Wprowadzenie do modelowania systemów biologicznych oraz ich symulacji w środowisku MATLAB

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UMCS

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt „Programowa i strukturalna reforma systemu kształcenia na Wydziale Mat-Fiz-Inf".
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Człowiek-najlepsza inwestycja

UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ
WYDZIAŁ MATEMATYKI, FIZYKI I INFORMATYKI
INSTYTUT INFORMATYKI

Wprowadzenie do modelowania systemów biologicznych oraz ich symulacji w środowisku MATLAB

Ryszard Tadeusiewicz
Joanna Jaworek
Eliasz Kańtoch
Janusz Miller
Tomasz Pięciak
Jaromir Przybyło

LUBLIN 2012

Instytut Informatyki UMCS
Lublin 2012

Ryszard Tadeusiewicz (**AGH** – Akademia Górniczo-Hutnicza)
Joanna Jaworek (**AGH** – Akademia Górniczo-Hutnicza)
Eliasz Kańtoch (**AGH** – Akademia Górniczo-Hutnicza)
Janusz Miller (**AGH** – Akademia Górniczo-Hutnicza)
Tomasz Pięciak (**AGH** – Akademia Górniczo-Hutnicza)
Jaromir Przybyło (**AGH** – Akademia Górniczo-Hutnicza)

WPROWADZENIE DO MODELOWANIA SYSTEMÓW BIOLOGICZNYCH ORAZ ICH SYMULACJI W ŚRODOWISKU MATLAB

Recenzent: prof. dr hab. inż. Marian Wysocki
Opracowanie techniczne: Marcin Denkowski
Projekt okładki: Agnieszka Kuśmierska

Praca współfinansowana ze środków Unii Europejskiej w ramach
Europejskiego Funduszu Społecznego

Publikacja bezpłatna dostępna on-line na stronach
Instytutu Informatyki UMCS: informatyka.umcs.lublin.pl

Wydawca
Uniwersytet Marii Curie-Skłodowskiej w Lublinie
Instytut Informatyki
pl. Marii Curie-Skłodowskiej 1, 20-031 Lublin
Redaktor serii: prof. dr hab. Paweł Mikołajczak
www: informatyka.umcs.lublin.pl
email: dyrii@hektor.umcs.lublin.pl

Druk
FIGARO Group Sp. z o.o. z siedziba w Rykach
ul. Warszawska 10
08-500 Ryki
www: www.figaro.pl

ISBN: 978-83-62773-31-2

SPIS TREŚCI

PRZEDMOWA

0.1. Wprowadzenie

Książka ta ma pomóc Czytelnikowi w zdobyciu podstawowych wiadomości na temat modelowania matematycznego i symulacji komputerowej. **Jest to skrypt przeznaczony dla studentów informatyki i będzie ukierunkowany głównie na przekazanie wiedzy informatycznej.** Jednak mówiąc o modelowaniu oraz o symulacji musimy wskazać, co modelujemy oraz czego zachowanie chcemy badać poprzez symulację. Bez tego wszystkie rozważania utoną w ogólnikach. Piszący ten skrypt naukowcy pracują w Laboratorium Biocybernetyki AGH w Krakowie. Z tego powodu najlepiej znane (i najbardziej lubiane) są dla nich modele systemów biologicznych. Właśnie te systemy będą więc w tym skrypcie „poligonem", na bazie którego przedstawione będą metody modelowania i techniki symulacji. Autorzy wyrażają przy tym nadzieję, że przedstawiane przykłady okażą się dla Czytelników interesujące i inspirujące, ponieważ każdy z nas posiada ciało, a jego funkcjonowanie jest dla nas z pewnością bardziej interesujące, niż zachowanie jakichś tam systemów technicznych czy ekonomicznych, które także mogłyby być przedmiotem modelowania i symulacji. Nie powinno to jednak ograniczać użyteczności wiedzy, jaką Czytelnik zdobędzie w wyniku studiowania tego skryptu, bowiem sposób postępowania podczas tworzenia modelu i podczas jego komputerowej symulacji jest zawsze taki sam, niezależnie od tego, jaki obiekt modelujemy i symulujemy.

0.2. Odrobina historii, filozofii i metodologii

Jak już wskazano w poprzednim podrozdziale, książka ta poświęcona jest **modelowaniu** jako metodzie opisywania różnych systemów oraz **symulacji** komputerowej jako metodzie wykorzystania narzędzi informatycznych dla eksperymentalnej eksploracji modeli. Oczywiście w książce będziemy się starali pokazać najnowsze metody modelowania i najnowsze narzędzia służące do komputerowej symulacji. Jak się okazuje, korzeni metodologicznych rozważanych tu metod oraz technik szukać trzeba w dość odległej przeszłości, warto więc temu aspektowi przyjrzeć się nieco dokładniej.

Już w głębokiej starożytności, najpierw w Babilonii i Asyrii, potem Egipcie faraonów, a na koniec (najpełniej i najdojrzalej) - w post-aleksandryjskiej Helladzie, w nauczaniu wielu mędrców, kapłanów i filozofów dominował pogląd, że **aby poznać prawdziwą strukturę świata trzeba ją badać poprzez**

geometrię i liczby, czyli – jakbyśmy to dziś powiedzieli – metodami matematycznymi. Taki pogląd i taką metodologię naukową wyznawali i tworzyli w szczególności następujący filozofowie greccy (wymieniamy ich chronologicznie):

- Tales z Miletu,
- Pitagoras,
- Zenon z Elei,
- Demokryt z Abdery,
- Arystoteles,
- Euklides.

Użyteczność tych – jak byśmy to dziś nazwali – modeli matematycznych badali w starożytności Archimedes i Heron z Aleksandrii, a bardzo bliskie współczesnej matematyce zagadnienia numeryczne rozważał Diofantes.

Wraz z upadkiem kultury i cywilizacji antycznej ta gałąź rozwoju nauki nie umarła. Przez niekorzystny dla nauki okres europejskiego Średniowiecza pogląd, że opis matematyczny jest kluczem do zrozumienia i prawidłowego opisania świata przechowywany był i rozwijany przez matematyków arabskich, takich jak:

- Abu al-Wafa,
- Al Battani,
- Omar Chajjam,
- Muhammad ibn Musa al-Chuwarizmi[1],
- Abu Rajhan Muhammad al-Biruni.

Na przełomie XII i XIII wieku myślenie matematyczne o świecie i o toczących się w nim procesach powróciło do Europy głównie za sprawą Leonarda z Pizzy bardziej znanego jako Fibonaccio. Swój wkład w ten renesans matematyki miał też trzysta lat później wszechstronny geniusz Renesansu, Leonardo da Vinci, z modeli matematycznych korzystali wielcy astronomowie:

- Mikołaj Kopernik,
- Tycho de Brache,
- Galileusz,
- Johannes Kepler.

Uczonymi, którzy matematyczną metodologię poznawania świata wynieśli na najwyższy poziom byli:

- Rene Descartes (Kartezjusz),
- Blaise Pascal,
- Izaak Newton,
- Gottfried Leibniz.

[1] Od jego nazwiska pochodzi popularna w informatyce nazwa „algorytm”.

Oczywiście tych badaczy i myślicieli oraz genialnych matematyków było więcej, ale żeby nie zamieniać tego skryptu w namiastkę podręcznika historii matematyki przytoczono tu tylko wybranych twórców i zwolenników matematycznego opisywania rzeczywistości. Współczesna matematyka potwierdza wyznawany przez nich pogląd, że tylko matematyczny opis świata jest opisem efektywnym pod względem możliwości sprawnego wnioskowania o naturze i istocie opisywanych fragmentów świata. Jednak współczesna matematyka znacząco wzbogaca ilościowo i jakościowo zasób narzędzi, którymi można się posługiwać przy próbach modelowania rzeczywistości. W ciągu ponad dwu i pół tysiąca lat rozwoju sztuki matematycznego opisywania świata wytworzono bardzo liczne metody formalne mogące do tego służyć, przy czym na szczególną uwagę w kontekście celów tej książki zasługuje algebra i analiza matematyczna.

Przy okazji tej ogólnej dyskusji skupmy przez chwilę uwagę na pewnej kontrowersji, która zaistniała jeszcze w starożytności, a która jest żywa wśród matematyków do dziś. Otóż jedni z tych badaczy uważają, że matematyka jest po prostu wygodnym i skutecznym językiem do opisu świata. Używają jej do tego, żeby stworzyć opis badanego systemu, procesu lub zjawiska, chociaż wiedzą, że często powstający opis nie oddaje całej złożoności i bogactwa zjawiska, które opisują. Jednak stosują ją, bo wiedzą, że od starożytności do dzisiaj matematyka to najlepszy znany nam język do opisu świata. Ale trzeba pamiętać, że wśród matematyków funkcjonuje jeszcze jedno podejście do relacji matematyki i rzeczywistego świata, zwane podejściem platońskim. Wyznawcy tego podejścia uznają matematykę za podstawową i esencjonalną istotę świata, a nie tylko za język do jego opisu. Przy takim założeniu modelowanie matematyczne staje się nie tyle **tworzeniem**, co **odszukiwaniem** struktur modeli istniejących obiektywnie w otaczającej nas rzeczywistości.

W tej książce stosować będziemy pierwsze podejście, opierające się metodologicznie na pracach holenderskiego matematyka **Luitzena Egbertusa Jana Brouwera**, według którego nie można mówić o istnieniu matematycznych bytów dopóki nie poda sie sposobu na ich skonstruowanie. Matematyczne modelowanie to właśnie konstruowanie formalnych opisów dla realnych bytów. Tym się właśnie zajmiemy w tej książce, wybierając (na zasadzie przykładu) **obiekty biologiczne** jako przedmiot szczegółowego modelowania. Powody takiego wyboru zostały wyżej wskazane.

Skupimy się też na tym, co nas czeka, gdy już skonstruujemy model.

Przez ponad dwa i pół tysiąca lat twórcy modeli matematycznych różnych elementów rzeczywistego świata skazani byli na to, że analizę właściwości tworzonych modeli musieli prowadzić „ręcznie", rozwiązując analitycznie tworzone przez siebie równania. Było to bardzo pracochłonne, a często także frustrująco nieskuteczne, bowiem spora część opisów matematycznych będących dobrymi modelami fragmentów rzeczywistego świata – analitycznych

rozwiązań po prostu nie posiada. Na szczęście od połowy XX wieku w sukurs twórcom i badaczom matematycznych modeli przyszedł komputer. Od tej pory modele matematyczne z reguły eksploatuje się z ich pomocą, a **symulacja komputerowa** to powszechnie stosowana metoda badania właściwości tych modeli z wykorzystaniem obliczeń numerycznych.

Stosując modelowanie matematyczne i symulację komputerową musimy jednak brać pod uwagę fakt, że model matematyczny jest tworem sztucznym, powiązanym z modelowanym obiektem jedynie za pomocą rozumowania prowadzonego przez tworzącego model badacza, i na skutek tego może mieć właściwości, których badany fragment rzeczywistości wcale nie posiada. Obiekt rzeczywisty zawsze istnieje w jakiś określony sposób, zachowuje się tak, jak możemy to zaobserwować, posiada właściwości, które często możemy zmierzyć lub inaczej zbadać – słowem samym swoim istnieniem obiekt taki dostarcza rozwiązania „danego przez Naturę". Jeśli jednak obiekt ten opiszemy za pomocą matematyki, to taki opis matematyczny zachowuje się odmiennie. Może on mieć jedno rozwiązanie, może mieć wiele rozwiązań, może mieć nawet nieskończenie wiele rozwiązań - albo może nie mieć wcale rozwiązania.

Tutaj komputer nie pomoże. Posługując się jego potęgą możemy bardzo trudne zadanie rozwiązać w stosunkowo krótkim czasie nie ponosząc z tego tytułu żadnego własnego wysiłku. Ale jeśli rozwiązywany problem matematyczny ma wiele rozwiązań – to komputer znajdzie jedno z nich, na ogół nie informując wcale o tym, że istnieją jakieś inne możliwości. Tymczasem często tak bywa, że to rozwiązanie skwapliwie dostarczone przez komputer będzie nie tym rozwiązaniem, które naprawdę potrzebujemy. Jeszcze gorzej jest, jeśli „zapuścimy" komputerowe obliczenia, a rozwiązywany problem (matematyczny, a nie ten rzeczywisty, dla którego rozwiązywany opis matematyczny miał być modelem) wcale nie ma rozwiązań. Zachowanie komputera może być wtedy trudne do przewidzenia i bardzo zaskakujące dla oczekującego na rozwiązanie badacza.

Początkujący twórcy modeli matematycznych ignorują ten fakt, ograniczając swój wysiłek do tego, żeby dla rozważanego obiektu napisać równania matematyczne, które go będą opisywać, a potem uruchomić komputer, żeby sprawdzić, jak ten model się zachowuje. Tymczasem to, co uzyskają z komputera, może być bardzo różne od tego, co by chcieli otrzymać.

Dlatego przed przystąpieniem do symulacji czyli skomplikowanych i często czasochłonnych obliczeń komputerowych warto sięgnąć do teorii i dowiedzieć się, czy problem, który badamy numerycznie ma w ogóle rozwiązanie oraz ilu rozwiązań powinniśmy się spodziewać. Zagadnienie to nie będzie w tej książce szczegółowo dyskutowane, ale ważne jest, by studenci poznający zasady modelowania i symulacji komputerowej byli świadomi, że także i w tej dziedzinie warto poznawać teoretyczne właściwości tworzonych modeli, a nie ograniczać się do samego tylko sprawnego stosowania narzędzi komputerowych.

0.3. Przeznaczenie książki i jej zawartość

Jak wspomniano wyżej, książka ma posłużyć do tego, żeby student mógł poznać pojęcie modelu matematycznego, stworzonego dla opisu i badania określonego obiektu lub zjawiska. Pisząc książkę staraliśmy się, żeby jej czytelnik poznał etapy tworzenia modelu i jego identyfikacji, a następnie żeby nauczył się korzystać z popularnego narzędzia informatycznego jakim jest środowisko MATLAB, bardzo wygodnego przy tworzeniu programów symulacyjnych dla komputerowego badania właściwości zbudowanych modeli matematycznych.

Wymienionym wyżej celom poświęcone są kolejne rozdziały książki. I tak w rozdziale 1. modelowanie i symulacja komputerowa omówione są w sposób ogólny. Rozdział ten daje całościowy pogląd na temat rozważanej w skrypcie problematyki, ale nie dostarcza jeszcze żadnych konkretnych narzędzi ani do tworzenia modeli ani do ich komputerowej symulacji.

Narzędzia formalne, służące do tworzenia modeli matematycznych rozważanych systemów, opisane są w rozdziale 2. Zdefiniowane są w nim podstawowe pojęcia i podstawowe zależności, które muszą być uwzględniane w procesie modelowania. Rozważania w tym rozdziale prowadzone są na bazie bardzo prostych modeli traktowanych całkiem abstrakcyjnie (bez wiązania z jakimkolwiek konkretnym obiektem), ponieważ celem tego rozdziału jest dostarczenie Czytelnikom podstawowych wiadomości, które powinny być użyteczne niezależnie od tego, co i w jakim celu Czytelnicy będą chcieli w przyszłości modelować. Przykłady modeli konkretnych systemów, wraz ze szczegółowym pokazaniem drogi ich tworzenia i komputerowej symulacji, znajdują się w rozdziale 4.

Po zdefiniowaniu w rozdziale 2. metod formalnych służących do budowy matematycznych modeli, w następnym rozdziale (o numerze 3.) opisywane jest narzędzie, pozwalające wygodnie i sprawnie przenosić modele matematyczne do komputerów. Narzędziem tym jest program MATLAB oferujący całe środowisko dla prac związanych z tworzeniem modeli symulacyjnych, ich eksperymentalnym badaniem oraz przedstawianiem w wygodnej formie (z intensywnym wykorzystaniem grafiki komputerowej) wyników tej symulacji.

Po poznaniu ogólnych zasad tworzenia modeli matematycznych różnych systemów oraz po wprowadzeniu do metod korzystania z MATLABa Czytelnik będzie mógł prześledzić praktyczne stosowanie tych zasad i tych metod na bazie kilkunastu przykładowych modeli (matematycznych i symulacyjnych) przedstawionych w rozdziale 4. Modele te dotyczą systemów biologicznych o rosnącym stopniu złożoności – od modeli tak prostych, że prawie trywialnych, aż do modeli o sporym stopniu strukturalnej i funkcjonalnej złożoności. Śledząc budowę odpowiednich modeli oraz działanie ich symulacyjnych komputerowych reprezentacji – Czytelnik będzie mógł poznać **praktykę** modelowania i symulacji komputerowej, która jest co najmniej równie ważna, jak przedstawiana w skrypcie teoria.

Jak wspomniano wyżej (i jak wynika z tytułu skryptu) koncentrujemy się na

budowie modeli obiektów biologicznych. Jednak przy budowie takich modeli w przypadku gdy naszym głównym celem jest modelowanie obiektów biologicznych trzeba wykonać dość złożoną pracę, w której wyróżnić można następujące etapy:

1. Zebranie danych biologiczno-medycznych i ich systematyzacja.
2. Wyrażenie danych biologiczno-medycznych za pomocą pojęć matematycznych.
3. Wyszukanie teorii na temat funkcjonowania rozważanego biologicznego systemu i jego właściwości, a także uzgodnienie teoretycznych przesłanek tam, gdzie różni badacze przedstawiają różne poglądy.
4. Podanie odpowiedniego modelu matematycznego, np. funkcji, równania różniczkowego itp. jako propozycji modelu rozważanego obiektu lub zjawiska. Wybór modelu oparty jest na teoriach wybranych na etapie 3. oraz na analizie danych zebranych w etapie 1.
5. Zbadanie własności modelu matematycznego za pomocą metod matematycznych. W typowych praktycznych zagadnieniach robi się to metodą ogólnego sprawdzenia, jaki charakter zależności przewiduje model, chociaż w bardziej zaawansowanych badaniach można tu przewidzieć nawet **dowodzenie** ogólnych właściwości rozwiązań dla wybranych formuł matematycznych.
6. Znalezienie konkretnych wartości parametrów występujących w ogólnej strukturze matematycznego modelu (tak zwana identyfikacja).
7. Stworzenie przy użyciu odpowiedniego narzędzia informatycznego (w naszym przypadku MATLABa) - modelu symulacyjnego badanego obiektu.
8. Przeprowadzenie serii eksperymentów numerycznych (symulacji komputerowych modelu dla wybranych warunków).
9. Interpretacja biologiczno-medyczna konstatacji teoretycznych (wynikających z analizy formuł matematycznych) i szczegółowych wyników symulacji numerycznych.
10. Porównanie wyników teoretycznych z pomiarami rzeczywistymi wykonanymi na rzeczywistym obiekcie biologicznym.
11. Ewentualna korekta modelu w celu lepszego dopasowania do danych rzeczywistych.
12. Ewentualna propozycja nowych badań eksperymentalnych w celu dalszej weryfikacji modelu.

W niniejszym skrypcie głównym celem jest nauczenie Czytelnika sposobów budowy modeli oraz metod ich komputerowej symulacji. Z tego względu z tuzina wymienionych wyżej czynności, przewidzianych wyżej jako konieczne do wykonania przy **poważnym** modelowaniu systemów biologicznych – w tej książce będziemy prezentować wyłącznie czynności wymienione pod numerami 3. oraz 7. Nie zmienia to faktu, że uważny Czytelnik obok wiedzy informatycznej uzyska także małą pigułkę wiedzy biologicznej …

ROZDZIAŁ 1

MODELOWANIE I SYMULACJA KOMPUTEROWA – OMÓWIENIE OGÓLNE

1.1. Model jako narzędzie o wielu zastosowaniach

Komputery są dziś używane wszędzie i do wszystkiego, ponieważ dzięki wymiennym programom – komputer jest dziś bardziej wielozadaniowy niż najbardziej rozbudowany szwajcarski scyzoryk (Rys. 1.1).


Rys. 1.1. Komputer jest bardzo uniwersalnym narzędziem o wielu zastosowaniach (rysunek zmontowano z obrazka dostępnego jako MS ClipArt oraz ze zdjęcia ze strony http://www.geektoys.pl/foto/1331.jpg)

To zdanie i ten rysunek są świadomym nawiązaniem do wcześniej wydanego w tej samej serii podręcznika „*Informatyka medyczna*" (którego tekst można znaleźć na stronie http://informatyka.umcs.lublin.pl/files/tadeusiewicz.pdf), ponieważ chcemy w ten sposób zasygnalizować bliski związek treści niniejszego podręcznika z tamtym, wcześniej wydanym.

W tym podręczniku skupimy się na fakcie, że jednym z zadań, jakie może spełniać komputer, jest **symulacja**, do której niezbędnym wprowadzeniem jest tworzenie **modelu** rozważanego systemu. Modelować komputerowo można dosłownie wszystko. Modele stanowią bazę współczesnych badań naukowych we wszystkich prawie dziedzinach nauki. O tym wątku będzie nieco więcej mowy w następnym podrozdziale. Modele są podstawą nowoczesnej techniki, gdyż współczesny inżynier zanim zbuduje jakiekolwiek nowe urządzenie – zawsze najpierw sprawdza jego właściwości i funkcjonowanie za pomocą matematycznych modeli i komputerowych symulacji. Bardzo podobną rolę odgrywają model i symulacje przy przewidywaniu i sterowaniu rozwoju gospodarki. Modele wykorzystuje się do analizy, opisu i prognozowania procesów społecznych a także wykorzystuje się je szeroko w obszarze polityki i zarządzania.

Podana wyżej lista bynajmniej nie jest kompletna, ale wystarczy ta garść przykładów, żeby uzasadnić tezę, iż tworzenie modeli różnych systemów oraz ich badanie przy użyciu technik komputerowej symulacji – to ważny obszar zarówno współczesnej cywilizacji we wszystkich jej wymiarach jak i ważny obszar zastosowań informatyki, angażujący ogromne moce obliczeniowe. Analizując obciążenie największych superkomputerów (Rys. 1.2) można stwierdzić, że aktualnie najbardziej znaczącym źródłem zadań, których nie da się rozwiązać za pomocą komputerów typu laptop albo komputer stacjonarny

klasy PC (*Personal Computer*) – to są właśnie problemy modelowania. Na przykład mało kto zdaje sobie sprawę z tego, jak wielkie zapotrzebowanie na moc obliczeniową stwarzają prognozy pogody (oczywiście oparte na modelach i komputerowej symulacji) albo projektowanie nowych leków.

Zapamiętajmy dwie podstawowe definicje:

- Metodyka **modelowania** rozmaitych systemów polega na znajdowaniu dla nich opisów formalnych w postaci matematycznych modeli.
- Technika **symulacji** rozmaitych systemów polega na wykorzystaniu komputera do obliczania wartości występujących w modelach matematycznych oraz na wizualizacji wyników w postaci dogodnej do analizy.

Jak z tego wynika, przy **modelowaniu** dominuje pierwiastek intelektualny. Trzeba **wymyślić,** jaki kształt modelu zastosować, co uznać za jego sygnały wejściowe[2] i wyjściowe, jak zdefiniować zmienne stanu i parametry – słowem trzeba stworzyć **koncepcję.** Natomiast symulacja ma już charakter ewidentnie techniczny – trzeba zdecydować się, jakiego użyć komputera, jakiego języka lub programu symulacyjnego, jak wprowadzać dane, jak kontrolować przebieg eksperymentu i jak interpretować wyniki.

Rys. 1.2. Superkomputer w krakowskim centrum superkomputerowo-sieciowym Cyfronet z którego korzystają między innymi autorzy tego skryptu (Źródło: http://www.cyfronet.pl/uslugi_obliczeniowe/_content/hala.jpg, dostęp wrzesień 2011)

Zwykle jest tak, że badacz (lub praktyk) chcący posłużyć się modelem jakiegoś interesującego go systemu najpierw poszukuje modelu

[2] Przytaczane tu pojęcia i terminy będą dalej dokładnie omówione, więc na obecnym etapie jeśli Czytelnik nie do końca rozumie, o czym szczegółowo jest tu mowa – to niech przejdzie chwilowo nad tym do porządku, bo wszystko będzie wkrótce szczegółowo wyjaśnione, a na obecnym etapie terminy te można uznać po prostu za figury retoryczne.

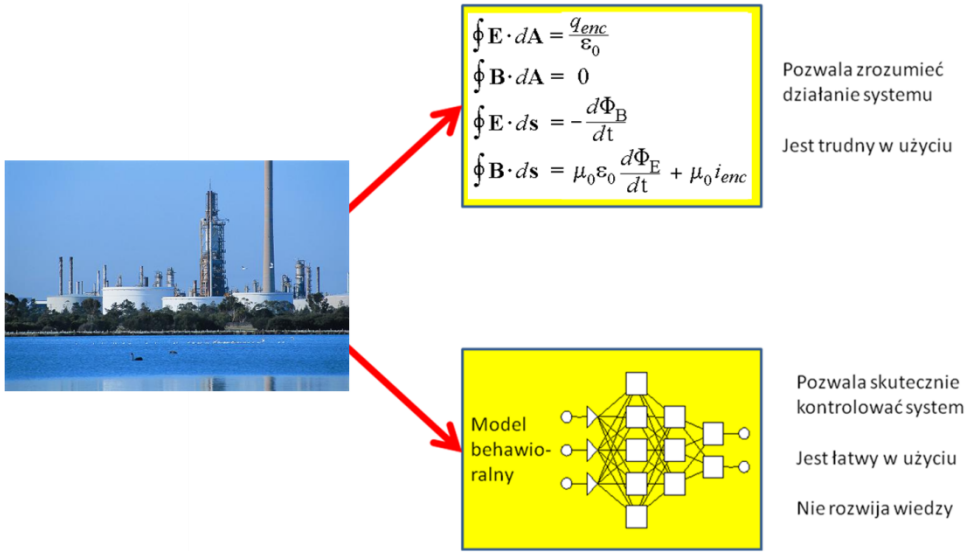
matematycznego, a potem na jego podstawie tworzy model symulacyjny, który wykorzystuje do swoich celów (opisanych skrótowo niżej). Warto może jednak wspomnieć w tym miejscu, że czasem buduje się model matematyczny nie mając na myśli przeprowadzania badań symulacyjnych, lecz po prostu traktując model jako syntetyczny i precyzyjny opis pewnego fragmentu rzeczywistości. W ten sposób powstawały modele matematyczne różnych zjawisk fizycznych, które czasem (mimo braku komputerów!) prowadziły do zupełnie nowych odkryć. Przykładowo model matematyczny elektromagnetyzmu stworzony w 1861 roku przez Jamesa Clerka Maxwella (Rys. 1.3) pozwolił odkryć fale radiowe, które bez tego modelu mogłyby być nieznane jeszcze przez całe stulecia.

Rys. 1.3. James Clerk Maxwell, którego model matematyczny nie był przedmiotem symulacji, ale i tak doprowadził do odkrycia fal radiowych (Źródło: http://66.90.101.64/arkblog/Maxwell-r.jpg, dostęp wrzesień 2011)

Możliwa jest też sytuacja odwrotna, to znaczy można prowadzić skuteczne symulacje komputerowe bez tworzenia modelu matematycznego rozważanego systemu. Możliwości takie stwarza tak zwany model behawioralny, to znaczy model którego **zachowanie** odtwarza zachowanie rzeczywistego systemu, chociaż budowa wewnętrzna modelu w niczym nie przypomina budowy rozważanego systemu, a przy tworzeniu modelu nie zachodziła potrzeba analizowania zasad działania systemu oraz nie było potrzeby śledzenia związków przyczynowo-skutkowych warunkujących jego zachowanie (Rys. 1.4).

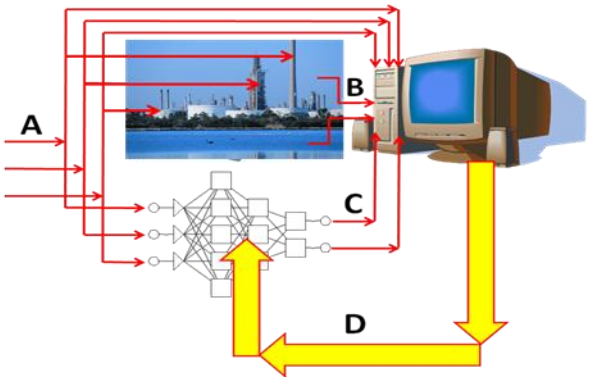
Takim behawioralnym modelem dla wielu systemów są obecnie sieci neuronowe. Nie jest to odpowiednie miejsce, żeby objaśniać, czym są te sieci i jak się je buduje (zainteresowani Czytelnicy mogą uzupełnić swoją wiedzę w tym zakresie korzystając z książki dostępnej za darmo w Internecie pod adresem http://winntbg.bg.agh.edu.pl/skrypty/0001/) natomiast warto wskazać, dzięki czemu mogą one (sieci neuronowe) modelować dowolny system nie wymagając tworzenia jego matematycznego modelu. Odpowiedzialny za to jest proces **uczenia** sieci neuronowej, którego ogólny (i bardzo uproszczony) schemat pokazano na rysunku 1.5. Na rysunku tym widać, że sieć neuronowa

(przedstawiona u dołu po lewej stronie), która ma się stać modelem rzeczywistego obiektu widocznego u góry po lewej stronie dostosowuje do tej roli komputer widoczny po prawej stronie, pełniący rolę „nauczyciela". Komputer ten obserwuje zachowanie modelowanego obiektu rejestrując sygnały sterujące docierające do niego (na rysunku oznaczone A) oraz pobierając informacje o zachowaniu obiektu (sygnał B na rysunku). Sygnały A docierają także do sieci neuronowej, która ze swojej strony produkuje sygnał C mający być przewidywaniem, co zrobi obiekt. Na początku oczywiście sygnały C mają się nijak do sygnałów B, ale rolą „nauczyciela" (realizującego odpowiedni program komputera) jest takie modyfikowanie parametrów sieci (droga tej modyfikacji jest narysowana grubymi strzałkami, oznaczona jako D), żeby stopniowo doprowadzić do uzgodnienia prognoz pochodzących z modelu (sygnał C) z rzeczywistym zachowaniem obiektu B.



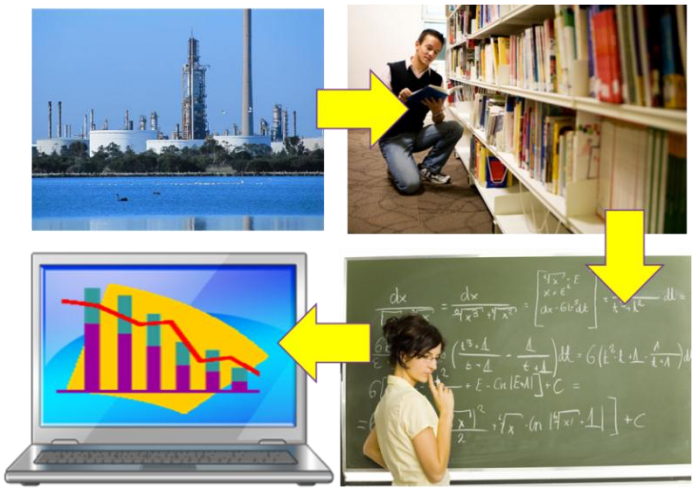
Rys. 1.4. Dla dowolnego rzeczywistego systemu możliwe jest zbudowanie dokładnego modelu matematycznego (u góry) lub modelu behawioralnego (na dole). Oba mogą być podstawą symulacji komputerowej

Opisany wyżej sposób tworzenia neuronowego modelu jest bardzo uproszczony i wiele ważnych szczegółów zostało tylko zasygnalizowanych, ale dostępne jest oprogramowanie, które potrafi robić to wszystko, co zostało wyżej opisane (komplet darmowego oprogramowania do samodzielnych eksperymentów z sieciami dostępny jest na stronie: http://home.agh.edu.pl/~tad/), więc można wypróbować działanie tego narzędzia bez zagłębiania się w szczegóły jego budowy i zasad działania.



Rys. 1.5. Tworzenie modelu poprzez uczenie sieci neuronowej. Opis w tekście.

Po tej krótkiej wycieczce do tematyki sieci neuronowych i ich stosowania jako programów symulujących różne systemy dla których nie możemy (lub nie chcemy) zbudować modelu matematycznego - wróćmy do typowego schematu postępowania, w którym budowa modelu matematycznego poprzedza komputerową symulację (Rys. 1.6) i zastanówmy się, co powoduje, że modelowanie i symulacje komputerowe są tak przydatne i w związku z tym tak popularne? Ujmując to samo zagadnienie nieco inaczej można powiedzieć tak: Zanim pokażemy, jak budować i jak wykorzystywać różne konkretne modele (w szczególności systemów biologicznych) – spróbujemy pokazać, do czego generalnie mogą się przydawać modele?

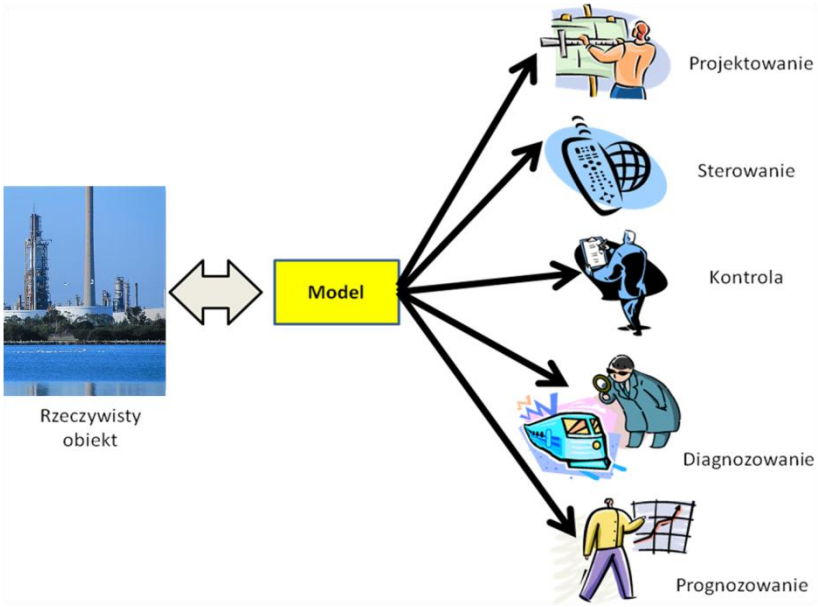


Rys. 1.6. Typowa droga od rzeczywistego obiektu, poprzez nagromadzenia i usystematyzowanie wiedzy na temat tego obiektu do modelu matematycznego i jego komputerowej symulacji (rysunek zmontowano z obrazków dostępnych w przyborniku MS ClipArt)

Otóż model dowolnego rozważanego systemu może się przydać do bardzo wielu różnych celów (Rys. 1.8).

Po pierwsze, model może być użyty na etapie projektowania. Może to być projekt różnych rzeczy: nowego systemu technicznego albo nowej strategii gospodarczej. Przy każdym projektowaniu nieocenione wręcz usługi oddaje model, który pozwala sprawdzić, jak będzie funkcjonowało to, co projektujemy, zanim jeszcze poniesiemy koszty, wysiłek i ryzyko związane z praktyczną implementacją projektowanego obiektu.

Po drugie, gdy przedmiot projektowania zostanie już zbudowany i uruchomiony (lub być może po prostu istnieje, bo był na przykład tworem przyrody a nie dziełem rąk ludzkich) to model nadal jest bardzo potrzebny, bo pozwala na lepsze sterowanie bieżącym funkcjonowaniem obiektu. Wygodnie jest bowiem mieć możliwość przewidywania na bieżąco, jakie będą skutki takiego lub innego oddziaływania na rozważany system, przed podjęciem rzeczywistego sterowania w rzeczywistym obiekcie. Błędne sterownie może bowiem drogo kosztować, a obliczenia komputerowe są tanie i coraz tańsze. Z tego powodu przy sterowaniu potrzebujemy narzędzia pozwalającego przewidywać jego skutki i umożliwiającego wybór takiego działania, które z jakiegoś punktu widzenia można uznać za najbardziej korzystne. Właśnie model symulacyjny jest takim narzędziem.



Rys. 1.8. Różne sposoby użycia modelu (rysunek zmontowano z obrazków dostępnych jako MS ClipArt)

Po trzecie, model może także pozwalać na bieżącą kontrolę funkcjonowania rozważanego systemu. Zwykle jest bowiem tak, że w skomplikowanym systemie nie wszystkie parametry charakteryzujące jego aktualne działanie da się prosto

zaobserwować i precyzyjnie pomierzyć. Często trzeba więc podejmować decyzję o tym, czy system działa dobrze, czy też zmierza do katastrofy, na podstawie obserwacji jego zachowania i oceny, jakie nieobserwowalne czynniki wewnątrz systemu takie właśnie zachowanie wymuszają. Model jest w takich sytuacjach niezwykle pomocny, gdyż pozwala łatwo i tanio sprawdzić wiele hipotez i ocenić działanie systemu bardzo precyzyjnie mimo posiadania jedynie nielicznych i zdecydowanie niekompletnych informacji.

Czwarty obszar zastosowań modelu pojawia się w przypadku, gdy monitoring wykryje, że w systemie dzieje się coś złego. Zwykle mamy wtedy do czynienia z jakąś awarią, ale przy chorym systemie podobnie jak przy chorym człowieku – bardzo ważne jest określenie rodzaju problemu, czyli postawienie diagnozy. Model może oddać przy tym wręcz nieocenione usługi!

Piąty i ostatni (z tu wymienianych) obszar zastosowań modelu matematycznego i komputerowej symulacji wiąże się z możliwościami prognostycznymi, jakie uzyskujemy gdy mamy do dyspozycji dobry model. Przewidywanie przyszłości jest ważne, potrzebne – oraz trudne. Odrzucając metody nie naukowe (wróżka, kryształowa kula, pisma kabalistyczne itp.) jako jedyne godne uwagi narzędzia przewidywania przyszłości pozostają właśnie modele. Wprawdzie różne czynniki mogą ograniczać wiarygodność prognoz uzyskiwanych przy użyciu modeli, ale wszelkie inne metody prognozowania dają jeszcze gorsze rezultaty.

1.2. Modele wykorzystywane w badaniach naukowych

Modele matematyczne i programy symulacyjne wykorzystuje się w różnych dziedzinach, bo potrzeby wymienione pod koniec poprzedniego podrozdziału pojawiają się w różnych kontekstach. Żeby jednak nie popaść w ogólnikowe stwierdzenia, że modele i symulacje przydają się **zawsze** i **wszędzie** – przedstawimy w tym podrozdziale obszerny przykład pokazujący, jak ważne i potrzebne są zastosowania modeli w jednej wybranej dziedzinie, a mianowicie w badaniach naukowych.

Odwołując się do częstej metafory, określającej obszar już poznany naukowo jako **oświetlony**, a obszar Nieznanego jako strefę **mroku** - można zastosowanie modeli wyobrazić sobie w ten sposób: Granicy między znanym i Nieznanym nie oświetla się dziesiątkami równomiernie rozmieszczonych reflektorów, bo to by było zbyt kosztowne. Tę granicę sonduje się wąską wiązką światła, tak cienką, jak promień lasera, ale dzięki temu tak jak laser przenikliwą.

Taka metaforyczna wąska wiązka światła to oczywiście konkretny eksperyment naukowy albo konkretne studium jakiegoś wybranego tematu. Jest oczywiste, że wąska wiązka światła wymaga mniej energii (czyli jest tańsza), ale w wybranym punkcie może dostarczyć więcej informacji niż ogromny reflektor. Żeby jednak przyniosła znaczący postęp wiedzy – trzeba wiedzieć, gdzie tę wąską wiązkę światła skierować. Odchodząc od metafor i koncentrując ponownie uwagę na praktyce badań naukowych można stwierdzić, że największy postęp w eksplorowaniu tych problemów naukowych, dla których

aktualnie brak niezbędnej wiedzy uzyskuje się wtedy, gdy badania dokładnie się skoncentruje na tych problemach, których rozwiązanie najskuteczniej wzbogaci wiedzę. Trzeba jednak wiedzieć, jakie to są problemy. Do tego właśnie potrzebna jest teoria, a dokładniej – najsprawniejsze jej narzędzia: matematyczne modele i symulacyjne programy.

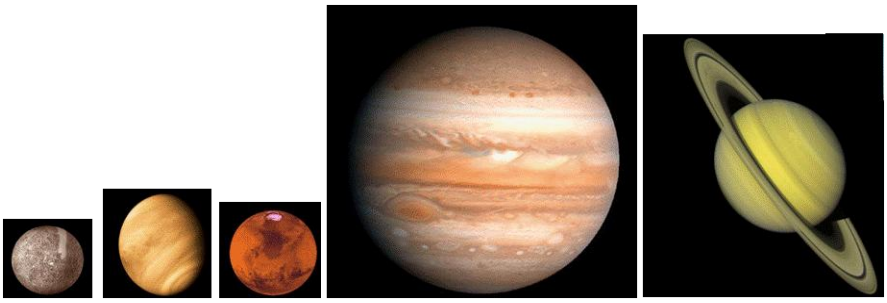
Nie mówimy tu nic nowego, bo tak właśnie badacze postępują od lat. Na przykład w chemii zwykle najpierw analizuje się rozważany problem w sposób teoretyczny (Rys. 1.9), tworzy się jego model, a dopiero potem przystępuje się do doświadczeń na rzeczywistych substancjach (Rys. 1.10).

Rys. 1.9. Rozważania teoretyczne poprzedzające badania eksperymentalne w chemii (obraz ze zbioru ClipArt programu MS Office)

Rys. 1.10. Dobrze przygotowany eksperyment chemiczny przynosi wyniki, które są w znacznym stopniu przewidywalne, a przez to są łatwiejsze do interpretacji i do wykorzystania (obraz ze zbioru ClipArt programu MS Office)

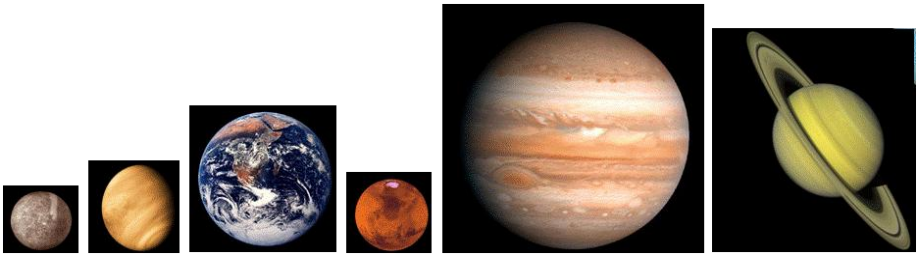
Warto przytoczyć pouczający przykład tego, jak modelowanie matematyczne przyczyniło się do sukcesu naukowego, opisując (naturalnie z uproszczeniami i w dużym skrócie) jak ludzie dokonywali odkryć w **astronomii**.

Do tego, żeby odkryć pięć pierwszych planet Układu Słonecznego (do Saturna włącznie) wystarczyła uważna obserwacja nieba. Dlatego odkrycia tego dokonali jeszcze starożytni, a astronomowie przez kilka tysięcy lat przyjmowali ich istnienie (i ich liczbę) jako pewnik (Rys. 1.11).



Rys. 1.11. Układ planetarny według starożytnych astronomów, którzy znali tylko następujące planety (od lewej do prawej): Merkury, Wenus, Mars, Jowisz, Saturn (rysunek pokazuje, które planety są większe, a które mniejsze, ale nie zachowuje proporcji)

Do tego, by Ziemię uznać za szóstą planetę potrzebny był geniusz Kopernika (Rys. 1.12), ale obserwacja tego, że Ziemia istnieje, nie wymagała specjalnego wyposażenia – wystarczyło spojrzeć dookoła.



Rys. 1.12. Układ planetarny po uwzględnieniu odkrycia Kopernika

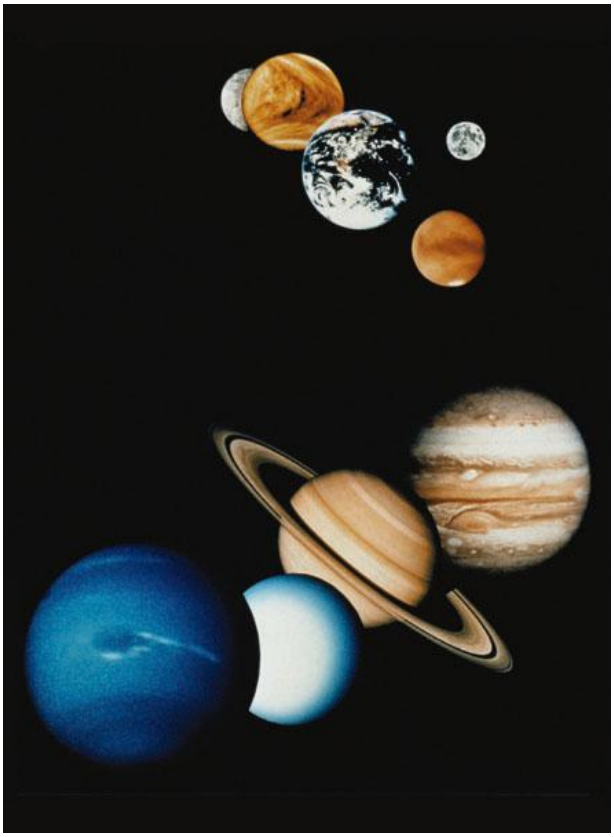
Odkrycie kolejnej planety Układu Słonecznego, nieznanego wcześniej Urana (dokonane przez Williama Herschela w 1781 roku) było skutkiem zastosowania doskonalszych narzędzi astronomicznych. Był to więc sukces czystej empirii.

Jednak następna planeta Układu, błękitny Neptun, mogła się długo ukrywać przed okiem nawet najlepiej wyposażonych astronomów, bo w miarę rozwoju coraz doskonalszych teleskopów przybywało też obiektów, które można było dostrzegać, obserwować i badać. Dlatego odkrycie Neptuna nastąpiło nie w wyniku dokładniejszych obserwacji, ale dopiero po tym, jak w 1846 roku Urbain Le Verrier przewidział (na podstawie matematycznego modelu ruchu planet), w którym miejscu sfery niebieskiej należy go szukać. Kierując teleskop we wskazany przez Le Verriera punkt nieba Johann Gottfried Galle odkrył

w ciągu kwadransa obecność planety, którą wcześniej przez setki lat obserwowały całe rzesze astronomów (jako pierwszy widział Neptuna przez swą lunetę Galileusz w 1612 roku), nie wiedząc, co w istocie obserwują.

To, że dzisiaj mamy kompletny obraz planet Układu Słonecznego (Rys. 1.13) zawdzięczamy więc najpierw dociekliwym obserwacjom starożytnych astronomów, potem genialnej teorii Kopernika, następnie udoskonalonym przyrządom Herschela i na koniec **modelowi matematycznemu**, który stworzył Le Verrier.

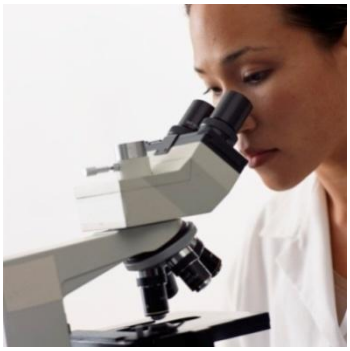
Jak wynika z przytoczonych rozważań, w naukach opisujących Przyrodę Nieożywioną modele odgrywają bardzo dużą rolę. Natomiast Przyroda Ożywiona, w tym także wiedza o naszych własnych ciałach, ich budowie, funkcjach i chorobach do niedawna uchodziła za całkowicie niepodatną na zastosowania opisanej wyżej metodologii, ponieważ złożoność procesów życiowych i ich różnorodność dosłownie obezwładniała badaczy, którzy próbowali do niej „podchodzić" z różnymi teoriami i opartymi na nich modelami. Dlatego badania biologiczne i medyczne do niedawna w niewielkim jedynie stopniu korzystały z możliwości, jakie stwarza metodyka oparta na koncepcji modelowania, a postęp w nich był w dużej mierze następstwem mnogości badań empirycznych oraz doskonalenia aparatury.



Rys. 1.13. Aktualny obraz rodziny planet Układu Słonecznego z Uranem i Neptunem na pierwszym planie oraz z Księżycem towarzyszącym Ziemi (Źródło: http://archiwum.wiz.pl/images/duze/1999/05/99054201.JPG, dostęp wrzesień 2011)

Ta sytuacja uległa jednak korzystnej zmianie. Udane próby matematycznego opisu pewnych procesów biologicznych, na przykład przemian, jakim podlega podany do organizmu lek, spowodowały, że także biolodzy i lekarze zaczęli doceniać zalety formalnego modelowania badanej rzeczywistości, połączonego z techniką komputerowej symulacji. Szczególnie ten ostatni element jest istotny, bowiem pozwala na prowadzenie eksperymentów z tworzonymi modelami. Przyjrzyjmy się może, jak ta metoda w praktyce funkcjonuje.

Model musi być oparty na dokładnych faktach empirycznych, więc w pierwszej kolejności wykonuje się liczne eksperymenty biologiczne (Rys. 1.14), a także zbiera się mnóstwo obserwacji klinicznych (Rys. 1.15).

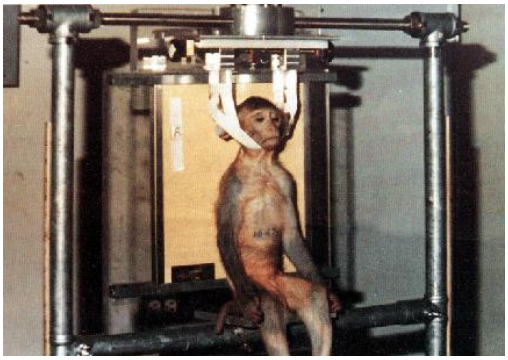


Rys. 1.14. Doświadczenia biologiczne są kosztowne i wymagają dużego wysiłku (obraz ze zbioru ClipArt programu MS Office)

Warto dodać, że wiąże się to z wielkim wysiłkiem, z dużym kosztem finansowym, a także – w wielu wypadkach – z ogromnym kosztem moralnym (związanym z koniecznością zadawania cierpienia i śmierci zwierzętom doświadczalnym Rys. 1.16).

Rys. 1.15. Również sporego wysiłku wymaga gromadzenie i systematyzacja obserwacji klinicznych (obraz ze zbioru ClipArt programu MS Office)

Tymczasem przyrost wiedzy uzyskiwany w następstwie tych wszystkich starań i wysiłków bywa niewielki, bo każda obserwacja biologiczna i każde badanie laboratoryjne obciążone jest całym mnóstwem czynników przypadkowych, zakłócających i zniekształcających poszukiwaną i odkrywaną prawdę. Dlatego mimo różnorakich kosztów wzmiankowanych wyżej nie można poprzestawać na niewielu obserwacjach i badaniach, lecz trzeba ich gromadzić dużo, weryfikując je, konfrontując i powtarzając wielokrotnie. Dopiero po odpowiednim opracowaniu statystycznym z mozaiki różnych cząstkowych obserwacji biologicznych wyłania się jakiś zestaw faktów i ustaleń, które można uznać za pewne i potwierdzone naukowo. Tak się płaci za wkraczanie na teren Nieznanego bez uprzedniego zbadania teoretycznego wchodzących w grę możliwości i bez modelowego antycypowania wyników eksperymentów jeszcze przed ich wykonaniem.



