP);
szText = '--> Интеграл потерь I=';
szText = strcat(szText, num2str(I_DP));
AppendLog(szText);

% Print the difference
szText = '==> Разница dI=';
szText = strcat(szText, num2str(abs(I_AKR - I_DP)));
AppendLog(szText);
```

На этом написание программы можно считать завершенным. Осталось лишь написать код для кнопки btnReset и служебную функцию AppendLog, отвечающую за вывод сообщений.

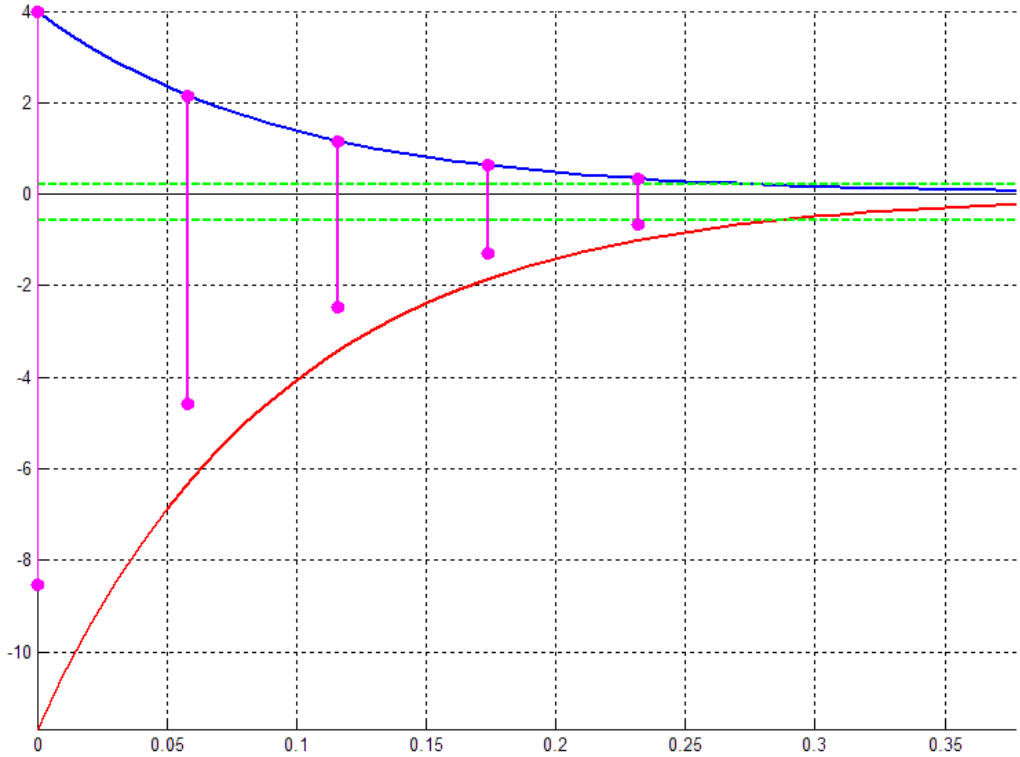
```
function btnReset_Callback(hObject, eventdata, handles)
set(handles.btnDP, 'Enable', 'off');
set(handles.btnAKR, 'Enable', 'on');
set(handles.lstLog, 'String', '');
axes(handles.axPlot);
cla();
clear();


function AppendLog(szText)
global hLog
global LogCnt
szTemp = get(hLog, 'String');
szTemp = strvcat(szTemp, szText);
LogCnt = LogCnt + 1;
