

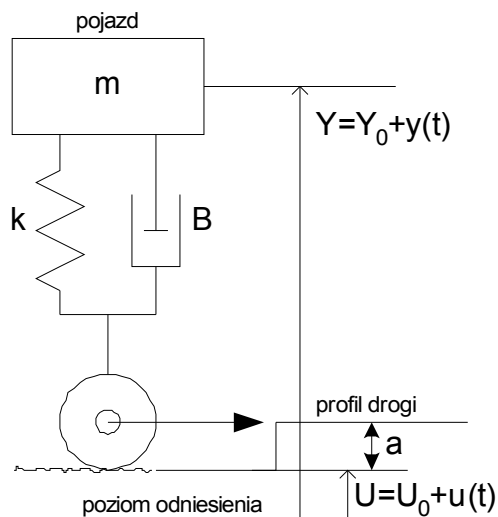
MODELOWANIE I SYMULACJA

Pomoce projektowe

Przykład tworzenia modelu z wykorzystaniem programu MATLAB

1. Tworzenie maski systemu w programie SIMULINK

1.1 Schemat modelu



Rys. 1. Uproszczony model zawieszenia pojazdu

1.2 Równanie modelu:

$$m \frac{d^2 y(t)}{dt^2} + B \frac{dy(t)}{dt} + ky(t) = B \frac{du(t)}{dt} + ku(t) \quad (1)$$

Podstawienie: $y_1 = y$; $y_2 = y'_1$; $y'_2 = y''_1 = y''$ prowadzi do równań:

$$\begin{aligned} my'_2 + By_2 + ky_1 &= B \frac{du(t)}{dt} + ku(t) \\ y'_2 + \frac{B}{m} y_2 + \frac{k}{m} y_1 &= \frac{B}{m} \frac{du(t)}{dt} + \frac{k}{m} u(t) \\ y'_2 &= -\frac{k}{m} y_1 - \frac{B}{m} y_2 + \frac{B}{m} \frac{du(t)}{dt} + \frac{k}{m} u(t) \end{aligned} \quad (2)$$

1.3 Opis w postaci równań stanu:

Powyżej wyprowadzone równanie nie może zostać bezpośrednio przedstawione w postaci normalnej równań stanu. Aby takie równanie zapisać wektorowo należy wprowadzić w modelu dodatkowe wejście (odpowiadające pochodnej $du(t)/dt$).

Podstawienie: $x_1 = y$; $x_2 = x'_1$ pozwala zapisać równania stanu:

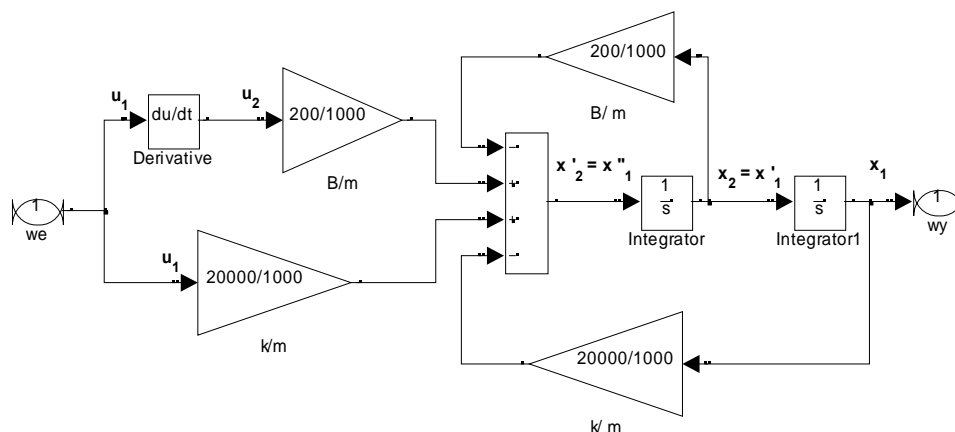
$$\begin{cases} x'_1 = x_2 \\ x'_2 = -\frac{k}{m}x_1 - \frac{B}{m}x_2 + \frac{B}{m}u_2 + \frac{k}{m}u_1 \end{cases} \quad (3)$$

W zapisie wektorowym:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{B}{m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \frac{B}{m} & \frac{k}{m} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix},$$

przy czym sygnał wejściowy u_2 jest pochodną sygnału u_1 .