Rys. 1.16. Eksperymenty na zwierzętach wnoszą – obok innych kosztów – także koszt związany z problemem moralnym prawa do zadawania cierpień w imię nauki (Źródło: http://www.yafa.co.uk/images/ae5.gif, dostęp wrzesień 2011)

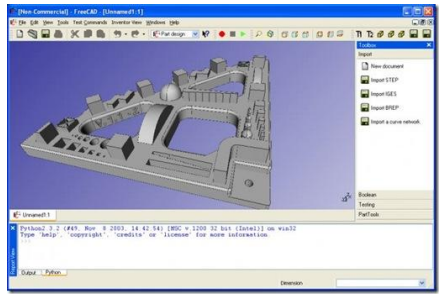
Tymczasem oparcie badań biologicznych na fundamencie teoretycznym jest możliwe i jest w pełni wykonalne. Odpowiednią bazą teoretyczną dla badań biologicznych może być omawiana dokładniej w czwartym rozdziale książki modelowanie cybernetyczne systemów biomedycznych.

1.3. Modele w zastosowaniach technicznych i gospodarczych

Modelowanie cybernetyczne jako metodologia naukowa oraz symulacja komputerowa jako technika będąca na jej usługach, są obecnie jednym z bardziej efektywnych narzędzi współczesnej nauki. Badacze korzystają z metod modelowania i symulacji komputerowej dla zgłębiania trudnych do zbadania zjawisk, a także do odkrywania współzależności i procesów zachodzących w obrębie obiektów już wstępnie poznanych, ale nie przebadanych do końca. Byłoby jednak sądem wysoce niekompletnym i wręcz nieprawdziwym, gdybyśmy ograniczyli obszar stosowania modeli matematycznych i komputerowych symulacji wyłącznie do sfery nauki. Dobrze dobrane i właściwie stosowane modele stanowią nieocenione wprost narzędzie wspomagające także sferę szeroko rozumianej praktyki.

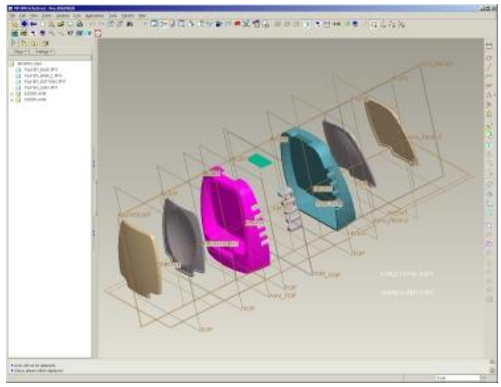
Praktycy stosują metody modelowania i symulacji aby sprawdzić konsekwencje pewnych działań, zanim je podejmą w rzeczywistości. Wymieńmy kilka przykładów.

W technice projektanci sprawdzają swoje pomysły przy użyciu modelowania i symulacji. Jest to obecnie obligatoryjny składnik procesu komputerowo wspomaganego projektowania inżynierskiego (Rys. 1.17).



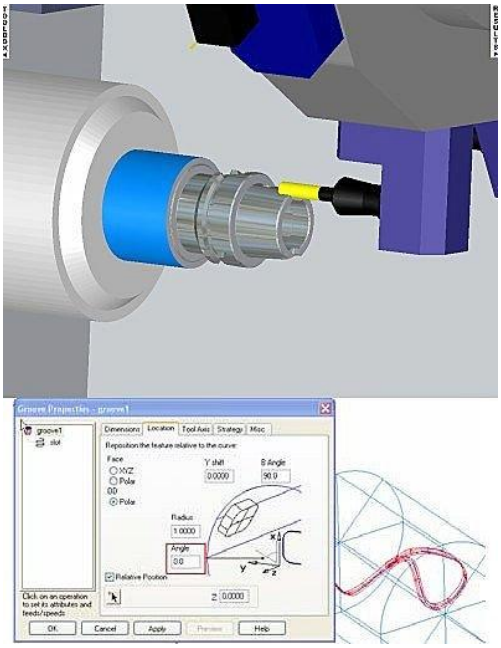
Rys. 1.17. Przy projektowaniu inżynierskim szeroko korzysta się z modeli (Źródło: http://www.dreamcss.com/2009/03/12-graphicsimagesphotographs-freeware.html, dostęp lipiec 2011)

Modelowanie komputerowe obiektów technicznych na etapie prac koncepcyjnych i konstrukcyjnych jest bardzo cennym narzędziem w ręku projektanta, bo tylko w ten sposób projektowany system może zostać sprawdzony jeszcze przed jego realizacją. Sprawdzenia może dotyczyć spraw bardzo prostych – na przykład tego, czy oddzielnie wykonywane fragmenty większej konstrukcji będą po wykonaniu pasowały do siebie (Rys. 1.18).

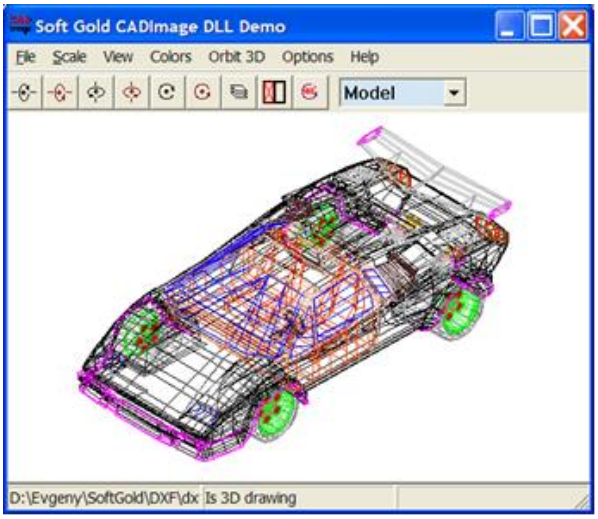


Rys. 1.18. Model jako składnik procesu projektowania inżynierskiego pozwala sprawdzić wiele właściwości projektowanego obiektu przed jego realizacją (Źródło: http://www.mclellanengineering.com/images/proescreen_u.jpg, dostęp lipiec 2011)

Modelowanie i symulacja komputerowa mogą być zastosowane nie tylko do tego, żeby projektowany element zaprojektować w sensie jego parametrów technicznych i w sensie jego wyglądu, ale także żeby zaprojektować sposób wytworzenia projektowanego urządzenia. W szczególności można na etapie projektowania wspomaganego modelem zaprojektować działanie numerycznie sterowanej obrabiarki, która zgodnie z tym projektem wykona wymaganą część mechanizmu lub inny przewidziany do wykonania detal (Rys. 1.19).



Rys. 1.19. Model pozwala zaprojektować także sposób wykonania nowego wyrobu (Źródło: http://metal-engravings.com/wp-content/uploads/2011/08/Cad-Cam.jpg, dostęp lipiec 2011)



Rys. 1.20. Ekran projektanta korzystającego z symulacji komputerowej podczas projektowania samochodu. (Źródło: http://www.safefreesoftware.com/Software/ CADSoftTools/cadimagedll.png, dostęp wrzesień 2011)

Znaczenie ważniejsze jest to, że tworząc model określonego systemu technicznego na etapie jego projektowania można także zasymulować jego działanie i sprawdzić, jak projektowany obiekt będzie się zachowywał w warunkach eksploatacji. Może to dotyczyć między innymi projektowania samochodów (Rys. 1.20). Zanim nowoczesny samochód zostanie zbudowany w formie doświadczalnego prototypu – wszystkie jego części są wszechstronnie przebadane w warunkach symulacyjnych. Badania symulacyjne kontynuuje się potem łącząc przebadane już części w większe podzespoły (na przykład elementy zawieszenia samochodu w model całego podwozia), aż wreszcie uzyskuje się model symulacyjny całego auta, który można wypróbować w warunkach symulowanej jazdy po gładkiej autostradzie, ale także w warunkach symulowanej jazdy po symulowanych nierównościach terenu.

Rys. 1.21. Modele matematyczne i symulacja komputerowa są też niezbędnymi składnikami symulatorów wykorzystywanych w celach szkoleniowych (Źródło: http://www.funfly.pl/assets/images/flight-simulator2004.jpg, dostęp lipiec 2011)

W efekcie takiego symulacyjnego „maltretowania" projektu do minimum zredukowane jest niebezpieczeństwo tego, że po realnym wybudowaniu samochodu ujawnią się jakieś nieprzewidziane usterki i prototyp trzeba będzie przebudowywać. Ma to szczególne znaczenie w przemyśle lotniczym, gdzie dzięki zastosowaniu modeli matematycznych i symulacji komputerowej do minimum zredukowano prawdopodobieństwo wypadku podczas oblatywania prototypu. Wcześniej zawód pilota-oblatywacza związany był z ogromnym ryzykiem.

Rys. 1.22. Wykorzystanie symulatora w szkoleniu kierowców (Źródło: http://resources3.news.com.au/images/2010/11/03/1225947/497239-driving-simulator.jpg, dostęp lipiec 2011)

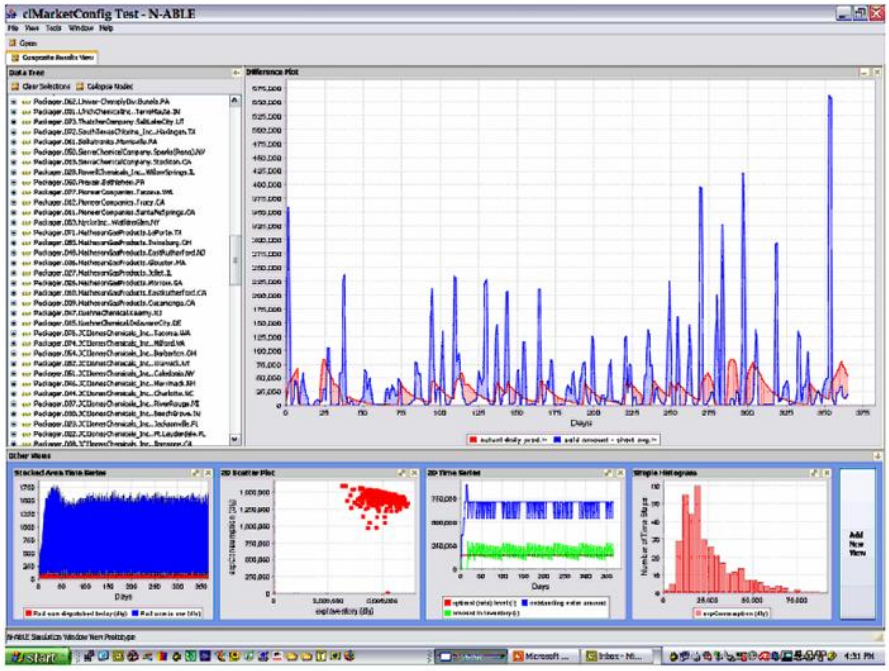
Model projektowanego pojazdu może zresztą przydać się także później, gdy pojazd ten (na przykład wspomniany wyżej samolot) będzie już zbudowany i będzie trzeba szkolić personel, który go będzie obsługiwał. Nauka pilotażu na symulatorze (Rys. 1.21) to dziś nieodzowny element szkolenia lotniczego, a potwierdzona w praktyce przydatność takich symulatorów w lotnictwie spowodowała, że analogiczne techniki stosuje się obecnie także przy szkoleniu kierowców (Rys. 1.22), a także w innych zawodach (Rys. 1.23).

Rys. 1.23. Symulatory stosuje się przy szkoleniu także w wojsku (Źródło: http://gadgets.softpedia.com/images/news/Cubic-Corporation-Develops-COMBATREDI-Fight-Simulator-2.jpg, dostęp lipiec 2011)

Podziwiając znakomitą grafikę komputerową używaną do realistycznego

przedstawienia symulowanych sytuacji oraz licznych urządzeń wspomagających działanie szkoleniowych symulatorów (na przykład w symulatorach lotu imitowany jest nie tylko obraz za oknami kokpitu, dźwięki docierające do pilotów, wibracje samolotu – ale także przyspieszenia, jakim podlegają ich ciała w różnych fazach lotu) – musimy pamiętać, że „sercem" wszystkich tych maszyn są programy symulacyjne oparte na modelach matematycznych – czyli właśnie to, o czym jest mowa w tym skrypcie.

Modele matematyczne i programy symulacyjne doskonale sprawdzają się także w zastosowaniach ekonomicznych. Jeśli potrafimy zamodelować określone zjawiska i procesy ekonomiczne, to taki model może być potem użytkowany jako element systemu wspomagania zarządzania. Jest to bardzo korzystne, bo dzięki użyciu modelu zarządzający może sprawdzić skutki określonego działania zanim się je zastosuje w realnym obiekcie (Rys. 1.24).



Rys. 1.24. Model jako element systemu wspomagania zarządzania (Źródło: http://www.sandia.gov/nisac/images/nable_1_lg.jpg, dostęp lipiec 2011)

Modele matematyczne określonych procesów ekonomicznych mogą stanowić także jądro narzędzi edukacyjnych zwanych grami kierowniczymi. Podobnie jak symulatory lotu pozwalają bezpiecznie szkolić pilotów, tak gry kierownicze dają możliwość szkolenia menedżerów, którzy mogą sprawdzić swoją wiedzę i umiejętności podejmowania trafnych decyzji na komputerowym symulatorze zanim podejmą trud prawdziwego zarządzania w warunkach prawdziwej, a nie symulowanej gry rynkowej. Ekonomiczne gry kierownicze nie

są tak fotogeniczne jak pokazane na rysunkach 1.21 i 1.22 symulatory wykorzystywane w technice, jednak obserwacja zachowań uczestników tych gier (Rys. 1.25) upewnia, że angażują one nie tylko intelekt uczestników, ale także ich emocje.

Rys. 1.25. Symulacja procesów ekonomicznych może wzbudzać prawdziwe emocje (Źródło: http://www.email-management-training.com/, dostęp lipiec 2011)

Przykłady zastosowania modeli i symulacji można by mnożyć. Nauczyciele uciekają się do modeli, bo przy ich pomocy można lepiej wyjaśnić istotę przedstawianych uczniom obiektów i systemów, niż operując samą teorią. Podczas eksperymentów symulacyjnych można dokładniej kontrolować przebieg demonstrowanego zjawiska niż w przypadku eksperymentu prowadzonego na rzeczywistych obiektach. Lekarze szkolą się na wirtualnych pacjentach. Twórcy gier komputerowych wykorzystują symulację do tego, żeby postacie i obiekty występujące w grach realistycznie poruszały się i prawidłowo reagowały na zdarzenia, będące istotą gry.

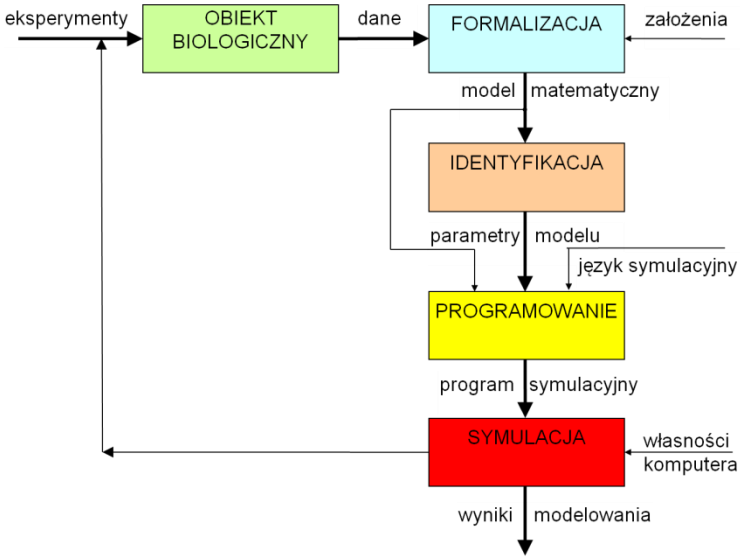
Można by było podać jeszcze bez liku przykładów pokazujących, jak bardzo użyteczna jest metodologia modelowania i technika symulacji, ale można założyć, że Czytelnik sam potrafi wskazać dalsze przykłady pozyskiwania informacji naukowych z pomocą modeli i skutecznego rozwiązywania problemów praktycznych z pomocą symulacji. Wszystkie te przykłady świadczą o tym, że metodologię modelowania oraz technikę symulacji komputerowej warto poznać i umiejętnie stosować. Temu służą następne rozdziały skryptu.

ROZDZIAŁ 2

PODSTAWOWE POJĘCIA I METODY ZWIĄZANE Z MODELOWANIEM

2.1. Wstęp

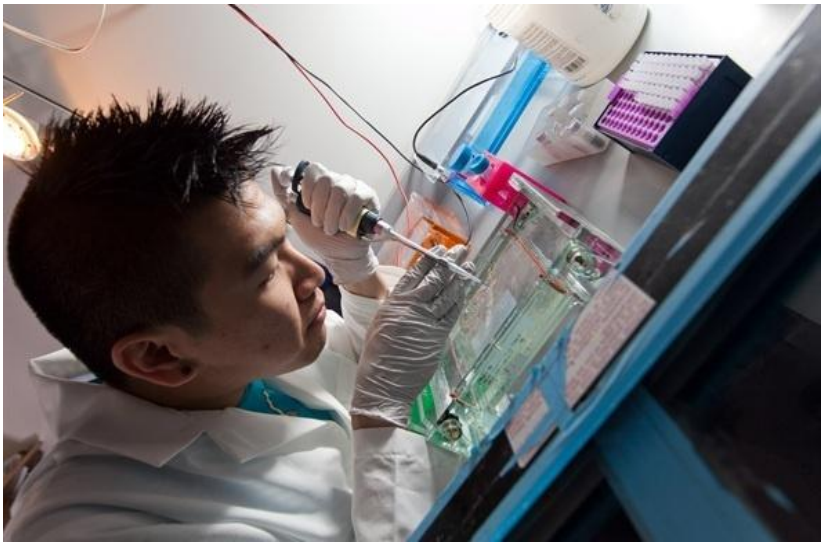
Tworzenie modelu dowolnego systemu jest procesem, w którym trzeba wykonać określone czynności w określonej kolejności i z określonymi powiązaniami pomiędzy nimi, bo tylko w ten sposób można mieć gwarancję osiągnięcia postawionego celu. Metodyka tworzenia modeli systemów była dyskutowana przez wielu autorów w wielu artykułach i książkach, my jednak (ze względów czysto sentymentalnej natury) przytoczymy tu schemat, który został po raz pierwszy zaprezentowany w 1978 roku na konferencji „Modelowanie Cybernetyczne Systemów Biologicznych", a potem został opublikowany w 1979 roku w materiałach z tej konferencji[3]. Schemat ten przedstawia rysunek 2.1. Był on po raz pierwszy wydrukowany ponad 30 lat temu – ale jego treść pozostaje aktualna do dziś.



Rys. 2.1. Elementy modelowania i komputerowej symulacji

Na rysunku 2.1 sugerowane jest, że modelowanym obiektem jest OBIEKT BIOLOGICZNY – bo właśnie taki był temat konferencji, na której schemat ten był prezentowany i taki jest także tytuł tej książki. Jednak od razu warto podkreślić, że metodyka modelowania i symulacji jest taka sama niezależnie od tego, czym jest modelowany obiekt i jakie jest przeznaczenie wyników symulacji. Na rysunek 2.1 można więc patrzeć jako na schemat bardzo ogólnej metodyki.

[3] Tadeusiewicz R.: *Metodologia modelowania cybernetycznego systemów biologicznych.* Modelowanie Cybernetyczne Systemów Biologicznych, Akademia Medyczna, Kraków 1979, str. 26-33

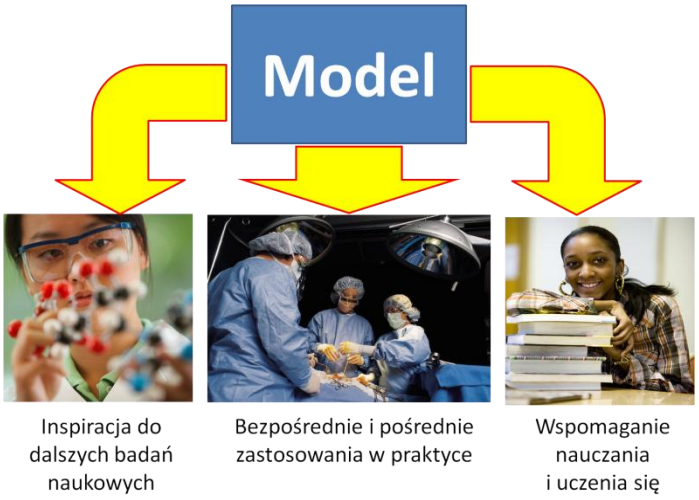


Rys. 2.2. Źródłem danych do modelowania mogą być doświadczenia in vitro (Źródło: http://medicine.yale.edu/neurobiology/Images/bannerFull_1280_70965YMS810_0021_Neurology.jpg, dostęp wrzesień 2011)

Punktem wyjścia procesu przedstawionego na rysunku 2.1 są **dane**, jakie potrafimy zebrać o modelowanym obiekcie. Dane te mogą pochodzić z biernej obserwacji zachowania obiektu i jego reakcji na zdarzenia, które towarzyszą jego normalnemu bytowaniu, względnie można prowadzić specjalne eksperymenty ukierunkowane na to, żeby te potrzebne dane uzyskać. W pierwszym przypadku mówimy o tak zwanej **identyfikacji biernej**, w drugim mamy natomiast do czynienia z tak zwaną **identyfikacją czynną.** W odniesieniu do rozważanych w tej książce systemów biologicznych identyfikacja bierna jest jedyną dozwoloną w odniesieniu do człowieka (chyba że uzyskamy zgodę komisji bioetyki na eksperyment z udziałem ludzi – w szczególności eksperyment kliniczny). Natomiast identyfikacja czynna możliwa jest do wykonania z wykorzystaniem specjalnie hodowanych lub wyosobnionych z organizmu pojedynczych komórek czy tkanek (tak zwany *eksperyment in vitro* – Rys. 2.2) względnie czynną identyfikację można prowadzić na zwierzętach doświadczalnych (tak zwany *eksperyment in vivo* – Rys. 2.3) – chociaż w tym ostatnim przypadku komisja bioetyki także musi wydać zezwolenie, bo w historii nauki zbyt często sukces badawczy był okupiony niepotrzebnym cierpieniem i śmiercią zwierząt doświadczalnych.

Rys. 2.3. Źródłem danych do modelowania mogą być też doświadczenia in vivo (Źródło: http://rockingyourboat.files.wordpress.com/2011/04/lab_mouse.jpg, dostęp wrzesień 2011)

Po zgromadzeniu odpowiedniego zasobu danych następuje etap ich FORMALIZACJI, czyli ustalania kształtu **modelu matematycznego**. Zwykle najpierw tworzony jest tak zwany **model konceptualny** czyli model obejmujący ogólne **założenia** i definicje analizowanego procesu. Stanowi on podstawę do opracowania modelu matematycznego bądź numerycznego.

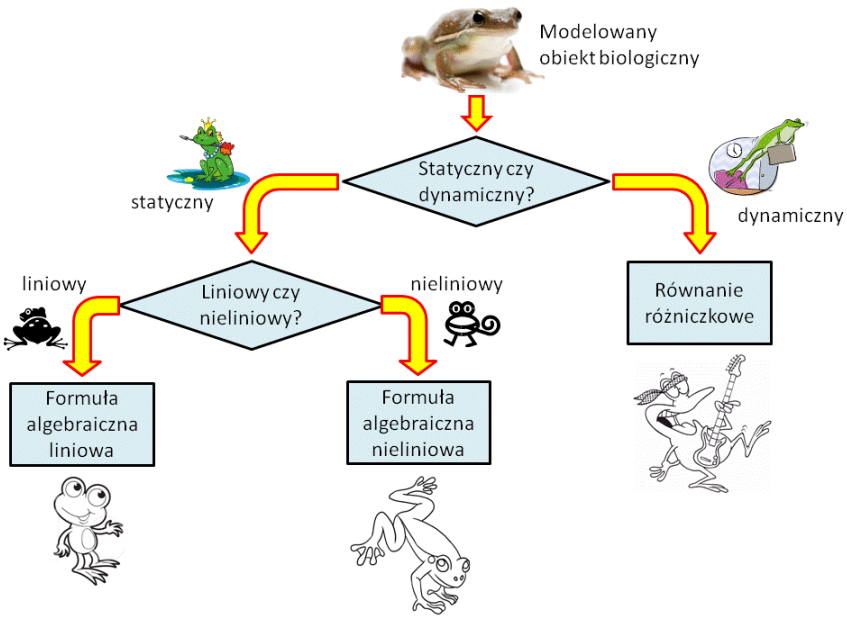


Rys. 2.4. Scenariusze wykorzystania modelu

Ważne i zaznaczone na schemacie 2.1. pojęcie **założenia** obejmuje wszystkie przyjmowane na początku procesu tworzenia modelu ustalenia determinujące kształt modelu. Założenia formułuje twórca modelu biorąc pod uwagę właściwości modelowanego obiektu oraz cel i zakres modelowania, a

także przewidywane scenariusze wykorzystania modelu. Scenariusze te mogą obejmować wykorzystanie modelu jako źródła dodatkowej inspiracji naukowej, mogą wychodzić naprzeciw określonym potrzebom praktyki (na przykład w zastosowaniu do diagnostyki medycznej i do planowania oraz kontrolowania terapii), a także mogą przewidywać wykorzystanie modelu jako narzędzia dydaktycznego (Rys. 2.4).

Ze stworzonego modelu konceptualnego wynika kształt modelu matematycznego. Oczywiście wynikanie to nie ma charakteru automatycznego, ale kierowane jest naturą modelowanego systemu. Uproszczony schemat procesu decyzyjnego towarzyszącego znajdowaniu właściwej formy modelu matematycznego przedstawia rysunek 2.5.



Rys. 2.5. Najważniejsze decyzje podejmowane podczas wybierania kształtu modelu matematycznego rozważanego obiektu (rysunek zmontowano z obrazków dostępnych w przyborniku MS ClipArt)

Po zdefiniowaniu modelu matematycznego pod względem jego kształtu zachodzi zwykle konieczność IDENTYFIKACJI tego modelu, to znaczy wyznaczenia takich wartości **parametrów** znajdujących się w wybranych (co do kształtu) równaniach, które zapewnią zgodność zachowania modelu ze **znanymi** formami zachowania rozważanego obiektu. Właśnie w tym momencie istotny staje się wspomniany wcześniej podział metod badania obiektu na identyfikację bierną i identyfikację czynną. Identyfikacja czynna trwa krócej i dostarcza bardziej wartościowych wyników, jednak nie zawsze jest możliwa do przeprowadzenia. Dane pochodzące z identyfikacji (zarówno czynnej, jak i biernej) wymagają odpowiedniego obrobienia matematycznego, żeby można

było na ich podstawie znaleźć potrzebne wartości **parametrów** modelu (patrz także podrozdział 2.2.5). Rozważmy prosty **przykład**[4].

Powiedzmy, że chcemy zbudować model związków między ilością witaminy D w pożywieniu a występowaniem zmian w kościach znanych pod krótką nazwą krzywicy. Założymy (patrz Rys. 2.5), że będzie to model statyczny i że modelowaną zależność będziemy aproksymowali modelem liniowym. W ten sposób stworzyliśmy model konceptualny rozważanego problemu.

Z podanych założeń wynika, że matematyczny model wiążący stopień zmian wywołanych krzywicą (oznaczony jako *Y*) z ilością witaminy D w pożywieniu (oznaczoną jako *X*) będzie miał formę:

$$Y = A\,X + B \tag{2.1}$$

Dla kompletu definicji modelu matematycznego musimy jeszcze określić, jak będą wyrażane ilościowo wartości *X* i *Y*. Otóż wartość *X* określająca ilość witaminy D w pożywieniu będzie określana jako *logarytm* ilości witaminy D przyjmowanej w rozważanym okresie czasu (wyrażonej w miligramach). Zastosowanie skali logarytmicznej ma w tym przypadku uzasadnienie w fakcie, że oddziaływanie większości czynników fizycznych i chemicznych na organizmy żywe jest proporcjonalne właśnie do logarytmu rozważanej wielkości, a nie do niej samej. Powszechnie znanym przykładem jest wyrażanie głośności dźwięków w decybelach, czyli w wartościach określających *logarytm* wielkości energii niesionej przez dźwięk, co lepiej odpowiada subiektywnemu wrażeniu, niż gdyby skalowana była sama energia (albo moc) fali dźwiękowej.

Wartość *Y* mająca interpretację wielkości zmian będących następstwem krzywicy określana będzie w ten sposób, że będą wykonywane zdjęcia rentgenowskie prawego stawu kolanowego i będą oceniane wielkości zmian krzywiczych widocznych na tym zdjęciu poprzez ich porównywanie z zestawem fotografii standardowych opatrzonych numerami od 1 do 12 (stopniowanie idzie w kierunku wzrastającego wyleczenia krzywicy). Każde zdjęcie rentgenowskie jest oceniane przez kilku doświadczonych radiologów, a ich oceny są uśredniane – i właśnie ta średnia stanowi rozważaną w modelu wartość *Y*.

Zadaniem, które teraz przed nami stoi jest identyfikacja modelu matematycznego podanego wzorem (2.1) to znaczy wyznaczenie wartości występujących w tym wzorze parametrów *A* i *B*. Właściwie powinniśmy się posłużyć identyfikacją bierną, to znaczy wybrać grupę dzieci (bo krzywica jest chorobą wieku dziecięcego) i przez dłuższy okres czasu (na przykład przez rok) obserwować wszystko, co jedzą, analizować sumaryczną ilość witaminy D w tym pożywieniu, a potem u wszystkich dzieci wykonać opisane wyżej badanie radiologiczne prawego stawu kolanowego – no i próbować korelować uzyskane wyniki dla określenia potrzebnych wartości parametrów *A* i *B*. Jak widać, zgodnie z tym, co zapowiedziano w trakcie dyskusji ogólnej (teoretycznej)

[4] Przykład ten zaczerpnięty został z książki: *Tadeusiewicz R., Izworski A., Majewski J.: Biometria. Skrypt Uczelniany AGH, Kraków, 1993*

identyfikacja bierna jest procesem długotrwałym, uciążliwym i kosztownym.

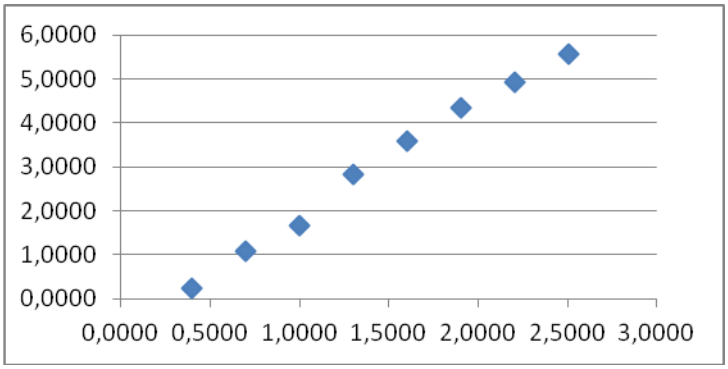
Dlatego zdecydujemy się na identyfikację czynną. Weźmiemy 8 grup młodych szczurów po 10 zwierząt w każdej grupie. Wszystkie te szczury najpierw przez dwa tygodnie będziemy karmić dietą ubogą w witaminę D, żeby wywołać u nich krzywicę. Następnie przez kolejne dwa tygodnie szczurom w każdej grupie podawać będziemy preparat zawierający witaminę D. Żeby określić wzrost skuteczności leczenia ze wzrostem dawki witaminy D – każdej grupie szczurów podamy inną dawkę. Szczury pierwszej grupy otrzymają dawkę 2,5 mg witaminy D (co odpowiada po zlogarytmowaniu wartości $X = 0{,}398$), a szczury każdej kolejnej grupy będą otrzymywały dawkę dwa razy większą, niż grupa poprzednia (druga grupa dostanie więc 5 mg co odpowiada wartości $X = 0{,}699$, a ostatnia grupa dostanie dawkę aż 320 mg co odpowiada wartości $X = 2{,}505$).

Szczury każdej grupy zostaną potem poddane badaniu radiologicznemu prawego kolana w celu ustalenia, na ile zmiany związane z krzywicą cofnęły się w wyniku leczenia witaminą D. Uśrednione wyniki w postaci wartości X i Y zestawiono w tabeli 2.1.

Tabela 2.1. Przykładowe wyniki potrzebne do identyfikacji modelu (2.1)

X	0,3980	0,6990	1,0000	1,3010	1,6020	1,9030	2,2040	2,5050
Y	0,2500	1,0833	1,6667	2,8333	3,5833	4,3333	4,9167	5,5555

Wyniki zebrane w tabeli 2.1 można obejrzeć w formie graficznej na rysunku 2.6. Widać, że rozważana zależność jest liniowa.



Rys. 2.6. Wykres danych użytych do budowy modelu

Po przeprowadzonej identyfikacji możemy przystąpić do wyznaczenia wartości parametrów modelu A i B. W rozważanym tu przypadku sprawa jest prosta, bo są łatwo dostępne narzędzia, za pomocą których można wyznaczyć te parametry jako tak zwane współczynniki regresji liniowej. Jest to możliwe do wykonania w popularnym Excelu, ale tak proste zadanie może być wykonane

nawet za pomocą kieszonkowego kalkulatora. Czytelnicy zainteresowani formułami matematycznymi zgodnie z którymi wyznaczane są parametry A i B mogą sięgnąć do wyżej cytowanej książki z której pochodzi przedstawiany przykład. Jest to łatwe, ponieważ książka ta dostępna jest w całości pod adresem:

http://winntbg.bg.agh.edu.pl/skrypty2/0086/main.html

Odpowiednie obliczenia dostarczają następującego wyniku:

$$A = 2{,}5115$$

$$B = -\ 0{,}6454$$

W ten sposób **identyfikacja** modelu została dokonana. Jak wynika z rysunku 2.1 identyfikację można niekiedy pominąć (gdy nie zależy nam na zgodności modelu z jakimś konkretnym obiektem, tylko budujemy ogólny model przeglądowy), ale zwykle identyfikację przeprowadzić trzeba.

Mając model matematyczny wyposażony w komplet prawidłowo dobranych parametrów możemy przystąpić do budowy programu symulacyjnego. Na tę czynność, określoną na rysunku 2.1. jako PROGRAMOWANIE wpływ ma wybrany do realizacji naszego modelu język symulacyjny. W tej książce założono, że używanym w niej językiem symulacyjnym (a dokładniej – środowiskiem symulacyjnym) będzie MATLAB, więc na ten temat wiele więcej w tym miejscu mówić nie będziemy, odsyłając Czytelnika do następnego (trzeciego) rozdziału, w którym będziemy mieli do czynienia z opisem MATLABa i nauką jego stosowania.

Następnym (ostatnim już) blokiem występującym na schemacie 2.1 jest SYMULACJA. Symulacja pozwala uzupełnić wyniki eksperymentów prowadzonych na rzeczywistym obiekcie biologicznym o wyniki uzyskane z komputera w rezultacie przeprowadzonych na tym komputerze eksperymentów numerycznych. W okresie, gdy przestawiony na rysunku 2.1 schemat był tworzony (to znaczy w latach 70. XX wieku) komputery były bardzo zróżnicowane, stąd zastosowana na rysunku strzałka wskazująca, że na wyniki symulacji mają wpływ właściwości używanego komputera. Dziś technika komputerowa jest bardziej zunifikowana, więc symulacja tego samego modelu na bardzo różnych komputerach (na przykład na małym laptopie albo na ogromnym superkomputerze) przebiega prawie identycznie, a różnica polega jedynie na tym, że z pomocą superkomputera wyniki modelowania możemy uzyskać znacznie szybciej, niż przy użyciu laptopa. Niemniej symulację przeprowadza się zawsze tak samo i zawsze w podobny sposób wykorzystuje się jej wyniki. Na temat tego ostatniego zagadnienia obszerniej podyskutujemy w ostatnim podrozdziale tego rozdziału.

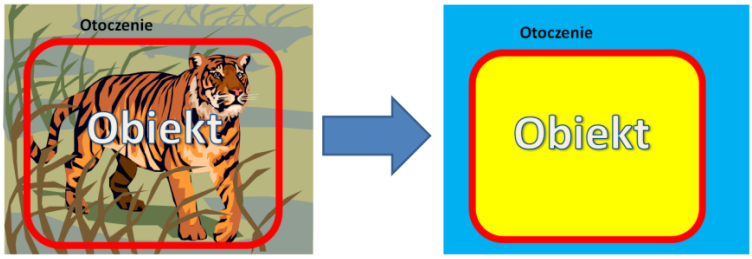
2.2. Ogólna metodyka tworzenia modeli

2.2.1. Uwagi wstępne

Przeprowadzona wyżej ogólna dyskusja i przedstawiony ogólny schemat z rysunku 2.1 muszą teraz zostać nasycone konkretną treścią, żeby Czytelnik chcący się nauczyć trudnej sztuki modelowania matematycznego i symulacji komputerowej – dysponował szczegółowymi wskazówkami, określającymi, jak powinien postępować i na co winien zwracać uwagę. Opisane niżej etapy postępowania przy tworzeniu modelu mają zastosowanie przy modelowaniu dowolnych obiektów, jednak zgodnie z metodyką przyjętą w całej tej książce będziemy ogólne rozważania ilustrowali przykładami nawiązującymi do modelowania obiektów i systemów biologicznych.

2.2.2. Wydzielenie obiektu modelowania

Przedstawiając metodykę tworzenia modeli złożonych systemów zacząć musimy od tego, że czynnością wstępną, niezmiernie potrzebną przy tworzeniu każdego modelu jest wyodrębnienie modelowanego **obiektu** i potraktowanie wszystkich innych elementów świata jako **otoczenia** (Rys. 2.7).



Rys. 2.7. Początek tworzenia modelu złożonego systemu: wydzielenie obiektu modelowania oraz jego otoczenia

W dalszym ciągu zajmować będziemy się wyłącznie obiektami, ignorując otoczenie. Zakładamy przy tym, że granice obiektu (poza ściśle zdefiniowanymi kanałami wejściowymi i wyjściowymi) są nieprzenikliwe dla informacji. Oznacza to, że do wnętrza obiektu nie docierają żadne sygnały informacyjne, jak również z wnętrza obiektu nie wydostają się żadne sygnały na zewnątrz (Rys. 2.8). Oczywiście chodzi o sygnały nie kontrolowane, bo sens modelowania polega na tym, że na wydzielony obiekt będziemy chcieli wpływać za pomocą kontrolowanych przez nas **sygnałów wejściowych**, a jego zachowanie będziemy chcieli obserwować za pomocą **sygnałów wyjściowych**. O tych sygnałach będzie jednak mowa w kolejnym podrozdziale. Natomiast sygnały przedstawione na rysunku 2.8 jako „odbijające się” od granic obiektu są to sygnały nad którymi nie mamy kontroli (w odniesieniu do sygnałów próbujących wniknąć do wnętrza obiektu) oraz sygnały, których nie potrafimy zaobserwować i wykorzystać (to w odniesieniu do sygnałów, które usiłują się

wydostać z obiektu na zewnątrz).

Rys. 2.8. Obiekty w modelowaniu traktujemy jako izolowane pod względem informacyjnym – informacja z zewnątrz może docierać do nich tylko przez wyróżnione i obserwowane kanały

Jeśli warunek wyrażony obrazowo na rysunku 2.8 nie jest spełniony, to znaczy jeśli do wnętrza obiektu dostają się niekontrolowane sygnały lub jeśli takie sygnały wydostają się na zewnątrz – to mówimy o **zakłóceniach.** Jak widać zakłócenia mogą być egzogenne lub endogenne. Egzogenne biorą się stąd, że otoczenie komunikuje się z rozważanym obiektem i wpływa na jego stan oraz determinuje jego działanie w sposób niezależny od kanałów wejściowych za pomocą których my chcemy rozważanym obiektem sterować. Na skutek oddziaływania zakłóceń egzogennych obiekt może się zachowywać w sposób dla nas nieoczekiwany, zaskakujący, nie dający się opanować – są więc one zdecydowanie szkodliwe. Zakłócenia endogenne powodują, że nasz obiekt „sieje" do otoczenia różne nie kontrolowane przez nas (i często nie dające się nawet obserwować) przekazy informacyjne. Może to być szkodliwe, lecz nie musi, więc zakłóceniami endogennymi zwykle mniej się przejmujemy.

Izolacja obiektu, o której wspomniano wyżej i której symbolem jest rysunek 2.8, dotyczy tylko aspektu informacyjnego. Rozważane przez nas obiekty, będące przedmiotem modelowania mogą otrzymywać z zewnątrz energię oraz mogą energię na zewnątrz przekazywać. Jako przykład można wskazać nagrzewanie ciała człowieka przez słońce podczas pobytu na plaży oraz wydzielanie ciepła do otoczenia przez ciało człowieka znajdującego się bez odpowiednio izolującego ubrania w chłodnym pomieszczeniu. Jeśli jednak te przekazy energetyczne nie niosą ze sobą interesujących nas informacji – to przyzwalamy na nie, eliminując je jednak z naszego pola widzenia podczas budowy modelu.

Podobnie do modelowanego obiektu mogą być dostarczane różne składniki materialne (na przykład pożywienie, woda, tlen w powietrzu oddechowym) nie psując jego doskonałej izolacji pod względem informacyjnym. To samo dotyczy ewentualnego transferu składników materialnych od rozważanego obiektu do otoczenia. Przedstawione stwierdzenia ilustruje (symbolicznie) rysunek 2.9.

Rys. 2.9. Rozważany obiekt może swobodnie wymieniać z otoczeniem energię i składniki materialne nie przestając być obiektem odosobnionym

2.2.3. Określenie sygnałów wejściowych i wyjściowych

Komunikacja obiektu z otoczeniem jest realizowana w modelu za pomocą ściśle zdefiniowanych kanałów wejściowych i wyjściowych. W kanałach tych występują określone informacje, przekazywane z zewnątrz do obiektu lub pozyskiwane z wnętrza obiektu i wykorzystywane na zewnątrz (być może przez inne obiekty). Informacje generalnie mogą mieć różną postać. Informacją jest ustnie przekazany komunikat, informacją jest wydrukowana książka, informacją jest mapa albo inny obraz, informacje w żywym organizmie przechowują geny i obrabiają komórki nerwowe. Informacją są także nasze myśli, uczucia i przeczucia. To jedno z najbardziej ogólnych i pojemnych pojęć!

Przy modelowaniu i symulacji komputerowej używać będziemy informacji występujących w bardzo szczególnej postaci. Informacje przyjmowane lub wysyłane przez modelowane obiekty w tej szczególnej postaci będziemy dalej nazywali **sygnałami**. Zgodnie ze słownikową definicją sygnał jest informacją, która daje się wyrazić liczbowo (jest mierzalny), może zmieniać się w czasie lub może zależeć od miejsca w przestrzeni, z którego się go pobiera, zwykle daje się wyrazić i opisać matematycznie. Pobieranie lub wysyłanie sygnałów związane jest z pozyskiwaniem lub przekazywaniem informacji, ale sygnał jest tylko **nośnikiem** informacji, gdyż przenoszona przez sygnał wiadomość staje się informacją dopiero wtedy, gdy trafia do obiektu (naturalnego lub sztucznego), który ją może zinterpretować i wykorzystać.

Rozważane przy modelowaniu sygnały docierają do obiektów modelowania nie w sposób dowolny i nie kontrolowany, ale ściśle wyznaczonymi kanałami. Możemy więc w modelowanym obiekcie wyróżnić **kanał wejściowy** i **sygnał wejściowy**, który będziemy się starali wiązać w dalszych rozważaniach z symbolem *X* (Rys. 2.10).

Rys. 2.10. Wejście obiektu

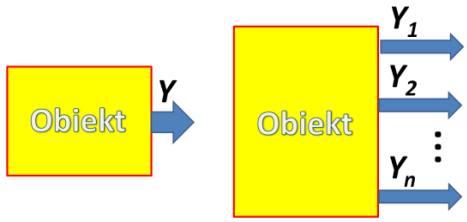
Zwykle zakłada się, że sygnał *X* przedstawiany jako wejście do rozważanego obiektu jest w jakiś sposób zależny od osoby prowadzącej modelowanie (lub eksperymentującej z rzeczywistym obiektem) albo przynajmniej może być przez nią dokładnie obserwowany. W przypadku obiektów technicznych może to być sygnał sterujący działaniem obiektu (na przykład naciskanie na pedał „gazu" przez kierowcę samochodu) albo sygnał informujący o stanie otoczenia mającym wpływ na obiekt (na przykład obserwacja, że samochód znajduje się na zboczu góry i stacza się z rosnącą prędkością, więc trzeba zamiast gazu użyć hamulca). W ekonomii źródłem sygnałów mogą być decyzje rządu lub działania banków (to sygnały sterujące) a także notowania giełdowe lub ceny surowców oraz informacje o zmiennym popycie (to sygnały obserwowalne, ale niesterowalne). W naukach społecznych sygnałem sterującym może być oddział policji wysłany do „przeprowadzenia dialogu" z demonstrantami, a sygnałem obserwowalnym – wyniki badania opinii społecznej na temat polityki rządu lub osoby premiera. Przykłady można by mnożyć.

Sygnał wejściowy może być pojedynczy (na przykład naciśnięcie przycisku dzwonka), ale może mieć także wiele składowych (na przykład zbiór czynników wpływających na nasze samopoczucie). W tym drugim przypadku chętnie uciekamy się do opisu sygnału w postaci **wektora.** Wtedy poszczególne rozważane składowe są w modelu kolejnymi składowymi tego wektora. W takim przypadku mamy do czynienia nie z jednym kanałem wejściowym, ale z wieloma (Rys. 2.11), a rozważany obiekt nazywamy **obiektem wielowymiarowym**.

Rys. 2.11. Wejście obiektu wielowymiarowego

Analogiczna sytuacja jak w opisanej wyżej kwestii przekazywania informacji od otoczenia do modelowanego obiektu – ma miejsce także przy przekazywaniu informacji z wnętrza obiektu do otoczenia. Przez analogię zdefiniujemy w związku z tym **kanał wyjściowy** oraz **sygnał wyjściowy**, który

będziemy się starali[5] oznaczać symbolem Y (Rys. 2.12).

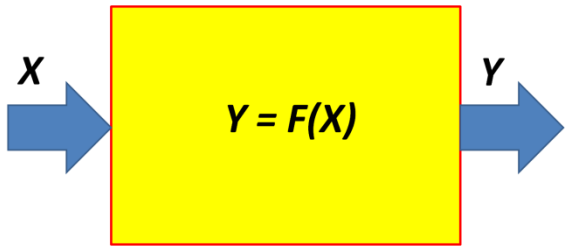


Rys. 2.12. Wyjścia jedno- i wielowymiarowego obiektu

Jest rzeczą oczywistą, że informacje, którym w rozważanym modelu przypiszemy rolę sygnałów wyjściowych, muszą być **obserwowalne.** W teorii systemów dynamicznych to pojęcie ma znacznie głębszy sens, natomiast na użytek prowadzonych tu rozważań poprzestaniemy na intuicyjnym rozumieniu tego słowa. Sygnał będziemy więc uważali za obserwowalny jeśli w dowolnym momencie czasu możliwe będzie określenie jego wartości.

2.2.4. Określenie funkcji przejścia i jej ewentualna linearyzacja

Przyjmując zgodnie z ustaleniami podanymi w poprzednim podrozdziale, że sygnały docierające do modelowanego obiektu oznaczymy X, a sygnały wysyłane przez obiekt na zewnątrz oznaczymy Y – możemy ogólny model obiektu przedstawić tak, jak to widać na rysunku 2.13.



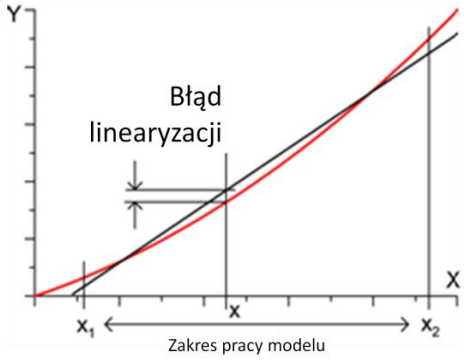
Rys. 2.13. Abstrakcyjna reprezentacja obiektu – wstęp do tworzenia modelu

Funkcja $F(X)$ wiążąca sygnały X i Y nazywana bywa **funkcją przejścia.** Jest ona głównym przedmiotem wysiłków wszystkich twórców modeli. W

[5] Sformułowanie *będziemy się starali* jest tu użyte z tego powodu, że twórca modelu nie zawsze ma możliwość swobodnego wybierania używanych oznaczeń, w wielu przypadkach bowiem pewne informacje mające charakter sygnałów wyjściowych z modelowanego obiektu mają swoje tradycyjne oznaczenia, które trudno jest *ad hoc* zmieniać. Niemniej zawsze, gdy to jest możliwe, dobrze jest stosować oznaczenia Y (lub y) dla sygnałów wyjściowych, podobnie jako oznaczenia X (lub x) dla sygnałów wejściowych.

istocie wszystkie modele opisywane w rozdziale 4. książki będą zmierzały do tego, żeby dla takich czy innych obiektów (głównie biologicznych) zaproponować sensowną postać funkcji *F(X).*

Znaleziona postać funkcji *F(X)* najlepiej opisująca rozważany obiekt (albo rozważne zjawisko bądź proces, jako że one także bywają przedmiotem modelowania) bywa czasem dość skomplikowana. Dlatego podczas modelowania zwykle najpierw buduje się funkcję przejścia możliwie dobrze przystającą do natury badanego zjawiska – ale najczęściej skomplikowaną i trudną w realizacji jak i w interpretacji. Uzyskawszy tę funkcję i upewniwszy się, że dobrze odwzorowuje ona rozważany obiekt - zmierza się potem do jej uproszczenia – żeby w efekcie uzyskać model wygodniejszy w użyciu. Metod takiego upraszczania jest wiele, ale zdecydowanie najpopularniejsza jest tu **linearyzacja.** Polega ona na tym, że próbujemy budować zależność liniową możliwie najbardziej podobną do zbudowanej wcześniej funkcji *F(X).* Ilustracją tego działania może być rysunek 2.14.



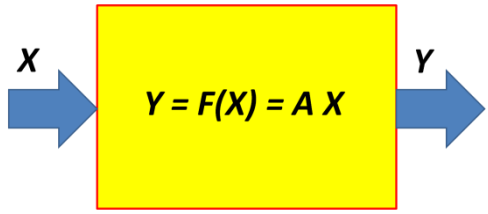
Rys. 2.14. Linearyzacja modelu

Rzeczywisty przebieg funkcji *F(X)* ilustruje krzywa linia. Dla osób czytających tę książkę w Internecie jest ona koloru czerwonego, natomiast dla osób czytających książkę w wersji papierowej jedynym kryterium wyróżniającym jest wyraźnie krzywoliniowy przebieg. Linearyzacja polega na tym, że zamiast używać oryginalnej funkcji *F(X)* decydujemy się używać zależności liniowej, przedstawionej na rysunku 2.14 w postaci prostej czarnej linii. Oczywiście taki zabieg wiąże się z powstaniem w trakcie procesu modelowania **błędu linearyzacji**. Błąd ten polega na tym, że (jak pokazano na rysunku 2.14) dla każdej wartości sygnału wejściowego *X* możemy otrzymać dwie różne wartości sygnału wyjściowego *Y* – jedną z linii krzywej reprezentującej funkcję *F(X)* i drugą z linii prostej czyli z modelu liniowego. Zakładając, że za pomocą funkcji *F(X)* otrzymujemy wartości prawdziwe (dokładne) – rozbieżność pomiędzy wartością *Y* otrzymaną dla linii prostej i wartością *Y* otrzymaną dla linii krzywej jest miarą błędu, jaki popełniać będziemy, gdy zdecydujemy się używać prostszego i wygodniejszego w użyciu modelu liniowego, zamiast dokładnego, ale niewygodnego modelu

nieliniowego. To właśnie jest błąd linearyzacji. Może on być uważany za swoistą „cenę" jaką musimy „zapłacić" za to, że korzystamy z prostszego modelu.

