1.3. Proste przykłady wykorzystania Scicosa

1.3.1. Generacja sinusoidy

Spotkanie z Scicosem rozpoczniemy od bardzo prostego przykładu generowania funkcji $\sin(x)$ i wyświetlania jej w oknie graficznym.

Tworzenie schematu połączeń generatora

Po uruchomieniu w menu *Applications* aplikacji Scicos otrzymujemy nowe okno służące do budowy aplikacji Scicosa. W obrębie proponowanego menu wybieramy *Edit* a następnie Paletts.

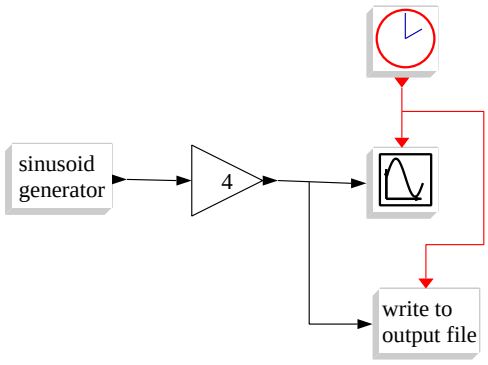
Następnie w sześciu krokach budujemy schemat generatora.

1. Spośród 11 dostępnych palet wybieramy Sources (rys. 1.1) a następnie wybieramy blok generujący sinusoidę (`sinusoid generator`).
2. Wybieramy paletę *Linear* (rys. 1.3) i pobieramy z niej blok wzmacniacza (`Gain`).
3. Wybieramy paletę *Sinks* (rys. 1.2) i pobieramy z niej blok oscyloskopu (`Scope`) pozwalający na wykreślenie generowanego przebiegu.
4. Z palety *Sinks* wybieramy również blok `write to file` pozwalający zapisać w zbiorze plikowym wygenerowane wartości funkcji sinus,
5. Wybieramy paletę *Events* (rys. 1.5) a z niej blok stopera (`Activation clock`) .
6. Posługując sie opcją `link` z menu *Edit*, łączymy odpowiednie wejścia i wyjścia poszczególnych bloków, uzyskując schemat pokazany na rys. 1.11.

Ustawiania parametrów pracy poszczególnych bloków tworzących schemat generatora

Ustawienia parametrów pracy poszczególnych bloków schematu uzyskujemy klikając myszą na odpowiednie bloki.

1. W bloku generującym sinusoidę ustawiamy:
   - *Magnitude : 1* - amplitudę,
   - *Frequency : 1* - częstotliwość
     (należy w tym miejscu zauważyć że częstotliwość zadawana jest w rad/sec, jeżeli chcemy zadać częstotliwość f w Hz należy obliczyć wyrażenie $\omega = 2\pi f$, w naszym przykładzie f=0.159 Hz),
   - *Phase : 0* - przesunięcie fazowe (w radianach).



Rys. 1.11. Prosty przykład generatora funkcji sinus

2. Dla bloku wzmacniacza ustawiamy parametr *Gain* = 4.
3. Oscyloskop wymaga ustawienia aż 10 parametrów, co może wydawać się skomplikowane i trudne, jednak większość posiada ustawienia defaultowe, które przy pierwszych próbach są wystarczające. Duża ilość parametrów podlegających ustawieniu pozwala dopasować wyświetlanie do naszych indywidualnych potrzeb.
   W bloku oscyloskopu ustawiamy:
   - *color : []* pozostawiamy ustawienia defaultowe
   - Output window number : -1 - numer otwartego okna, -1 oznacza automatyczną numerację okna graficznego,
   - *Output window position : [x;y]* - miejsce w którym wyświetlane jest okno graficzne,
   - *Output window size : [300;200]* - wielkość okna,
   - *Ymin : -5* - minimalna wielkość wartości wyświetlanej,
   - *Ymax : 5* - maksymalna wielkość wartości wyświetlanej,
   - *Refresh period : 10* - zmienna określa długość (w sekundach) jednorazowo wyświetlanego odcinka generowanej funkcji, jeśli symulacja trwa dłużej niż 10 sekund, to wyświetlane jest kolejnych 10 sekund,
   - *Buffersize : 2* - wielkość bufora danych,
   - *Accepted herited events 0/1 : 0* - pozostawiamy ustawienia defaultowe
   - *Name of Scope* - nazwa okna.

