Laboratorium 7

Łukasz Janusz, grupa 3

Wstęp

Na zajęciach zapoznałem się z możliwością wykorzystania S-funkcji do tworzenia modeli w simulinku. W ramach ćwiczenia utworzyłem model symetrycznego układu hamowania samolotu na lotniskowcu korzystając z S-funkcji.

Stworzona S-funkcja

```
 function [sys, x0, str, ts] = Janusz (t, x, u,flag)
 % Type "open sfunctmpl" in command to open new S-Function
 % Save it as your own file,like "example" in this case;
 % Change the function name as your file name
 % Goto line 114, there you will define the differential eq in "function
sys=mdlDerivatives(t,x,u)"
 % Goto line 82, to define the continous states, inputs from simulink block and
output in simulink in "function [sys,x0,str,ts]=mdlInitializeSizes"
 % Goto line 94, for Initial conditions of differential eq in "function
[sys,x0,str,ts]=mdlInitializeSizes"
 % Goto line 151, where you will declare the outputs in "function
sys=mdlOutputs(t,x,u)"

 %
 % The following outlines the general structure of an S-function.
 %
 switch flag

   %%%%%%%%%%%%%%%%%%
   % Initialization %
   %%%%%%%%%%%%%%%%%%
   case 0
     [sys,x0,str,ts]=mdlInitializeSizes;

   %%%%%%%%%%%%%%%
   % Derivatives %
   %%%%%%%%%%%%%%%
   case 1
     sys=mdlDerivatives(t,x,u);

   %%%%%%%%%%
   % Update %
   %%%%%%%%%%
   case 2
```

```
    sys=mdlUpdate(t,x,u);

  %%%%%%%%%%
  % Outputs %
  %%%%%%%%%%
  case 3
    sys=mdlOutputs(t,x,u);

  %%%%%%%%%%%%%%%%%%%%%%
  % GetTimeOfNextVarHit %
  %%%%%%%%%%%%%%%%%%%%%%
  case 4
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  %%%%%%%%%%%%
  % Terminate %
  %%%%%%%%%%%%
  case 9
    sys=mdlTerminate(t,x,u);

  %%%%%%%%%%%%%%%%%%%%
  % Unexpected flags %
  %%%%%%%%%%%%%%%%%%%%
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl

%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts]=mdlInitializeSizes

%
% call simsizes for a sizes structure, fill it in and convert it to a
% sizes array.
%
% Note that in this example, the values are hard coded.  This is not a
% recommended practice as the characteristics of the block are typically
% defined by the S-function parameters.
%
```

```
sizes = simsizes;

sizes.NumContStates  = 6;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 3;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
x0  = [0 67 0 0 0 0];

%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
ts  = [0 0];

% end mdlInitializeSizes

%
%=============================================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================================
%
function sys=mdlDerivatives(t,x,u)

    h = 42;       %[m]
    m1 = 14000;   %[kg]
    m2 = 450.28;  %[kg]
    m3 = 200;     %[kg]
    k1 = 54700;   %[N]
    k2 = 303600;  %[N]

    % interpolacja
    wezlyF3 = [0 10 20 30 40 50 60 70 80 90 94 98 102 104 107 120];
```

```
    wartosciF3 = [833 400 160 320 520 520 660 830 1070 1600 2100 2800 4100 5000
9000 9000];
    funF3 = interp1(wezlyF3,wartosciF3,x(5),'pchip');
    Fb = funF3*x(6)^2;

    % zmienne stanu
    y1 = sqrt(x(1)^2 + h^2) - h;
    sin_theta = x(1)/sqrt(x(1)^2 + h^2);
    if y1 - 2*x(3) >= 0
        Fk1 = k1 * (y1 - 2*x(3));
    else
        Fk1 = 0;
    end
    if x(3) - x(5) >= 0
        Fk2 = k2 * (x(3) - x(5));
    else
        Fk2 = 0;
    end
    dx(1) = x(2);
    dx(2) = (-2 * Fk1 * sin_theta) / m1;
    dx(3) = x(4);
    dx(4) = (2 * Fk1 - Fk2) / m2;
    dx(5) = x(6);
    dx(6) = (Fk2 - Fb) / m3;
sys = [dx(1);dx(2);dx(3);dx(4);dx(5);dx(6)];

%
%=============================================================================
% mdlOutputs
% Return the block outputs.
%=============================================================================
%

function sys=mdlOutputs(t,x,u)
    h = 42;
    m1 = 14000;
    k1 = 54700;
    y1 = sqrt(x(1)^2 + h^2) - h;
    sin_theta = x(1)/sqrt(x(1)^2 + h^2);
    if y1 - 2*x(3) >= 0
        Fk1 = k1 * (y1 - 2*x(3));
    else
        Fk1 = 0;
    end
    acc = (-2 * Fk1 * sin_theta) / m1;
```

```
sys = [x(1) x(2), acc];

%
%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
%

function sys=mdlUpdate(t,x,u)
sys = [];


%
%=============================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
%=============================================================================
%

function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1;    %  Example, set the next hit to be one second later.
sys = t + sampleTime;


%
%=============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================================
%


function sys=mdlTerminate(t,x,u)

sys = [];
```

Wynik

Podsumowanie

Jak widać, uzyskany wynik jest identyczny z wynikiem uzyskanym na poprzednich zajęciach. Dzięki temu wiem, że moja funkcja działa poprawnie i nie wymaga edycji. Z perspektywy simulinka, implementacja modelu jest bardzo prosta, jednakże pierwszy raz spotkałem się z tworzeniem S-funkcji, przez co nie uważam tego za najprzyjemniejszy sposób wykonania zadania. Bardzo pomocną rzeczą okazały się być komentarze zostawione w szablonie funkcji, dzięki nim wiedziałem, gdzie jakie dane wpisać.