set(hLog, 'String', szTemp);
set(hLog, 'Value', LogCnt);
drawnow();
```

Результат работы написанной программы

Проведём расчёт для следующих значений:

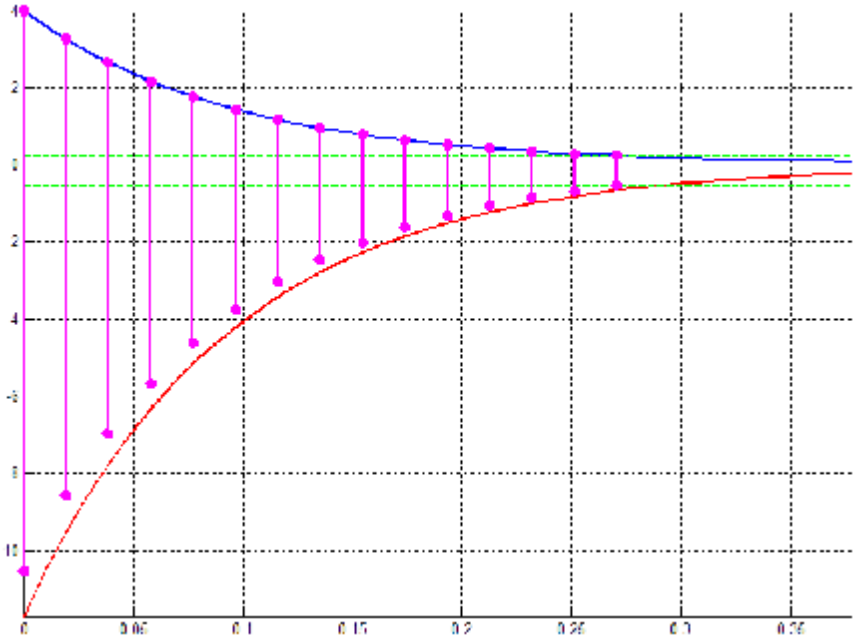
- C1 = 1
- C2 = 0.1
- k = 4
- T = 1.2

Количество итераций N = 5:



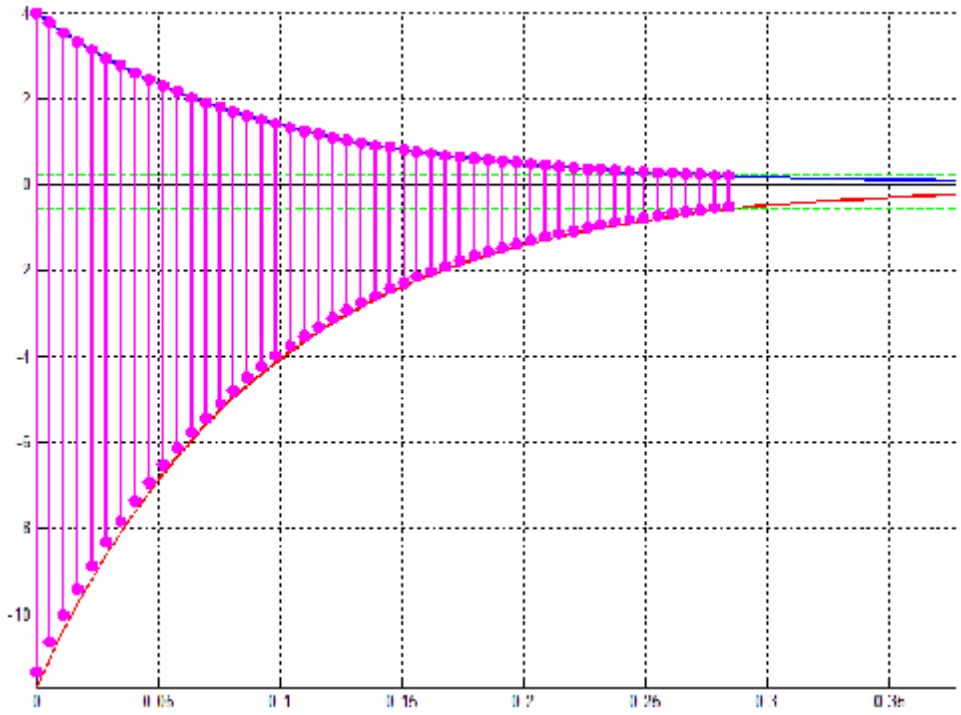
Интеграл потерь по методу АКР:	1.4026
Интеграл потерь по методу ДП:	1.8992
Разница:	0.49661

Количество итераций N = 15:



Интеграл потерь по методу АКР:	1.4026
Интеграл потерь по методу ДП:	1.5501
Разница:	0.14749

Количество итераций N = 50:



Интеграл потерь по методу АКР:	1.4026
Интеграл потерь по методу ДП:	1.4428
Разница:	0.04015

Из полученных результатов можно сделать вывод, что увеличение числа итераций приводит к значительному увеличению точности результата.