4. Blok *write to file* wymaga ustawienia czterech parametrów.
   - *Input size : 1* - pozostawiamy bez zmian
   - *Output file name : sinusoida.dat* - określa nazwę pliku danych, plik zapisywanych jest w bieżącym katalogu roboczym Scilaba.
   - *Output format : (2(e10.3,1x))* określa format zapisu liczb do pliku. Format buduje się zgodnie ze stylem stosowany w Fortranie, zapis `e10.3` oznacza zapis wykładniczy składający się 10 pól w tym trzy zarezerwowano dla wykładnika, zapis `1x` oznacza spację.W pliku *sinusoida.dat* mamy dwie kolumny liczb, pierwsza zawiera chwile czasowe, druga wartości funkcji.
   - *Buffer size : 2* - pozostawiamy bez zmian,
5. Stoper wymaga ustawienia dwu parametrów:
   - *Period : 0.01* - okres próbkowania generowanego sygnału,
   - *Init time : 0* - chwila początkowa.

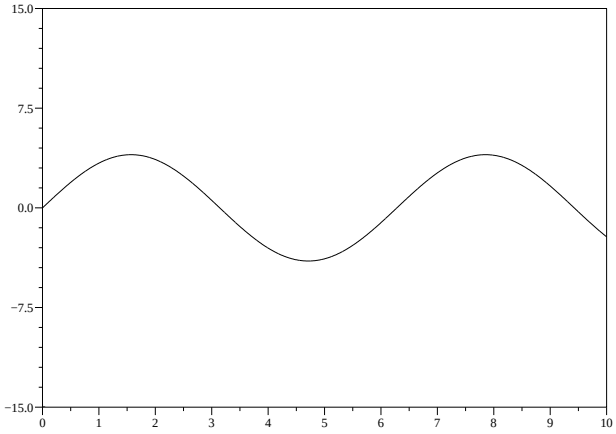
Ustawienie parametrów symulacji

Spośród ośmiu parametrów sterujących symulacją siedmiu pozostawiamy wielkości defaultowe, zmianie podlega jedynie *parametr final integration time* który kontroluje długość (w sekundach) generowanego przebiegu. W naszym przypadku ustawiamy go na 10 sekund, identycznie jak parametr *Refresh period* z bloku oscyloskopu.

Uruchomienie generatora, zapis

Uruchomienie zbudowanego generatora polega na wybraniu z menu *Simulate* opcji *Run*, zatrzymanie nastąpi samoczynnie po wygenerowaniu 10 sekund przebiegu lub po wybraniu opcji *stop*. Rezultaty symulacji, w postaci graficznej, możemy zobaczyć na rys. 1.12.

Zbudowany schemat generatora możemy oczywiście zapisać do pliku (z rozszerzeniem `*.cos`) posługując się poleceniem *Save* lub *Save as* z menu *Diagram*.



Rys. 1.12. Wygenerowana funkcja sinus

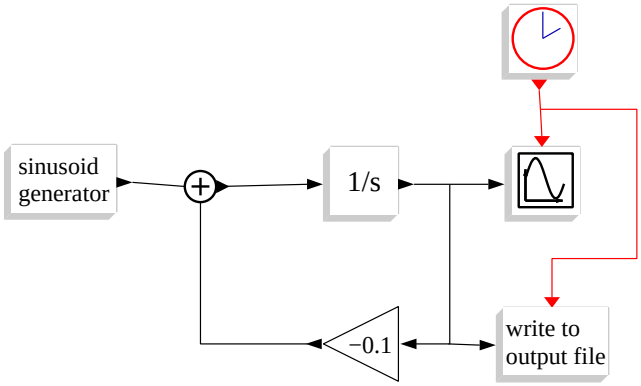
1.3.2. Rozwiązywanie równania różniczkowych rzędu pierwszego

Przedstawiony w poprzednim punkcie 1.3.1 schemat generatora funkcji sinus może posłużyć do budowy schematu blokowego służącego do rozwiązywania równań różniczkowych. Jako przykład weźmy równanie wraz z warunkiem początkowym:

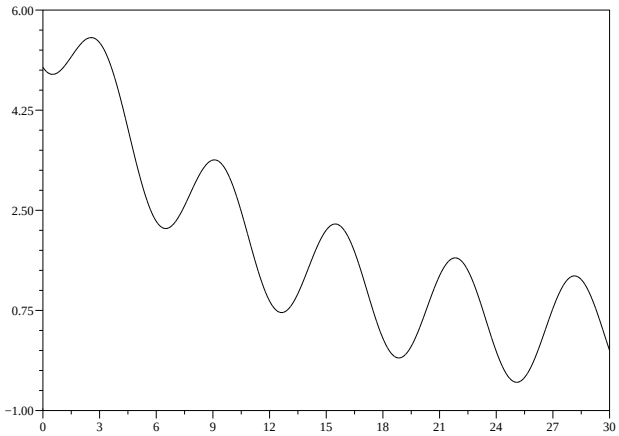
$$\begin{aligned} y'(t) + 0.1y(t) &= \sin(t), \\ y(0) &= 5. \end{aligned} \tag{1.3.1}$$

Rozpocznijmy od wprowadzenie bloku całkującego (paleta *Old blocks* - blok `INTEGRAL_f`) oraz bloku sumatora (paleta *Linear* - blok `Addition`), otrzymując schemat na rys. 1.13. Warunek początkowy (druga linijka w równaniu 1.3.1) ustawiamy klikając na blok całkujący, ustawiając parametr *Initial state : 5*. Rozwiązanie równania 1.3.1 możemy zobaczyć na ekranie oscyloskopu (rys. 1.14) oraz zapisać w pliku w postaci dyskretnej.

Zwróćmy w tym miejscu uwagę na fakt, że uzyskane rozwiązanie obejmuje jedynie pierwszych 30 sekund przebiegu funkcji $y = y(t)$, co oczywiście możemy zmienić poprzez zmianę parametrów symulacji. Nie mamy również możliwości podania analitycznej postaci rozwiązania



Rys. 1.13. Model graficzny rozwiązujący równanie (1.3.1)



Rys. 1.14. Rozwiązanie równania (1.3.1)

równania (1.3.1). Rozwiązanie równania (1.3.1) możemy również uzyskać

odwołując się tylko do Scilaba.

```
function [ydot]=f(t,y)
 ydot=-0.1*y+sin(t);
endfunction

// warunki początkowe
y0=5;t0=0; t=0:0.1:30;

//rozwiązanie
y=ode(y0,t0,t,f);

//wykres
plot2d(t,y);
```

Uzyskując rozwiązanie identyczne z tym pokazanym na rys. 1.14.

Analityczną postać funkcji $y = y(t)$ uzyskać możemy wykonując w środowisku Maximy [2] następujący skrypt:

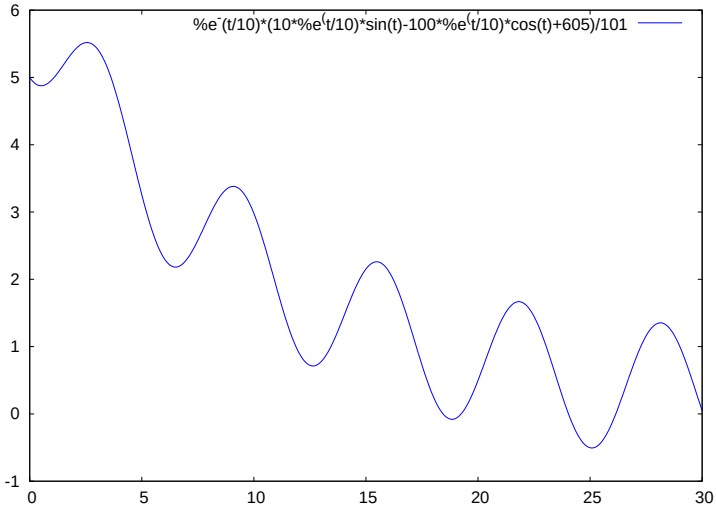
```
depends(y,t)$ eq1:ode2(diff(y,t)+0.1*y=sin(t),y,t);
ic1(eq1,t=0,y=5);

/* analityczna postać funkcji */
y1:rhs(%);
tex(%)

/* blok instrukcji związany z wyświetlanie wyników */
/* w postaci rysunku */
tex(%);
set_plot_option([gnuplot_term,ps]);
set_plot_option([gnuplot_out_file,"c:/praca/równanie.ps"]);
plot2d(y1,[t,0,30],[y,-1,6]);
```

