KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Politechnika Wrocławska

UNIA EUROPEJSKA
EUROPEJSK
FUNDUSZ SPOŁECZNY

Scilab i Matlab
- podstawowe zastosowania inżynierskie

Anna Czemplik

„Wzrost liczby absolwentów w Politechnice Wrocławskiej
na kierunkach o kluczowym znaczeniu dla gospodarki opartej na wiedzy"

**Projekt współfinansowany ze środków Unii Europejskiej
w ramach Europejskiego Funduszu Społecznego**

Recenzent:
dr hab. inż. Iwona Karcz-Dulęba

OFICYNA WYDAWNICZA POLITECHNIKI WROCŁAWSKIEJ
Wybrzeże Wyspiańskiego 27, 50 – 370 Wrocław

ISBN 978 – 83 – 7493 – 714 – 6

Spis treści

Oznaczenia:

- czcionką Arial wyróżniane są nazwy i słowa kluczowe Matlaba i Scilaba oraz przykładowe nazwy zmiennych i fragmenty kodu
- czcionka *Arial z kursywą* oznacza miejsce na podanie konkretnego parametru (wartości, zmiennej, wyrażenia)
- czcionka zwykła i *z kursywą* zastosowana w liniach poleceń (fragmentach programu) oznacza dodatkowy komentarz (poza kodem programu)
- po znaku → podawany wynik działania funkcji
- symbol 💻 oznacza różnicę w prezentacji wyniku na tekstowych konsolach programów, na przykład w symbolicznych postaciach wielomianów i transmitancji wykorzystuje się dolne i górne indeksy a na tekstowych konsolach programów zapis ten obejmuje kilka linii, np.:
  wielomian $a + x + 2x^2$ będzie na konsoli widoczny w postaci:
  $$\begin{matrix} & & 2 \\ a + x + 2x \end{matrix}$$
- skrypty/funkcje pochodzące z edytorów Matlaba i Scilaba zachowują oryginalne kolorowanie składni
- fragmenty, które opisują różnice Matlaba i Scilaba są prezentowane w dwóch kolumnach, oznaczonych odpowiednio ikonami programów (mniej znaczące elementy porównania są wyświetlane na szaro)

1. Oprogramowanie do obliczeń naukowych i inżynierskich

1.1 Przegląd oprogramowania komercyjnego

Jeśli zostanie podane hasło „oprogramowanie do obliczeń naukowych i inżynierskich", to najczęściej w środowisku akademickim od razu pojawi się skojarzenie Matlab i/lub Mathematica.

Matlab jest jednym z najbardziej popularnych pakietów oprogramowania naukowo-inżynierskiego. Nazwa Matlab jest skrótem od słów *MATrix* LABoratory, co oddaje pierwotne przeznaczenie programu do numerycznych obliczeń macierzowych. Pra-początki Matlab-a sięgają lat siedemdziesiątych, kiedy to na zlecenie National Science Foundation powstały biblioteki języka Fortran do obliczeń macierzowych. W 1980 powstał program, który umożliwiał korzystanie z tych bibliotek w formie prostego interaktywnego języka poleceń (bez potrzeby programowania w Fortranie). Stał się on pierwowzorem Matlaba. W 1983 powstała firma The MathWorks Inc., która do dziś zajmuje się rozwojem i sprzedażą pakietu Matlab. Pierwsza wersja Matlaba pojawiła się w 1985 roku.

Obecnie pakiet Matlab jest niemalże międzynarodowym standardem obliczeń numerycznych, nauczanym prawie na wszystkich uczelniach. Podstawowe funkcje pakietu uzupełniają dodatkowe biblioteki (tak zwane toolbox'y), zawierające specjalistyczne funkcje z różnych obszarów matematyki, techniki, ekonomii, itp. Pakiet jest dostępny na różnych systemach operacyjnych Windows, Linux/Unix, Macintosh. Ma prosty interfejs oparty na komendach tekstowych oraz możliwość pisania i uruchamiania skryptów (tak zwane m-pliki) i programów w języku C lub Fortran. Obiektowy system graficzny pozwala na prezentację danych i wyników na wykresach dwu- i trójwymiarowych. Matlab umożliwia również pracę poprzez interfejs graficzny bardzo ceniony przez mniej wprawnych programistów (toolbox Simulink). Wszystko to sprawia, że jest to jedno z najbardziej ulubionych narzędzi inżynierów. Więcej na stronie http://www.mathworks.com/products/.

Mathematica - to system obliczeń symbolicznych i numerycznych opracowany w 1988 przez S. Wolframa. Założona przez niego firma Wolfram Research kreuje i realizuje własną wizję wspomagania obliczeń naukowych. W środowisku naukowców i inżynierów program jest bardzo popularny ze względu na wydajność, możliwości wizualizacji i prezentacji danych oraz przenośność. System zawiera bogatą kolekcję algorytmów do numerycznego i symbolicznego rozwiązywania równań algebraicznych, różniczkowych, rekurencyjnych i funkcyjnych, do przetwarzania równań, nierówności i macierzy oraz prezentacji rozwiązań w postaci symbolicznej i graficznej.

System obsługuje bardzo różne pojęcia, jak wyrażenia matematyczne, listy czy grafika, przy czym do ich definicji stosuje zunifikowany model wyrażenia i jest to kluczowa cecha programu, która powoduje, że każdy obiekt ma taką samą strukturę więc do przetwarzania wystarcza stosunkowo mały zestaw funkcji. Więcej na stronie http://www.wolfram.com/mathematica/features.

Matlab i Mathematica są niewątpliwie najpopularniejszymi programami do obliczeń naukowych i inżynierskich, ponieważ tworzą przenośne, bogate i ciągle rozwijane środowisko do prowadzenia badań w różnych dziedzinach. Spośród innych komercyjnych programów wspomagających obliczenia numeryczne i symboliczne (CAS, ang. **Computer Algebra System**) [I8] warto wymienić następujące:

Maple – program typu CAS, stworzony w 1981 roku przez *Symbolic Computation Group* na Uniwersytecie Waterloo (Kanada). Od 1988 rozwijany i sprzedawany komercyjnie przez *Waterloo Maple Inc.* Język programowania Maple jest językiem interpretowanym o dynamicznych typach danych. Wyrażenia symboliczne przechowywane są w pamięci jako skierowane grafy acykliczne. Więcej na stronie www.maplesoft.com.

Mathcad – program typu CAS, stworzony i rozwijany przez firmę *Mathsoft*. Cechą charakterystyczną Mathcada jest łatwość obsługi, a w szczególności łatwość tworzenia rozmaitych wykresów. Interfejs Mathcada imituje notatnik - równania i wyrażenia algebraiczne wyświetlane są w postaci graficznej, a nie tekstowej. Więcej na stronie www.mathcad.com.

MuPAD – program typu CAS, opracowany przez grupę badawczą MuPAD z Uniwersytetu w Paderborn we współpracy z firmą *SciFace Software GmbH*. Język MuPAD jest językiem obiektowy wzorowanym na Pascalu. Do wersji 3.2 Pro, w zastosowaniach *non-profit* (cele badawcze i

edukacyjne) można było wykorzystywać nieodpłatnie nieco ograniczoną wersję programu. Jednakże firma *SciFace* i prawa własności do programu MuPad zostały wykupione przez firmę MathWorks, która zintegrowała program ze swoim dodatkiem do obliczeń symbolicznych Symbolic Math Toolbox.

Wymienione programy należą do najbardziej uniwersalnych i popularnych ale nie wyczerpują dostępnej oferty. Jest wiele programów dedykowanych dla określonych obszarów, jak na przykład system Gauss przeznaczony do analizy danych, oparty na języku programowania macierzowego Gauss, opracowanym przez Aptech Systems na potrzeby statystyki, ekonometrii, optymalizacji i wizualizacji 2D/3D (więcej na stronie www.aptech.com).

1.2 Oprogramowanie niekomercyjne

Komercyjne systemy obliczeń naukowo-inżynierskich zapewniają bogate biblioteki algorytmów i przyjazne interfejsy użytkownika. Mają tylko jedną istotną wadę - są drogie. Nie jest to krytyczne dopóki można korzystać z wersji edukacyjnych albo ma się zapewniony dostęp w miejscu pracy. W zastosowaniach „domowych" warto zastanowić się nad alternatywnym rozwiązaniem, jakim jest korzystanie z oprogramowania typu open source, które w szerokim zakresie zastosowań zapewni podobną funkcjonalność. Dzięki zaangażowaniu wielu pasjonatów zaistniały i są rozwijane programy obliczeniowe, które chociaż są „darmowe", to nie są proste i bezwartościowe. Można w tej grupie wyróżnić dwa nurty. Jeden nurt stanowią programy, które narodziły się w tym samym czasie co obecne rozwiązania komercyjne - zawierają duże bogactwo algorytmów ale posługują się własnym interfejsem. Drugi nurt to programy, które powstały jako klony popularnych programów komercyjnych, czyli z założenia mają zestaw funkcji i interfejs użytkownika wzorowany na tych programach. Innym kryterium podziału niekomercyjnego oprogramowania do obliczeń naukowo-inżynierskich jest zasadnicze przeznaczenie: do obliczeń numerycznych (symulacji), podobnie jak Matlab, lub do obliczeń symbolicznych, podobnie jak Mathematica.

Najbardziej popularne bepłatne klony Matlaba to Octave, Freemat i Scilab [I2],[I4].

GNU Octave został zainicjowany około 1988 roku przez naukowców z University of Wisconsin–Madison i University of Texas, a przeznaczony był pierwotnie jako wsparcie obliczeń w projektowaniu reaktorów chemicznych. Znaczący rozwój oprogramowania rozpoczął się w 1992 roku i zmierza do rozszerzenia zarówno obszaru zastosowań (nie tylko projektowanie reaktorów) jak i kręgu użytkowników (studenci, naukowcy, projektanci).

Jest to język wysokiego poziomu, dostępny w środowisku Linuks i Windows. Pozwala rozwiązywać liniowe i nieliniowe problemy w obszarze równań algebraicznych i różniczkowych. Udostępnia wiele funkcji do wizualizacji danych i wyników. Użytkownik ma do dyspozycji interfejs poprzez linię komend i skrypty. Octave zachowuje bardzo dobrą zgodność z Matlabem (te same nazwy funkcji i składnia komend). Obecnie dostępna jest wersja 3.6.2. Więcej na stronie http://www.gnu.org/software/octave/.

FreeMat – powstał około 2004 roku i był dostępny na licencji Massachusetts Institute of Technology (obecnie jako GPL). W dokumentacji deklarowana jest zgodność z Matlabem na poziomie 95%. Wiele złożonych funkcji Matlaba nie jest dostępnych w programie ale można je znaleźć w sieci. Zapewniona jest pełna zgodność w składni skryptów i na poziomie obsługi grafiki. Program jest dostępny na systemach Windows, Linuks i MacOS. Obecnie dostępna jest wersja 4.1 wraz z podręcznikiem użytkownika. Więcej na stronie http://freemat.sourceforge.net/.

Scilab – został stworzony w 1990 roku przez INRIA (francuski narodowy instytut badań w dziedzinie komputerów) i ENPC (École nationale des ponts et chaussées). Od 1994 udostępniany bezpłatnie, a od 2003 rozwijany przez specjalnie utworzone Scilab Consortium. Od 2012 rozwój oprogramowania ma zabezpieczyć firma Scilab Enterprisess - The Service Company for Open Source Scilab Software (http://www.scilab-enterprises.com)

Program zawiera najobszerniejszy zestaw funkcji wśród omawianych. Chociaż narodził się jako klon Matlaba to nie zachowuje pełnej zgodności na poziomie nazw i składni. Jednak w kolejnych wersjach programu zgodność jest coraz większa, a poza tym można skorzystać z konwertera skryptów z Matlaba. Dostępny na systemy Windows, Linuks i MacOS, w różnych wersjach językowych. Wyróżniającą cechą Scilab jest edytor graficzny Xcos (odpowiednik Simulinka). Obecnie dostępna jest wersja 5.3.3 (oraz 5.4beta). Więcej na stronie http://www.scilab.org

Spośród różnych opracowań, które zawierają porównanie tych trzech projektów z Matlabem warto polecić [5], [6]. Najbardziej znaczące różnice [I2], to poziom zgodności z Matlabem na poziomie nazw i składni[1], algorytmów całkowania[2], dostępności na różnych systemach operacyjnych[3], interfejsu użytkownika[4], popularności[5]. Opinie użytkowników wskazują dość jednoznacznie, że jeśli decydujące znaczenie ma zgodność z Matlabem to najlepszym wyborem bezpłatnego rozwiązania jest Octave, który zapewnia przede wszystkim bezproblemowe przenoszenie skryptów. Natomiast gdy decydującą cechą ma być edytor graficzny, to jedynym rozwiązaniem jest Scilab[6].

Spośród niekomercyjnych programów do obliczeń symbolicznych najczęściej wymienia się dwa:

Maxima – program typu CAS, wywodzi się programu Macsyma opracowanego pod koniec lat 60 w Massachusetts Institute of Technology na zlecenie Departamentu Energii USA. W 1998 roku, za zgodą Departamentu Energii, udostępniono kod programu na licencji GPL i odtąd Maxima rozwija się w wersjach Windows, Linux i MacOs.
System realizuje operacje symboliczne i numeryczne, w tym różniczkowanie, całkowanie, szeregi Taylora, transformaty Laplace'a, równania różniczkowe zwyczajne, układy równań liniowych, wielomiany, oraz zbiory, listy, wektory macierze i tensory. Zapewnia także grafikę dwu- i trójwymiarową do prezentacji danych i funkcji. Głównym składnikiem programu jest interpreter. Maxima posiada własny, prosty interfejs graficzny - XMaxima. Niezależnie rozwijany jest wieloplatformowy interfejs wxMaxima. Maximę można również uruchamiać w edytorze tekstu Emacs oraz TeXmacs. Więcej na stronie http://maxima.sourceforge.net/.

Axiom – program typu CAS, powstał w IBM w 1971 roku pod nazwą Scratchpad i był platformą badawczą do testowania nowych idei w obliczeniach. Od 1990 program nabyła firma NAG (Numerical Algorithms Group), która rozwijała go pod nazwą Axiom ale nie odniosła sukcesu. Od 2001 roku Axiom jest wspierany przez CAISS (Center for Algorithms and Interactive Scientific Software) i udostępniany na licencji free software. Prace prowadzone obecnie mają na celu dostosowanie systemu do nowych wymagań (np. poprawa interfejsu użytkownika) tak by utrzymać „przy życiu" ogromną bazę wiedzy jaką stanowi kod programu. Więcej na stronie http://axiom-developer.org.

Wypada wspomnieć o jeszcze jednej grupie programów do prowadzenia badań symulacyjnych – o rozwiązaniach opartych na języku **Modelica**, opracowanego w latach 1995-1997 roku w ramach projektu SiE-WG [I5],[I7]. Jest to język do obiektowego modelowania układów fizycznych, który

[1] duża zgodność Octave i FreeMat, ograniczona zgodność Scilab
[2] żaden z programów nie dorównuje ofercie Matlaba, ale najuboższy jest FreeMat, który nie zawiera algorytmów do rozwiązywania równań sztywnych
[3] wszystkie są dostępne pod Windows, Linux i MacOS, ale w każdym występują preferencje – FreeMat jest najprostszym rozwiązaniem dla Windows, Octave najlepiej działa na Linux a i Scilab na Windows
[4] tylko Scilab ma edytor graficzny – odpowiednik Simulinka
[5] najmniej użytkowników ma FreeMat
[6] jedyna bezpłatna alternatywa xcos jest w środowisku FOSS

umożliwia nie tylko definiowanie własnych modeli (za pomocą równań różniczkowych lub różniczkowo-algebraicznych), ale udostępnia bezpłatną bibliotekę gotowych modeli (około 1300 modeli i 900 funkcji). Modele i biblioteki modeli zgodne ze specyfiką języka mogą być używane w różnych środowiskach oprogramowania zintegrowanego z Modeliką – są wśród nich pakiety komercyjne z dodatkowymi bibliotekami, np. Dymola, Vertex, Converge, SimulationX, ale także pakiety bezpłatne, np. JModelica.org, Modelicac, OpenModelica, SimForge. Są to interdysplinarne środowiska z edytorami graficznymi do konstrukcji złożonych modeli oraz sterowania symulacją. Od 1996 roku rozwojem standardu zajmuje się Modelica Association (organizacja non-profit). Więcej na stronie https://www.modelica.org/.

1.3 Dlaczego książka o Scilabie?

Już z powyższego, krótkiego przeglądu niekomenrcyjnego oprogramowania wspomagającego obliczenia naukowo-inżynierskie widać, że Scilab nie jest ani jedynym, ani najmocniejszym rozwiązaniem. Dlaczego więc stał się przedmiotem niniejszego opracowania? Podstawowe przesłanki są następujące: jest zbliżony do Matlaba, ma interfejs graficzny do rysowania schematów, już ma bogata bibliotekę funkcji i ciągle się rozwija. Dzięki inicjatywie Scilab Enterprises projekt ma zapewnione dalsze wsparcie. Poza tym wszystkim jest to inicjatywa europejska.

Ze względu na ciągły rozwój programu najbardziej aktualne informacje są dostępne przede wszystkim na oficjalnej stronie projektu http://www.scilab.org/. Istnieją też różne opracowania na temat Scilaba wydawane w postaci książek, co świadczy o powadze i zainteresowaniu tematem - dla przykładu po angielsku: [2], [3], [4], [9], [10], [11], [12], ale także po polsku [8]. Znaczna część opracowań dotyczy podstawowego wprowadzenia do Scilaba – komendy systemowe, operacje na macierzach, sposób pisania skryptów i funkcji, itp. Szczególnie warto polecić drugie już wydanie podręcznika [3], który jako jeden z niewielu omawia zarówno Scilaba jak i zagadnienia związane z edytorem graficznym Scicos, z tym, że opis jest oparty na wersji 4.4 a od wersji 5.2 Scilab jest udostępniany z nowym edytorem graficznym Xcos, który zastąpił poprzednie rozwiązanie.

Niniejsze opracowanie prezentuje podstawowe własności Scilaba i Xcosa z odniesieniem do Matlaba i Simulinka[1]. Są to własności wybrane głównie pod kątem wspomagania badań symulacyjnych układów dynamicznych [1], a więc w typowym obszarze zastosowania oprogramowania przeznaczonego do obliczeń naukowych i inżynierskich. Opracowanie może stanowić praktyczne wprowadzenie dla nowych użytkowników zarówno Scilaba jak i Matlaba, aczkolwiek głównym adresatem są użytkownicy Matlaba, którzy zaczynają pracą pod Scilabem, zwłaszcza z zastosowaniem środowiska graficznego. To co często zniechęca użytkowników do korzystania z bezpłatnego oprogramowania to właśnie słabsze interfejsy graficzne niekomenrcyjnego oprogramowania – „słabsze" lub „nieco inne" niż przyzwyczajenia nabyte w komercyjnych programach. Opracowanie nie jest dokumentacją Matlaba czy Scilaba – takową zapewniają łatwo dostępne przeglądarki pomocy oraz książkowe i internetowe podręczniki dedykowane poszczególnym programom. Plan niniejszego opracowania odpowiada typowej kolejności poznawania własności programu, która umożliwia szybkie podjęcie pracy w nowym środowisku: definicję modelu oraz zautomatyzowaną realizację programu badań i prezentacji wyników.

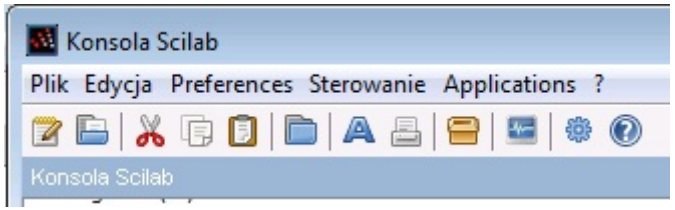
W tym miejscu pragnę podziękować grupie studentów, którzy włączyli się w doświadczalne badania środowiska Scilab, w szczególności:
Andrzej Gnatowski, Marcin Spik, Adrian Szymański, Przemysław Kochański, Michalina Kotyla, Marcin Bogner, Paweł Dmochowski, Adam Przybyła, Marcin Strojny. Zainteresowanych tematem wykorzystania Scilaba w badaniach lub włączeniem się w prace nad rozwojem tego oprogramowania zapraszam do kontaktu przez email anna.czemplik@pwr.wroc.pl. Porównując w dalszej części rozbudowane możliwości komercyjnego Matlaba z funkcjonalnością Scilaba, który rozwija się jako Open Source, można wskazać różne obszary z zakresu metod numerycznych, automatyki czy informatyki stosowanej oczekujące na dopracowanie i dalszy rozwój, co może być szansą na włączenie się w poważny projekt dla inżynierów różnych specjalności.

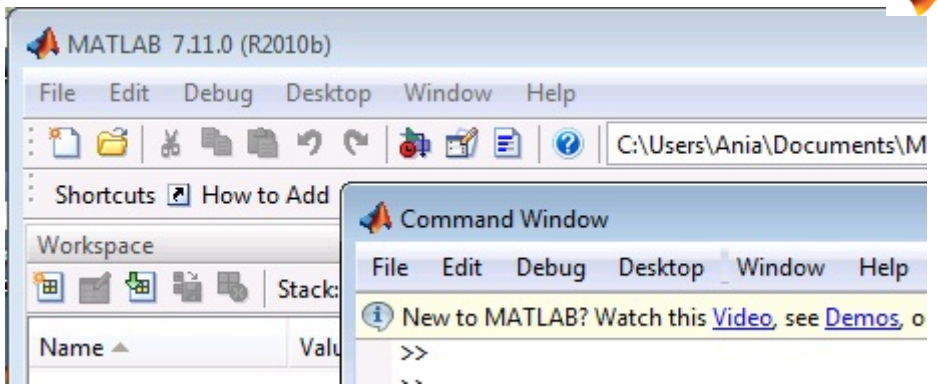
[1] Scilab w wersji 5.3.3 (z polskim interfejsem) oraz Matlab w wersji R2010b (lub co najmniej 5.3)

2. Podstawowe aplikacje i komendy

2.1 Główne konsole programów

2.1.1 Menu konsoli (okna) komend





File – Plik - operacje na plikach i kartotekach
Edit – Edycja – operacje na schowku
Preferences – konfiguracja konsoli
Control - Sterowanie – sterowanie symulacją
Applications – uruchamianie aplikacji, w tym edytorów

? – Przeglądarka pomocy, Przykłady, Linki w sieci

File – operacje na plikach i kartotekach
Edit – operacje na schowku i na danym oknie

Debug – sterowanie symulacją
Deskop – zarządzanie oknami i uruchamianie aplikacji, w tym edytora
Window – wybór aktywnego okna
Help – Przeglądarka pomocy i funkcji, Linki w sieci, Przykłady

Główna konsola Scilaba obsługuje tekstowy interfejs użytkownika (wywoływanie komend, wyświetlanie odpowiedzi). W obecnych wersjach Matlaba funkcję tę realizuje okno komend (Command Window), które może być wyświetlane oddzielnie (domyślnie) lub w obszarze głównej konsoli. We wcześniejszych wersjach programów, np.: Scilab 5.2 i Matlab 5.3, było widoczne większe podobieństwo konsol obu programów.

W obu programach główna konsola umożliwia również zarządzanie oknami i aplikacjami pakietu. Zestaw tych aplikacji jest następujący:

SciNotes (edytor tekstowy), Xcos (edytor schematów),
Translator Matlab to Scilab,
ATOMS (manager module)
Variable Browers (Przeglądarka zmiennych)
Command History (Historia poleceń)
Okna są wyświetlane oddzielnie
Aplikacje są przygotowane do przełączania wersji językowych (w wer.5.3.3 jeszcze nie wszystko działa; patrz: getlanguage / setlanguage)

Command Window, Editor (edytor tekstowy), Figures,
Current Folder, Workspace, Help (Przeglądarka pomocy),
Profilter, File Exchange, Web Browers, Comparision Tool
Variable Editor (Przeglądarka zmiennych)
History Window (Historia poleceń),
Okna są wyświetlane oddzielnie lub w obszarze głównej konsoli

2.1.2 Podstawowe komendy systemowe

Za pomocą komend systemowych wpisywanych do konsoli głównej (okna komend w Matlabie) można uzyskać informację i wykonać podstawowe operacje na kartotekach, plikach i zmiennych.

help *nazwa_funkcji* - uruchamia Help Browser z opisem funkcji	help *nazwa_funkcji* - wyświetla informację o funkcji na konsoli
pwd - bieżąca kartoteka (print work directory); też: getcwd	pwd - bieżąca kartoteka
cd - zmiana kartoteki (change directory); też: chdir	cd - zmiana kartoteki
ls - zawartość bieżącej kartoteki (list)	ls - zawartość bieżącej kartoteki
who / whos - pokaż listę zmiennych w tym systemowych, funkcji, ...	who / whos - pokaż listę zmiennych użytkownika
clear - kasuj zmienne niezabezpieczone	clear - kasuj wszystkie zmienne

Podstawowa różnica w działaniu tych komend dotyczy obsługi zmiennych, ponieważ w Scilabie wszystkie nazwy reprezentujące zmienne użytkownika, stałe i zmienne zdefiniowane w systemie, biblioteki funkcji, komendy są zmiennymi różnych typów (→). Przykłady:

whos - wyświetl pełną informację o wszystkich nazwach zmiennych, funkcjach, bibliotekach, ...	whos - wyświetl pełną informację o wszystkich zmiennych użytkownika
whos -name 'a' - jw. ale o nazwach zaczynających się na literę 'a'	whos a* - jw. ale o nazwach zaczynających się na literę 'a'

2.1.3 Typy i nazwy plików

W Scilabie i Matlabie występują dwa podstawowe typy plików: tekstowe (skrypty i funkcje) oraz definiujące schematy blokowe modeli.

Typy plików są identyfikowane na podstawie rozszerzenia w nazwie:

*.sce, *.sci - skrypty i funkcje	*.m - skrypty i funkcje
*.xcos - schematy, obsługiwane przez Xcos	*.mdl - schematy, obsługiwane przez Simulik
Nazwy plików podaje się zawsze z rozszerzeniem (zazwyczaj w cudzysłowie). Nazwy bez rozszerzenia Scilab interpretuje dosłownie. Scilab poszukuje wskazanych plików w bieżącej kartotece. W nazwie pliku można również wskazać ścieżkę.	Nazwy plików podaje się bez podaje się bez rozszerzenia. Podaną nazwę Matlab próbuje zinterpretować kolejno jako: komenda/funkcja, zmienna, schemat, skrypt. Matlab poszukuje wskazanych plików w kartotece bieżącej oraz w ścieżkach ustawionych z menu File\SetPath.

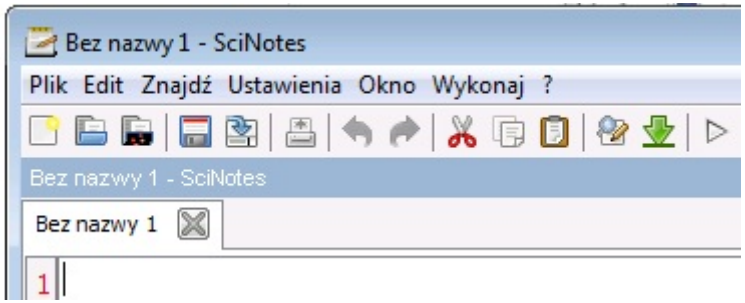
.

2.2 Aplikacje: edytor tekstowy (skrypty i funkcje)

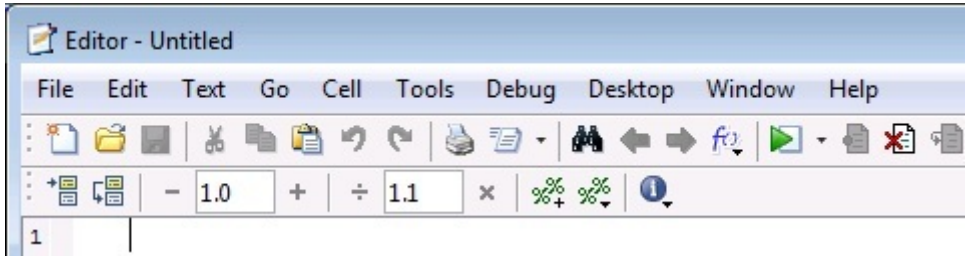
2.2.1 Uruchomienie i funkcje edytora tekstowego

Skrypty i funkcje mogą być edytowane pod dowolnym edytorem tekstowym, który nie wprowadza formatowania. Jednakże specjalizowane edytory Matlaba i Scilaba zapewniają dodatkowo kontekstowe kolorowanie składni i funkcje do uruchamiania.

Z konsoli Scilaba:	Z konsoli Matlaba lub z okna komend:
- menu Applications/SciNotes (uruchamia edytor)	- menu File/New/.... (uruchamia edytor i tworzy plik wybranego typu)
- menu Plik/Open (uruchamia edytor i otwiera wybrany plik)	- menu File/Open (uruchamia edytor i otwiera wybrany plik)
- edit – polecenie w linii komend (uruchamia edytor)	- edit – polecenie w linii komend (uruchamia edytor)



File – Plik – otwieranie i zapamiętywanie plików
Edit – obsługa schowka
Search – Znajdź – przeszukiwanie pliku
Preferences – Ustawienia – konfiguracja edytora
Window – Okno – zarządzanie oknem edytora
Execute – Wykonaj – uruchamianie skryptu
? – Przeglądarka pomocy na temat edytora



File – otwieranie i zapamiętywanie plików
Edit – obsługa schowka i przeszukiwanie pliku
Text, Go, Cell, Tools – funkcje dodatkowe (dla skryptów)
Debug – uruchamianie skryptu
Desktop – zarządzanie oknem edytora
Window – zarządzanie oknem edytora
Help – Przeglądarka pomocy na temat edytora

Oba edytory umożliwiają otwarcie kilku plików jednocześnie i udostępniają je na oddzielnych zakładkach w oknie edytora.