1.4 Konstrukcja modelu w SIMULINK



Rysunek 2. Schemat symulacyjny modelu (nakładka SIMULINK)

1.5 Maskowanie systemu

W nakładce SIMULINK należy wprowadzić powyższy schemat. W bloki Gain wprowadzić zmienne wg opisów pod tymi blokami (np. b/m). Po zaznaczeniu wszystkich obiektów modelu (**Ctrl-A**), wywołać polecenie **Edit/Create Subsystem**. Zaznaczyć utworzony blok Subsystem. Wywołać polecenie **Edit/Mask Subsystem**. Po wyświetleniu się okna dialogowego, przejść na zakładkę **Initialization**. W polu **Mask Type** wpisać: Model zawieszenia pojazdu. Kliknąć na przycisk **Add**, w polu **Prompt** wpisać: Masa pojazdu [kg]. W polu **Variable** wpisać nazwę zmiennej modelu: m (masa pojazdu). Kliknąć na polu **<<end of parameters list>>** a następnie ponownie na **Add**. W polu **Prompt** wpisać: Współczynnik sprężystości zawieszenia k [N/m] a w polu **Variable** wpisać: k. Dodać kolejną zmienną (w sposób podany powyżej) wpisując **Prompt**: Współczynnik tłumienia zawieszenia B [Ns/m], **Variable**: b. Do wejścia zamaskowanego bloku podłączyć wymuszenie skokowe. Ustawić **Step time**: 0. **Final value**: 0.05. Dwukrotnie kliknąć na zamaskowanym bloku. Przeprowadzić symulację dla przykładowych wartości parametrów:

Wielkość	Zmienna modelu	Wartość	Jednostka
Masa pojazdu	m	1000	kg
Współczynnik sprężystości	k	20 000	N/m
Współczynnik tłumienia	B	200, 5000, 9000	Ns/m

2. Rozwiązanie równań różniczkowych z wykorzystaniem mechanizmu ODE (Ordinary Differential Equations solver)

Aby używać ODE dla równań wyższych rzędów, należy zastąpić równanie wyższego rzędu układem równań rzędu pierwszego. Równanie:

$$y^{(n)} = f(t, y, y', \dots, y^{(n-1)}), \quad (4)$$

możemy zastąpić układem równań przez podstawienie:

$$y_1 = y, y_2 = y', \dots, y_n = y^{(n-1)}. \quad (5)$$

Powstaje układ równań:

$$\begin{cases} y'_1 = y_2 \\ y'_2 = y_3 \\ \dots \\ y'_n = f(t, y_1, y_2, \dots, y_n) \end{cases} \quad (6)$$

2.1 Przykład: Równanie Van Der Pola

$$y''_1 = -\mu(1 - y_1^2)y'_1 + y_1. \quad (7)$$

Rozwiązanie:

$$\begin{cases} y'_1 = y_2 \\ y'_2 = \mu(1 - y_1^2)y_2 - y_1 \end{cases} \quad (8)$$

2.1.1 Zapis ODE.

```
function dydt = vdp1(t,y)
dydt = [y(2); (1-y(1)^2)*y(2)-y(1)] %wektor prawych
                                     %stron
```

2.1.2 Wywołanie solwera ODE.

Przedział czasu: 0 – 20s, Warunki początkowe: $y(0) = 2$; $y'(0) = 0$

```
[t,y] = ode45('vdp1', [0 20], [2;0]);
```

2.1.3 Wyniki.

```
plot(t,y(:,1), '- ', t,y(:,2), '--')
title('Rozwiązanie równania van der Pola, \mu=1');
xlabel('czas t');
ylabel('rozwiązanie, y');
legend('y_1', 'y_2');
```