Modelowanie systemów dynamicznych

Laboratorium 10

Identyfikacja obiektów dynamicznych

Łukasz Janusz
Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

14 stycznia 2025

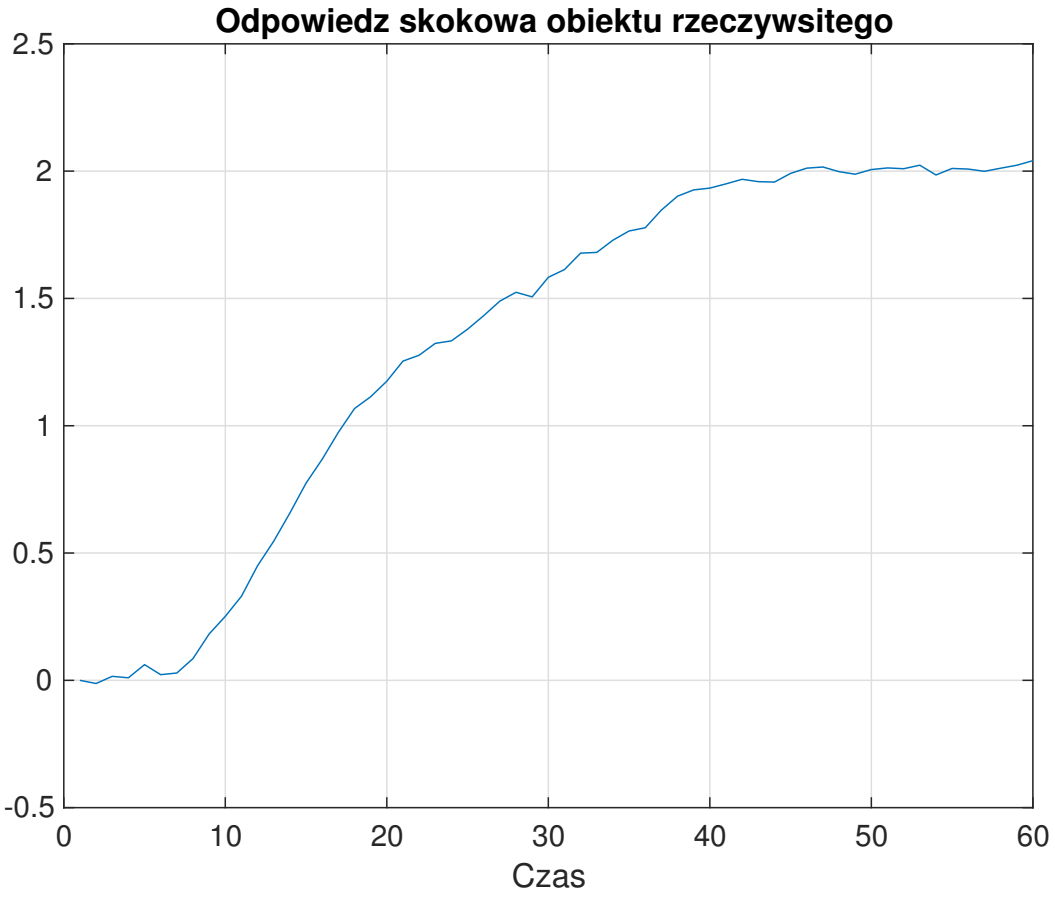
Wstęp

Celem laboratorium jest identyfikacja podanego obiektu rzeczywistego przy pomocy modeli inercyjnych I i II rzędu oraz wieloinercyjnego. W tym celu wykorzystane zostaną metody optymalizacji oraz funkcje matlaba do identyfikacji modeli.

Z trywialnych przyczyn na początku należy załadować odpowiedź skokową obiektu rzeczywistego.

```
load("obiekt.mat")\break
```

```
plot(y)
title("Odpowiedz skokowa obiektu rzeczywsitego")
xlabel("Czas")
grid on
```