2.2.2 Uruchomienie istniejącego skryptu użytkownika

Pod edytorem Scilaba:	Pod edytorem Matlaba:
– menu Execute\... - Wykonaj\.... - do wyboru sposób wykonania Uruchomienie skryptu pod edytorem domyślnie jest związane z zapamiętaniem skryptu ale można uruchomić zawartość bez zapamiętywania	– menu Debug/Run - także w trybie debug Uruchomienie skryptu pod edytorem realizuje kod zawarty w oknie edytora
Z konsoli Scilaba: – exec('*nazwa_skryptu*.sce'); - polecenie w linii komend (nazwa skryptu z rozszerzeniem; średnik na końcu wyłącza echo) – menu File\Execute – Plik\Wykonaj - uruchamia wybrany skrypt	Z okna komend Matlaba: – *nazwa_skryptu* - jako polecenie w linii komend

2.2.3 Wywołanie funkcji użytkownika zapisanej w pliku

Załóżmy, że istnieje plik w którym zdefiniowano pewną funkcję o nazwie suma (zasady definiowania funkcji opisano w punkcie 5.3).

Przed wykonaniem funkcje z pliku muszą być wczytane (załadowane) do przestrzeni roboczej Scilab; potem można ich używać:	Plik zawierający potrzebną funkcję powinien znajdować się w bieżącej kartotece lub w zdefiniowanych ścieżkach
exec('nazwa_pliku.sce') - wczytuje funkcje zawarte w podanym pliku, x=suma(1,2); - użycie funkcji	x=suma(1,2); - użycie funkcji

2.3 Aplikacje: edytor graficzny (schematy/diagramy)

2.3.1 Uruchomienie edytora graficznego

Z konsoli Scilaba :
- menu Applications/Xcos
- ikona Xcos
- xcos - polecenie w linii komend

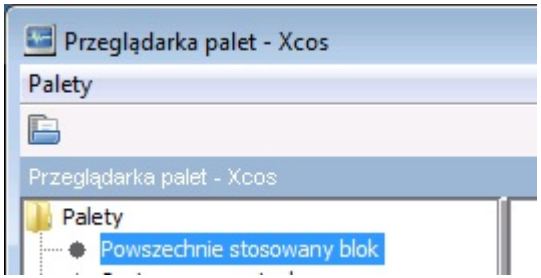
Operacja uruchamia przeglądarkę biblioteki (Palette browser Xcos - Przeglądarka palet Xcos) oraz puste okno edytora Xcos

Z konsoli Matlaba
– ikona Simulink

Z okna komend Matlaba: Simulink

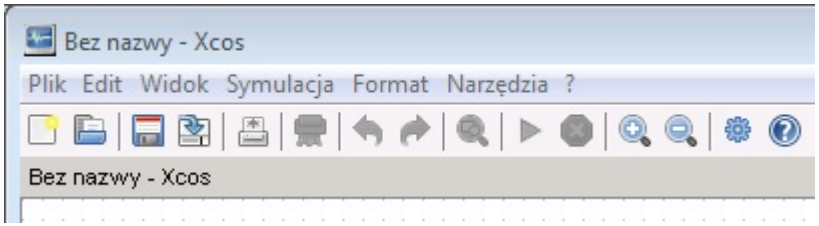
Operacja uruchamia tylko przeglądarkę biblioteki (Simulink Library Browser). Z menu przeglądarki (File) można otworzyć puste okno edytora Simulik

2.3.2 Przeglądarka bibliotek

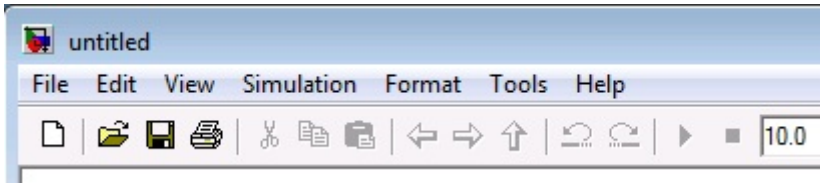


Nazwy podstawowych bibliotek Xcos i Simulink są podobne (więcej w punkcie 7.2). Po wybraniu biblioteki wyświetlana jest jej zawartość.

2.3.3 Edytor graficzny - funkcje



File – Plik – otwieranie i zapamiętywanie plików
Edit – obsługa schowka oraz wybranych bloków na schemacie
View – Widok – konfiguracja edytora, otwieranie przeglądarki bibliotek
Simulation – Symulacja – parametry i uruchamianie symulacji
Format – formatowanie bloków
Tools – Narzędzia – dodatkowe narzędzia (obecnie: Generowanie kodu)
? – Przeglądarka pomocy na temat edytora



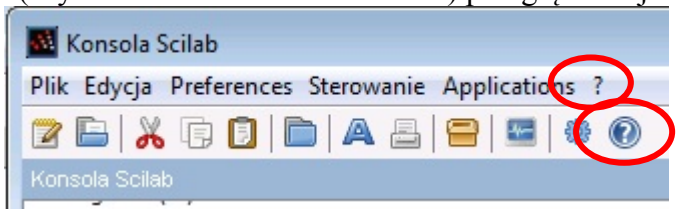
File – otwieranie i zapamiętywanie plików ze schematami
Edit – obsługa schowka oraz wybranych bloków na schemacie
View – Widok – konfiguracja edytora, otwieranie przeglądarki bibliotek
Simulation – parametry i uruchamianie skryptu
Format – formatowanie bloków
Tools – dodatkowe narzędzia
Help – Przeglądarka pomocy na temat edytora

Dodawanie bloków na schemacie odbywa się metodą przeciągania (wybór bloku z przeglądarki, przeciągnięcie na schemat i upuszczenie) lub kopiowania bloków, które już występują na schemacie (ctr+c, ctr+v). Wybór i łączenie bloków liniami odbywa się także za pomocą myszy. Ponieważ środowisko Xcos ma swoją specyfikę, która odbiega od przyzwyczajeń użytkowników w Dodatku 10.1 przedstawiono krótka instrukcję.

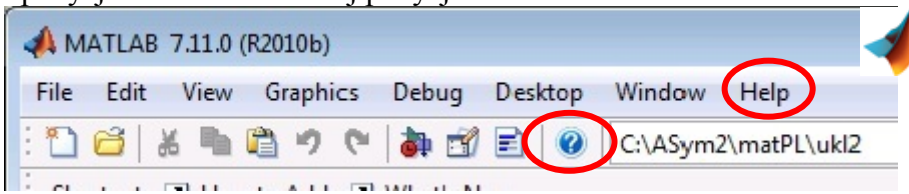
2.4 Aplikacje: przeglądarka pomocy

2.4.1 Uruchomienie przeglądarki pomocy

Każde okno aplikacji Matlab i Scilab zawiera w swoim menu pozycję i/lub ikonę związaną z uruchomieniem przeglądarki pomocy. W zależności jest od aplikacji (czyli od kontekstu uruchomienia) przeglądarka jest automatycznie pozycjonowana właściwej pozycji.



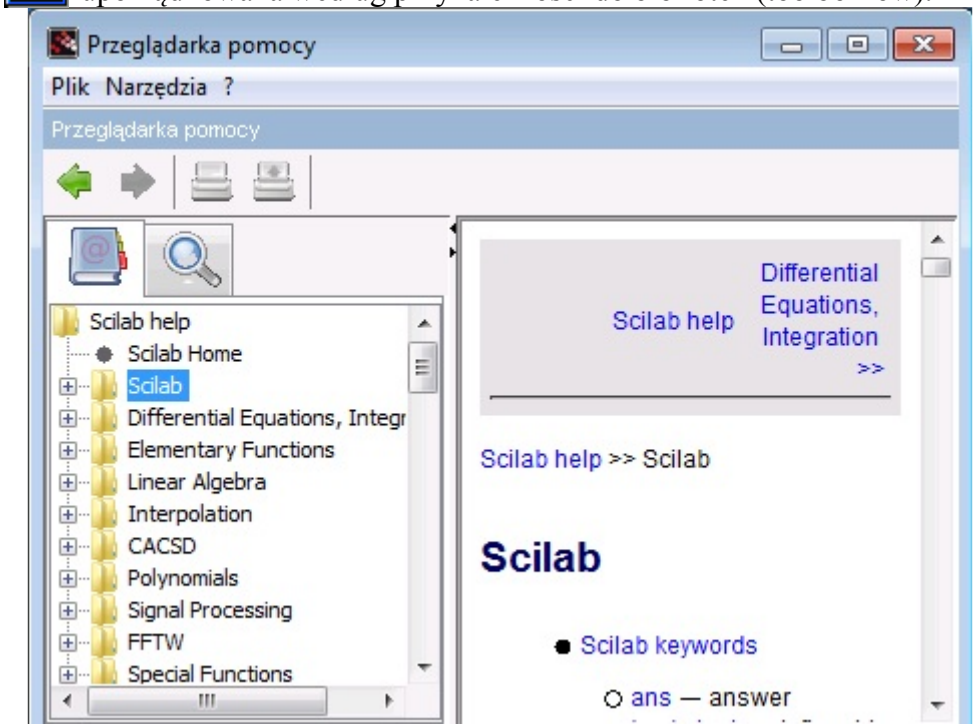
?\Help Scilab - ?\Pomoc Scilab – przeglądarka pomocy (od wersji 5.0)

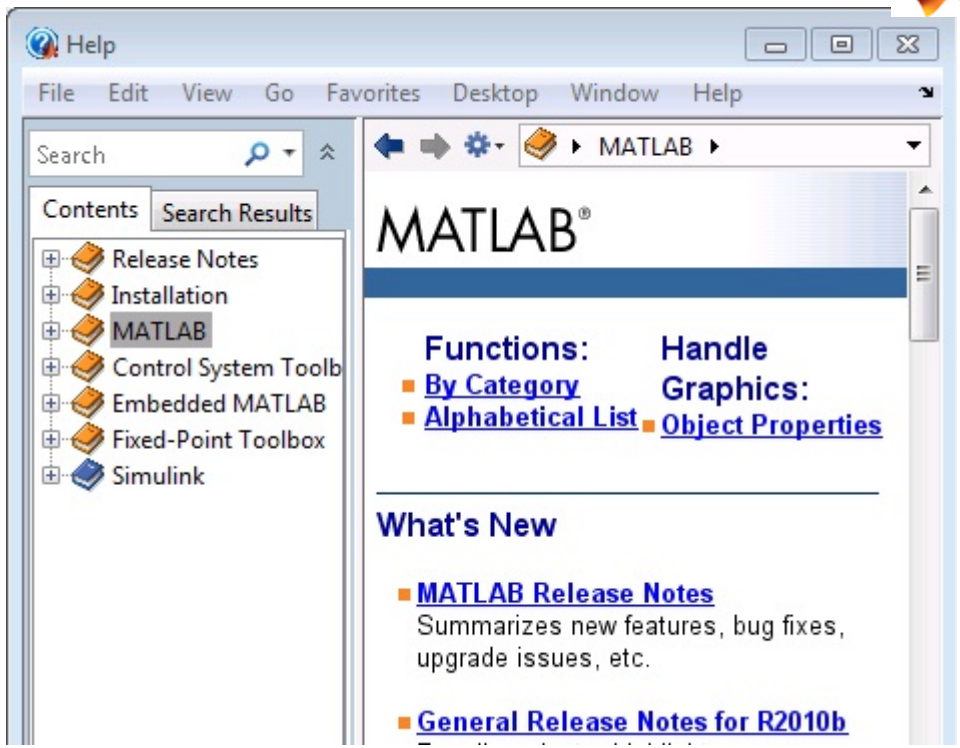


Help\Product Help – przeglądarka pomocy
Help\Function Browser – przeglądarka funkcji (wykaz funkcji)

2.4.2 Organizacja przeglądarki pomocy

Podstawowe funkcje przeglądarek są podobne – przeglądanie zawartości oraz wyszukiwanie wskazanego hasła. Zawartość jest uporządkowana według przynależności do bibliotek (toolbox'ów).



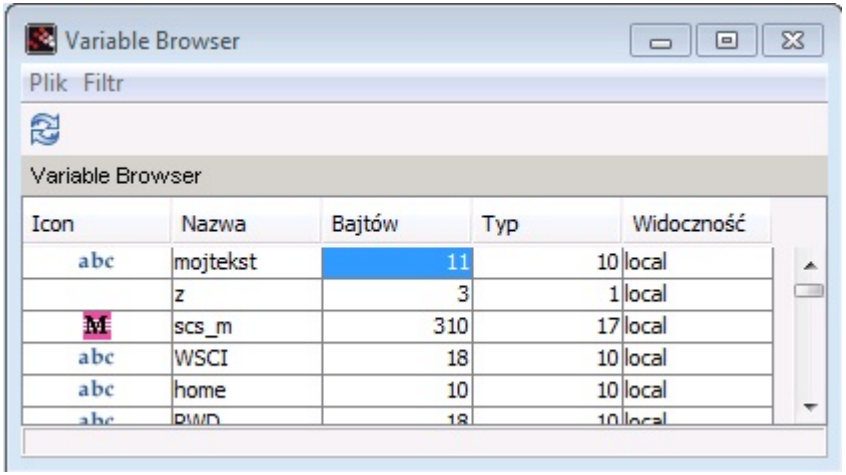


Oczywiście wsparcie dla Matlaba jest znacznie większe niż w przypadku Scilaba, ale oba obszary są ciągle rozwijane. Warto zwrócić uwagę na informacje dostępne poprzez strony internetowe obu produktów http://www.mathworks.com/ , http://www.scilab.org/.

2.5 Aplikacje: przeglądarka i edytor zmiennych

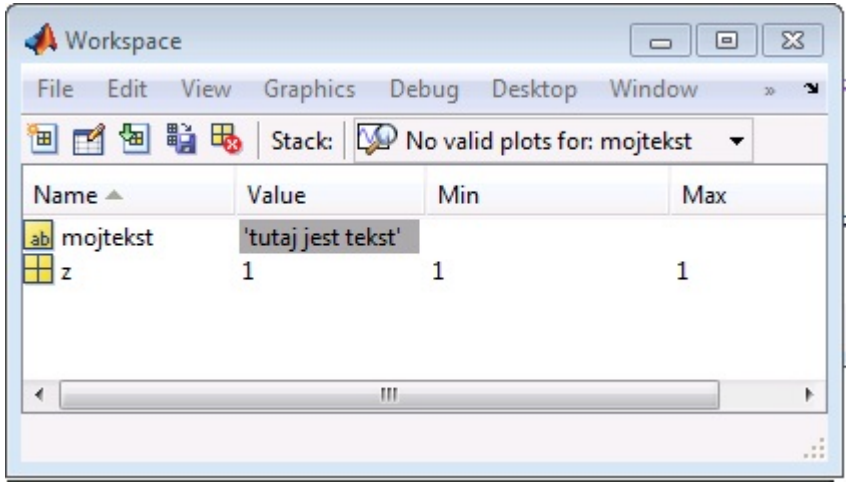
2.5.1 Uruchomienie przeglądarki zmiennych

Z konsoli Scilaba :
- menu Applications/Variable Browser - Applications/Przeglądarka zmiennych



Przeglądarka pokazuje wszystkie zmienne (także systemowe). Nie zwiera informacji o wartości zmiennych i jest oświeżana na żądanie.

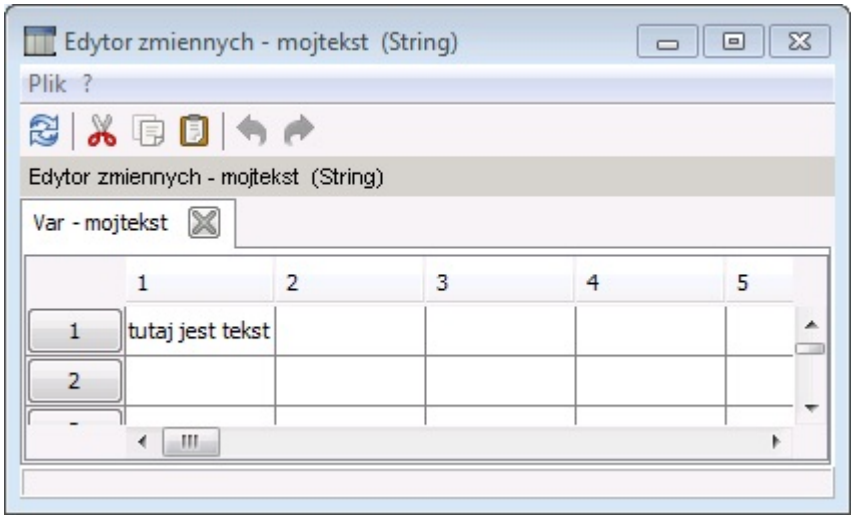
Z konsoli Matlaba
– menu Desktop/Workspace



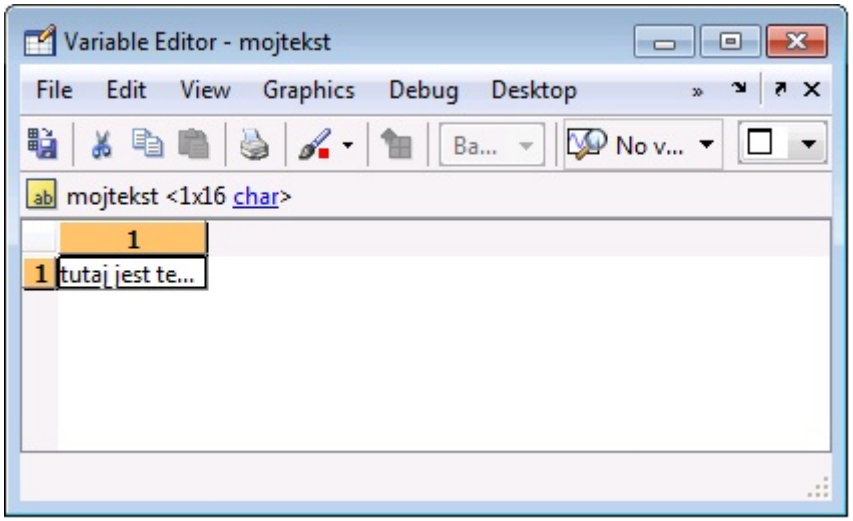
Przeglądarka pokazuje tylko zmienne użytkownika. Zawiera informacje o wartości zmiennych i jest automatycznie odświeżana.

2.5.2 Uruchomienie edytora zmiennych

Z przeglądarki zmiennych – podwójne kliknięcie na wybranej zmiennej



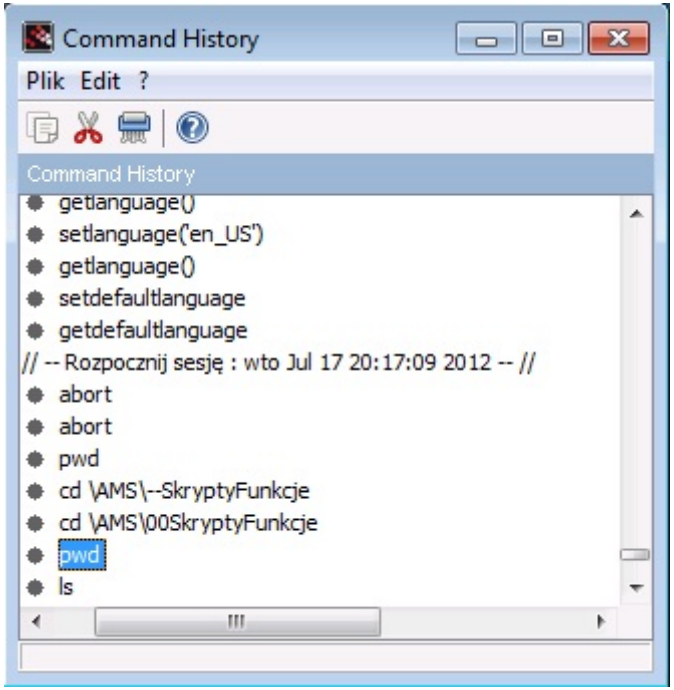
Z przeglądarki zmiennych – podwójne kliknięcie na wybranej zmiennej
Z konsoli Matlaba – menu Desktop/Variable Editor (otwiera pusty edytor)



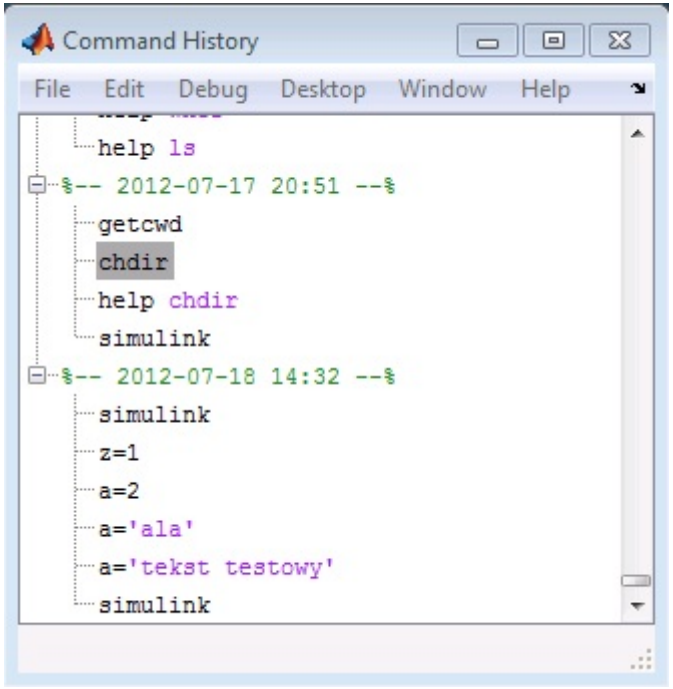
2.6 Aplikacje: historia poleceń

2.6.1 Podgląd historii poleceń

Z konsoli Scilaba :
- menu Applications/Command History - Applications/Historia poleceń



Z konsoli Matlaba
– menu Desktop/Command History



W historii poleceń rejestrowane są daty uruchomienia programu oraz komendy wydawane przez użytkownika w danej sesji. Każde polecenie można powtórzyć przez wskazanie i podwójne kliknięcie myszą. Polecenia z bieżącej sesji można także przywołać za pomocą klawisza ↑ w oknie komend (przywołane polecenia można też poprawić i powtórnie wykonać).

3. Podstawowe operacje i funkcje

3.1 Stałe, operacje, wektory, macierze, algebra macierzowa

3.1.1 Podstawowe typy zmiennych

Najprostsze obiekty Matlaba i Scilaba to skalary, znaki, wartości logiczne, które są podstawą do konstrukcji bardziej złożonych obiektów jak wektory, macierze, teksty (łańcuchy znaków), liczby zespolone, wielomiany, struktury. Przykłady:

- predefiniowane stałe matematyczne: %i, %pi, %e, %inf, %nan, %eps, - predefiniowane wartości logiczne: %f, %t - standardowe wejście i wyjście: %io(1), %io(2),	- predefiniowane stałe matematyczne: I, j, pi, exp(1), inf, nan, eps, - predefiniowane wartości logiczne: logical(0), logical(1)
- skalary, wektory, macierze: 3; [1 2 3]; [1 2 3; 4 5 6];	- skalary, wektory, macierze: 3; [1 2 3]; [1 2 3; 4 5 6];
- znaki i łańcuchy znaków: 'z', "z", 'cos'; "cos"	- znaki i łańcuchy znaków: 'z'; 'cos'
- liczby zespolone: 2+1.5*%i → 2. + 1.5i	- liczby zespolone: 2+1.5*i; 2+21.5; → 2.0000 + 1.5000i

Uwaga; znakiem dziesiętnym liczb jest zawsze kropka (niezależnie od ustawień systemowych). Przecinek i średnik mają inne zastosowanie:

– przecinek – oddziela parametry funkcji oraz elementy w wierszach macierzy,

– średnik – na końcu linii wyłącza echo (wyświetlanie odpowiedzi systemu po wykonaniu polecenia), a w macierzy służy do oddzielania kolumn.

Nazwy zmiennych rozpoczynają się od litery, następnie może wystąpić dowolna kombinacja liter (bez polskich znaków), cyfr i znaków podkreślenia. Zmienne nie wymagają deklarowania. Zmienna powstaje w momencie nadania jej wartości – podstawienia liczby, znaków lub wyniku funkcji. Wartość zmiennej można sprawdzić w przeglądarce zmiennych lub po jej wpisaniu nazwy w linii komend. Zmienne można usunąć:

clear [*nazwa_zmiennej*] – kasowanie wszystkich lub podanej zmiennej

Jedna z fundamentalnych własności Matlaba i Scilaba to ukierunkowanie na operacje macierzowe (zgodne z regułami algebry liniowej). Stąd skalary i wektory są uznawane jako szczególne przypadki macierzy. Macierz można zdefiniować przez:

– wymienienie elementów (kwadratowe nawiasy, elementy w wierszu oddzielane przecinkiem lub spacją, kolumny - średnikiem), np:

wiersz=[1,3,5,9]; wiersz=[1 3 5 9]; kolumna=[2;4;3;6]; tablica=[1,2,3; 4,5,6]; tablica=[1 2 3; 4 5 6];

– wygenerować za pomocą operatora dwukropka: minimum : krok : maksimum, na przykład:

wektor = 1 : 0.5 : 3; wektor = [1 : 0.5 : 3] ; wektor = (1 : 0.5 : 3);

– za pomocą funkcji, generujących szczególne typy macierzy (np. jednostkową, jedynkową, zerową, losową):

- o zadanej ilości wierszy (w) i kolumn (k): eye(w,k); ones(w,k); zeros(w,k);rand(w,k) - o wymiarze takim jak wymiar obiektu n: eye(n); ones(n), zeros(n), rand(n) Uwaga: eye(2) → 1 *(bo obiekt „2" ma wymiar 1)*	- o zadanej ilości wierszy (w) i kolumn (k): eye(w,k); ones(w,k), zeros(w,k), rand(w,k) - o zadanym wymiarze (n): eye(n); ones(n), zeros(n), rand(n)

Odwołanie do elementów macierzy (w okrągłych nawiasach) przez:

- podanie numeru wiersza i kolumny, na przykład:

tablica(w, k); wiersz(3); kolumna(2); A (:,2)

- podanie zakresu wierszy i kolumn (od,do) za pomocą operatora dwukropka, na przykład:
  - pierwsze trzy elementy wektorów (wierszowych i kolumnowych): wiersz(1:3); kolumna(1:3);
  - drugi wiersz macierzy oraz pierwsza kolumna macierzy: tablica (2,:); tablica (:,1);

3.1.2 Operacje i funkcje matematyczne

Zestaw podstawowych operatorów matematycznych w Scilabie i Matlabie jest praktycznie taki sam. Operatory działań arytmetycznych występują w dwóch wariantach: bez i z kropką. Operatory bez kropki oznaczają operacje macierzowe, a operatory z kropką operacje na elementach macierzy.

+ - * / \ ^ (lub **) .* ./ .\ .^ ‘ .’ & | ~	+ - * / \ ^ .* ./ .\ .^ ‘ .’ & | ~ (oraz && ||)
= == < <= > >= ~= (lub <>)	= == < <= > >= ~=

Uwaga: Warunkiem koniecznym wykonania operacji na macierzach lub elementach macierzy jest zachowanie odpowiedniej zgodności wymiarów.

Nazwy podstawowych funkcji matematycznych są również w większości takie same (ale zdarzają się wyjątki):

abs, cos, sin, tan, exp, log, log10, sqrt
ceil, fix, floor, imag, real, round

modulo	mod

Parametrami wszystkich funkcji mogą nie tylko proste skalary czy zmienne ale także wyrażenia matematyczne. Jeśli parametrem funkcji będzie macierz to operacja zostanie wykonana na wszystkich elementach macierzy i wynikiem działania też będzie macierz (zgodnie z macierzowym charakterem programów).

Nazwy funkcji realizujących specjalistyczne operacje na macierzach są także w większości takie same w obu programach. Na przykład obliczanie charakterystycznych parametrów i podstawowe przekształcenia danej macierzy (A):

[w,k]=size(A), length(A), diag(A), max(A), min(A), mean(A), median(A), det(A), inv(A)

eig(A) lub spec(A)	eig(A)

Warto jednak przetestować działanie funkcji, ponieważ zdarza, że funkcje mają tą samą nazwę, ale działają inaczej. Załóżmy macierz A=[1, 2, 3; 4, 5, 6].

sum(A) → 21 - *suma w macierzy*	sum(A) → 5 7 9 - *suma w kolumnach*
sum(A, 'r') *lub* sum(A, 'row') → 5 7 9	sum(A,1) → 5 7 9
sum(A, 'c') *lub* sum(A, 'col') → 6 15	sum(A,2) → 6 15

Scilab udostępnia funkcje, które dokładnie odpowiadają funkcjom Matlaba – ich nazwy mają przedrostek mtlb* (patrz Help Browser), np.:

mtlb_max, mtlb_min, ..., mtlb_zeros, mtlb_eig, ...	max, min, ..., zeros, eig, ...

3.1.3 Operacje na tekstach

Matlab i Scilab umożliwiają definicje i operacje na zmiennych tekstowych (łańcuchach znaków). Zmienne tekstowe definiuje się przez proste podstawienie lub konstrukcję tekstu za pomocą funkcji, na przykład:

- podstawienie: txt123='123'; txtcos= 'cos'; znaka='a'; znakb='b';
- konwersję typów: int2str(12) → '12'
- formatowanie tekstu: sprintf('zmienna=%3d i tekst=%s ',12, txtcos) → 'zmienna= 12 i tekst=cos'
- łączenie tekstów: strcat(['tekst1', '+', 'tekst2']) → 'tekst1+tekst2'
- łączenie tekstów i wartości: strcat([tekst123, '+', int2str(12)])] → '123+12'

Należy zwrócić uwagę na pewne ważne różnice w obszarze operacji na tekstach, wynikające skąd, że Matlab traktuje tekst tak samo jako wektor wartości (każdy znak ma wartość wynikającą z kodu ASCII), natomiast Scilab zachowuje rozróżnienie liczby i znaku. Przykłady:

znaka + znakb → 'ab'	znaka + znakb → 195 - *'a'=61H=97, 'b'=62H=98*
tab=['a1','a2','a33'] → a1 a2 a33; tab(1) → 'a1'	tab=['a1','a2','a33'] → 'a1a2a33' tab(1) → 'a'

4. Grafika – wykresy

Typowy sposób prezentacji wyników obliczeniowych to wykresy w prostokątnym układzie współrzędnych. Podstawowe funkcje realizujące to zadanie w Matlabie i Scilabie zachowują duży poziom zgodności[1].