„Cena" jaką jest błąd linearyzacji jest w wielu przypadkach możliwa do zaakceptowania, ponieważ dla większości rzeczywistych obiektów wielkość tego błędu jest niewielka, a ponadto raz on jest dodatni (to znaczy wartość Y obliczona za pomocą modelu liniowego jest większa od wartości prawdziwej, jaką byśmy otrzymali korzystając z dokładnej funkcji $F(X)$), a innym razem ujemny, więc w wielu zastosowaniach te błędy mogą się w jakiś sposób kompensować. Błąd linearyzacji może być mało znaczący zwłaszcza wtedy, gdy zakres pracy modelu jest niewielki. Zakres pracy modelu to odstęp między maksymalną wartością sygnału X występującą podczas normalnej eksploatacji modelu, a wartością minimalną. Na rysunku 2.14 zakres pracy modelu wyznaczają wartości X_1 oraz X_2. Intuicyjnie można się domyślić, że jeśli ten zakres pracy jest wąski to zależność wyznaczana za pomocą dokładnej funkcji $F(X)$ „nie zdąży się wygiąć" tak bardzo, żeby to spowodowało powstanie dużych błędów linearyzacji.

Przyjmując do wiadomości podane wyżej rozumowanie i wynikające z niego wnioski będziemy się starali w dalszych rozważaniach korzystać z modeli liniowych, bo są one o wiele wygodniejsze. W związku z tym schemat obiektu z rysunku 2.13 przerysujemy w nowej formie (Rys. 2.15).



Rys. 2.15. Linearyzacja funkcji przejścia obiektu modelowania

W tym miejscu ustosunkujemy się do pewnego zagadnienia szczegółowego, związanego z linearyzacją. Jak wiadomo ogólna formuła liniowa ma postać daną wzorem (2.1), który tu przypomnimy:

$$Y = A\,X + B \tag{2.1}$$

Tymczasem na rysunku 2.15 zasugerowano formułę:

$$Y = A\,X \tag{2.2}$$

Otóż zrobiono to celowo, gdyż zależność dana wzorem (2.2) będzie bardzo łatwa w dalszej obróbce matematycznej, gdy zaczniemy rozważać modele bardziej złożonych systemów, składających się z wielu odpowiednio połączonych obiektów. Gdybyśmy upierali się przy wzorze (2.1) – wszystkie rozważania niepotrzebnie by się skomplikowały. W dodatku zastąpienie wzorem

(2.2) wzoru (2.1) w istocie oznacza tylko to, że otrzymywane z modelu wartości sygnału Y będą wymagały skalowania (poprzez odjęcie składowej stałej B), co jest czynnością elementarną.

Rys. 2.16. Najprostsza reprezentacja cybernetyczna rozważanego obiektu

Podsumowując: W dalszych rozważaniach używać będziemy zlinearyzowanych modeli o strukturze pokazanej na rysunku 2.16, przy czym wewnątrz bloku symbolizującego obiekt będzie wpisywana nazwa **parametru** (patrz następujący podrozdział) występującego w liniowej funkcji przejścia – w rozważanym tu przykładzie będzie to symbol *A*.

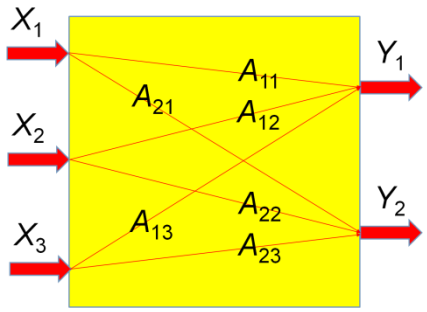
2.2.5. Parametry modelu

Model określonego systemu ma – jak już wiemy – określony kształt matematyczny. Zwykle jest to równanie (lub układ równań), które pozwala na wyznaczenie wartości sygnału wyjściowego Y na podstawie znajomości sygnału wejściowego X. Jednak w każdym takim równaniu występują pewne **parametry**, czyli współczynniki, których wartości ustalane są w procesie identyfikacji modelu (patrz podrozdział 2.1, a zwłaszcza zawarty w nim przykład). Dzięki obecności parametrów i dzięki procesowi identyfikacji, który pozwala ustalić ich wartości – ogólny model, reprezentujący pewną **klasę** obiektów, zostaje zamieniony na konkretny model konkretnego obiektu.

W przykładach, które będziemy rozważali dalej, posługiwać się będziemy możliwie najprostszymi modelami, opisywanymi równaniami podobnymi do równania 2.2 tylko oczywiście z różnymi parametrami wiążącymi X i Y. Parametry te (w równaniu 2.2 jest to parametr A) określają, ile razy większa jest każdorazowo wartość sygnału Y od wartości sygnału X, dlatego będziemy niekiedy nazywali te parametry **współczynnikami wzmocnienia** charakterystycznymi dla rozważanego obiektu. Oczywiście zależnie od wartości parametru A „wzmocnienie" to może być w istocie osłabieniem (gdy $A < 1$), a także może się wiązać ze zmianą znaku wartości sygnału (gdy $A < 0$). W tym ostatnim przypadku mówimy czasem, że sygnały X i Y są „**w przeciwfazie"**.

Duże modele dużych obiektów muszą być charakteryzowane przez większą liczbę parametrów. Zobaczymy to dokładniej w rozdziale 4, gdzie przytoczone są liczne przykłady modeli różnych obiektów i systemów biologicznych od najprostszych do takich właśnie bardziej złożonych, mających znaczną liczbę parametrów. Jednak żeby nie poprzestawać na gołosłownym zapewnieniu, że model obiektu może mieć wiele parametrów - pokażemy jeszcze tutaj prosty obiekt liniowy mający wiele parametrów na skutek tego, że ma wiele wejść

i wyjść (Rys. 2.17). Takie obiekty zdarzają się w praktyce i są nazywane obiektami **wielowymiarowymi**. Tu poprzestaniemy skromnie na obiekcie mającym zaledwie trzy wejścia i dwa wyjścia, ale bywają obiekty mające dziesiątki albo setki wejść i wyjść. Wejścia w naszym przykładowym obiekcie oznaczymy odpowiednio jako X_1, X_2 oraz X_3. Podobnie wyjścia tego obiektu oznaczymy odpowiednio jako Y_1 oraz Y_2.



Rys. 2.17. Przykładowy obiekt wielowymiarowy mający wiele parametrów

Założenie liniowości modelu obiektu wielowymiarowego jest równoznaczne z założeniem, że oddziaływanie każdego wejścia na każde wyjście ma oddzielny współczynnik wzmocnienia, zaś na wyjściach oddziaływania, pochodzące od poszczególnych wejść, są sumowane. Oznaczając przez A_{ij} wzmocnienie na drodze od wejścia o numerze j do wyjścia o numerze i otrzymujemy dla wyjść rozważanego obiektu równania:

$$Y_1 = A_{11} X_1 + A_{12} X_2 + A_{13} X_3 \tag{2.3}$$

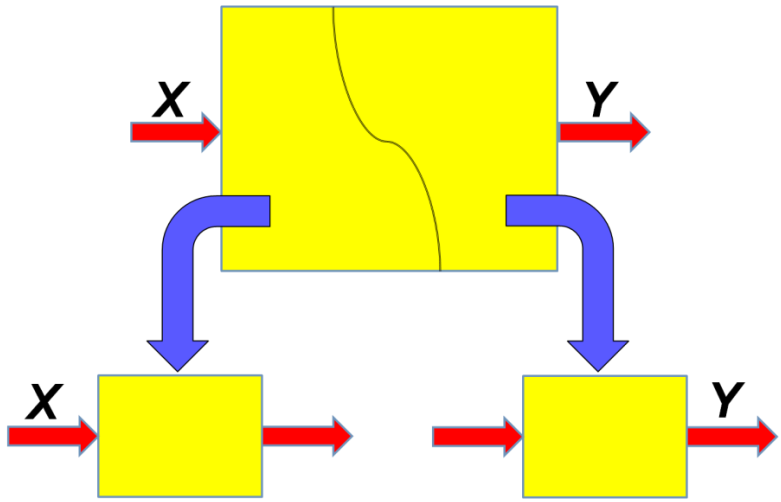
$$Y_2 = A_{21} X_1 + A_{22} X_2 + A_{23} X_3 \tag{2.4}$$

Równania te pokazują, jaką rolę odgrywają poszczególne parametry występujące w opisie modelu.

2.3. Łączenie modeli obiektów dla uzyskania modelu całego złożonego systemu

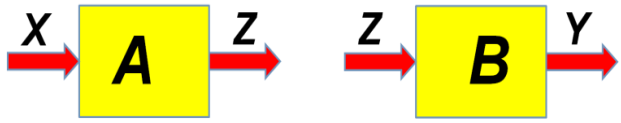
Przy tworzeniu modeli matematycznych skomplikowanych systemów bardzo użyteczna jest okoliczność, że taki złożony system możemy zazwyczaj podzielić na części, które można potraktować – każdą z osobna - jako oddzielne obiekty modelowania. Zwykle jest tak, że obiekty będące częściami modelowanego systemu, można znacznie łatwiej modelować, niż system jako całość. Dokonując kolejnych podziałów możemy więc dojść do w miarę prostych do zamodelowania kawałków będących częściami nawet bardzo skomplikowanego systemu. Wskazaną drogę postępowania możemy jednak zastosować wyłącznie wtedy, gdy potrafimy modele tych oddzielnych obiektów scalić do postaci całego systemu. I właśnie o tym traktuje niniejszy podrozdział: poznamy w nim sposoby łączenie modeli prostych elementów składowych (oddzielnych obiektów) w model całego złożonego systemu.

Sposób scalania modeli obiektów będących częściami skomplikowanego systemu w model większej całości zależy oczywiście od tego, jak te obiekty zostały wydzielone i jak w związku z tym wygląda schemat ich wzajemnych powiązań. Zacznijmy od najprostszego przykładu:



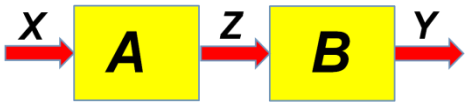
Rys. 2.18. Podział złożonego systemu na dwa układy celem ułatwienia modelowania

Jeśli umówimy się, że rozważany system został podzielony na dwie części (Rys. 2.18) i jeśli zamodelowaliśmy w sposób opisany w podrozdziale 2.2.4 te **dwa** obiekty, które powstały po podziale (Rys. 2.19)



Rys. 2.19. Oddzielne modele dwóch składowych złożonego systemu

to można rozważyć układ, który powstanie, gdy te obiekty zostaną z powrotem połączone ze sobą (Rys. 2.20).



Rys. 2.20. Dwa modelowane obiekty połączone ponownie w jeden system

Elementarne przekształcenia matematyczne pozwalają stwierdzić, co się wtedy stanie. Opis:

$$Z = A\,X \tag{2.5}$$

Opis modelu drugiego obiektu ma postać:

$$Y = B\,Z \tag{2.6}$$

Podstawienie wzoru (2.5) do (2.6) daje opis:

$$Y = B\,A\,X \tag{2.7}$$

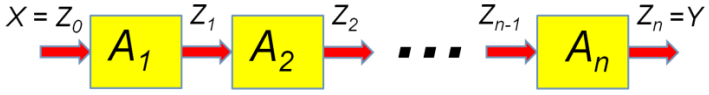
Okazuje się, że taki układ dwóch połączonych obiektów można zastąpić jednym obiektem o odpowiednio zmodyfikowanej funkcji przejścia (Rys. 2.21).

Rys. 2.21. Obiekt zastępczy o takich samych właściwościach jak system przedstawiony na rys. 2.20 – mogący być modelem systemu z rysunku 2.19.

Warto zauważyć, że obliczony współczynnik wzmocnienia w łącznej funkcji przejścia pokazanej na rysunku 2.21 zapisano w postaci *BA*, a nie bardziej naturalnej (jak by się wydawało) postaci *AB*. Nie jest to pomyłka ani nie jest to przypadek. *Z* formalnych przekształceń wynika niewątpliwie *BA* a nie *AB*. Gdyby parametry *A* i *B* traktować jako po prostu liczbowe wartości odpowiednich wzmocnień to kolejność mnożenia nie ma znaczenia i może być

użyty bardziej „przyjazny” zapis *AB*. Gdyby jednak natura modeli składowych była bardziej złożona, na przykład gdyby to były obiekty wielowymiarowe – wówczas w roli parametrów *A* i *B* występowałaby macierze – a mnożenie macierzy jest nieprzemienne i dla zachowania prawidłowej notacji trzeba by było zachować porządek *BA*.

Zasadę tworzenia modelu złożonego z wydzielonych i połączonych obiektów można także uogólnić i zastosować w odniesieniu do dowolnej liczby obiektów połączonych tak, że wyjście poprzedniego obiektu staje się wejściem obiektu następnego (Rys. 2.22). Taki sposób łączenia obiektów nazywa się **systemem szeregowym**.



Rys. 2.22. Tworzenie modelu systemu złożonego z dowolnej liczby szeregowo połączonych obiektów

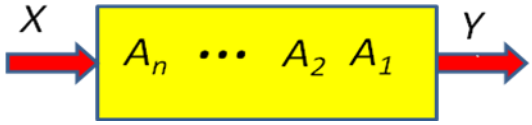
Każdy z obiektów składowych można opisać zależnością:

$$Z_i = A_i Z_{i-1} \quad \text{dla } i = 1, 2, \ldots, n \tag{2.8}$$

W oczywisty sposób uogólniając postępowanie zastosowane uprzednio do wzorów (2.5) i (2.6) otrzymujemy formułę:

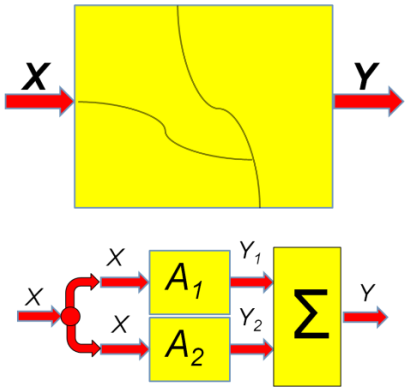
$$Y = A_n A_{n-1} \ldots A_2 A_1 X \tag{2.9}$$

pozwalającą na narysowanie schematu modelu całego systemu (Rys. 2.23).



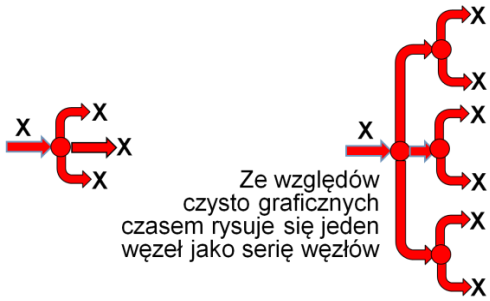
Rys. 2.23. Model systemu powstałego z wielu szeregowo połączonych obiektów

Obok połączeń szeregowych przy budowie systemów biocybernetycznych stosowane są **połączenia równoległe** (Rys. 2.24). W tym przypadku w schemacie zastępczym odpowiednie mnożniki (reprezentujące wzmocnienia zastępujące w modelu liniowym funkcje przejścia obiektów) są dodawane, a nie mnożone (Rys. 2.24).



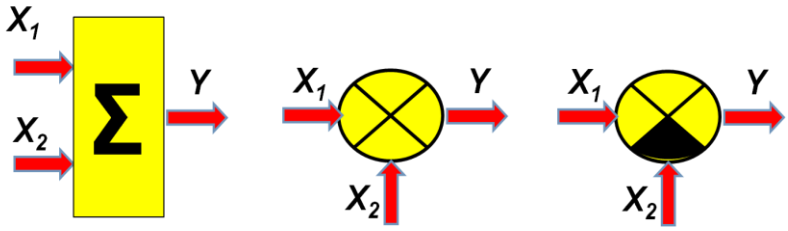
Rys. 2.24. Złożony system reprezentowany jako równoległe połączenie dwóch obiektów

Na rysunku 2.24 zastosowano symbol, którego jeszcze nie używaliśmy, więc należy go objaśnić. Chodziło o to, że sygnał wejściowy X jest dostarczany na wejście zarówno obiektu o wzmocnieniu A_1 jak i do obiektu A_2. Takie „rozmnożenie" sygnału jest zawsze możliwe, ponieważ informację można powielać bez żadnych ograniczeń. Dla oznaczenia takiego miejsca, w którym jeden sygnał jest pobierany do różnych odbiorników stosuje się tak zwany **węzeł zaczepowy**. Jego użycie w dwóch bardziej rozbudowanych strukturach powielania sygnałów przedstawia przykładowo rysunek 2.25.



Rys. 2.25. Przykłady węzłów zaczepowych służących do „rozmnażania" sygnałów dla różnych odbiorców

Obok węzła zaczepowego na schemacie przedstawionym na rysunku 2.24 użyto jeszcze jednego nietypowego oznaczenia, mianowicie sygnału z obiektów A_1 i A_2 są sumowane w bloku oznaczonym symbolem Σ. Dla potrzeb tego przykładu tak może zostać, ale generalnie warto wiedzieć, że w asortymencie symboli graficznych stosowanych przy modelowaniu obiektów jest specjalny symbol oznaczający operację sumowania sygnałów – tak zwany węzeł sumacyjny, przedstawiony wraz z komentarzem na rysunku 2.26.



Rys. 2.26. Węzeł sumacyjny. Kolejno od lewej do prawej pokazany jest blok używany do zaznaczania operacji sumowania sygnałów w tym podrozdziale (niezgodny z normami), typowy węzeł sumacyjny oraz specjalny typ węzła sumacyjnego, w którym sygnał X_2 jest odejmowany od sygnału X_1, a nie dodawany do niego

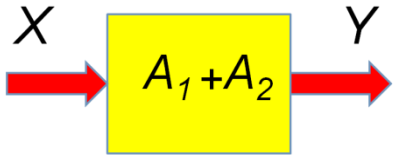
Wróćmy jednak do schematu przedstawionego na rysunku 2.24 i określmy, jak można taką strukturę „zwinąć" do pojedynczego bloku. W tym celu obliczmy zależności pomiędzy sygnałami X i Y. Odpowiednie przeliczenia wyglądają tak:

$$Y_1 = A_1 X \tag{2.10}$$

$$Y_2 = A_2 X \tag{2.11}$$

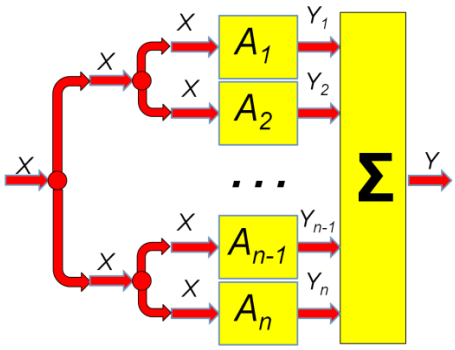
$$Y = Y_1 + Y_2 = A_1 X + A_2 X = (A_1 + A_2) X \tag{2.12}$$

Jak z tego wynika, blok zastępujący cały układ pokazany na rysunku 2.24 może być przedstawiony tak, jak na rysunku 2.27.

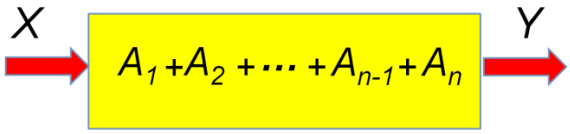


Rys. 2.27. Model całego systemu w którym występuje połączenie równoległe

Analogicznie tworzyć można schematy strukturalne oraz modele całości dla systemów złożonych z większej liczby równolegle sprzężonych obiektów, co przedstawiono na rysunkach 2.28 i 2.29. Warto zauważyć, że w tym przypadku wartości wzmocnień poszczególnych obiektów można było przy sumowaniu przedstawiać w naturalnej kolejności od A_1 do A_n, ponieważ sumowanie (w odróżnieniu od mnożenia) jest zawsze przemienne.

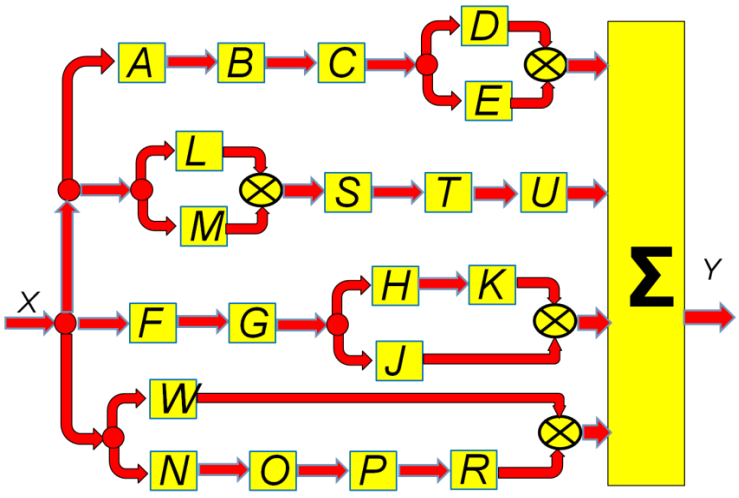


Rys. 2.28. Większy system o strukturze równoległej



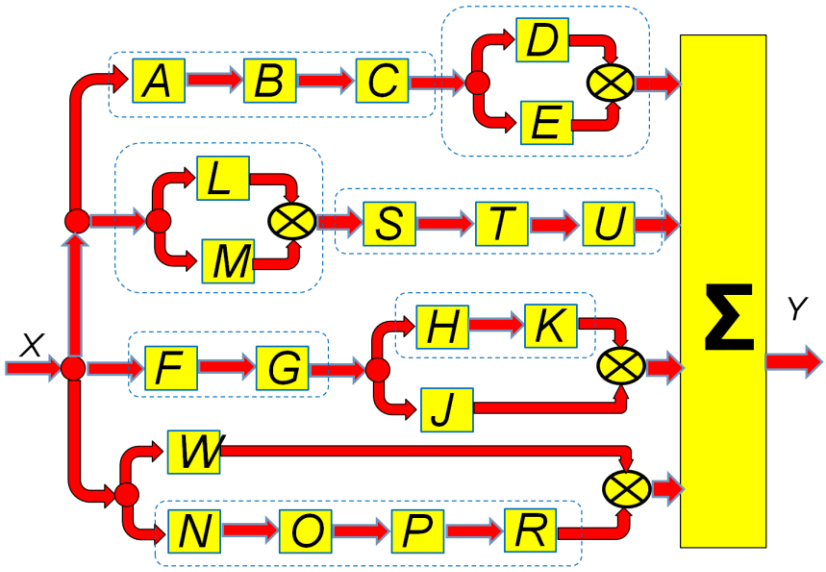
Rys. 2.29. Zagregowany schemat zastępczy systemu z rysunku 2.28

Przedstawiając struktury złożonych systemów jako kombinacje obiektów połączonych ze sobą w formie sprzężeń szeregowych i równoległych możemy tworzyć modele nawet bardzo skomplikowanych systemów. Przykład takiego skomplikowanego systemu przedstawia rysunek 2.30, a na rysunkach 2.31 do 2.33 pokazane są kolejne etapy „zwijania" tego schematu do postaci zagregowanego schematu zastępczego.

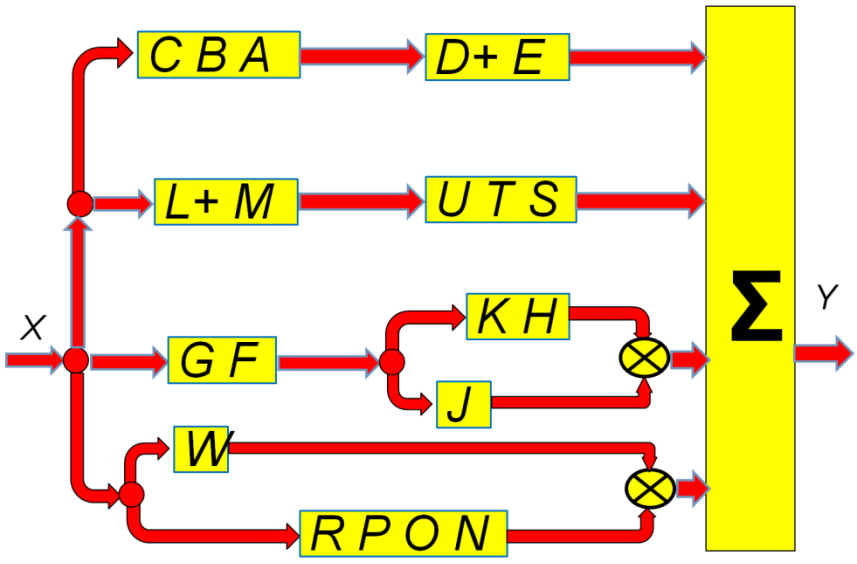


Rys. 2.30. Przykładowy złożony system przeznaczony do „zwijania"

Pierwszym krokiem w procesie „zwijania” jest znalezienie systemów zastępczych dla połączeń obiektów, które są typowymi układami o połączeniach szeregowych lub równoległych. Na rysunku 2.31 pokazano te połączenia obiektów i zaznaczono je ramkami złożonymi z linii przerywanych.

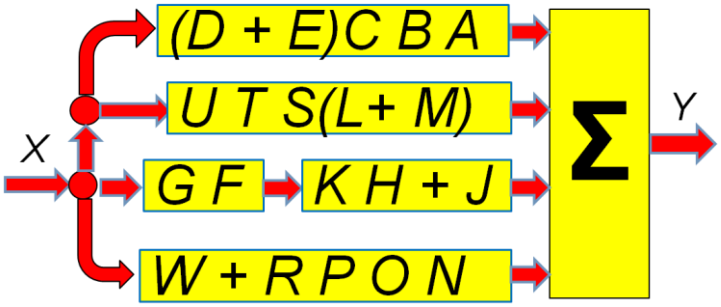


Rys. 2.31. Pierwszy etap „zwijania” – wydzielenie układów szeregowych i równoległych nadających się do redukcji



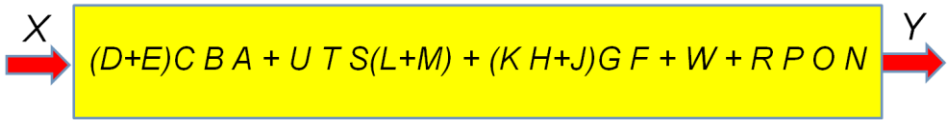
Rys. 2.32. System po pierwszej serii redukcji

Proces redukcji w taki sam sposób kontynuujemy dalej otrzymując już stosunkowo prosty model (Rys. 2.33).



Rys. 2.33. Przedostatnia faza „zwijania" systemu

Z tego prostego modelu bardzo łatwo jest już uzyskać syntetyczny model całego systemu (Rys. 2.34).



Rys. 2.34. Ostateczna forma syntetycznego modelu całego systemu

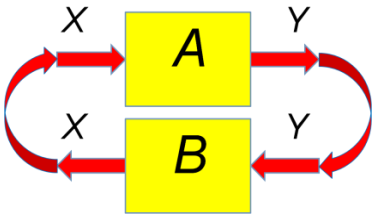
Wypisanie formuły tego modelu „z marszu", dla całego złożonego systemu, byłoby przypuszczalnie bardzo trudne – nie mówiąc o identyfikacji tak złożonego modelu. A tymczasem dzieląc system na poszczególne obiekty, a potem „zwijając" go zgodnie z podanymi wyżej regułami można taki skomplikowany model uzyskać w miarę łatwo. Nie bez znaczenia jest też fakt, że badanie poprawności zbudowania i dokładności działania modeli poszczególnych układów jako elementów składowych systemu jest nieporównanie łatwiejsze i bezpieczniejsze, niż analiza działania całego systemu.

2.4. Uwzględnienie w modelu sprzężeń zwrotnych i sygnałów zmiennych w czasie

2.4.1. Ogólny schemat systemu ze sprzężeniem zwrotnym

Oglądając sekwencje rysunków od 2.30 do 2.34 możemy odnieść wrażenie, że oto dzięki opisanej i zastosowanej metodzie dla każdego napotkanego systemu potrafimy stworzyć jego model. Możemy tego dokonać dzieląc najpierw ten system na części, następnie identyfikując obiekty będące częściami systemu, a potem scalając wyniki. Niestety, nie jest tak dobrze, bo budując model i analizując strukturę modelowanego systemu możemy często napotkać

na strukturę przedstawioną (w najprostszym możliwym wariancie) na rysunku 2.35.



Rys. 2.35. Najprostsza struktura systemu ze sprzężeniem zwrotnym

Jest to struktura systemu ze **sprzężeniem zwrotnym**. Wnosi ona nową jakość do prowadzonych tu rozważań i dlatego strukturze tej przyjrzymy się z należytą uwagą.

Próba opisu tego, co się dzieje w takim systemie, metodami analogicznymi do tych, jakie stosowaliśmy przy analizie systemów złożonych z obiektów łączonych szeregowo albo równolegle, prowadzi do sprzeczności. Mamy bowiem dwie zależności:

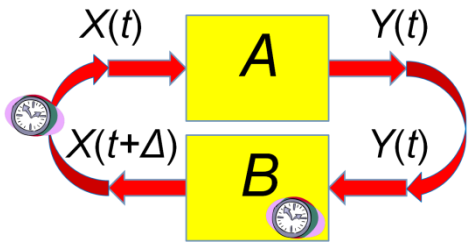
$$Y = A\,X \tag{2.13}$$

$$X = B\,Y \tag{2.14}$$

W ogólnym przypadku te dwie równości nie mogą być równocześnie prawdziwe, a przypadek szczególny, gdy $B = 1/A$ jest trywialny, bo wtedy dla dowolnego X $Y = A\,X$ - i ten stan trwa bez żadnych zmian dowolnie długo.

Żeby zobaczyć, do czego naprawdę zdolny jest system ze sprzężeniem zwrotnym – musimy wprowadzić sygnały zmienne w czasie. Przy wszystkich poprzednich obiektach i systemach, które rozważaliśmy w tej książce, istotne było pytanie: *Jaką wartość ma wejściowy sygnał X?* albo *Jaką wartość przybierze w związku z tym wyjściowy sygnał Y?* natomiast nie zastanawialiśmy się nad pytaniem *Kiedy sygnał na wejściu obiektu ma wartość X?* albo nad kwestią *Po jakim czasie sygnał na wyjściu obiektu przyjmie wartość Y?* - bo te kwestie nie podlegały dyskusji. Czas nie był ważny w naszych rozważaniach.

Teraz musimy to zmienić. Będziemy zamiast stałych i niezmiennych sygnałów X i Y rozważali sygnały zmieniające się w czasie, co odnotujemy pisząc funkcje $X(t)$ oraz $Y(t)$. Pozwoli to nam rozwiązać problem związany z pozorną sprzecznością zapisów (2.13) i (2.14) opisujących system pokazany na rysunku 2.35. Jeśli założymy, że w jednym z obiektów narysowanych na tym rysunku zachodzi nie tylko przekształcenie sygnału X na Y (lub odwrotnie), ale dodatkowo sygnał na wyjściu takiego obiektu pojawia się z pewnym **opóźnieniem** – to schemat rozważanego systemu ze sprzężeniem zwrotnym będzie wyglądał tak, jak to pokazano na rysunku 2.36.



Rys. 2.36. Schemat układu ze sprzężeniem zwrotnym w którym sygnały X i Y są funkcjami czasu, a w obiekcie B występuje opóźnienie sygnału

Warto zwrócić uwagę, że na tym schemacie, odmiennie od wszystkich wcześniej rysowanych, rozważane sygnały są oznaczone jako funkcje czasu $X(t)$ oraz $Y(t)$. Dla zaznaczenia, że w obiekcie B występuje opóźnienie sygnału narysowano dodatkowo w bloku reprezentującym ten obiekt symbol zegarka. Taki sam symbol umieszczony jest na drodze między wyjściem układu B i wejściem układu A żeby wyeliminować paradoks polegający na tym, że na tej samej linii przesyłającej sygnały w jednym miejscu jest inny sygnał $X(t)$, a w innym miejscu jest sygnał $X(t + \Delta)$. Po prostu te dwa sygnały występują na tej samej linii, ale w dwóch różnych chwilach czasowych – odpowiednio t oraz $t + \Delta$. Nie jest to oznaczenie standardowe i nie jest zapewne używane w podobnych pracach innych autorów. W porządnych pracach z zakresu automatyki używane jest w tym miejscu niekiedy oznaczenie z^{-1} nawiązujące do tak zwanej transformacji Z – ale nie ma tu miejsca ani potrzeby, żeby tę skomplikowaną transformację operatorową odwołującą się do liczb zespolonych dokładnie przedstawiać i dyskutować. Po prostu na użytek tego skryptu możemy się umówić, że w taki właśnie sposób (symbolem zegarka) będziemy zaznaczali opóźnienie – i tyle.

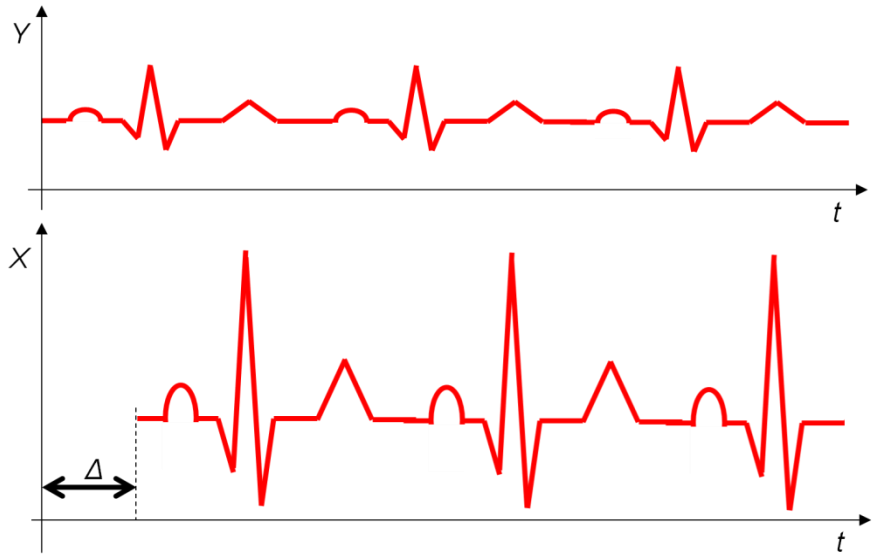
Możemy natomiast zapisać ogólne równania opisujące zachowanie systemu przedstawionego na rysunku 2.36. Dla pierwszego obiektu zapis jest elementarny:

$$Y(t) = A\,X(t) \tag{2.15}$$

Dla drugiego opis jest odrobinę bardziej skomplikowany:

$$X(t + \Delta) = B\,Y(t) \tag{2.16}$$

Wyjaśnijmy jeszcze dokładnie, co oznacza fakt opóźnienia sygnału. Popatrzmy na rysunek 2.37.



Rys. 2.37. Sygnały z rysunku 2.36. Widoczne jest wzmocnienie oraz opóźnienie sygnału *X* w stosunku do *Y*

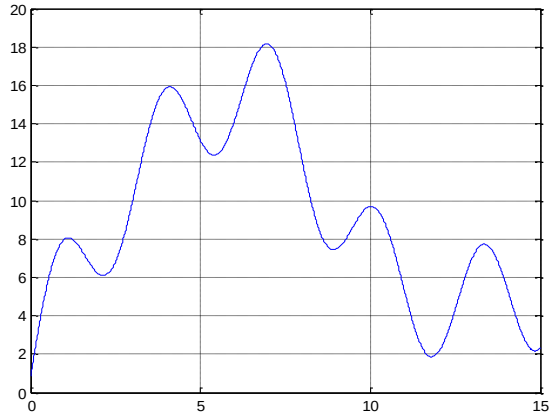
Pokazano na nim w górnej części pewien sygnał, który jest funkcją czasu *t*. Sygnał ten przypomina sygnał EKG, ale to w tym momencie nieistotne. Górny wykres opisano na osi pionowej symbolem *Y* nawiązując w ten sposób do oznaczeń użytych na rysunku 2.36. Sygnał *Y* wchodzi (jako sygnał wejściowy) do bloku symbolizującego dolny obiekt (tego, którego współczynnik wzmocnienia oznaczono *B*) i podlega tam dwóm procesom: Po pierwsze zostaje wzmocniony (zakładamy $B > 1$) oraz po drugie – ulega opóźnieniu o odcinek czasu oznaczony jako Δ. Odpowiednie wartości sygnału wyjściowego *X* (wychodzącego z obiektu *B*) pojawiają się nie w momencie, w którym na wejście obiektu podawane są odpowiednie wartości sygnału *Y* – lecz **później.**

W dalszych rozważaniach będziemy funkcje *X(t)* oraz *Y(t)* rozważali jako funkcje **dyskretne**. Tak będzie o wiele wygodniej, a ponadto taki sposób odwzorowania rzeczywistych sygnałów będzie wręcz konieczny w przypadku, gdy od ogólnego modelowania matematycznego rozważanych obiektów przejdziemy do ich komputerowej symulacji. Nasuwa się tu ogólniejsze stwierdzenie, które warto przytoczyć. Otóż w kontekście wszelkich obliczeń komputerowych odnoszących się do jakichkolwiek sygnałów pochodzących z rzeczywistego świata - jednym z podstawowych wymagań, jakie trzeba spełnić, jest używanie wszystkich sygnałów właśnie w postaci **dyskretnej.** *Dyskretność* ta nie powinna być jednak kojarzona z *tajnością,* tylko powinna być traktowana jako przeciwieństwo **ciągłości**. Zatrzymajmy się nad tym szczegółem, bo ma on duże znaczenie we wszystkich dalszych rozważaniach.

2.4.2. Sygnały ciągłe i dyskretne

Rzeczywiste sygnały, z którymi mamy do czynienia w obiektach należących do rzeczywistego świata, z reguły są **ciągłe**. Oznacza to, że wartość takiego sygnału jest określona dla każdej chwili czasowej (jeśli jest to sygnał zmienny w czasie) a także oznacza, że wartość sygnału może być dowolna. Co więcej, ciągłość zakłada, że potencjalnie możliwy jest pomiar sygnału o nieograniczonej dokładności. Dzięki temu dwa sygnały różniące się o najmniejszą nawet wartość będą mogły być zawsze rozpoznane jako **różne**. Praktyka pokazuje, że ta ostatnia własność ma charakter jedynie potencjalny, ponieważ nie istnieją przyrządy pomiarowe dostarczające informacji o wartości sygnału z nieograniczoną dokładnością, a ponadto sygnały ciągłe łatwo ulegają zniekształceniom pod wpływem różnych czynników zwanych **zakłóceniami.** Z tego powodu sygnały dyskretne (zwane także sygnałami cyfrowymi) chętnie stosuje się do różnych zastosowań nie mających wcale związku z techniką komputerową, bo sygnały dyskretne są o wiele bardziej odporne na zakłócenia. Wystarczy porównać dźwięk z tradycyjnego gramofonu (takiego z winylowymi płytami), gdzie muzyka zapisana jest w formie analogowej, z dźwiękiem odtwarzanym przez nowoczesny *iPad* (gdzie oczywiście używane są sygnały dyskretne), żeby przekonać się, jak ważna jest ta różnica.

Rozważmy to zagadnienie dokładniej. Przykład sygnału **ciągłego** pokazano na rysunku 2.38, chociaż próbki sygnału ciągłego pojawiały się już w tej książce wcześniej – między innymi na rysunku 2.37.



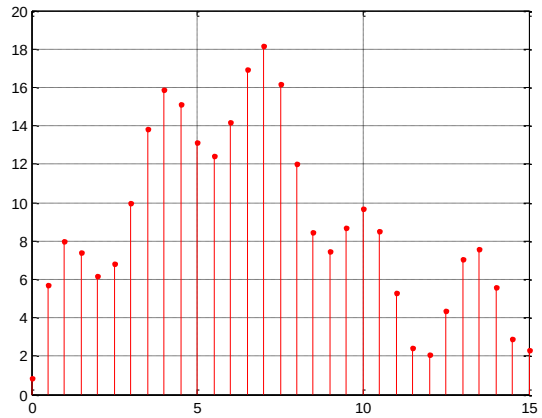
Rys. 2.38. Przykład sygnału ciągłego

Tak, jak powiedziano wyżej, ciągła linia reprezentująca sygnał **ciągły** na prezentowanym rysunku podaje jego wartość w każdej chwili czasu oraz wartość ta może być absolutnie dowolna.

W odróżnieniu od tego sygnał **dyskretny**, jakim może operować komputer na przykład podczas symulacji, ma ustalone wartości jedynie dla **niektórych**

chwil czasu. Wynika to z faktu, że do przechowywania sygnału w pamięci komputera przeznacza się pewien skończony obszar. Obszar ten zawiera pewną liczbę miejsc (grup bajtów) przeznaczonych do tego, żeby w nich zapamiętać wartości sygnału. Taki obszar może być bardzo duży, ale zawsze zmieści się w nim tylko **skończona** liczba wartości sygnału, podczas gdy w sygnale ciągłym wartości jest nieskończenie wiele, bo są one określone dla **każdej** chwili czasowej – a tych chwil czasowych jest przecież nieskończona ilość[6]. W związku z tym trzeba wybrać pewną skończoną liczbę chwil czasowych i tylko w tych chwilach rejestrować i rozważać wartości sygnału, nazywane wtedy **próbkami** sygnału. Tych próbek może być dużo, ale zawsze jest ich skończona liczba.

W dodatku wartości tych próbek nie mogą być całkiem dowolne, tylko muszą dać się odwzorować w strukturach elektronicznych komputera. Pamięć komputera, jego procesor, a także urządzenia łączności cyfrowej (na przykład systemy internetowe) dopuszczają jedynie informacje w formie serii **bitów** (zer albo jedynek). Tych bitów może być dużo, ale zawsze jest ich skończona liczba, a to oznacza, że dopuszczalne są tylko **niektóre** wartości. Przykład sygnału dyskretnego przedstawiono na rysunku 2.39.

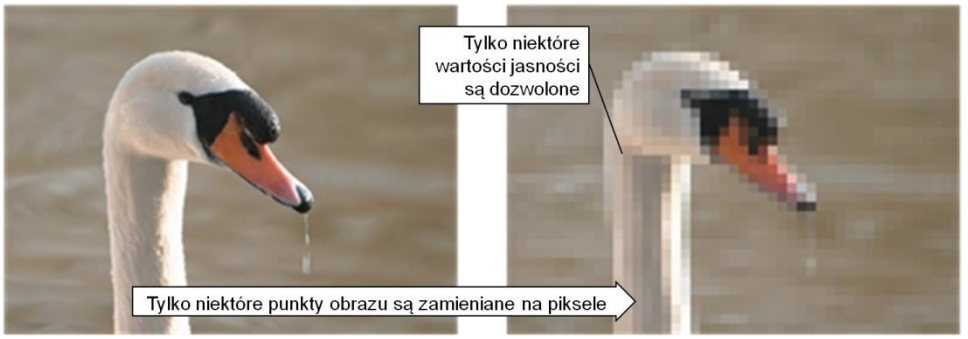


Rys. 2.39. Sygnał dyskretny (reprezentuje ten sam przebieg co na rysunku 2.38)

Zapamiętajmy i bierzmy pod uwagę we wszystkich dalszych rozważaniach: w systemie komputerowym sygnał jest zawsze reprezentowany w formie **próbkowanej** (to znaczy jego wartości są podawane tylko w niektórych punktach, dla wybranych chwil czasowych) oraz **skwantowanej** (czyli jego

[6] Pomiędzy dwoma dowolnymi chwilami czasowymi można w sygnale ciągłym wyróżnić jeszcze jedną chwilę czasową mieszczącą się pomiędzy nimi, a potem jeszcze jedną i jeszcze jedną – w nieskończoność.

wartość może przyjmować wyłącznie niektóre, z góry zadane wartości). Ilustruje to rysunek 2.40, na którym pokazano po lewej stronie obraz (to także rodzaj sygnału!) ciągły, a po prawej stronie obraz dyskretny.

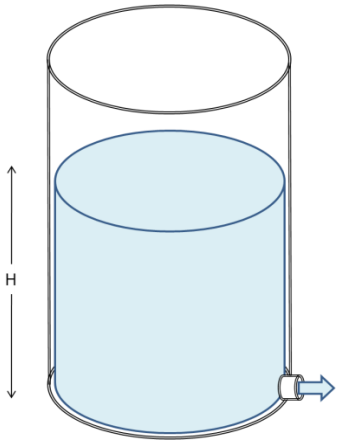


Rys. 2.40. Porównanie ciągłej i dyskretnej reprezentacji sygnału na przykładzie obrazu

Z oglądania rysunku 2.40 nie należy wyciągać wniosku, że sygnał w postaci dyskretnej (cyfrowej) jest gorszy od sygnału analogowego. Zniekształcenia spowodowane próbkowaniem i kwantowaniem sygnału można uczynić dowolnie małymi, a ze względu na odporność na zakłócenia, łatwość przechowywania, przesyłania, kopiowania, wyszukiwania i komputerowego przetwarzania sygnałów dyskretnych mają one zdecydowaną przewagę nad sygnałami ciągłymi. Niemniej używając sygnałów dyskretnych trzeba mieć świadomość, czym one w istocie są – i trzeba z tego wyciągać właściwe wnioski – między innymi w kontekście modelowania systemów i ich komputerowej symulacji.

2.4.3. Dyskretyzacja opisu obiektu dla potrzeb symulacji

Dyskretny charakter sygnałów w modelu symulacyjnym w niektórych przypadkach może pozostawać w sprzeczności z ciągłą naturą świata, w którym żyjemy. Rozważmy bardzo prosty przykład systemu, który chcielibyśmy zamodelować. Wyobraźmy sobie naczynie z dziurką przy dnie, napełnione wodą (lub innym płynem) – Rys. 2.41.



Rys. 2.41. Obiekt na przykładzie którego dyskutowana jest różnica między ciągłą rzeczywistością i dyskretnym modelem

Początkowo dziurka jest zatkana i poziom wody w naczyniu wynosi H_0. Gdy dziurka zostanie odetkana woda zacznie wypływać i jej poziom w naczyniu będzie opadał. Ponieważ woda wypływa pod wpływem ciśnienia wywieranego przez słup wody w naczyniu – szybkość jej wypływania jest proporcjonalna do tego ciśnienia, a ciśnienie jest proporcjonalne do poziomu wody *H*.

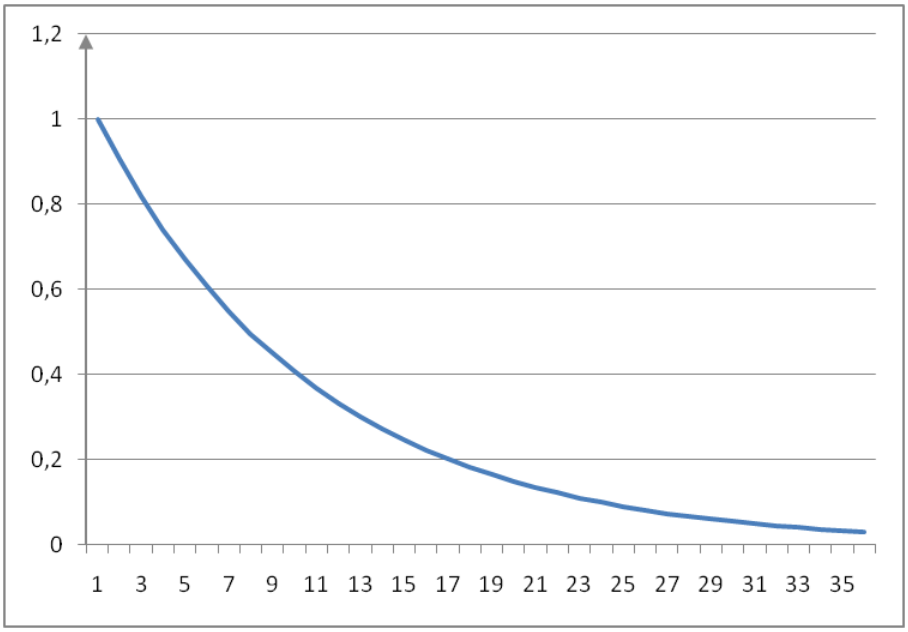
Wypływanie wody powoduje, że poziom *H* maleje. Szybkość tego malenia, wyrażana pochodną *dH/dt*, jest proporcjonalna do *H*. Można więc napisać równanie:

$$\frac{dH}{dt} = -\,k\,H \tag{2.17}$$

wyrażające tę zależność, w którym *k* oznacza współczynnik szybkości wypływu wody (jest tym większe, im większy jest otwór, przez który woda wypływa). Rozwiązanie równania (2.15), pokazujące jak poziom wody w naczyniu będzie się zmieniał w czasie, jest znane i ma postać

$$H(t) = H_0\, e^{-kt} \tag{2.18}$$

Jest to funkcja ciągła, której wykres pokazuje rysunek 2.42. Przebieg tej funkcji przedstawia wiernie sytuację znaną z codziennego doświadczenia: poziom w naczyniu opada, przy czym spadek ten jest początkowo szybki, a potem w miarę ubywania wody – coraz mniejszy.



Rys. 2.42. Przebieg rzeczywistego procesu w obiekcie z rysunku 2.41 jest z samej swojej natury ciągły

Przebieg rzeczywistego procesu jest z samej swojej natury ciągły, to znaczy w każdej chwili czasowej t możliwe jest podanie konkretnej wartości poziomu wody $H(t)$.

Gdybyśmy jednak chcieli opisać ten proces w formie programu symulacyjnego (żeby komputer symulował wypływ wody) to nie wolno by nam było odwołać się do jakiegokolwiek procesu ciągłego – wszystko musiałoby się odbywać w sposób dyskretny. Konieczne więc byłoby wprowadzenie **dyskretnej skali czasu**. Zamiast zmiennej ciągłej t reprezentującej czas w fizyce musimy wprowadzić dyskretne numery chwil czasowych n, odległych od siebie o pewien skończony **krok czasowy** (odstęp czasu) Δ, i dyskretne wartości interesującej nas zmiennej (poziomu wody) $H(n)$ w tych właśnie branych pod uwagę chwilach czasowych. Ustalamy, że rozważać będziemy chwile czasowe z pewnego ustalonego przedziału: $n = 1, 2, ..., N$. Dodatkowo umawiamy się, że znak „=” w zapisie algorytmu nie oznacza równości w sensie matematycznym, tylko nakazuje obliczenie wartości występującej po prawej stronie tego znaku i nadanie tej wartości zmiennej znajdującej się po lewej jego stronie. Schemat takiej czynności (ogólnie znanej w informatyce jako tzw. instrukcja podstawienia) jest następujący:

```
zmienna_której_będzie_nadana_wartość = formuła_obliczająca_wartość
```
(2.19)

Warto uważnie przyjrzeć się podanemu wyżej schematowi, bo jego dokładne zrozumienie ustrzeże nas przed zdziwieniem towarzyszącym oglądaniu zapisu:

$$n = n + 1 \tag{2.20}$$

Gdyby powyższy zapis traktować jako równanie matematyczne - to byłaby to **sprzeczność**, bo niezależnie od tego, co byśmy chcieli podstawić jako aktualną wartość zmiennej *n*, nigdy nie uda się sprawić, by owa wartość była równa … samej sobie powiększonej o jeden!