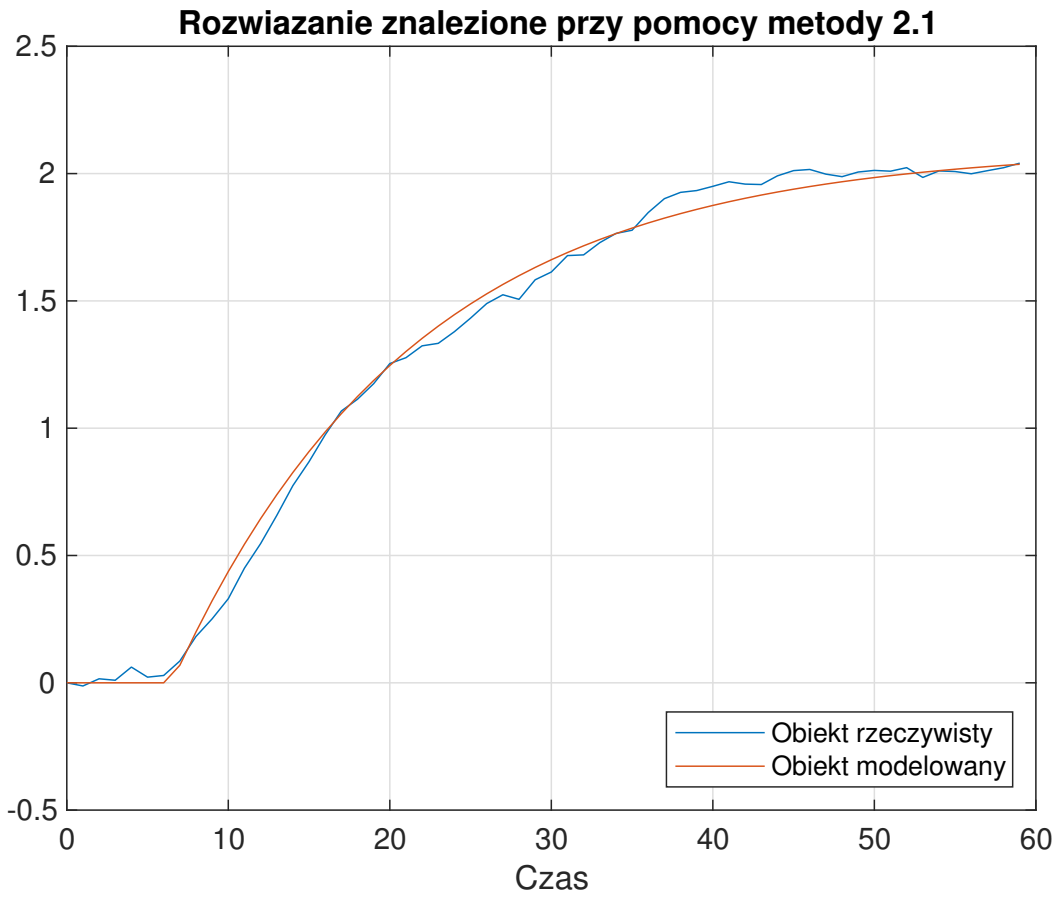


Model A - obiekt inercyjny I rzędu z opóźnieniem

W pierwszym kroku należy zidentyfikować parametry modelu inercyjnego I rzędu z opóźnieniem. W tym celu najpierw użyłem mojej wiedzy o parametrach takowego obiektu, które w przybliżeniu można odczytać z wykresu. Następnie wykorzystałem funkcję *fminsearch*, w której wywołałem napisaną przeze mnie funkcję *ident.m*, która oblicza błąd średniokwadratowy między odpowiedzią rzeczywistą

a odpowiedzią modelowaną. Dzięki temu uzyskałem numeryczne rozwiazanie problemu. Na końcu tej części porównałem wyniki obu metod używając jako kryterium wartości błędu średniokwadratowego.

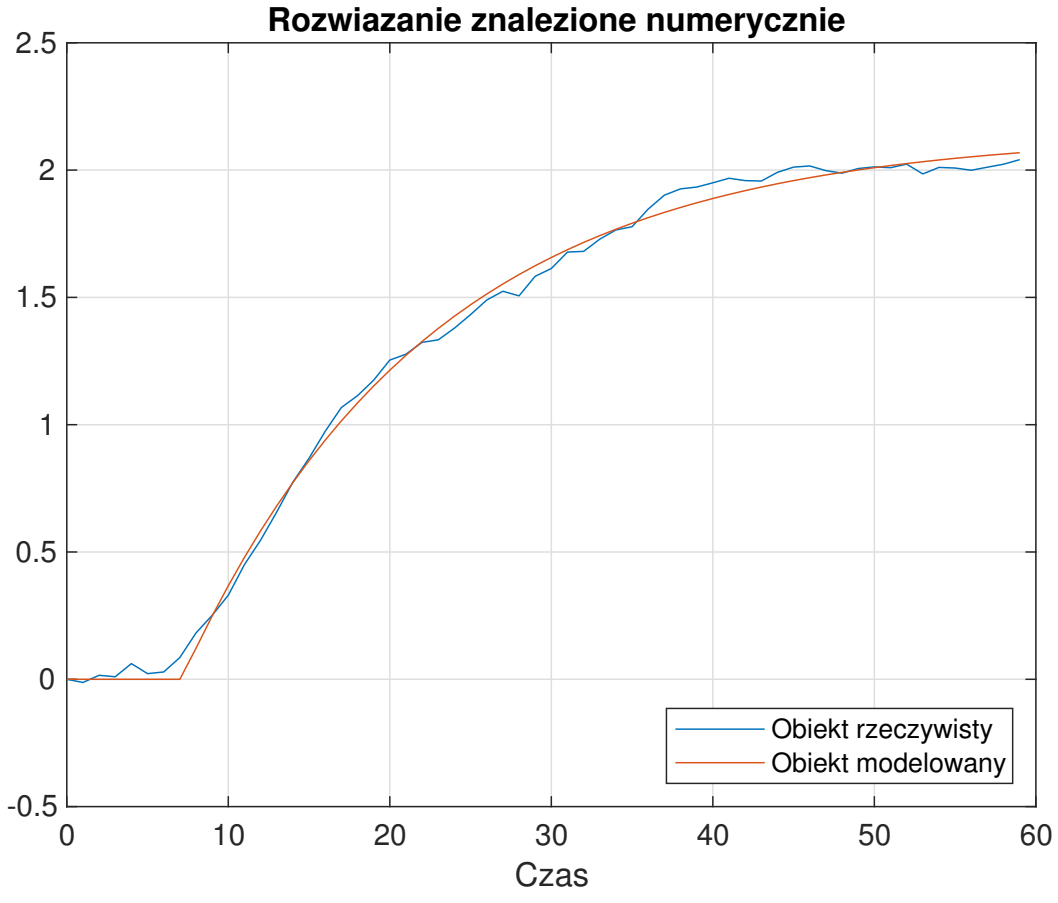
```
%Potrzebne współczynniki
%k
%T
%Theta
t = 0:length(y)-1;
T = 15;
k = 2.1;
Theta = 6.5;
obiekt = tf(k, [T,1],'OutputDelay', Theta);
res = step(obiekt, t);
plot(t,y,t,res)
grid on
legend("Obiekt rzeczywisty", "Obiekt modelowany", Location="southeast")
xlabel("Czas")
title('Rozwiazanie znalezione przy pomocy metody 2.1')
```



```
e = y - res;
blad = sum(e.^2)/length(e);
fprintf('MSE: %.4f\n', blad);
```

```
MSE: 0.0023
```

```
[parametry, blad] = fminsearch('ident',[k,T,Theta]);
k_n = parametry(1);      % 2.1422
T_n = parametry(2);      % 15.4255
Theta_n = parametry(3);  % 7.0971
obiekt = tf(k_n, [T_n,1],'OutputDelay', Theta_n);
res = step(obiekt, t);
plot(t,y,t,res)
grid on
legend("Obiekt rzeczywisty", "Obiekt modelowany", Location="southeast")
title('Rozwiazanie znalezione numerycznie')
xlabel("Czas")
```




```
fprintf('MSE: %.4f\n', blad);
```

```
MSE: 0.0015
```

Jak widać, wyniki obu metod są bardzo zbliżone, jednak metoda numeryczna daje nieco lepsze wyniki, czego można się było spodziewać.