4.1 Okno graficzne

Wykresy (i inne elementy graficzne) są rysowane w uniwersalnym oknie graficznym, które można utworzyć wcześniej za pomocą funkcji figure. Utworzyć można wiele okien a każde z nich ma swój identyfikator (uchwyt lub wskaźnik), który można zapamiętać w momencie tworzenia. Jeśli na ekranie istnieje wiele okien to funkcje graficzne działają na oknie bieżącym (okno ostatnio utworzone lub wskazane jako bieżące). Opisane funkcje są realizowane w następujący sposób:

– utworzenie okna:

h1 = figure(); - *zawsze nawias (oraz średnik, tzn. bez echa)*

- h1 = *struktura danych okna (wskaźnik)*

fig1 = figure();
fig1 = figure;
- fig1 = *uchwyt okna (numer)*

– przełącz/utwórz bieżące okno:

figure(h1); - *lub* scf(h1); (set current figure)
figure(*numer*); - *lub* scf(*numer*);

figure(fig1) - wskazane okno zostanie wyciągnięte na wierzch
okno wyciągnięte na wierzch staje się także oknem bieżącym

– zamknięcie okna – za pomocą menu lub ikony dostępnej w oknie oraz za pomocą funkcji:

close, close(h1) - *zamknij bieżące okno i okno h1*

close, close(fig1) - *zamknij bieżące okno i okno* fig1
close all - *zamknij wszystkie okna*

Domyślne własności okien graficznych (po utworzeniu) wykazują pewne różnice w Matlabie i Scilabie:

– autokasowanie wyłączone – kolejne funkcje plot dorysowują wykresy
– numery i tytuły kolejnych okien:
0 - Okno graficzne numer 0
1 - Okno graficzne numer 1
...

– autokasowanie włączone - kolejne funkcje plot kasują poprzednią zawartość okna
– numery i tytuły kolejnych okien:
1 - Figure 1
2 - Figure 2
...

Więcej informacji o własnościach okien graficznych można uzyskać wykorzystując obiektową strukturę tych obiektów (p.9.1).

4.2 Wykresy 2D

4.2.1 Funkcja plot

Funkcja plot rysuje w bieżącym oknie graficznym układ współrzędnych oraz wykres, zdefiniowany za pomocą wektorów ze współrzędnymi osi 0x i 0y kolejnych punktów wykresu. Wektory mogą być zdefiniowane lub obliczone przed wywołaniem funkcji, lub obliczane na podstawie wzoru. Warunkiem koniecznym jest by podane wektory (lub ich fragmenty) były tej samej długości.

```
x=1:5;  y=2.*x+1;  plot(x, y);
x=1:5;  plot(x, 2.*x+1);
plot((1:5), 2.*(1:5)+1);
x=1:5;  y=3:2:20; plot(x, y(1:5));
```

[1] zgodność tą osiągnięto po przebudowaniu funkcji graficznych Scilaba (od wersji 5.1)

Wykresy mogą być rysowane różnymi kolorami i typami linii, a w jednym układzie współrzędnych można narysować wiele wykresów:

```
plot(x1, y1, format1, x2, y2, format2, ...);
```

W ciągu formatującym linii używane są następujące symbole:

- kolor linii: r g b c m y k w (red, green, blue, cyan, magenta, yellow, black, white)
- styl linii: - -- : -. (ciągła, przerywana, kropkowana, kropki i linie)
- znacznik: + o * . x ^ v > < 's' (lub 'square') 'd' (lub 'diamond') 'p' (lub 'pentagram')

Przykład – kilka wykresów w różnych kolorach w jednym układzie współrzędnych:

```
t=0 : 3.14/20 : 2*3.14;
plot(t, sin(t), 'ro-.', t, cos(t), 'cya+', t, abs(sin(t)), '--mo')
```

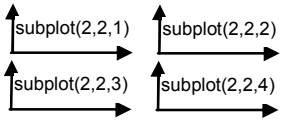
Okno graficzne można podzielić na kilka części za pomocą funkcji subplot i każdą wypełniać w różny sposób (np. używając funkcji plot lub innych):

subplot(*nmp*) lub subplot(*n, m, p*)

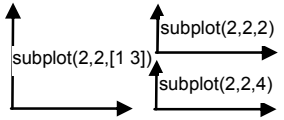
gdzie: *n* - ilość wierszy, *m* - ilość kolumn, *p* – numer wybranej części w i-tym wierszu i j-tej komunie liczona wg wzoru p= (i-1)*n + j

Przykład – podział okna graficznego na kilka części, które można wypełnić na przykład oddzielnymi układami współrzędnych

```
subplot(2,2,1)
...
subplot(2,2,2)
...
```




```
subplot(2,2,1)
...
subplot(2,2,2)
...
```




```
subplot(2,2,[1 3])
...
subplot(2,2,2)
...
```

Funkcja subplot w Matlabie daje większe możliwości niż symetryczny podział okna na kilka wierszy i kolumn.

4.2.2 Podstawowe funkcje do formatowania wykresu

Dokumentacja badań wymaga opisania wykresów - tytułów, osi, poszczególnych krzywych. Najprostszym sposobem jest wykorzystanie następujących funkcji, które podobnie jak funkcja plot działają na bieżącym oknie:

```
title('tytul wykresu');
xlabel('Czas [s]');
ylabel('Wyjście');
xstring(1, 27, 'dowolny'); - tekst na wykresie
legend('A', 'B');          - uruchamiać po narysowaniu wykresów
```

```
title('tytul wykresu');
xlabel('Czas [s]');
ylabel('Wyjście');
text(1,27, 'dowolny');     - tekst na wykresie
legend('A', 'B');          - uruchamiać po narysowaniu wykresów
```

Wśród wielu innych własności wykresów koniecznie należy wymienić jeszcze dwie operacje (w prosty sposób dostępne tylko w Matlabie):

– włączenie lub wyłącznie autokasowania wykresów podczas wykonywania kolejnych operacji plot:

```
set(gca(), 'auto_clear ', 'off ');      - autokasowanie wył,  domyślne
set(gca(),'auto_clear ', 'on ');        - autokasowanie wł
```

```
hold on      - zatrzymanie zawartości wł (autokasowanie wył )
hold off     - zatrzymanie zawartości wył (autokasowanie wł), domyślne
```

– włączenie lub wyłącznie siatki:

```
xgrid(1); lub set(gca(),'grid', [1,1]);      - wł siatki (na obu osiach)
xgrid(8); lub set(gca(),'grid', [-1,-1]);    - wył siatkę, domyślne
```

```
grid on      - wł siatki (na obu osiach)
grid off     - wył siatki (na obu osiach), domyślne
```

5. Programowanie

5.1 Instrukcje sterujące

Poza opisanymi powyżej komendami systemowymi oraz funkcjami i operacjami matematycznymi i tekstowymi, interpretery poleceń w Matlabie i Scilabie obsługują jeszcze typowe instrukcje sterujące działaniem programów: instrukcje warunkowe if i swich oraz pętle for i while.

W poniższym zestawieniu instrukcji sterujących Matlaba i Scilaba wyróżniono kolorem różnice w składni (Scilab oferuje różne warianty składni, co wynika z konieczności zachowania zgodności z Matlabem i ze swoimi starszymi wersjami). Najpoważniejsze różnice to oznaczenie komentarza (// lub %) i instrukcja switch (słowo kluczowe select lub switch).

// komentarz (do końca linii)		% komentarz (do końca linii)

if *warunek1*
  kod1
[**elseif** *warunek2*
  kod2]
[**else**
  kod3]
end
	if *warunek1* **then**
	  kod1
	[**elseif** *warunek2* **then**
	  kod2]
	[**else**
	  kod3]
	end
		if *warunek1*
		  kod1
		[**elseif** *warunek2*
		  kod2]
		[**else**
		  kod3]
		end

select *wyrazenie*
case *wartosc1*
  kod1
case *wartosc2*
  kod2
[**else**
  kod3]
end
	select *wyrazenie*
	case *wartosc1* **then**
	  kod1
	case *wartosc2* **then**
	  kod2
	[**else**
	  kod3]
	end
		switch *wyrazenie*
		case *wartosc1* **then**
		  kod1
		case *wartosc2* **then**
		  kod2
		[**otherwise**
		  kod3]
		end

for *i=pocz:krok:koniec*
  kod
end
	for *i=pocz:krok:koniec* **do**
	  kod
	end
		for *i=pocz:krok:koniec*
		  kod
		end

while *warunek*
  kod1
[**else**
  kod2]
end
	while *warunek* **do**
	  kod1
	[**else**
	  kod2]
	end
		while *warunek*
		  kod1
		end

	while *warunek* **then**
	  kod1
	[**else**
	  kod2]
	end	

Zaleca się na zakończenie każdej funkcji kodu wstawiać znak średnika. Brak średnika oznacza włączenie echa, czyli wyświetlanie odpowiedzi na konsoli komend, co pomaga śledzić przebieg realizacji funkcji ale spowalnia obliczenia (w Scilabie może dodatkowo wstrzymywać realizację funkcji ponieważ czasem trzeba potwierdzić coś naciskając klawisz).

Instrukcje, podobnie jak inne polecenia i funkcje, mogą być zapisywane w jednej linii, a do oddzielenia poszczególnych części stosuje się przecinek albo średnik, przy czym średnik jednocześnie wyłącza echo. Zapis złożonych instrukcji w jednej linii umożliwia ich zastosowanie nie tylko w skryptach i funkcjach ale także w linii komend. Na przykład:

if *warunek*, *kod1*, **else** *kod2*, **end** **if** *warunek* **then** *kod1*, **else** *kod2*, **end**	**if** *warunek*, *kod1*, **else** *kod2*, **end**
select *wyrazenie*, **case** *wartosc1*, *kod1*, **else** *kod2*, **end**	**switch** *wyrazenie*, **case** *wartosc1*, *kod1*, **otherwise** *kod2*, **end**
for *i=pocz:krok:koniec*, *kod*, **end** **for** *i=pocz:krok:koniec* **do** *kod*, **end**	**for** *i=pocz:krok:koniec*, *kod*, **end**
while *warunek*, *kod1* [,**else** *kod2*], **end** **while** *warunek* **do** *kod1* [,**else** *kod2*], **end**	**while** *warunek*, *kod1*, **end**

Można również dzielić długie polecenia - wstawiając na końcu linii trzy kropki po spacji (**...**) i wówczas następna linia będzie interpretowana jako ciąg dalszy polecenia.

5.2 Skrypty

Skrypt to zewnętrzny plik tekstowy zawierający sekwencje instrukcji i poleceń, które mogą być zrealizowane w przestrzeni roboczej (Workspace) Matlaba/Scilaba, to znaczy są uruchamiane w środowisku Matlaba/Scilaba i działają na zmiennych w tym środowisku. Uruchomienie wykonuje się zwykle pod edytorem tekstowym lub z konsoli Matlaba/Scilaba (p. 2.2.2). Działanie skryptu jest takie samo jak działanie analogicznej sekwencji poleceń wykonywanej w linii komend – można więc również zawartość skryptu skopiować do konsoli komend i tam wykonać, również wówczas gdy skrypt zawiera instrukcje zapisane w kilku liniach, ponieważ uruchomienie skopiowanej sekwencji następuje dopiero po naciśnięciu klawisza Enter.

Zastosowanie instrukcji sterujących pozwala na zautomatyzowanie realizacji programu badań. Warto je stosować nawet do realizacji prostych zadań jako pewną formę dokumentowania przeprowadzonego badania, które można poza tym zawsze powtórzyć.

Skrypty mogą zawierać wszystkie rodzaje poleceń (komendy systemowe, funkcje i operacje matematyczne i tekstowe) oraz odwołania do innych skryptów i funkcji zdefiniowanych przez użytkownika. Skrypty mogą odwoływać się do wszystkich zmiennych w przestrzeni roboczej a także definiować nowe zmienne w tej przestrzeni, które w niej pozostają po zakończeniu działania skryptu. Przekazywanie informacji pomiędzy skryptami następuje poprzez takie zmienne, ewentualnie za pośrednictwem różnego typu plików (skrypty nie zwracają wartości).

5.3 Funkcje użytkownika

Funkcja to sekwencja instrukcji i poleceń, zwykle zapisana w zewnętrznym pliku, która może mieć zdefiniowane parametry wejściowe (argumenty) i która może zwracać wartości parametrów wyjściowych (wynik).

5.3.1 Funkcje definiowane z linii komend

Jednym ze sposobów definiowania prostych funkcji użytkownika jest zastosowanie funkcji definiującej, w której podawana nazwa, wzór i parametry definiowanej funkcji

deff('[*wy*]=*nazwa*(*arg1*, *arg2*, ...)', '*wy=wzor*')	*nazwa* = **inline**('*wzor*', '*arg1*', '*arg2*, ...')

Przykład funkcji definiowanych bezpośrednio z linii komend: 1) z=sin(x)*cos(x), 2) $y=x_1+x_2$

- definicja 1: **deff**('y=z(x)', 'y=cos(x)+sin(x)', 'x') - definicja 2: **deff**(' [y]=mojplus(x1,x2)', 'y=x1+x2') - wywołania: z(%pi), mojplus(1,2) → -1 3	- definicja 1: z=**inline**('cos(x)+sin(x)', 'x'); - definicja 2: mojplus = **inline**('x1+x2', 'x1', 'x2') - wywołania: z(pi), mojplus(1,2) → -1 3

Funkcje zdefiniowane w ten sposób są przechowywane w przestrzeni roboczej do czasu zakończenia pracy z programem.

5.3.2 Funkcje definiowane w pliku

Funkcje definiowane przez użytkownika w pliku tekstowym muszą zachować strukturę, która pozwala określić nazwę funkcji oraz jej parametry wejściowe i wyjściowe (to główna cecha, na podstawie której programy odróżniają funkcję od skryptu[1]). W typowej definicji funkcja jest zawarta w pliku o tej samej nazwie:

Nazwa pliku: *nazwafun*.sci

```
function[wy1, ...] = nazwafun(arg1,...)
  instrukcje
  wy1=...                    – podstawienie wartości wyjściowej
endfunction
```

Nazwa pliku: *nazwafun*.m

```
function[wy1, ...] = nazwafun(arg1,...)
  instrukcje
  wy1=...                    – podstawienie wartości wyjściowej
[end]                        – niekonieczne jeśli jedna funkcja
```

Jako przykład takiej definicji przedstawiono zawartość pliku o nazwie 'iloczyn' (obliczenie iloczynu liczb naturalnych od 1 do n):

```
function[a] = iloczyn(n)
    a=1
    for i = 1:n;  a = a*i;  end
endfunction
```

```
function[a] = iloczyn(n)
    a=1
    for i = 1:n;  a = a*i;  end
end
```

Wywołanie funkcji zapamiętanej w pliku 'iloczyn' ma postać:

– wczytanie funkcji z pliku do przestrzeni roboczej:
  exec(„iloczyn.sci”);
– użycie funkcji (wywołanie funkcji z przestrzeni roboczej):
  x = iloczyn(4)

– plik z funkcją musi być w kartotece bieżącej lub zdefiniowanej w ścieżkach przeszukiwania (główna konsola: File/Set Path)
– użycie funkcji
  x = iloczyn(4)

W kolejnym przykładzie plik 'modulo10' zawiera definicje dwóch funkcji:

```
function [ile, reszta, pelne]=modulo10(n)
   reszta = modulo(n,10);
   ile = (n-reszta)/10;
   pelne = pelne10(n);
endfunction

function [p]=pelne10(n)
   if modulo(n,10)> 0; p=0; else p=1; end
endfunction
```

```
function[ile, reszta, pelne] = modulo10(n)
   reszta = mod(n,10);
   ile = (n-reszta)/10;
   pelne = pelne10(n);
end

function[p] = pelne10(n)
   if mod(n,10)> 0; p=0; else p=1; end
end
```

Główna funkcja modulo10 zwraca tym razem trzy parametry i używa funkcji pomocniczej pelne:

```
exec('modulo10.sci');        – wczytanie funkcji z pliku
[x r dzies]  = modulo10(12)
dzies2 = pelne10(14)         – wykorzystanie f.pomocniczej
```

```
[x r dzies]  = modulo10(12)
%dzies2 = pelne10(12)        - f.pomocnicza niedostępna
```

W przypadku Scilaba nazwy funkcji nie muszą być takie same jak nazwy plików i wszystkie funkcje w zdefiniowane pliku są dostępne (także funkcje pomocnicze), ponieważ program nie poszukuje funkcji w plikach ale w przestrzeni roboczej. Informację o wczytanych funkcjach można uzyskać za pomocą komendy whos[2].

[1] Scilab przewiduje rozszerzenie *.sce dla pliku skryptu i *.sci dla pliku funkcji, ale ma to charakter porządkowy bo nazwę pliku podaje się zawsze z rozszerzeniem
[2] komenda wyświetla wszystkie nazwy występujące w przestrzeni roboczej (zmienne, funkcje, biblioteki, ...)

W funkcji można operować na parametrach wejściowych i wyjściowych ale tylko wartości parametrów wyjściowych zostaną przekazane do miejsca wywołania funkcji. Zmienne zainicjowane we funkcji są zmiennymi lokalnymi i znikają po wyjściu z funkcji. Poza tym w funkcji można zainicjować i odwoływać się do zmiennych globalnych. W Scilab można też odczytać wartości zmiennych z przestrzeni roboczej ale w momencie zapisania wartości tworzony jest lokalny duplikat zmiennej, który znika po wyjściu z funkcji (zmiany nie zostaną przekazane do miejsca wywołania). Funkcje są bardziej czytelne jeśli zamiast odwoływania się do zmiennych z przestrzeni roboczej czy zmiennych globalnych ich wartości zostaną przekazane poprzez listę parametrów wejściowych.

Poza typowymi instrukcjami i poleceniami, wewnątrz funkcji stosuje się także polecenie wyjścia z funkcji w określonym miejscu oraz odczytanie ilości parametrów wejściowych i wyjściowych:

return	return
[lsh rhs] = argn()	nargin, nargout

5.4 Zasięg zmiennych

Podstawowy obszar pamięci, na którym operują Matlab/Scilab nazywany jest przestrzenią roboczą (Workspace). W przestrzeni roboczej istnieją zmienne, które powstają przez realizację poleceń w linii komend lub w skrypcie. Zmienne z przestrzeni roboczej można:

odczytać – w linii komend, w skryptach, w funkcjach[1]	odczytać – w linii komend, w skryptach
zapisać - w linii komend, w skryptach	zapisać - w linii komend, w skryptach

Poza zmiennymi w podstawowej przestrzeni roboczej programu można definiować zmienne globalne, które są dostępne do odczytu i zapisu we wszystkich obszarach (skryptach, funkcjach), gdzie zostaną zadeklarowane jako globalne. Dla zachowania przejrzystości skryptów i funkcji powinno to być rozwiązanie stosowane z dużym umiarem.

Zmienne lokalne to dostępne tylko w obszarze funkcji, w której zostały zdefiniowane – zanikają po wyjściu z funkcji.

[1] lepiej unikać tego rozwiązania i przekazywać wartość przez listę parametrw funcji

6. Elementarne przykłady zastosowania

6.1 Badanie przebiegu funkcji

6.1.1 Wykresy funkcji (dwu- i trójwymiarowe)

Jedno z najprostszych zastosowań Matlaba i Scilaba to rysowanie wykresów funkcji zadanych w postaci wzoru. Dla ilustracji zagadnienia narysujmy rodzinę funkcji $y = ax^2$, dla trzech różnych wartości parametru a, którym odpowiadają trzy różne kolory i typy linii. Realizacja zadania została przedstawiona w dwóch wariantach - przy użyciu najprostszych komend oraz z elementami programowania:

```
x = [0:2:20];
plot(x, x.^2*1,'r-',x, x.^2*5,'g--',x, x.^2*6,'b.' );
xgrid(1);                    - lub set(gca(),'grid', [1,1]);
```

```
x = [0:2:20];
plot(x, x.^2*1,'r-',x, x.^2*5,'g--',x, x.^2*6,'b.' );
grid on
```

```
figure();set(gca(), 'auto_clear ', 'off '); xgrid(1);
a = [1 5 6];
x = [0:2:20];
typ=['r-','g--','b.'];          - tablica tekstów
for i=1:3, plot(x, x .^2*a(i), typ(i)); end
```

```
figure, hold on, grid on
a = [1 5 6];
x = [0:2:20];
typ=['r- ';'g--';'b. '];        - tablica 2-wymiarowa (spacje uzupełaniające)
for i=1:3, plot(x, x .^2.*a(i), typ(1,:)); end
```

Poza podstawowymi wykresami dwuwymiarowymi dostępne są również różne warianty wykresów trójwymiarowych. Przykład takiego wykresu przedstawiony poniżej przedstawia wykres funkcji $z = x \cdot e^{\left(-x^2 - y^2\right)}$, gdzie dziedzinę funkcji wyznaczają x=-2÷2 i y=-2÷3.

```
[x, y] = meshgrid(-2 : .2 : 2 , -2 : .2 : 3);
z = x .* exp(-x .^2 - y.^2);
surf(z);                    - lub mesh(z);
```

```
[x, y] = meshgrid(-2 : .2 : 2 , -2 : .2 : 3);
z = x .* exp(-x .^2 - y.^2);
surf(z);                    - lub mesh(z);
view(obrót_poziomy, obrót_pionowy);
```

Postać wykresu 3D zależy od wybranej funkcji graficznej i jest różna pod Matlabem i Scilabem.

6.1.2 Poszukiwanie miejsc zerowych i ekstremów funkcji

Podstawowe badania przebiegu zmienności funkcji obejmują wyznaczanie miejsc zerowych i ekstremów. Przykłady funkcji, które realizują najprostsze zadania dotyczące przypadku funkcji nieliniowej jednej zmiennej przedstawiono poniżej:

```
y = fsolve(x1, fnazwa)
```

```
fzero(badana_funkcja,  zakres_zmiennej_wejściowej)
fminbnd(badana_funkcja, zakres_zmiennej_wejściowej)
```

Dokładny sposób i przykłady zastosowania funkcji opisano w pomocy programów – poniżej dla ilustracji dwa proste wywołania:

```
xzero = fzero(@sin, 3)
xmin = fminbnd(@sin, 0, 2*pi)
```

Już na przykładzie tych dwóch funkcji widoczne są różnice, które dotyczą tego obszaru zastosowania Matlaba i Scilaba, począwszy od nazw funkcji i możliwości definiowania badanej funkcji.

6.2 Rozwiązywanie zagadnień algebraicznych

6.2.1 Definicja i operacje na wielomianach

Najprostszym sposobem definicji wielomianu jest wykorzystanie wektorów, to znaczy współczynniki wielomianu są zapisywane w wektorze, w którym pierwszy element odpowiada współczynnikowi przy najwyższej potędze zmiennej, na przykład:

- wielomian $x^2 + 2x + 5$ jest reprezentowany przez wektor `w1 = [1 0 0 2 5];`

Wektor współczynników wielomianu może być parametrem funkcji, na przykład do wyznaczenia pierwiastków wielomianu (w1):

```
p1=roots(w1)   → Scilab  1.0687573 + 1.2688874i      Matlab   1.0688 + 1.2689i
                         1.0687573 - 1.2688874i               1.0688 - 1.2689i
                       - 1.0687573 + 0.8212241i             -1.0688 + 0.8212i
                       - 1.0687573 - 0.8212241i             -1.0688 - 0.8212i
```

Kolejnym sposobem definicji wielomianu jest zapis symboliczny za pomocą zadeklarowanej wcześniej zmiennej symbolicznej (x):

```
x = poly(0, 'x');          - definicja zmiennej wielomianu (symbolu)
w2 = 1 + x + 2*x^2;        → 1 + x + 2x^2
```

Symboliczna postać wielomianu może być również używana jako parametr funkcji, obsługujących wielomiany, np.: `roots(w2)` .

Wielomian można również odtworzyć na podstawie jego pierwiastków, przekazanych do funkcji w postaci wektora, np.:

```
w3 = poly([1,2],'x')   → 2 - 3x + x^2
                       - wielomian w postaci symbolicznej
                       - sprawdzenie: roots(w3)  → 1  2
```

```
w3 = poly([1,2])   → 1   -3   2    czyli  x^2 - 3x + 2
                   - wielomian w postaci wektora
                   - sprawdzenie: roots(w3)  → 1  2
```

Inne funkcje - patrz przeglądarki pomocy na temat:

Polynomials | Matlab / Functions / Mathematics / Polynomials

6.2.2 Układy równań algebraicznych

Układy równań algebraicznych można zapisywać w sposób macierzowy, to znaczy układ m równań z n zmiennymi (x_1, ..., x_n) zapisany w postaci macierzowej $\mathbf{A}x = \mathbf{b}$ jest reprezentowany przez macierz A o wymiarach m x n, wektor kolumnowy b zawierający m wartości oraz wektor kolumnowy x reprezentujący zmienne układu. Wykonując operacje na macierzach można wyznaczyć wzór na poszukiwany wektor rozwiązań $x = \mathbf{A}^{-1}\mathbf{b}$ i zrealizować go różne sposoby:

- `x = A \ b` (zalecane rozwiązanie z zastosowaniem operatora lewostronnego dzielenia macierzy)
- `x = inv(A) * b`
- `x = A ^ (-1) * b`

6.3 Analiza danych

6.3.1 Interpolacja

Interpolacja polega na znalezieniu krzywej, która przechodzi przez zadane punkty nazywane węzłami interpolacji.

yi = interp1(xw, yw, xi);
yi = interp1(xw, yw, xi ,'*metoda*', [ekstrapolacja]);
gdzie: '*metoda*' = 'linear', 'spline', 'nearst'
	yi = interp1(xw, yw, xi);
	yi = interp1(xw, yw, xi ,'*metoda*', [ekstrapolacja]);
	gdzie: '*metoda*' = 'linear', 'spline', 'nearst', 'cubic'

Patrz też: interp2d, interp3d, interpln, interp	Patrz też: interp2, interp3, interp1q, pchip, interpft, interpn, spline

Dla ilustracji przykład interpolacji krzywej (wektory xi i yi) na podstawie kilku punktów funkcji sin (wektory xw i yw)

```
x=0:10; y=sin(x); xi=0:0.1:10;
yi = interp1(x, y, xi);
plot(x, y, 'o', xi, yi, '--')
```

```
x=0:10; y=sin(x); xi=0:0.1:10;
yi = interp1(x, y, xi);
plot(x, y, 'o', xi, yi, '--')
```

Możliwości powyższych funkcji w Matlabie i Scilabie różnią się w zakresie ekstrapolacji, czyli określania wartości funkcji poza znanym obszarem.

6.3.2 Aproksymacja

Aproksymacja oznacza przybliżanie, czyli zastąpienie jednych wartości innymi. Typowe zastosowanie to aproksymacja serii danych wielomianem wskazanego stopnia.

```
p = polyfit(x, y, stopien);
```

W przykładzie poniżej serię pomiarową (wektory x i y) wygenerowano jako fragment przebiegu sinusoidalnego, który próbowano aproksymować wielomianami drugiego i trzeciego stopnia – wynikiem jest wektor współczynników wielomianu (od najwyższej potęgi):

```
x=0:0.1:1; y=sin(x);
p2 = polyfit(x,y,2);
p3 = polyfit(x,y,3);
```

6.3.3 Całkowanie i różniczkowanie

Przykładem inne typowych operacji związanych z analizą danych jest całkowanie i różniczkowanie krzywych zadanych przez wektory wartości:

```
z = inttrap([x,] y);      - całkowanie metodą trapezów
```

```
z = trapz([x,] y);        - całkowanie metodą trapezów
```

```
y = diff(x [,n]); - różnica kolejnych wartości (n-ilość powtórzeń)
```

Poniżej wykonano aproksymację pola pod „połówką” sinusa i obliczono aproksymowany przebieg pochodnej tej krzywej:

```
X = 0:%pi/100:%pi;  Y = sin(X);   - „połówka” sinusa
Z = inttrap(X,Y)                  → Z = 1.9998355 (pole pod krzywą)
X1 = diff(X);                     - wektor przyrostów zmiennej X
diff(Y)./diff(X)                  - aproksymacja pochodnej dY/dX
```

```
X = 0:pi/100:pi;  Y = sin(X);     - „połówka” sinusa
Z = trapz(X,Y)                    → Z = 1.9998 (pole pod krzywą)
X1 = diff(X);                     - przyrosty zmiennej X
diff(Y)./diff(X)                  - aproksymacja pochodnej dY/dX
```

W powyższym przykładzie przebieg został określony wzorem, można więc zastosować alternatywny sposób wyznaczenia pola:

```
Z = integrate('sin(x)', 'x', 0, %pi)     → Z = 2
```

6.4 Rozwiązywanie równań różniczkowych zwyczajnych

Symulacyjne rozwiązywanie równań różniczkowych to jedna z kluczowych funkcjonalności oprogramowania do obliczeń naukowo-inżynierskich. Dostępne są różne algorytmy (ang. solver) obliczania rozwiązania równań różniczkowych przy zadanych warunkach początkowych, różniące się dokładnością, szybkością działania, przeznaczeniem do określonego typu równań[1]. W funkcjach uruchamiających obliczenia poza równaniem różniczkowym i warunkami początkowymi oraz nazwą algorytmu podaje się zakres zmiennej niezależnej (często jest to czas *t*) i parametry algorytmu:

x = ode(**x0**, **t0**, **t**, **funkcja**) [**x**, w, iw]=ode([type], **x0**, **t0**, **t** [,rtol [,atol]], **funkcja** [,jac] [,w,iw])	[t,**x**] = *solver* (**'funkcja'**, [**t0** tf], **x0**) [t,**x**] = *solver* (**'funkcja'**, [**t0** tf], **x0**, options, p1, p2, ...)