2.1.4 Przekazywanie dodatkowych parametrów do funkcji ODE.

Uogólnione rozwiązanie równania (dla zmiennego μ)

```
function dydt = vdp1(t,y,options,mi)
dydt = [y(2); mi*(1-y(1)^2)*y(2)-y(1)];
```

2.1.5 Wywołanie solwera.

Aby przekazać parametr μ do funkcji ODE należy wpisać go po parametrze **options** solwera:

```
[t,y] = ode45('vdp1',tspan,y0,[],mu);
```

2.1.6 Rozwiązanie dla $\mu = 1000$ (równanie sztywne – stiff).

```
%Nieprawidłowy solwer
```

```
[t,y] = ode45('vdp1',[0 3000],[2;0],[1],1000);
```

```
%Wywołanie prawidłowe
```

```
[t,y] = ode15s('vdp1',[0 3000],[2;0],[1],1000);
```

W zagadnieniach równań sztywnych, rozwiązania mogą zmieniać się w bardzo krótkiej, w porównaniu z krokiem całkowania, skali czasu ale interesuje nas rozwiązanie w dużo dłuższej skali czasowej. Metody dla równań niesztywnych nie są efektywne dla kroków całkowania w przedziałach wolnych zmian rozwiązania gdyż kroki te są odpowiednio małe by zapewnić rozwiązanie dla możliwie najszybszych jego zmian.

3. Algorytmy numeryczne rozwiązywania równań stanu

3.1 Algorytmy Rungego – Kuty

$x(t)$ – rozwiązanie dokładne równania różniczkowego

$$\dot{x} = f(x, t) \quad x(t_0) = x_0 \quad (9)$$

Rozwijamy $x(t)$ w otoczeniu punktu $t=t_0$:

$$x(t_n + h) = x(t_n) + h \dot{x}(t_n) + \frac{1}{2} h^2 \ddot{x}(t_n) + \dots + \frac{1}{p!} h^p x^{(p)}(t_n) + O(h^{p+1}), \quad (10)$$

$x^{(i)}(t_n)$ – i-ta pochodna; $O(h^{p+1})$ – lokalny błąd obcięcia; p – rząd rozwinięcia. Ponieważ:

$\dot{x} = f(x, t)$, więc:

$$x_{n+1} = x_n + h f(x_n, t_n) + \frac{1}{2} h^2 f'(x_n, t_n) + \dots + \frac{1}{p!} h^p f^{(p-1)}(x_n, t_n) + O(h^{p+1}), \quad (11)$$

gdy $p = 1$ – algorytm ekstrapolacyjny Eulera:

$$x_{n+1} = x_n + h f(x_n, t_n). \quad (12)$$

Jeśli $p > 1$ wymagane jest określenie pochodnych funkcji $f(t_n, x_n)$. Runge i Kutta wprowadzili funkcję aproksymującą składniki wyższego rzędu w rozwinięciu Taylora. Funkcja jest tak dobrana aby rząd dokładności był taki sam. Najczęściej stosuje się:

3.1.1 Algorytm numeryczny Rungego – Kuty II rzędu.

$$\begin{aligned} k_1 &= h f(x_n, t_n) \\ k_2 &= h f\left(x_n + \frac{1}{2} k_1, t_n + \frac{1}{2} h\right) \\ x_{n+1} &= x_n + (1-a) k_1 + a k_2 + O(h^3) \end{aligned} \quad (13)$$

3.1.2 Algorytm Heuna – zmodyfikowany algorytm trapezów ($a = \frac{1}{2}$).

$$\begin{aligned} k_1 &= h f(x_n, t_n) \\ k_2 &= h f(x_n + k_1, t_n + h) \\ x_{n+1} &= x_n + \frac{1}{2} k_1 + \frac{1}{2} k_2 + O(h^3) \end{aligned} \quad (14)$$