Analityczna postać rozwiązania równania 1.3.1 jest pokazana poniżej

$$y = \frac{e^{-\frac{t}{10}} \left(10\, e^{\frac{t}{10}} \sin t - 100\, e^{\frac{t}{10}} \cos t + 605\right)}{101} \tag{1.3.2}$$

[2] Matlab Scilab Maxima, Opis i przykłady zastosowań, Cyprian T. Lachowicz, Oficyna Wydawnicza, Opole 2005. Program Maxima, typu freeware, jest dostępny w Internecie.

Oczywiście Maxima również umożliwia na wyświetlenie uzyskanego rozwiązania (1.3.2) w postaci wykresu (rys. 1.15).



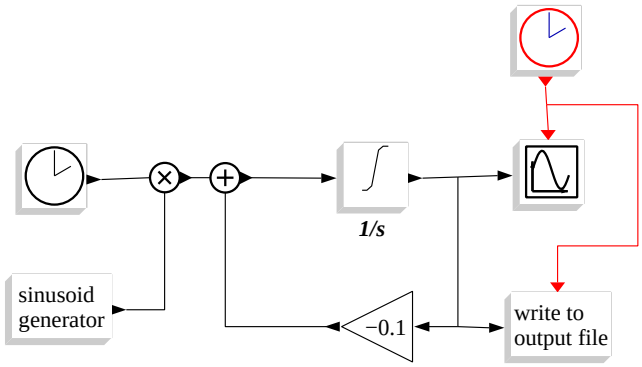
Rys. 1.15. Rozwiązanie równania (1.3.1), wykres funkcji 1.3.2

Równanie różniczkowe rzędu pierwszego (1.3.1) nie zawierało jawnie wprowadzonej zmiennej niezależnej *t*. Spróbujmy więc lekko je zmodyfikować do postaci:

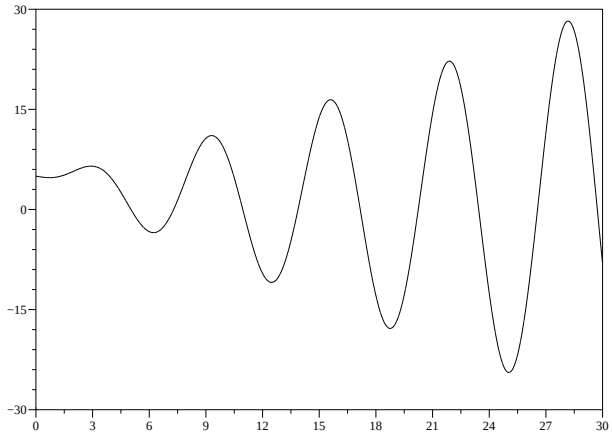
$$\begin{aligned} y'(t) + 0.1y(t) &= t\sin(t), \\ y(0) &= 5. \end{aligned} \tag{1.3.3}$$

Modyfikacja schematu rys.1.13 polega na wprowadzeniu bloku `mnożenia` (paleta *Non-linear* - blok `Multiplication`) oraz zmiennej niezależnej w postaci bloku `czas` (paleta *Sources* - blok `TIME`). Zmiana bloku całkującego na inny blok całkujący `Integration` (o nieco większej liczbie parametrów) pobrany z palety *Linear* nie wpływa na obliczenia.

Otrzymane rozwiązanie (rys. 1.17) równania 1.3.3 jest oczywiście zasadniczo różne od tego otrzymanego poprzednio.