Podstawowe parametry funkcji:

x – wynik obliczeń, **x0** – warunki początkowe, **t0** – czas początkowy, **funkcja** – nazwa pliku z równaniami różniczkowymi

t – wektor czasu	**[t0 tf]** – wektor z czasem początkowym i końcowym
Wybór solvera przez parametr type : 'lsoda' (domyślnie) 'adams', 'stiff', 'rk', 'rkf', 'fix', 'discrete', 'roots' Opcje solvera zawane w parametrach funkcji: rtol, atol	Wybór solvera przez nazwę funkcji *solver*: ode45 (najczęściej) ode23, ode113, ode15s, ode23s, ode23t, ode23tb, Opcje solvera options definiowane za pomocą funkcji, np.: options = odeset('RelTol', 1e-4, 'AbsTol', [1e-4 1e-4 1e-5]);

Równania różniczkowe, które solvery potrafią rozwiązać mogą być liniowe lub nieliniowe ale muszę to być równania pierwszego rzędu lub układy równań pierwszego rzędu [1][7]. Równania różniczkowe należy zapisać w pliku funkcyjnym.

1. Przykład równania pierwszego rzędu: $\dot{x}(t) = \sin(t * x(t)),\ x(0) = 0.2$

```
function [wynik]=sin_rownanie(t, x)
   wynik = sin(t*x);
endfunction
exec('sin_rownanie.sci');          //wczytanie funkcji
x0 = 0.2 ;  t = 0:0.1:15;          //war.początkowe i wektor czasu (50s)
x= ode(x0, 0, t, sin_rownanie);
plot(t, x)
```

```
function[wynik] = sin_rownanie(t, x)
 wynik = sin(t*x);


x0 = 0.2;  tend = 15;          %war.początkowe i czas trwania
[t,x] = ode45('sin_rownanie',[0 tend],x0);
plot(t, x)
```

2. Przykład równania drugiego rzędu: $\ddot{x}(t) - c\left(1 - x^2(t)\right)\dot{x}(t) + x(t) = 0 \Rightarrow \begin{bmatrix} x = x_1 & \leftarrow x \\ \dot{x}_1 = x_2 & \leftarrow \dot{x} \\ \dot{x}_2 & \leftarrow \ddot{x} \end{bmatrix} \Rightarrow \begin{cases} \dot{x}_1(t) = x_2(t) \\ \dot{x}_2(t) = c\left(1 - x_1^2(t)\right)x_2(t) - x_1(t) \end{cases}$

```
function [xprim]=vdp_rownanie(t, x)
  global c;                        //działa też bez globlal ale nie polecane
  xprim(1) = x(2);
  xprim(2)=c* (1 - x(1)^2) * x(2) - x(1);
endfunction
global c
c=10;
x0 = [-2.5; 2.5];  t =0 :.01:50;   //war.początkowe i wektor czasu (50s)
exec('vdp_rownanie.sci');          //wczytanie funkcji
[x] = ode(x0, 0, t, vdp_rownanie);
plot( t, x(1,:),'r' )
```

```
function[xprim] = vdp_rownanie(t, x)
 global c;                         %koniecznie global
 xprim = [x(2); c* (1 - x(1)^2) * x(2) - x(1)];


global c
c=10;
x0 = [-2.5; 2.5]; tend=50;         %war.początkowe i czas trwania = 50 sek

[t, x] = ode45('vdp_rownanie',[0 tend],x0);
plot( t,x(:,1),'r' );
```

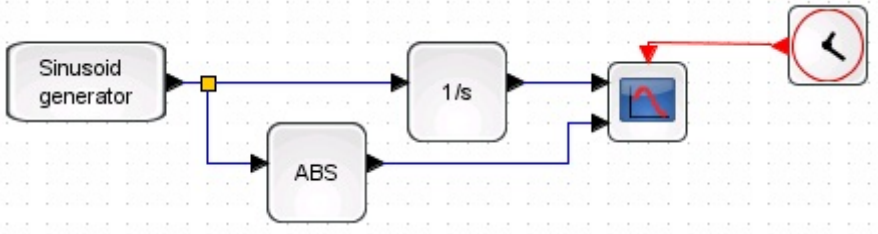
[1] np. równania „zwykłe" (nonstiff) i znacznie trudniejsze do obliczania równania sztywne (stiff)

7. Badania układów dynamiki w trybie graficznym

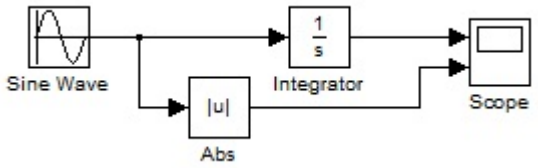
Cenioną przez użytkowników własnością opisywanych programów obliczeniowych jest możliwość graficznego definiowania badanych układów. Tą funkcjonalność zapewniają dodatkowe narzędzia: Xcos w Scilabie i Simulink w Matlabie.

7.1 Sterowanie symulacją

Scilab jest systemem sterowanym zdarzeniami, które w najprostszym przypadku są generowane przez blok zegara. Stąd charakterystyczną cechą schematów Xcos są dwa typy linii i bloków: czarne (sygnały przetwarzane) i czerwone (sygnały sterujące).



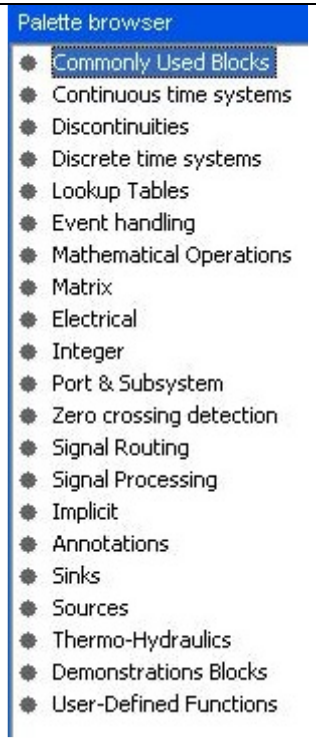
Dobór kroków obliczeniowych zależy od solvera, który optymalizuje czas i dokładność obliczeń oraz ilość danych, które są przekazywane na zewnątrz. *Ilość danych zależy od czasu próbkowania. Niektóre bloki mają na liście parametrów czas próbkowania (Sample time), ale zwykle pozostawia się wartość domyślną (optymalizowaną przez solver).*



7.2 Biblioteki elementów - przegląd

7.2.1 Przeglądarka elementów i dostęp do pomocy

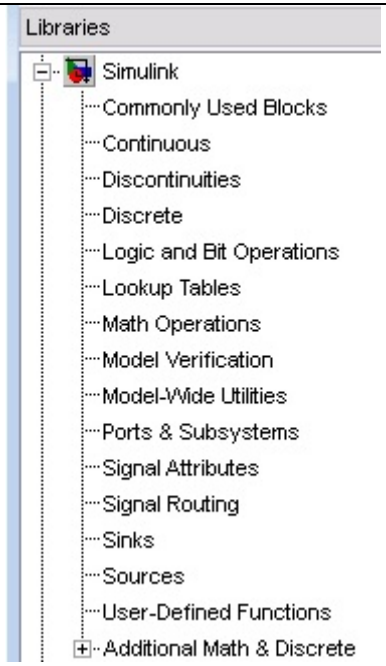
Palette Browser



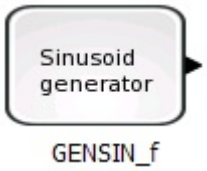
Biblioteki wspólne

Commonly Use Blocks – wybór najczęściej używanych
Continuous – ciągłe układy dynamiki
Discontinues – układy nieciągłe
Discrete – dyskretne układy dynamiki
Logic and Bit Operations
Lookup Tables
Event handling
Mathematical Operations – operacje i funkcje matematyczne
Matrix, Electrical, Integer / Model Verification, Model-Wide Utilities
Port & Subsystem – porty i podsystemy
Zero crossing detection Signal Attributes
Signal Routing – łączenie sygnałów i wektorów
Signal Processing, Implicit, Annotations
Sinks - wyjścia
Sources - źródła
Thermo-Hydraulics, Demostrations Bloks
User-Defined Functions – funkcje użytkownika (np., wyrażenia)
Additional Math & Discrete

Simulink Library Browser



7.2.2 Podstawowe źródła sygnałów

Source	Sources
Wymuszenie stałe (liczba, zmienna, wyrażenie)	
 *Commonly UB*	Parametry: – Constant
 *Commonly UB*	Parametry (wybrane): – Constant value
Wymuszenie skokowe – w chwili skoku (t_s) sygnał o wartości początkowej *wp* zmienia się do wartości końcowej *wk*	
	Parametry: – Step time (t_s) – Initial value (*wp*) – Final value (*wk*)
	Parametry (wybrane): – Step time (t_s) – Initial value (*wp*) – Final value (*wk*)
Generator sinus	
	Parametry: – Magnitude (=1), – Frequency (rad/s) (=1) – phase (0=)
	Parametry: – Amplitude (=1), – Bias (=0) – Frequency (rad/s) (=1) – phase (rad) (0=)
Zegar	
	Parametry: – Period (=0.1) - co jaki czas generuje zdarzenie – Init time (=0.1) – czas pierwszego zdarzenia Do sterowania zbieraniem danych
	Parametry: – Display time (decimation) Podaje bieżący czas symulacji (momenty w których wykonywane są obliczenia)

7.2.3 Zbieranie danych – wybrane funkcje

Sinks | **Sinks**

Oscyloskop - jedno wejście na które można podać pojedynczy sygnał lub wektor (rysuje w jednym układzie współrzędnych)

Parametry (wybrane, które trzeba dobrać):
- Ymin, Ymax – wyświetlany zakres wartości
- Refresh period – wyświetlany zakres czasu
- Buffer size – wielkość bufora danych -po wypełnieniu zawartość bufora i okna jest kasowana na nowe dane

Blok nie ma automatycznego skalowania

Patrz też blok CMSCOPE – kilka wejść, 1 okno, oddzielne wykresy

Commonly UB

Parametry (wybrane) - ikona Parameters:
- General – ilość osi, zakres czasu
- Data History – wielkość bufora danych oraz możliwość zapisywania do zmiennej

Blok ma automatyczne skalowanie w trakcie symulacji

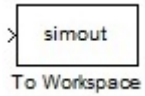
Patrz też blok XY Graph

Zbieranie danych – sygnał wejściowy zostanie zapamiętany w zmiennej o podanej nazwie (*xx*)

Parametry (wybrane, które trzeba dobrać):
- Size of buffer – wielkość bufora danych
- Scilab variable name (*xx*)

Zmienna jest strukturą wektorów: values + time

Uwaga: Zmienna zostanie utworzona w trakcie symulacji

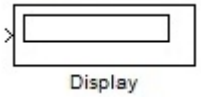


Parametry (wybrane):
- Variable name (*xx*)
- Save format – Structure with time, Structure, Array

Uwaga: Domyślnie ilość danych nieograniczona

Wyświetlacz cyfrowy – pokazuje wartość pojedynczego sygnału lub wektora

Parametry (wybrane):
- Input Size - wymiar macierzy
- Total number of digits
- Number of rational part digits



Parametry (wybrane):
- Format (np. short, long, …)
- Decimation

7.2.4 Sygnały i wektory sygnałów – wybrane funkcje

Signal Routing | **Signal Routing**

Multiplekser – łączenie sygnałów w wektor

Parametry:
- Number of input ports or vector of size

Commonly UB

Parametry:
- Number of input – ilość lub wektor, np. [2,1,3]
- Display option (none, bar, signals)

Demultiplekser - rozdzielanie wektora sygnałów

Commonly UB

Parametry:
- Number of output ports or vector of size

Uwaga: Należy podać podział wektora na sygnały, np. podział 2-sygnałowego wektora na pojedyncze sygnały = [1,1] (podanie ilości wyjść nie działa dobrze)

Commonly UB

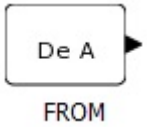
Parametry:
- Number of input – ilość lub wektor, np. [2,1,3]
- Display option (none, bar, signals)

Etykieta „Go to”

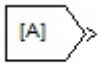
Parametry:
- Tag – domyślnie A (nazwa widoczna na bloku)
- Tag Visibility - zasięg

Parametry:
- Go Tag – domyślnie A (nazwa widoczna na bloku)
- Tag Visibility – zasięg

Etykieta „From”



Parametry:
- Tag – domyślnie A (nazwa widoczna na bloku)

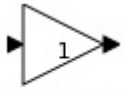


Parametry:
- Tag – domyślnie A (nazwa widoczna na bloku)

7.2.5 Operacje i wyrażenia matematyczne – podstawowe funkcje

Mathematical Operations	Math Operations

Wzmocnienie – mnożenie sygnału przez wartość (liczba, zmienna, wyrażenie)



Parametry:
- Gain

Commonly UB

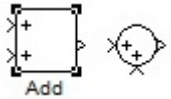
Parametry (wybrane):
- Gain

Suma – dodawanie zadanej ilości sygnałów lub dodawanie/odejmowanie sygnałów zgodnie z podanym wektorem działań

Commonly UB

Parametry:
- Inputs ports signs/gain, np.: [1;-1]
  1=dodawanie, -1=odejmowanie. Jeśli jest odejmowanie, to na bloku pojawią się znaki.

Patrz też bok SUMMATION – określa się typ danych, przekroczenie zakresu (nie pokazuje znaków).
Blok SUM_f (okrągły) – stała ilość wejść.



Commonly UB

Parametry (wybrane):
- Icon shape (rectangular, round)
- List of signs (np.: +-)

Jeśli wszystkie sygnały mają być dodawane to można podać ilość wejść, przy różnych znakach podaje się listę znaków

Iloczyn – mnożenie zadanej ilości sygnałów lub mnożenie/dzielenie sygnałów zgodnie z podanym wektorem działań

Commonly UB

Parametry:
- Number of inputs or sign vector, np. [1;-1]
  1=mnożenie, -1=dzielenie. Jeśli jest dzielenie, to na bloku pojawią się znaki

Patrz też bok Summation – określa się typ danych, przekroczenie zakresu. Blok PROD_f (okrągły) – stała ilość wejść.

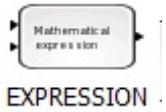
Commonly UB

Parametry (wybrane):
- Number of inputs (lub lista znaków, np.:*/)

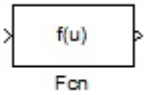
Jeśli wszystkie sygnały maja być mnożone to można podać ilość wejść lub listę znaków.

User-Defined Function	User-Defined Function

Wyrażenie matematyczne – na sygnałach z wektora wejściowego *u* i zmiennych



Parametry:
- number of inputs (max 8)
- scilab expression, np. (u1>0)*A*sin(u2)^2
- use zero-crossing



Parametry:
- expression, np. sin(u(1)*exp(2.3*(-u(2))))

Na wejście można podać pojedynczy sygnał lub wektor.
Jeśli symbol jest odpowiednio duży, to wyrażenie będzie widoczne na bloku

7.2.6 Ciągłe układy dynamiki – podstawowe funkcje

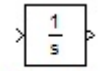
Continuous time system	Continuous

Blok całkujący – całkuje z sygnał wejściowy. Stan na wyjściu w chwili zero określa parametr wartość początkowa *wp*

Commonly US

Parametry:
- Initial state (*wp*)

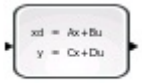
Patrz też blok INTEGRAL_m – rozbudowane całkowanie (m.in. z nasyceniem)



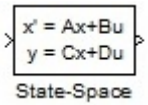
Commonly US

Parametry (wybrane):
- Initial condition (*wp*)
- Upper saturation limit (domyślnie = inf)
- Lower saturation limit (domyślnie= - inf)

Równania stanu zdefiniowane za pomocą macierzy A, B, C, D. Na wyjściu bloku dostępny jest tylko wektor sygnałów wyjściowych *y*. Można podać również wektor wartości początkowych *wp* zmiennych stanu *x*

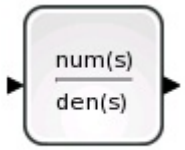


Parametry: A, B, C, D i *wp*
Wektor x(0) musi mieć odpowiedni wymiar nawet jeśli są to wartości zerowe

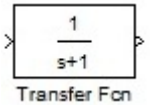


Parametry (wybrane):
- A, B, C, D i *wp*

Transmitancja – definiowana w postaci funkcji wymiernej (stopień wielomianu w liczniku musi być mniejszy niż stopień wielomianu w mianowniku).

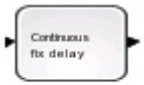


Parametry:
- Numerator (s), np.: 1
- Denominator (s), np.: 1+s



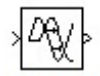
Parametry (wybrane):
- Numerator coefficients, np.: [1]
- Denominator coefficients, np.: [1 1]

Człon opóźniający – blok opóźniający sygnał wejściowy o stałą wartość (T_0). Dopóki nie pojawi się opóźniony sygnał wejściowy, to na wyjściu wystawiana jest wartość początkowa *wp* . W grupie podstawowych członów dynamiki to jedyna transmitancja, która nie jest funkcją wymierną.



Parametry:
- Delay (T_0)
- initial output (*wp*)
- Buffer size (domyślnie 1024)

Patrz też blok Variable_delay – opóźnienie zmienne w czasie

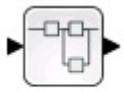


Parametr (wybrane):
- Time delay (T_0)
- Initial output (*wp*)
- Initial buffer size (domyślnie = 1024)

Patrz też blok Variable Transport Delay – opóźnienie zmienne w czasie

Port & Subsystem	Port & Subsystem

Podsystem – możliwość zgrupowania części schematu w jeden blok



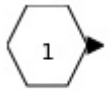
User-Defined Fun.

Ilość wejść i wyjść zależy od portów wejściowych i wyjściowych zawartych w bloku

Commonly UB

Ilość wejść i wyjść zależy od portów wejściowych i wyjściowych zawartych w bloku. Nazwy portów są widoczne na bloku.
Schemat można sparametryzować (operacja Mask Subsystem)

Port wejściowy



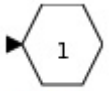
Sources
Commonly UB

Parametry: Port number
Uwaga: Program sygnalizuje błędy w numeracji portów – podwójne numery, brak ciągłości
Patrz też blok CLKINV_f – port wejściowy sygnału sterującego

Sources
Commonly UB

Parametry: Port number
Można zmieniać podpis pod blokiem (pojawi się automatycznie na bloku Subsytem)

Port wyjściowy



Sources
Commonly UB

Parametry: Port number
Uwaga: Program sygnalizuje błędy w numeracji portów – podwójne numery, brak ciągłości
Patrz też blok CLKOUTV_f – port wyjściowy sygnału sterującego

Sources
Commonly UB

Parametry: Port number
Można zmieniać podpis pod blokiem (pojawi się automatycznie na bloku Subsytem)

7.3 Konstrukcja schematów na bazie bloków całkujących

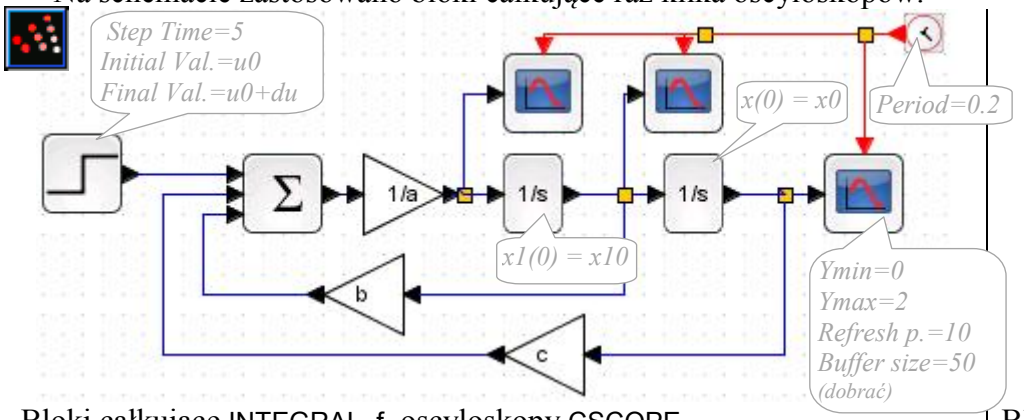
7.3.1 Zasady konstrukcji schematów

Zasada konstrukcji modeli graficznych zostanie przedstawiona na przykładzie liniowego równania różniczkowego $a\ddot{x}(t)+b\dot{x}(t)+cx(t)=u(t)$, że skokowym sygnałem wymuszającym i zadanymi warunkami początkowymi $\dot{x}(0)=x10$, $x(0)=x0$. Rysowanie schematu odbywa się po przetworzeniu równania do postaci: $\ddot{x}(t)=\big(u(t)-b\dot{x}(t)-cx(t)\big)/a$. Stąd wynika ilość bloków całkujących i zestaw sygnałów kierowanych na blok sumacyjny. Zmienne wykorzystane na schemacie można zainicjować wykonując podstawienie na głównej konsoli programu lub w skrypcie. Takie zmienne znajdują się w przestrzeniu roboczej programu, a tym samym są dostępne z poziomu schematu[1]. Wartości zmiennych zależą od typu badania:

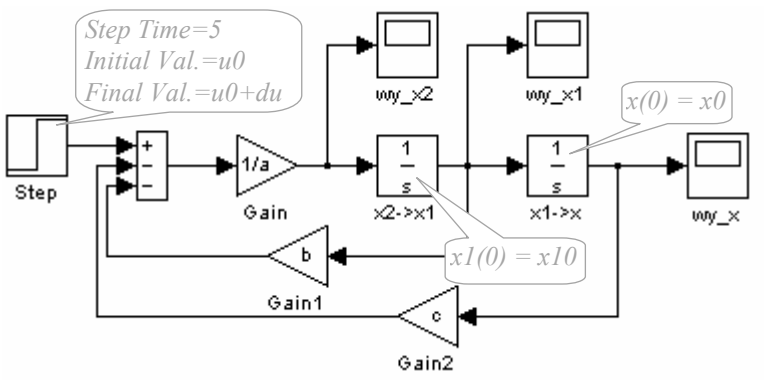
– odpowiedź skokowa (reakcja na skok wymuszenia od wartości 0 o wartość 1)	– reakcja na dowolne wymuszenie skokowe (tj. od dowolnego stanu początkowego *u0* o dowolną wartość *du*)
a = 2; b = 8; c = 2; u0=0; du=1; t0=5; - *skok jednostkowy, w chwili 5* x0=0; x10=0; - *warunki początkowe*	a = 2; b = 8; c = 2; u0=2; du=0.5; t0=5; - *skok wartości na wejściu i czas skoku* x0=u0/c; x10=0; - *warunki początkowe (stan równowagi)*

1. Schemat z blokiem całkującym i bieżącą prezentacją wykresów w oddzielnych oknach

Na schemacie zastosowano bloki całkujące raz kilka oscyloskopów.



Bloki całkujące INTEGRAL_f, oscyloskopy CSCOPE



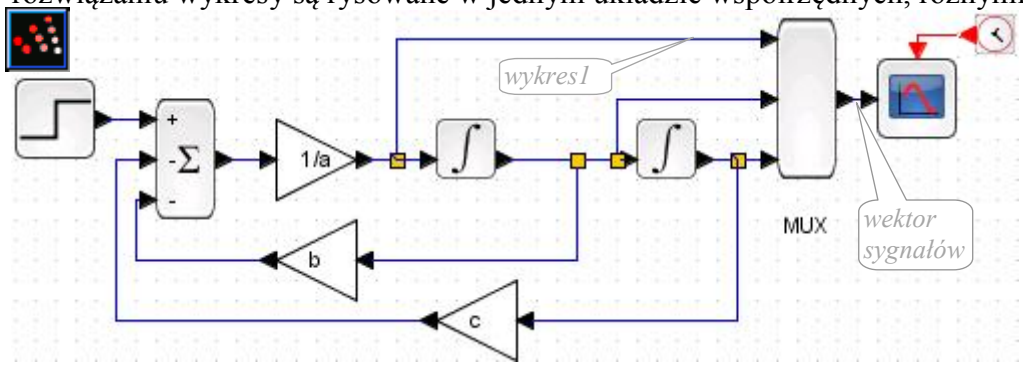
Bloki całkujące Integrator, oscyloskopy Scope

Okno bloku CSCOPE otwiera się automatycznie po uruchomieniu symulacji. Blok CSCOPE nie ma autoskalowania, więc przed symulacją należy ustawić odpowiednie zakresy skali. Aby cały przebieg symulacji był widoczny należy zdefiniować: Refresh period = czas symulacji (na schemacie założono 10s); Buffer size = czas symulacji / Period (z zegara).	Prezentacja danych w bloku Scope odbywa się tylko wtedy, gdy okno wykresów zostało otwarte przez operatora. Jeśli ilość wyświetlanych wartości przekroczy pojemność bufora bloku, to zawartość bufora zostanie przesunięta (widoczne będą ostatnie wartości). Rozmiar bufora można ustawić w oknie właściwości bloku (na zakładce Data history należy wyłączyć ograniczenie lub ustawić większą wartość).
Warto wykorzystać możliwość definiowania treści na pasku tytułowym okna wykresów (jako identyfikacja wykresów) – jest to jeden z pozycji w oknie edycji parametrów bloku: Name of Scope (label&Id)	Na pasku tytułowym okna wykresów pojawia się etykieta bloku Scope (podpis pod blokiem). Domyślną etykietę można zmienić (np. na opis zmiennej) i wykorzystać do identyfikacji wyświetlanych wykresów

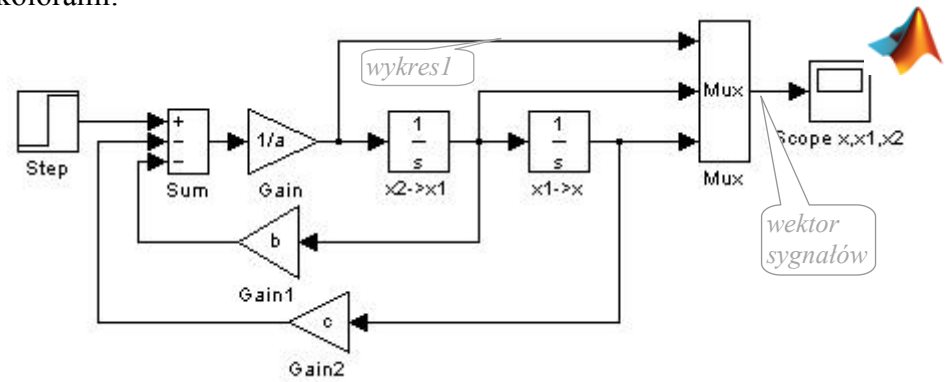
[1] W trakcie edycji schematu Scilab pozwala użyć tylko tych zmiennych, które zostały zdefiniowane wcześniej. Matlab sprawdza istnienie zmiennych dopiero przed symulacją

2° Schemat z blokiem całkującym i bieżącą prezentacją wykresów we wspólnym oknie

Blok oscyloskopu ma domyślnie jedno wejście ale można na nie skierować wektor sygnałów, tworzony za pomocą bloku multipleksera. W tym rozwiązaniu wykresy są rysowane w jednym układzie współrzędnych, różnymi kolorami.

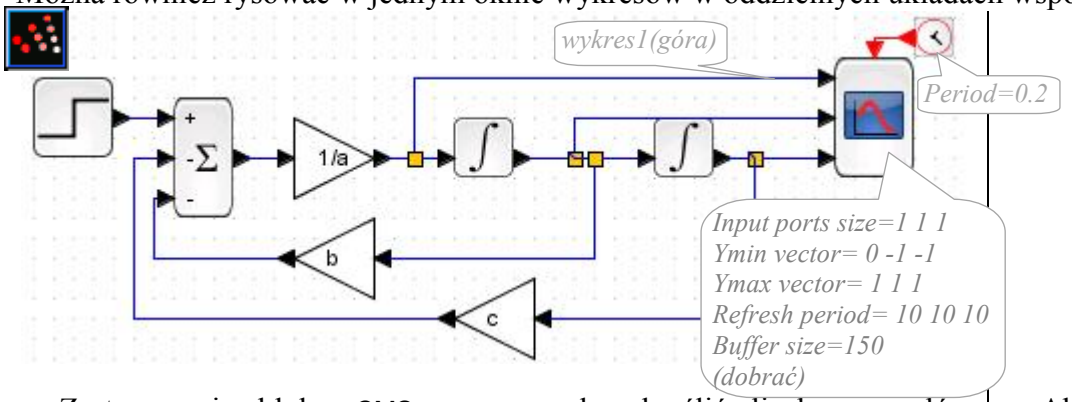


Blok multipleksera Mux. Kolory kolejnych wykresów: black, green, red, ... Jako całkujący zastosowano tym razem INTEGRAL_m, który zawiera w sobie między innymi funkcję nasycenia.

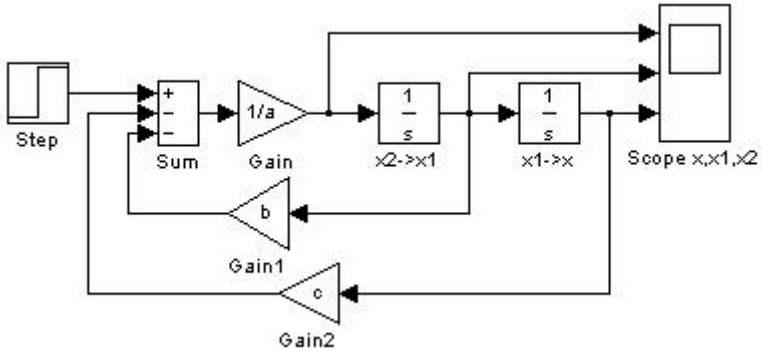


Blok multipleksera Mux. Kolory kolejnych wykresów: yellow, magenta, cyan, red, green, blue.