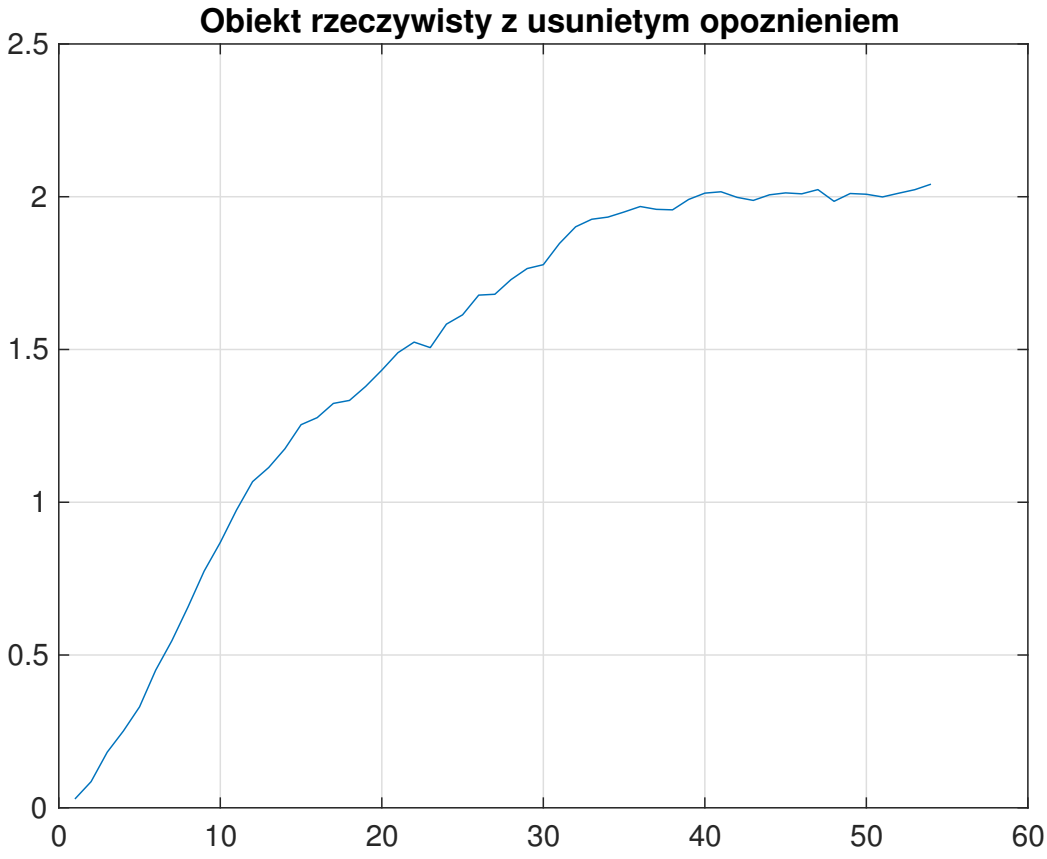
Model B - obiekt inercyjny II rzędu z opóźnieniem

Drugim modelem, przy pomocy którego zostanie zamodelowany obiekt jest model inercyjny II rzędu z opóźnieniem. W tym celu postępuję analogicznie jak w poprzednim przypadku, jednak tym razem muszę na początku znaleźć dodatkowy parametr - czas opóźnienia. W tym celu wykorzystam wiedzę z poprzedniego zadania, gdzie opóźnienie wynosiło około 7 sekund. Następnie, zgodnie z instrukcją odczytałem z wykresu wartość $y = 0,714k$, która wystąpiła dla $t_a = 22$. Na tej podstawie obliczyłem wartość $t_b = \frac{t_a}{4}$ oraz $y_{t_b} = \frac{y(t_b-0.5)+y(t_b+0.5)}{2}$. Następnym krokiem było wyznaczenie współczynnika $\frac{y_{t_b}}{k}$ i na podstawie jego wartości znalezienie w podanej tabelce wartości T_2/T_1. Ostatecznie, na podstawie tych danych oraz wzorów z instrukcji, znalazłem optymalne wartości parametrów modelu.

```
%Z poprzedniego zadania wiem, że opóźnienie wynosi ok 7 sekund
Delay = 7
```

```
Delay = 7
```

```
t_new = Delay:60;
y_new = y(t_new);
t_new = t_new - (Delay-1);
plot(t_new,y_new)
grid on
title("Obiekt rzeczywisty z usunietym opoznieniem")
```