Rys. 1.16. Model graficzny rozwiązujący równanie (1.3.3)



Rys. 1.17. Rozwiązanie równania (1.3.3)

Podobnie jak poprzednio rozwiązanie równania (1.3.3) można uzyskać posługując się tylko Scilabem

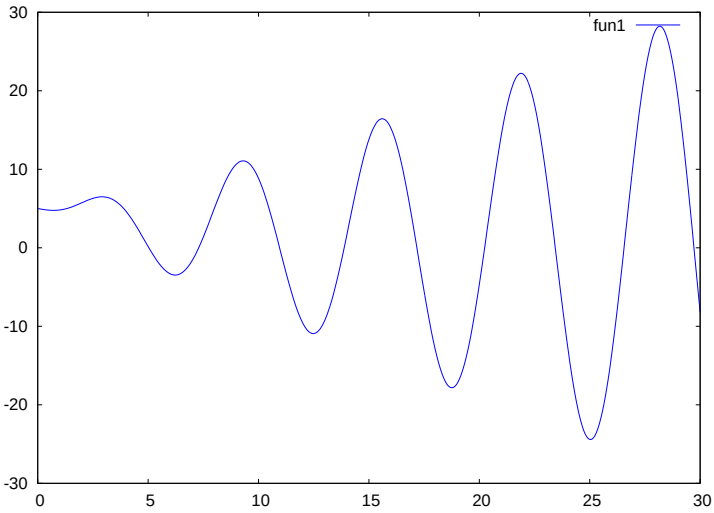
```
function [ydot]=f(t,y)
 ydot=-0.1*y+t*sin(t);
```

```
endfunction

// warunki początkowe
y0=5;t0=0; t=0:0.1:30;

//rozwiązanie
y=ode(y0,t0,t,f);

//wykres
plot2d(t,y);
```

Rozwiązanie analityczne (zal. 1.3.4, rys. 1.18) mamy po wykonania skryptu Maximy:

```
depends(y,t)$ eq1:
ode2(diff(y,t)+0.1*y=t*sin(t),y,t);
ic1(eq1,t=0,y=5);

y1:rhs(%);

tex(%);
set_plot_option([gnuplot_term,ps]);
set_plot_option([gnuplot_out_file,"c:/praca/rownanie20.ps"]);
plot2d(y1,[t,0,30],[y,-30,36]);
```

$$y = \frac{e^{-\frac{t}{10}} \left((1010\,t + 9900)\; e^{\frac{t}{10}}\; \sin t + (2000 - 10100\,t)\; e^{\frac{t}{10}}\; \cos t + 49005 \right)}{10201} \tag{1.3.4}$$



Rys. 1.18. Rozwiązanie równania (1.3.3), wykres funkcji (1.3.4)

1.4. Ogólny sposób rozwiązywanie zwyczajnego równania różniczkowego

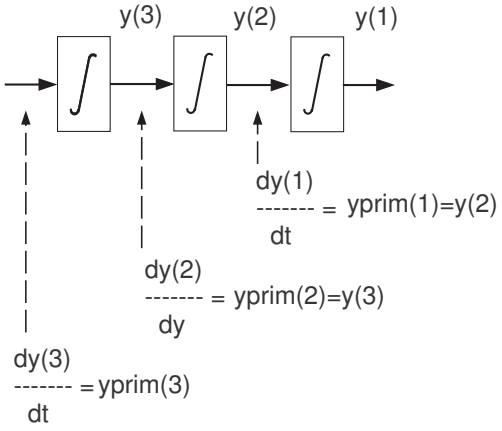
1.4.1. Opis

Rozwiązanie zwyczajnego równania różniczkowego w Scilabie wymaga wykorzystania polecenia `ode()`. Algorytm wykorzystywany w poleceniu `ode()` wymaga zapisania równania w postaci macierzowej jako zbioru równań pierwszego rzędu. Przykładowo rozwiążmy zwyczajne równanie różniczkowe stopnia 3.

$$\frac{d^3y}{dt^3} + \frac{d^2y}{dt^2} + \frac{dy}{dt} + 0.3y = 2 \tag{1.4.1}$$

Załóżmy na chwilę, że pochodna $\frac{d^3y}{dt^3}$ jest znana, jeśli scałkujemy ją trzykrotnie, to otrzymamy poszukiwaną funkcję $y(t)$. Graficznie zilustrowano to na rys. 1.19.