Jeśli jednak uwzględnimy interpretację podaną formułą (2.19) to zapis podany we wzorze (2.20) staje się jasny: po prostu należy wziąć poprzednią (dotychczasową) wartość zmiennej *n* i zwiększyć ją o 1, zaś to, co wyjdzie, należy dalej wykorzystywać jako nową wartość zmiennej *n*.

Po tych wyjaśnieniach możemy opisać algorytm, który zapewni symulację rozważanego zjawiska (wypływu wody z naczynia):

1. Ustaw $n = 1$ oraz $H(n) = H(1) = H_0$
2. Oblicz spadek poziomu wody $U(n)$, spowodowany tą wodą, która wypłynie z naczynia w ciągu odstępu czasu Δ pomiędzy chwilą czasową n a chwilą czasową $n+1$. Ten ubytek $U(n)$ możemy wyliczyć ze wzoru:

$$U(n) = H(n)\, k\, \Delta$$

3. Oblicz poziom wody w chwili n+1 odejmując od poziomu $H(n)$ spadek poziomu $U(n)$, jaki nastąpił pomiędzy chwilą czasową n a chwilą czasową $n+1$. Korzystamy przy tym ze wzoru:

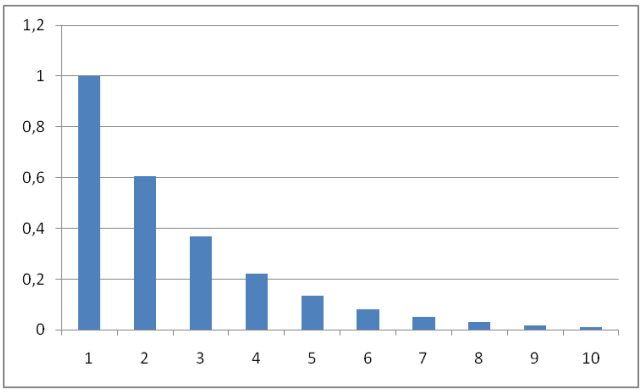
$$H(n+1) = H(n) - U(n)$$

4. Zmieniamy numer rozważanej chwili czasowej, to znaczy wartość zmiennej n zwiększamy o jeden:

$$n = n + 1$$

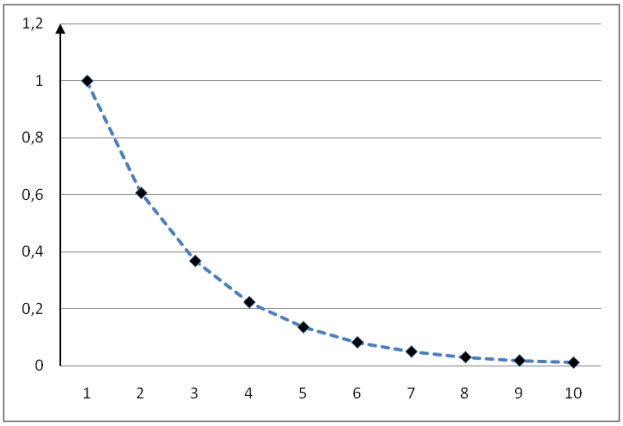
5. Jeśli $n < N$ to powtarzamy powyższe czynności poczynając od punktu 2.
6. Do tego punktu docieramy, gdy n osiągnie założoną wartość N. Oznacza to zakończenie obliczeń i zatrzymanie algorytmu.

W następstwie wykonania tego algorytmu możemy uzyskać serię **dyskretnych** wartości, podających poziom wody w naczyniu w wybranych dyskretnych momentach czasu (Rys. 2.43), co jednak różni się dosyć istotnie od przebiegu ciągłego ilustrującego rzeczywisty przebieg zjawiska (por. rys. 2.42).



Rys. 2.43. Zbiór dyskretnych wartości przybliżających – z wykorzystaniem algorytmu - przebieg procesu opróżniania naczynia

Dyskretny charakter algorytmu nie musi być istotną przeszkodą przy uzyskiwaniu użytecznych praktycznie wyników. Interpolując dyskretne punkty uzyskane przy pomocy algorytmicznej symulacji możemy uzyskać przebieg (pokazany linią przerywaną na rysunku 2.44), który całkiem dobrze aproksymuje ciągłą rzeczywistość. Niemniej to fundamentalne rozróżnienie pomiędzy ciągłym (Rys. 2.42) i dyskretnym (Rys. 2.43) obrazem tych samych zjawisk i procesów musi towarzyszyć nam przy każdej próbie symulacji komputerowej.

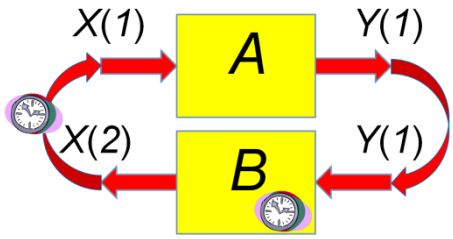


Rys. 2.44. Odtwarzanie aproksymacji przebiegu ciągłego z serii dyskretnych wartości za pomocą interpolacji

2.4.4. Charakterystyka procesów w systemie ze sprzężeniem zwrotnym dodatnim i ujemnym

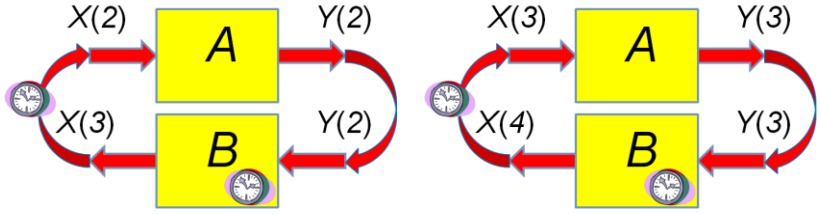
Po dygresji, jaką była przeprowadzona wyżej dyskusja na temat dyskretnego charakteru sygnałów dynamicznych (czyli będących funkcjami czasu), wrócimy

do rysunku 2.36 i do modelowania procesów toczących się w systemie ze sprzężeniem zwrotnym. Uwzględniając dyskusję przeprowadzoną wyżej możemy rozważyć, jak będzie wyglądał układ dyskretnych sygnałów w systemie ze sprzężeniem zwrotnym w kolejnych dyskretnych chwilach czasowych. Numerując te chwile i przyjmując jako **krok czasowy** dyskretyzacji wartość opóźnienia Δ występującego w układzie B możemy sytuację początkową w układzie ze sprzężeniem zwrotnym przedstawić tak, jak to pokazano na rysunku 2.45.



Rys. 2.45. Początek funkcjonowania systemu ze sprzężeniem zwrotnym

Kolejne dwa kroki procesu działania systemu ze sprzężeniem zwrotnym przedstawia rysunek 2.46. Dalszego ciągu łatwo się domyślić.



Rys. 2.46. Dalszy ciąg funkcjonowania systemu ze sprzężeniem zwrotnym

Działanie omawianego systemu można teraz opisać matematycznie. Dla pierwszego kroku mamy:

$$Y(1) = A\,X(1) \tag{2.21}$$

$$X(2) = B\,Y(1) = B\,A\,X(1) \tag{2.22}$$

Drugi krok obliczamy analogicznie:

$$Y(2) = A\,X(2) \tag{2.23}$$

$$X(3) = B\,Y(2) = (B\,A)^2\,X(1) \tag{2.24}$$

Moglibyśmy bez trudu opisać co dzieje się w trzecim, czwartym i w dalszych krokach, ale łatwa do wykrycia regularność w strukturze wzorów (2.22) i (2.24) pozwala wypisać od razu ogólną formułę, pozwalającą wyznaczyć

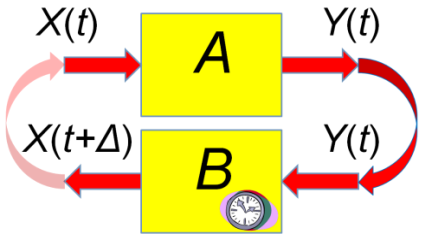
sygnał *X(n)* pojawiający się w systemie na *n*-tym kroku jego działania. Taka ogólna formuła ma postać:

$$X(n) = (B\,A)^n\,X(1) \qquad (2.25)$$

Jak widać zachowanie systemu ze sprzężeniem zwrotnym determinowane jest przez dwa czynniki: iloczyn *B A* (nazwiemy go funkcją przejścia systemu otwartego) oraz wartość sygnału *X*, która na początku obliczeń oznaczana jest jako *X(1)* (wartość tę określimy jako **warunek początkowy**).

Zajmiemy się teraz tymi dwoma czynnikami rozpatrując wpływ, jaki mają one na zachowanie rozważanego systemu.

Wyjaśnijmy może najpierw, dlaczego iloczyn *B A* nazywamy funkcją przejścia systemu otwartego. Popatrzmy na rysunek 2.47. Widać na nim, że w momencie gdy sprzężenie zwrotne „otworzymy" (usuwając połączenie wiążące obiekt *B* z obiektem *A*) – to powstaje znany nam system dwóch połączonych szeregowo obiektów, którego łączna funkcja przejścia wyraża się właśnie wzorem *B A*.



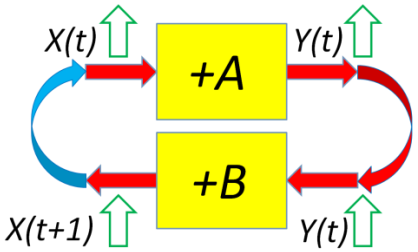
Rys. 2.47. Funkcja przejścia układu otwartego

Co możemy wywnioskować analizując funkcję przejścia systemu otwartego?

W pierwszej kolejności możemy rozróżnić dwa rodzaje systemów: **ze sprzężeniem zwrotnym dodatnim i ujemnym**.

System ze sprzężeniem zwrotnym dodatnim mamy wtedy, gdy $B\,A > 0$. Taki wynik możemy otrzymać, gdy $A > 0$ i równocześnie także $B > 0$. Zauważmy, że przy założeniu[7] $|BA| > 1$ w takim systemie każda zmiana dowolnego sygnału będzie miała tendencję do pogłębiania się. Na przykład jeśli z jakichś powodów w pewnym momencie *X(t)* wzrośnie – to na skutek tego, że $A > 0$ wzrośnie także *Y(t)*, zaś ten wzrost wymusi zwiększenie $X(t + \Delta)$ po okresie opóźnienia – i koło się zamyka, bo w następstwie także $Y(t + \Delta)$ wzrośnie itd. Ilustruje to rysunek 4.28, gdzie pustymi strzałkami zaznaczono zmiany wartości odpowiednich sygnałów.

[7] Znaczenie tego założenia zostanie przedyskutowane dalej.



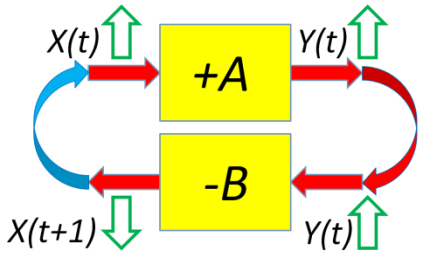
Rys. 2.48. Krąg zmian sygnałów w systemie z dodatnim sprzężeniem zwrotnym

Czytelnik może łatwo sprawdzić, że efekt samorzutnego pogłębiania się zmian zadziała w rozważanym systemie również wtedy, gdy początkową odchyłką będzie **zmalenie** (a nie **wzrost**) jednego z rozważanych sygnałów, a także w przypadku, kiedy efekt $B\,A > 0$ będzie uzyskana dla $A < 0$ oraz $B < 0$.

Sprzężenie zwrotne dodanie jest potencjalnie destrukcyjne – nawet mała zmiana jednego z rozważanych sygnałów (na przykład wynikająca z tego, że na jeden z obiektów zadziałało zakłócenie) doprowadzi do tego, że zmiana będzie się powiększała i pogłębiała – aż wreszcie doprowadzi to do katastrofy. Tak najczęściej jest w rzeczywistości i naprawdę wiele katastrof różnych systemów zostało spowodowane tym, że ujawniło się w nich dodatnie sprzężenie zwrotne. Na przykład rozwój raka jest przykładem systemu ze sprzężeniem zwrotnym dodatnim. Im więcej komórek zaatakowanego przez nowotwór narządu przechodzi do fazy komórek proliferujących (patrz podrozdział 4.7.2) – tym więcej takich zmienionych rakowo komórek może się ujawnić w kolejnym okresie czasu. W efekcie rak rośnie coraz szybciej, nacieka sąsiednie (zdrowe) tkanki i wreszcie prowadzi do śmierci zaatakowanego narządu i całego organizmu. Jednak tak być nie musi. Można sprawić, żeby system z dodatnim sprzężeniem zwrotnym pozostawał w równowadze – o czym przekonamy się w kolejnym podrozdziale.

Zanim to jednak nastąpi rozważmy przypadek sprzężenia zwrotnego **ujemnego.**

W systemie cechującym się takim typem sprzężenia mamy oczywiście $B\,A < 0$, co w rozważanym systemie będzie uzyskane dla $A > 0$ oraz $B < 0$. Łatwo prześledzić (na przykład na rysunku 2.49), że gdy w takim systemie zaistnieje jakaś **odchyłka** jednego z sygnałów (na przykład wzrost $X\,(t)$) – to system będzie dążył do jej zlikwidowania (czyli pojawi się tendencja do obniżenia $X\,(t)$).



Rys. 2.49. Przebieg zmian wartości sygnałów w systemie ze sprzężeniem zwrotnym ujemnym. Oznaczenia $+A$ oraz $-B$ użyto wyłącznie w tym celu, żeby podkreślić iż A jest dodatnie a B ujemne.

W przeciwieństwie do grożącego katastrofą sprzężenia dodatniego – sprzężenie ujemne sprawia wrażenie czynnika sprzyjającego stabilizacji zachowania systemu. Jest to wrażenie trafne, gdyż większość systemów, w których wbrew działającym zakłóceniom udaje się zachować stałą wartość jakiegoś istotnego parametru – oparta jest na działaniu ujemnego sprzężenia zwrotnego.

Przykładem bardzo dobrze działającego systemu z ujemnym sprzężeniem zwrotnym jest system termoregulacji organizmu, dzięki któremu ciało człowieka zachowuje stałą temperaturę (36,6 ^{o}C) mimo przebywania zarówno w środowiskach bardzo gorących jak i bardzo chłodnych.

Gdy człowiek znajdzie się w bardzo gorącym miejscu (na przykład na pustyni lub przy hutniczym piecu) temperatura jego ciała także ma tendencje do wzrastania, co w skrajnym przypadku może być niebezpieczne dla życia lub zdrowia. Ale w tym momencie ujemne sprzężenie zwrotne stabilizujące ciepłotę ciała człowieka (jako jeden z mechanizmów tak zwanej homeostazy organizmu), powoduje zwiększenie przepływu krwi w warstwie podskórnej (co się objawia w tym, że na przykład twarz czerwienieje) oraz skóra zaczyna wydzielać duże ilości potu, którego parowanie powoduje odbiór ciepła od organizmu i jego chłodzenie. Zauważmy, że jest to klasyczny system ze sprzężeniem zwrotnym ujemnym. Im wyższa jest ciepłota ciała człowieka, tym intensywniej wytwarzany jest pot i tym więcej ciepła jest oddawane do otoczenia, co powoduje obniżenie ciepłoty ciała. Dopóki ten mechanizm działa sprawnie – człowiek jest w stanie utrzymać normalną ciepłotę ciała mimo znalezienia się w tropikalnym upale.

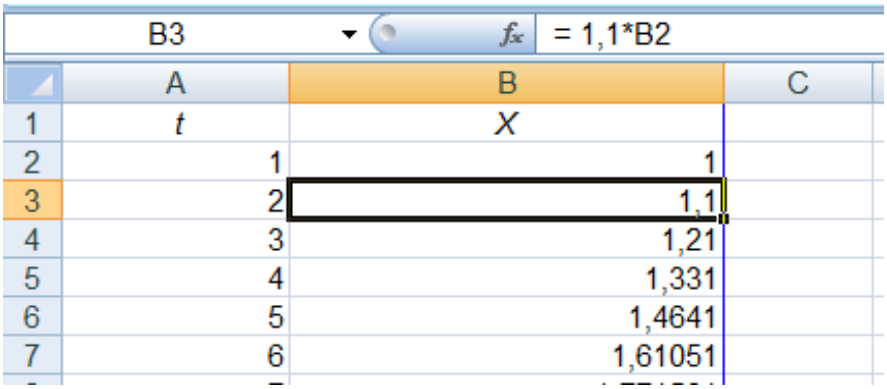
Podobny mechanizm działa także w przypadku, gdy człowiek znajdzie się w zimnym otoczeniu. Ochłodzenie ciała powoduje zwiększenie produkcji ciepła w tkankach, w których metabolizowana jest glukoza pochodząca z pokarmu. Proces „spalania" tego biologicznego „paliwa" wytwarza potrzebne ciepło, a ponieważ to spalanie najintensywniej zachodzi w pracujących mięśniach (wystarczy spojrzeć na rozgrzane ciało sportowca po dużym wysiłku) – dlatego organizm chcąc wytworzyć więcej ciepła wprawia niekiedy ciało w drżenie, podczas którego naprzemiennie pracujące mięśnie zginaczy i prostowników

wytwarzają potrzebne ciepło. Nietrudno zauważyć, że tu także działa ujemne sprzężenie zwrotne.

Jak powiedziano wyżej – sprzężenie zwrotne ujemne zwykle stabilizuje system. Zwykle to nie znaczy jednak, że zawsze. Przyjrzymy się teraz temu problemowi nieco dokładniej.

2.4.5. Stabilność w systemie ze sprzężeniem zwrotnym

Żeby się przekonać, jak zachowują się systemy ze sprzężeniem zwrotnym dodatnim albo ujemnym w różnych warunkach – wykonajmy prostą symulację zachowania takiego systemu wykorzystując wzór (2.25). Najwygodniej jest to wykonać za pomocą arkusza kalkulacyjnego (na przykład Excela – Rys. 2.50).



Rys. 2.50. Symulacja systemu ze sprzężeniem zwrotnym dodatnim za pomocą arkusza kalkulacyjnego

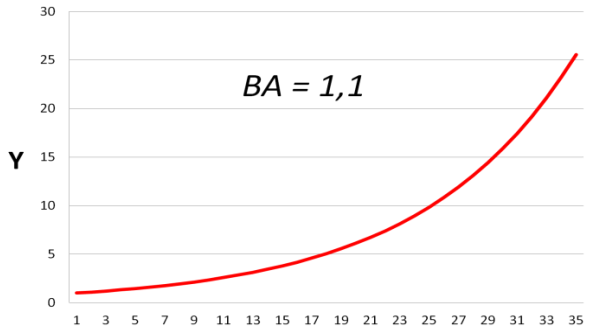
Załóżmy, że chcemy zaobserwować działanie systemy ze sprzężeniem dodatnim dla:

$$B\,A = 1{,}1.$$

W pierwszej kolumnie arkusza (w kolumnie A) wpisujemy oznaczenie *t* a następnie w kolejnych komórkach numery kolejnych chwili czasowych:

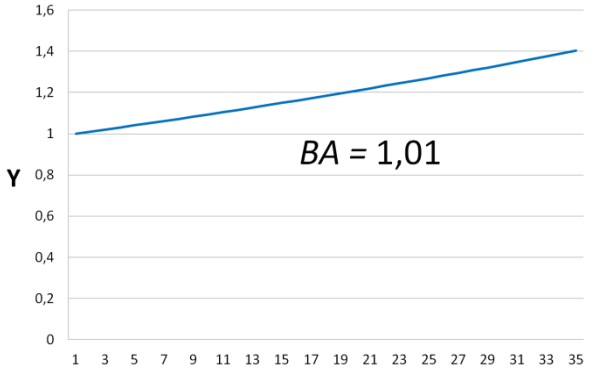
1, 2, 3 …

na początku drugiej kolumny wpisujemy oznaczenie *X* a w następnej komórce warunek początkowy *X(1)* (na przykład wartość 1). Dla przeprowadzenia symulacji w kolejnej komórce drugiej kolumny (komórka B3) wpisujemy formułę odpowiadającą wzorowi (2.25) dla $n = 1$ (patrz Rys. 2.50). Formułę tę kopiujemy następnie do kolejnych komórek kolumny B arkusza – i już możemy oglądać przebieg sygnału *X* w kolejnych chwilach czasowych w postaci tabelki wartości widocznych w arkuszu albo w postaci wykresu, który z pomocą narzędzi arkusza kalkulacyjnego można też bardzo łatwo sporządzić (Rys. 2.51).



Rys. 2.51. Niestabilne zachowanie systemu ze sprzężeniem zwrotnym dodatnim

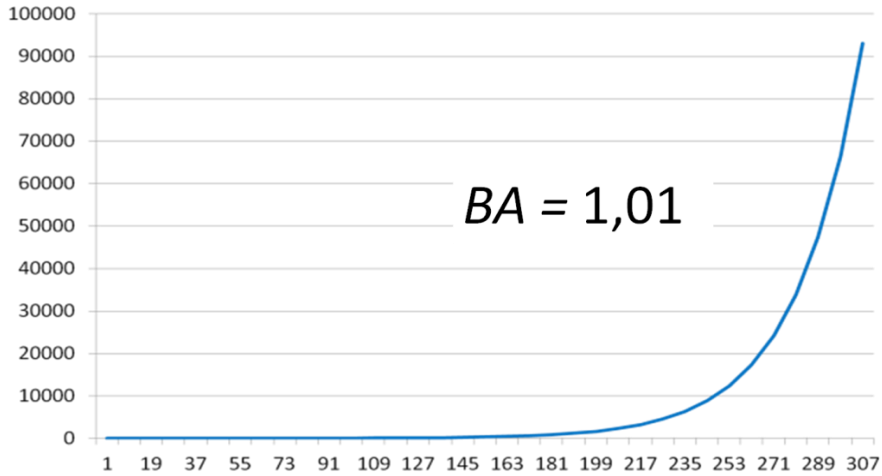
Zachowanie systemu przedstawione na rysunku 2.51 jest dokładnie takie, jakiego mogliśmy się spodziewać po systemie ze sprzężeniem zwrotnym dodatnim. W miarę upływu czasu obserwowany sygnał narasta, co więcej jego wzrost jest coraz szybszy i łatwo przewidzieć, że wszystko to zmierza do katastrofy. Jednak ta sytuacja związana jest z wartością iloczynu *A B*, który wcześniej pozwalał nam odróżnić system ze sprzężeniem dodatnim od systemu ze sprzężeniem ujemnym, a teraz pozwoli odróżnić system niestabilny od systemu stabilnego. Rozważmy bowiem zachowanie systemu ze sprzężeniem zwrotnym dodatnim, ale o mniejszej wartości iloczynu *A B* (Rys. 2.52).



Rys. 2.52. Dynamika układu ze sprzężeniem zwrotnym: sprzężenie dodatnie słabo niestabilne

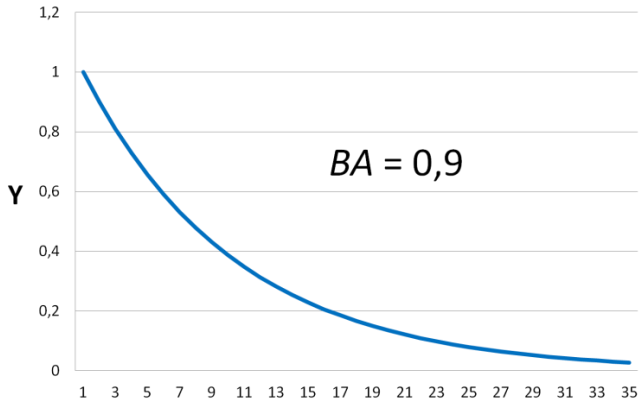
Symulację zachowania takiego systemu możemy uzyskać zmieniając formułę w komórce B3 arkusza i kopiując do kolejnych komórek B4 – B35. W taki właśnie sposób uzyskano wykres pokazany na rysunku 2.52. Widać, że w tym systemie nadal wartość sygnału wzrasta, ale wzrost ten jest powolny, przynajmniej na początku. Jeśli więc rozważany system reprezentuje proces niekorzystny – na przykład wspomniany w podrozdziale 2.4.4 i modelowany

w podrozdziale 4.7.2 wzrost raka – to przy niewielkiej dynamice obserwowanego wzrostu jest czas na interwencję medyczną, która pozwoli sytuację opanować. Oczywiście zwlekać nie można, bo każdy system ze sprzężeniem zwrotnym dodatnim ma tendencję do coraz szybszego wzrostu występujących w nim sygnałów, więc jeśli nawet początek zmian jest bardzo powolny i łagodny, to po upływie pewnego czasu sprawy nabierają dramatycznego tempa (Rys. 2.53).



Rys. 2.53. Po upływie dłuższego czasu każdy system ze sprzężeniem zwrotnym dodatnim wykazuje tendencję do nieograniczonego wzrostu sygnałów

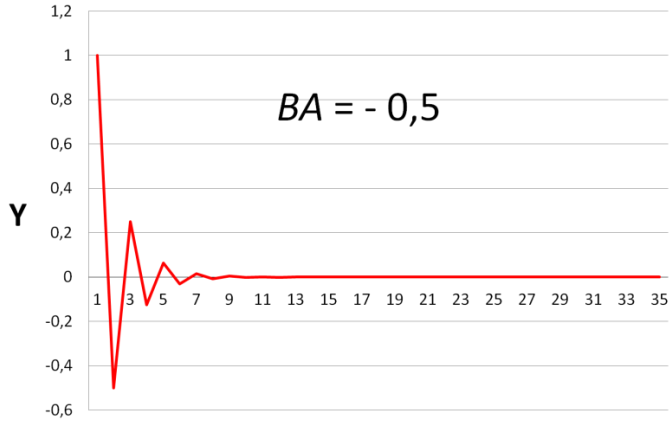
Jednak przekonanie o tym, że każdy system ze sprzężeniem zwrotnym dodatnim musi zmierzać do katastrofy byłoby przekonaniem błędnym. Okazuje się, że gdy wartości iloczynu $A\ B$ są mniejsze od 1 to system ze sprzężeniem zwrotnym dodatnim nie tylko nie wykazuje tendencji do samozagłady, ale wręcz samorzutnie się stabilizuje, eliminując początkowe zaburzenie (Rys. 2.54).



Rys. 2.54. Zachowanie systemu ze sprzężeniem zwrotnym dodatnim zachowującego jednak stabilność

Jak widać z przeprowadzonych eksperymentów – nie każdy system ze sprzężenie zwrotnym dodatnim musi dążyć do katastrofy, chociaż najczęściej systemy tego rodzaju skłonności do samozagłady mają wbudowane i trudno się ich pozbyć.

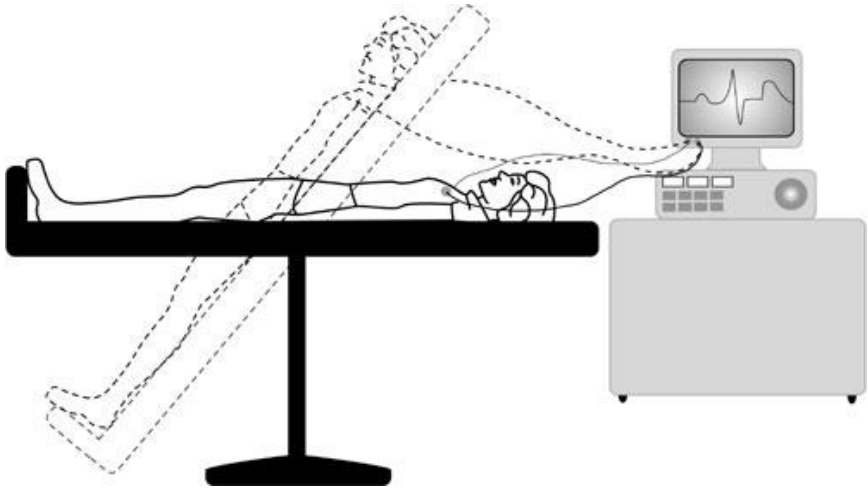
Zajmijmy się teraz systemem ze sprzężeniem zwrotnym ujemnym. Do badania jego zachowania użyjemy tego samego arkusza kalkulacyjnego, tylko do komórki B3 wpiszemy formułę zawierającą ujemny czynnik odpowiadający iloczynowi *A B,* kopiując ją potem do odpowiednich dalszych komórek kolumny B arkusza. Na początek przyjmijmy *A B* = -0,5. Zachowanie systemu można prześledzić na rysunku 2.55. Widać, że początkowe zaburzenie (bo tak może być traktowana każda niezerowa wartość *X(1)*) po krótkim czasie zostaje stłumione.



Rys. 2.55. Zachowanie typowego systemu z ujemnym sprzężeniem zwrotnym

Warto zauważyć jakościową różnicę pomiędzy przebiegiem sygnału pokazanym na rysunku 2.55, a rozważanym wcześniej przebiegiem z rysunku 2.54. W jednym i drugim przypadku zaburzenie jest tłumione, jednak przebieg na rysunku 2.55 wykazuje charakterystyczne **oscylacje**. Takie zafalowania sygnału są charakterystyczne dla systemów ze sprzężeniem zwrotnym ujemnym – jeśli gdziekolwiek cokolwiek oscyluje to na pewno jest w tym „zamieszany" jakiś system z ujemnym sprzężeniem zwrotnym. Na przykład drżenie rąk występujące u ludzi w starszym wieku lub chorych na chorobę Parkinsona jest skutkiem tego, że mięśnie używane podczas wykonywania dowolnych ruchów są sterowane przez nerwowo-mięśniowy system regulacji obejmujący między innymi tak zwane motoneurony w rogach przednich rdzenia kręgowego, nazywany pętlą gamma. A każdy system regulacji – to ujemne sprzężenie zwrotne.

Na szczególną uwagę zasługuje fakt, że usuwając zaburzenie w postaci niezerowej **dodatniej** wartości $X(1) = 1$ system ze sprzężeniem zwrotnym wygenerował odchyłkę o przeciwnym znaku $X(2)$ – w rozważanym przypadku wynoszącą $X(2) = -0{,}5$. To także charakterystyczna cecha systemów z ujemnym sprzężeniem zwrotnym – regulując określoną nieprawidłowość pojawiającą się w formie odchyłki od prawidłowych wartości sygnałów występujących w systemie system w pierwszej chwili wytwarza także odchyłkę, tylko przeciwnie skierowaną. Ta odchyłka wygenerowana w systemie nazywana jest **przeregulowaniem** i jest znana z codziennej obserwacji systemów, w których mamy do czynienia ze sprzężeniem zwrotnym i z jego regulacyjną rolą. Każdy Czytelnik mógłby z własnych doświadczeń życiowych przytoczyć liczne przykłady tego, jak po okresie nadmiernego „luzu" następowało przesadnie silne „dokręcenie śruby", jednak konsekwentnie trzymając się tego, że w tej książce chętnie odwołujemy się do przykładów biomedycznej natury omówimy następujące zjawisko: Otóż jednym z systemów ze sprzężeniem zwrotnym występujących w organizmie człowieka jest system regulacji ciśnienia krwi. Gdy ciśnienie to z jakichś powodów spadnie – uruchamiane są różne czynniki, które to ciśnienie mogą podnieść. Głównie następuje wzmocnienie akcji serca (powiększa się częstość uderzeń serca oraz siła jego skurczu) a także kurczą się naczynia krwionośne, co powoduje zwiększenie oporu jaki napotyka płynąca krew. Skurcz naczyń powoduje podwyższenie ciśnienia krwi podobnie jak zapora na rzece podnosi poziom wody. Wzmocniona akcja serca powoduje silniejsze pompowanie krwi i też podnosi ciśnienie. W sumie odchyłka (spadek ciśnienia) zostaje skompensowana, mamy więc z pewnością do czynienia ze sprzężeniem zwrotnym ujemnym. Analogiczny krąg przyczyn i skutków można by było prześledzić dla sytuacji kiedy poziom ciśnienia krwi wzrósł ponad miarę – serce wtedy spowalnia, a naczynia krwionośnie wiotczeją, co powoduje spadek ciśnienia.

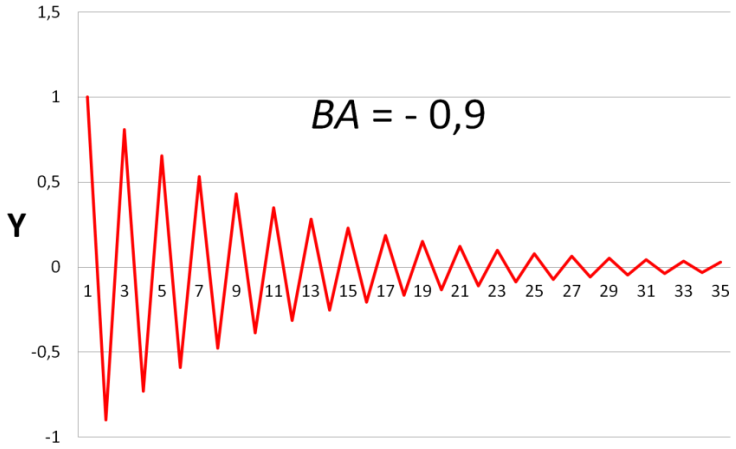


Rys. 2.56. Test pionizacji jako sposób badania systemu ze sprzężeniem zwrotnym związanego ze stabilizacją ciśnienia krwi (źródło: http://wdict.net/img/tilt+table+test.jpg, dostęp lipiec 2011)

Jeśli do tego systemu wprowadzimy celowe zakłócenie to możemy niekiedy zaobserwować efekt przeregulowania. Na przykład u osób, których układ sercowo-naczyniowy nie działa prawidłowo, prostym sposobem przetestowania funkcjonowania omówionego systemu regulacji jest tak zwany *test pionizacji*. Kładziemy pacjenta na łóżku, czekamy aż rytm serca (rejestrowany na leżąco) uspokoi się i ciśnienie krwi się ustabilizuje – i każemy pacjentowi wstać albo podnosimy go prawie do pionu wraz ze stosownie zmechanizowanym łóżkiem (Rys. 2.56).

Następuje skok ciśnienia związany z tym, że serce, które do tej pory pompowało krew płynącą wyłącznie w poziomie – musi teraz pokonać także opór słupa cieczy wznoszącego się od serca aż do poziomu czubka głowy. Serce w pierwszej chwili nie wydala, czego skutkiem jest spadek ciśnienia krwi. Układ regulacji ciśnienia wykrywa ten fakt i wywołuje przyspieszenia akcji serca i skurcz naczyń, co powoduje wzrost ciśnienia krwi. Występuje przy tym z reguły przeregulowanie i pacjentowi (jak to się obrazowo mówi) uderza krew do głowy, czyli po nadmiernym spadku ciśnienia następuje jego nienaturalny wzrost. Obserwując ten proces doświadczony kardiolog dowiaduje się bardzo wiele o tym, jak działa serce i układ krwionośny pacjenta, a o to w tym przypadku chodzi. Ale przyjemne to nie jest…

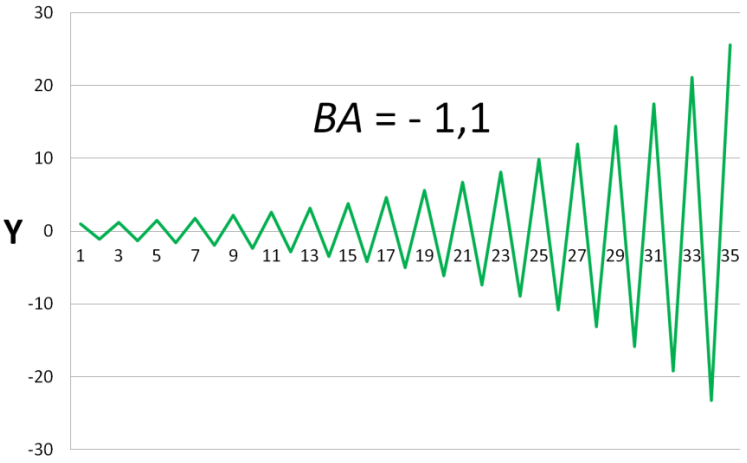
Wróćmy do oglądania przebiegów zmienności czasowej sygnałów różnych systemów ze sprzężeniem zwrotnym ujemnym. Przebieg pokazany na rysunku 2.55 powstał w systemie, w którym iloczyn $A\,B$ był ujemny, ale miał niewielką wartość bezwzględną. Co będzie, gdy tę wartość zwiększymy? Odpowiedź przynosi rysunek 2.57.



Rys. 2.57. Zachowanie systemu z ujemnym sprzężeniem zwrotnym przy większej wartości bezwzględnej współczynnika wzmocnienia $B\,A$

Jak widać system nadal ma właściwości stabilizujące (zaistniałe zakłócenie maleje do zera), ale proces przejściowy trwa długo, sygnał oscyluje i drgania gasną bardzo wolno, słowem – zbiera się na coś niedobrego.

To, co niedobrego może nas spotkać pokazuje rysunek 2.58. Po kolejnym zwiększeniu wartości bezwzględnej współczynnika wzmocnienia systemu otwartego $A\,B$ – system ze sprzężeniem zwrotnym **utracił stabilność**. Zaczął dążyć do katastrofy chociaż był systemem ze sprzężeniem zwrotnym ujemnym!



Rys. 2.58. Niestabilność w systemie ze sprzężeniem zwrotnym ujemnym

Podsumowując można stwierdzić, że w systemach ze sprzężeniem zwrotnym występują zjawiska i procesy, które w systemach złożonych z obiektów łączonych w inny sposób są wręcz nie do pomyślenia. System bez

sprzężeń zwrotnych, nawet tak duży jak przedstawiony na rysunku 2.30, zachowuje się w sposób prosty i ogólnie przewidywalny. Na wejściu pojawia się jakaś wartość sygnału *X*, w systemie sygnał ten jest jakoś przetwarzany, a w wyniku tego na wyjściu systemu pojawia się jakiś sygnał *Y*. Na tym praca systemu się kończy. Co więcej, jeśli do takiego systemu bez sprzężeń zwrotnych nie doprowadzimy żadnego zewnętrznego sygnału (sygnał wejściowy $X = 0$) – to system nie będzie nic robił i w przypadku jeśli jest systemem liniowym w takim sensie, jak to opisano w podrozdziale 2.2.4 – także sygnał wejściowy $Y = 0$.

Natomiast system ze sprzężeniem zwrotnym może przejawiać bardzo skomplikowane formy zachowania (rysunki od 2.51 do 2.58 przedstawiają i tak wyłącznie najprostsze przykłady) bez żadnych sygnałów docierających z zewnątrz! Co więcej, będzie przejawiał inne zachowanie, gdy będzie systemem ze sprzężeniem zwrotnym dodatnim, inne gdy sprzężenie będzie ujemne, inne w przypadku gdy zagwarantowana będzie stabilność systemu, jeszcze inne gdy granica stabilności będzie przekroczona – słowem możliwości jest wiele.

Cóż dopiero będzie się działo, gdy system wykazujący skomplikowaną wewnętrzną dynamikę będzie poddany zewnętrznym wymuszeniom, które będą skomplikowanymi funkcjami czasu?

Aby jednak budować i badać takie skomplikowane systemy musimy sięgnąć do aparatu matematycznego o znacznie bogatszych możliwościach niż ten (elementarny) jakiego do tej pory używaliśmy. O tym aparacie matematycznym będzie mowa w kolejnym podrozdziale.

2.5. Równania różniczkowe jako narzędzie opisu systemów dynamicznych

Modele różnych systemów tworzone zgodnie z zasadami opisywanymi w tej książce podzielić można na modele **statyczne** i **dynamiczne**. Modele **statyczne** charakteryzują się tym, że występujące w nich sygnały (wejściowe i wyjściowe) mają wartości nie zmieniające się w czasie. Taka sytuacja w modelowaniu i symulacji komputerowej określana jest jako **stan ustalony.** W książce[8] wydanej w 1975 roku jeden z autorów tego skryptu zapisał następującą uwagę:

Stan ustalony większości procesów jest stosunkowo łatwy do analizy. Jest to stan, w którym zjawiska trwają w niezmienionej postaci i można rozpatrywać je kolejno skupiając uwagę na poszczególnych fragmentach większej całości. Mamy przy tym gwarancję, że przebieg procesu we fragmentach już wcześniej rozpatrzonych nie ulegnie w czasie analizy żadnym zmianom; możemy zatem nie mieć ich stale „na oku" i zająć się dalszymi fragmentami.

Ten sposób analizy, aczkolwiek często i chętnie stosowany, nie jest jednak w pełni wystarczający. Rzeczywistość, jaka nas otacza, jest ***zmienna.*** *Większość*

[8] Tadeusiewicz R., Wajs W.: *Maszyny analogowe i ich programowanie*. Skrypt uczelniany AGH, nr 428, Kraków 1975

interesujących zjawisk polega właśnie na pewnych zachodzących w czasie i przestrzeni ***zmianach.***

W tej samej książce nieco dalej napisano:

Nasze zdolności analizy myślowej zawodzą przy analizie dynamiki złożonych procesów. Gubimy się, gdy „wszystko od wszystkiego zależy", gdy we wszystkich elementach następują równoczesne zmiany, gdy ruchów jest zbyt wiele.

Potrzebujemy więc przy analizie dynamiki zjawisk aparatu matematycznego, pozwalającego w sformalizowany sposób opisać interesujące nas fragmenty rzeczywistości.

Dalej w cytowanej książce dyskutowane są różne modele matematyczne pozwalające na opis dynamiki zjawisk, między innymi równania całkowe, równania różnicowe, równania różniczkowo-różnicowe, równania cząstkowe, by na końcu tej dyskusji sformułować następującą **tezę** (cytując ją tutaj zmieniono używane w oryginalnej książce oznaczenie *X* na używane w tej książce do tego samego celu oznaczenie *Y*). Z tą zmianą odpowiedni cytat brzmi:

Równania różniczkowe są ***najprostszym*** *możliwym do przyjęcia opisem dynamiki zjawisk. Zastanówmy się: w jaki najprostszy sposób można wprowadzić uzależnienie zmiennej Y od czasu t? Najlepiej (ze względu na łatwość zapisu) określać nie* ***wartość*** *Y dla wszystkich chwil czasu t, a określić w* ***danej*** *chwili czasowej* ***zmienność*** *Y. Zmienność tę można wyrazić w postaci prędkości chwilowej dY/dt, przyspieszenia* d^2Y/dt^2 *czy też dalszych pochodnych.*

Dalej następuje kluczowe dla przytaczanych tu rozważań stwierdzenie:

Zmienność Y w danej chwili czasu zależy jedynie od tej chwili czasu i od wartości Y w danej chwili.

Zapisując to stwierdzenie matematycznie otrzymujemy

$$\frac{dY}{dt} = f(t, Y(t)) \tag{2.26}$$

a każdy rozpozna, że wzór (2.26) przedstawia równanie różniczkowe zwyczajne.

W związku z tym dla zdecydowanej większości systemów dynamicznych ich modelu matematycznego poszukujemy w formie równania różniczkowego. Przykłady tworzenia takich modeli wykorzystujących równania różniczkowe podane będą w rozdziale 4. i tam odsyłamy Czytelnika po bliższe szczegóły. Natomiast jeszcze w tym rozdziale musimy podjąć dyskusję na temat tego, jak równania różniczkowe mogą być rozwiązywane za pomocą komputera, gdyż każdy model tylko wtedy może być praktycznie przydatny, gdy na jego podstawie potrafimy zbudować program komputerowy symulujący zachowanie tego modelu. W przypadku systemów dynamicznych opisywanych równaniami różniczkowymi symulacja oznacza konieczność numerycznego rozwiązywania tych równań – i o takim właśnie numerycznym rozwiązywaniu równań różniczkowych będzie mowa w następnym podrozdziale.

2.6. Równania różniczkowe zwyczajne pierwszego rzędu

W zagadnieniach praktycznych niejednokrotnie zdarza się, że znalezienie

rozwiązania równania różniczkowego lub układu równań modelujących pewien proces biologiczny wymaga czasochłonnych rachunków albo jest wręcz niemożliwe do wykonania przy wykorzystaniu znanych metod analitycznych. W tym przypadku z pomocą przychodzą metody numeryczne, przybliżające rozwiązanie numeryczne w postaci tabeli konkretnych wartości funkcji będącej rozwiązaniem równania przy zadanych warunkach początkowych na określonym przedziale. Takie rozwiązanie jest łatwo i wygodnie osiągalne (w końcu obliczenia wykonuje automatycznie komputer) jednak obarczone jest pewnym błędem w porównaniu do rozwiązań dokładnych, a ponadto jest zawsze rozwiązaniem szczegółowym, nie dającym podstaw do rozważań ogólnych i do ogólnych wniosków.

Zasadniczym typem równań różniczkowych, używanych w procesach modelowania różnych systemów, są równania różniczkowe zwyczajne pierwszego rzędu. Ponieważ równanie wyższego rzędu można zawsze przekształcić do postaci układu równań pierwszego rzędu, dlatego w dalszym ciągu tego podrozdziału będziemy się zajmowali równaniem postaci:

$$\frac{dY}{dt} = f(t, Y) \tag{2.27}$$

Równania tego typu przedstawiają zależność pomiędzy nieznaną funkcją *Y*, zależną od zmiennej *t,* a jej pierwszą pochodną. Rozwiązanie równania (2.27), polega na znalezieniu funkcji $Y = Y(t)$ określonej i różniczkowalnej na ustalonym przez nas przedziale $[a,b]$, która spełnia zależność:

$$\frac{dY}{dt} = f(t, Y(t)) \tag{2.28}$$

Przedział $[a,b]$ może wynikać z przyjętych przez nas założeń lub też być zdeterminowany przez dziedzinę funkcji po prawej stronie równania (2.28).

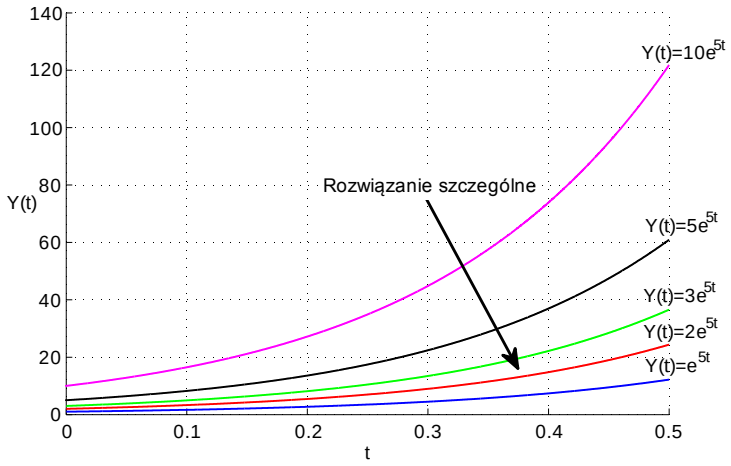
Przykład #1

Dla równania $\frac{dY}{dt} = 5Y(t)$ rozwiązaniem jest każda funkcja $Y(t) = Ce^{5t}$, gdzie $C \in \Re$.

W postaci analitycznej, przy braku warunku początkowego, rozwiązanie *Y(t)* nie jest jednoznaczne. Zależy od pewnej stałej całkowania *C*. Formuła taka nosi nazwę rozwiązania ogólnego.

Jeżeli do podanego przykładu zostanie dodany warunek początkowy np. $Y(0) = 2$, wówczas rozwiązanie przyjmie postać $Y(t) = 2e^{5t}$. Rozwiązanie to jest już jednoznaczne i nazywa się rozwiązaniem szczególnym (Rys. 2.59). Jest ono jednym z możliwych rozwiązań wywiedzionych z rozwiązania ogólnego. Na rysunku 2.59 pokazano tylko kilka spośród możliwych rozwiązań wywiedzionych z rozwiązania ogólnego, w istocie jest ich nieskończenie wiele.

Równanie różniczkowe (2.27) z warunkiem początkowym postaci $Y(t_0) = Y_0$ nosi nazwę zagadnienia początkowego Cauchy'ego.



Rys. 2.59. Wybrane rozwiązanie szczególne oraz kilka innych możliwych rozwiązań wywiedzionych z rozwiązania ogólnego

Jeżeli dodatkowo funkcja $f(t,Y)$ jest ciągła ze względu na zmienną *t* w ustalonym przez nas przedziale $[a,b]$ oraz spełnia warunek Lipschitza ze względu na zmienną *Y*:

$$\exists_{L\in\Re}\forall_{Y_1,Y_2\in\Re}\left|f(t,Y_1)-f(t,Y_2)\right|\le L\left|Y_1-Y_2\right| \tag{2.29}$$

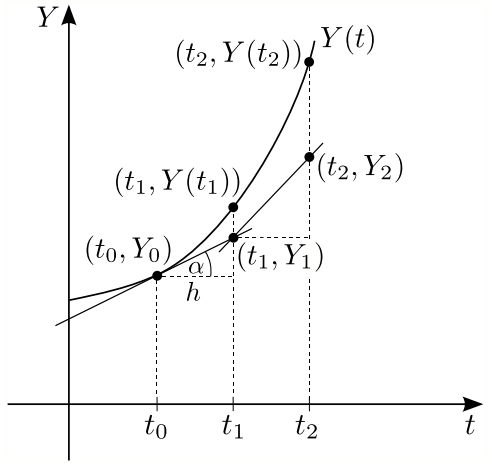
wówczas w przedziale $[a,b]$ istnieje dokładnie jedno rozwiązanie *Y(t)* równania (2.27) z warunkiem początkowym $Y(t_0)=Y_0$. Warunek (2.29) oznacza, że jesteśmy w stanie dobrać pewną stałą *L*, dla której, dla dowolnych Y_1 i Y_2, istnieje górne ograniczenie szybkości zmienności funkcji.

Tak zdefiniowane problemy mogą być rozpatrywane przy wykorzystaniu numerycznych metod całkowania. Wymienimy teraz kilka z nich.

2.6.1. Metoda Eulera

Najprostszym algorytmem numerycznego rozwiązywania zagadnienia początkowego Cauchy'ego jest metoda Eulera[9], nazywana często metodą stycznych (Rys. 2.60).

[9] Leonhard Euler (1707 - 1783) – szwajcarski matematyk i fizyk



Rys. 2.60. Graficzna interpretacja metody Eulera. Styczna w punkcie (t_0, Y_0) wyznacza kolejny punkt (t_1, Y_1) aproksymacji rozwiązania. Wartość $|Y(t_1) - Y_1|$ jest błędem bezwzględnym metody numerycznej dla pierwszego kroku

Niech dana będzie krzywa *Y(t)* będąca rozwiązaniem równania różniczkowego postaci (2.27) oraz punkty (t_0, Y_0), (t_1, Y_1) i (t_2, Y_2) takie że (Rys. 2.60):

$$t_1 = t_0 + h, \quad t_2 = t_1 + h \tag{2.30}$$

gdzie *h* jest stałym krokiem całkowania.