Można również rysować w jednym oknie wykresów w oddzielnych układach współrzędnych kolejno od góry (analogicznie jak w funkcji subplot)



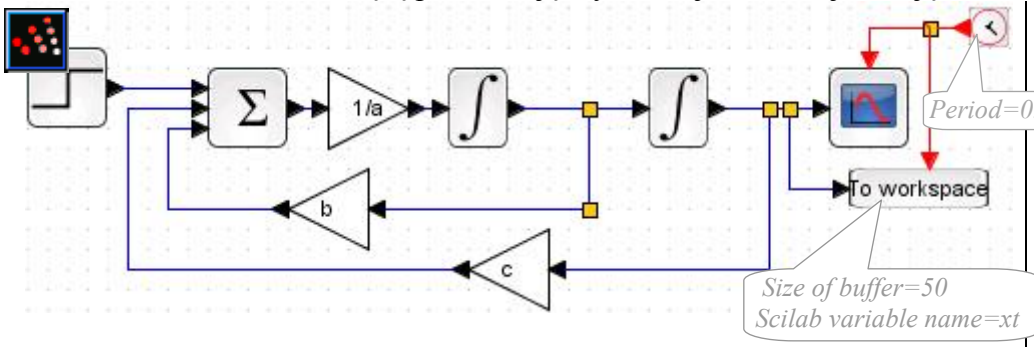
Zastosowanie bloku CMScope pozwala określić liczbę sygnałów wejściowych, które zostaną wyświetlone na oddzielnych wykresach. Parametry dla poszczególnych wykresów są podawane w postaci wektorów, na przykład wektor wartości minimalnych, wektora wartości maksymalnych. Rozmiar bufora danych dotyczy sumy wszystkich sygnałów: Refresh period = czas symulacji (na schemacie założono 10s); Buffer size = 3*czas symulacji / Period (z zegara).

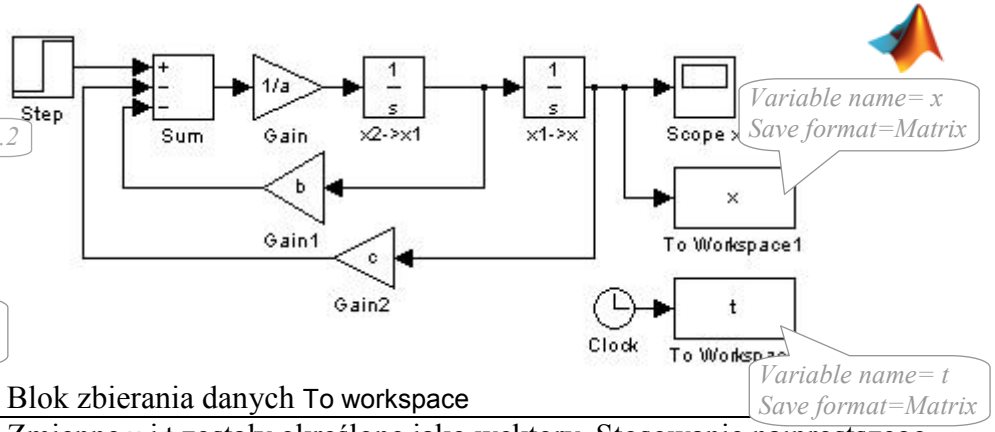


Aby zmienić ilość wejść bloku Scope należy w oknie właściwości bloku na zakładce „General” wpisać ilość osi (np. 3) – każde wejście będzie wyświetlane na oddzielnym wykresie.

3° Schemat z blokiem całkującym, bieżącą prezentacją wykresów i rejestracją danych

Przebieg symulacji można zarejestrować a następnie odtworzyć go za pomocą funkcji plot. Do rejestracji można wykorzystać blok zbierania danych (To workspace), w którym należy zdefiniować nazwę zmiennej (w Matlabie także format, np. struktura lub wektor). Na schematach poniżej zastosowano zarówno bieżącą prezentację wyników jak i ich rejestrację.



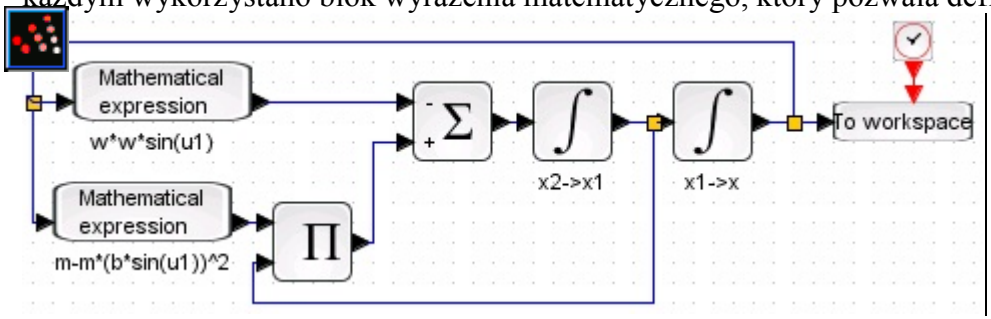


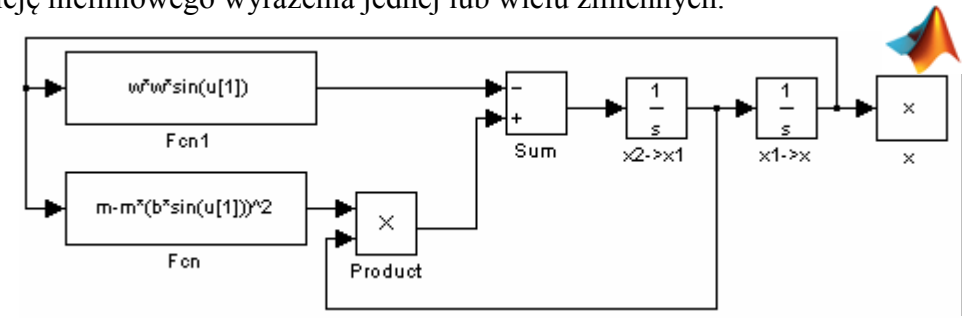
Blok zbierania danych To workspace	Blok zbierania danych To workspace
Zmienna xt jest strukturą. Format danych ma zawsze postać struktury zawierającej wektor wartości zmiennej i wektor czasu. Rozmiar bufora musi być nie mniejszy niż: Size of buffer = czas symulacji / Period (z zegara). Zastosowanie: plot(xt.time, xt.values)	Zmienne x i t zostały określone jako wektory. Stosowanie najprostszego formatu wektora danych (Matrix) wymaga aby zarejestrować również wektor czasu – na schemacie zastosowano w tym celu źródło Clock. Zastosowanie: plot(t, x)
	Powyższe rozwiązanie zawiera pewien nadmiar, ponieważ blok Scope także umożliwia rejestrację danych, jeśli w oknie właściwości bloku na zakładce Data history zostanie włączona opcja Save data to workspace, podana nazwa i format zmiennej.

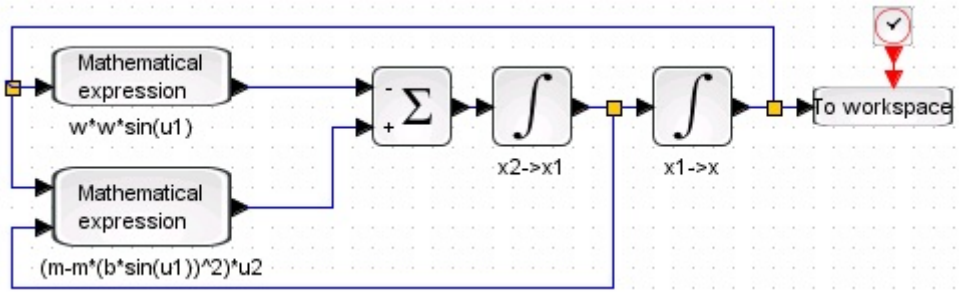
7.3.2 Schematy równań i układów równań różniczkowych nieliniowych

1. Przykład nieliniowego równania różniczkowego

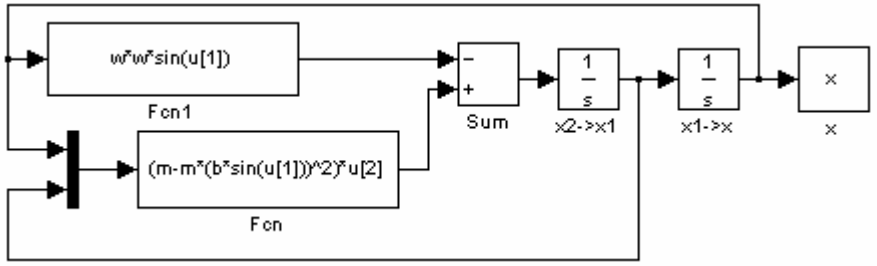
Równanie zadane w dowolnej postaci, na przykład $\ddot{x} - m(1 - b^2 \sin^2 x)\dot{x} + w^2 \sin x = 0$ zostaje przekształcone w typowy sposób, to znaczy aby po lewej stronie pozostała najwyższa pochodna zmiennej wyjściowej: $\ddot{x} = m(1 - b^2 \sin^2 x)\dot{x} - w^2 \sin x$. Przedstawiono po dwa warianty schematu – na każdym wykorzystano blok wyrażenia matematycznego, który pozwala definicję nieliniowego wyrażenia jednej lub wielu zmiennych.







Blok wyrażenia matematycznego Mathematical expression

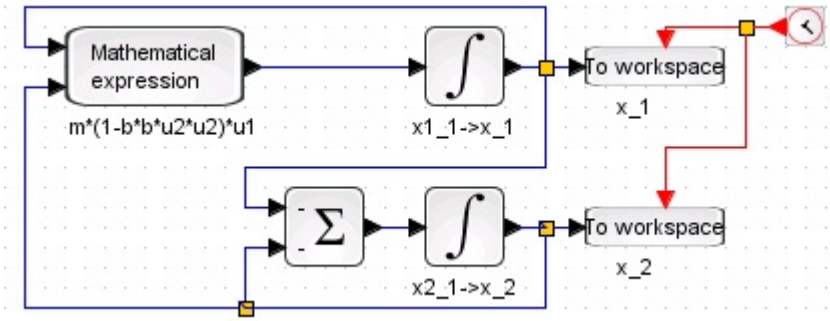


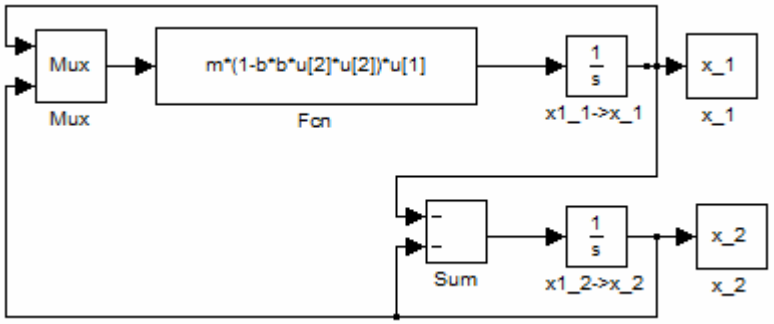
Blok wyrażenia matematycznego Fcn

2. Przykład nieliniowego układu równań różniczkowych

Zasady konstrukcji układów równań różniczkowych są analogiczne jak dla pojedynczych równań. Dla ilustracji przedstawiony zostanie przykład schematu do rozwiązania układów dwóch równań:

$$\begin{cases} \dot{x}_1 - m(1-b^2x_2^2)x_1 = 0 \\ \dot{x}_2 + x_1 + x_2 = 0 \end{cases} \Rightarrow \begin{cases} \dot{x}_1 = m(1-b^2x_2^2)x_1 \\ \dot{x}_2 = -x_1 - x_2 \end{cases}$$





7.4 Parametry i uruchomienie symulacji z okna edycji schematu

Menu w oknie edycji schematu zawiera pozycje i ikony pozwalające bezpośrednio zdefiniować parametry i uruchomić symulację, w tym:

Simulation – Symulacja – parametry i uruchamianie symulacji:
- Setup - Ustawienia – parametry symulacji
- Execution trace and Debug - Wykonaj śledź i analizuj
- Set Context - Ustaw kontekst
- Compile - Skompiluj
- Modelica initialize - Inicjalizacja Modelica
- Start – uruchomienie symulacji
- Stop – zatrzymanie symulacji

Simulation – parametry i uruchamianie symulacji:
- Start – uruchomienie symulacji
- Stop – zatrzymanie symulacji
- Configuration Parameters – parametry symulacji
- Normal
- Accelerator
- Rapid Accelerator

Przed uruchomieniem symulacji należy zainicjować zmienne, które są używane na schemacie (np. uruchomić skrypt zawierający definicje/obliczenia zmiennych) oraz określić parametry symulacji (przynajmniej czas trwania).

7.4.1 Parametry symulacji

Parametry symulacji określają typ oraz parametry algorytmów obliczeniowych (ang. solver). Możliwość dobrania parametrów symulacji pozwala realizować obliczenia szybko i dokładnie. Zazwyczaj (w typowych zadaniach) wartości domyślnie spełniają swoje zadanie i poza czasem obliczeń nie ma potrzeby wprowadzania zmian. Parametry symulacji są zapamiętywane pliku wraz ze schematem.

Wybrane parametry symulacji, które można odczytać/ustawić w menu Simulation:

Final integration time – Ostateczny czas integracji	**= 1.0E05**
Realtime scaling – Skalowanie czasu rzeczywistego	= 0.0E00
Integrator absolute tolerance - Integrator absolutnej toleracji	= 1.0E-04
Integrator relative tolerance - Integrator względnej toleracji	= 1.0E-06
Tolerance on time – Toleracja w czasie	= 1.0E-10
Max interation time interval – Maks.przedział czasu całkowania	= 1.00001E05
Solver 0 (CVODE) - 100 (IDA)	= 0 (CVODE)
Max step size (0 means no limit) – Maks. rozmiar kroku (0=brak limitu)	= 0
Set context - Ustaw kontekst – Definicja zmiennych kontekstowych	

Simulation time:
Start time = 0.0 **Stop time = 10.0**
Solver options:
Type = Variable-step Solver: ode45
Max step size = auto Relative tolerance = 1e-3
Min step size = auto Absolute tolerance= auto
Initial step size = auto
....

Uwagi:
- Domyślny czas symulacji (Final integration time) jest bardzo długi (10^5). Na początek lepiej zadać krótszy czas
- W parametrach symulacji nie można wpisywać zmiennych
- W symulacji wykorzystywany jest algorytm stałokrokowy (jedyny dostępny solver 0)

Uwagi:
- Domyślny czas symulacji (Stop time) jest dość krotki (10) i najprawdopodobniej trzeba go będzie wydłużyć.
- W parametrach symulacji można wpisać także zmienne
- Domyślnie stosowany jest zmiennokrokowy algorytm i to zapewnia zazwyczaj dobrą symulację. Ale jeśli czas symulacji jest długi, to należy określić maksymalny krok obliczeń (Max step size) – domyślnie jest on dobierany automatycznie jako 0.001 czasu symulacji (i zwiększa się przy wydłużaniu czasu)

7.4.2 Uruchomienie i zatrzymanie symulacji

Symulacja uruchamiana w oknie edytora nie wymaga zapamiętania schematu w pliku. Symulacja jest zwykle wykonywana w „pełnym biegu" ale może być też uruchamiana w trybie śledzenia.

Simulation\Start – Symulacja\Start
– uruchomienie symulacji w ustawionym trybie
Simulation\Execution trace and Debug - Symulacja\Wykonaj śledź i analizuj
– ustawienie trybu uruchomienia (poziom śledzenia)

Simulation\Start
– uruchomienie symulacji w pełnym biegu
Tools\Simulink Debugger
– uruchomienie symulacji w trybie debug

7.5 Parametry i uruchomienie symulacji w trybie wsadowym

Schemat zapisany w pliku można uruchomić w trybie wsadowym (batch mode) za pomocą funkcji wywołanej z głównej konsoli programu lub w skrypcie. Najprostsze wywołania funkcji są następujące:

scicos_simulate(scs_m);
gdzie: scs_m - struktura opisująca schemat

sim(*schemat*);
gdzie: *schemat* - nazwa pliku zawierającego schemat

Zmienna scs_m to zmienna systemowa, tworzona i aktualizowana po wczytaniu i uruchomieniu symulacji schematu w edytorze xcos.

Sekwencja poleceń przy uruchomienia symulacji bez otwierania edytora xcos jest następująca[1]:

```
loadXcosLibs();
```
- załadowanie bibliotek xcos (raz po uruchomieniu Scilaba)

```
importXcosDiagram('schemat.xcos');
```
- wczytanie schematu (tworzy/aktualizuje zmienną scs_m)
- opcjonalnie można otworzyć schemat xcos schemat1.xcos;

```
scicos_simulate(scs_m);
```
- uruchomienie symulacji (według zmiennej scs_m)

Plik ze schematem może znajdować się w kartotece ustawionej w File\SetPath (menu na głównej konsoli). Może to jednak powodować niejednoznaczności gdy w różnych kartotekach znajdują pliki o takiej samej nazwie (np. poprzednie wersje schematów). Bezpieczniej jest pracować na kartotece bieżącej.

Funkcje do uruchamiania symulacji pozwalają ustawić także parametry symulacji. Parametry, które zostaną w ten sposób podane przykrywają parametry zapamiętane wraz ze schematem. Podstawowe zastosowanie to możliwość ustalenia czasu symulacji (np. na 1000 jednostek):

```
scs_m.props.tf = 1000;      - struktura scs_m wczytana wcześniej
scicos_simulate(scs_m);
```

sim(*schemat, 1000*);

W przypadku Matlaba możliwe jest również odczytywanie wartości wynikowych, na przykład rejestrowanie wektora czasu (zamiast rejestrowania wartości z bloku Clock na schemacie – p.7.3.1):

```
[t] = sim(schemat, 1000);
```

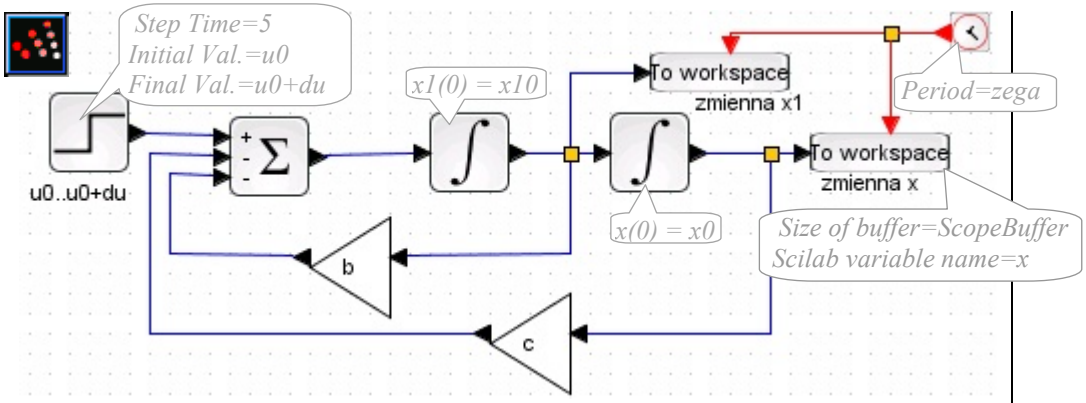
Możliwość uruchamiania symulacji w trybie wsadowym pozwala w prosty sposób zautomatyzować realizację badań.

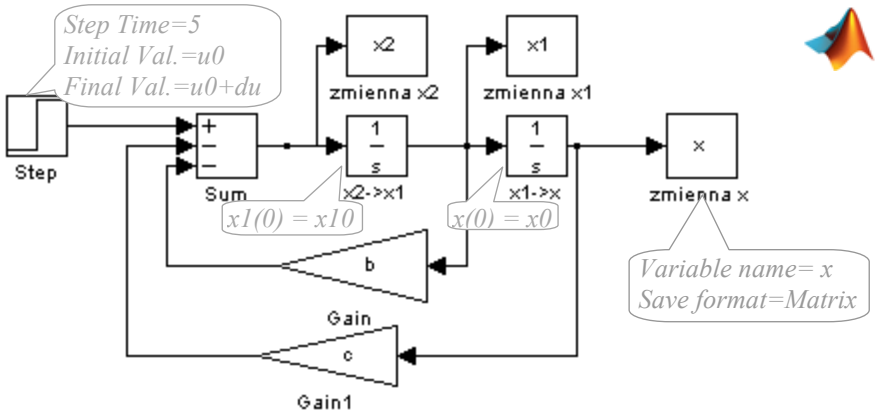
7.6 Przykład automatycznej realizacji programu badań

Zadanie polega na wygenerowaniu i porównaniu odpowiedzi układu $\ddot{x}(t)+b\dot{x}(t)+cx(t)=u(t)$ na wymuszenie skokowe *du* dla różnych wartości parametru *b*. Ponieważ należy wykonać kilka symulacji a ich wyniki dla porównania przedstawić na wspólnym wykresie, warto więc skorzystać z możliwości wsadowego uruchamiania symulacji modelu zapamiętanego w pliku graficznym.

Model układu jest zapisany w pliku graficznym o nazwie „wzór2”. Schemat jest sparametryzowany, to znaczy że parametry poszczególnych bloków są podawane w postaci zmiennych, które zostaną zainicjowane w skrypcie.

[1] Uwaga: Stan według wersji 5.3.3 (we wcześniejszych wersjach może być inaczej)





Cały program badań jest realizowany za pomocą skryptu, który obejmuje zainicjowanie zmiennych, wsadowe uruchomienie symulacji dla różnych wartości parametrów i wygenerowanie wykresów. Poniżej skrypty z zachowaniem oryginalnych kolorów i czcionek edytorów tekstowych.

```
tytul='Wplyw parametru b';
model = 'wzor2.xcos';              //pełna nazwa pliku ze schematem
loadXcosLibs();                    //wczytaj biblioteki
importXcosDiagram(model);          //wczytaj model (tworzy zmienną scs_m)
czas=50; zegar=.2;                 //czas symulacji i parametr: Clock/Period
ScopeBuffer = czas/zegar;          //parametr To workspace/Size of buffer
scs_m.props.tf=czas;               //czas trwania symulacji (zamiast czasu z pliku)

kolor = ['red','green','blue','cyan','magenta','yellow'];
c = 1;
tab_b = [2  4  8];                 //tablica parametru b
imax = size(tab_b,2);              //ilość parametrów (ograniczenie pętli)
u0=2; du=1;                        //parametry skoku (w bloku Step)
x10=0;                             //war.początkowy x1(0)
x0=u0/c;                           //war.początkowy x(0)

figure(1); set(gca(),"auto_clear","off"), xgrid(2), ylabel(strcat([tytul,' - x']))
figure(2); set(gca(),"auto_clear","off"), xgrid(2), ylabel(strcat([tytul,' - x1']))

for i=1:imax
   form = kolor(i);
   b = tab_b(i);
   scicos_simulate(scs_m);                 //czas symulacji: scs_m.props.tf
   figure(1);plot(x.time, x.values,form)   //dane z bloków "To workspace"
   figure(2);plot(x1.time, x1.values,form) //jw
end
```

```
tytul = 'Wpływ parametru b';
model = 'wzor2';                %nazwa pliku ze schematem

czas = 50;                      %czas trwania symulacji (patrz: sim)



kolor = 'rgbcmy';               %red,green,blue,cyan,magenta,yellows
c = 1;
tab_b = [1  2  3];              %tablica parametru b
imax = size(tab_b, 2);          %ilość parametrów (ograniczenie pętli)
u0=0; du=1;                     %parametry skoku (w bloku Step)
x10=0;                          %war.początkowy x1(0)
x0 =u0/c;                       %war.początkowy x(0)

fig1=figure, hold on, grid on,  ylabel(strcat(tytul, ' - x'))
fig2=figure, hold on, grid on,  ylabel(strcat(tytul, ' - x1'))

for i=1:imax
    form = kolor(i);            % format linii
    b = tab_b(i);               % kolejna wartość parametru b
   [t] = sim(model, czas);      % „[t]=” zamiast bloku „Clock”
   figure(fig1), plot(t, x, form);  % dane z bloków „To Workspace”
    figure(fig2), plot(t, x1, form); % jw
end
```

7.7 Schematy układów liniowych

7.7.1 Modele układów liniowych (jedno- i wielowymiarowe)

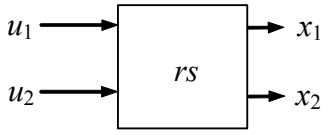
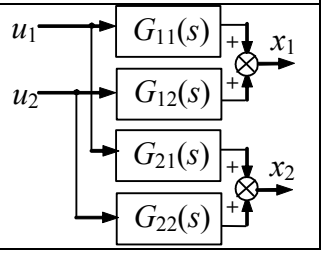
Metoda konstrukcji schematu modelu na bazie bloków całkujących przedstawiona powyżej (p.7.3), jest dość ogólna – umożliwia badanie zarówno równań różniczkowych liniowych jak i nieliniowych. W przypadku równań liniowych (zlinearyzowanych) istnieją alternatywne sposoby definiowania modeli oparte na macierzowym zapisie równań stanu oraz na transmitancjach [1][7].

Załóżmy, że przedmiotem badań jest równanie $\ddot{x}(t)+b\dot{x}(t)+cx(t)=u(t)$, czyli model, który był już zdefiniowany wcześniej za pomocą bloków całkujących (7.6).

Model i konwersja na:	równania stanu	transmitancję
$\ddot{x}(t)+b\dot{x}(t)+cx(t)=u(t)$ war.począt.: $\dot{x}(0)=0,\ \ddot{x}(0)=0$ tzn. stan równowagi: $x(0)=u(0)/c$	$x=x_1 \quad \leftarrow x$ $\dot{x}_1=x_2 \quad \leftarrow \dot{x}$ $\dot{x}_2 \quad \leftarrow \ddot{x}$ $\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix}=\begin{bmatrix}0 & 1\\ -c & -b\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix}+\begin{bmatrix}0\\ 1\end{bmatrix}u(t)$ $\dot{\mathbf{X}}(t)= \quad \mathbf{A} \quad \mathbf{X}(t)+\ \mathbf{B}\ \ \mathbf{U}(t)$	Zakładając $\dot{x}(0)=0$ i $\ddot{x}(0)=0$ po zastosowaniu transformaty Laplace'a mamy: $s^2x(s)+sbx(s)+cx(s)=u(s)$ $x(s)=\frac{1}{s^2+bs+c}u(s)$
Parametry bloków:	$\mathbf{A}=\begin{bmatrix}0 & 1\\ -c & -b\end{bmatrix},\ \mathbf{B}=\begin{bmatrix}0\\ 1\end{bmatrix},\ \mathbf{C}=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix},\ \mathbf{D}=\begin{bmatrix}0\\ 0\end{bmatrix}$ stan równowagi $\mathbf{X}(0)=-\mathbf{A}^{-1}\mathbf{B}\mathbf{U}(0)$	licznik = 1 → wektor współczynników = [1] mianownik =s^2+bs+c → wektor współczynników = [1 b c] zawsze zerowe warunki początkowe $u(0)=0,\ x(0)=0$

Jeden ze sposób realizacji zadania polega na utworzeniu schematu z wykorzystaniem bloków dedykowanych do równań stanu i transmitancji.

Analizowany model ma jedno wejście i jedno wyjście (model SISO), co jest typowe w przypadku transmitancji. W równaniach stanu tego modelu występuje jedna zmienna wejściowa i dwuelementowy wektor zmiennych wyjściowych, zawierający właściwą zmienną wyjściową i jej pochodną. Równania stanu są typową formą do badania obiektów o wielu wejściach i wielu wyjściach (MIMO), natomiast zastosowanie transmitancji w takich przypadkach wymaga zdefiniowania kilku wyrażeń, na przykład dla układu o dwóch wejściach i dwóch wyjściach:

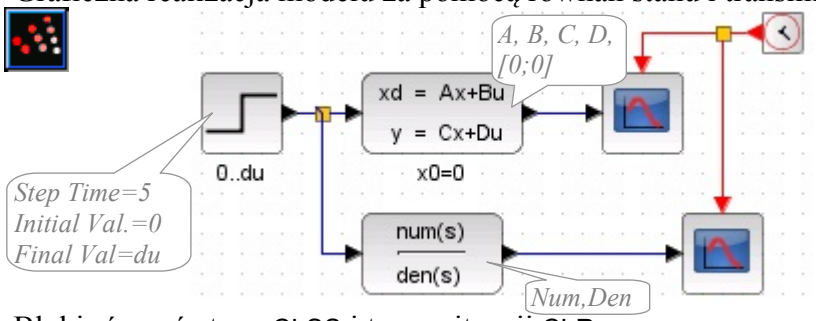
równania stanu	transmitancje[1]
$\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix}=\begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix}+\begin{bmatrix}b_{11} & b_{12}\\ b_{21} & b_{22}\end{bmatrix}\begin{bmatrix}u_1(t)\\ u_2(t)\end{bmatrix}$ 	$x_1(s)=G_{11}(s)u_1(s)+G_{12}(s)u_2(s)$ $x_2(s)=G_{21}(s)u_1(s)+G_{22}(s)u_2(s)$ 

[1] Uwaga: Często nie ma potrzeby stosowania schematu blokowego transmitancji ponieważ badania ograniczają się wyznaczania reakcji na zmiany poszczególnych wejść.

7.7.2 Symulacja od zerowych warunków początkowych

W pierwszym podejściu przygotowane zostaną schematy i skrypty do wyznaczenia odpowiedzi skokowej układu $\dddot{x}(t)+b\ddot{x}(t)+cx(t)=u(t)$, czyli reakcji układu na skok od wartości 0 do wartości *du*=1 (np. w chwili 5), przy zerowych warunkach początkowych (stan równowagi dla $u(0)=0$).