Wprowadźmy teraz zmienne pośrednie umieszczone pomiędzy blokami symbolizującymi całkowanie $y(1) \dots y(3)$. Związki pomiędzy tymi zmiennymi są następujące: $\frac{dy(1)}{dt} = y(2)$, $\frac{dy(2)}{dt} = y(3)$. Zwróćmy uwagę, że pochodna $\frac{dy(3)}{dt}$ jest sumą kolejnych składników równania (1.4.1)



Rys. 1.19. Model graficzny układu rozwiązującego równanie różniczkowe

$$\frac{d^3y}{dt^3} = -\frac{d^2y}{dt^2} - \frac{dy}{dt} - 0.3y + 2 \tag{1.4.2}$$

W celu ułatwienia zapisu macierzowego równania (1.4.1 lub 1.4.2) wprowadźmy następujące oznaczenie $\frac{dy(1)}{dt} = yprim(1)$, $\frac{dy(2)}{dt} = yprim(2)$, $\frac{dy(3)}{dt} = yprim(3)$.

Rozwiązanie równania wymaga definiowanie warunków początkowych. W naszym przypadku wartość funkcji $y(t)$ oraz wszystkie jej pochodne są równe zero dla chwili początkowej $t_0 = 0$.

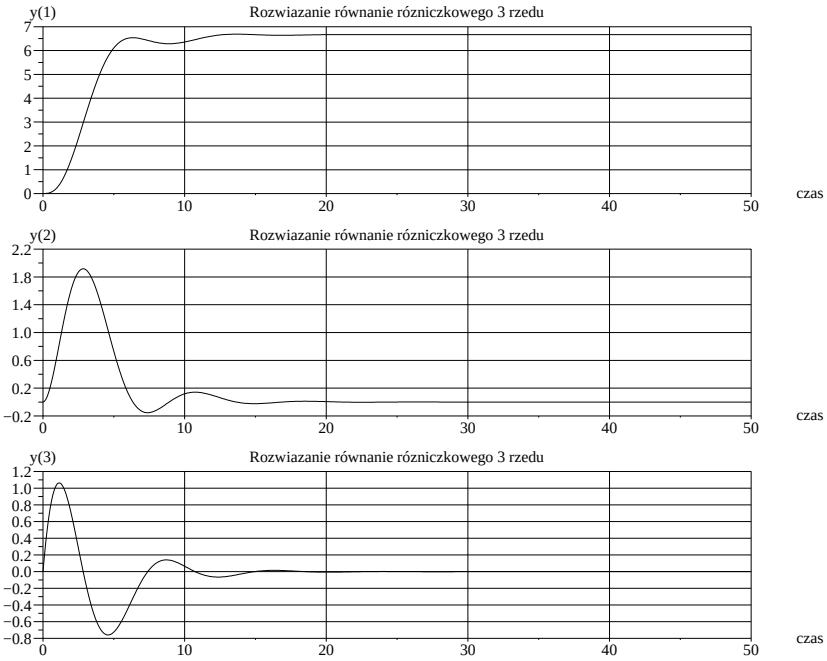
Scilab

Do rozwiązywania równania (1.4.1) można użyć skryptu pokazanego poniżej. Proszę zwrócić uwagę na macierzowy sposób zdefiniowania równania (1.4.1 lub 1.4.2) oraz na macierzowy sposób zapisu warunków początkowych.

```
// ************************************
// liniowe równanie różniczkowe trzeciego rzędu:
// y''' + y'' + y'+ 0.3*y = 2
// ************************************

// definicja równania
```

```
function [yprim]=diffeq(t,y)
    yprim(1)=y(2);
    yprim(2)=y(3);
    yprim(3)=-y(3)-y(2)-0.3*y(1)+2;
endfunction

// warunki początkowe
y(1)=0; y(2)=0; y(3)=0;
y0=[y(1); y(2); y(3);];

// definujemy umowna oś czasu, .1 to 'okres próbkowania'
t0=0; t=0:.1:50;

// rozwiązanie będzie macierzą o 3 wierszach i 501 kolumnach
// wiersz 1: y(1) 501 wartości
// wiersz 2: y(2) = yprim(1) = dy(1)/dt 501 wartości
// wiersz 3: y(3) = yprim(2) = dy(2)/dt 501 wartości

// tu rozwiązujemy równanie różniczkowe
y=ode(y0,t0,t,diffeq);

// -------------------------------------------------------
//ilustacja graficzna
xset('window',1); xbasc();
xsetech([0,0,1,1/3])
plot2d(t,y(1,:)); xgrid()
xtitle("Rozwiązanie równanie różniczkowego 3 rzedu","czas","y(1)");

xsetech([0,1/3,1,1/3])
plot2d(t,y(2,:)); xgrid()
xtitle("Rozwiązanie równanie różniczkowego 3 rzedu","czas","y(2)");

xsetech([0,2/3,1,1/3])
plot2d(t,y(3,:)); xgrid()
xtitle("Rozwiązanie równanie różniczkowego 3 rzedu","czas","y(3)");
```