Tangens kąta nachylenia stycznej do krzywej *Y(t)* w dowolnym punkcie (t, Y) do osi odciętych jest wartością pochodnej funkcji w tym punkcie. W oparciu o równanie (2.27), pochodną funkcji *Y(t)* w punkcie (t_0, Y_0) można zastąpić wyrażeniem $f(t_0, Y_0)$. Korzystając z własności funkcji tangens i nachylenia stycznej do krzywej *Y(t)*, możliwe jest powiązanie punktów (t_0, Y_0) i (t_1, Y_1) oraz wartości funkcji $f(t_0, Y_0)$ (Rys. 2.60):

$$tg\alpha = f(t_0, Y_0) = \frac{Y_1 - Y_0}{t_1 - t_0} = \frac{Y_1 - Y_0}{h} \tag{2.31}$$

$$hf(t_0, Y_0) = Y_1 - Y_0 \tag{2.32}$$

Ostateczna formuła pozwalająca znaleźć wartość punktu Y_1 jest następująca:

$$Y_1 = Y_0 + hf(t_0, Y_0) \tag{2.33}$$

Podstawiając $Y_0 = Y_1$ oraz $Y_1 = Y_2$, wartość punktu Y_2 wynosi:

$$Y_2 = Y_1 + hf(t_1, Y_1) \tag{2.34}$$

Dokonując uogólnienia formuły (2.34) dla kroku *n+1*, otrzymano algorytm iteracyjny aproksymacji rozwiązania równania różniczkowego, znany z literatury jako metoda Eulera:

$$Y_{n+1} = Y_n + hf(t_n, Y_n), \quad n = 0,1,2,... \tag{2.35}$$

Przykład #2

Wykorzystując metodę Eulera, rozwiążemy teraz w sposób numeryczny równanie różniczkowe (2.36) w przedziale $[0,1]$ przyjmując krok całkowania $h = 0.1$ oraz warunek początkowy $Y(0) = 0.4$

$$\frac{dY}{dt} = 5Y(t) - 1 \tag{2.36}$$

Kolejne iteracje numerycznego całkowania, w oparciu o formułę (2.35), przedstawiają się następująco:

$$Y_1 = Y_0 + h(5Y_0 - 1) = 0.4 + 0.1(5 \cdot 0.4 - 1) = 0.5 \tag{2.37}$$

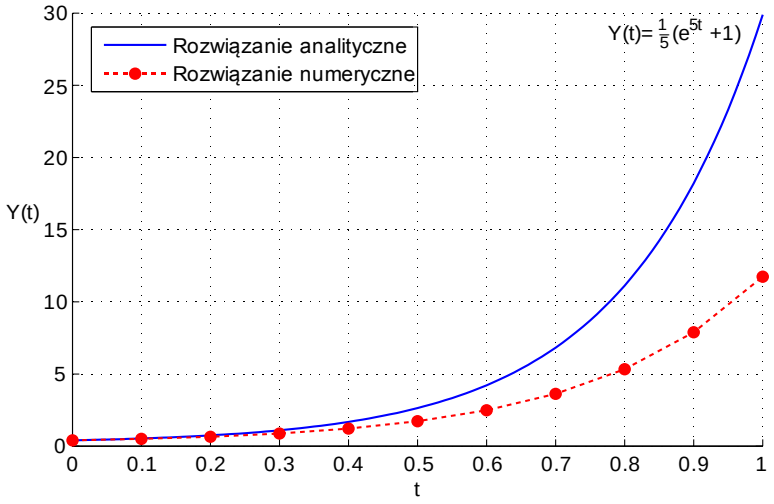
$$Y_2 = Y_1 + h(5Y_1 - 1) = 0.5 + 0.1(5 \cdot 0.5 - 1) = 0.65 \tag{2.38}$$

...

$$Y_{10} = Y_9 + h(5Y_9 - 1) = 7.888 + 0.1(5 \cdot 7.888 - 1) = 11.773 \tag{2.39}$$

Analityczne rozwiązanie tego równania, przy wykorzystaniu metody uzmienniania stałej, to krzywa (Rys. 2.61):

$$Y(t) = \frac{1}{5}(e^{5t} + 1) \tag{2.40}$$



Rys. 2.61. Porównanie analitycznego i numerycznego rozwiązania równania różniczkowego (2.36) na przedziale $[0,1]$.

Uważny czytelnik z pewnością zauważył, iż przebiegi rozwiązania analitycznego i numerycznego nie dość, że nie pokrywają się, to są określone na różnych zbiorach (Rys. 2.61). W przypadku numerycznego rozwiązania równania różniczkowego zwyczajnego mamy do czynienia tylko z wartościami funkcji określonej w dyskretnych chwilach czasu *t*. Odległość pomiędzy dwoma sąsiednimi punktami określona jest za pomocą kroku całkowania *h*. W pozostałych punktach wartość rozwiązania nie jest znana. Zmniejszenie kroku całkowania *h* wiąże się ze zwiększeniem dokładności rozwiązania (mniejsze błędy numeryczne), jednak obarczone jest dłuższym czasem obliczeń. W przeciwieństwie do rozwiązania numerycznego, analityczna postać funkcji $Y(t)$ określona jest dla dowolnej chwili czasu $t \geq 0$.

2.6.2. Metoda Rungego-Kutty IV rzędu

Wyprowadzenie formuł numerycznych dla metody Rungego-Kutty IV rzędu jest czasochłonne i wymaga skrupulatnych rachunków. Podamy więc jedynie końcowe wzory metody jednocześnie odsyłając zainteresowanego czytelnika do prac w których zawarto odpowiednie wyprowadzenia[10] [11].

$$Y_{n+1} = Y_n + \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F_4), \quad n = 0,1,2,... \tag{2.41}$$

gdzie:

$$F_1 = hf(t_n, Y_n) \tag{2.42}$$

$$F_2 = hf(t_n + \frac{1}{2}h, Y_n + \frac{1}{2}F_1) \tag{2.43}$$

$$F_3 = hf(t_n + \frac{1}{2}h, Y_n + \frac{1}{2}F_2) \tag{2.44}$$

$$F_4 = hf(t_n + h, Y_n + F_3) \tag{2.45}$$

2.6.3. Podsumowanie

Przedstawione powyżej algorytmy są jedynie „wierzchołkiem góry lodowej" wśród numerycznych metod rozwiązywania równań różniczkowych

[10] D. Kincaid, W. Cheney, *Analiza numeryczna*, Wydawnictwo Naukowo-Techniczne, Warszawa, 2006

[11] E. W. Cheney, D. R. Kincaid, *Numerical Mathematics and Computing*, Brooks/Cole, 2008

zwyczajnych. Niemniej, są one na tyle istotne, że zostały uwzględnione w niniejszym skrypcie. Należy być świadomym, że oprócz metod jednokrokowych, o których wspomniano, istnieją metody wielokrokowe. Różnica między nimi polega na tym, że metody wielokrokowe do wyznaczenia punktu Y_{n+1} korzystają z wartości w kilku poprzednich punktach $Y_n, Y_{n-1}, .., Y_{n-k}$. Obszerny przegląd metod wielokrokowych można znaleźć w pracy[12].

Oddzielnym zagadnieniem są metody o zmiennym kroku *h* oraz zmiennym rzędzie metody. Znajdują one zastosowania w rozwiązaniach sztywnych układów równań różniczkowych, tzn. takich, w których jedna ze składowych $Y_i(t)$ zmienia się bardzo szybko (lub bardzo wolno) w porównaniu do pozostałych. Sztywne układy równań pojawiają się m. in. w precyzyjnym określaniu trajektorii obiektów kosmicznych, jak również w kontroli przebiegu procesów chemicznych.

Analiza numeryczna to nie tylko zaprogramowanie algorytmu rozwiązującego zagadnienie w postaci numerycznej. Jej ważnym etapem jest badanie zbieżności i szacowanie błędów numerycznych metod. Są to zagadnienia trudne i wymagające znajomości wielu metod analizy matematycznej. Więcej szczegółów na ten temat wnikliwy czytelnik znajdzie w pracach cytowanych w wykazie literatury na końcu książki.

[12] B. Bożek, *Metody obliczeniowe i ich komputerowa realizacja*, Uczelniane Wydawnictwa Naukowo-Dydaktyczne, Kraków, 2005

ROZDZIAŁ 3

WPROWADZENIE DO ŚRODOWISKA MATLAB

3.1. Wstęp

Prezentowany skrypt ma – jak wiadomo – trzy cele.

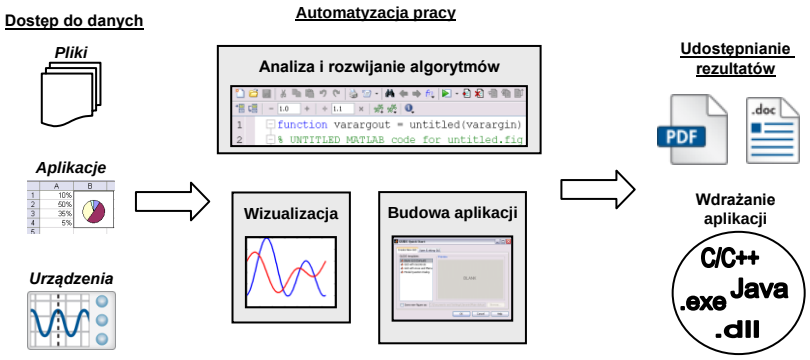
Po pierwsze ma nauczyć podstaw modelowania matematycznego różnych obiektów. Temu służyły dwa poprzednie rozdziały. Po drugie ma nauczyć tego, jak można abstrakcyjny model matematyczny przekształcić na program symulacyjny, pozwalający na prowadzenie badań modelowanego obiektu. To będzie głównie przedstawione w rozdziale 4 na licznych przykładach o rosnącym stopniu złożoności. Zwornikiem pomiędzy modelem rozumianym jako zbiór formuł matematycznym, a modelem rozumianym jako działający program pozwalający na komputerową symulację rozważanego systemu – jest informatyczne narzędzie wybrane do tego, by na podstawie równań tworzyć symulacyjne programy.

Takim informatycznym narzędziem może być teoretyczne dowolny język programowania. Jeden z autorów tej książki pierwsze swoje modele (oczywiście systemów biocybernetycznych) budował pisząc własne programy w Algolu, w Fortranie, nawet w popularnym kiedyś Basicu[13] a ostatnio także w języku C i w jego odmianie zwanej C#. Programy te w wersji skompilowanej (gotowej do użycia), ale także w postaci kodów źródłowych w języku C# dostępne są na stronie: http://home.agh.edu.pl/~tad, można więc zobaczyć, jak takie modele wyglądają i jak działają.

Jednak samodzielne pisanie programów symulacyjnych przy użyciu języków programowania przeznaczonych do ogólnego zastosowania - jest pracochłonne. Dlatego wraz z rozpowszechnianiem się technik modelowania i symulacji komputerowej – zaczęły powstawać specjalne środowiska programowe służące do tego, by tworzyć dobre programy symulacyjne przy minimalnym wysiłku i przy bardzo dobrym (także z punktu widzenia estetyki) wyniku końcowym.

W niniejszym skrypcie posłużono się środowiskiem MATLAB. Środowisko to stało się faktycznym standardem we współczesnych obliczeniach naukowo-technicznych, pozwalającym na przyspieszenie rozwiązywania różnorodnych problemów badawczych oraz inżynierskich. Przyspieszenie to jest osiągane przede wszystkim przez automatyzację rutynowych czynności obejmującą wszystkie fazy rozwiązywania danego zadania: od zbierania danych, poprzez ich analizę i rozwijanie algorytmów, do prezentacji wyników i wdrażania aplikacji. Ogólny schemat środowiska MATLAB przedstawia rysunek 3.1.

[13] Kilkadziesiąt programów w Basicu symulujących systemy biocybernetyczne znane jako sieci neuronowe znaleźć można w książce: Tadeusiewicz R., *Elementarne wprowadzenie do sieci neuronowych z przykładowymi programami*, Akademicka Oficyna Wydawnicza, Warszawa, 1998. Książka ta dostępna jest na stronie http://winntbg.bg.agh.edu.pl/skrypty2/0263/



Rys. 3.1. Środowisko MATLAB oferujące między innymi wsparcie dla twórcy modeli symulacyjnych w typowych zadaniach

MATLAB to duże i wielozadaniowe narzędzie z którego w tym skrypcie będziemy korzystali w sposób wybiórczy, skupiając uwagę wyłącznie na tych elementach, które będą potrzebne do budowy prostych modeli biocybernetycznych.

Osoby, które znają MATLAB mogą dalszą część tego rozdziału pominąć. Osoby, które go nie znają, a chcą poznać go w całości – także powinny zrezygnować z czytania tego rozdziału, a zamiast tego powinny sięgnąć po jeden z licznych łatwo dostępnych podręczników opisujących całego MATLABa[14].

W tym rozdziale przedstawione zostały bowiem wyłącznie wybrane informacje na temat środowiska MATLAB, pozwalające Czytelnikowi, niezaznajomionemu wcześniej z tym narzędziem, rozpocząć pracę i wykorzystać możliwości tego oprogramowania **do modelowania systemów biologicznych omawianych w skrypcie.** Jednak nawet ci bardzo początkujący użytkownicy środowiska MATLAB nie powinni czytać tego rozdziału „jednym tchem" od początku do końca. Środowisko MATLAB przeznaczone jest bowiem dla **praktyków**, dla ludzi, którzy komputera **używają**, a nie studiują teoretycznie jego działanie. Dlatego szkoląc się w korzystaniu ze środowiska MATLAB trzeba po pierwsze jak najszybciej znaleźć komputer, na którym to środowisko jest zainstalowane i działa, a potem skupić się na jego **używaniu** poprzez wprowadzanie i uruchamianie najpierw prostych, a potem coraz bardziej skomplikowanych modeli symulowanych właśnie w tym środowisku. Idealnym zasobem takich właśnie przykładowych modeli jest rozdział 4 tego skryptu, w którym przedstawiono programy modelujące różne systemy biologiczne w

[14] Tytułem przykładu wskażemy tu niedawno (12/2010) wydaną czterystustronicową książkę: *Mrozek B., Mrozek Z., MATLAB i Simulink.* Poradnik użytkownika. Wyd. III, Helion

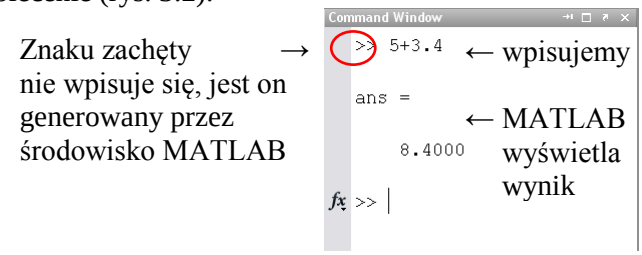
postaci M-kodów (patrz dalej w tym rozdziale) dla środowiska MATLAB. W kolejnych podrozdziałach tego rozdziału omówione zostaną możliwości środowiska przydatne do zrozumienia sposobu budowy i działania tych przykładowych modeli prezentowanych w rozdziale 4 skryptu.

Optymalny sposób korzystania z wiadomości zawartych w tym rozdziale polega więc na tym, żeby przeczytać dokładnie pierwsze dwa podrozdziały tego rozdziału, a resztę przejrzeć powierzchownie – i przejść do czytania rozdziału 4 i do wykonywania z pomocą komputera zawartych w nim programów symulacyjnych. W przypadku napotkania w rozdziale 4 niezrozumiałej komendy MATLABa – należy wrócić do tego rozdziału, odnaleźć stosowne wyjaśnienie, przyswoić je sobie – i powrócić do przykładów w rozdziale 4.

Warto przy tym wziąć pod uwagę następującą wskazówkę: Ze względu na fakt, iż środowisko MATLABa podlega ciągłemu rozwojowi[15], niektóre informacje oraz zamieszczone w skrypcie obrazy i schematy mogą się nieznacznie różnić od aktualnych. Nie powinno to stanowić problemu dla Czytelnika ze względu na spójność wyglądu i nazewnictwa pomiędzy kolejnymi wersjami środowiska. Ponieważ dokumentacja środowiska jest w języku angielskim, wszędzie tam gdzie jest to konieczne podawane są nazwy anglojęzyczne.

3.2. Podstawy pracy ze środowiskiem

MATLAB jest aplikacją udostępniającą środowisko programistyczne wysokiego poziomu, tzn. pozwala na szybką realizację różnego rodzaju obliczeń numerycznych, bez potrzeby tworzenia dużej ilości kodu. Jako podstawowy przykład można wziąć sytuację dodawania dwóch liczb. Można to osiągnąć po prostu wpisując w linii komend MATLABa („*Command Window*") odpowiednie polecenie (rys. 3.2).



Rys. 3.2. Dodawanie dwóch liczb w MATLABie

W dalszej części skryptu, polecenia wpisywane w linii komend będą oznaczane za pomocą znaków >> (znaków tych nie należy przepisywać).

Wynik obliczeń jest przypisywany do zmiennej domyślnej oznaczonej jako „ans". Czytelnik powinien również zwrócić uwagę, iż separatorem dziesiętnym

[15] Przykłady zawarte w niniejszym skrypcie zostały utworzone w wersji R2011a MATLABa. W innych wersjach widoki ekranu mogą się nieznacznie różnić.

w MATLABie jest **kropka**, a nie przecinek.

Na podstawie powyższego przykładu widoczne jest, iż okno poleceń MATLABa pełni rolę rozbudowanego kalkulatora. To co odróżnia MATLABa od popularnych kalkulatorów, to możliwość definiowania różnego rodzaju zmiennych, przypisywania im wartości i wykorzystania ich w dalszych obliczeniach. Obrazuje to przykład 3.1, w którym definiujemy zmienne „a" i „b" – a następnie dodajemy je, rezultat przypisując do zmiennej „c". Warto zwrócić uwagę, iż MATLAB jest wrażliwy na wielkość liter, tzn. zmienna oznaczona małą literą „a" to nie jest tożsama ze zmienną oznaczoną wielką literą „A".

Bardzo często w przykładach kodu MATLABa można zauważyć, iż na końcu polecenia jest wstawiony znak średnika (;). Znak ten umieszczony na końcu polecenia powoduje, iż środowisko nie wyświetla rezultatu w linii poleceń, ale polecenie jest wykonywane – patrz zapis ze zmienną „d" w przykładzie 3.1.

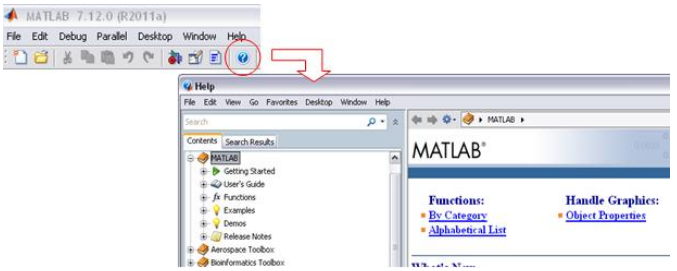
Przykład 3.1 – Definiowanie zmiennych i wykorzystanie ich do obliczeń

```
>> a=5
a =
     5

>> b=3.4
b =
    3.4000

>> c=a+b
c =
    8.4000

>> d=a-b;
>> e=d*10

    16
```

Jako kolejny przykład weźmy obliczanie pierwiastka kwadratowego z liczby (przykład 3.2). Można zadać pytanie – skąd Czytelnik może wiedzieć jakie polecenie w MATLABie służy do wyznaczania pierwiastka kwadratowego? O ile operatory nakazujące dodawanie, odejmowanie oraz inne podstawowe operacje matematyczne są raczej oczywiste, to znajomość sposobu zapisywania funkcji stanowiących wyposażenie środowiska może stanowić początkowo problem.

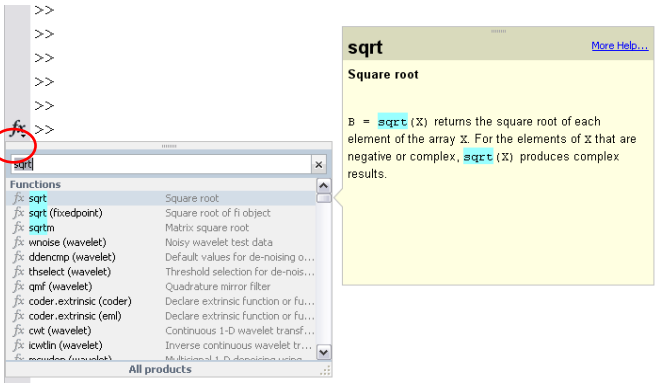
Autorzy skryptu zachęcają w tym kontekście do korzystania z systemu pomocy MATLABa – jest on bardzo pomocny, szczególnie dla początkujących użytkowników. Dostęp do systemu pomocy jest możliwy na kilka sposobów:

– z menu „Help”,
– z paska narzędzi (ikona „pytajnika” – rys.3.3),



Rys. 3.3. System pomocy środowiska MATLAB

za pomocą ikony znajdującej się obok znaku zachęty w linii poleceń (rys.3.4).



Rys. 3.4. Pomoc kontekstowa w linii poleceń

polecenie `help <nazwa funkcji>` wpisywane w oknie poleceń środowiska („*Command Window*” – patrz dalej),
– polecenie `doc <nazwa funkcji>` wpisywane w oknie poleceń środowiska („*Command Window*”),
– zaznaczenie poszukiwanego słowa i naciśnięcie klawisza F1.

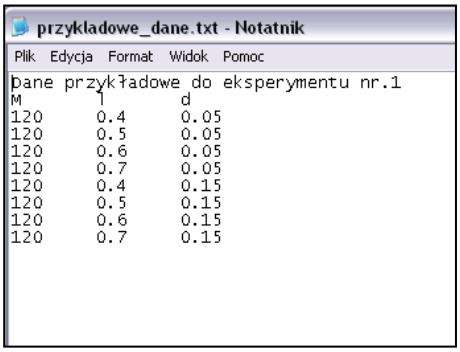
System pomocy oferuje kilka przydatnych możliwości. Są to m.in.:
- wyszukiwarka wg słów kluczowych,
- podział na kategorie (każdy moduł rozszerzający środowiska posiada osobny rozdział w dokumentacji),
- listę funkcji (zakładka „*Functions*”), dzięki której możliwe jest zorientowanie się w funkcjonalności danego modułu,
- przykłady (zakładki „*Examples*” oraz „*Demos*”), prezentujące ciekawe zastosowania danego modułu.

Przykład 3.2 – Użycie przykładowego polecenia - pierwiastek kwadratowy z liczby

```
>> f=sqrt(e)
f =
      4
```

3.3. Dostęp do danych

W procesie modelowania dowolnego fragmentu rzeczywistości, nieodłącznym etapem jest zbieranie lub tworzenie danych – na podstawie których dokonywana jest symulacja. Środowisko MATLAB zapewnia odpowiednią funkcjonalność oraz szereg narzędzi, pozwalający na szybki import danych, praktycznie w każdym formacie plikowym, bądź też bezpośrednio z aplikacji i urządzeń zewnętrznych (np. kamery, karty akwizycji, urządzenia pomiarowego).



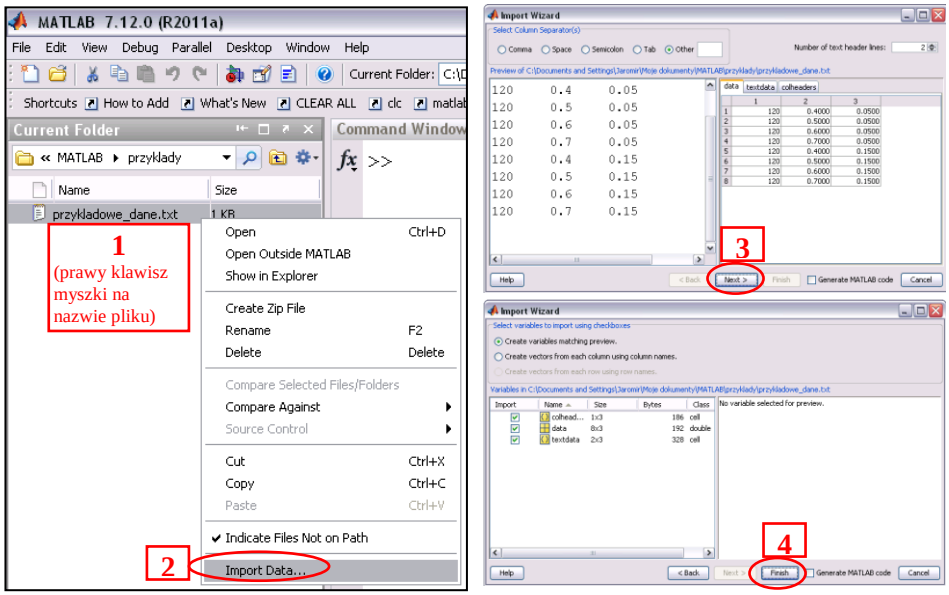
Rys. 3.5. Przykładowy plik tekstowy wykorzystany do importu danych

Jednym z tych narzędzi jest „*Import Wizard*" dostępny z poziomu przeglądarki plików. Aby przećwiczyć importowanie danych Czytelnik może utworzyć przykładowy plik tekstowy korzystając z dowolnego edytora tekstu (rys.3.5) i zapisać pod znaną nazwą (tu *przykładowe_dane.txt*) w jakimś katalogu, którego nazwę trzeba zapamiętać. Po uruchomieniu MATLABa, w oknie interfejsu oznaczonym jako „*Current Folder*" należy przejść do katalogu, w którym znajduje się plik który ma być importowany. Dalsze kroki przedstawione zostały na rysunku 3.6:

- uruchomienie narzędzia (kroki 1-2),
- możliwość podglądu danych (krok 3) oraz
- możliwość wyboru i nazwania zmiennych (krok 4) które będą utworzone w przestrzeni roboczej środowiska.

Za pomocą opisanego narzędzia, import danych jest w pełni interaktywny (na zasadzie „klikania" i wyboru opcji). Należy zauważyć, iż dostępne opcje są zależne od typu wybranego pliku. Jako ćwiczenie Czytelnik może spróbować zaimportować inny rodzaj pliku, np. obraz.

Import danych jest również możliwy poprzez użycie odpowiednich funkcji odczytu danych, wpisywanych w oknie poleceń „*Command Window*". Sposób wpisywania poleceń podany był w poprzednim podrozdziale. Aby zobaczyć rezultaty wykonania tych poleceń, należy je wpisać w oknie komend MATLABa (rezultaty nie są prezentowane w niniejszym skrypcie ze względu na zbyt dużą ilość miejsca które mogłyby zajmować).



Rys. 3.6. Import danych z pliku tekstowego za pomocą narzędzia „*Import Wizard*"

Najbliższym odpowiednikiem dla opisanego wyżej interaktywnego „*Import Wizarda*" w programach pisanych dla środowiska MATLAB jest polecenie `importdata`. Pozwala ono na import danych z różnych rodzajów plików (przykład 3.3).

Przykład 3.3 – programowy import danych z pliku

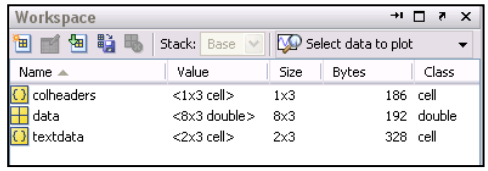
```
>> dane=importdata('przykladowe_dane.txt')
```

MATLAB wyposażony jest również w funkcje dedykowane do odczytu danych z określonych formatów plikowych – przykładowo:

- `imread` – pozwala na wczytywanie obrazów,
- `xlsread` – pliki programu Excel,
- `textscan` – pliki tekstowe.

Pełną listę dostępnych funkcji odczytu można znaleźć w dokumentacji (`doc fileformats`), przy czym należy zwrócić uwagę iż moduły rozszerzające oferują bardzo często dodatkowe funkcje importu danych (nie zamieszczone na tej liście).

Po zaimportowaniu lub utworzeniu danych są one widoczne w przeglądarce przestrzeni roboczej MATLABa (rys. 3.7). Zakładka ta wyświetla istotne informacje na temat zmiennych aktualnie znajdujących się w pamięci (nazwa, rozmiar, typ danych, wielkość w pamięci itd.). Widoczna jest tutaj jedna z charakterystycznych cech środowiska MATLAB – wszystkie dane i zmienne są tablicami (pojedyncza liczba to tablica o rozmiarze 1x1). Inną istotną informacją jest typ danych. Domyślny typ danych (wystarczający do obliczeń oraz przykładów podawanych w niniejszym skrypcie) to typ „*double*" oznaczający liczby zmiennoprzecinkowe (tzn. takie które mogą mieć część ułamkową) o podwójnej precyzji. Praca z tablicami oraz inne typy danych zostaną omówione w dalszych podrozdziałach.

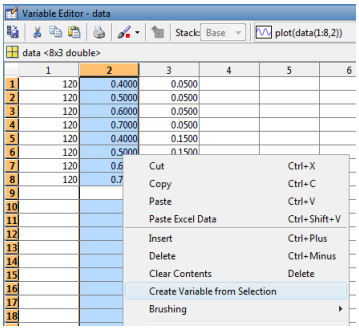


Rys. 3.7. Przestrzeń robocza środowiska „*Workspace*"

3.4. Praca z danymi i wizualizacja w środowisku MATLAB

Nieodłącznym elementem modelowania jest wizualizacja oraz wstępna ocena zaimportowanych danych. Środowisko MATLAB oferuje szereg narzędzi pozwalających na szybką wizualizację i przetwarzanie danych.

Z poprzedniego przykładu (import pliku tekstowego) powinna zostać utworzona w przestrzeni roboczej zmienna `data`, zawierająca zaimportowane dane numeryczne. Podwójne kliknięcie na nazwie zmiennej (w "*Workspace*") uruchamia edytor tablic ("*Variable editor*") umożliwiający podgląd oraz edycję danych (rys.3.8).

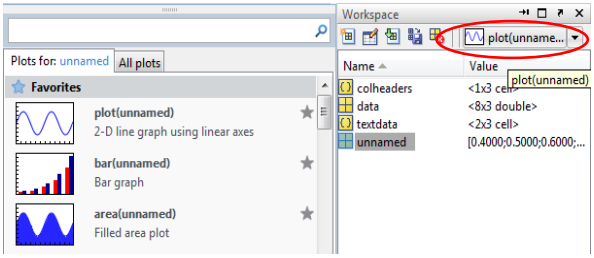


Rys. 3.8. Edytor tablic - tworzenie nowej zmiennej

Jedną z możliwości edytora jest tworzenie nowych zmiennych – wystarczy zaznaczyć fragment z danymi i wybrać z menu kontekstowego (prawy klawisz myszki) polecenie "*Create Variable from Selection*". Można również utworzyć nową zmienną i wypełnić ją własnymi danymi, podobnie jak w Excelu. Dane mogą również zostać utworzone bezpośrednio w MATLABie.

Mając dane zaimportowane do przestrzeni roboczej (bądź też utworzone bezpośrednio w tym środowisku MATLAB) można rozpocząć pracę z nimi.

MATLAB pozwala szybko i wygodnie wizualizować dane. W sposób interaktywny można to osiągnąć wybierając zmienną w zakładce "*Workspace*", następnie klikając w ikonę katalogu **wykresów** (rys.3.9). Z wyświetlonej listy należy wybierać potrzebny typ wykresu.



Rys. 3.9. Katalog wykresów

Wykresy mogą być również tworzone za pomocą poleceń (co jest wygodniejsze dla zaawansowanych użytkowników). Istnieje wiele różnych poleceń tworzenia wykresów – najprostszą drogą do ich poznania jest użycie katalogu wykresów zawierającego szczegółowe opisy. Jednym z najczęściej używanych poleceń jest komenda `plot`. Ma ona wiele odmian składni:

```
plot(y)        % gdzie y to nazwa zmiennej której
               % wykres chcemy wyświetlić

plot(x,y)      % j.w. ale dla danych x i y
```

W przykładach podanych powyżej pokazano możliwość zaopatrywania poleceń dla MATLABa w **komentarze**. Do tworzenia komentarzy używa się znaku %. Jeśli się go umieści w jakiejś linii, to cały dalszy tekst od tego znaku do końca linii jest przez MATLAB ignorowany, użytkownik może więc umieścić tam dowolne wyjaśnienia i komentarze, które nie będą wpływały na zachowanie komputera, ale będą pozwalały na umieszczenie dowolnych objaśnień i notatek czytelnych tylko dla ludzkiego oka.

Jeśli mówimy o komendzie `plot` to szczególnie przydatna jest jej składnia pozwalająca na zdefiniowanie rodzaju linii, markera oraz koloru dla tworzonego wykresu.

```
plot(x,y,'r') % dodatkowy parametr 'r' określający
              % rodzaj linii i markera oraz kolor
```

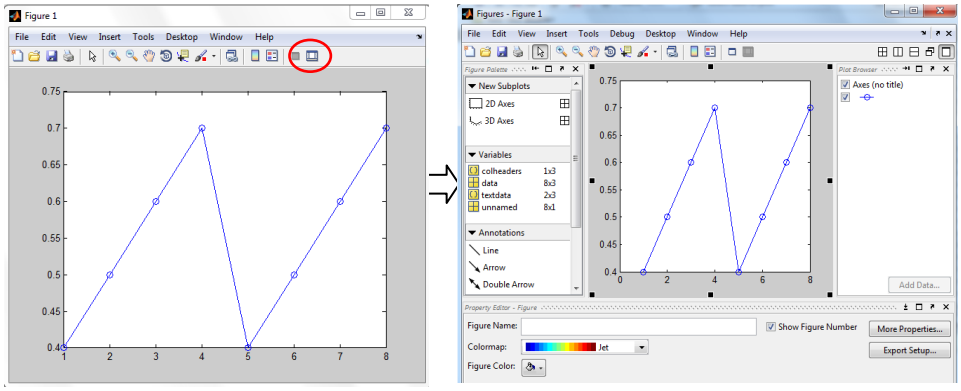
Parametr `'r'` definiuje się tekstowo jako złożenie kilku symboli. Symbole te opisane są szczegółowo w dokumentacji dla polecenia `plot`. Jako przykład Czytelnik może utworzyć wykres utworzonej wcześniej zmiennej:

Przykład 3.4 – programowe tworzenie wykresu

```
>> plot(unnamed,'-bo')
```

W powyższym przykładzie (3.4), parametr `'-bo'` określa linię ciągłą (`-`), kolor niebieski (`b`) oraz marker w kształcie kółka (`o`).

Po utworzeniu wykresu istnieje możliwość dostosowania jego wyglądu za pomocą narzędzia "*Plot Tools*" (lub w sposób programowy). Dostęp do tego narzędzia pokazany jest na rys.3.10. Jest to rozbudowany przybornik pozwalający na dowolną edycję wyglądu wykresu. Przykładowo możliwe jest dodanie opisów osi, zmiana koloru i rodzaju linii, dodanie legendy itd. W ramach ćwiczenia Czytelnik może dostosować wygląd utworzonego wcześniej wykresu do własnego gustu. W celu programowego dostosowania wykresu należy skorzystać z dokumentacji - często używane polecenia to: `xlabel`, `ylabel` (opisy osi x i y), `title` (tytuł osi wykresu), `legend` (dodanie legendy).



Rys. 3.10. Narzędzie "Plot Tools"

Bardzo często zachodzi potrzeba aby na jednym oknie wykresu umieścić kilka wykresów. MATLAB oferuje kilka sposób osiągnięcia tego celu.

Pierwszym z nich są tzw. „*subplots*", czyli możliwość umieszczenia na jednym oknie wykresu kilku niezależnych osi współrzędnych. Sposób interaktywny (za pomocą „*Plot Tools*") został w tym skrypcie pominięty ponieważ jest trywialny (wybór odpowiednich opcji z podsuwanych przez program narzędzia możliwości). W sposób programowy „*subplots*" tworzy się poleceniem:

```
subplot(m,n,k)
```

Polecenie to ma 3 argumenty wejściowe:

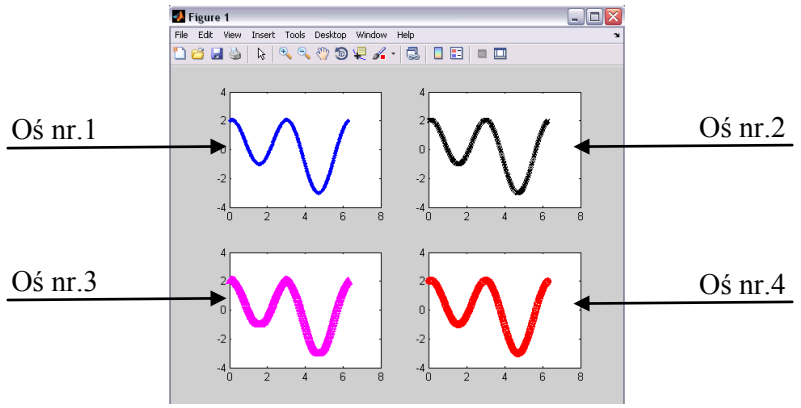
- `m, n` – oznaczają ilość części na jaką ma być dzielone okno w pionie i poziomie,
- `k` – oznacza numer osi która ma być uaktywniona (rys.3.11).

Przykład 3.5 – tworzenie wykresów (`subplot`)

```
>> x=0:pi/180:2*pi;      % tworzenie wektora x
>> y=sin(x)+2*cos(2*x); % tworzenie wektora y
>> subplot(2,2,1)        % oś wykresu nr.1
>> plot(x,y,'-b.')
>> subplot(2,2,4)        % oś wykresu nr.4
>> plot(x,y,'-ro')
>> subplot(2,2,2)        % oś wykresu nr.2
>> plot(x,y,'-kx')
>> subplot(2,2,3)        % oś wykresu nr.3
>> plot(x,y,'-m^')
```

Powyższy przykład (3.5) tworzy okno wykresu zawierające 4 osie, na każdej z nich tworzony jest ten sam wykres, ale innym kolorem. Wykonując ten

przykład warto obserwować rezultaty poszczególnych poleceń.

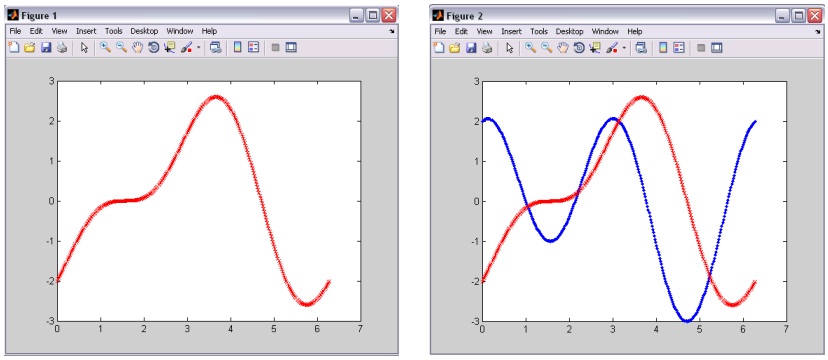


Rys. 3.11. Rezultat wykonania przykładu 3.5.

Jeżeli zachodzi potrzeba umieszczenia kilku wykresów na tej samej osi, konieczne jest zastosowanie polecenia `hold`. Polecenie to informuje środowisko MATLABa o tym, aby kolejne rezultaty poleceń graficznych były umieszczane na tym samym wykresie. Domyślnie MATLAB usuwa poprzedni wykres. Dla porównania czytelnik może wykonać następujący przykład (3.6):

Przykład 3.6 – tworzenie wykresów (`hold`)

```
>> figure                  % tworzenie okna wykresu
>> x=0:pi/180:2*pi;        % tworzenie wektora x
>> y1=sin(x)+2*cos(2*x); % tworzenie wektora y1
>> y2=sin(2*x)-2*cos(x); % tworzenie wektora y2
>> plot(x,y1,'-b.')
>> plot(x,y2,'-rx')      % nadpisanie wykresu innym

>> figure                  % tworzenie okna wykresu
>> plot(x,y1,'-b.')
>> hold on             % blokada nadpisania  wykresu
>> plot(x,y2,'-rx')    % nadpisanie wykresu innym
>> hold off            % wyłączenie blokady
```



Rys. 3.12. Rezultaty wykonania przykładu 3.6.

3.5. Udostępnianie rezultatów oraz automatyzacja pracy

Prezentacja wyników na wykresach (bądź też rezultatów numerycznych w linii poleceń) nie kończy pracy z danymi. Bardzo często zachodzi potrzeba wyeksportowania rezultatów i wykresów poza środowisko MATLABa w celu ich dalszej obróbki przy użyciu innych programów.

Eksport danych może być zrealizowany poprzez odpowiednie funkcje zapisu danych (`doc fileformats`). Najprostszą z nich jest funkcja `save`, pozwalająca na zapis danych z przestrzeni roboczej środowiska do pliku MAT (binarny format danych MATLABa). Obrazuje to przykład 3.7.

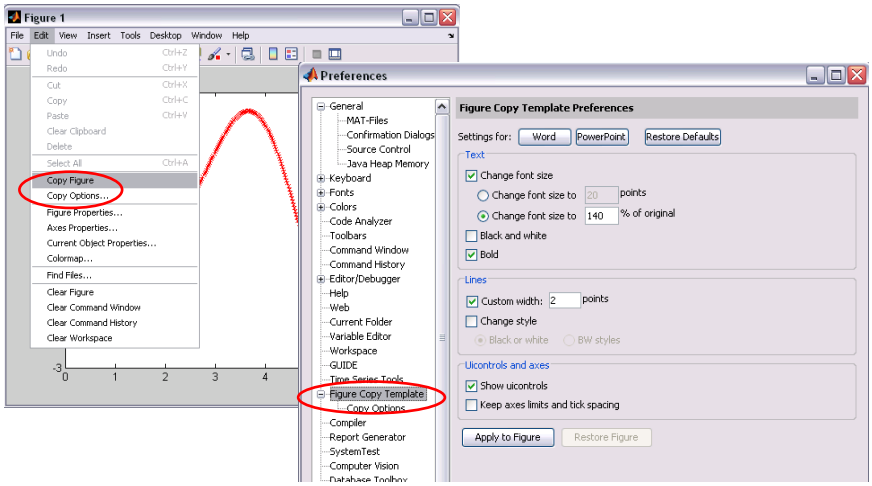
Przykład 3.7 – różne sposoby programowego eksportu danych

```
>> clear all              % czyszczenie przestrzeni
                          % roboczej

>> x=0:pi/180:2*pi;       % tworzenie wektora x
>> y=sin(x)+2*cos(2*x);   % tworzenie wektora y

>> save mojedane x y      % zapis zmiennych do pliku MAT
>> clear all
>> plot(x,y)              % błąd - brak zmiennych
Undefined function or variable 'x'.

>> load mojedane          % wczytanie danych z pliku MAT
>> plot(x,y)

% programowy eksport wykresu do pliku
>> saveas(gcf,'przyklad3-7.tiff','tiffn')
```

W przypadku wykresów eksport może być zrealizowany poprzez:

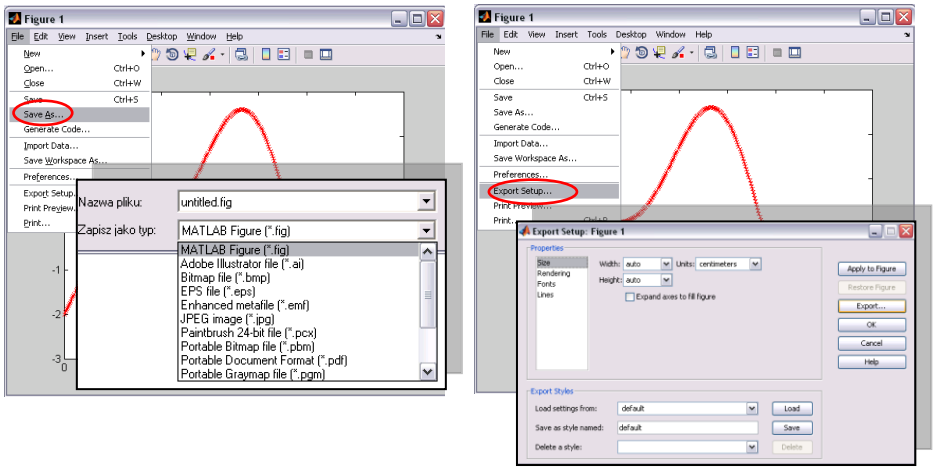
- Kopiowanie wykresu przez schowek – rys 3.13 – z menu „*Edit*” okna wykresu wybieramy polecenie „*Copy Figure*”. Wcześniej zalecane jest

ustawienie parametrów kopiowania poprzez wybór opcji „*Copy Options"*. Przykładowo można wybrać szablon kopiowania dla popularnych aplikacji takich jak Word lub PowerPoint ("*Figure Copy Template > Settings for"*).



Rys. 3.13. Kopiowanie wykresu przez narzędzie schowka, ustawianie preferencji

– Zapis do pliku graficznego za pomocą narzędzia eksportu danych (z menu „File" okna wykresu wybieramy polecenie „*Export setup…*" lub „*Save as…*", a następnie wskazujemy określony format pliku (rys. 3.14).



Rys. 3.14. Zapis wykresu do pliku graficznego.

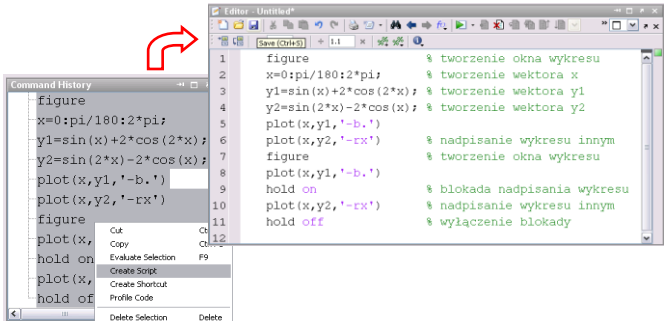
– Zapis do pliku poleceniem `saveas` lub `print` – są to programowe odpowiedniki ww. narzędzi (przykład 3.7).

- Wykorzystanie możliwości generacji raportów. Jest to jedna z bardziej interesujących możliwości środowiska MATLAB, służąca do automatycznego tworzenia raportów z uruchomienia M-plików. Zostanie ona omówiona dokładnie poniżej.

Eksport wykresów do plików graficznych nie jest jedyną możliwością udostępniania rezultatów. Użytkownicy środowiska MATLAB bardzo często wymieniają się nie tylko wynikami obliczeń, ale również tzw. M-plikami.

M-pliki są to pliki tekstowe z rozszerzeniem `*.m`, zawierające zapisane polecenia MATLABa, pozwalające odtworzyć wykonywane wcześniej obliczenia. Aby to zobrazować można posłużyć się przykładem 3.6. Po wpisaniu i wykonaniu poleceń, są one automatycznie zapamiętywane w zakładce historii poleceń („*Command History*"). Zakładka ta pozwala m.in. na szybkie powtarzanie komend (poprzez zaznaczanie i przeciąganie ich do okna poleceń) oraz na błyskawiczne skopiowanie zaznaczenia do edytora MATLABa (prawy klawisz myszki i wybór z okna opcji *Create Script*) – rys.3.15. Edytor MATLABa można również otworzyć korzystając z menu (*File>New>Script*).

Po skopiowaniu lub wpisaniu poleceń w edytorze należy go zapisać na dysku z wybraną nazwą (np. z nazwą `przyklad3_6.m`). Powstaje wtedy w aktualnym katalogu, tzw. m-skrypt. M-skrypt to inaczej zapamiętane komendy MATLABa wraz z kolejnością ich wykonania, zapisane w pliku tekstowym.



Rys. 3.15. Tworzenie m-skryptu – zakładka historii poleceń, edytor MATLABa

Utworzony w ten sposób m-skrypt może zostać uruchomiony poleceniem wskazującym jego nazwę (patrz przykład 3.8). Wszystkie zawarte w nim polecenia będę wtedy kolejno wykonywane.

Przykład 3.8 – uruchomienie m-skryptu z linii komend
`>> przyklad3_6`

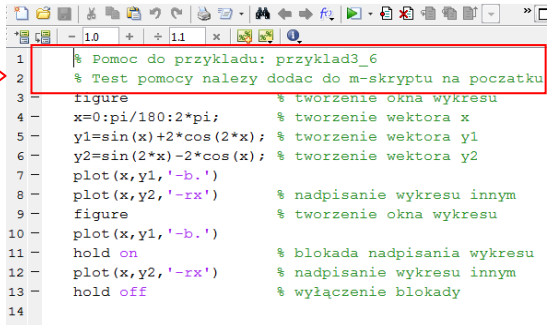
Tworzenie m-skryptów pozwala na automatyzację pracy oraz na wymianę wiedzy między użytkownikami.

Tworząc powyższy M-plik Czytelnik mógł zauważyć, iż oprócz poleceń środowiska zostały w nim dodane również komentarze. Była o nich wzmianka już wcześniej, ale teraz dodamy jeszcze kilka dodatkowych wyjaśnień. Komentarze zaczynają się od znaku `%` lub `%%`. MATLAB automatycznie koloruje składnię podkreślając komentarze na zielono. Dodawanie takich opisów jest bardzo istotne. Dzięki temu tworzona jest od razu prosta dokumentacja m-skryptu, pozwalająca na jego zrozumienie innym osobom (a także samemu twórcy jeśli zagląda do m-skryptu raz na kilka miesięcy). Szczególne znaczenie mają komentarze w pierwszych liniach M-pliku. Jako ćwiczenie Czytelnik może dodać do powyższego przykładu dodatkowy opis (3.9), a następnie w linii poleceń wpisać `help <nazwa_M-pliku>`.

Przykład 3.9 – tworzenie dokumentacji do m-skryptu

```
>> help przyklad3_6    % przed dodaniem opisu

No help found for przyklad3-6.m.
```

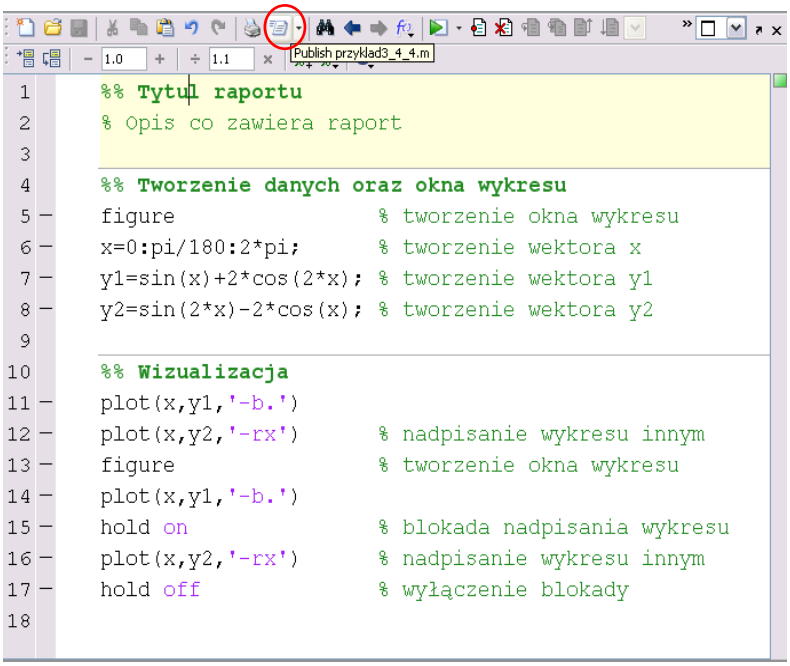
Uzupełnienie m-skryptu komentarzem



```
1   % Pomoc do przykladu: przyklad3_6
2   % Test pomocy nalezy dodac do m-skryptu na poczatku
3   figure                  % tworzenie okna wykresu
4   x=0:pi/180:2*pi;        % tworzenie wektora x
5   y1=sin(x)+2*cos(2*x); % tworzenie wektora y1
6   y2=sin(2*x)-2*cos(x); % tworzenie wektora y2
7   plot(x,y1,'-b.')
8   plot(x,y2,'-rx')        % nadpisanie wykresu innym
9   figure                  % tworzenie okna wykresu
10  plot(x,y1,'-b.')
11  hold on                 % blokada nadpisania wykresu
12  plot(x,y2,'-rx')        % nadpisanie wykresu innym
13  hold off                % wyłączenie blokady
14
```

```
>> help przyklad3_6    % po dodaniu komentarza

  Pomoc do przykladu: przyklad3_6
  Test pomocy nalezy dodac do m-skryptu na
  poczatku
```

Dodawanie komentarzy do m-skryptu ma również inne zastosowanie. Są one wykorzystywane przez narzędzie do generacji raportów. Ilustruje to poniższy przykład (rys.3.16). Komentarz oznaczony podwójnym znakiem procentu (`%%`) ma specjalne znaczenie – tworzy tzw. **celki** umożliwiające wykonywanie fragmentów m-skryptu. Naciśnięcie przycisku „*Publish*" spowoduje automatyczne uruchomienie m-skryptu, zebranie wszystkich wyników i wygenerowanie raportu w domyślnym formacie (HTML). Możliwe są również inne formaty (DOC, PPT, TeX, PDF…).