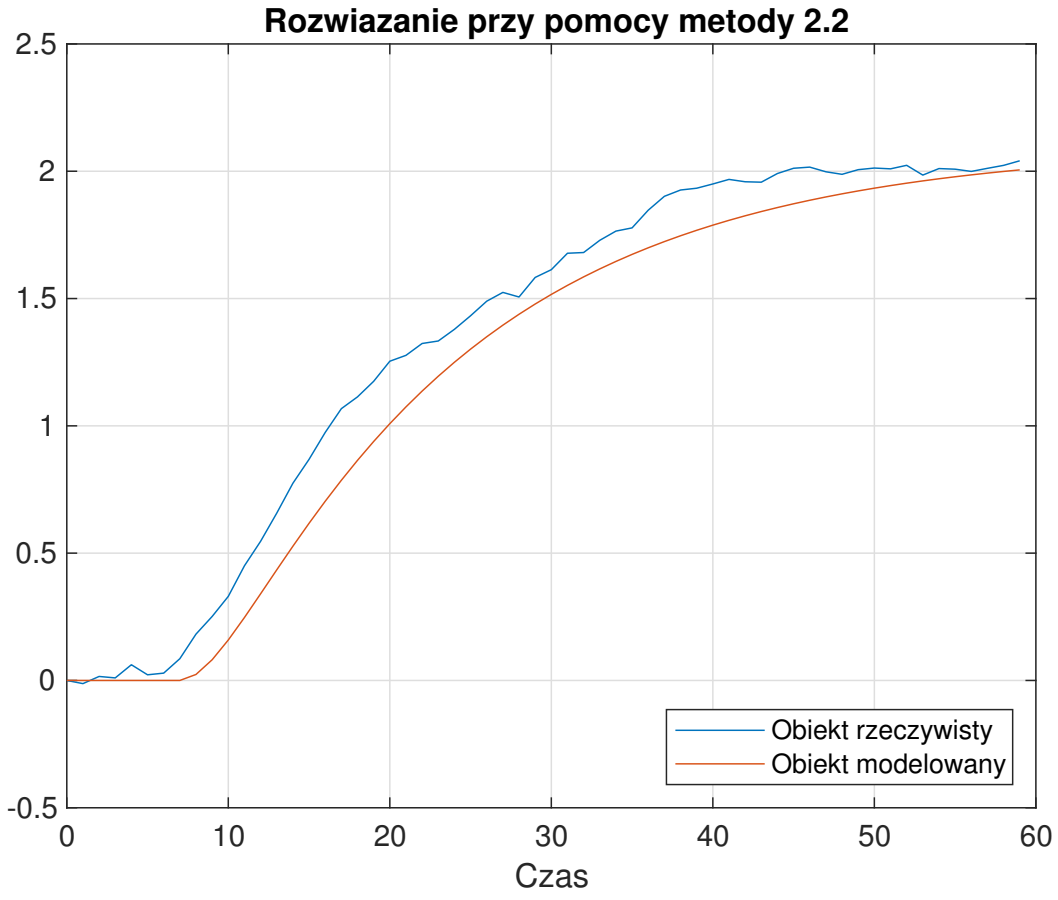



```
ta = 22;
tb = ta*0.25;
y_tb = y_new(tb-0.5)/2 + y_new(tb+0.5)/2;
wspolczynnik = y_tb/k
```

```
wspolczynnik = 0.1859
```

```
%ponieważ uzyskana wartość jest mniej więcej w połowie między danymi z
%tabelki użyję T2/T1 = 0.15
T1 = ta/(1.2*(1+0.15));
T2 = T1 * 0.15;
licz = k;
mian = [T1*T2 T1+T2 1];
obiekt2 = tf(licz,mian);
set(obiekt2, 'outputdelay', Delay)
res = step(obiekt2, t);
plot(t,y,t,res)
grid on
legend("Obiekt rzeczywisty", "Obiekt modelowany", Location="southeast")
title('Rozwiazanie przy pomocy metody 2.2')
```

```
xlabel("Czas")
```



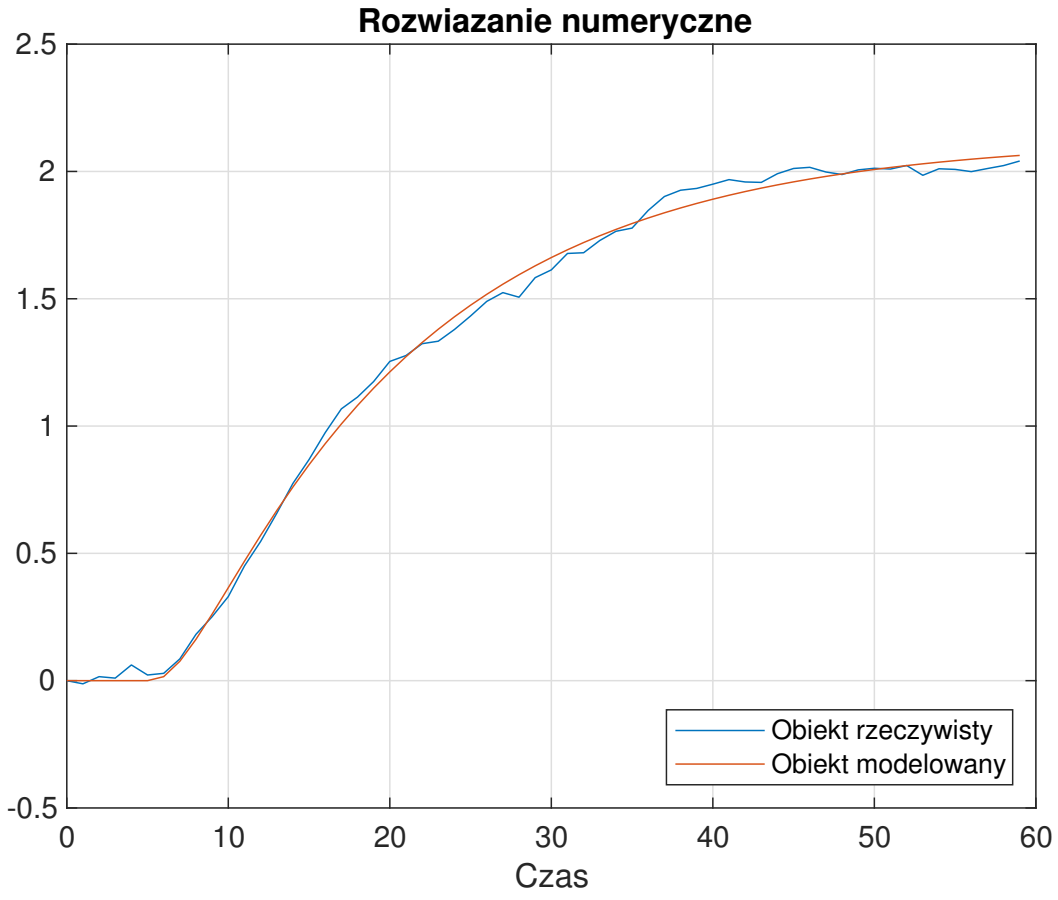

```
e = y - res;
err = sum(e.^2)/length(e);
fprintf('MSE: %.4f\n', err);
```

```
MSE: 0.0205
```