3.1.3 Algorytm Rungego – Kuty IV rzędu.

$$\begin{aligned}
 k_1 &= hf(t_n, x_n) \\
 k_2 &= hf\left(t_n + \frac{1}{2}h, x_n + \frac{1}{k_1}\right) \\
 k_3 &= hf\left(t_n + \frac{1}{2}h, x_n + \frac{1}{2}k_2\right) \\
 k_4 &= hf(t_n + h, x_n + k_3) \\
 x_{n+1} &= x_n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 + O(h^5)
 \end{aligned} \tag{15}$$

Rozwiązanie problemu początkowego:

$$y' = f(t, y) \quad a \leq t \leq b \quad y(a) = \alpha \tag{16}$$

w $(N+1)$ równomiernie rozłożonych punktach w przedziale $[a, b]$ z warunkiem początkowym $y(0) = \alpha$, realizuje następujący algorytm:

3.1.4 Zapis algorytmu Rungego – Kuty w postaci pseudokodu.

```

WEJŚCIE a, b, N, α
KROK 1: PRZYPISZ h = (a-b)/N ;
           t = a ;
           w = α ;

WYJŚCIE (t, w)
KROK 2: DLA i = 1, 2, ..., N POWTARZAJ KROKI 3-5
KROK 3: PRZYPISZ K1 = hf(t, w) ;
           K2 = hf(t+h/2, w+K1/2) ;
           K3 = hf(t+h/2, w+K2/2) ;
           K4 = hf(t+h, w+K3)
KROK 4: PRZYPISZ w = w+(K1 + 2K2 + 2K3 + K4)/6 ;
           PRZYPISZ t = a+ih ;
KROK 5: WYJŚCIE (t, w)
KROK 6: STOP
    
```

4. MATLAB GUI (Graphical User Interface)

4.1 Programowanie obiektowe w języku MATLAB

Przykład tworzenia obiektów graficznych. Kod oblicza wartość funkcji i tworzy trzy obiekty graficzne wykorzystując operacje na właściwościach graficznych.

```

[x,y] = meshgrid([-2:.4:2]);
Z = x.*exp(-x.^2-y.^2);
fh = figure('Position',[350 275 400 300],...
           'Color','w');
ah = axes('Color',[.8 .8 .8], 'XTick',...
          [-2 -1 0 1 2], 'YTick',[-2 -1 0 1 2]);
sh = surface('XData',x, 'YData',y, 'ZData',Z,...
           'FaceColor',get(ah, 'Color')+.1,...
           'EdgeColor','k', 'Marker','o',...
           'MarkerFaceColor',[.5 1 .85]);
    
```

Do ustawiania i odczytywania właściwości obiektów służą funkcje: `set()` i `get()`. Podstawowa składnia dla ustawiania wartości właściwości istniejącego obiektu:

```
set(object_handle, 'PropertyName', PropertyValue);
```

Do uzyskiwania aktualnej wartości właściwości obiektu:

```
returned_value = get(object_handle, 'PropertyName');
```

Uwaga: Nazwy właściwości zawsze zamknięte są w cudzysłowie (pojedynczym)

object_handle – uchwyt obiektu (najczęściej uzyskiwany podczas tworzenia obiektu).
Jest on zwracany przez funkcję użytą do utworzenia obiektu.

PropertyName – nazwa właściwości (można ją uzyskać za pomocą `get(object_handle)`).

PropertyValue – wartość właściwości (domyślnie dotyczy `PropertyName`).