Rys. 3.16. Generacja raportów

3.6. Posługiwanie się wektorami i tablicami

W powyższych rozdziałach zostały omówione podstawowe sposoby użycia środowiska MATLAB, obejmujące import i wizualizację danych oraz automatyzację pracy i udostępnianie rezultatów. Aby Czytelnik mógł wykorzystać możliwości tego środowiska w większym stopniu, konieczne jest rozszerzenie informacji dotyczących pracy z wektorami i tablicami.

Podstawową jednostką danych w środowisku MATLAB jest tablica o określonej liczbie wymiarów (najczęściej 2), zawierająca dane w kolejnych komórkach. Pojedyncza liczba to tablica o rozmiarze 1x1 i można ją utworzyć przypisując liczbę do zmiennej. Do modelowania, bardzo często wykorzystywane są również tablice jednowymiarowe (wektory), pozwalające np. określić wektor czasu lub wektor odległości rozważanego punktu od jakiegoś wyróżnionego miejsca, istotnego dla danej symulacji. Przykłady tworzenia zmiennych i wektorów zostały zamieszczone poniżej (przykład 3.10).

Przykład 3.10 – tworzenie zmiennych i wektorów

```
>> p = 1.5       % przypisanie do zmiennej liczby,
                 % znak kropki jest separatorem
                 % dziesiętnym w MATLABie
p =

    1.5000
```

```
>> t = 1:4       % tworzenie wektora t w zakresie
                 % od 1 do 4, domyślna wartość
                 % kroku jest równa 1 - stąd ilość
                 % elementów wektora będzie równa 4

t =

     1     2     3     4

>> d = 0:0.5:2   % tworzenie wektora d w zakresie
                 % od 0 do 2 z krokiem 0.5 - ilość
                 % elementów wektora będzie równa 5
d =

          0    0.5000    1.0000    1.5000    2.0000

>> d1 = d'       % transpozycja wektora (zamiana
                 % wektora wierszowego na kolumnowy
d1 =

          0
     0.5000
     1.0000
     1.5000
     2.0000

>> k = linspace(1,3,4) % tworzenie wektora poprzez
     % podanie przedziału (w tym  przypadku od 1 do 3)
     % oraz ilości części na którą ten przedział jest
     % dzielony (tutaj 4)
k =

    1.0000    1.6667    2.3333    3.0000

>> x = 0:pi/180:2*pi; % tworzenie wektora zawierającego
     % wiele elementów; aby uniknąć wypisywania całego
     % wektora w oknie Command Window zastosowano
     % średnik umieszczony na końcu wyrażenia.
     % Wyrażenie "pi" pozwala na uzyskanie gotowej
     % wartości stałej matematycznej
```

Tworzenie tablic w MATLABie może być zrealizowane, albo poprzez ręczne ich wpisywanie, albo za pomocą odpowiednich funkcji do tworzenia typowych tablic. W przypadku ręcznego tworzenia tablic stosuje się **nawias prostokątny**. Poszczególne kolumny oddzielane są spacjami (lub przecinkami), natomiast wiersze – średnikami. Należy zwrócić uwagę, iż inne znaczenie ma umieszczenie średnika na końcu wyrażenia (opisane wyżej jako polecenie nie wyświetlania wartości powstającej w wyniku wykonania wyrażenia), a inne wewnątrz nawiasów prostokątnych (tutaj jest to – jak wspomniano – separator

wierszy tablicy). Różne techniki tworzenia tablic zostały przedstawione na poniżej (przykłady 3.11a-b). Studiując ten przykład warto zwrócić uwagę na nowe polecenia MATLABa: `randn`, `zeros`, `ones`, `eye`, `repmat` – które są w tym przykładzie zastosowane. Polecenia te nie były wcześniej omawiane, ale ich znaczenie w tak oczywisty sposób wynika z ich użycia, że nie zachodzi zapewne potrzeba dodatkowych wyjaśnień.

Przykład 3.11a – tworzenie tablic

```
>> A = [1 2 3;4 5 6;7 8 9] % tworzenie tablicy
                           % o wymiarach 3x3
A =

     1     2     3
     4     5     6
     7     8     9

>> B = [-1 2 0;3 5 0] % tablica o wymiarach 2x3

B =

    -1     2     0
     3     5     0

>> R1 = randn(2,4)
     % tworzenie tablicy 2x4 liczb pseudolosowych
     % o rozkładzie normalnym (średnia równa 0
     % i odchylenie standardowe 1)
R1 =

    0.5377   -2.2588    0.3188   -0.4336
    1.8339    0.8622   -1.3077    0.3426

>> R2 = 3+5*randn(2,4)
    % tworzenie tablicy 2x4 liczb pseudolosowych
    % o rozkładzie normalnym i zadanych parametrach
    % (średnia=3, odchylenie standardowe=5)
```

Przykład 3.11b – tworzenie tablic

```
>> Z = zeros(2,2)  % tworzenie tablicy 2x2 zer
Z =

     0     0
     0     0
```

```
>> O = ones(2,2)    % tworzenie tablicy 2x2 jedynek
O =

     1     1
     1     1

>> E = eye(2,2)     % tworzenie macierzy
                    % jednostkowej o rozmiarze 2x2
E =

     1     0
     0     1

>> F = repmat(B,2,1) % powielanie tablicy B, dwa
                     % razy w pionie i jeden raz
                     % w poziomie

F =

    -1     2     0
     3     5     0
    -1     2     0
     3     5     0
```

Więcej informacji na temat funkcji do tworzenia tablic Czytelnik znajdzie w dokumentacji MATLABa wpisując polecenie:
`help elmat`.

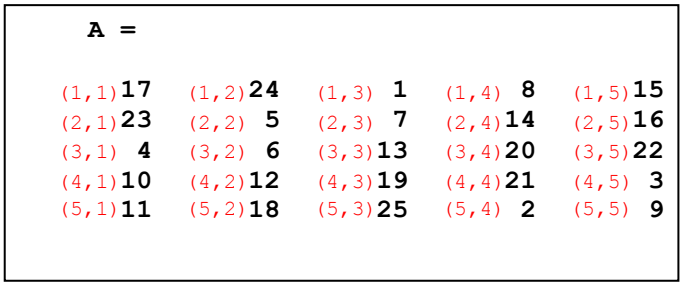
Tablice i wektory mogą być łączone ze sobą, tworząc w ten sposób nową tablicę. Istotne jest jednak zachowanie wymiarów łączonych elementów, tzn. odpowiadające sobie wymiary (wiersze lub kolumny) muszą być identyczne. Istotne również jest aby tablice były tego samego typu (np. double). W przeciwnym wypadku MATLAB zgłosi błąd. Obrazują to następujące przykłady (przykład 3.12).

Przykład 3.12 – łączenie tablic

```
>> A = [1 2 3;4 5 6;7 8 9];
>> B = [-1 2 0;3 5 0];
>> C = [2 4;5 9;0 1]
C =

     2     4
     5     9
     0     1
```

```
>> x = 1:3
x =

     1     2     3

>> D1 = [A;B]     % poprawne łączenie tablic
    % (ilość wierszy tablic A i B jest taka sama).
    % Średnik wewnątrz nawiasów prostokątnych pełni
    % rolę separatora wierszy tablicy)
D1 =

     1     2     3
     4     5     6
     7     8     9
    -1     2     0
     3     5     0

>> D2 = [A B]     % niepoprawne łączenie tablic
                  % (ilości kolumn są różne) - BLAD
Error using horzcat
CAT arguments dimensions are not consistent.

>> D3 = [A C]     % poprawne łączenie tablic
                  % (ilość kolumn tablic A i C
                  % jest taka sama
D3 =

     1     2     3     2     4
     4     5     6     5     9
     7     8     9     0     1

>> D4 = [A;x]     % poprawne łączenie wektora
                  % i tablicy
D4 =

     1     2     3
     4     5     6
     7     8     9
     1     2     3
```

Bardzo istotna podczas pracy w MATLABie jest znajomość sposobów indeksowania, czyli dostępu do elementów tablic. Podstawowym trybem dostępu jest indeksowanie typu wiersz-kolumna (patrz rys. 3.17 oraz przykłady 3.13a-c). W środowisku MATLAB indeksy rozpoczynają się od jeden, a nie tak jak np. w języku C od zera. Istnieje możliwość wyboru więcej niż jednego elementu z tablicy – w takim przypadku podawane jest kilka indeksów jednocześnie. Dopuszczalne jest także indeksowanie poprzez zmienną, tzn. utworzenie wektora liczb całkowitych i podanie go jako indeksów do tablicy.



Rys. 3.17. Indeksowanie wiersz-kolumna

Zmiana elementów tablicy odbywa się podobnie jak ich wybór – poprzez wyspecyfikowanie odpowiednich indeksów. Należy jednak pamiętać aby ilość elementów wstawianych do tablicy odpowiadała ilości podanych indeksów (wyjątkiem jest wstawianie do tablicy pojedynczej liczby).

Przykład 3.13a – indeksowanie typu wiersz-kolumna

```
>> A = magic(5); % tworzenie przykładowej tablicy
                 % (kwadrat magiczny) za pomocą
                 % odpowiedniej funkcji MATLABa.

>> a = A(2,3)    % wybór liczby z tablicy (wiersz
                 % nr.2, kolumna nr.3)
a =

     7

>> b = A(2,3:5)  % wybór trzech liczb z tablicy
                 % z wiersza nr.2
b =

     7    14    16
```

Przykład 3.13b – indeksowanie typu wiersz-kolumna

```
>> c = A(:,2)    % wybór drugiej kolumny z tablicy
c =

    24
     5
     6
    12
    18
```

```
>> d = A(2,[1 5])% wybór pierwszego i piątego
                 % elementu z wiersza nr.2
d =

    23    16

>> e = A(2,[3 3])% wybór dwa razy trzeciego
                 % elementu z wiersza nr.2
e =

     7     7

>> i = 1:2; f = A(2,i) % wybór z drugiego wiersza
                 % elementów wg numerów kolumn
                 % określonych w wektorze i
f =

    23     5

>> j = 0:1; g = A(2,j) % próba wyboru elementów
                 % określonych przez błędne
                 % indeksy w wektorze j (BŁĄD)
Subscript indices must either be real positive
integers or logicals.

>> A(2,3) = 5    % zmiana elementu tablicy
                 % (wiersz 2, kolumna 3)
A =

    17    24     1     8    15
    23     5     5    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

Przykład 3.13c – indeksowanie typu wiersz-kolumna

```
>> A(2,3:4) = [2 1] % zmiana dwóch elementów
                 % w tablicy (poprzez podanie 2-
                 % elementowego wektora)
A =

    17    24     1     8    15
    23     5     2     1    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

```
>> A(2,3:4) = 2  % zmiana dwóch elementów w tablicy
                 % (podanie pojedynczej liczby, która
                 % jest automatycznie powielana)
A =

    17    24     1     8    15
    23     5     2     2    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

>> A(2,3) = [2 1]% próba zmiany dwóch elementów
                 % w tablicy, ale błędne podanie tylko
                 % jednego indeksu)
Subscripted assignment dimension mismatch.
```

Oprócz indeksowania typu wiersz-kolumna istnieją jeszcze dwa sposoby dostępu do elementów tablicy. Pierwszy z nich to indeksowanie liniowe, w którym podawany jest pojedynczy numer kolejnego elementu. Sposób numerowania (kolejno kolumnami) przedstawiony jest na rysunku 3.18. Do konwersji indeksów pomiędzy obydwoma sposobami indeksowania służą polecenia: `ind2sub` (indeks liniowy na indeks typu wiersz kolumna) oraz `sub2ind` (indeks typu wiersz kolumna na indeks liniowy).

```
     A =

  (1)17    (6) 24   (11)  1   (16)  8   (21) 15
  (2)23    (7)  5   (12)  7   (17) 14   (22) 16
  (3) 4    (8)  6   (13) 13   (18) 20   (23) 22
  (4)10    (9) 12   (14) 19   (19) 21   (24)  3
  (5)11   (10) 18   (15) 25   (20)  2   (25)  9
```

Rys. 3.18. Indeksowanie liniowe

Kolejny sposób dostępu do tablic jest szczególnie użyteczny – jest to indeksowanie logiczne, pozwalające na wybór z tablicy elementów spełniających określony warunek logiczny. Informacje o operatorach logicznych oraz porównań można uzyskać w systemie pomocy (`help relop`). Warto tutaj zwrócić jednak uwagę na zapis następujących operatorów:

- alternatywa - znak `|`
- koniunkcja - znak `&`
- negacja - znak `~`
- porównanie czy A jest równe B: `A==B` (należy uważać przy używaniu tego

operatora ponieważ bardzo łatwo go pomylić z operatorem przypisania `A=B`).

Ideę indeksowania logicznego przedstawia przykłady 3.14a-b.

Przykład 3.14a – indeksowanie logiczne

```
>> A = magic(5);  % tworzenie przykładowej tablicy

>> I = A>5 & A<12 % warunek – rezultat to
                  % tablica prawda-fałsz
                  % o wymiarach takich samych jak
                  % wymiary tablicy A
I =

     0     0     0     1     0
     0     0     1     0     0
     0     1     0     0     0
     1     0     0     0     0
     1     0     0     0     1

>> a = A(I)       % wybór z tablicy A elementów
                  % spełniających zadany warunek
                  % (wykorzystanie tablicy I do
                  % indeksowania)
a =

    10
    11
     6
     7
     8
     9
```

Przykład 3.14b – indeksowanie logiczne

```
>> i = find(I)    % znalezienie indeksów liniowych
                  % dla elementów tablicy
                  % spełniających warunek
i =

     4
     5
     8
    12
    16
    25
```

```
>> [j,k] = find(I)% znalezienie indeksów
                  % wiersz-kolumna
                  % dla elementów tablicy
                  % spełniających warunek
j =

     4
     5
     3
     2
     1
     5

k =

     1
     1
     2
     3
     4
     5
```

3.7. Podstawowe operacje matematyczne i statystyczne

W przypadku operacji matematycznych (dodawanie, odejmowanie, logarytm, sinus itd.) MATLAB stosuje dany operator niezależnie dla każdego z elementów tablicy (przykład 3.15). Jest to wygodne ponieważ pozwala na szybkie otrzymanie rezultatów, bez konieczności stosowania instrukcji pętli. Lista funkcji matematycznych jest dostępna w dokumentacji (`help elfun`).

Przykład 3.15 – operacje matematyczne

```
>> A = magic(5); % tworzenie przykładowej tablicy

>> B = sin(A)    % wartości sinusów dla każdego
                 % elementu tablicy (traktowanego
                 % jako kąt w radianach)
B =

   -0.9614   -0.9056    0.8415    0.9894    0.6503
   -0.8462   -0.9589    0.6570    0.9906   -0.2879
   -0.7568   -0.2794    0.4202    0.9129   -0.0089
   -0.5440   -0.5366    0.1499    0.8367    0.1411
   -1.0000   -0.7510   -0.1324    0.9093    0.4121
```

Wykonanie operacji statystycznej (średnia, maksimum itd.) realizowane jest

przez środowisko w inny sposób. Wiersze tablicy traktowane są jako kolejne obserwacje, natomiast kolumny jako elementy wielowymiarowej zmiennej losowej (3.16). Lista funkcji statystycznych jest dostępna w dokumentacji (`help datafun`).

Przykład 3.16 – operacje statystyczne

```
>> A=[1 2 3 4;5 6 7 8;9 10 11 12]
                % tworzenie przykładowej tablicy

A =

     1     2     3     4
     5     6     7     8
     9    10    11    12

>> m = mean(A)  % wyznaczenie średniej dla tablicy
                % Rezultatem jest 4-elementowy
                % wektor zawierający średnie
                % z każdej kolumny macierzy A

m =

     5     6     7     8
```

Odpowiednikiem tablicy dwuwymiarowej w matematyce jest macierz. W środowisku MATLAB tablice mogą być również traktowane jako macierze. W przypadku dodawania i odejmowania nie ma to znaczenia (te operacje są wykonywane niezależnie dla każdego elementu tablicy). Różnice pojawiają się w momencie stosowania takich operatorów jak: mnożenie, potęgowanie oraz dzielenie. Ogólna zasada, pozwalająca odróżnić obydwa typy operacji od siebie, jest następująca. W przypadku zastosowania operatorów mnożenia oraz potęgowania (`*` `^`), wykonywane są one wg zasad rachunku macierzowego. Aby zrealizować te operacje jako tablicowe, należy zastosować kropkę przed operatorem (`.*` `.^` `./`). Ilustruje to przykład (3.17).

Przykład 3.17 – operacje tablicowe i macierzowe

```
%% tworzenie przykładowych tablic A,B,C
>> A=[1 2 3 4;5 6 7 8;9 10 11 12]

A =

     1     2     3     4
     5     6     7     8
     9    10    11    12
```

```
>> B=[5 4 3 2;9 8 7 6;1 2 3 5]
B =

     5     4     3     2
     9     8     7     6
     1     2     3     5

>> C=[3 4;5 8;9 10; 6 4]
C =

     3     4
     5     8
     9    10
     6     4

>> D=A.*B       % mnożenie tablicowe – istotna
                % zgodność wymiarów tablic
D =

     5     8     9     8
    45    48    49    48
     9    20    33    60

>> D=A.*C       % mnożenie tablicowe – błąd,
                % niezgodność wymiarów tablic
Error using  .*
Matrix dimensions must agree.

>> D=A*C        % mnożenie macierzowe – ok,
                % wg zasad rachunku macierzowego
D =

    64    66
   156   170
   248   274
```

Operator dzielenia (/ oraz \ ) w MATLABie ma specjalne znaczenie. Jego użycie pozwala na rozwiązywanie układów równań liniowych.

Rozwiązując układy równań, należy mieć na uwadze właściwą interpretację danych i rezultatów. Należy upewnić się z jakim rodzajem układu równań mamy do czynienia (oznaczony, nieoznaczony, sprzeczny itd.). Pomocna może w tym być funkcja `rank` podająca rząd macierzy głównej. Szczegóły dotyczące rozwiązywania układów równań liniowych może Czytelnik znaleźć w odpowiedniej literaturze poświęconej algebrze.

Zadanie rozwiązywane w przykładzie 3.18 polega na rozwiązaniu układu równań (3.1):

$$\begin{cases} x_1 + x_2 - x_3 = 1 \\ 2x_1 + x_2 + x_3 = 2 \\ x_1 - 3x_2 = 2 \end{cases} \tag{3.1}$$

Układ ten można zapisać w formie macierzowej (3.2):

$$\underbrace{\begin{bmatrix} 1 & 1 & -1 \\ 2 & 1 & 1 \\ 1 & -3 & 0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}}_{b} \tag{3.2}$$

Krótki zapis tego zadania ma więc postać (3.3):

$$A \cdot x = b \tag{3.3}$$

Sposób zdefiniowania tego zadania w MATLABie oraz polecenie jego rozwiązania przedstawiono w przykładzie 3.18.

Przykład 3.18 – rozwiązywanie układów równań liniowych

```
>> A = [1 1 -1; 2 1 1; 1 -3 0];
>> b = [1; 2; 2];
>> x = A \ b
x =

    1.1818
   -0.2727
   -0.0909
```

3.8. Typy danych dostępne w środowisku MATLAB

Domyślnym typem danych w środowisku MATLAB są liczby zmiennoprzecinkowe podwójnej precyzji (tzw. *double*). Oprócz tego MATLAB pozwala na pracę z innymi typami danych numerycznych – liczby zmiennoprzecinkowe pojedynczej precyzji (*single*) oraz liczby całkowite ze znakiem i bez znaku (uint8, uint16, uint32, uint64, int8, int16, int32, int64). Liczba w typie *uint/int* oznacza ilość bitów danego typu - np. *uint8* oznacza liczby całkowite bez znaku w zakresie 0-255. Wybór typu danych zależy od wymaganej dokładności obliczeń oraz ew. ograniczeń pamięci operacyjnej. Przykładowo – liczby typu `double` zajmują 8 bajtów, natomiast `uint8` tylko jeden. Warto pamiętać, iż precyzja obliczeń dla określonego typu danych nie jest tym samym co dokładność wyświetlania liczb w oknie poleceń (przykład 3.19).

Przykład 3.19 – numeryczne typy danych

```
>> format short     % ustawienie formatowania liczb
                    % wyświetlanych w oknie poleceń
                    % MATLABa na format skrócony
                    % (do 5 cyfr)

>> x1 = pi          % domyślny typ double
x1 =

    3.1416

>> x2 = single(pi) % konwersja na single
x2 =

    3.1416

>> format long      % ustawienie formatowania liczb
                    % wyświetlanych w oknie poleceń
                    % MATLABa na format rozszerzony
                    % (do 7 dla typu single lub
                    % 15 cyfr dla typu double)

>> x1               % ponowne wyświetlenie x1
x1 =

   3.141592653589793

>> x2               % ponowne wyświetlenie x2
x2 =

   3.1415927
```

Przy operacjach matematycznych na różnych typach danych, należy mieć na uwadze możliwość wystąpienia niepożądanych efektów. Przykładem może być przepełnienie, zaokrąglanie lub brak możliwości wykonania operacji na różnych typach danych (przykłady 3.20a-b).

Przykład 3.20a – operacje matematyczne na różnych typach danych

```
>> x3 = uint8(250) % typ całkowitoliczbowy,
                   % 8 bitów, bez znaku
x3 =

  250

>> x4 = uint8(50)  % typ całkowitoliczbowy,
                   % 8 bitów, bez znaku
```

```
x4 =

   50

>> x5 = int8(-20)  % typ całkowitoliczbowy,
                   % 8 bitów, ze znakiem
x5 =

  -20

>> y1 = x3 + x4    % dodawanie liczb UINT8 –
                   % skutkiem jest przepełnienie
y1 =

  255

>> intmin('uint8') % funkcje pomocnicze podające
                   % minimalną i maksymalną
                   % wartość dla typu
                   % całkowitoliczbowego
ans =

    0

>> intmax('uint8')
ans =

  255
```

Przykład 3.20b – operacje matematyczne na różnych typach danych

```
>> y2 = x3 + x5    % nie jest możliwe dodawanie
                   % różnych typów danych
                   % (całkowitoliczbowych)
Error using  +
Integers can only be combined with integers of the
same
class, or scalar doubles.

>> y3 = x3 + x1    % możliwe są jednak operacje
                   % pomiędzy pojedynczą liczbą double
                   % a tablicą liczb
                   % całkowitoliczbowych (występuje
                   % jednak zaokrąglanie wyniku)
y3 =

  253
```

Oprócz typów danych numerycznych, w środowisku MATLAB dostępne są inne rodzaje danych. Poniżej przedstawiono kilka z nich (przykład 3.21). Typ danych logicznych (przyjmujących wartości „prawda" albo „fałsz" co jest kodowane jako 1 lub 0 odpowiednio) powstaje po zastosowaniu do danych typu numerycznego lub znakowego operatorów logicznych – przykładowo: `==, ~=, >, <, >=, <=` .

Przykład 3.21 – tablice logiczne

```
>> a = [1 2 3]          % przykładowy wektor double
a =

     1     2     3

>> b = a > 2            % przykład wytworzenia
                        % wartości logicznej
                        % (prawda/fałsz, 1/0 )
b =

     0     0     1

>> a(b)                 % indeksowanie logiczne (wybór
                        % z tablicy "a" elementów
                        % spełniających warunek a>2)
ans =

     3
```

Tablice znakowe (przykład 3.22) służą do zapamiętywania tekstów, nazw parametrów, itp.

Przykład 3.22 – tablice znakowe

```
>> s1 = 'tablica'       % tablice znakowe
s1 =

tablica

>> s2 = 'znaków'
s2 =

znaków

>> s2(3)                % co będzie rezultatem tego
                        % polecenia?
ans =

a
```

```
>> s3 = [s1 ' ' s2]     % łączenie tablic znaków
                        % (poziomo)
s3 =

tablica znaków

>> s4 = [s1 ; s2]       % próba łączenia tablic znaków
                        % (pionowo) zakończona błędem
                        % ze względu na różne wymiary
                        % tablic
Error using vertcat
CAT arguments dimensions are not consistent.

>> s5 = strvcat(s1,s2) % prawidłowe łączenie tablic
                        % znaków o różnych wymiarach,
                        % przy użyciu odpowiedniej
                        % funkcji
s5 =

tablica
znaków
```

Dane w tablicach muszą być jednorodne, tzn. wszystkie elementy tablicy są tego samego typu. Często istnieje jednak potrzeba, aby w jednej zmiennej zapamiętać różne typy danych. MATLAB posiada dwa typy danych, które to umożliwiają. Pierwszym z nich są tablice komórkowe (ang. *cell arrays*). Są to tablice ogólnego przeznaczenia, pozwalające zapamiętać dowolne typy danych (przykłady 3.23a-b). Tablice komórkowe tworzone są z wykorzystaniem nawiasów klamrowych `{}`. Dla przypomnienia - zwykłe tablice tworzymy przy użyciu nawiasów prostokątnych `[]`. W przypadku tablic komórkowych, inaczej realizowane jest również indeksowanie do elementów takiej tablicy. Aby to wyjaśnić posłużmy się następującą analogią. Wyobrazimy sobie, że tablica komórkowa to taka szafka z szufladami. Jeśli do indeksowania zastosujemy nawiasy okrągłe `()`, to rezultatem będzie tablica komórkowa (inaczej wyjmiemy z szafki całą szufladę lub kilka szuflad). Jeżeli natomiast zastosujemy nawiasy klamrowe to rezultatem będzie zawartość danej komórki (czyli otworzymy szufladę i wyjmiemy tylko jej zawartość).

Przykład 3.23a – tablice komórkowe

```
>> C = {pi rand(3);-4 uint8(5)}
                    % tworzenie tablicy komórkowej
                    % stosowane nawiasy {},
                    % tablica zawiera w kolejnych
                    % komórkach: liczbę pi,
                    % tablicę 3x3, liczbę -4 oraz
                    % liczbę całkowitoliczbową 5
```

```
C =

    [3.1416]    [3x3 double]
    [    -4]    [          5]

>> a = C{1,2}       % wybór z tablicy komórkowej
                    % zawartości komórki (szuflady)
                    % o indeksie (1,2)
a =

    0.1270    0.0975    0.9575
    0.9134    0.2785    0.9649
    0.6324    0.5469    0.1576

>> b = C{1,1}       % wybór z tablicy komórkowej
                    % zawartości komórki (szuflady)
                    % o indeksie (1,1)
b =

    3.1416

>> c = C(1,1)       % wybór z tablicy komórkowej
                    % komórki (całej szuflady)!
                    % o indeksie (1,1)
c =

    [3.1416]
```

Przykład 3.23b – tablice komórkowe

```
>> b+c              % błąd – próba dodania liczby do
                    % komórki
Undefined function 'plus' for input arguments of type
'cell'.

>> class(b)         % sprawdzenie typu danych
ans =

double

>> class(c)         % sprawdzenie typu danych
ans =

cell
```

Kolejnym typem są **struktury**, szczególnie przydatne do grupowania

danych w formie czytelnej dla człowieka (patrz przykłady 3.24a-b). Definiuje się je poprzez podanie nazwy struktury oraz (po kropce) określenie pól w których będą zapamiętywane dane (lub z których te dane będzie się pobierać). Struktury o rozmiarze 1x1 (czyli zawierające tylko jeden rekord) nazywane są strukturami płaskimi. Możliwe jest tworzenie tablic struktur.

Przykład 3.24a – struktury

```
% struktura płaska "dane" zawierająca
% następujące pola: "x", "y", "param1"
>> dane.x = [1 2 3]
dane =

    x: [1 2 3]

>> dane.y = rand(1,3)
dane =

    x: [1 2 3]
    y: [0.9706 0.9572 0.4854]

>> dane.param1 = '-ro'
dane =

         x: [1 2 3]
         y: [0.9706 0.9572 0.4854]
    param1: '-ro'
```

Przykład 3.24b – struktury

```
% przykładowe użycie danych ze struktury
>> plot(dane.x, dane.y, dane.param1)

% uzupełnienie struktury o kolejny rekord
% (utworzenie tablicy struktur)

>> dane(2).x = [1 2 3];
>> dane(2).y = rand(1,3);
>> dane(2).param1 = '-bx'
dane =

1x2 struct array with fields:
    x
    y
    param1
```

```
% przykładowe użycie tablicy struktur (wybór drugiego
% rekordu)

>> hold on
>> plot(dane(2).x, dane(2).y, dane(2).param1)
>> hold off
```

Kolejny, bardzo użyteczny typ danych, to tzw. **uchwyty do funkcji** (ang. *function handles*). Są one wykorzystywane wszędzie tam, gdzie zachodzi potrzeba podania referencji do funkcji. Przykładem może być całkowanie numeryczne realizowane przez funkcję `quad`. W dokumentacji do funkcji (`help quad`), można zauważyć iż wymaga ona podania jako argumentu wywołania – funkcji całkowanej. W jaki sposób można to zrobić? Otóż właśnie za pomocą uchwytów do funkcji (przykład 3.25). Inne zastosowania uchwytów do funkcji to: optymalizacja, różniczkowanie, interfejsy graficzne itd.

MATLAB zawiera także inne, ciekawe typy danych - przykładem może być możliwość bezpośredniego użycia klas języka Java. Istnieje również możliwość tworzenia własnych typów danych - przy pomocy programowania obiektowego - które jest również wspierane przez środowisko MATLAB.
Omówienie tej tematyki wykracza poza potrzeby niniejszego skryptu. Więcej informacji na temat typów danych można znaleźć w dokumentacji MATLABa (`help datatypes`).

Przykład 3.25 – uchwyty do funkcji

```
>> f1 = @sin          % uchwyt do funkcji sin
f1 =

    @sin

>> f2 = @(x) x.^2-2*x % uchwyt do tzw. funkcji
                      % anonimowej,
                      % jednoargumentowej
f2 =

    @(x)x.^2-2*x

>> quad(f1,0,pi)      % użycie uchwytu do funkcji
                      % (całkowanie numeryczne)
ans =

    2.0000

>> fplot(f2,[0,pi])   % użycie uchwytu do funkcji
                      % (rysowanie wykresu)
```

3.9. Język programowania MATLAB

Oprócz możliwości automatyzacji pracy przy pomocy m-skryptów, MATLAB pozwala na tworzenie całych aplikacji. Jest to możliwe dzięki efektywnemu językowi programowania osadzonemu w tym środowisku, licznym modułom rozszerzającym oraz wielu narzędziom pomocniczym.

Język programowania MATLABa jest językiem wysokiego poziomu, zapewniającym między innymi automatyczne zarządzanie dynamiczną alokacją pamięci dla zmiennych (ang. *garbage collection*). Mimo iż powszechnie uważa się że jest to język interpretowany, to faktycznie środowisko zapewnia automatyczną analizę składni kodu w tle. Przyspiesza to w dużym stopniu wykonywanie M-kodu.

Szczegółowe omówienie składni i poleceń języka MATLABa nie jest potrzebne do zrozumienia przykładów zamieszczonych w niniejszym skrypcie. Większość informacji można znaleźć w dokumentacji zawierającej obszerne przykłady. Przydatne będzie jeśli Czytelnik zapozna się z instrukcjami warunkowymi (`if else`, `switch case`), pętlami (`for`, `while`) oraz poleceniami wyświetlania danych (`disp`, `fprintf`).

Istotne będzie jednakże rozróżnienie między **m-skryptami**, a **m-funkcjami**.

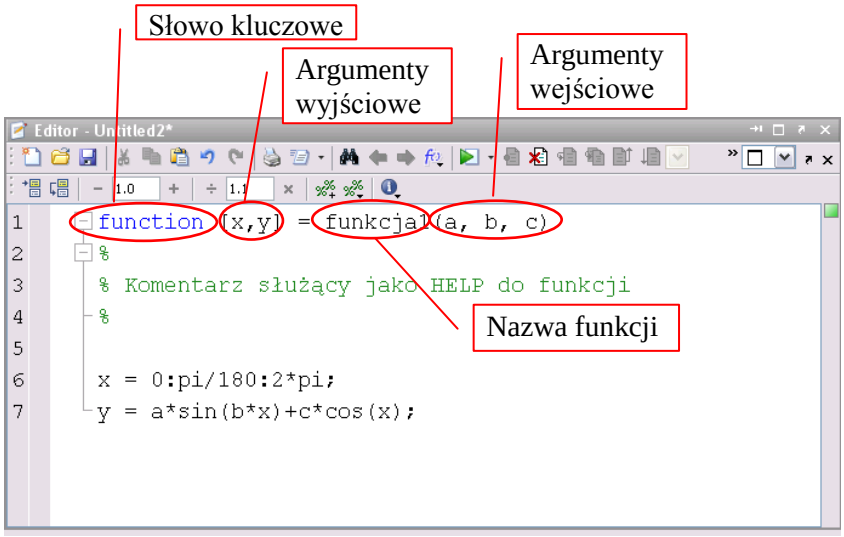
Co to są m-funkcje? Podobnie jak m-skrypty, są to pliki tekstowe zawierające polecenia MATLABa. To co odróżnia m-funkcje od m-skryptów to m.in.: składania oraz sposób zapamiętywania i dostępu do zmiennych w pamięci. Podczas uruchamiania m-skryptów, wszystkie zmienne są zapamiętywane w tzw. głównej przestrzeni roboczej MATLABa. Może to w pewnych przypadkach rodzić problemy, związane z nadpisywaniem zmiennych innymi wartościami Problem ten ilustruje przykład 3.26. M-funkcje z kolei, posiadają osobną przestrzeń roboczą – niezależną od głównej przestrzeni roboczej. Wymiana danych między przestrzeniami roboczymi odbywa się poprzez argumenty wejściowe i wyjściowe funkcji. Stąd wynika inna składnia M-pliku funkcyjnego (rys. 3.19). Przy zapisywaniu pliku istotne jest aby nazwa M-pliku odpowiadała nazwie funkcji.

Przykład 3.26 – przykład problemu nadpisywania zmiennej

```
>> clear all              % czyszczenie głównej
                          % przestrzeni roboczej

>> W([1 2 3])=[2 4 6]     % przypisanie wektora
```

```
W =

     2     4     6

>> W = 'abc'                % nadpisanie zmiennej
                            % innym typem danych
W =

abc

>> W([1 2 3])=[72 74 76] % ponowne przypisanie,
                            % wynik to
                            % nieoczekiwany rezultat
W =

HJL

>> class(W)
ans =

char
```

Uruchomienie m-funkcji odbywa się podobnie jak m-skryptu, poprzez wpisanie nazwy funkcji w oknie poleceń (przykład 3.27). Jedyną różnicą jest konieczność podania argumentów wejściowych i wyjściowych (zgodnie z definicją funkcji). Wszystkie zmienne używane wewnątrz funkcji są lokalne, tzn. istnieją tylko w czasie wywołania funkcji w jej lokalnej przestrzeni roboczej.

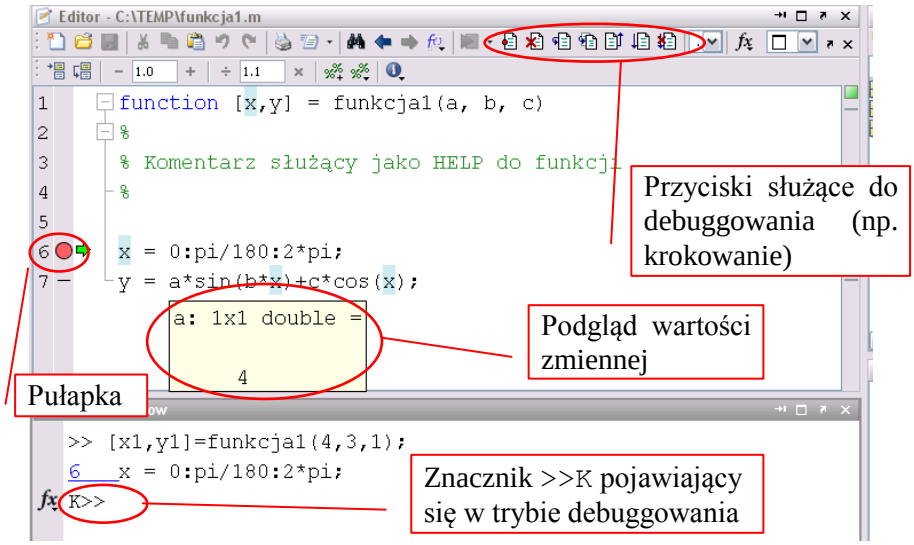


Rys. 3.19. Przykładowa składnia m-funkcji

Przykład 3.27 – przykład wywołania m-funkcji

```
>> clear all
>> [x1,y1]=funkcja1(4,3,1);
>> subplot(2,1,1);plot(x1,y1)

>> [x2,y2]=funkcja1(2,2,-1);
>> subplot(2,1,2);plot(x2,y2)

>> x              % błąd - zmienna x nie istnieje
                  % w głównej przestrzeni roboczej
Undefined function or variable 'x'.
```

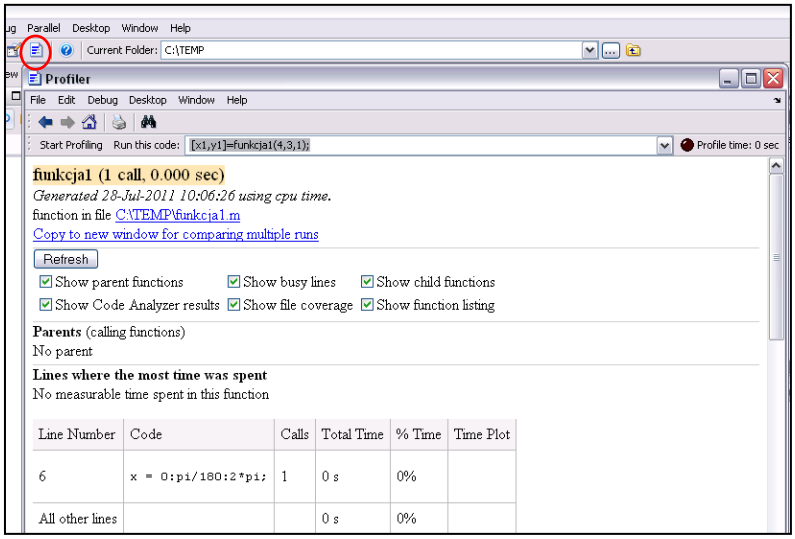
Jak każde środowisko programistyczne, MATLAB posiada narzędzia pozwalające na śledzenie wykonywania M-kodu (ang. *debugger*). Przy jego pomocy możliwe jest m.in. ustawianie pułapek (ang. *breakpoints*), krokowe wykonywanie kodu oraz podgląd i modyfikacja wartości zmiennych lokalnych w czasie wykonywania M-kodu. Okno debuggera zostało przedstawione na rysunku 3.20. Posługiwanie się tym narzędziem zostało dokładnie omówione w dokumentacji oprogramowania MATLAB.



Rys. 3.20. Przykład „debuggowania" m-funkcji

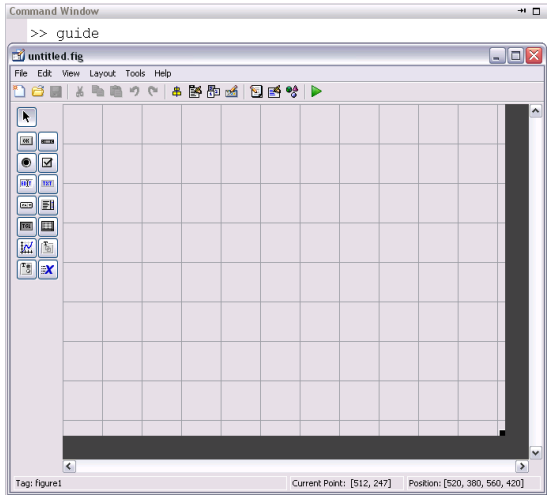
Środowisko MATLAB oferuje również szereg innych, przydatnych dla programisty narzędzi. Ich omówienie wykracza poza zakres tego skryptu, dlatego poniżej zostały one tylko wymienione i krótko scharakteryzowane. Wśród narzędzi możemy wyróżnić:

- **Profiler** – narzędzie pozwalające na analizę szybkości wykonania kodu oraz identyfikację potencjalnych problemów wydajnościowych (rys.3.21).



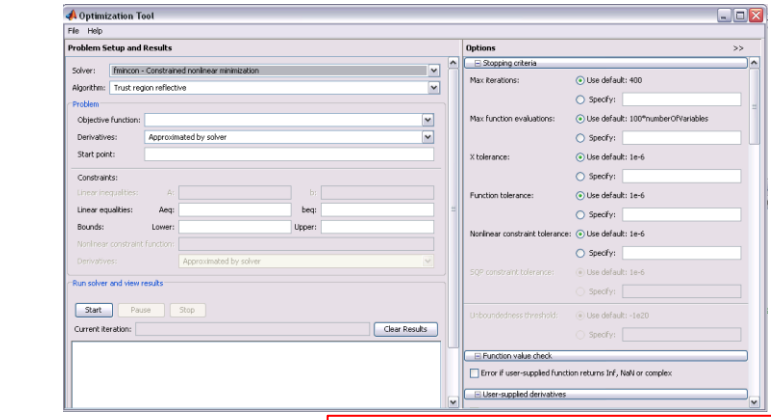
Rys. 3.21. Narzędzie Profiler

- **GUIDE** – narzędzie pozwalające w szybki sposób tworzyć interfejsy graficzne (GUI) w MATLABie (rys.3.22).



Rys. 3.22. Narzędzie GUIDE

– Narzędzia specjalistyczne modułów rozszerzających – prawie każdy moduł MATLABa zawiera graficzne narzędzia (rys.3.23), pozwalające bez znajomości poleceń danego modułu wygodnie go używać. Dodatkową zaletą tych narzędzi jest możliwość generacji M-kodu na podstawie zapamiętanych czynności.



Menu START środowiska MATLAB – zapewnia łatwy dostęp do narzędzi

Rys. 3.23. Przykładowe narzędzie z modułu *Optimization Toolbox.*

3.10. Przegląd metod numerycznego rozwiązywania równań różniczkowych w środowisku MATLAB

W podrozdziale 2.5. omówione zostały powody, dla których przy modelowaniu systemów dynamicznych z zasady sięgamy do równań różniczkowych jako do aparatu matematycznego który przy tym znajduje zastosowanie. Przypomnijmy tu wzór (2.26):

$$\frac{dY}{dt} = f(t, Y(t))$$

który definiuje typ równania, jaki będziemy chcieli dalej rozważać.

W podrozdziale 2.6. przedstawiono metody, które są używane do tego, żeby takie równania różniczkowe, które w samej swojej istocie odwołują się do funkcji ciągłych, mogły być rozwiązywane przez komputery przy zastosowaniu odpowiednich metod numerycznych. Napisanie własnego programu implementującego jedną ze znanych metod numerycznych jest możliwe (zajmiemy się tym w następnym podrozdziale), ale jest nieco kłopotliwe. Na szczęście środowisko MATLAB zawiera własne implementacje różnych metod numerycznego rozwiązywania równań różniczkowych. Wywołanie takiego gotowego algorytmu (nazywanego dalej *solverem*) przedstawia się następująco:

```
[wektor_czasu, rozwiazanie] = solver(@uchwyt_do_funkcji,
                              czas, warunek_poczatkowy,
                              opcje);
```

gdzie:

- `wektor_czasu` – to zwracany przez funkcję *solvera* wektor zawierający w formie numerycznej wartości odpowiadające kolejnym chwilom czasu, w których ustalone będą wartości funkcji będącej rozwiązaniem równania;
- `rozwiazanie` – to zwracany przez funkcję *solvera* wektor wartości numerycznego rozwiązania równania różniczkowego czyli wartości funkcji *Y(t)* w kolejnych chwilach czasu;
- `@uchwyt_do_funkcji` – jest to sposób wskazania funkcji *f(t, Y(t))* obliczającej prawą stronę równania różniczkowego; funkcja taka może być zapisana w M-pliku (i wtedy ma swoją nazwę wykorzystywaną jako *uchwyt*) lub może być zdefiniowana na potrzeby doraźnie jako funkcja anonimowa;
- `czas` – dwuelementowy wektor definiujący odcinek czasu, w którym chcemy uzyskać rozwiązanie rozważanego równania w postaci: `[czas_poczatkowy, czas_koncowy];`
- `warunek_poczatkowy` – warunek początkowy (długość tego wektora zależna jest od liczby rozwiązywanych równań) – jest to wartość *Y(t)* dla czasu: `t=czas_poczatkowy;`
- `opcje` – opcje numerycznego rozwiązywania równania (m. in. początkowy krok całkowania, dopuszczalne błędy metody); szczegóły można znaleźć w dokumentacji pod poleceniem: `odeset;`
- `solver` – pod tą nazwą kryje się zbiór dostępnych w środowisku MATLAB algorytmów (Tabela 3.1).

Tabela 3.1 Przegląd dostępnych w środowisku MATLAB solverów numerycznego rozwiązania równań różniczkowych zwyczajnych.

Nazwa solvera	**Metoda**
`ode23`	Bogacki-Shampine
`ode45`	Dormand-Prince
`ode113`	Adams-Bashforth-Moulton
`ode15s`	metoda oparta na formułach numerycznego różniczkowania (opcjonalnie metoda Gear'a)
`ode23s`	zmodyfikowana formuła Rosenbrocka drugiego rzędu
`ode23t`	implementacja wzoru trapezów
`ode23tb`	reguła TR-BDF2

Przykład 3.28 zawiera sposób wywołania wbudowanej funkcji środowiska MATLAB, wykorzystującej metodę Dormand-Prince dla przykładowego problemu początkowego (3.4):

$$\begin{cases} \dfrac{dY}{dt} = 5Y - 1 \\ Y(t_0) = 0.4 \end{cases} \tag{3.4}$$

Wcześniej jednak należy zdefiniować prawą stronę równania różniczkowego (3.4) w osobnym M-pliku `rownanie1.m` (Przykład 3.29), do którego uchwyt `@rownanie1` jest zdefiniowany w przykładzie 3.28 (`rozwiazanie1.m`).

W celu rozwiązania problemu początkowego określono przedział $t \in [0,1]$ oraz początkowy krok całkowania *h = 0.025*. Warunek początkowy, to *Y(0) = 0.4*.

Przykład 3.28 – Rozwiązanie równania różniczkowego (`rozwiazanie1.m`)

```
% Rozwiązanie równania różniczkowego dY/dt = 5Y - 1
% metodą Dormand-Prince
t0 = 0;     % czas początkowy symulacji
tk = 1;     % czas końcowy symulacji
h = 0.025; % krok rozwiązywania (początkowy)
Y0 = 0.4;  % warunek początkowy równania różniczkowego

% ustawienia solvera: początkowa długość kroku,
% tolerancja względna i bezwzględna
opcje = odeset('InitialStep', h, 'AbsTol', 1e-6,
'RelTol', 1e-4);

% rozwiązanie równania różniczkowego metodą przybliżoną
[wektor_czasu, rozwiazanie] = ode45(@rownanie1, [t0 tk],
Y0, opcje);

% wykres rozwiązania równania
plot(wektor_czasu, rozwiazanie);
title('Rozwiązanie numeryczne równania różniczkowego');
xlabel('Czas'); ylabel('Y(t)'); grid on
```

Przykład 3.29 – Równanie różniczkowe (`rownanie1.m`)

```
%  dY/dt = 5Y - 1
function [Y] = rownanie1(t, arg)
Y = 5*arg - 1;
```

W każdej iteracji algorytmu estymowany jest błąd $error_n$. Musi być on mniejszy lub równy od akceptowalnych wartości tolerancji względnej i bezwzględnej podanych przez użytkownika zgodnie z zależnością (3.5):

$$\left|error_n\right| \le \max(RelTol \cdot \left|Y(t_n)\right|, AbsTol) \tag{3.5}$$

gdzie:

- $\left|Y(t_n)\right|$ – wartość bezwzględna rozwiązania numerycznego w kroku *n*-tym
- $RelTol$ – tolerancja względna (ang. *relative tolerance*); wartość domyślna `1e-3`
- $AbsTol$ – tolerancja bezwzględna (ang. *absolute tolerance*); wartość domyślna `1e-6`

Użycie powyższych parametrów oznacza, że użytkownik oczekuje, iż wszystkie rozwiązania będą poprawne z tolerancją *RelTol* z wyjątkiem tych

rozwiązań, które są mniejsze niż próg definiowany przez *AbsTol*. Wartość *AbsTol* jest progiem poniżej którego rozwiązanie jest nieistotne dla rozważanego problemu.

AbsTol najczęściej jest wektorem, który posiada liczbę elementów równą liczbie równań różniczkowych rozwiązywanych przez *solver*.

3.11. Własna implementacja metody Rungego-Kutty IV rzędu

W celu praktycznej implementacji metody Rungego-Kutty IV rzędu posłużono się skryptami: `solve_rk4.m`, `rozwiazanie2.m` oraz zdefiniowanym w poprzednim przykładzie skryptem `rownanie1.m`.

Pierwszy z nich jest programową realizacją algorytmu Rungego-Kutty IV rzędu w postaci funkcji zwracającej ciąg dyskretnych chwil czasu oraz wartości rozwiązania równania (3.4) dla przedziału [0,1] i kroku $h = 0.025$ (Przykład 3.30).