Drugim sposobem na znalezienie optymalnych parametrów modelu inercyjnego II rzędu z opóźnieniem jest wykorzystanie funkcji *fminsearch* oraz napisanej przeze mnie funkcji *ident2.m* - analogicznej jak w poprzednim zadaniu, z tą różnicą, że ma parametry przystosowane do modelu inercyjnego II rzędu. Po porównaniu wyników błędów średniokwadratowych obu metod, można zauważyć, że metoda numeryczna daje lepsze wyniki.

```
[parametry, blad] = fminsearch('ident2',[k,T1,T2,Theta]);
k = parametry(1);
T1 = parametry(2);
T2 = parametry(3);
Theta = parametry(4);
licz = k;
mian = [T1*T2 T1+T2 1];
obiekt2_n = tf(licz,mian);
set(obiekt2_n, 'outputdelay', Theta);
```

```
result = step(obiekt2_n, t);
plot(t,y,t,result)
grid on
legend("Obiekt rzeczywisty", "Obiekt modelowany", Location="southeast")
title('Rozwiazanie numeryczne')
xlabel("Czas")
```




```
fprintf('MSE: %.4f\n', blad);
```

```
MSE: 0.0013
```

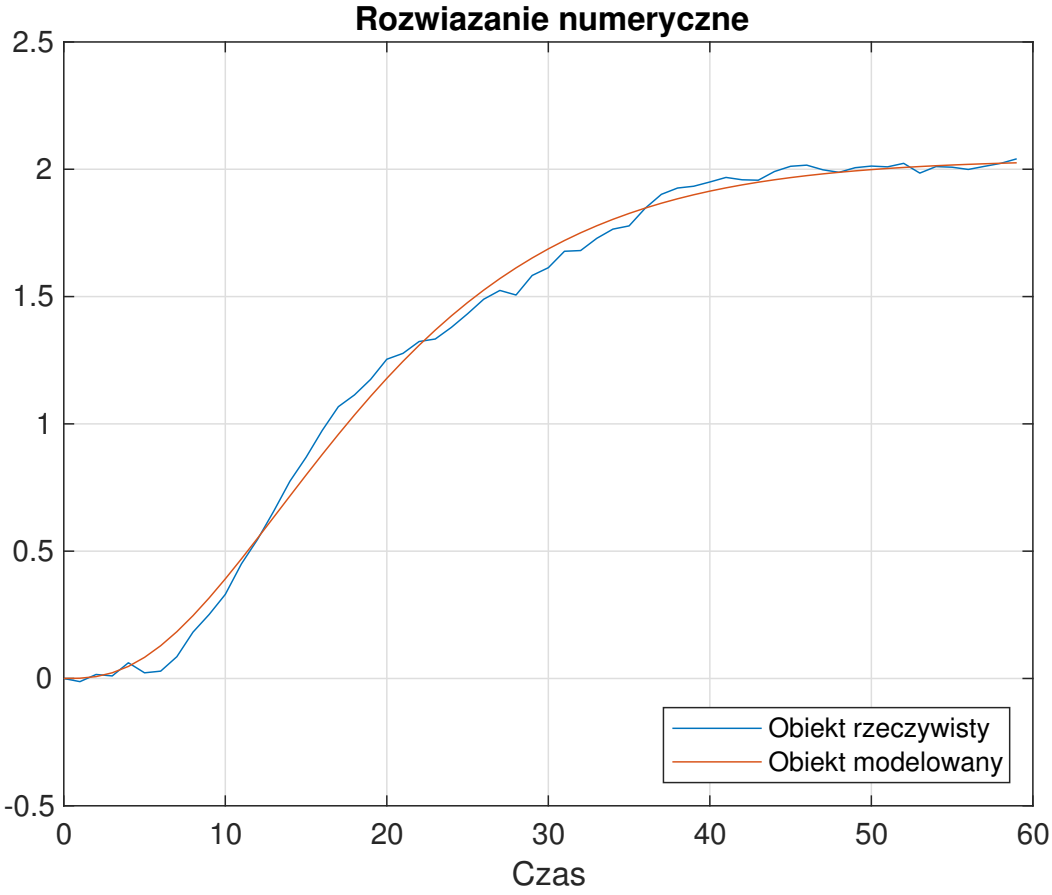
Model C - obiekt wieloinercyjny bez opóźnienia

Ostatnim modelem, który zostanie użyty do zamodelowania obiektu jest model wieloinercyjny bez opóźnienia. W tym celu postępuję analogicznie jak w poprzednich zadaniach, jednak tym razem muszę znaleźć optymalną wartość parametru n. W tym celu użyję funkcji *fminsearch* oraz napisanej przeze mnie funkcji *ident3.m*, która oblicza błąd średniokwadratowy między odpowiedzią rzeczywistą a odpowiedzią modelowaną. Po znalezieniu optymalnej wartości parametru n oraz optymalnych wartości parametrów k i T wyznaczyłem błąd średniokwadratowy, w celu porównania otrzymanego wyniku z pozostałymi.

```
% parametry k, T, n
k = 2.1;
T = 30;
n_values = 2:7;
for n = n_values
    global n;
    [~, blad] = fminsearch('ident3', [k,T]);
    disp(blad)
end
```

```
    0.0052

    0.0024

    0.0040

    0.0071

    0.0106

    0.0142
```

```
%widać, że najmniejszy błąd wystąpił dla n=3 oraz daleszemu wzrostowi n
%towarzyszył wzrost błędu
n = 3;
global n;
[parametry, blad] = fminsearch('ident3', [k,T]);
k = parametry(1);
T = parametry(2);
poles = (-1/T) .* ones(1, n);
obiekt = zpk([], poles, k/(T^n));
result = step(obiekt, t);
plot(t,y,t,result)
grid on
legend("Obiekt rzeczywisty", "Obiekt modelowany", Location="southeast")
title('Rozwiazanie numeryczne')
xlabel("Czas")
```




```
fprintf('MSE: %.4f\n', blad);
```

```
MSE: 0.0024
```