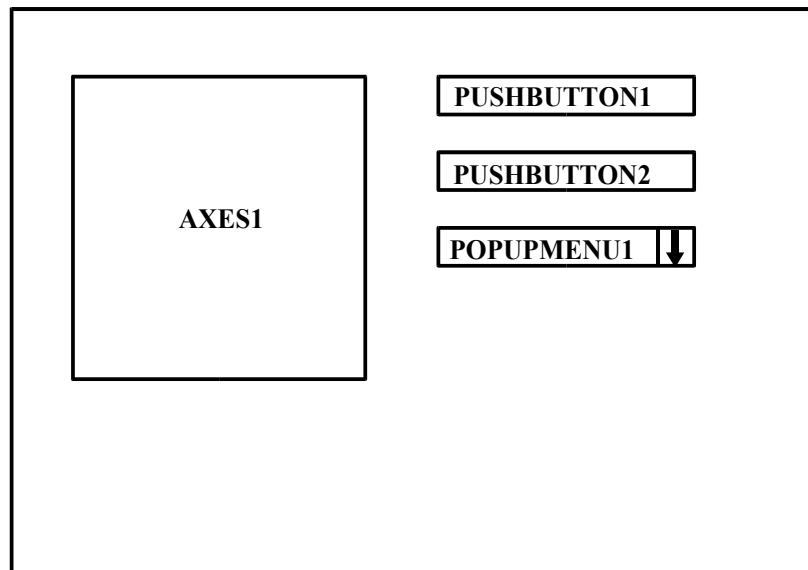
Funkcja `findobj()` znajduje obiekty graficzne i zwraca ich uchwyty.

Przykład:

```
h =findobj(gca,'Type','Line')
```

Znajduje wszystkie istniejące obiekty (linie) w aktualnym obiekcie `axes` (układ współrzędnych, osie) i zwraca ich uchwyty do wektora `h`.

4.2 Przykład tworzenia GUI w Matlabie



Rysunek 3. Widok projektowanego interfejsu użytkownika

4.2.1 Uruchomienie środowiska projektowego (GUIDE – Graphical User Interface Development Environment).

W lini poleceń wpisać:

```
guide
```

4.2.2 Ustawianie rozmiaru okna graficznego (obiekt figure).

1. Wybrać Property Editor z okna Guide Control Panel.
2. Wybrać właściwość Units i z listy rozwijalnej wybrać inches.
3. Wybrać właściwość Position, wpisać 4 jako szerokość (trzeci element wektora) i 4 jako wysokość (czwarty element wektora).
4. Zmienić jednostki na characters.

4.2.3 Dodawanie komponentów.

1. Z palety obiektów (New Object Palette) wstawić do utworzonego okna : dwa przyciski (push-buttons), tekst statyczny (static text), menu rozwijalne (pop-up menu), osie (axes).

2. Ustawić rozmiar układu współrzędnych (axes) na: 2 cale x 2 cale.

4.2.4 Wyrównywanie obiektów.

1. Zaznaczyć przyciski (pushbuttons) klikając na każdym z nich z naciśniętym klawiszem Ctrl.
2. Wybrać narzędzie Alignment Tool z Guide Control Panel.
3. Wybrać wyrównanie do lewej oraz ustawić 20 pikseli pomiędzy każdym przyciskiem.

4.2.5 Programowanie GUI.

1. Ustawić tytuł aplikacji wyświetlany w belce tytułowej okna graficznego (figure): Moje pierwsze GUI.
2. Ustawić właściwość NumberTitle na: 'off'.
3. Ustawić właściwość String przycisków pushbutton odpowiednio.

Pushbutton1: Powierzchnia

Pushbutton2: Siatka

1. Ustawić właściwość String obiektu pop-up menu na: peaks | membrane | sinc.
2. Uruchomić Callback Editor z Guide Control Panel.
3. Zaznaczyć pole Show Object Browser.

4.2.6 Wybrać **Pushbutton1**, (powierzchnia) dodać kod:

```
popup=findobj(gcf,'Tag','PopupMenu1')
val=get(popup,'Value');
p=peaks(35);
m=membrane;
[x,y]=meshgrid(-8:.5:8);
r=sqrt(x.^2+y.^2+eps);
s=sin(r)./r;
switch val
case 1
    surf(p);
case 2
    surf(m);
case 3
    surf(s);
end
```

4.2.7 Wybrać **Pushbutton2**, (siatka) dodać kod:

```
popup0=findobj(gcf,'Tag','PopupMenu1')
val=get(popup,'Value');
p=peaks(35);
m=membrane;
[x,y]=meshgrid(-8:.5:8);
r=sqrt(x.^2+y.^2+eps);
s=sin(r)./r;
switch val
case 1
    mesh(p);
case 2
    mesh(m);
case 3
    mesh(s);
end
```