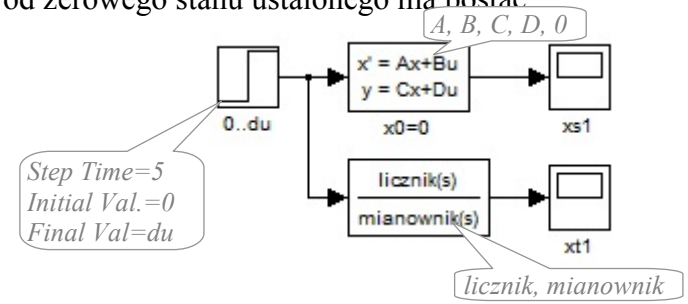
Graficzna realizacja modelu za pomocą równań stanu i transmitancji do symulacji od zerowego stanu ustalonego ma postać:



Bloki równań stanu CLSS i transmitancji CLR

Uwaga: Należy podać warunki początkowe dla wszystkich zmiennych stanu

Format dość dowolny: [0;0] lub [0,0] lub 0,0.



Bloki równań stanu State-Space i transmitancji Transfer Fcn

Parametry bloków zostały zdefiniowane za pomocą zmiennych wygenerowanych za pomocą skryptu:

```
b = 8; c = 2;              //parametry układu
du=1;                      //wartość skoku na wejściu

//definicja macierzy
A=[0, 1; -c, -b];  B=[0; 1];  C=[1, 0; 0, 1];  D=[0; 0];

//definicja transmitancji
s = poly(0,'s');           //def.zmiennej wielomianu
Num = 1;                   //wielomian licznika
Den = s^2+b*s+c;           //wielomian mianownika
```

```
b = 8; c = 2;
du=1;          %wartość skoku na wejściu

%definicja macierzy
A=[0, 1; -c, -b]; B=[0; 1]; C=[1, 0; 0, 1]; D=[0; 0];

%definicja transmitancji
licznik = 1;               %współczynniki wielomianu licznika
mianownik= [1 b c];        %współczynniki wielomianu mianownika
```

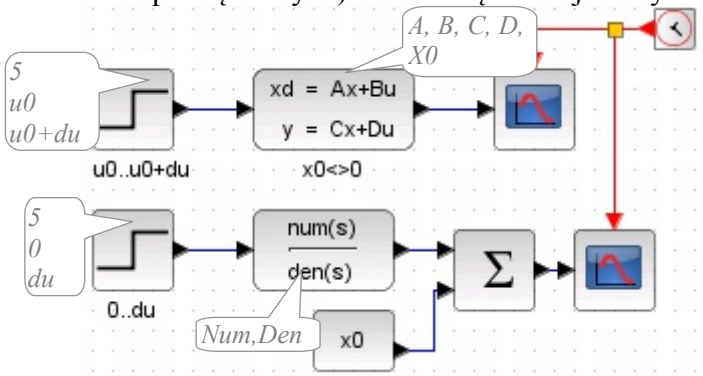
W blokach równań stanu wykorzystano zmienne macierzowe definiujące równania stanu $\dot{\mathbf{x}}=\mathbf{Ax}+\mathbf{Bu}$ oraz równania wyjściowe $\mathbf{y}=\mathbf{Cx}+\mathbf{Du}$, ponieważ równania wyjściowe są jedynym sposobem przekazania wyników symulacji na zewnątrz tego bloku. W bloku równań stanu podawany jest również zestaw warunków początkowych w postaci wektora wartości zmiennych stanu w chwili 0 – w tym wypadku są to warunki zerowe.

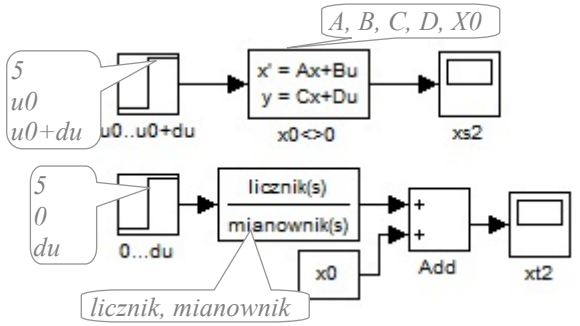
Ten sam model definiowany w bloku transmitancji wykorzystuje zmienne zawierające wielomiany licznika i mianownika. Uzyskane wyniki symulacji (wyświetlane na wykresach) nie zależą od sposobu zdefiniowania modelu, z tym, że na wyjściu bloku równań stanu dostępny wektor sygnałów, który odpowiada zmiennej *x* i jej pochodnej.

7.7.3 Symulacja od dowolnego stanu równowagi

Kolejne podejście do badania układów liniowych jest bardziej uniwersalne ponieważ przedstawia sposób badania reakcji układu $\ddot{x}(t)+b\dot{x}(t)+cx(t)=u(t)$ na dowolny skok od dowolnego stanu równowagi, to znaczy dowolna zmiana (*du*) od dowolnego stanu równowagi zależnego od początkowej wartości wymuszenia (*u0*).

Graficzna realizacja modelu do symulacji od dowolnego stanu ustalonego w przypadku równań stanu nie zmieni się, ponieważ na wejście bloku zawierającego równania stanu można podać zakłócenie skokowe od dowolnej wartości początkowej, a w parametrach bloku ustawić wektor odpowiednich wartości początkowych, obliczony na przykład w skrypcie z wykorzystaniem operacji macierzowych (zmienna X0). Uzyskanie tego samego wyniku w bloku transmitancji można zrealizować przez złożenie dwóch sygnałów: reakcji transmitancji na wymuszenie skokowe (*du*) od wartości 0 oraz wartości początkowej odpowiadającej stanowi równowagi (zmienna *x0*). W badaniach dynamiki modelu opisanego transmitancją zazwyczaj nie stosuje się tego przesunięcia (wiadomo, że własności układu liniowego nie zależą od punktu pracy więc można badać układ od zerowych warunków początkowych). Przesunięcie x0 jest wykorzystywane w symulacjach do przedstawienia wartości w odpowiedniej skali fizycznej.





Parametry powyższych bloków zostały zdefiniowane za pomocą zmiennych wygenerowanych za pomocą skryptu:

```
b = 8; c = 2;
u0=2; du=1;              //wartość początkowa i skok na wejściu

//definicja macierzy
A=[0, 1; -c, -b]; B=[0; 1]; C=[1, 0; 0, 1]; D=[0; 0];
U0=[u0];                 //dowolna wartość początkowa
X0=-inv(A)*B*U0;         //wektor war.początkowych dla U0

//definicja transmitancji
s=poly(0,'s');           //def.zmiennej wielomianu
Num=1;                   //wielomian licznika
Den=a*s^2+b*s+c;         //wielomian manownika
x0=u0/c;                 //stan początkowy dla u0
```

```
b = 8; c = 2;
u0=2; du=1;              %wartość początkowa i skok na wejściu

%definicja macierzy
A=[0, 1; -c, -b]; B=[0; 1]; C=[1, 0; 0, 1]; D=[0; 0];
U0=[u0];                 %dowolna wartość początkowa
X0=-inv(A)*B*U0;         %wektor war.początkowych dla U0

%definicja transmitancji
licznik = 1;             %współczynniki wielomianu licznika
mianownik= [1 b c];      %współczynniki wielomianu mianownika
x0=u0/c;                 %stan początkowy dla u0
```

Uwaga: Bloki transmitancji z warunkami początkowymi, które pojawiają się w nowszych wersjach programów, realizują to zadanie przez konwersję transmitancji do równań stanu.

7.8 Schematy złożone

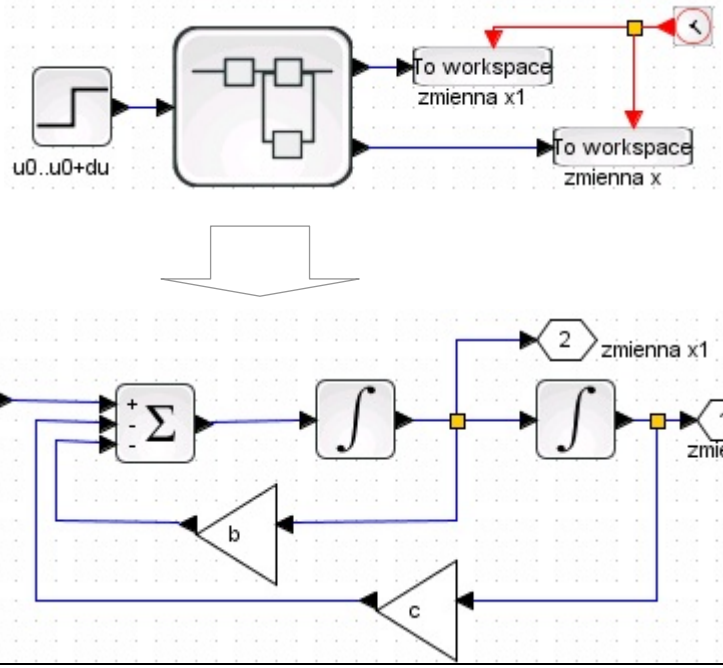
Rrozbudowane schematy modeli można uporządkować za pomocą operacji grupowania. Zgrupowany fragment schematu jest widoczny w postaci wydzielonego bloku („podsystemu"), który może łączyć się z zewnętrznymi elementami za pomocą portów wejściowych i wyjściowych. Wydzielony blok można następnie sparametryzować, można go też dołączyć do bibliotek Simulinka.

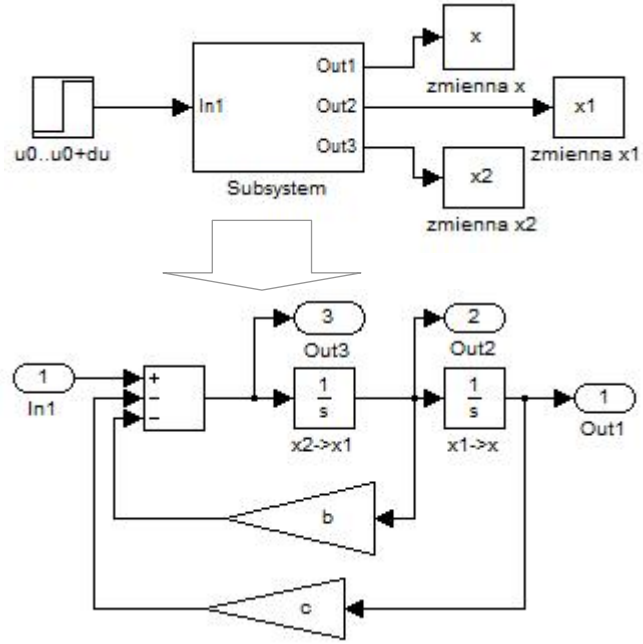
7.8.1 Grupowanie (blokowanie)

Jednym z możliwych sposobów wprowadzenia „podsystemu" jest zaznaczenie myszą obszaru, który ma zostać zgrupowany i wykonaniu operacji grupowania z menu okna schematu:


Edit/Region to Superblock – Edit/Obszar superbloku | Edit/Create Subsystem

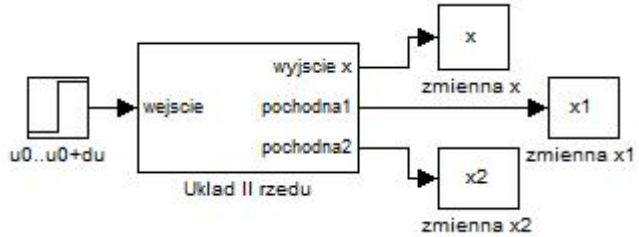

Tam gdzie zaznaczenie obszaru przecinało linie sygnałowe zostaną utworzone porty wejściowe (In) i wyjściowe (Out) bloku. Dla ilustracji grupowanie zastosowano na schemacie równania $\ddot{x}(t)+b\dot{x}(t)+cx(t)=u(t)$ (p.7.6) – poniższy rysunek przedstawia efekt po wykonaniu operacji grupowania oraz zawartość podsystemu

Porty wejściowe i wyjściowe nie maja własnych etykiet (podpisy widoczne przy portach wyjściowych powyżej są zwykłaymi tekstami dopisanymi do schematu)

Zmieniając etykiety portów wejściowych i wyjściowych uzyskuje się blok z podpisanymi wejściami i wyjściami



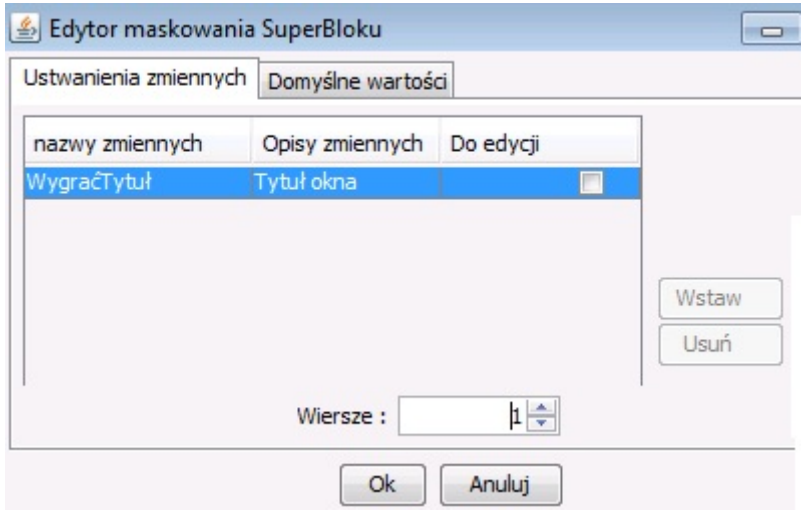
Zawartość bloku można edytować, w tym także zmieniać ilość i kolejność portów. Zgrupowanie elementów w bloku jest zabiegiem organizacyjnym i nie wpływa na symulacje ani na łączność ze zmiennymi z przestrzeni roboczej.

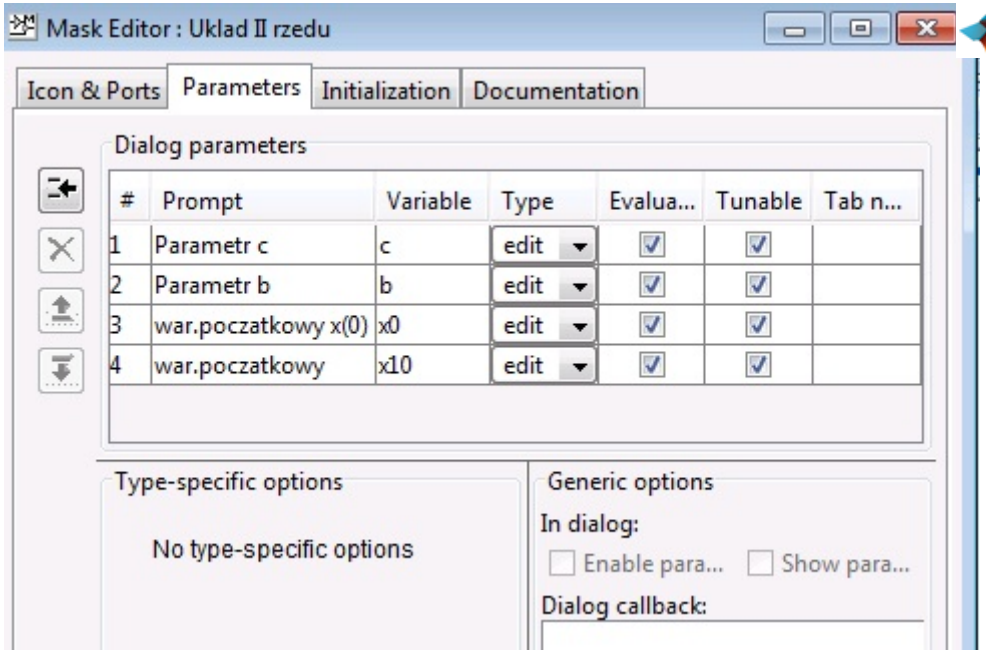
7.8.2 Parametryzowanie (maskowanie)

Blok z podsystemem można sparametryzować, co powoduje, że zmienne używane wewnątrz bloku stają się zmiennymi lokalnymi a określanie ich wartości następuje za pomocą listy parametrów (analogicznie jak dla wszystkich bloków z biblioteki bloków). Parametryzowanie bloku z podsystemem odbywa się przez jego zaznaczenie i wykonanie operacji maskowania, która powoduje otwarcie okna edycji parametrów bloku.

z menu bloku: Maska superbloku / Utwórz | z menu okna schematu: Edit/Mask Subsystem

W oknie edycji parametrów bloku zostaną zdefiniowane wszystkie zmienne używane wewnątrz bloku (parametry równania c, b oraz warunki początkowe x0, x10) – zmienne staną sie zmiennymi lokalnymi posystemu.





Zdefiniowana w ten sposób lista parametrów podsystemu rozwija się po typowym podwójnym kliknięciu na sparametryzowany blok. Jako wartości parametrów można podać (tak jak w każdym innym bloku) wartości lub zmienne z przestrzeni roboczej.

Sparametryzowany blok można poprawić w następujący sposób:

edycja maski z menu bloku: Maska superbloku / Dostosuj | edycja maski z menu okna schematu: Edit/Edit mask

edycja zawartości z menu okna schematu Edit/Look under mask

8. Badania układów dynamiki w trybie tekstowym

Definicja układów dynamiki w środowisku graficznym jest bardzo uniwersalną metodą prowadzenia badań. Jednak większość z nich można zrealizować znacznie szybciej – definiując w skrypcie, za pomocą funkcji, zarówno modele jak i zaplanowane badania, szczególnie gdy przedmiotem badań są układy liniowe. W Matlabie wymaga to jednak dodatkowego narzędzia Control System Toolbox.

8.1 Liniowe modele dynamiki – funkcje

8.1.1 Definicja modeli

Podstawowe funkcje do definiowania modeli opierają się na równaniach stanu i transmitancjach, które stają się parametrami funkcji definiujących:

syslin - *linear system definition* | ss, tf, zpk – *LTI model*

Wynikiem działania tych funkcji są obiekty typu „model", czyli zmienne do przechowywania modeli w przestrzeni roboczej. W zależności od podanych parametrów te same funkcje pozwalają zdefiniować modele ciągłe i dyskretne.

1. Definicja modelu w postaci równań stanu $\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu}$ zawsze wymaga uzupełnienia o równania wyjściowe $\mathbf{y} = \mathbf{Cx} + \mathbf{Du}$:

mSS = syslin('c', A, B, C [,D [,x0]]) - *model ciągły*	mSS = ss(A, B, C, D) - *model ciągły*
mSS = syslin('d', A, B, C [,D [,x0]]) - *model dyskretny*	mSS = ss(A, B, C, D, TS) - *model dyskretny [TS- czas próbkowania]*

gdzie: A, B, C, D - macierze równań stanu i równań wyjściowych, a x0 - wektor warunków początkowych.

Przykład układu drugiego rzędu : $\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}-1 & 1\\ 2 & -3\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}1 & 0\\ 2 & 1\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix}, \begin{bmatrix}y_1\\ y_2\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix}$

A=[-1,1; 2,-3];B=[1,0; 2, 1];C=[1,0; 0,1]; D=[0,0; 0,0];	A=[-1,1; 2,-3];B=[1,0; 2, 1];C=[1,0; 0,1]; D=[0,0; 0,0];
SS1=syslin('c', A, B, C); - *utwórz i zapamiętaj model o nazwie SS1*	SS1=ss(A, B, C, 0); - *utwórz i zapamiętaj model o nazwie SS1*
SS1=syslin('c', A, B, C, D); - *jw*	SS1=ss(A, B, C, D); - *jw*
[a,b,c,d] = abcd(SS1) - *podstaw macierze modelu* SS1 *do* a,b,c,d	[a,b,c,d] = ssdata(SS1) - *podstaw macierze modelu* SS1 *do* a,b,c,d
SS1('A'), SS1('X0') - *podaj macierz A i war.początk. modelu* SS1	

2. Definicja modelu w postaci transmitancji $FW(s) = \dfrac{WN(s)}{WD(s)}$ jest dedykowana przede wszystkim do funkcji wymiernych:[1]

s=poly(0,'s'); - *definicja zmiennej wielomianu*	mTF = tf(N, D [,TS]); - *model ciągły lub dysktretny (*TS*)*
mTF = syslin('c', WN, WD); - *model ciągły (lub* 'd' - *dysktretny)*	
s=poly(0,'s');	s = tf('s'); - *definicja zmiennej Laplace'a*
mTF = syslin('c', FW); - *jw*	mTF = FW; - *model ciągły (tylko SISO)*
	mZPK = zpk(Z, P, K [,TS]); - *model ciągły lub dysktretny (*TS*)*
gdzie: WN, WD – wielomian licznika i mianownika (funkcje s)	gdzie: N, D – wektory współczynników licznika i mianownika
FW – funkcja wymierna (funkcja s)	Z, P – wektory zer i biegunów; K – wzmocnienie
Tylko modele SISO.	FW – funkcja wymierna (funkcja s)
Modele MIMO definiowane tak jak macierze, np. [mTF1, mTF2]	Modele SISO i MIMO

[1] Uwaga: W programach symulacyjnych (także w Matlabie i Scilabie) występuje ograniczenie by stopień wielomianu licznika był < (lub <=) od stopnia wielomianu mianownika transmitancji, która ujawnia się albo już na etapie definicji, albo na etapie przetwarzaniu modeli.

Różne sposoby definiowania typowych transmitancji zostaną przedstawione na przykładzie: $G(s) = \frac{1}{4s^2+4s+1} = \frac{1}{(2s+1)^2} = \frac{1}{4(s+\frac{1}{2})^2}$

```
s = poly(0,'s');
G1 = syslin('c',1, 4*s^2+4*s+1)    - transmitancja G1

G2 = syslin('c',1, (1+2*s)^2)      - transmitancja G2
G3 = syslin('c', 1/((1+2*s)^2))    - transmitancja G3

H4 = 1/((1+2*s)^2)                 - funkcja wymierna (wyr.algebraiczne)
G4 = syslin('c',H4)                - transmitancja G4
```

```
G1 = tf (1, [4 4 1]);              - transmitancja G1

s = tf('s');
G2 = 1/(4*s^2+4*s+1)               - transmitancja G2
G3 = 1/((1+2*s)^2)                 - transmitancja G3

G4 = zpk([], [-0.5 -0.5], 1/4)     - transmitancja G4
```

Przykłady szczególnych przypadków transmitancji: $G_p(s) = 3$, $G_r(s) = s$, $G_o(s) = e^{-s5}$

```
Gp = 3                             - wzmocnienie jako prosta zmienna
~~Gp = syslin('c',3/1)~~           - niepoprawne
Gp = syslin('c',3/(0*s+1))         - wzmocnienie (istnieje zmienna s)
~~Gr = syslin('c',s)~~             - niepoprawne
Gr = syslin('c',s/(0*s+1))         - cz.różniczkujący (istnieje zmienna s)
```

```
Gp = 3                             - wzmocnienie jako prosta zmienna
Gp = tf(3)                         - wzmocnienie
Gp = 3/(0*s+1);                    - wzmocnienie (istnieje zmienna s)
Gp = s                             - cz.różniczkujący (istnieje zmienna s)
Gp = tf(s)                         - jw
Go = exp(-5*s);                    - opóźnienie (istnieje zmienna s)
```

Typowe transmitancje mają postać funkcji wymiernych, w których stopień licznika jest mniejszy (lub ≤) niż stopień mianownika. Transmitancje, które nie spełniają tego założenia mają ograniczony zakres przetwarzania, więc w razie konieczności dla tych szczególnych przypadków stosuje się przybliżenia:

$G_p(s) = k_p \rightarrow \frac{k_p}{T_r s + 1}$

człon proporcjonalny rzeczywisty

$G_r(s) = s \rightarrow \frac{s}{T_r s + 1}$

człon różniczkujący rzeczywisty

$G_o(s) = e^{-sT_o} \rightarrow \frac{1 - sT_0/2}{1 + sT_0/2}$

aproksymacja Padé

8.1.2 Operacje na modelach

Elementarne modele zdefiniowane na podstawie równań stanu i transmitancji są obiektami, które mogą być przedmiotem dalszego przetwarzania, a w szczególności konwersji typu modelu oraz konstrukcji schematów blokowych.

1. Konwersja modelu umożliwia przekształcenie równań stanu na transmitancje i odwrotnie.

```
ss2tf, tf2ss                       - funkcje konwersji
```

```
ss, tf, zpk                        - funkcje definicji zastosowane do konwersji
ss2tf, ss2zpk, tf2ss, tf2zpk, zpk2ss, zpk2tf   - funkcje konwersji
```

Załóżmy że model mSS to równania stanu, model mTF to transmitancja oparta na wielomianach, wówczas:

```
mTF = ss2tf(mSS)
mSS = tf2ss(mTF)
[off, WN, WD] = ss2tf(mSS)
```

gdzie: WN, WD – wielomian licznika i mianownika (funkcje s)

```
mTF = tf(mSS)
mSS = ss(mTF)
[N,D] = ss2tf(A, B, C, D, nr_wejscia)
[A,B,C,D] = TF2SS(N,D)
```

gdzie: N, D – wektory współczynników licznika i mianownika

Możliwości konwersji w Matlabie są szersze ze względu transmitancje zdefiniowane przez zera i bieguny (model mZPK). Funkcje zpk oraz ss2zpk i tf2zpk można zastosować nie tylko do zmiany typu modelu ale pośrednio do wyznaczenia zer i biegunów liniowych modeli

2. Konstrukcja schematów blokowych na bazie obiektów zdefiniowanych wcześniej opiera się na kilku podstawowych operacjach i funkcjach, które realizują połączenie równoległe (+), szeregowe (*), ze sprzężeniem zwrotnym (/.).

S1 + S2	S1 + S2;	parallel(S1,S2 [,wej,wyj])
S1 * S2	S1 * S2;	series(S1,S2 [,wej,wyj])
Sl1 /. Sl2	--------	feedback(S1,S2 [,wej,wyj])
--------	--------	connect(S1,S2,wej,wyj)
--------	S1 \ S2 oraz S1 / S2	inv(S1) * S2 oraz S1 * inv(S2)

Można łączyć modele różnych typów, ważny może być jedynie rozmiar modelu, czyli ilość wejść i wyjść. Rozmiar każdego modelu prostego czy złożonego można sprawdzić za pomocą funkcji size(nazwa_modelu).

8.2 Podstawowe badania obiektów liniowych

Badania dynamicznych własności obiektów wchodzą obszar narzędzi CACSD[1] i w najprostszym przypadku polegają na zarejestrowaniu odpowiedzi skokowej lub impulsowej oraz wyznaczeniu charakterystyk częstotliwościowych obiektu [1].

8.2.1 Charakterystyki czasowe

Odpowiedź skokowa jest reakcją obiektu na skok jednostkowy (od 0 do 1) podawany w stanie równowagi w zerowych warunkach początkowych. Analogicznie odpowiedź impulsowa jest reakcją na impuls Diraca w takich samych warunkach. Takie eksperymenty realizują następujące funkcje:

csim - *odpowiedzi czasowe układów liniowych*	step - *odpowiedz skokowa układu liniowego* impulse - *odpowiedz impulsowa układu liniowego*
Tylko dla modeli SISO. Funkcja zwraca wektor wartości.	Modele SISO i MIMO. Funkcje tworzą wykres lub zwracają wartości.

W przykładzie poniżej zdefiniowano prosty człon inercyjny i wyznaczono jego odpowiedzi czasowe - skokową i impulsową.

```
s=poly(0,'s'); S1=syslin('c', 1/(2*s+1) ); t=0:0.05:5;
subplot(2, 1, 1)
plot(t,csim('step', t, S1));          //wyznacz i rysuj odp.skokową
subplot(2, 1, 2)
odpi = csim('impulse', t, S1);        // wyznacz odp.impulsową
plot(t, odpi);                        // rysuj odp.impulsową
```

```
s = tf('s'); S1 = 1/(2*s+1);
subplot(2, 1, 1)
step(S1,5)                    % wyznacz i rysuj odp.skokową
subplot(2, 1, 2)
[odpi, t] = impulse(S1,5);    % wyznacz odp.impulsową
plot(t, odpi);                % rysuj odp.impulsową
```

W kolejnym przykładzie wyznaczono odpowiedzi dwuwymiarowego modelu opartego na równaniach stanu

```
A=[-1,1; 2,-3];B=[1,0; 2, 1];C=[1,0; 0,1]; D=[0,0; 0,0];
S2=syslin('c', A, B, C);  S2t=ss2tf(S2);
t=0:0.05:5;
```

- Ze względu na ograniczenie do modeli SISO należałoby robić badania poszczególnych transmitancji z modelu S2t

```
A=[-1,1; 2,-3];B=[1,0; 2, 1];C=[1,0; 0,1]; D=[0,0; 0,0];
S2=ss(A, B, C, D);
subplot(2, 1, 1)
step(S2,5)                          % wyznacz i rysuj odp.skokową
subplot(2, 1, 2); hold on; ylabel('x1'); xlabel('u1 i u2');
[odpi, t] = impulse(S2,5);          % wyznacz odp.impulsową
plot(t, odpi(:,1,1)); plot(t, odpi(:,1,2));     % rysuj x1(u1) i x1(u2)
```

[1] Computer Aided Control System Design

8.2.2 Charakterystyki częstotliwościowe

W badaniach układów dynamiki duże znaczenie mają charakterystyki częstotliwościowe [1][7]. Punktem wyjścia dla nich jest opis modelu za pomocą transmitancji, w której zastosowano podstawienie $s=j\omega$, a następnie wykorzystano własności liczb zespolonych:

$$G(j\omega) = \frac{L(j\omega)}{M(j\omega)} = P(\omega) + jQ(\omega) = A(\omega)e^{j\varphi(\omega)}$$

Fizyczna interpretacja zmiennej ω[1] i różne postacie transmitancji są podstawą do konstrukcji kilku typów charakterystyk:

- charakterystyki części rzeczywistej $P(\omega)$ i części urojonej $Q(\omega)$ transmitancji,
- charakterystyka amplitudowo-fazowa $P(Q)$ – wykres Nyquista,
- charakterystyka amplitudowa $A(\omega)$ i charakterystyka fazowa $\varphi(\omega)$,
- logarytmiczna charakterystyka modułu $M(\omega) = 20\lg A(\omega)$ i fazy $\varphi(\omega)$ - wykresy Bodego,
- logarytmiczna charakterystyka amplitudowo-fazowa $M(\varphi)$.