Używane funkcje

W zadaniach używałem poniższych funkcji do numerycznego wyznaczania parametrów modelu. Każda z nich wyznacza błąd średniokwadratowy między odpowiedzią rzeczywistą a odpowiedzią modelowaną. Jedynymi różnicami między poniższymi miejscami są parametry, które są przystosowane do konkretnych modeli obiektu rzeczywistego.

```
function blad = ident(X0)
k = X0(1);
T = X0(2);
Theta = X0(3);
%--------------------------------------- ------%
% tutaj kod, który będzie obliczał %
% odpowiedź skokową obiektu symulowanego %
% o takiej samej długości jak odpowiedź %
% obiektu rzeczywistego %
%--------------------------------------- ------%
load("obiekt.mat", 'y');
```

```
t = 0:length(y)-1;
obiekt = tf(k, [T,1],'OutputDelay', Theta);
y_sym = step(obiekt, t);
e = y - y_sym;
blad = sum(e.^2) / length(e);
```

```
function blad = ident2(X0)
k = X0(1);
T1 = X0(2);
T2 = X0(3);
Theta = X0(4);
%---------------------------------------- ------%
% tutaj kod, który będzie obliczał %
% odpowiedź skokową obiektu symulowanego %
% o takiej samej długości jak odpowiedź %
% obiektu rzeczywistego %
%---------------------------------------- ------%
load("obiekt.mat", 'y');
t = 0:length(y)-1;
obiekt = tf(k, [T1*T2 T1+T2 1],'OutputDelay', Theta);
y_sym = step(obiekt, t);
e = y - y_sym;
blad = sum(e.^2) / length(e);
```

```
function blad = ident3(X0)
k = X0(1);
T = X0(2);
global n;
%---------------------------------------- ------%
% tutaj kod, który będzie obliczał %
% odpowiedź skokową obiektu symulowanego %
% o takiej samej długości jak odpowiedź %
% obiektu rzeczywistego %
%---------------------------------------- ------%
load("obiekt.mat", 'y');
t = 0:length(y)-1;
poles = (-1/T) .* ones(1, n);
obiekt = zpk([], poles, k/(T^n));
y_sym = step(obiekt, t);
e = y - y_sym;
blad = sum(e.^2) / length(e);
```

Podsumowanie

Wszystkie modele, które zostały użyte do zamodelowania obiektu rzeczywistego, dały zadowalające wyniki. Wszystkie błędy średniokwadratowe są na akceptowalnym poziomie, jednak najlepsze wyniki dał model inercyjny II rzędu z opóźnieniem, co jest zgodne z moją oceną na podstawie wzrokowego porównania linii wykresu odpowiedzi skokowej obiektu rzeczywistego i modelowanego. Warto zauważyć, że metoda numeryczna daje lepsze wyniki niż metoda analityczna, co nie jest niespodzianką.

Intro

Data is the fundamental source of information in every area of life. It describes the reality surrounding us and allows us to know it better. Data can come from various sources, such as machine sensors, generators, software, databases, files and so on.

Data engineering refers to the techniques aimed at collecting, storing, managing and transforming raw data into a usable format for further analysis. This is typically a multi-step process. Before we can use the collected data, we must first import, analyze and preprocess it in the right way.

Preparing good quality data is usually time-consuming. However, it's worth investing the time to perform this. High quality data will allow for better analysis, visualization and overall understanding of the object being studied. This also enables the creation of higher quality object models, for example for machine learning purposes.

Theoretical Background

Data engineering is very closely related to the language of technical computing (LTC) workflow which is illustrated in figure 1. The presented workflow can be divided into 3 main stages: Access to data (import and collect data from various sources), Explore & Discover (analysis, processing and preparing for sharing), and Sharing results (documentation, data, code or deployment). MATLAB enables each step of the presented LTC workflow to be implemented using dedicated features, functions and interactive tools.

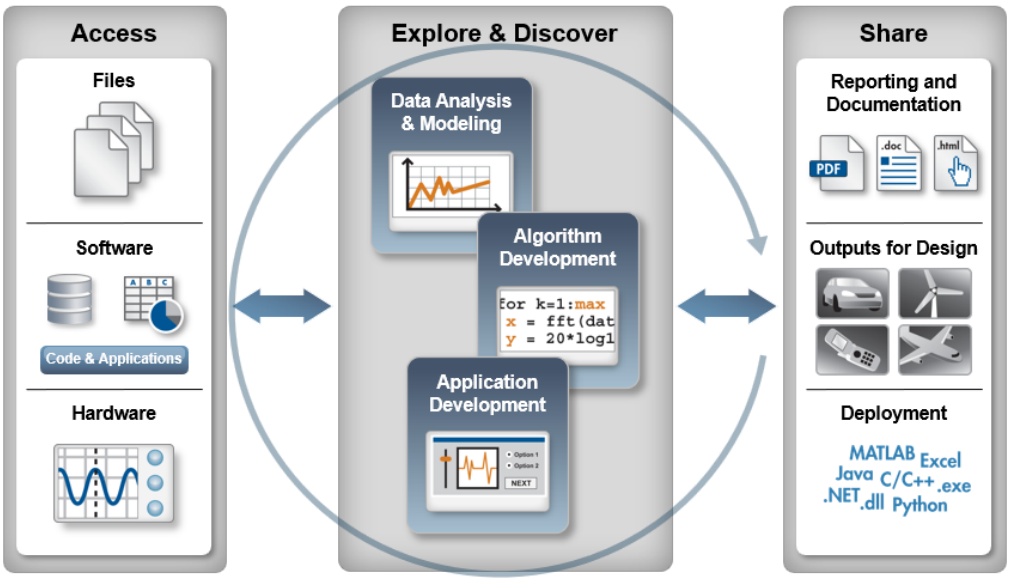


Fig. 1: Language of technical computing (LTC) workflow

Data can be stored in many formats. One of the most popular is CSV (comma-separated values) file format. In this type of file the data is organized in such a way that subsequent variables are stored in columns, while subsequent values (probes) are stored in rows and are usually separated by commas.

Data preprocessing and analyzing is one of the most important part of data engineering. In this part the following tasks are usually performed:

- data visualization and content analysis,
- improving data quality by filling in missing data, cleaning outliers and removing any noise or trends,
- detecting and assessing relationship between data and selecting the most valuable variables.