Przykład 3.30 – Numeryczne rozwiązanie równania różniczkowego (`solve_rk4.m`)

```
function [t, Y] = solve_rk4(uchwyt, czas, Y0, h)
% czas - początkowa i końcowa chwila czasowa: [t0, tk]
% Y0 - warunki początkowe
% h - długość kroku

t0 = czas(1);           % początkowa chwila czasowa
tk = czas(2);           % końcowa chwila czasowa
t = czas(1):h:czas(2); % wektor czasu

liczba_krokow = ((tk - t0)/h); % liczba kroków algorytmu
Y = zeros(length(Y0), liczba_krokow+1); % wektor wyników
Y(:, 1) = Y0(:);               % warunek początkowy dla
                               % pierwszej iteracji

for i = 1:liczba_krokow
      F1 = h*uchwyt(t(i), Y(:,i));
      F2 = h*uchwyt(t(i) + h/2, Y(:,i) + 0.5*F1(:));
      F3 = h*uchwyt(t(i) + h/2, Y(:,i) + 0.5*F2(:));
      F4 = h*uchwyt(t(i) + h, Y(:,i) + F3(:));
      Y(:, i+1) = Y(:,i) + (F1(:) + 2*F2(:) + 2*F3(:) +
F4(:))/6;
end;
```

Przykład 3.31 – Wywołanie solvera Rungego-Kutty IV rzędu (`rozwiazanie2.m`)

```
t0 = 0;     % czas początkowy symulacji
tk = 1;     % czas końcowy symulacji
h = 0.025; % krok rozwiązywania
Y0 = 0.4;  % warunek początkowy

% Rozwiązanie równania różniczkowego metodą
% Rungego-Kutty IV rzędu
[wektor_czasu, rozwiazanie] = solve_rk4(@rownanie1, [t0
tk], Y0, h);

% Wykres rozwiązania równania
plot(wektor_czasu, rozwiazanie);
title('Rozwiązanie numeryczne równania różniczkowego');
xlabel('Czas');
ylabel('Y(t)');
grid on
```

Skrypt `rozwiazanie2.m` zawiera wywołanie metody numerycznej Rungego-Kutty IV rzędu w oparciu o przedstawioną powyżej zawartość pliku `solve_rk4.m`. Najważniejszym elementem skryptu jest konstrukcja `solve_rk4(@rownanie1, ...);` której celem jest uruchomienie procedury numerycznej rozwiązania równania różniczkowego (Przykład 3.31). Jako pierwszy argument funkcji `solve_rk4` przekazany zostaje uchwyt do funkcji zawartej w pliku `rownanie1.m`. Funkcja zawiera równanie różniczkowe (ewentualnie układ równań różniczkowych), które będzie rozwiązane numerycznie.

W zasobach internetowych książki zawarto również własną implementację metody Eulera (`solve_euler.m`, `rozwiazanie3.m`). Przygotowano również skrypt, który dokonuje porównania rozwiązania analitycznego oraz rozwiązań numerycznych metodami Eulera i Rungego-Kutty IV rzędu dla problemu początkowego (3.6):

$$\begin{cases} \dfrac{dY}{dt} = 5Y - t \\ Y(t_0) = 1 \end{cases} \tag{3.6}$$

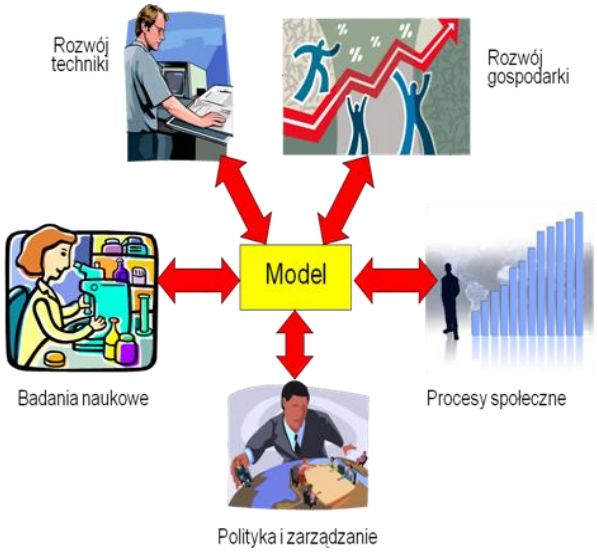
Porównanie zawarte jest w skrypcie `porownanie_rozwiazan.m`, zaś równanie w pliku `rownanie2.m`.

ROZDZIAŁ 4

MODELE WYBRANYCH SYSTEMÓW BIOLOGICZNYCH

4.1. Ogólna charakterystyka prezentowanych przykładów

W przedstawianym rozdziale **zilustrujemy** ogólne rozważania zawarte w poprzednich rozdziałach poprzez prezentację serii **przykładów** modeli oraz związanych z nimi programów symulacyjnych w środowisku MATLAB. Jako obiekty, które będą podlegać modelowaniu, brane będą pod uwagę różne systemy **biologiczne**. Wynika to między innymi z zainteresowań naukowych autorów tej książki, którzy wywodzą się z Laboratorium Biocybernetyki AGH i mają odpowiednie do tego faktu zainteresowania. W dodatku wybór właśnie modelowania systemów biologicznych jako „poligonu doświadczalnego", na którym ćwiczyć będziemy techniki modelowania i symulacji, ma swoje racjonalne uzasadnienie, które zostanie podane za chwilę.



Rys. 4.1. Przykładowe obszary zastosowań modeli (rysunek zmontowano z obrazków dostępnych jako MS ClipArt).

Jednak przed podaniem tego uzasadnienia warto sobie uświadomić, że okoliczność, iż prezentowane modele dotyczą właśnie systemów biologicznych, a nie czego innego, nie ma zasadniczego wpływu na sam sposób ich modelowania i komputerowej symulacji. W dokładnie taki sam sposób można modelować w razie potrzeby urządzenia techniczne, na przykład pojedyncze części maszyn, a także całe maszyny oraz systemy złożone z wielu maszyn, takie jak elektrownia jądrowa, rakieta kosmiczna czy autostrada pełna samochodów. W ten sam sposób można również modelować procesy ekonomiczne, zjawiska społeczne albo fenomeny psychologiczne, bo natura modelowanego systemu nie ma wpływu na sposób jego modelowania i na przebieg jego komputerowej symulacji.

Jak wiadomo, obiektem modelowania może być w istocie dowolny fragment rzeczywistości. Co więcej – obiektem modelowania może być także byt nie istniejący w rzeczywistości, lecz wytworzony przez wyobraźnię i fantazję badacza. Na przykład można stworzyć program symulujący zachowanie wymyślonych przez programistę kosmitów. Z możliwości tej skwapliwie korzystają między innymi producenci gier komputerowych, a także filmów animowanych, których produkcja obecnie bardzo silnie związana jest z grafiką komputerową, ale także z symulacją cyfrową zachowania (działania) dowolnie wybranego (wyobrażonego) obiektu.

My jednak skupimy tu uwagę na **modelowaniu systemów biologicznych.** Ma to swoje uzasadnienie. Gdybyśmy przedstawiali jako przykłady modele systemów technicznych, to z reguły trzeba by było najpierw opisać taki wybrany system techniczny (bo trudno zakładać, że każdy Czytelnik tej książki zna – na przykład - budowę i działanie silnika asynchronicznego). Trzeba by było też przypomnieć fizyczne podstawy rozważanego systemu, bo nawet w przypadku urządzeń technicznych z którymi jesteśmy oswojeni na co dzień, ich zasada działania dla wielu jest dość tajemnicza. Ilu spośród Czytelników książki potrafiłoby opisać procesy termodynamiczne, dzięki którym działa zwykła lodówka? Ponadto wszystkie niebanalne systemy techniczne są dość złożone. Wystarczy pomyśleć, z ilu części składa się zwykły samochód – że o telewizorze, telefonie komórkowym czy komputerze nie wspomnimy.

Na podobnej zasadzie można by było wykazać, że potrzebujemy wiedzy ekonomicznej aby zrozumieć cele i sposoby działania modeli oraz programów symulacyjnych wykorzystywanych w prognozowaniu procesów gospodarczych, wymagamy wiedzy socjologicznej od kogoś, kto chciałby śledzić szczegóły modeli wykorzystywanych w socjologii itp.

Tymczasem obiekty biologiczne w większości nie wymagają takiego zasobu początkowej wiedzy. Po prostu znamy je z codziennego doświadczenia. Każdy z nas ma ciało i wie, jak ono funkcjonuje. Każdy też wyniósł ze szkoły jakieś pojęcie o tym, jak są zbudowane poszczególne narządy i inne części tego ciała. Jeśli więc będziemy rozważać (w jednym z najbliższych podrozdziałów) modelowanie kości nogi lub ręki – to nie trzeba będzie osobno tłumaczyć, co to jest kość i jak wygląda w oryginale (a nie w modelu). Co więcej, jeśli będziemy rozważali za pomocą symulacji komputerowej możliwość złamania kości pod wpływem obciążenia, to każdy Czytelnik bez dodatkowych wyjaśnień będzie wiedział, co to jest złamanie kości, podczas gdy wyjaśnienie na czym polega na przykład uszkodzenie cewek stojana prądnicy synchronicznej wymagałoby pewnego wysiłku.

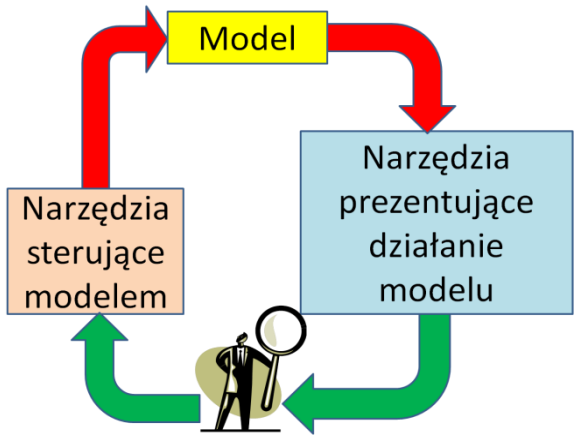
Będziemy więc dalej rozważali modele systemów biologicznych. Zgodnie z metodyką opisaną we wcześniejszych rozdziałach najpierw będziemy się starali rozważany obiekt opisać za pomocą formuł matematycznych, które będą ujmowały w sposób ilościowy interesujące (z punktu widzenia celów modelowania) zależności i relacje zachodzące w tym obiekcie. Następnie będziemy tworzyli dla tego matematycznego modelu odpowiedni program

w środowisku MATLAB, prowadząc do tego, by stworzyć narzędzie pozwalające modelowany obiekt badać w sposób symulacyjny.

Zaczniemy od modeli obiektów bardzo prostych, wręcz elementarnych, które będą modelami **statycznymi**, czyli będą opisywały zjawiska, w których czynnik czasu nie odgrywa istotnej roli. Potem jednak zmierzać będziemy do modeli bardziej złożonych systemów, a w szczególności do modeli uwzględniających **dynamikę** rozważanych systemów, czyli takich, które dostarczają rozwiązań będących procesami przebiegającymi w pewnym czasie. Będą one znacznie ciekawsze (model będzie „żył" i przejawiał pewną aktywność), ale też takie modele trudniej będzie zbudować.

4.2. Trzy składowe modelu symulacyjnego

Skoro w zakończeniu poprzedniego podrozdziału wspomniano o trudnościach, to warto odnotować, że przy budowie i przy opisywaniu modelu symulacyjnego pojawia się zawsze pewien praktyczny problem. Wynika on z tego, że model symulacyjny musi nie tylko **opisywać** modelowany system, ale dodatkowo powinien dostarczać narzędzi, z pomocą których tym systemem można **manipulować**, a także powinien być zaopatrzony w udogodnienia pozwalające na wygodne **obserwowanie** wyników symulacji, czyli na kontrolowanie działania modelu (Rys. 4.2).



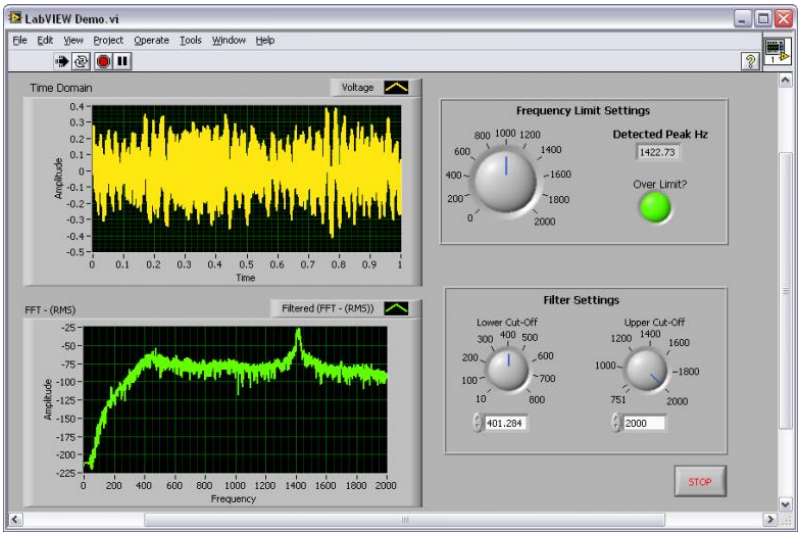
Rys. 4.2. Trzy części modelu symulacyjnego

Warto zwrócić uwagę na proporcje rozmiarów bloków symbolizujących poszczególne części modelu symulacyjnego. Ta część modelu symulacyjnego, która opisuje strukturę i zasady działania modelowanego systemu (blok u góry rysunku opisany słowem „model"), jest z reguły najmniejsza, chociaż to ona musi być wykonana najdokładniej, bo tu mieści się cała jego merytoryczna wartość. Na tym elemencie będziemy głównie skupiać uwagę we wszystkich przykładach przedstawianych w tym rozdziale. Tutaj nie ma mowy o żadnych ograniczeniach czy skrótach – model symulacyjny musi w całości odwzorować

wszystko to, co przewidziano i uwzględniono w modelu matematycznym. Jednak mimo tego, że właśnie część programu, określona jako „model", jest najważniejsza z punktu widzenia meritum – ma ona relatywnie mały rozmiar. Małej powierzchni odpowiedniego bloku na rysunku 4.2 odpowiada w programie komputerowym niewielka liczba dobrze zbudowanych instrukcji używanego języka symulacyjnego (w tej książce – MATLABa). Tę małą liczbę instrukcji stosunkowo łatwo jest przeczytać, zrozumieć, powiązać z wcześniej podanym modelem matematycznym – słowem łatwo się tego nauczyć. I bardzo dobrze, bo to jest właśnie ta najważniejsza cześć.

Odmienna sytuacja ma miejsce w odniesieniu do elementów sterujących pracą modelu a także w odniesieniu do elementów pozwalających na prezentację (wizualizację) jego działania. Są to dwa duże bloki w dolnej części rysunku 4.2, kontaktujące się z modelem i z umownie zaznaczonym u dołu rysunku „badaczem", który używa ich do praktycznego eksperymentowania z modelem.

Skupmy na chwilę uwagę na roli i znaczeniu tych dodatkowych elementów.



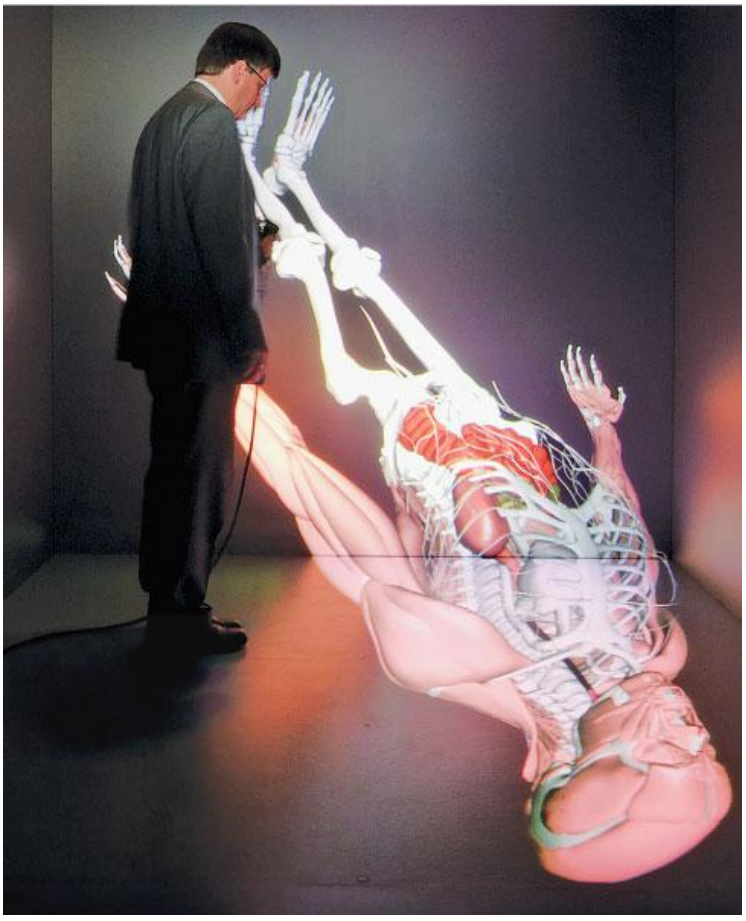
Rys. 4.3. Przykładowy moduł sterujący pracą symulacyjnego modelu stworzony w środowisku LabView (źródło: http://www.avrkwiat.nstrefa.pl/omnie/zdjecia/labprog.png, dostęp wrzesień 2011)

Dzięki narzędziom sterującym użytkownik programu symulacyjnego może wymuszać takie lub inne zachowanie modelu w takich lub innych symulowanych warunkach. Przypomnijmy tu, że użytkownik odnosi korzyść z modelu głównie dlatego, że może z nim eksperymentować, bo zwykle zastępuje to w znacznym stopniu eksperymenty z rzeczywistym systemem. Sterując modelem możemy się dowiedzieć, jak zachowa się modelowany system w takich lub innych warunkach, przy czym informację tę otrzymamy z modelu szybko, wygodnie, tanio i bezpiecznie. Eksperymenty symulacyjne są więc

wysoce pożądane. Przeprowadza się je wysyłając do modelu odpowiednie sygnały sterujące jego pracą. Z punktu widzenia użytkownika modelu generacja tych sygnałów powinna być łatwa, wygodna i intuicyjna, więc pożądane są tu różne widoczne na ekranie przyciski, pokrętła, suwaki i inne manipulatory, tworzące rodzaj **wirtualnego laboratorium**, w którym nasz model poddawany jest badaniom. Przykład takiego wirtualnego przyrządu pomiarowego zbudowanego w programie symulacyjnym LabView pokazano na rysunku 4.3.

Niestety oprogramowanie tych wszystkich sympatycznych gadżetów jest bardzo pracochłonne i może się zdarzyć, że część programu obsługująca te elementy manipulacyjne zajmuje więcej miejsca, niż merytoryczna część modelu. Zostało to już zasygnalizowane poprzez rozmiar odpowiedniego bloku na rysunku 4.2. Sposób przezwyciężenia tej trudności będzie dalej szczegółowo omówiony, tutaj sygnalizujemy sam fakt, że trudność taka ma miejsce.

Podobna sytuacja ma miejsce w odniesieniu do **wyjścia** modelu. Wyniki eksperymentów symulacyjnych trzeba móc obserwować, bo tylko w ten sposób można uzyskać wszystkie omawiane we wcześniejszych rozdziałach korzyści z posiadania modelu. Jednak wyniki te można obserwować w różny sposób. Użytkownik modelu chciałby otrzymać te wyniki w formie ładnych obrazów (dopracowanych graficznie, często trójwymiarowych – rys. 4.4.) a także – dla modeli systemów dynamicznych – dopracowanych technicznie animacji, które miło jest oglądać i z pomocą których wygodnie jest oceniać model. Tutaj wysokie wymagania jakościowe są jeszcze bardziej uzasadnione, niż omówione wyżej oczekiwania związane z jego sterowaniem, ponieważ wyłącznie w wyniku obserwacji wyjść modelu i ich oceny można się przekonać, czy zachowanie modelu rzeczywiście symuluje rozważany system. Oczywiście obserwacja wyników działania modelu symulacyjnego jest tym łatwiejsza i tym przyjemniejsza, im doskonalsza jest jakość wizualizacji tych wyników. Niestety opracowanie dobrej jakości modułów wizualizacji wyników symulacji jest jeszcze bardziej pracochłonne, niż omówione wyżej opracowanie graficznego interfejsu użytkownika po stronie wejścia modelu (patrz rozmiar odpowiedniego bloku na rysunku 4.2). Kreowanie grafiki wymaga dużego wysiłku, a stosowne fragmenty programu są obszerne, nawet przy używaniu bardzo nowoczesnych narzędzi komputerowych. I znowu pojawia się tu trudność polegająca na tym, że prezentując model mający rozbudowany moduł wyjściowy z zaawansowaną grafika możemy doprowadzić do tego, że w kodzie programu sam model będzie zajmował bardzo niewiele miejsca, natomiast znaczna część programu poświęcona będzie sprawie prezentacji graficznej, a więc sprawie w istocie drugorzędnej. W następnym podrozdziale pokażemy, jak przezwyciężamy tę trudność w tej książce.



Rys. 4.4. Prezentacja wyników symulacji może być nawet trójwymiarowa. (Źródło: http://kh-mmk.blogspot.com/2009/07/caveman-3-d-virtual-patient-is-holodeck.html, dostęp – wrzesień 2011)

4.3. Sposób prezentacji modeli

Wprowadźmy najpierw użyteczne pojęcie: Otóż narzędzia służące do manipulowania modelem i do obserwacji jego działania nazywane są ogólnie **interfejsem użytkownika**. Jak stwierdzono wyżej - zrobienie dobrego interfejsu użytkownika związane jest z dużym wysiłkiem. Przy niewielkich i prostych modelach interfejs użytkownika (jako fragment odpowiedniego programu) może wielokrotnie przekraczać swoimi rozmiarami i stopniem komplikacji tę część programu, która związana jest z samym modelem jako takim.

Przedstawiając w dalszej części tego rozdziału modele przygotowane dla potrzeb tego skryptu autorzy mieli problem. Z jednej strony tym, co chcieliśmy zaprezentować, jest **model** jako taki. Na szczęście istotna cześć programu symulującego działanie tego modelu daje się w większości przypadków

przedstawić w postaci kilku linii w MATLABie. Na tym właśnie powinna się skupić uwaga Czytelnika.

Z drugiej jednak strony istotą symulacji są eksperymenty, które można wykonać przy użyciu komputerowego modelu, więc do modelu trzeba dobudować interfejs użytkownika, pozwalający modelem operować. Tymczasem, jak stwierdzono wyżej, ten interfejs musi być bardzo duży (jako fragment odpowiedniego programu), jeśli ma być atrakcyjny i wygodny w użyciu. Taki duży fragment kodu opisującego interfejs w istocie zmniejszy znacząco czytelność samego modelu, na którym powinna się skupiać uwaga Czytelnika.

Zastosowano następujące rozwiązanie tego dylematu:

W tekście książki podawane są teksty programów (M-kody MATLABa) zawierające kompletny model symulujący określony system (biologiczny). Do tego pełnego modelu dodawany jest najbardziej ubogi, jak tylko się dało zrobić, absolutnie minimalny zestaw narzędzi pozwalający tym modelem operować na komputerze. Taki skrajnie zubożony interfejs użytkownika jest niewygodny w użyciu, ale jest **wystarczający** do tego, żeby model dało się uruchomić i przebadać. Dlatego taki właśnie prosty i ubogo wyposażony interfejs oraz **kompletny** model rozważanego systemu biologicznego Czytelnik może wpisać do swojego komputera, uruchomić i zbadać jego działanie.

Ktoś z Czytelników może się żachnąć w tym momencie. Ręcznie przepisywać do komputera program wydrukowany w książce – to przecież anachronizm! Czyż nie można było dodać do książki CD z tekstami wszystkich występujących w niej programów, żeby wystarczyło program wgrać do komputera albo łatwo i wygodnie skopiować, a nie męczyć się jakimś przepisywaniem!

Otóż zastosowane rozwiązanie jest przemyślane i celowe. Kopiując automatycznie tekst programu z jednego miejsca do drugiego niczego się nie nauczymy. Strumień instrukcji przepłynie poza naszym mózgiem. Tymczasem proces ręcznego wpisywania M-kodów prezentowanych dalej przykładowych programów jest tą chwilą, kiedy Czytelnik powinien zastanowić się, dlaczego to jest właśnie tak zapisane, jaką rolę pełnią poszczególne składniki modelu, jaka jest zależność między modelem wyrażonym w formie równań matematycznych i modelem w postaci zapisów komputerowych itp. Z tego właśnie powodu teksty odpowiednich programów w M-kodzie są wydrukowane w książce, ale nie są dostępne w formie plików tekstowych na CD czy w Internecie, bo od mechanicznego kopiowania tekstu (na zasadzie Ctrl-C - Ctrl-V) się nie mądrzeje, podczas gdy przepisywanie tekstu połączone z chwilą refleksji „czemu właśnie tak” – takiemu mądrzeniu zdecydowanie sprzyja.

Żeby jednak nie pozostawiać Czytelnika tylko z tym „zgrzebnym” modelem ręcznie przepisanym z książki - dla każdego takiego modelu stworzono także drugą wersję. Ta druga wersja dostępna jest na stronie internetowej

www.Tadeusiewicz.pl w zakładce **Materiały dydaktyczne dla studentów.** Wersja ta pod względem budowy merytorycznego modelu jest identyczna z tym modelem, który jest opisany na stronach skryptu, ale jest ona wyposażona w różne elementy graficznego interfejsu (a także dodatkowe obrazki podnoszące jej atrakcyjność), dzięki czemu eksperymentowanie z tym modelem może być łatwe, wygodne i przyjemne.

Wyjaśniwszy tę kwestię możemy przystąpić do prezentacji kolejnych modeli, zaczynając od najprostszych, a kończąc na takich, które mogą być nawet użytkowane w celach praktycznych.

4.4. Symulacja modelu kości, badanie wielkości naprężeń i sposobu poruszania się zwierzęcia

4.4.1. Zawartość podrozdziału

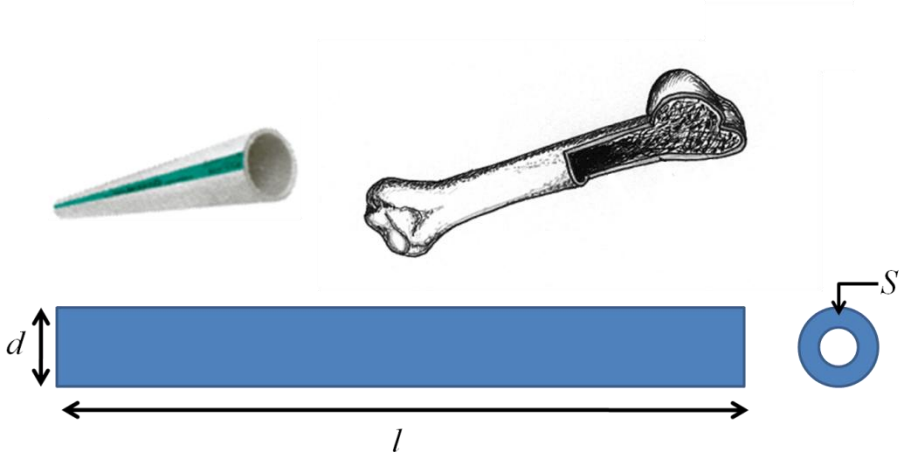
W niniejszym rozdziale opisany został najprostszy model biocybernetyczny, mianowicie model kości podlegającej statycznemu obciążeniu stałą siłą o ustalonej wartości. Jest to model skrajnie prostego systemu biologicznego, na bazie którego poznamy sposób tworzenia modelu i sposób przenoszenia modelu z formy matematycznej do formy użytecznej komputerowo.

4.4.2. Tworzenie modelu

W tej części rozdziału wyprowadzono podstawowe wzory umożliwiające wyliczenie naprężeń osiowych i poprzecznych w kościach oraz zdefiniowano współczynnik „zapasu wytrzymałości”. Opracowany model został w dalszej części rozdziału zaimplementowany w postaci M-pliku.

4.4.2.1. Model kości

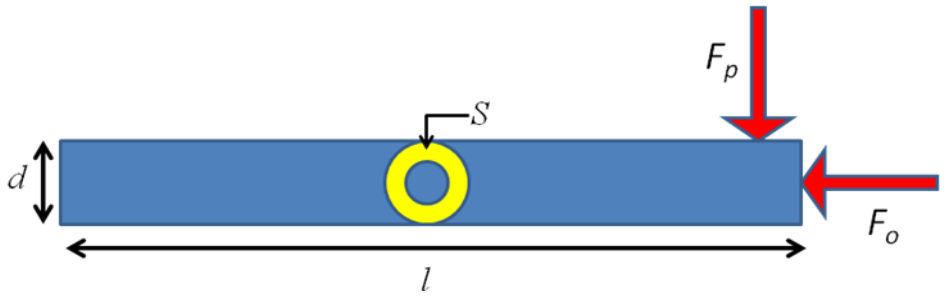
Dla celów modelowania zastąpiono rzeczywisty kształt kości rurą o określonej długości l, średnicy d i polu przekroju S (Rys. 4.5).



Rys. 4.5. Model kości

4.4.2.2. Wielkości naprężeń

Kość może zostać obciążona siłą osiową F_o lub siłą poprzeczną F_p (Rys. 4.6).



Rys. 4.6. Model kości poddany działaniu siły osiowej i poprzecznej

Efektem działania siły osiowej F_o będzie powstanie naprężenia ściskającego, które można wyrazić wzorem:

$$\sigma_{\mathrm{o}} = \frac{F_o}{S} \tag{4.1}$$

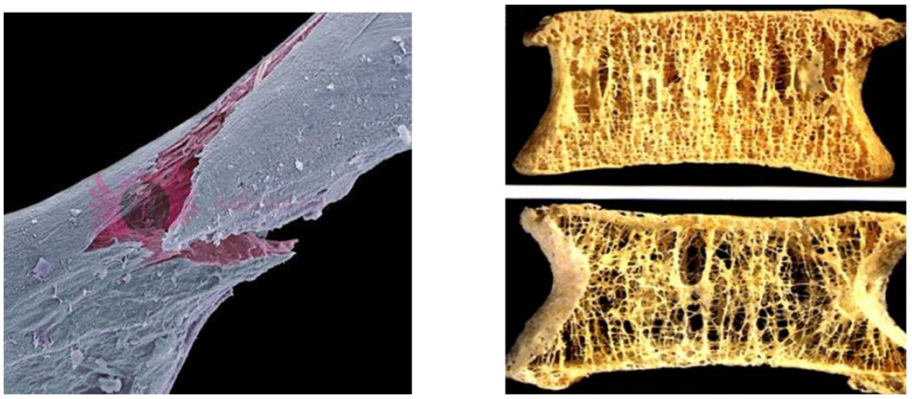
Natomiast w wyniku działania siły poprzecznej powstanie naprężnie zginające, które można wyrazić wzorem:

$$\sigma_{\mathrm{p}} = \frac{F_p l}{Sd} \tag{4.2}$$

Długość kości jest zwykle większa od jej grubości, dlatego naprężenia zginające σ_p wywołane siłą poprzeczną jest większe od naprężenia ściskającego σ_o:

$$\sigma_p >> \sigma_o \tag{4.3}$$

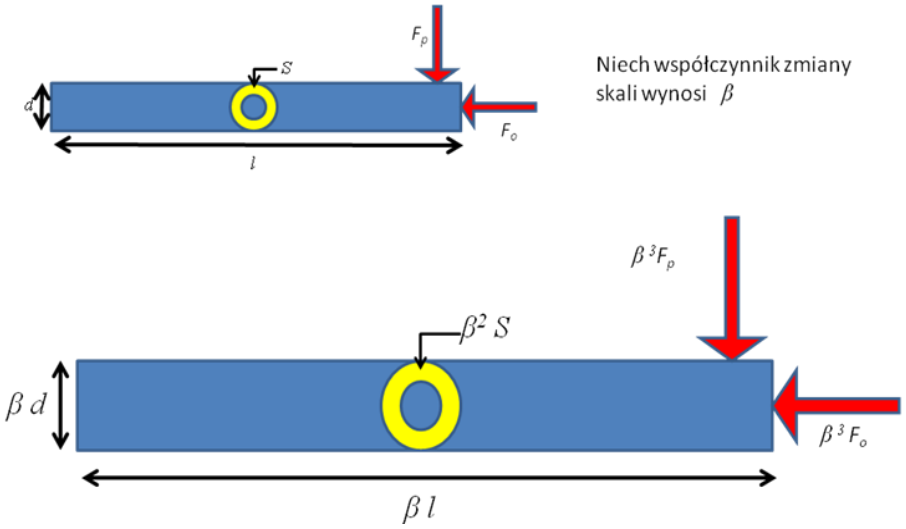
Porównując naprężenia zginające σ_p z naprężeniem ściskającym σ_o można wywnioskować, że kości częściej się łamią, niż ulegają zgnieceniu (Rys. 4.7).



Rys. 4.7. Elementy kostne znacznie częściej są złamane (po lewej) niż zgniecione (po prawej). (Źródło: http://www.feelbeauty.eu/news/show/id/3/osteoporoza-co-o-niej-wiesz.html, dostęp lipiec 2011)

4.4.3. Wielkość naprężeń, a rozmiar zwierzęcia

Możliwe jest oszacowanie w jaki sposób zmienia się wielkość naprężeń wraz z wielkością zwierzęcia. Wprowadzając współczynnik zmiany skali modelu β otrzymujemy (Rys. 4.8):



Rys. 4.8. Model kości poddany działaniu siły osiowej i poprzecznej z uwzględnieniem zmiany skali

Po uwzględnieniu współczynnika zmiany skali modelu β we wzorach na naprężenia ściskającego σ_o i naprężenia zginające σ_p otrzymujemy:

$$\sigma_{\mathrm{ow}} = \frac{\beta^3 F_o}{\beta^2 S} = \beta\sigma_o \tag{4.4}$$

$$\sigma_{\mathrm{pw}} = \frac{\beta^3 F_p \beta l}{\beta^2 S \beta d} = \beta\sigma_p \tag{4.5}$$

Na podstawie analizy naprężeń z uwzględnieniem skali można wywnioskować, że kości większych zwierząt poddawane są większym naprężeniom, zatem, aby je zminimalizować słoń musi chodzić na prostych nogach (Rys. 4.9), natomiast małe gryzonie mogą chodzić na nogach silnie ugiętych (Rys. 4.10).

Rys. 4.9. Słoń na prostych nogach

Rys. 4.10. Gerbil na ugiętych nogach

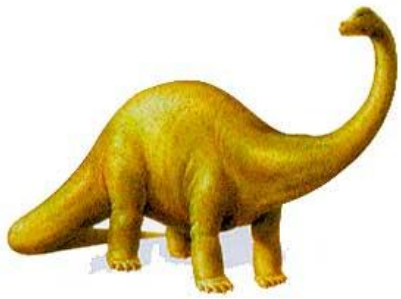
4.4.4. Zapas wytrzymałości

Na podstawie podobnej analizy można ustalić jak chodziły dinozaury.

Wskaźnik określający „zapas wytrzymałości" kości dla zwierzęcia o masie ciała M definiujemy następująco:

$$\eta = \frac{Sd}{Ml} \tag{4.6}$$

Zwierzęta mające mały wskaźnik η muszą się poruszać powoli, bo dynamiczny bieg grozi im złamaniem kości. Słoń ma $\eta = 7$ a bardzo dynamiczny gepard ma $\eta = 22$. Natomiast dla dinozaura o nazwie *Diplodokus* $\eta = 3$.



Rys. 4.11. Dinozaur o nazwie *Diplodokus* mógł się tylko wolno poruszać ze względu na kości (Źródło: http://lamochila.espectador.com/media/xnwslite//1237238017_245_diplodocus.jpg, dostęp lipiec 2011)

4.4.5. Implementacja podstawowego modelu

Dokonano implementacji modelu kości w Matlabie w postaci M-pliku.

Uwaga: Wprowadzając tekst podanego niżej M-kodu można pomijać wszystkie komentarze zaczynające się – jak wiadomo – od znaku %. W książce komentarze są potrzebne, bo dzięki nim Czytelnik łatwiej się orientuje, jaką rolę odgrywają takie czy inne fragmenty M-kodu. Natomiast w komputerze są one zbędne. Odpowiedni program będzie doskonale funkcjonował także bez nich. No więc można je opuścić – chyba że Czytelnik ma jakieś sobie znane powody, by tego nie robić.

Budując model (w postaci M-kodu) na początku należało ustawić parametry modelu takie jak długość kości, średnicę kości i pole poprzecznego przekroju kości. Te ustawienia pełnią rolę prymitywnego interfejsu użytkownika, bo z ich pomocą można sterować zachowaniem modelu (Przykład 4.1).

Przykład 4.1 – Definiowanie parametrów modelu

```
M = 120;           % masa ciała [kg]
l = 0.4;           % długość kości [m]
d = 0.05;          % średnica kości [m]
S = 0.2*pi*d^2/4; % 20% przekroju pełnego [m2]
```

Następnie dokonano obliczenia sił działających na kość oraz naprężeń ściskających i zginających. To jest ten właściwy model naszego systemu biologicznego (Przykład 4.2).

Użytkownicy mogą poprzez modyfikację pliku dowolnie sterować parametrami modelu, a korzystając ze standardowych możliwości MATLABa mogą odczytywać i porównywać rezultaty. To jest ta najbardziej „ascetyczna" postać modelu.

Przykład 4.2 – Treść modelu

```
% Obliczenia wartości sił działających na kość
% nogi ustawionej pod katem alfa do pionu:

F = 9.81*M/4;        % ciężar przypadający na jedną nogę
alfa = pi/6;         % kąt ustawienia nogi [rad]
Fo = F*cos(alfa);    % siła ściskająca
Fp = F*sin(alfa);    % siła zginająca

% Obliczenia wartości naprężeń ściskającego i
% zginającego
sigma0 = Fo/S;        % naprężenie ściskające
sigmap = Fp*l/(S*d); % naprężenie zginające

% Obliczenia wartości współczynnika zapas wytrzymałości
ni = S*d/(M*l);
```

4.4.6. Prezentacja wzbogaconego modelu

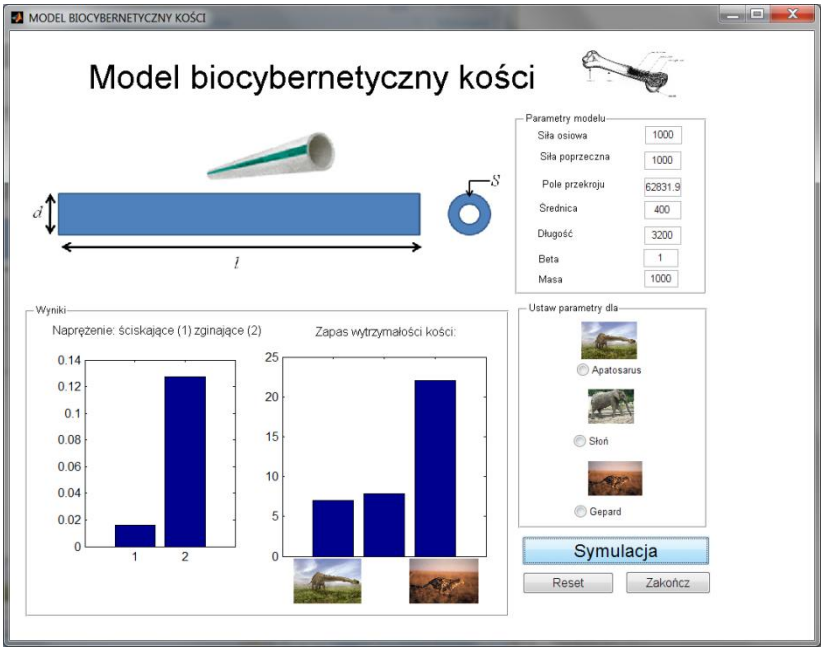
Na stronie www.Tadeusiewicz.pl dostępna jest także rozszerzona wersja rozważanej tu aplikacji, umożliwiająca wygodne dobieranie parametrów modelu oraz graficzną wizualizację otrzymanych danych (Rys. 4.12).

W celu łatwej nawigacji przygotowano graficzny interfejs użytkownika, który został podzielony na 3 części:

1. W prawym górnym rogu okna - ustawianie parametrów zdefiniowanych przez użytkownika takich jak:

    - Siła osiowa
    - Siła poprzeczna
    - Pole przekroju kości
    - Średnica kości,
    - Długość kości,
    - Współczynnik skali zwierzęcia (zmiana przeskalowuje całość),
    - Masa zwierzęcia.

    Możliwe jest także skorzystanie z gotowych propozycji w ramce poniżej panelu ustawianie parametrów. Pole wyboru pozwala na zastosowanie wartości parametrów zdefiniowanych wstępnie dla trzech zwierząt: geparda, słonia i dinozaura. Wybór jednej z tych opcji powoduje, że ustawione zostają automatycznie na właściwe wartości wszystkie rozmiary, siły i masy w celu zamodelowania obciążenia kości tych właśnie wybranych zwierząt.



Rys. 4.12. Okno wyników symulacji dla wersji modelu z rozbudowanym interfejsem użytkownika

2. W prawym dolnym rogu okna dostępne są przyciski kontrolne umożliwiają uruchomienie symulacji lub zakończenie aplikacji.

3. W lewej części okna odbywa się wyświetlanie wyników symulacji w postaci graficznej:

    - U góry przypomniany jest rozważany model kości
    - U dołu pokazany jest wykres naprężeń osiowych i poprzecznych

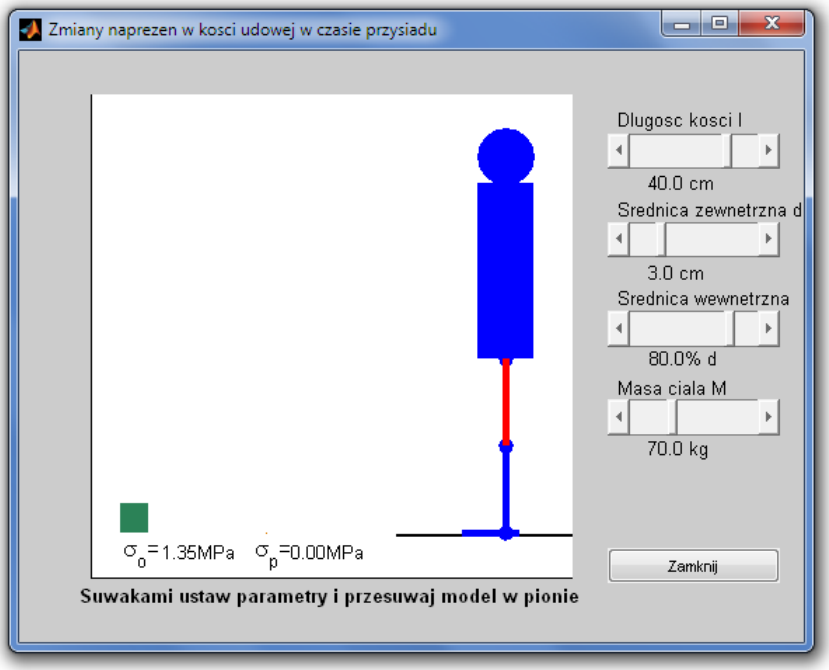
Takiego modelu wygodniej się używa, ale obejrzawszy M-kod dostępny na stronie www.Tadeusiewicz.pl można się przekonać, jakim kosztem się to odbywa: trzeba napisać naprawdę wiele dodatkowych instrukcji, znacząco powiększając złożoność tworzonego programu, tylko po to, żeby uzyskać takie ładne obrazki jak na rysunku 4.12.

4.4.7. Przykład modelu użytkowego – symulacja przysiadu

Model opisany wyżej nawet w wersji „luksusowej” był modelem odwołującym się do sytuacji trochę abstrakcyjnej, bo modelowana kość była obciążana jakimiś tam siłami, przy czym nie do końca można było powiązać te siły i ich działanie z konkretnymi sytuacjami znanymi nam z życia codziennego. Tymczasem modelowanie i symulacja komputerowa są obszarem informatyki

stosowanej, więc powinniśmy się odnieść do jakiegoś konkretu. Dodamy więc teraz do naszego modelu kości model jej zmiennego obciążenia związanego z wykonywaniem prostej czynności: przysiadu.

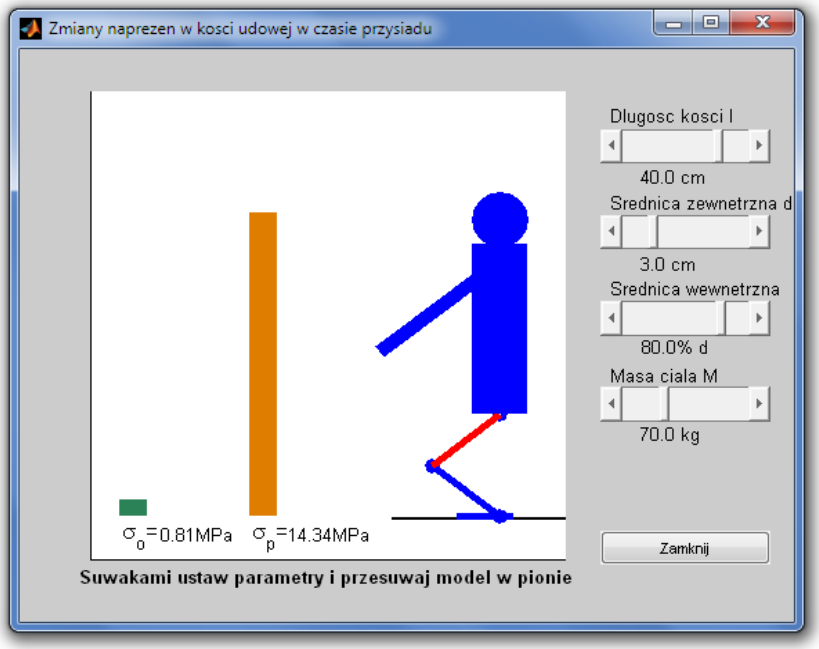
Odpowiedni model zawiera dość dużo elementów związanych z interfejsem graficznym, nie nadaje się więc do tego, żeby go tu zaprezentować w tekście książki, ale odpowiedni M-kod jest dostępny na stronie www.Tadeusiewicz.pl.



Rys. 4.13. Model przystosowany do wykonywania przysiadów. Rozważana kość udowa jest wyróżniona czerwonym kolorem

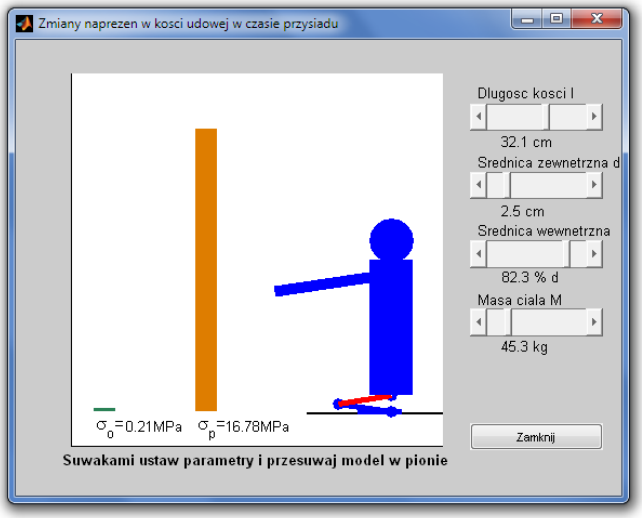
Zobaczmy jak ten model działa. Po uruchomieniu aplikacji pojawia się obraz jak na rysunku 4.13. Na obrazie tym widoczna jest uproszczona sylwetka człowieka (którą będzie można zmuszać do wykonywania przysiadu „ciągnąc" ją w odpowiednią stronę (w górę lub w dół) za pomocą myszki.

Pociągając tę sylwetkę w dół zmuszamy ją do „wykonania przysiadu" obserwując równocześnie zmieniające się naprężenia osiowe i poprzeczne w kości udowej w postaci słupków o zmieniającej się wysokości po lewej stronie okna (Rys. 4.14).



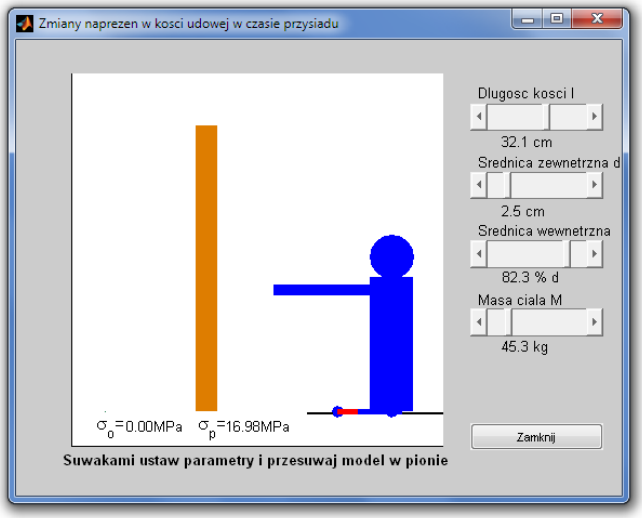
Rys. 4.14. Zmiany naprężeń osiowych i poprzecznych w modelu kości udowej człowieka podczas symulowanego przysiadu

W modelu tym można oczywiście zmieniać wymiary symulowanej kości korzystając z suwaków po prawej stronie okna. Na rysunku 4.15 pokazano wynik symulowanych zmian rozmiarów kości – wielkość niebezpiecznego (grozi złamanie!) naprężenia poprzecznego kości wyraźnie wzrosła!



Rys. 4.15. Symulacja tej samej sytuacji jak na rysunku 4.14 przy zmienionych wymiarach kości

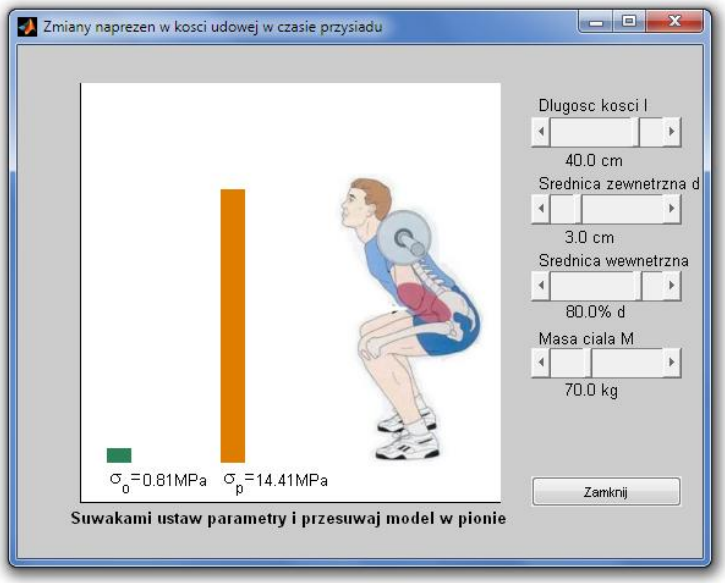
Model jest tak zbudowany, że głębokość przysiadu można regulować „aż do podłogi" co powoduje, że naprężenia osiowe maleją w kości do zera, za to naprężenia poprzeczne osiągają bardzo duże wartości (Rys. 4.16).



Rys. 4.16. Naprężenia kości w maksymalnie pogłębionym przysiadzie

Zachęcamy Czytelnika, żeby sam poeksperymentował z opisanym modelem (pobierając go ze strony www.Tadeusiewicz.pl) i poznał jego dalsze możliwości. Oczywiście rozważany tu model można by było jeszcze bardziej udoskonalić

pod względem graficznym doprowadzając go na przykład do takiej postaci jak przedstawiona (hipotetycznie) na rysunku 4.17 – ale nakład pracy przy oprogramowywaniu animacji takiego modelu byłby już jednak nieopłacalny.