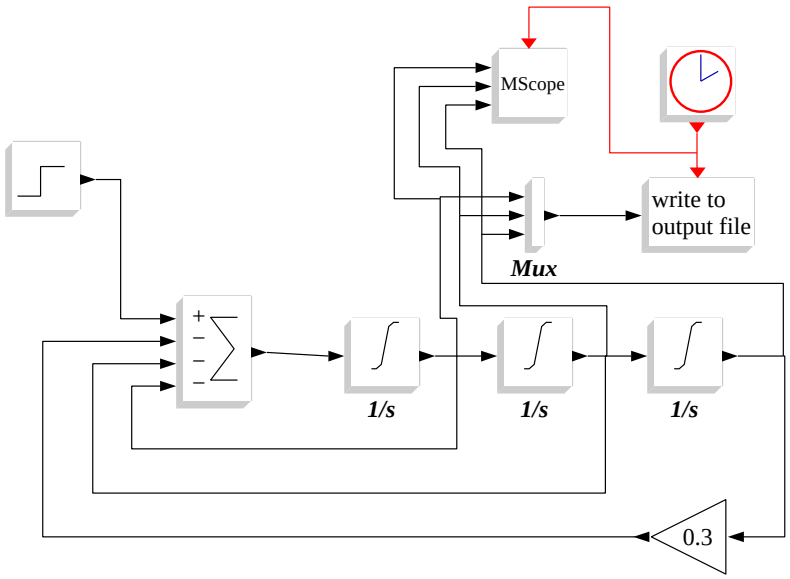
Na rys.1.20 pokazano graficzną ilustrację otrzymanych rozwiązań.



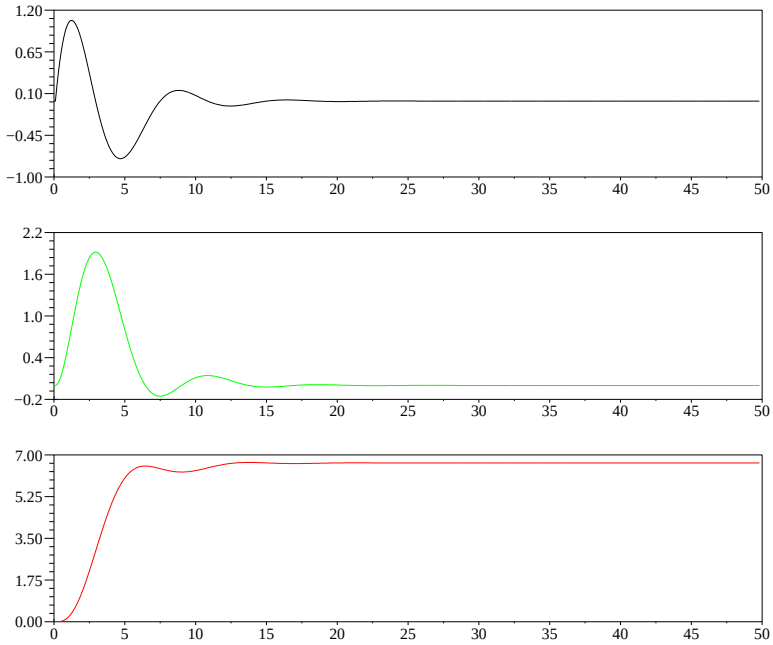
Rys. 1.20. Rozwiązania równania 1.4.1

Scicos

Maxima



Rys. 1.21. Model graficzny Scicosa do rozwiązywania równania 1.4.1



Rys. 1.22. Rozwiązania równania 1.4.1 w kolejności y(3), y(2),y(1)