Najczęściej stosowane charakterystyki mają wsparcie w następujących funkcjach:

bode, gainplot	- *ch. Bodego (oś częstotliwości w Hz)*	bode, bodemag	- *ch. Bodego (oś częstotliwości w rad/sek)*
nyquist	- *ch.Nyquista*	nyquist	- *ch.Nyquista*
black	- *ch.Nicholsa (Black-Nichols)*	nichols	- *ch.Nicholsa*
nicholschart	- *siatka do ch.Nicholsa*		
g_margin, p_margin	- *charakterystyczne parametry ch.częstotliw.*	allmargin, margin	- *charakterystyczne parametry ch.częstotliw.*
repfreq	- *odpowiedź częstotliwościowa*	freqresp	- *odpowiedź częstotliwościowa*
Tylko dla modeli SISO i SIMO. Funkcja zwraca wektor wartości.		Modele SISO i MIMO. Funkcje tworzą wykres lub zwracają wartości.	

8.2.3 Projektowanie układów sterowania

Narzędzia typu CACSD obejmują wiele specjalistycznych funkcji przeznaczonych do projektowania układów sterowania. Wśród nich znajdują się narzędzia związane z operacjami na macierzach i sterowaniem liniowo-kwadratowy-Gaussa (LQG – ang. *Linear-quadratic-Gaussian*), które jest fundamentalnym zagadnieniem sterowania optymalnego. Regulator LQG jest kombinacją filtru Kalmana z regulatorem liniowo-kwadratowym (LQ). Problem LQ dotyczy sterowania układem dynamicznym opisanych przez układ liniowych równań różniczkowych, które minimalizuje koszt opisany funkcjonałem kwadratowym. Rozwiązanie tego problemu można z łatwością obliczyć za pomocą komputera dysponując na przykład funkcjami:

ricc	- *równanie Riccatiego*	care	- *równanie Riccatiego*
linmeq	- *równania Sylvester and Lyapunov*	lyap	- *równanie Lyapunova*
lqe	- *linear quadratic estimator (Kalman Filter)*	kalman	- *estimator Kalmana*
lqr	- *kompensator LQ*	lqr	- *regulator LQ*
lqg	- *kompensator LQG*	lqg	- *synteza sterowania LQG*

8.2.4 Definicja i własności podstawowych członów dynamiki

Dla ilustracji badań układów liniowych w poniższych skryptach zdefiniowano kilka członów dynamiki i wyznaczono ich charakterystyki czasowe i częstotliwościowe. Przy okazji w komentarzach zaznaczono ograniczenia w definicji lub w badaniach szczególnych przypadków transmitancji, gdy stopień licznika jest większy niż stopień mianownika.

[1] ω to pulsacja [rad/sek]; $\omega=2\pi f$, gdzie f to częstotliwość [Hz]

```
T1=2; T2=20; K1=3;       //parametry czlonów
Ti=2; Td=2; krz = 0.1;
exec('CzlonyFunWykresy.sce')
s=poly(0,'s');
t=0:0.05:5;                  //wektor czasu (dla odp.czasowych)
//inercyjny 1.rzedu, 2.rzedu
CzlonI1 = syslin('c', 1/(T1*s+1));
CzlonI2 = syslin('c', 1/((T1*s+1)*(T2*s+1)));
CzlonyFunWykresy(t, CzlonI1, 'inercja',1,1)
CzlonyFunWykresy(t, CzlonI2, 'inercja 2',1,1)

//proporcjonalny
//CzlonP = syslin('c', K1);                  //błąd w definicji
CzlonP = syslin('c', K1/(0*s+1));
CzlonyFunWykresy(t, CzlonP, 'prop',0,1)    //step-nie,bode-ok

//calkujacy
CzlonI = syslin('c', 1/(Ti*s));
CzlonyFunWykresy(t, CzlonI, 'calka',1,1)

//różniczkujacy (idealny)
//CzlonD = syslin('c', Td*s);                  //blad w definicji
CzlonD = syslin('c', Td*s/(0*s+1));
CzlonyFunWykresy(t, CzlonD, 'rózn',0,1)    // step-nie,bode-ok

//różniczkujacy (rzeczywisty)
CzlonDr = syslin('c', Td*s/(krz*T1*s+1));
CzlonyFunWykresy(t, CzlonDr, 'rózn',1,1)
```

```
T1=2; T2=20; K1=3;       %parametry członów
Ti=2; Td=2; krz =0.1;

s=tf('s');

%inercyjny 1.rzędu, 2.rzędu
CzlonI1 = 1/(T1*s+1);              %=tf([1],[T1,1])
CzlonI2 = 1/((T1*s+1)*(T2*s+1));
CzlonyFunWykresy(CzlonI1, 'inercja',1,1)
CzlonyFunWykresy(CzlonI2, 'inercja 2',1,1)

%proporcjonalny
CzlonP = tf(K1);             %wzmocnienie/macierz wzmocnień
CzlonyFunWykresy(CzlonP, 'prop',1,1)


%calkujący
CzlonI = 1/(Ti*s);
CzlonyFunWykresy(CzlonI, 'calka',1,1)

%różniczkujący (idealny)
CzlonD = Td*s;                   %błąd definicji
CzlonyFunWykresy(CzlonD, 'rozn.ideal',0,1)  %step-nie,bode-ok


%różniczkujący (rzeczywisty)
CzlonDr = Td*s/(krz*T1*s+1);
CzlonyFunWykresy(CzlonDr, 'rozn.rzecz',1,1)
```

```
function CzlonyFunWykresy(t, czlon, opis, fs, fb)
  h1 = figure();
  if fs>0
     subplot(2,1,1); plot(t,csim('step', t, czlon)); xgrid(1)
  end
  if fb>0
    subplot(2,1,2); bode(czlon, 0.01,10, opis);     //bode(czlon);(Hz)
  end
  title(opis)
endfunction
```

```
function CzlonyFunWykresy(czlon, opis, fs, fb)
   fig1 = figure;
   if fs>0
     subplot(2,1,1); step(czlon); grid on;
   end
   if fb>0;
     subplot(2,1,2); bode(czlon);
   end
   title(opis)
```

9. Interfejs graficzny użytkownika (GUI)

9.1 Obiektowy system graficzny

9.1.1 Identyfikacja i własności obiektów graficznych

Najprostszym sposobem rysowania i konfiguracji wykresów i okien graficznych jest są dedykowane funkcje, opisane w punkcie 4.2, np. title(), legend(), plot(). Dotyczą one jednak podstawowych własności wykresów i okien, i operują obiektach bieżących, to znaczy ostatnio utworzonych lub wskazanych za pomocą odpowiednich funkcji. Te i wszystkie pozostałe parametry tych obiektów można jednak odczytać, a niektóre zmienić, wykorzystując to, że system graficzny Matlaba i Scilaba ma charakter obiektowy, to znaczy, że:

– obiekty graficzne (nadrzędne okno graficzne, układ współrzędnych, okno menu, przyciski i suwaki etc.) są hierarchicznie uporządkowane w postaci drzewa - każdy obiekt ma jednego przodka i może mieć dowolną ilość potomków (dziedziczenie),
– każdy obiekt graficzny ma swoje właściwości (kolor, rozmiar, położenie, itp.) i swój identyfikator (wskaźnik, numer).

1. Identyfikatory i własności obiektów

Identyfikator umożliwia rozróżnianie obiektów. Na jego podstawie można odczytać strukturę, w której zawarte są aktualne wartości wszystkich własności obiektu:

– identyfikatorem obiektu jest wskaźnik na strukturę z danymi, np. *wskaźnik*
– odczytanie struktury z danymi: *wskaźnik*
– odczytanie własności: *wskaźnik.własność*

Uwaga: *wskaźnik* jest adresem na dane obiektu

– identyfikatorem jest numer, na podstawie którego można odczytać strukturę z danymi, np.*numer*
– odczytanie struktury z danymi: get(*numer*)
– odczytanie własności: *struktura* = get(*numer*); *struktura.własność*

Uwaga: *struktura* jest kopią danych obiektu

2. Odczytywanie wybranej własności obiektu

Na podstawie identyfikatora obiektu oraz nazwy własności można odczytać aktualną wartość tej własności - za pomocą uniwersalnej funkcji get lub stosując notację obiektową (z kropką):

get(*wskaźnik*, '*własność*')
wskaźnik.własność

get(*numer*, '*własność*')
struktura = get(*numer*); *struktura.własność*

3. Zmiana wybranej własności obiektu

Część własności obiektu można zmieniać za pomocą uniwersalnej funkcji set:

set(*wskaźnik, 'własność', wartość*)
set(*wskaźnik, 'własność', 'tekst'*)

wskaźnik.własność = wartość; - zmiana na obiekcie
wskaźnik.własność = '*tekst*'; - jw

set(*numer, 'własność', wartość*)
set(*numer, 'własność', 'tekst'*)
struktura = get(*numer*);
struktura.własność = wartość; - zmiana tylko na kopii danych
struktura.własność = '*tekst*'; - jw
x.własność1 = wartość; - nowa zmienna *x*
x.własność2 = wartość; - rozszerzenie zmiennej *x*
set(*numer, nowa*) - zmiana na obiekcie („wkopiowanie” *x*)

Uwaga: set(*numer, struktura*) - zabronione bo *struktura* zawiera także własności tylko do odczytu

4. Uzyskanie identyfikatora obiektu

Identyfikator obiektu można uzyskać (określić):

- bezpośrednio przez zapamiętanie jego wartości (wskaźnika lub numeru) w momencie tworzenia obiektu
- dochodząc do identyfikatora przez kolejnych potomków: *potomek* = get(*rodzic*, 'children')
- odczytując za pomocą funkcji identyfikator obiektu bieżącego lub domyślnego: gcf (get current figure), gca (get current axes), gdf (get default figure), gda (get default axes),

9.1.2 Drzewo obiektów

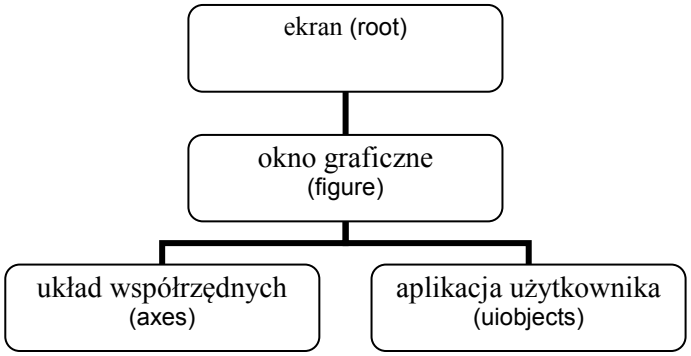
Drzewo obiektów graficzych opiera się oknach (figure). Wirtualnym korzeniem wszystkich okien aplikacji jest ekran komputera.

get(0, *'własność'*)
- *tylko wskana własność ekranu*

Uwaga:
- *ekran nie udostępnia własności* 'children'

gcf()
- *identyfikator aktualnego okna (struktura)*
- *własności aktualnego okna*

Puste okno ma niewidocznego potomka axes



get(0 [,*'własność'*])
- *wszystkie lub wskana własność ekranu*

wfig = get(0,'Children')
- *wektor identyfikatorów istniejących okien*
- *wektor pusty gdy nie ma okien*

gcf()
- *identyfikator aktualnego okna (numer)*

get(wfig(1)), get(gcf())
- *własności pierwszego i aktualnego okna*

Puste okno nie ma potomków ale dorysowanie jakiegokolwiek elementu wstawia też axes

Funkcje do rysowania i opisywania wykresów wpisują tworzone obiekty w strukturę obiektów axes (układów współrzędnych). Można się odwołać do tych elementów na sposób obiektowy (p.9.2.1) lub poprzez edytor własności (p.9.2.2) oraz dorysować kolejne elementy (p.9.2.3). Obok tej funkcjonalności istnieje możliwości tworzenia aplikacji użytkownika za pomocą biblioteki obiektów uiobject (p.0).

9.2 Okno z układem współrzędnych

9.2.1 Hierarchia obiektów w oknie z wykresami

Dla ilustracji obiektowości systemu graficznego przedstawiona zostanie struktura obiektów jaka powstaje po narysowaniu dwóch wykresów z jednym układzie współrzędnych, np.: t=0 : 3.14/20 : 2*3.14; plot(t, sin(t), 'r', t, cos(t), 'c')

- **figure** – okno graficzne; potomek:
  - **axes** – układ współrzędnych; potomek:
    - **compound** – zbiór wykresów; lista potomków:
      - **polyline** – wykres 1
      - **polyline** – wykres 2

- **figure** – okno graficzne; potomek:
  - **axes** – układ współrzędnych; lista potomków:
    - **polyline** – wykres 1
    - **polyline** – wykres 2

1. Identyfikator oraz własności okna graficznego można uzyskać w następujący sposób

```
h1 = figure();              - wskaźnik utworzonego okna
h1 = gcf();                 - wskaźnik bieżącego okna
h1 = get('current_figure'); - jw
h1                          - struktura z własnościami okna
```

```
nr1 = figure();      - numer utworzonego okna
nr1 = gcf();         - numer bieżącego okna

kopia1 = get(nr1)    - kopia struktury z własnościami okna
```

Zmiana wybranych własności okna (pozycja i rozmiar okna, rozmiar osi , nazwa okna, tło):

set(h1,'figure_position',[10 10]); - *ustaw w lewym górnym roku* set(h1,'figure_size',[500 400]); - *ustaw rozmiar okna*	set(nr1,'position',[10 10 500 400]); - *ustaw w lewym dolnym roku oraz ustaw rozmiar okna*
h1.figure_position = [10, 10]; - *jw* h1.figure_size= [500,400]; - *jw* h1.axes_size= h1.axes_size/2; - *ustaw rozmiar osi (dopasuj okno)*	x.Position = [10 10 500 400]; set(nr1,x) - *jw (x – nowa zmienna)*
set(h1,'background', color(255,255,255)); - *biały kolor tła*	set(nr1, 'color', [1 1 1]) - *biały kolor tła*
h1.background=color(255,255,255); - *jw*	x.Color = [1 1 1]; set(nr1,x) - *jw (x – nowa zmienna)*
h1.figure_name= "Moje okno"; - *nazwa okna na belce*	

2. Identyfikator oraz własności układu współrzędnych w oknie (wskazanym lub bieżącym):

h2 = get(h1, 'children'); - *wskaźnik układu* h2 = gca(); - *wskaźnik bieżącego układu* h2 = get('current_axes'); - *jw* h2 *lub* get(h2) - *struktura z własnościami układu*	nr2=get(nr1, 'children') - *numer układu* nr2 = gca(); - *numer bieżącego układu* kopia2 = get(nr2) - *kopia struktury z własnościami układu*

Zmiana wybranych własności układu współrzędnych (wł/wył autokasowanie, wł/wył siatka, wł/wył opis siatki, zmiana skali):

set(h2, 'auto_clear', 'off'); - *wyłącz (domyślne)* set(h2, 'auto_clear', 'on'); - *włącz*	set(nr2,'NextPlot','add');); - *tzn. wyłącz autokasowanie* set(nr2,'NextPlot','replace'); - *tzn. włącz autokasowanie (domyślne)*
h2.auto_clear = 'off'; - *jw* h1.children.auto_clear='off'; - *jw*	x.NextPlot = 'add'; set(nr1,x) - *jw (x – nowa zmienna)*
set(h2, 'grid', [1,1]); - *włącz; domyślnie jest* [-1,-1]	set(nr2, 'XGrid', 'on'); set(nr2,'YGrid','on'); - *odpowiednik: grid on*
h2. grid = [1,1]; - *jw* h1.children.grid = [1,1]; - *jw*	x.XGrid = 'on'; x.YGrid = 'on'; set(nr1,x) - *jw (x – nowa zmienna)*
h2.axes_visible = 'on' ; - *opisane obie osie* h2.axes_visible = ['on', 'off']; - *opisana tylko oś pozioma* h2.axes_reverse= ['on', 'off']; - *oś pozioma odwrócona*	
h2.data_bounds = [*xmin, ymin; xmax, ymax*]; - *zakres osi*	

3. Identyfikator oraz własności wykresów w układzie współrzędnych (wskazanym lub bieżącym):

h30 = get(h2, 'children') - *wskaźnik zbioru wykresów* h3 = get(h30, 'children') - *wektor wskaźników wykresów* h3(1) - *struktura z własnościami 1 wykresu* h3(2) - *struktura z własnościami 2 wykresu*	nr3=get(nr2, 'children') - *wektor numerów wykresów* kopia31 = get(nr3(1)) - *kopia struktury z własnościami 1 wykresu* kopia32 = get(nr3(2)) - *kopia struktury z własnościami 2 wykresu*

Zmiana wybranych własności wykresu 1 (grubość linii, kolor):

set(h3(1), 'thickness', 2); - *grubość linii* set(h3(1), 'line_style', 2); - *typ liniii (tu przerywana)* set(h3(1), 'foreground', color(0,0,255)); - *typ liniii (tu blue)*	set(nr3(1),'LineWidth', 2); - *grubość linii* set(nr3(1),'LineStyle', '--'); - *typ liniii (tu przerywana)* set(nr3(1), 'color', [0 0 1]); - *kolor linii (tu blue)*
h3(1).thickness=2; - *jw*	x.LineWidth=2; x.LineStyle='--';x.Color=[0 0 1]; set(nr3(1),x) - *jw (x – nowa zmienna)*

```
h3(1).line_style=2;                    - jw
h3(1).foreground=color(0,0,255);       - jw
```

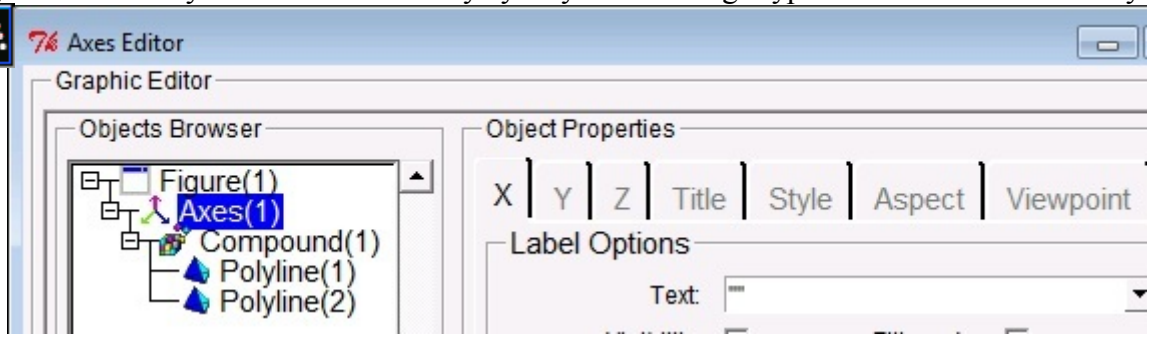
Uwaga 1. W notacji obiektowej należy zachować rozróżnienie dużych i małych liter w nazwach własności. We funkcjach get i set, gdy nazwy własności są podawane w cudzysłowach, wielkość liter nie jest istotna.
Uwaga 2. Jak widać nawet tego krótkiego przeglądu powyżej zestaw i nazwy własności obiektów graficznych pod Matlabem i Scilabem znacznie się różnią. Aby zwiększyć przenaszalność kodu lepiej do konfigurowania wykresów używać dedykowanych funkcji. Można też przygotować zestaw własnych funkcji w wersji dla Matlaba i Scilaba.

9.2.2 Edytor własności wykresów

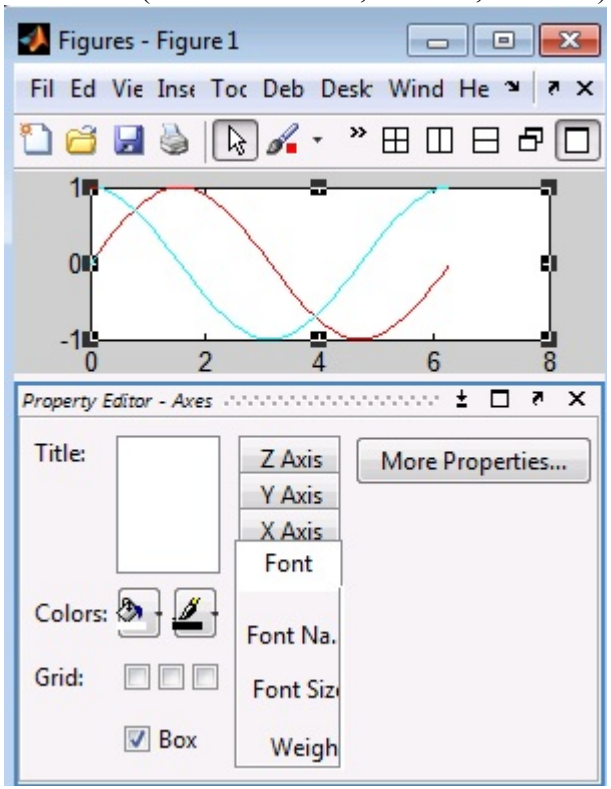
Omawiane już (p. 9.1.2) przykładowe okno z wykresami t=0 : 3.14/20 : 2*3.14; plot(t, sin(t), 'r', t, cos(t), 'c') , można poddać edycji za pomocą edytora obiektów graficznych, który można uruchomić z okna zawierającego wykres.


Edycja / Parametry okna graficznego | View / Property Editor

Rozwiązania szczegółowe edytorów obiektów Matlaba i Scilaba są różne ale zasada obsługi jest podobna – wybór obiektu do edycji powoduje wyświetlenie listy własności charakterystycznych dla danego typu obiektu i możliwość edycji tych własności (zmiana wartości, tekstów, kolorów).



Wybór obiektu przez wskazanie go w strukturze obiektów (Object Browser) lub w oknie graficznym po przejściu w tryb wyboru:
Edycja / Uruchom wybór obiektów



Poza edycją własności elementów wykresu w oknie, edytory mają przewidzianą możliwość dorysowywania dodatkowych elementów – p. 10.2.

9.2.3 Elementarne obiekty graficzne

Funkcje przeznaczone do rysowania i opisywania wykresów realizują to zadanie na bazie elementarnych obiektów graficznych (graphics primitives): układ współrzędnych, linia łamana, prostokąt, koło, tekst, bitmapa. Obiekty te można wstawiać również indywidualnie za pomocą elementarnych funkcji. Położenie dodawanych elementów jest określane we współrzędnych obiektu nadrzędnego, którym jest układ współrzędnych (axes).
Zestawienie funkcji związanych z konstrukcją elementranych obiektów można znaleźć w przeglądarce pomocy:

pod hasłem: graphics entities		pod hasłem: core graphics objects	
drawaxis([opcje])	- *ukł.współrzędnych*	axis([xmin xmax ymin ymax])	- *ukł.współrzędnych (też w plot, surf, bar)*
xarc(x,y,szer,wys,..), xfarc(...)	- *elipsa lub jej część*	image(x,y,c, *'własność', wartość*,...)	- *obraz*
xpoly(xv,yv, ...), xfpoly()	- *łamana, wielokąt*	light(*'własność', wartość*,...)	- *źródło światła*
xrect(x,y,szer,wys), xfrect(...)	- *prostokąt*	line(x,y,z, *'własność', wartość*,...)	- *linia łamana (np. w plot)*
surf(x,y,z, ...), grayplot(...) , matplot(....)	- *powierzchnia*	patch(x,y,c, *'własność', wartość*,...)	- *wielokąt (też w fill, contour3)*
xstring(x,y,tekst,...)	- *tekst*	rectangle(*'własność', wartość*, ...)	- *od prostokąta do elipsy*
xtitle, label(), legend()	- *teksty*	surface((x,y,z,c, *'własność', wartość*,...)	- *powierzchnia 3D (też w surf, mesh)*
xsegs(xv,yv,...)	- *oddzielne segmenty*	text(x,y,z,'string', *'własność', wartość*....)	- *tekst (też w title,xlabel,)*
champ(x,y,fx,fy,.....)	- *pole wektorowe*		
xarrows(nx,ny,[arsize,style])	- *wektory*		

Powyższe zestawienie pokazuje różnice zarówno w nazwach jak i w parametrach – wielokropki sygnalizują, że funkcje mają jeszcze więcej opcjonalnych parametrów. Matlab dysponuje znacznie większą gamą możliwości, co może stanowić wyzwanie do dalszego kierunki rozwoju Scilaba.

9.3 Aplikacje GUI

W bardziej zaawansowanych zastosowaniach cenną funkcjonalnością oprogramowania w obszarze GUI są komponenty do tworzenia aplikacji użytkownika. Matlab ma w tym obszarze zdecydowanie większe możliwości, co można stwierdzić nawet przez proste porównanie funkcji z grupy ui* (User Interface Object):

uicontrol — tworzy obiekt sterujący	uicontrol – tworzy obiekt sterujący
uimenu — tworzy menu lub pod menu w oknie	uimenu – menu w oknie
unsetmenu — interaktywny przycisk lub menu	uicontextmenu – menu kontekstowe
	uipushtool – przycisk (push button)
	uitoggletool – przełącznik (toggle button)
	uibuttongroup – grupa przycisków
	uipanel – panel
	uitable – tabel
	uitoolbar – pasek narzędziowy

Dostępne są standardowe okna wyboru kartotek, plików, czcionek, kolorów (w Matlabie są jeszcze inne):

uigetdir, uigetfile, uigetfont, uigetcolor

10. Dodatki

10.1 Rysowanie schematu pod edytorem Xcos (krok po kroku)

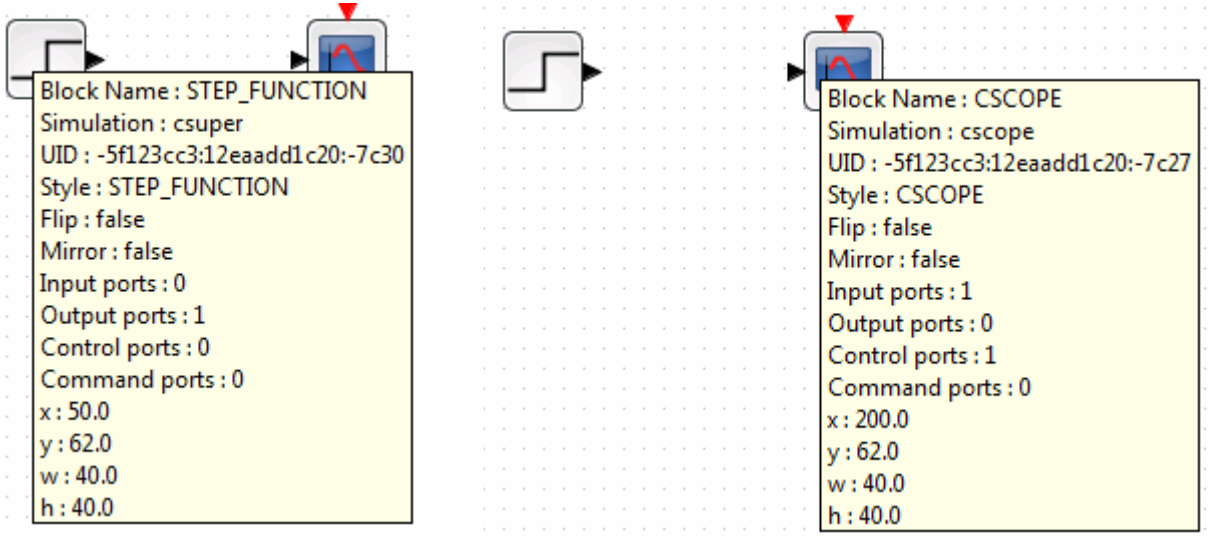
W krótkiej instrukcji krok po kroku przedstawiono sposób rysowania schematów pod edytorem Xcos, zwracając szczególną uwagę na różnice w stosunku bardziej intuicyjnego edytora Simulnika.

1. **Otwarcie biblioteki bloków**: z menu na konsli Scilab: /View/Palette browser. Otwiera się też automatycznie przy uruchomieniu Xcos. Zamknięcie wszystkich schematów automatycznie zamyka także bibliotekę.
2. **Wprowadzanie bloków:**

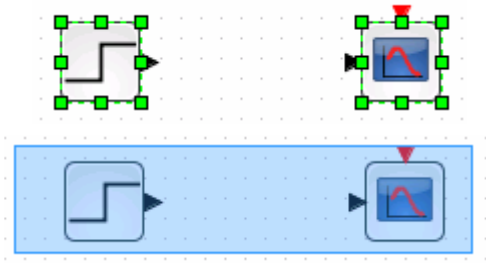
– przenoszenie bloków z biblioteki metodą przeciągania (tzn. zaznacz i trzymaj prawy lub lewy przycisk myszy, przeciągnij i puść),
– kopiowanie bloków na schemacie – metodą ctrl+c, ctrl+v.