Data visualization, content analysis, variables relationship and improvement of their quality are based largely on statistical analyses, from basic computations such as min, max, average or standard deviation, to more advanced techniques such as correlation, interpolation, data prediction and modeling. When filling in missing data that depend on several other variables, regression models can be used. One of the common used models is the linear regression model. Filling in missing data can prevent an entire data record from being deleted due to a single missing value.

One of the most convenient ways to store and work with data is the tabular format. Each column corresponds to a specific variable, and the rows contain their values. A single table can contain various types of data, such as numeric, string, logical or categorical. Working with tables also allows you to easily manage data, export it and share with others.

Goal

Import the external data, process it according to the given requirements, save in tabular format and export to a shareable file.

Provided files

DataEngineering_input.csv – file containing data for import and processing.

Task description

The whole task is divided into two parts, i.e. Part I and Part II, described below. The data you will process according to the requirements given below must be finally saved as a **table**.

Follow the steps given below to complete the task.

Part I

1. Import data from an external comma-separated values (CSV) file named ***DataEngineering_input.csv*** into the MATLAB workspace as a **table**.
2. The following table variables (columns): *MPG*, *Acceleration* and *Weight* must not contain any outliers. When filling in outliers in these variables use one of the common MATLAB methods: *center, clip, previous, next, nearest, linear* or *spline*. When detecting outliers and verifying the filling in results, use the **default** detection method in MATLAB. Do not fill possible outliers in any other variables.
3. There are some missing data in the table. Firstly, fill in missing data in *Horsepower* variable. When filling in missing data use one of the common MATLAB methods:

previous, next, nearest, linear, spline, pchip or *makima*, but try to use the most appropriate method to the characteristics of the *Horsepower* values (e.g. linear, non-linear). To obtain the correct result reorganize the data in the *Horsepower* **in a simple way before filling it in**. Hint: *Horsepower* and *Displacement* variables are strongly positive correlated. For now, **do not** fill in possible missing data in the remaining variables. You will address with this matter later.

4. The *Region* variable must have 3 region types: *Europe*, *Japan* or *USA*.
5. The table should contain a new variable called *Displacement_ccm* with values in cubic centimeters calculated from the data in the *Displacement* variable. Use 1 inch = 2,54 cm for calculation.
6. The table should also contain a new variable named *MPG_Pred*. This variable should contain *true* entries in these rows where the *MPG* variable has *NaN* entries, and *false* entries in these rows where the *MPG* variable has numeric values.
7. The table should not contain the *Time_period* variable.
8. The table must contain the data processed with respect to the recommendations given above and has variables arranged in the following order and type:
   Region – type 'categorical'
   Origin – type 'cell'
   Manufacturer – type 'cell'
   Model_Year – type 'double'
   Horsepower – type 'double'
   Displacement – type 'double'
   Displacement_ccm – type 'double'
   MPG – type 'double'
   MPG_Pred – type 'logical'
   Acceleration – type 'double'
   Weight – type 'double'
   Cylinders – type 'double'
   The table size should be: 402 rows and 12 variables (columns).
9. The table cannot contain any missing data **except** *MPG* variable – this will be the aim of the Part II of the task.
10. The processed data must be saved in a **table** named: ***DataEng_output_table***

Part II

There are still some missing data in the *MPG* variable. Since this variable depends on several factors, its values can be predicted using a linear regression model.

1. Create a **linear regression model** fit to the input data using:
   - *Displacement*, *Weight* and *Model_Year* as the predictors,
   - *MPG* as the response data.

   When creating a linear regression model use the MATLAB **default model specification** input argument.

2. Using created linear regression model **predict** the missing values in the *MPG* using default input arguments, i.e. model and **new** predictors. Enter predicted *MPG* values in the table instead of missing values.

Finally, the team must save the processed data stored in the table ***DataEng_output_table*** to a file named ***DataEngineering_output.mat*** (MAT-file format).
A team can complete only Part I of the task and save the processed data to a table ***DataEng_output_table*** in the file ***DataEngineering_output.mat***, but then they may only receive a percentage of the points, as described in the TASK EVALUATION section below.
If a team completes both Part I and Part II of the task, they should save the results in the same table and file and then they can receive the maximum number of points.
The data to be processed is based on the MathWorks *carbig.mat* dataset containing measurements of cars from 1970–1982.

Task evaluation

Regarding the Part I of the task, the following will be assessed:

- whether the table is saved in the file with the required name and format,
- whether the processed data is saved in the table with the required name and size,
- correctness of the names and types of all variables, and their proper order in the table,
- cleaning outliers in the *MPG*, *Acceleration* and *Weight* variables,
- no missing data except MPG variable,
- proper fill in the missing data in the *Horsepower* with a MAPE error of no more than 1% compared to the reference data,

additionally, correctness of the data in the following variables:

- *Region*: 3 categories – *Europe*, *Japan* or *USA*,
- *Displacement_ccm*: a MAPE error of no more than 2% compared to the reference data,
- *MPG_Pred*: 4 *true* entries, other entries – *false*.

Regarding the Part II of the task, the following will be assessed:

- whether all predicted values in *MPG* do not exceed an absolute error of 10% compared to the reference values.

- If one or more of the requirements of Part I are not met, the team receives **0 points**.
- If all the requirements of Part I are met, the team receives **5 points**.
- If the Part I is done correctly, the team receives an additional **2 points** for correctly performing the Part II.
- In total the team can receives **7 points**.

Files to be uploaded

DataEngineering_output.mat – a MAT-file with *DataEng_output_table* table.

Useful documentation:

https://www.mathworks.com/help/matlab/ref/isoutlier.html
https://www.mathworks.com/help/matlab/ref/filloutliers.html
https://www.mathworks.com/help/matlab/ref/fillmissing.html
https://www.mathworks.com/help/stats/fitlm.html
https://www.mathworks.com/help/stats/linearmodel.predict.html