3. **Operacje na blokach**

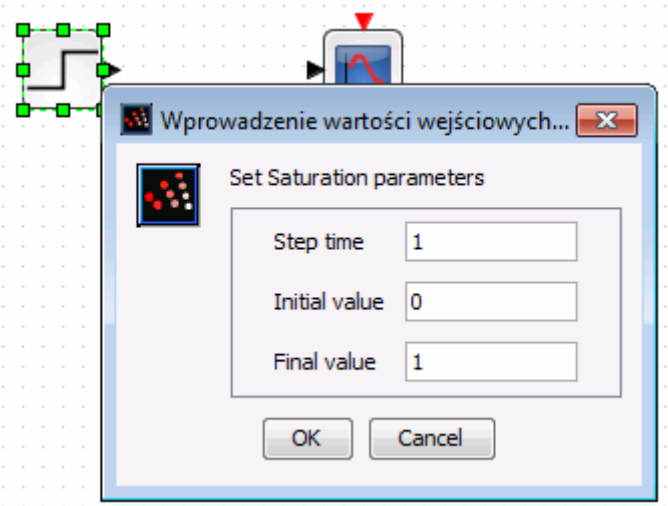
– Zatrzymanie kursora nad blokiem – wyświetlenie informacji o bloku i jego parametrach,



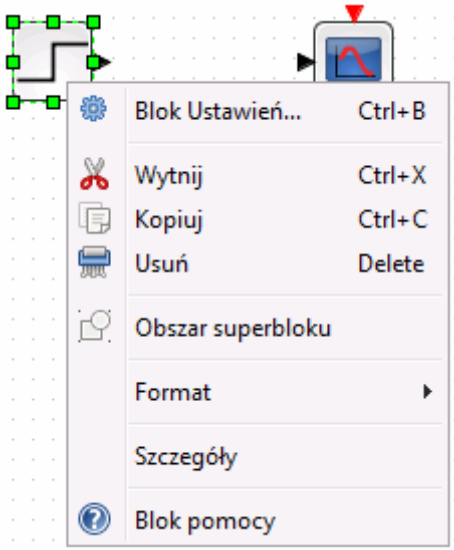
– Pojedyncze kliknięcie lewym/prawym przyciskiem myszy - wybór pojedynczego bloku,
– Pojedyncze kliknięcie z klawiszem ctrl - wybór kilku bloków czy linii (wybór z shift nie działa),
– Zaznaczenie obszaru myszą - wybór bloków i linii zawartych całkowicie w zaznaczonym obszarze,



– Podwójne kliknięcie lewym przyciskiem nad blokiem – wybiera blok i otwiera okno edycji parametrów właściwe dla danego bloku (Block Parameters),



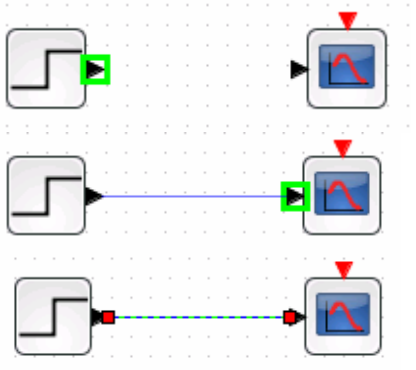
– Pojedyncze kliknięcie prawym przyciskiem na wybranym wcześniej bloku – otwarcie menu operacji na bloku



– Podwójne kliknięcie lewym przyciskiem nad pustym obszarem – wstawienie bloku, który umożliwia umieszczanie tekstów nas schemacie
  - obszar bez tekstu jest słabo widoczny – zawiera 3 kropki
  - tekst można zmienić po wybraniu obszaru - podwójne kliknięcie

4. Łączenie bloków

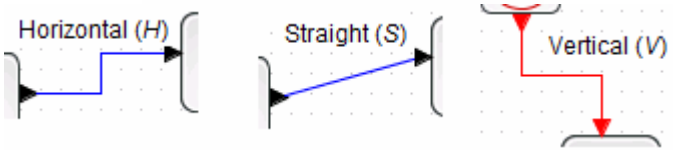
– Połączenie dwóch bloków – od portu wyjściowego do wejściowego lub odwrotnie:
  o wskazać kursorem port wyjściowy bloku (pokaże się zielony punkt)



  o trzymając lewy przycisk myszy dociągnąć do portu wejściowego kolejnego bloku (pojawi zielony punkt w miejscu docelowym) i puścić przycisk

  o potwierdzeniem poprawnie wykonanego połączenia jest pojawienie się linii o czerwonych końcach (ten sam sposób przedstawiana jest istniejące połączenie po jego wybraniu)

– Połączenia mogą być rysowane jako linie proste lub łamane pod kątem prostym – według ustawienia w Format\Link Style



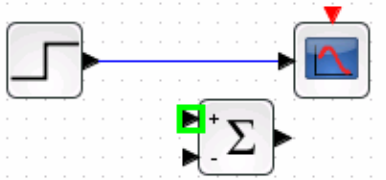
– Łączenie portów sterujących odbywa się analogicznie jak sygnałowych, jedynie gotowe połączenia różnią się kolorem – sygnałowe (niebieskie), sterujące (czerwone). Nie da się połączyć portów sygnałowych (czarnych) z portami sterującymi (czerwonymi).

– Typowe błędy podczas łączenia bloków
  o puszczenie przycisku niedokładnie nad portem (bez pojawienia się zielonego punktu) – tworzy się załamanie linii a nie jej zakończenie – najlepiej przerwać rysowanie klawiszem Esc

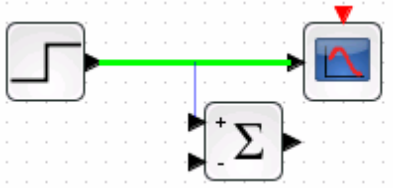
  o próba zakończenia linii przez podwójne kliknięcie (bez podłączenia do portu) – linia znika a podwójne kliknięcie wstawia blok tekstowy (trzy kropki) - nie da się rysować takich niepodłączonych linii

5. Rozgałęzienia połączeń

– Połączenie od portu wejściowego bloku do linii:
  o wskazać kursorem port wejściowy bloku (pokaże się zielony punkt)

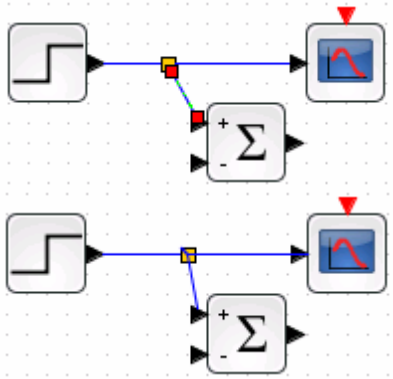


  o trzymając lewy przycisk myszy dociągnąć do linii gdzie (mniej więcej) ma być rozgałęzienie (linia zmieni kolor na zielony) i puścić przycisk – uwaga edytor może zmienić położenie punktu rozgałęzienia na linii („dociąga” do siatki)



- potwierdzeniem poprawnie wykonanego połączenia jest pojawienie się linii o czerwonych końcach i żółtego punktu rozgałęzienia (ten sam sposób przedstawiana jest istniejące połączenie po jego wybraniu)

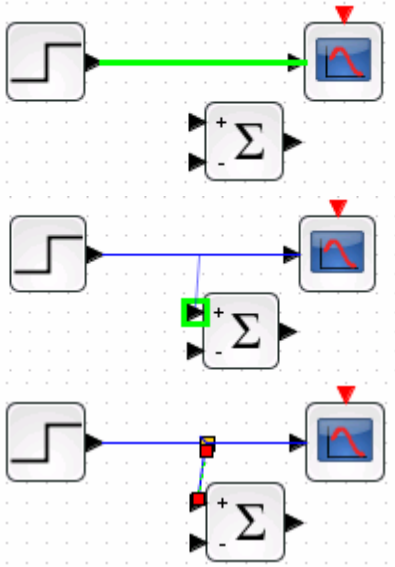
- punkt rozgałęzienia (żółty) jest widoczny na schemacie zawsze



– Połączenie od linii do portu wejściowego bloku:
  - wskazać kursorem linię gdzie (mniej więcej) ma być rozgałęzienie (linia zmieni kolor na zielony)

  - trzymając lewy przycisk myszy dociągnąć do portu wejściowego bloku (aż pokaże się zielony punkt) i puścić przycisk – uwaga edytor może zmienić położenie punktu rozgałęzienia na linii („dociąga" do siatki)

  - potwierdzeniem poprawnie wykonanego połączenia jest pojawienie się linii o czerwonych końcach i żółtego punktu rozgałęzienia



– Typowe błędy podczas tworzenia rozgałęzień
  - próba wykonania rozgałęzienia przez podwójne kliknięcie na linii - tworzy się zielony punkt do załamania linii a nie do rozgałęzienia
  - próba precyzyjnego wyznaczenia punktu rozgałęzienia na linii – poza tym, że edytor „dociąga" punkt to siatki, to zdarza się, że ta zmiana jest większa (?) – łatwiej jest najpierw zadbać o poprawne połączenie a potem można skorygować położenie punktu (⇨)

6. Operacje na liniach połączeń

– zatrzymanie kursora nad linią – zmienia się kolor odcinek linii łączący porty bloków i/lub punkty rozgałęzienia,

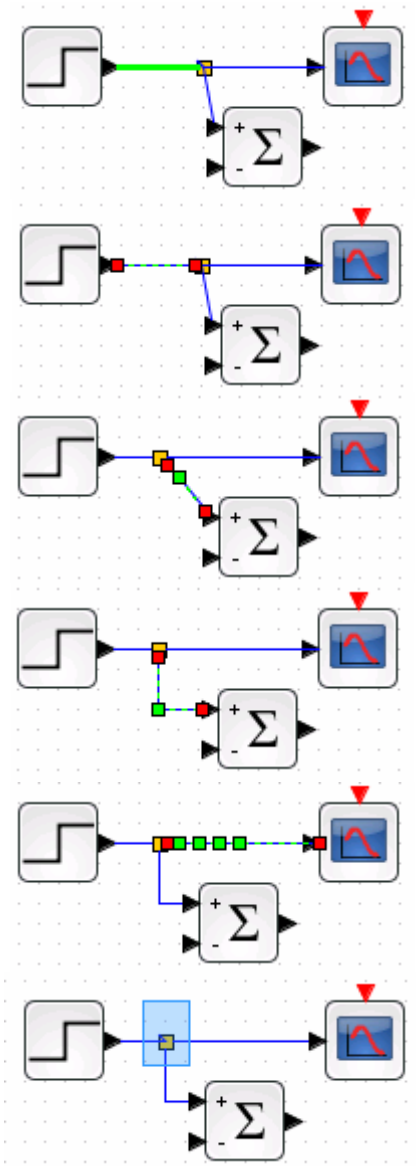
– pojedyncze kliknięcie na linii – okazują się zakończenia odcinka linii (czerwone punkty)

– podwójne kliknięcie na linii – zostaje dodany punkt (zielony), w którym można załamać linię (miejsce dodania punktu jest „dociągnięte” do siatki)

– trzymając przycisk myszy nad punktem załamania można go przenieść w inne miejsce siatki

– operację zmiany położenia punktu załamania można cofnąć, ale operacji dodawania punktów załamania nie da się cofnąć (przypadkowe kliknięcie na linię powoduje powstawanie niepotrzebnych punktów)

– przesunięcie punktu rozgałęzienia najłatwiej wykonać wybierając go (przez zaznaczenie obszaru) a następnie używając klawiszy kursora (myszą trudno to zrobić)



7. Inne operacje

- kasowanie elementów schematu
  – wybrać elementy pojedynczo (pojedyncze kliknięcie) czy grupowo (zaznaczenie obszaru) i nacisnąć klawisz Del (lub przez menu Edit \ Cut)
- cofanie operacji – klawisz ctrl+z, (lub przez menu Edit \ … lub przez ikonkę)
- skalowanie wyświetlania - View \ Zoom In, Zoom Out, …

8. Uwagi i błędy edytora xcos (!!!)

Zdarza się, że edycja uszkadza schemat w jakiś ukryty sposób – dotyczy to np. kasowania bloku „ze środka" schematu, które kasuje również połączenia związane z tym blokiem. Efekt jest taki, że przestaje działać symulacja, np.:

- nie kończy symulacji (Simulation in progress) i sygnalizuje jakieś błędy, których nie można „zidentyfikować" (np. Incorrect assignment[1])
- po przerwaniu symulacji nie można już otworzyć okna parametrów żadnego bloku na tym schemacie i na innych też (nie można uruchomić również symulacji)
- nie można połączyć któregoś z bloków na schemacie (nie pojawia się zielony punkt)

Rozwiązanie - należy skopiować zawartość schematu (ctr+c, ctr+v) do nowego, pustego okna i zapisać (uwaga pamiętać o ustawieniu czasu symulacji na schemacie). Jeśli wcześniej była przerwana symulacja, to trzeba także powtórnie uruchomić Scilaba.

10.2 Edytory graficznego interfejsu użytkownika (GUI)

Edytory GUI pozwalają przekształcić postać i zawartość okna graficznego. Takie okno może zawierać różnego typu wykresy, które za pomocą edytora można sformatować (zmienić kolory, typy linii). W oknie można dorysować i sformatować dodatkowe elementy, np. teksty, linie, figury.

10.2.1 Edytory GUI w Scilabie

Scilab udostępnia edytor Graphic Editor do wyboru i edycji elementów wykresu (osie, linie wykresów), włączany w oknie graficznym przez menu Edit. W przeglądarce pomocy wskazana jest również możliwość wstawiania nowych obiektów – menu Insert – w wersji 5.3.3 niedostepne.

10.2.2 Edytory GUI w Matlabie

Matlab oferuje znacznie bogatszy zestaw narzędzi do edycji obiektów graficznych – można je włączyć przez menu View w oknie graficznym:

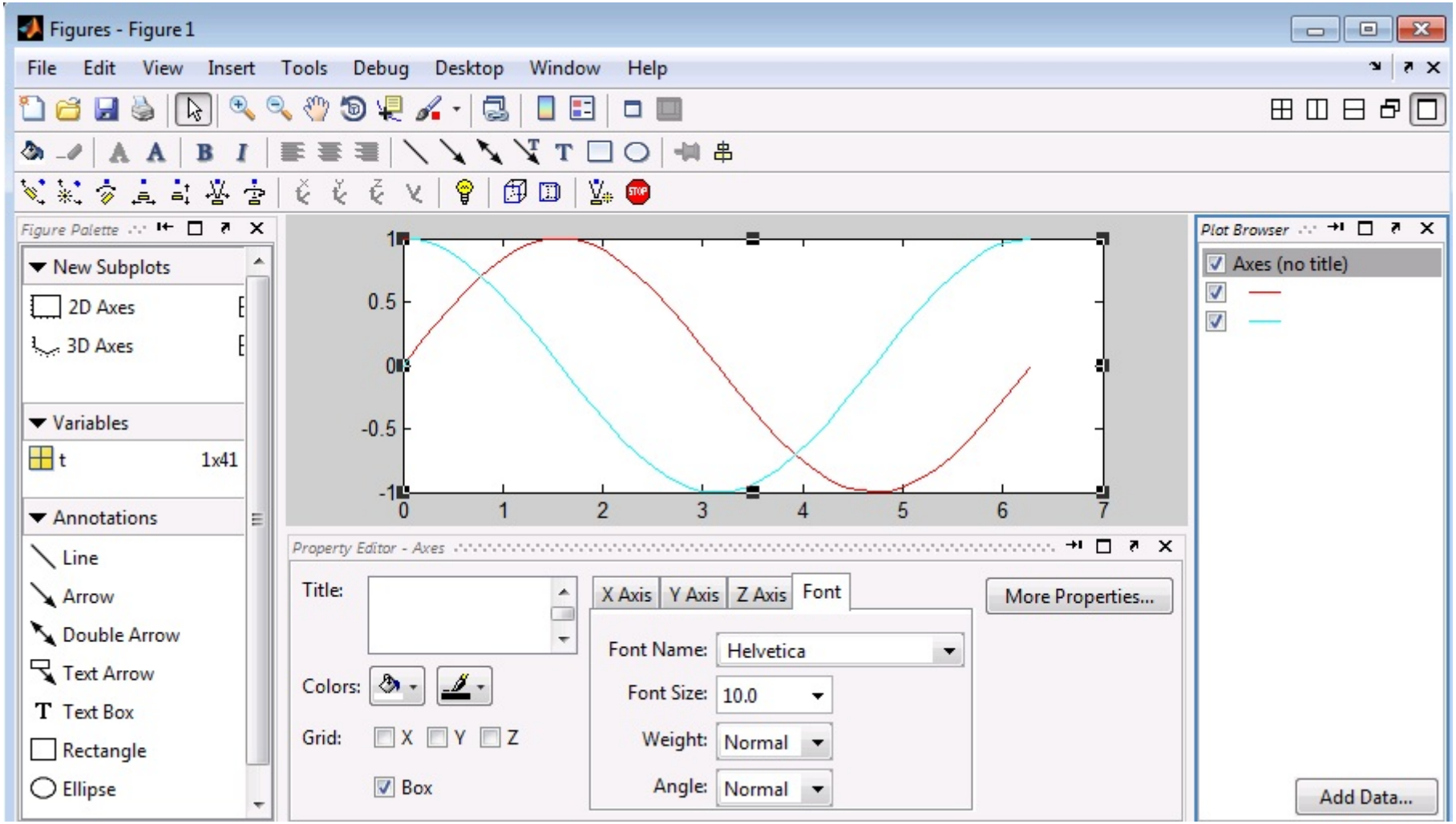
- Figure Toolbar – pasek ikon z funkcji dotyczącymi okna, np.: tworzenie i zapamiętywanie okien, obracanie, skalowanie, ...
- Camera Toolbar – pasek ikon dotyczących obrotów,
- Plot Edit Toolbar – pasek ikon dotyczących wstawiania i edycji obiektów,
- Figure Palette – lista obiektów, zmiennych, podziału okna,
- Plot Browser – wybór elementów wykresów (osie, linie wykresów),
- Property Editor – edytor dodatkowych obiektów graficznych (elementarne obiekty graficzne w oknie).

Poniżej widoczne jest okno graficzne po włączeniu wszystkich dostępnych pasków i edytorów.

---

[1] Przykład komunikatu o przerwanej symulacji uruchomionej poleceniem: exec('MojPogram.sce', -1)

```
 !--error 10000
Incorrect assignment.
at line      22 of function generic_i_s called by :
       ……..
at line     260 of function scicos_simulate called by :
   Info = scicos_simulate(scs_m,Info);//[t] = sim(model, czas, opcje);
at line      21 of exec file called by :
exec('MojPogram.sce', -1)
```


Figures - Figure 1
File Edit View Insert Tools Debug Desktop Window Help
Figure Palette
New Subplots
2D Axes
3D Axes
Variables
t 1x41
Annotations
Line
Arrow
Double Arrow
Text Arrow
Text Box
Rectangle
Ellipse
1
0.5
0
-0.5
-1
0 1 2 3 4 5 6 7
Property Editor - Axes
Title:
Colors:
Grid: X Y Z
Box
X Axis Y Axis Z Axis Font
More Properties...
Font Name: Helvetica
Font Size: 10.0
Weight: Normal
Angle: Normal
Plot Browser
Axes (no title)
Add Data...

10.3 Porównanie własności okien, układów i wykresów

Własności obiektów po wykonaniu dwóch wykresów: t=0 : 3.14/20 : 2*3.14; plot(t, sin(t), 'r', t, cos(t), 'c')

gcf()– current figure	get(gcf()) - current figure	gca() – current axes	get(gca()) – current axes
children: "Axes" **figure_position = [656,5]** **figure_size = [628,588]** axes_size = [610,460] auto_resize = "on" viewport = [0,0] figure_name = "Okno numer %d" figure_id = 0 info_message = "" color_map= matrix 32x3 pixmap = "off" pixel_drawing_mode = "copy" anti_aliasing = "off" immediate_drawing = "on" **background = -2** visible = "on" rotation_style = "unary" event_handler = "" event_handler_enable = "off" user_data = [] tag = ""	Alphamap = [(1 by 64) double array] CloseRequestFcn = closereq **Color = [0.8 0.8 0.8]** Colormap = [(64 by 3) double array] CurrentAxes = [173.001] CurrentCharacter = CurrentObject = [] CurrentPoint = [0 0] DockControls = on FileName = IntegerHandle = on InvertHardcopy = on KeyPressFcn = KeyReleaseFcn = MenuBar = figure Name = NextPlot = add NumberTitle = on PaperUnits = centimeters PaperOrientation = portrait PaperPosition = [0.634517 6.34517 20.3046 15.2284] PaperPositionMode = manual PaperSize = [20.984 29.6774] PaperType = A4 Pointer = arrow PointerShapeCData = [(16 by 16) double array] PointerShapeHotSpot = [1 1] **Position = [560 528 560 420]** Renderer = painters RendererMode = auto Resize = on ResizeFcn = SelectionType = normal ToolBar = auto Units = pixels WindowButtonDownFcn = WindowButtonMotionFcn = WindowButtonUpFcn = WindowKeyPressFcn = WindowKeyReleaseFcn = WindowScrollWheelFcn = WindowStyle = normal WVisual = 00 (RGB 16 GDI, Bitmap, Window) WVisualMode = auto BeingDeleted = off ButtonDownFcn = **Children = [173.001]** Clipping = on CreateFcn = DeleteFcn = BusyAction = queue	parent: Figure children: "Compound" visible = "on" axes_visible = ["on","on","on"] axes_reverse = ["off","off","off"] **grid = [-1,-1]** grid_position = "background" x_location = "bottom" y_location = "left" **title: "Label"** x_label: "Label" y_label: "Label" z_label: "Label" auto_ticks = ["on","on","on"] x_ticks.locations = [0;1;2;3;4;5;6;7] y_ticks.locations = matrix 11x1 z_ticks.locations = [] x_ticks.labels = ["0";"1";"2";"3";"4";"5";"6";"7"] y_ticks.labels = matrix 11x1 z_ticks.labels = [] box = "on" filled = "on" sub_ticks = [1,1] font_style = 6 font_size = 1 font_color = -1 fractional_font = "off" isoview = "off" cube_scaling = "off" view = "2d" rotation_angles = [0,270] log_flags = "nnn" tight_limits = "off" data_bounds = [0,-0.9999987;6.28,1] zoom_box = [] margins = [0.125,0.125,0.125,0.125] axes_bounds = [0,0,1,1] auto_clear = "off" auto_scale = "on" hidden_axis_color = 4 hiddencolor = 4 line_mode = "on" line_style = 0 thickness = 1 mark_mode = "off" mark_style = 0 mark_size_unit = "tabulated" mark_size = 0 mark_foreground = -1 mark_background = -2 foreground = -1	ActivePositionProperty = outerposition ALim = [0 1] ALimMode = auto AmbientLightColor = [1 1 1] Box = on CameraPosition = [3.5 0 17.3205] CameraPositionMode = auto CameraTarget = [3.5 0 0] CameraTargetMode = auto CameraUpVector = [0 1 0] CameraUpVectorMode = auto CameraViewAngle = [6.60861] CameraViewAngleMode = auto CLim = [0 1] CLimMode = auto Color = [1 1 1] CurrentPoint = [(2 by 3) double array] ColorOrder = [(7 by 3) double array] DataAspectRatio = [3.5 1 1] DataAspectRatioMode = auto DrawMode = normal FontAngle = normal FontName = Helvetica FontSize = [10] FontUnits = points FontWeight = normal GridLineStyle = : Layer = bottom LineStyleOrder = - LineWidth = [0.5] MinorGridLineStyle = : NextPlot = replace OuterPosition = [0 0 1 1] PlotBoxAspectRatio = [1 1 1] PlotBoxAspectRatioMode = auto Projection = orthographic Position = [0.13 0.11 0.775 0.815] TickLength = [0.01 0.025] TickDir = in TickDirMode = auto TightInset = [0.046428 0.040476 0.0089285 0.019047] **Title = [176.001]** Units = normalized View = [0 90] XColor = [0 0 0] XDir = normal **XGrid = off** XLabel = [177.001] XAxisLocation = bottom XLim = [0 7] XLimMode = auto XMinorGrid = off

	HandleVisibility = on HitTest = on Interruptible = on Parent = [0] Selected = off SelectionHighlight = on Tag = Type = figure UIContextMenu = [] UserData = [] Visible = on	background = -2 arc_drawing_method = "lines" clip_state = "clipgrf" clip_box = [] user_data = []	XMinorTick = off XScale = linear XTick = [(1 by 8) double array] XTickLabel = 0; 1; 2; 3; 4; 5; 6; 7 XTickLabelMode = auto XTickMode = auto YColor = [0 0 0] YDir = normal **YGrid = off** YLabel = [178.001] YAxisLocation = left YLim = [-1 1] YLimMode = auto YMinorGrid = off YMinorTick = off YScale = linear YTick = [(1 by 11) double array] YTickLabel = [(11 by 4) char array] YTickLabelMode = auto YTickMode = auto ZColor = [0 0 0] ... ZTickMode = auto BeingDeleted = off ButtonDownFcn = Children = [(2 by 1) double array] Clipping = on CreateFcn = DeleteFcn = BusyAction = queue HandleVisibility = on HitTest = on Interruptible = on Parent = [1] Selected = off SelectionHighlight = on Tag = Type = axes UIContextMenu = [] UserData = [] Visible = on

zbior=get(gca(),'children');
wyk=get(zbior,'children');
wyk(1)
	wyk=get(gca(),'children')
	get(wyk(1))		

parent: Compound children: [] visible = "on" data = matrix 41x2 closed = "off" line_mode = "on" fill_mode = "off" **line_style = 1** **thickness = 1** arrow_size_factor = 1 polyline_style = 1 **foreground = 4** background = -2 interp_color_vector = [] interp_color_mode = "off" mark_mode = "off" mark_style = 0 mark_size_unit = "point" mark_size = 0 mark_foreground = -1 mark_background = -2 x_shift = [] y_shift = [] z_shift = [] bar_width = 0 clip_state = "clipgrf" clip_box = [] user_data = []	DisplayName: '' Annotation: [1x1 hg.Annotation] **Color: [0 1 1]** **LineStyle: '-'** **LineWidth: 0.5000** Marker: 'none' MarkerSize: 6 MarkerEdgeColor: 'auto' MarkerFaceColor: 'none' XData: [1x41 double] YData: [1x41 double] ZData: [1x0 double] BeingDeleted: 'off' ButtonDownFcn: [] Children: [0x1 double] Clipping: 'on' CreateFcn: [] DeleteFcn: [] BusyAction: 'queue' HandleVisibility: 'on' HitTest: 'on' Interruptible: 'on' Selected: 'off' SelectionHighlight: 'on' Tag: '' Type: 'line' UIContextMenu: [] UserData: [] Visible: 'on' Parent: 173.0011 XDataMode: 'manual' XDataSource: '' XDataSource: '' ZDataSource: ''		

Literatura

Literatura podstawowa – wydawnictwa książkowe oraz dostępne w Internecie (chronologicznie)

1. Czemplik A., *Praktyczne wprowadzenie do opisu, analizy i symulacji dynamiki obiektów*; Oficyna Wyd. Politechniki Wrocławskiej 2012
2. Ramchandran H., Nair A.S.; *Scilab (A Free Software to Matlab);* 2011; ISBN: 978-8121939706
3. Campbell S.L., Chancelier J.P., Nikoukhah R.; *Modeling and Simulation in Scilav\b/Scicos with ScicosLab 4.4 (Second Edition*); Springer; 2010; ISBN 987-1-4419-5526-5
4. Lambert M. Surhone; *Scilab Image Processing*; 2010; ISBN: 978-6133459274
5. Brewster M.W., Gobbert M.K.; *A Comparative Evaluation of Matlab, Octave, FreeMat, and Scilab on Tara*; Technical Report HPCF–2011–10, University of Maryland, link: http://userpages.umbc.edu/~gobbert/papers/BrewsterGobbertTR2011.pdf
6. Sharma N., Gobbert M.K.; *A Comparative Evaluation of Matlab, Octave, Freemat, and Scilab for Research and Teaching*; Technical Report HPCF–2010–7, University of Maryland, link: http://userpages.umbc.edu/~gobbert/papers/SharmaGobbertTR2010.pdf
7. Czemplik A.; *Modele dynamiki układów fizycznych dla inżynierów*; WNT, Warszawa 2008
8. Lachowicz T.L.; *Matlab, Scilab, Maxima. Opis i przykłady zastosowania*; Oficyna Wydawnicza Opole 2005
9. Campbell S. L.; Nikoukhah R.; *Auxiliary Signal Design for Failure Detection*; 2004; ISBN: 0-691-09987-1
10. Urroz G.E.; *Numerical and Statistical Methods with SCILAB for Science and Engineering - Volume 1;* 2001; ISBN: 1-58898-304-8
11. Urro; G.E. *Numerical and Statistical Methods with SCILAB for Science and Engineering - Volume 2;* 2001; ISBN: 1-58898-305-6
12. Bunks C., Chancelier J.P., Delebecque F., Gomez C., Goursat M., Nikoukhah R., Steer S.; *Engineering and Scientific Computing with Scilab*; 1999, ISBN: 0-8176-4009-6
13. Pinçon B.; *Wprowadzenie do Scilaba (przekład z francuskiego z poprawkami do wersji 4.0)*; Université Henri Poincaré; link: http://www.iecn.u-nancy.fr/~szulc/intrscilabdoc.pdf

Literatura dodatkowa i strony internetowe (linki):

I1. http://wiki.scilab.org/Xcos
I2. *The best Matlab alternative* (2011), link: http://amca01.wordpress.com/2011/08/31/the-best-matlab-alternative/
I3. *Free platform for numerical computation* (2011), link: http://www.gnu.org/ghm/2011/paris/slides/sylvestre-ledru-scilab.pdf (Własności Scilab 5.3.3 i porównanie z Octave)
I4. *Scientific computation*, link http://www.dedoimedo.com/computers/scientific.html (Freemat, Ocvate, Scilab jako alternatywy Matlaba)
I5. M.Rola, Ł.Grabowski, P.Jakliński, *Dydaktyczne znaczenie programu Modelica w kształceniu technicznym*, Postępy nauki i techniki nr1/2007, link: http://pnt.pollub.pl/pdf/nr1/09.pdf
I6. Mrożek Z., *Modelowanie fizyczne, Pomiary Automatyka Robotyka nr 4/2003*, link: http://www.cyf-kr.edu.pl/~pemrozek/pdf/2003PAR.pdf
I7. Kerckoffs E.J.H., Vagheluwe H.L., Vansteenkiste G.C.,Geril P, *Simulation for the future: Progress of the Esprit Basic Research Working Group 8467*, Progress Report 1996, link: http://www.cs.mcgill.ca/~hv/publications/96.ESS.SiE.pdf (Etap do specyfikacji języka Modelica)
I8. http://pl.wikipedia.org