Rys. 4.17. Hipotetyczna (w rzeczywistości nie wykonana) wersja modelu z bardzo zaawansowaną grafiką

4.5. Modelowanie pracy mięśni. Symulacja biegu i skoku

4.5.1. Uwagi wstępne do modelowania pracy mięśni

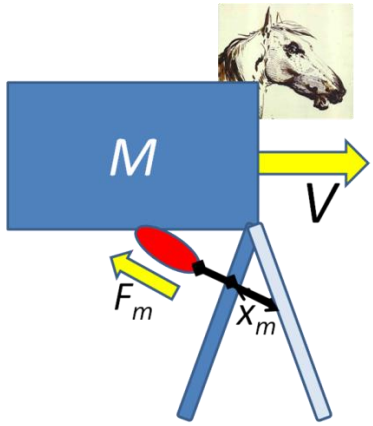
Omówione wyżej modelowanie kości pozwoliło na wyciąganie wniosków na temat wytrzymałości mechanicznej szkieletów zwierząt i ludzi, jednak w dużej mierze chodziło tu o obciążenia, jakim podlegają kości w sposób statyczny. Obecnie interesować nas będzie ruch, będący następstwem pracy mięśni.

Ruch jest jedną z głównych cech charakteryzującą organizmy zwierzęce. Zarówno dla zwierząt poruszających się w środowisku wodnym jak i lądowym, mobilność odgrywa kluczową rolę w poszukiwaniu pożywienia jak również podczas ucieczki przed drapieżnikiem. Zwierzęta wodne wykorzystują cztery zasadnicze sposoby przemieszczania się: siłę odrzutu, falowanie ciałem, wiosłowanie i ślizganie. Na zwierzęta poruszające się w wodzie działa siła wyporu, która pozwala zaoszczędzić energię na utrzymanie równowagi. Poruszanie się po lądzie jest znacznie trudniejsze i wymagało w procesie ewolucji na wypracowanie specjalnych narządów i technik, które nie tylko pozwalają na utrzymanie równowagi, ale także na szybkie przemieszczanie

się. Ruch kroczący jest typowy dla zwierząt posiadających odnóża kroczne, które muszą być na tyle silne, aby udźwignąć masę ciała oraz ją przenosić.

4.5.2. Model biegu

Wyobraźmy sobie teraz model zwierzęcia o masie M, które osiąga prędkość V[16]. Model taki naszkicowano na rysunku 4.18. Oczywiście koński łeb na tym rysunku odgrywa rolę jedynie dekoracyjną, a założenie, że zwierzę ma tylko jedną nogę, a ta noga napędzana jest przez jeden tylko mięsień rozwijający siłę F_m – jest w oczywisty sposób daleko idącym uproszczeniem rzeczywistości.



Rys. 4.18. Rozważany model biegnącego zwierzęcia. Opis w tekście.

Uproszczone będą także nasze rozważania dotyczące samego mechanizmu biegu jako takiego. Prawdziwy bieg polega na tym, że odpowiednio przebierając kończynami rozważana istota (człowiek, zwierzę, owad) utrzymuje pewną prędkość przez pewien okres czasu. Podczas biegu można wyróżnić fazę odbicia, kiedy kończyna opiera się na podłożu, a mięśnie wykonują pracę napędzając ciało do przodu oraz wyrzucając je do góry, bowiem po fazie odbicia następuje faza skoku (lotu, płynięcia) kiedy ciało biegnącej istoty porusza się ruchem balistycznym, jak wyrzucony pocisk. Ruch nóg podczas biegu wymaga ich wyrzucania do przodu (co wiąże się z koniecznością ich przyspieszania w tym właśnie kierunku i wydatkowania na to określonej energii), następnie wyhamowania tego ruchu do przodu i rozpoczęcia – po postawieniu nogi na ziemi – ruchu do tyłu, który popycha całe ciało naprzód (znowu wydatek energii) itd. Co więcej, biegnąc pokonujemy różne opory (na przykład opór powietrza), co także wymaga wydatkowania energii.

W sumie gdybyśmy chcieli zbudować **dokładny** model biegu – byłby on

[16] Model ten (podobnie jak opisany dalej model skoku) oparty został na książce Johna Maynarda Smitha zatytułowanej "*Matematyka w biologii*" (Wiedza Powszechna, Warszawa, 1974).

ogromnie skomplikowany. Modele takie buduje się na użytek między innymi naukowego wspomagania treningu sportowego[17] jednak w tym skrypcie chodzi nam o stworzenie modelu najprostszego, jaki tylko da się zbudować. W związku z tym rozważać będziemy głównie model rozpoczęcia biegu, kiedy ciało zwierzęcia (lub człowieka) poprzednio pozostające w spoczynku (nieruchome) zaczyna się poruszać i nabiera prędkości *V*. Potem tę prędkość musi utrzymać, a gdy biegnąca istota się zmęczy – będzie się musiała zatrzymać, ale tym się już nie zajmujemy. Interesuje nas tylko to, że ciało modelu podczas ruchu posiada energię kinetyczną równą (4.7):

$$E_k = \frac{1}{2} MV^2 \tag{4.7}$$

Energię tę musiało zdobyć (bo wcześniej jej nie miało będąc w spoczynku) na skutek ruchu wielu kończyn i aktywności wielu mięśni – bo w rzeczywistości rozpędzające się zwierzę nabiera szybkości po kilku lub kilkunastu krokach. My jednak w naszym modelu założymy, że cała energia kinetyczna powstała w wyniku ruchu jednej „zastępczej kończyny", napędzanej przez jeden „zastępczy mięsień" rozwijający siłę F_m i podlegający skróceniu o odcinek x_m. Praca wykonana przez kurczący się mięsień wyraża się wzorem:

$$L_m = F_m x_m \tag{4.8}$$

Przyrównując do siebie wzory (4.7) i (4.8) otrzymujemy:

$$\frac{1}{2} MV^2 = F_m x_m \tag{4.9}$$

Wyliczoną prędkość ruchu opisuje wzór (4.10):

$$V = \sqrt{\frac{2F_m x_m}{M}} \tag{4.10}$$

Rozważmy teraz powiększenie ciała zwierzęcia ze współczynnikiem β zgodnie z następującymi wzorami (4.11-4.13):

$$X_{wm} = \beta x_m \tag{4.11}$$

[17] Taki dokładny model biegu rozważany był między innymi w książce Marczewski W., Tadeusiewicz R.: *„Antropomotoryka biocybernetyczna"*, wydanej w 1993 roku przez Akademię Wychowania Fizycznego w Krakowie.

$$M_w = \beta^3 M \tag{4.12}$$

$$F_{mw} = \beta^2 F_m \tag{4.13}$$

Po podstawieniu wzorów (4.11-4.13) do wzoru (4.10) otrzymujemy równanie, które sugeruje, że prędkość nie zależy od wielkości współczynnika β:

$$V_w = \sqrt{\frac{2F_{mw}x_{wm}}{M_w}} = \sqrt{\frac{2\beta^2 F_m \beta x_m}{\beta^3 M}} = \sqrt{\frac{2F_m x_m}{M}} \tag{4.14}$$

Ze wzoru (4.14) można wywnioskować, że zwierzęta o różnych masach i rozmiarach uzyskują podobne szybkości biegu.

Tabela 4.1. przedstawia masy ciała oraz prędkości poruszania się dla różnych gatunków zwierząt. Tabela ta potwierdza przeprowadzony wyżej wywód. Potwierdza go także rysunek (4.19).

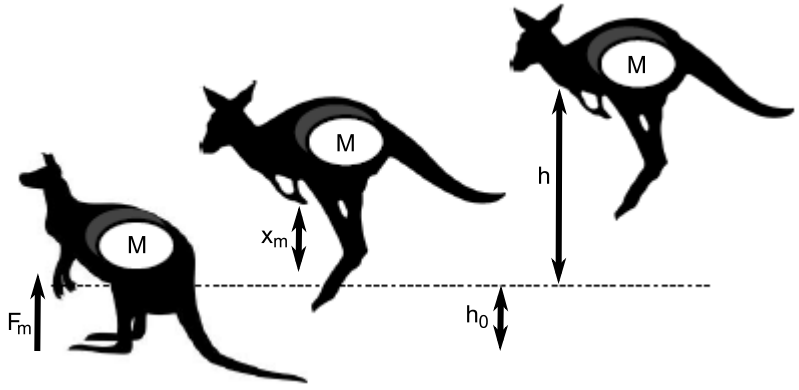
Tabela 4.1. Porównanie różnych gatunków zwierząt, które dla różnych mas osiągają podobne prędkości poruszania się.

Gatunek	Masa [kg]	Prędkość [km/h]
Pies	60	60
Koń	700	60
Kangur	70	45
Gerbil	0.1	50

Rys. 4.19. Obraz pokazujący, że psy i konie biegną z tą samą prędkością (Źródło: http://www.myartprints.co.uk/kunst/randolph_caldecott/fox_hunting_surrey_hi.jpg, dostęp – wrzesień 2011)

4.5.3. Model skoku

Kolejnym przykładem prostego modelu biologicznego związanego z narządem ruchu jest wysokość skoku osiągana przez różne organizmy. Podobnie jak w poprzednim przypadku wyobraźmy sobie, że organizm o masie M skacze na wysokość h (Rys. 4.20).



Rys. 4.20. Organizm o masie M skacze na wysokość h. (Źródło ikon: http://www.cksinfo.com/animals/kangaroos/index.htm, dostęp – lipiec 2011)

Mięśnie skaczącego zwierzęcia wytwarzają siłę F_m i działają na drodze x_m. Dzięki temu zwierzę o masie M skacze na wysokość h (Rys. 4.20). Obliczmy, od czego zależy ta wysokość.

Wzór (4.14) przedstawia konieczną tożsamość pomiędzy energią potencjalną, jaką osiągnie ciało zwierzęcia na wysokości h a pracą wykonaną przez mięśnie, które go na tę wysokość wyrzuciły.

$$Mgh = F_m x_m \tag{4.14}$$

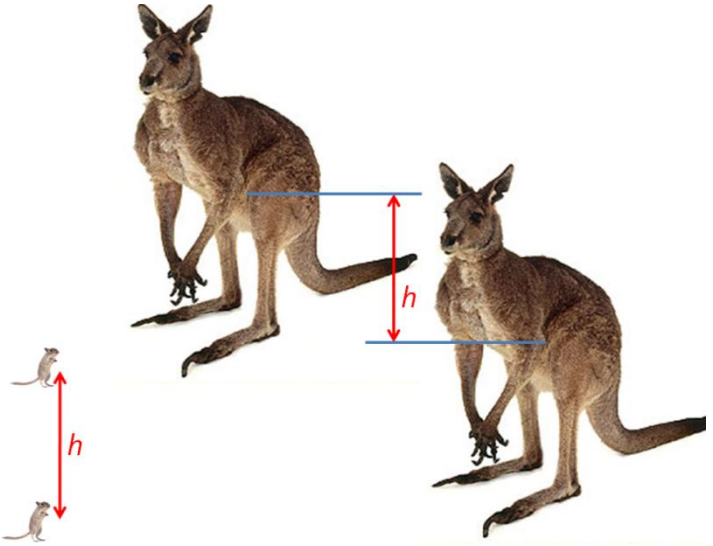
Po przekształceniu wzoru (4.14), otrzymujemy wzór (4.15), który przedstawia wysokość na którą skacze zwierzę o masie M.

$$h = \frac{F_m x_m}{Mg} \tag{4.15}$$

Rozważmy podobnie jak w poprzednim modelu powiększenie ciała zwierzęcia
ze współczynnikiem β zgodnie z wcześniej przytoczonymi wzorami (4.11-4.13). Po podstawieniu tych wzorów do wzoru (4.15) otrzymujemy:

$$h_w = \frac{F_{mw} x_{wm}}{M_w g} = \frac{\beta^2 F_m \beta x_m}{\beta^3 Mg} = \frac{F_m x_m}{Mg} \tag{4.16}$$

Współczynnik β we wzorze (4.16) znika, co sugeruje, że wysokość skoku nie zależy od rozmiarów osobnika. Stwierdzenie to wydaje się paradoksalne, ale zwróćmy uwagę na jeden czynnik: Powyższe rozważanie nie dotyczy wysokości, jaką osiągnie zwierzę w wyniku skoku tylko wysokość skoku jako takiego. Większe zwierzę ma wyżej środek ciężkości jeszcze przed skokiem, więc osiąga wyższy pułap, ale sam skok jest taki sam. Malutki gerbil skacze ma taką samą wysokość skoku jak duży kangur (Rys. 4.21) chociaż po skoku znajduje się znacznie niżej.



Rys. 4.21. Porównanie skoku kangura ze skokiem małego gerbila (Źródło: http://matematyka.lo4.poznan.pl/ oraz http://ottawahumane.ca/, dostęp - lipiec 2011)

4.5.4. Implementacja modeli

Poniższy fragment kodu dla MATLABa przedstawia model biegu. Model ten nie posiada oddzielnego interfejsu użytkownika, ale korzystający z modelu mogą poprzez modyfikację M-pliku określać poszczególne parametry modelu A oraz B (Przykład 4.3). W modelu tym obliczana jest wartość energii potrzebna do osiągnięcia podanej prędkości przez zwierzęta o różnych masach i rozmiarach.

Kolejny fragment kodu przedstawia model skoku (Przykład 4.4). Użytkownicy mogą poprzez prostą modyfikację M-pliku zmieniać podstawowe parametry symulacji takie jak: wysokość skoku, masa osobnika A i B, położenie środka ciężkości. Obliczane parametry to pułap skoku oraz energia potrzebna do jego osiągnięcia.

Obliczana jest wysokość skoku oraz wartość energii potrzebna do osiągnięcia podanej względnej wysokości przez zwierzęta o różnych masach i rozmiarach.

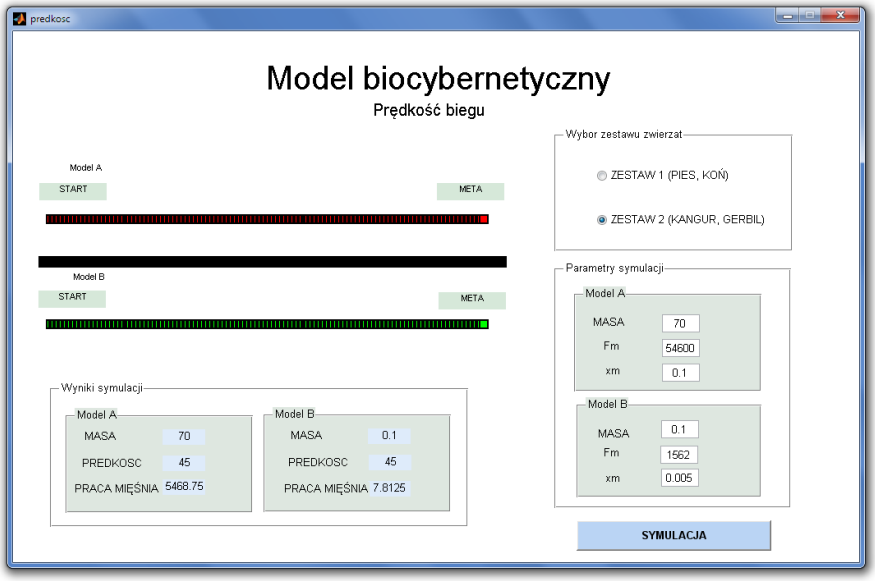
Uzyskane wyniki mogą być porównywane i analizowane.

Przykład 4.3 – Model biegu

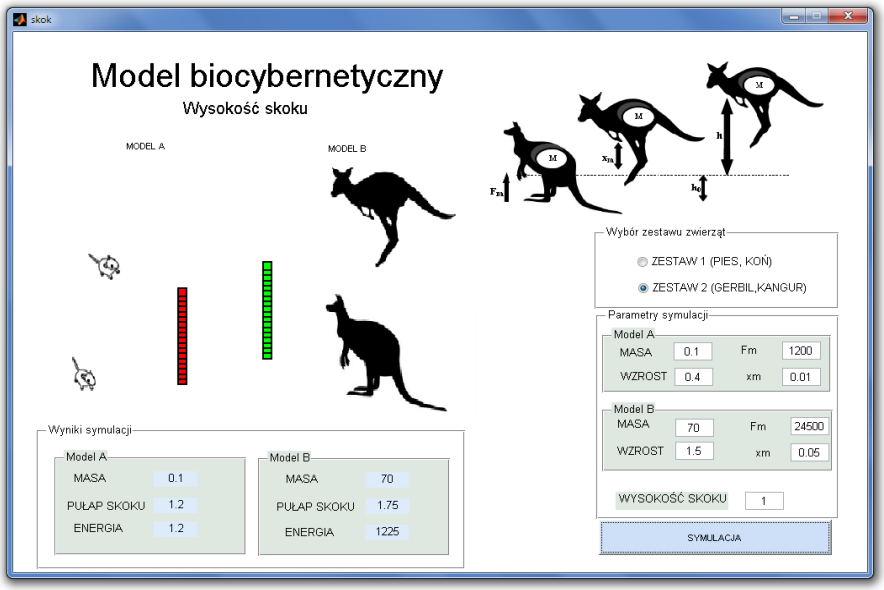
```
% Parametry modelu

% Model A - kangur
mk = 70;        % masa kangura [kg]
vk = 45;        % prędkość poruszania się [km/h]
xk = 0.1;       % skrócenie mięśnia [m]
Fmk = 54600;    % siła rozwijana przez mięśnie [N]

% Model B - gerbil
mg = 0.1;       % masa gerbila [kg]
vg = 45;        % prędkość poruszania się [km/h]
xg = 0.005;     % skrócenie mięśnia [m]
Fmg = 1562;     % siła rozwijana przez mięśnie [N]

% Obliczenia energii kinetycznej modelu
% Model A - kangur
EkA = mk*(vk/3.6)^2*0.5;  % energia modelu A [J]

% Model B - gerbil
EkB = mg*(vg/3.6)^2*0.5;  % energia modelu B [J]

% Współczynnik 3.6^2 pochodzi od zmiany jednostki z
% km/h na m/s

% Obliczenia prędkości biegu na podstawie: masy, siły
% mięśni i wielkości skurczu

% Model A - kangur
vA  =  ((2*Fmk*xk)/mk)^0.5*3.6      % prędkość modelu A
                                    % [km/h]

% Model B - gerbil
vB  =  ((2*Fmg*xg)/mg)^0.5*3.6      % prędkość modelu B
                                    % [km/h]
```

Podobnie jak w poprzednim podrozdziale – na stronie www.Tadeusiewicz.pl dostępne są rozszerzone wersje aplikacji umożliwiające wygodne wprowadzanie parametrów dla modelu A i B oraz przedstawiające prostą symulację biegu (Rys. 4.22) oraz skoku (Rys. 4.23).

Interfejs użytkownika został podzielony w obu przypadkach na 3 obszary. Pierwszy obszar zatytułowany Wybór zestawu zwierząt zawiera gotowy komplet parametrów dla dwóch zestawów zwierząt o różnych masach, ale podobnych prędkościach poruszania się lub wysokościach skoku. Obszar drugi Parametry symulacji pozwala na samodzielne ustawianie danych dla modelu A oraz B takich jak: masa, prędkość, wzrost, wysokość skoku. Obszar znajdujący się po lewej stronie zawiera wyniki symulacji oraz prostą animację.



Rys. 4.22. Okno aplikacji: Model biocybernetyczny - prędkość biegu



Rys. 4.23. Okno aplikacji: Model biocybernetyczny - wysokość skoku

Przykład 4.4 – Model skoku

```
% Parametry modelu

% Model A - kangur
mk = 70;      % masa kangura [kg]
hko = 0.75;   % środek ciężkości [m] (połowa średniej
              % wysokości)
xk = 0.05;    % skrócenie mięśnia [m]
Fmk = 24500; % siła rozwijana przez mięśnie [N]

% Model B - gerbil
mg = 0.1;     % masa gerbila [kg]
hgo = 0.2;    % środek ciężkości [m] (połowa średniej
              % wysokości)
xg = 0.001;   % skrócenie mięśnia [m]
Fmg = 1200;   % siła rozwijana przez mięśnie [N]

% Względna wysokość skoku (stała dla modelu A i B)
h0 = 1;       % względna wysokość skoku 1 [m]
g = 10;       % przyspieszenie ziemskie [m/s2]

% Obliczenia energii modelu oraz względnej wysokości
% skoku

% Model A - kangur
hpA = hko+h0          % wysokość skoku [m]
EpA = hpA*mk*g        % energia modelu A [J]

%Model B - gerbil
hpB = hgo+h0          % wysokość skoku [m]
EpB = hpB*mg*g        % energia modelu B [J]

% Obliczenia wysokości skoku na podstawie masy, siły
% mięśni i wielkości skurczu

% Model A - kangur
hA = (Fmk*xk)/(mk*g) % wysokość skoku [m]

% Model B - gerbil
hB = (Fmg*xg)/(mg*g) % wysokość skoku [m]
```

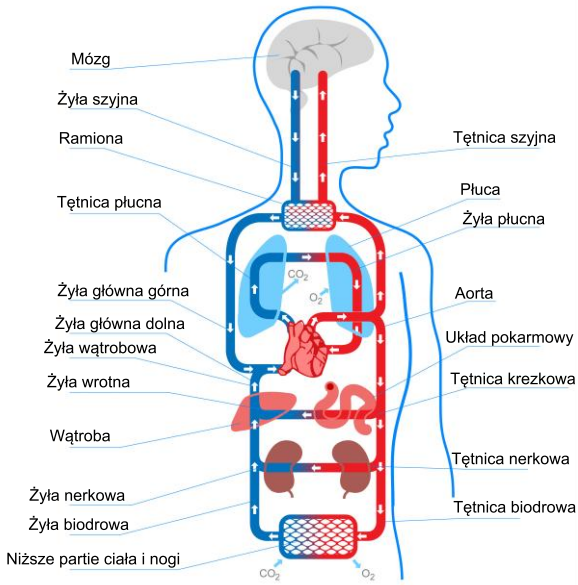
4.6. Model krążenia krwi i częstości tętna

4.6.1. Opis modelu matematycznego

Rozważania podobne do przedstawionych wyżej dla kości i mięśni można także prowadzić dla innych narządów. W niniejszym rozdziale rozważany będzie model krążenia krwi, który pozwoli (między innymi) na wyznaczenie

częstości tętna u zwierząt o różnych rozmiarach.

Układ krążenia jest jednym z najbardziej skomplikowanych i najbardziej interesujących systemów w organizmie człowieka (Rys. 4.24). Jego modelowanie, nawet w bardzo uproszczonej formie, jest celowe, ponieważ daje nowe możliwości wykorzystania techniki komputerowej w rozpoznawaniu, zapobieganiu i przewidywaniu wielu chorób układu krążenia. Aby móc w prosty i przystępny sposób zamodelować ten układ przyjmujemy zasadę maksymalnego uproszczenia jego budowy i działania. Sprawdzimy, czy pomimo wykorzystania tylko kilku podstawowych założeń co do tego systemu będziemy w stanie uzasadnić prawidłowości w jego działaniu.



Rys. 4.24. Schemat krążenia płucnego oraz obwodowego (Opracowanie własne na podstawie: http://www.heartzine.com/anatomy-physiology/the-circulatory-system.html)

Podstawowym parametrem układu krążenia jest ilość krwi Q_k dostarczana w jednostce czasu do mięśni. Przy pomocy krwi do tkanek dociera tlen, którego ilość oznaczymy przez Q_t, zaś tlen jest ściśle powiązany z mocą rozwijaną przez poszczególne mięśnie. W związku z tym ilość docierającego do mięśni tlenu jest związana z energią produkowaną przez organizm – i z tego skorzystamy.

Objętość krwi Q_k dostarczana w jednostce czasu do poszczególnych mięśni zależy od przekroju naczyń S_n (przyjmujemy tu przekrój aorty) oraz od szybkości przepływu krwi V_k. Wyobrażając sobie krew wypompowaną przez serce w jednostce czasu jako słup o wysokości odpowiadającej prędkości wypływu V_k i o przekroju równym przekrojowi aorty S_k otrzymujemy wzór (4.17) określający ilość krwi Q_k dostarczaną w jednostce czasu do mięśni:

$$Q_k = S_k V_k \tag{4.17}$$

Prędkość przepływu krwi V_k zależy od ciśnienia skurczowego p_k wywołanego przez serce (4.18):

$$V_k = f(p_k) \tag{4.18}$$

Dokładny opis tego zjawiska jest naprawdę skomplikowany i odwołuje się do praw hydrodynamiki, których matematyczny opis wymaga rozwiązywania skomplikowanych równań różniczkowych cząstkowych Naviera-Stokesa, trudnych do rozwiązania nawet przy użyciu największych dostępnych dziś superkomputerów. Dla potrzeb modelu rozważanego w tym rozdziale zastosujemy jednak bardzo uproszczone rozumowanie, oparte na stwierdzeniu oczywistego faktu, że czynnikiem wywołującym przepływ krwi jest ciśnienie skurczowe powstające w wyniku siły F_s rozwijanej przez kurczący się mięsień serca[18]. Siła F_s zależy od bardzo wielu czynników m.in. od wielkości serca, od grubości jego ścian, od prowadzonego trybu życia (wpływającego na sprawność mięśnia sercowego), od emocji rozważanej osoby i od jej uwarunkowań genetycznych. W naszych rozważaniach ograniczamy się jedynie do jednego parametru określającego cechy mięśnia sercowego, to znaczy do jego jednostkowej wytrzymałości mechanicznej σ_s oraz do jednego parametru opisującego rozmiary i kształt rozważanego serca, to znaczy do powierzchni pola jego przekroju poprzecznego S_s. Zależność siły skurczu od tych dwóch wartości także bardzo uprościmy zakładając, że jest ona proporcjonalna do ich iloczynu (4.19).

$$F_s = \sigma_s S_s \tag{4.19}$$

Siła F_s działa na krew zgromadzoną w sercu i powoduje jej wyrzut oraz dalsze przetłaczanie. Można zauważyć, że siła F_s rozwijana przez mięsień serca, jest równoważona przez siłę pochodzącą od ciśnienia krwi F_k (4.20).

$$F_k = p_k S_k \tag{4.20}$$

Warunek równowagi sił można zapisać w postaci (4.21):

$$\sigma_s S_s = p_k S_k \tag{4.21}$$

Z powyższego wzoru wyznaczamy wielkość skurczowego ciśnienia krwi (4.22).

[18] Taki dokładny model krążenia krwi i częstości tętna rozważany był między innymi w książce Marczewski W., Tadeusiewicz R.: *„Antropomotoryka biocybernetyczna”*, wydanej w 1993 roku przez Akademię Wychowania Fizycznego w Krakowie.

$$p_k = \frac{\sigma_s S_s}{S_k} \tag{4.22}$$

Podobnie jak w poprzednich rozdziałach sprawdzimy, czy rozmiar ciała rozważanego zwierzęcia wpływa na wartość ciśnienia krwi. Wprowadzimy współczynnik wielkości ciała β przy pomocy którego przeskalujemy odpowiednie powierzchnie.

$$S_{s\_new} = \beta^2 S_s \tag{4.23}$$

$$S_{k\_new} = \beta^2 S_k \tag{4.24}$$

Oczywiście parametr jednostkowej wytrzymałości mechanicznej σ_s nie zależy od rozmiarów.

Po podstawieniu wzorów (4.23) i (4.24) do wzoru (4.22) zauważamy, że współczynnik wielkości ciała β ulega uproszczeniu. Wynik ten wskazuje na brak zależności pomiędzy ciśnieniem krwi p_k, a rozmiarami ciała. Z tego wynika, że również prędkość przepływu krwi V_k nie ulega zmianie, mimo znacznych zmian rozmiarów ciała rozważanego zwierzęcia.

Przeanalizujmy teraz wzór określający ilość krwi dostarczanej do mięśni w jednostce czasu Q_k w odniesieniu do współczynnika β.

Jeżeli

$$Q_k = S_n V_k \tag{4.25}$$

oraz

$$S_{n\_new} = \beta^2 S_n \tag{4.26}$$

to

$$Q_{k\_new} = \beta^2 S_n V_k \tag{4.27}$$

Ostateczny rezultat przeprowadzonego rozumowania prowadzi do stwierdzenia, że moc pracujących mięśni zależy od ilości dostarczanej do mięśni krwi, a ta jest zależna od kwadratu rozmiarów ciała. Jak z tego wynika moc pracujących mięśni jest ograniczona przez wydolność systemu krążenia – i to jest ważny wniosek.

Serce posiada określoną pojemność, którą oznaczamy Q_s. Parametr ten jest oczywiście zależny od wielkości ciała (4.28).

$$Q_{s\_new} = \beta^3 Q_s \tag{4.28}$$

W ustalonym czasie t przez serce musi zostać przepompowana krew przepływająca przez aortę i wynosząca $S_n V_k t$. Przy określonej objętości Q_s serce musi skurczyć się n razy zgodnie ze wzorem (4.29) do którego podstawiono zależności (4.26) i (4.28).

$$n = \frac{\beta^2 S_n V_k t}{\beta^3 Q_s} \tag{4.29}$$

Po uproszczeniu współczynnika wielkości ciała oraz przyjmując $t=1$ otrzymujemy częstotliwość tętna (4.30).

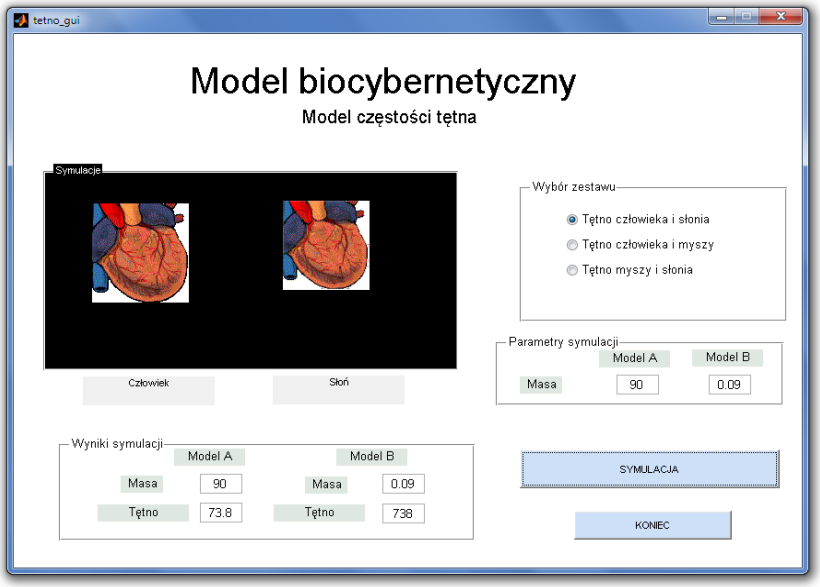
$$n = \frac{S_n V_k}{\beta Q_s} \tag{4.30}$$

Z powyższych rozważań wynika, że im większe zwierzę, tym niższa wartość tętna. Wniosek ten odpowiada dokładnie wartościom obserwowanym w świecie zwierząt. Tętno myszy jest dużo wyższe od tętna człowieka i wynosi ok. 500. Słoń, pomimo tego, że jest dużo większy od człowieka posiada tętno o połowę mniejsze, ok. 30.

4.6.2. Symulacja krążenia krwi i jego konsekwencji

Poniższy fragment kodu dla MATLABa (Przykład 4.5) przedstawia bardzo uproszczony model układu krążenia pozwalający na prognozowanie wartości tętna w zależności od wielkości osobnika. Za wzorzec przyjmowane są parametry człowieka (masa, tętno). W zależności od masy pozostałych modeli (słoń, mysz) wyliczane są poszczególne wartości tętna. Obliczana jest wartość tętna dla modelu A i B bazując na danych z modelu wzorcowego. Oznacza to, że znając masę osobnika jesteśmy w stanie prognozować jego tętno.

Podobnie jak w poprzednim podrozdziale – na stronie www.Tadeusiewicz.pl dostępna jest rozszerzona wersja aplikacji umożliwiające wygodne wprowadzanie parametrów dla modelu A i B oraz przedstawiające prostą symulację bicia serca dla poszczególnych modeli (Rys. 4.25).



Rys. 4.25 Okno aplikacji: Układ krwionośny i częstość tętna (Źródło ikon: http://www.thaifocus.com/elephant/heartbeat.htm, dostęp - lipiec 2011)

Przykład 4.5 – Model układu krwionośnego

```
% Dane modelu A - człowiek
% Model A jest modelem wzorcowym

m_w = 90;     % masa wzorcowa [kg]
vk_w = 1800; % prędkość przepływu krwi [cm/min]
sn_w = 24.6; % przekrój naczyń [cm^2]
qs_w = 600;  % pojemność serca [cm^3]

fprintf('Model A /człowiek/ - częstotliwość tętna:\n');
n = (vk_w*sn_w)/qs_w

% Dane modelu B - mysz
fprintf('Model B /mysz/ - częstotliwość tętna:\n');
m_m = 0.09;     % masa myszy [kg]
n = (vk_w*sn_w)/(qs_w*((m_m/m_w)^(1/3)))

% Dane modelu C - słoń
fprintf('Model C /slon/ - częstotliwość tętna:\n');
m_s = 3000;     % masa słonia [kg]
n = (vk_w*sn_w)/(qs_w*((m_s/m_w)^(1/3)))
```

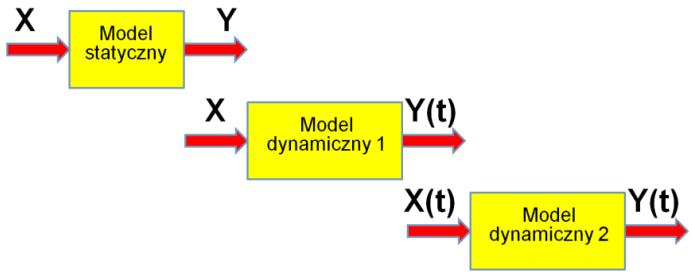
Interfejs użytkownika został podzielony na 4 obszary. Pierwszy obszar zatytułowany **Wybór zestawu** zawiera gotowy komplet parametrów dla trzech

par osobników. W każdym z tych przypadków osobniki znacząco różnią się masą, co odzwierciedlone zostaje w wartości tętna. Obszar drugi **Parametry symulacji** pozwala na samodzielne ustawianie masy dla modelu A oraz B. Obszar trzeci zatytułowany **Wyniki symulacji** pokazuje obliczoną wartość tętna. Obszar znajdujący się powyżej symuluje działanie serca.

4.7. Dynamiczne modele systemów biologicznych

4.7.1. Istota podziału na modele statyczne i modele dynamiczne

Przedstawiane wyżej modele były modelami **statycznymi**, czyli opisywały zjawiska, w których czynnik czasu nie odgrywał istotnej roli. Nawet jeśli rozważaliśmy model biegu czy skoku, to w istocie interesował nas wyłącznie jeden parametr opisujący ten bieg czy skok (odpowiednio szybkość biegu czy wysokość skoku) natomiast ignorowaliśmy na przykład opór powietrza na jaki napotyka się podczas biegu albo kolejne fazy ruchu kończyny skaczącego zwierzęcia. Teraz jednak nadeszła pora, żeby zacząć modelować procesy przebiegające w czasie. Takie modele określane są jako modele **dynamiczne** i są w istocie najciekawsze, bo coś się w nich dzieje.



Rys. 4.26. Model statyczny i dwa rodzaje modeli dynamicznych

Zasadniczą różnicę między modelem statycznym i dynamicznym ilustruje rysunek 4.26. Jak widać w modelu statycznym na wejściu i na wyjściu systemu mamy wartości liczbowe (oznaczone na rysunku odpowiednio jako X i Y). Model taki pozwala zmieniać wartości wejściowe X i obserwować związane z nimi zmiany wartości wyjściowej Y. Jest to jednak w sumie bardzo uboga forma wykorzystania wszystkich tych wspaniałych możliwości, jakie tkwią w matematycznym modelowaniu i w komputerowej symulacji. Dlatego i w modelach i w symulacjach sięga się do modeli dynamicznych.

Model dynamiczny tym się cechuje, że na jego wyjściu obserwujemy pewien **proces,** sekwencję wartości zmieniających się w czasie, stąd na rysunku 4.26 wyjścia modeli dynamicznych prezentują zamiast pojedynczej wartości *Y* – funkcję *Y(t)*, która w rzeczywistym systemie jest zwykle ciągła (ma określone wartości dla każdego momentu czasu *t*), zaś w modelu symulacyjnym jest sekwencją wartości zmiennej *Y* w kolejnych wybranych chwilach czasu *t*.

Jak pokazano na rysunku 4.26 - można rozważać dwa typy modeli dynamicznych. Model określony na rysunku jako numer 1 ma na wejściu tylko wartość liczbową *X*. Wartość ta jest parametrem, który określa sposób zachowania modelu (a także modelowanego systemu), jednak samo zachowanie tego modelu jest już spontaniczne. **Generuje** on pewien proces zachodzący w czasie *Y(t),* który jednak nie podlega sterowaniu. Do matematycznego opisu modeli dynamicznych tego typu używa się zwykle równań różniczkowych, przy czym dla zachowania prostoty działania modelu preferowane są równania różniczkowe zwyczajne.

Model oznaczony numerem 2 zachowuje się odmiennie niż model oznaczony numerem 1. On także wytwarza pewien proces zachodzący w czasie *Y(t),* którym jednak możemy na bieżąco sterować, ponieważ sygnał wejściowy do modelu także ma formę przebiegu zmiennego w czasie *X(t)* i na tę czasową zmienność twórca modelu (lub jego użytkownik) może mieć wpływ. Tu także znajdują zastosowanie równania różniczkowe zwyczajne, są one jednak zwykle (ze względu na obecność czynnika sterującego *X(t)*) równaniami niejednorodnymi.

Rozważymy teraz wybrane dynamiczne modele systemów biologicznych, pokazując najpierw jak tworzy się ich matematyczny opis, a potem jak się je symuluje w wybranym środowisku informatycznym (w MATLABie). Jako pierwszy rozważymy model rozwoju populacji komórek nowotworowych w przypadku nie leczonego raka.

4.7.2. Najprostszy model dynamiczny typu 1

Rozwój choroby nowotworowej polega na tym, że komórki zmienione nowotworowo ustawicznie proliferują (mnożą się) przez co następuje wzrost guza rakowego, który niszczy narząd, z którego rak powstał i w którym się rozwija. Jeśli nie uda się na ten proces wpłynąć hamująco lub jeśli nie zdążymy inwazyjnie rosnącego raka usunąć chirurgicznie – dojdzie do całkowitego zniszczenia zaatakowanego przez raka narządu i do śmierci pacjenta.

Zamodelujmy ten proces. W tym podrozdziale (i w dwóch następnych) będzie mowa o konkretnym przykładzie nowotworu wieku dziecięcego – *neuroblastomy*, jednak modelowanie każdego innego raka będzie przebiegać podobnie. Stan choroby można opisać liczbą nieprawidłowych (nowotworowo zmienionych) komórek. Liczba ta zmienia się w czasie, bo komórki mnożą się poprzez kolejne podziały. W największym uproszczeniu można byłoby przyjąć, że wszystkie komórki ulegają podziałowi w tym samym tempie, co oznacza, że przyrost liczby komórek jest proporcjonalny do ich liczby. Dla podkreślenia zmienności tej liczby w czasie oznaczymy ją symbolem *P(t).*

Zależność pozwalającą modelować i śledzić zmienność funkcji *P(t)* można zapisać w postaci równanie różniczkowego (4.31), w którym szybkość zwiększania się liczby komórek (lewa strona równania) jest proporcjonalna do aktualnej liczby komórek *P(t)*, a współczynnikiem proporcjonalności jest stała wzrostu γ .

$$\frac{dP}{dt} = \gamma P \tag{4.31}$$

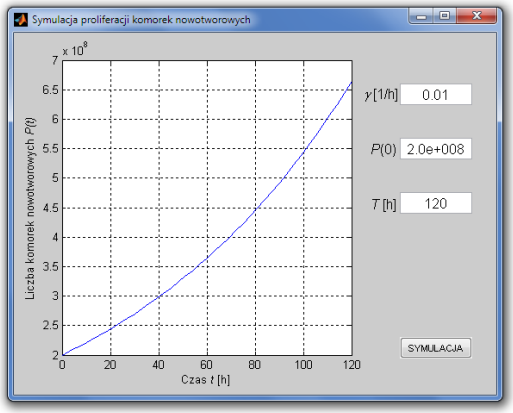
Oglądając tę zależność możemy stwierdzić, że jest to system określony na rysunku 4.26 jako *model dynamiczny 1*: ma jedno wejście, którym jest dowolnie ustalana ale niezmienna w trakcie modelowania stała γ pełniąca rolę wejścia do modelu:

$$X = \gamma \tag{4.32}$$

oraz mamy jeden proces zmienny w czasie – liczbę komórek powstających w wyniku proliferacji *P(t)*, co odpowiada narysowanej na schemacie modelu funkcji wyjściowej *Y(t)*:

$$Y(t) = P(t) \tag{4.33}$$

M-kod opisujący ten model przedstawia Przykład 4.6.



Rys. 4.27. Okno gotowego programu służącego do symulacji proliferacji komórek nowotworowych

W celu rozwiązania równania różniczkowego zwyczajnego pierwszego rzędu skorzystano z funkcji *ode45*, która realizuje metodę Dormand-Prince z automatyczną zmianą długości kroku. Formuła rozwiązywania równania jest następująca:

```
[t, dane] = ode45(@rownanie, przedział rozwiązań, dane początkowe, dodatkowe opcje)
```

Przykładowe wywołanie funkcji przedstawia przykład 4.6.

Przykład 4.6 – Model proliferacji komórek nowotworowych

```
% Ustawienie parametrów symulacji:
czas = [0,120];  % przedział czasu [h] (5 dni)
P0 = 2e8;        % początkowa liczba komórek (200 mln)

% Symulacja:
[t, P] = ode45(@rownanie_proliferacji, czas, P0);

% Wyprowadzenie wyników w formie graficznej:
plot(t, P);
title('Symulacja proliferacji komorek nowotworowych');
xlabel('Czas [h]');
ylabel('Liczba komorek nowotworowych Y(t) = P(t)');
```

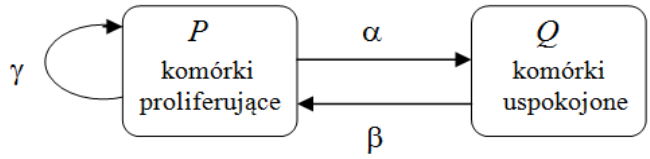
```
% W osobnym pliku rownanie_proliferacji.m zapisujemy
% poniższą funkcję

% Funkcja obliczająca prawą stronę równania
% różniczkowego
function dP_dt = rownanie_proliferacji(t, P)
gamma = 0.01;        % stała wzrostu [1/h]
dP_dt = gamma*P;     % równanie różniczkowe
```

Model ten ma też swoją wersję „luksusową”, przedstawioną na rysunku 4.27, pozwalającą na obserwację procesu namnażania się komórek rakowych w postaci graficznej – wygodniejszej do obserwacji i do interpretacji. Wersja ta jest dostępna – jak wszystkie pozostałe opisywane w tej książce – na stronie www.Tadeusiewicz.pl.

Warto w tym miejscu nawiązać do omawianego w punkcie 1.3.2. zagadnienia dyskretyzacji. Zwróćmy uwagę na sposób opisu zmienności w czasie liczebności pewnej populacji wyrażonej liczbą naturalną. W przypadku np. hodowli królików – gdy ich liczba jest rzędu kilkunastu lub kilkudziesięciu - naturalny będzie opis, w którym przebieg liczebności w czasie jest funkcją przedziałami stałą - bo między momentami zmiany liczby osobników upływa znaczący przedział czasu. Właściwą formą opisu są wtedy równania różnicowe. Stosowanie równań różniczkowych jest usprawiedliwione gdy liczebność jest duża i czas między momentami zmiany liczebności jest krótki. Wtedy „wysokość schodka” funkcji $P(t)$ jest mała w stosunku do wartości funkcji, a „długość schodka” – krótka w odniesieniu do czasu symulacji. W przypadku nowotworu, gdy liczba komórek guza o średnicy np. 2 cm jest rzędu 10^9, a czas między tworzeniem się kolejnych komórek jest rzędu ułamka sekundy - błąd zastąpienia opisu „schodkowego” (równaniem różnicowym) opisem ciągłym (równaniem różniczkowym) można uznać za pomijalnie mały.

4.7.3. Model dynamiczny typu 1 o wielu wejściach i wyjściach

Jeżeli chcemy, aby budowany model rozwoju raka lepiej odwzorowywał realne zjawisko narastania guza, to musimy odrzucić przyjęte w modelu (4.33) założenie o pełnej jednorodności komórek guza. W rzeczywistym guzie można wyróżnić komórki, które intensywnie się rozmnażają (blisko powierzchni guza) – ich liczbę będziemy oznaczać w dalszym ciągu przez $P(t)$ – oraz komórki nowotworowe, które przestały się już dzielić („uspokojone", wewnątrz guza) – $Q(t)$. Liczebność obu grup się zmienia – część komórek pozostaje w „swojej" grupie, a część zmienia charakter. Proces ten ilustruje rysunek 4.28.



Rys. 4.28. Grupy komórek wewnątrz guza nowotworowego. Opis w tekście.

Jak widać z rysunku $\alpha P(t)$ komórek proliferujących uspokaja się, a $\beta Q(t)$ komórek uspokojonych powraca do mnożenia się (α i β są współczynnikami proporcjonalności). Modelem matematycznym opisanych procesów są równania (4.34) i (4.35).

$$\frac{dP}{dt} = [\gamma - \alpha]P + \beta Q \tag{4.34}$$

$$\frac{dQ}{dt} = \alpha P - \beta Q \tag{4.35}$$

Jak wynika z przedstawionych rozważań, modelem który dobrze opisze rozważany tu system będzie model typu 1 o trzech wejściach:

$$X_1 = \alpha \tag{4.36}$$

$$X_2 = \beta \tag{4.37}$$

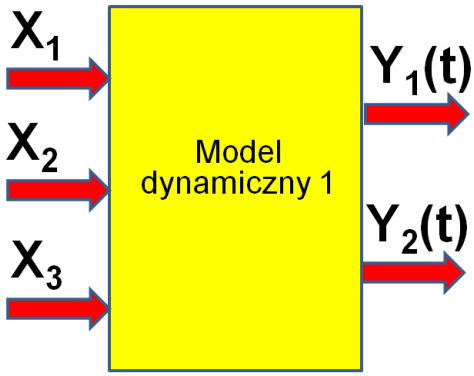
$$X_3 = \gamma \tag{4.38}$$

oraz o dwóch wyjściach:

$$Y_1(t) = P(t) \tag{4.39}$$

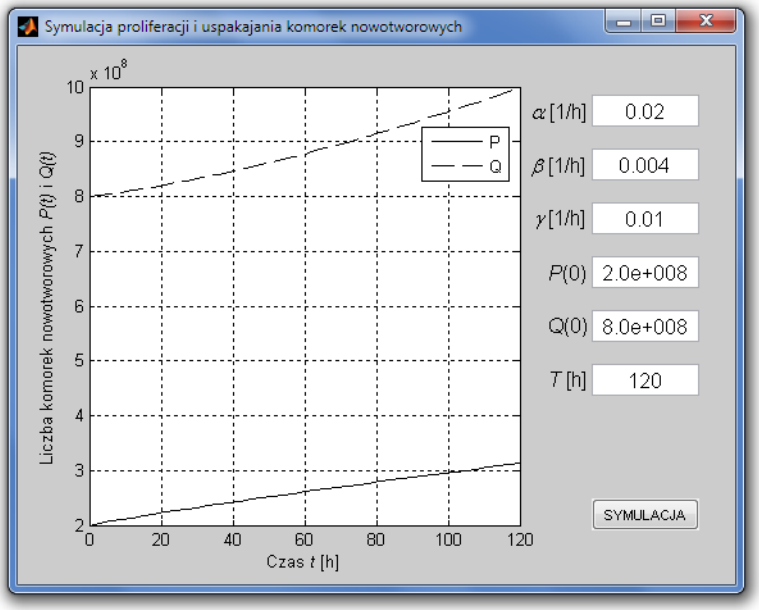
$$Y_2(t) = Q(t) \tag{4.40}$$

Model ten symbolicznie przedstawiono na rysunku 4.29.



Rys. 4.29. Wzbogacony model procesu nowotworowego

Program symulujący działanie opisywanego modelu przedstawiony jest niżej jako Przykład 4.7. Jego wersja z bogatszym interfejsem użytkownika dostępna jest na stronie www.Tadeusiewicz.pl, zaś widok panelu sterującego tego ulepszonego modelu przedstawiony jest na rysunku 4.30.



Rys. 4.30. Okno gotowego programu służącego do symulacji proliferacji i uspokajania komórek nowotworowych

Przykład 4.7 – Model proliferacji i uspakajania komórek nowotworowych

```
% Ustawienie parametrów symulacji:
czas = [0,120];  % przedział czasu [h] (5 dni)
P0 = 2e8;        % początkowe liczby komórek P i Q
Q0 = 8e8;

% Symulacja:
[t,PQ] = ode45(@rownanie_guza,czas,[P0, Q0]);

% Wyprowadzenie wyników w formie graficznej:
plot(t,PQ);
title(['Symulacja proliferacji i uspokajania'...
' komorek nowotworowych']);
xlabel('czas [h]');
ylabel('Liczba komorek guza');
legend('Komorki proliferujace P',...
    'Komorki uspokojone Q','Location','Best');
```

```
% W osobnym pliku rownanie_guza.m zapisujemy poniższą
% funkcję

% Funkcja obliczająca prawą stronę równ. różniczkowego
function dPQ_dt = rownanie_guza(t, PQ),

% Parametry komórek guza:
alfa = 0.02;  % stała uspokajania  [1/h]
beta = 0.004; % stała powrotu do dzielenia [1/h]
gamma = 0.01; % stała wzrostu [1/h]

% Równanie różniczkowe:
P = PQ(1);  Q = PQ(2);          % pobranie z wektora
dP_dt = (gamma-alfa)*P + beta*Q;% równanie (4.34)
dQ_dt = alfa*P - beta*Q;        % równanie (4.35)
dPQ_dt = [dP_dt; dQ_dt];        % spakowanie do wektora
```

4.7.4. Model dynamiczny typu 2

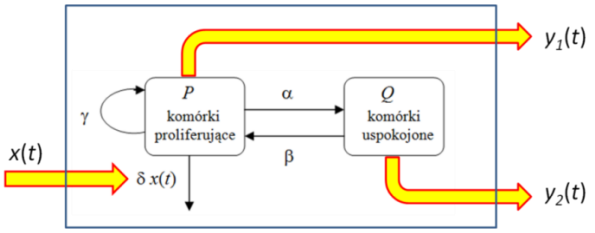
Modele systemów dynamicznych których zachowanie można sterować poprzez zmianę parametrów są bardziej interesujące, niż modele statyczne, ale nadal są to modele na których działanie użytkownik ma stosunkowo ograniczony wpływ. Dlatego przy posługiwaniu się modelami różnych systemów, w tym także omawianych w tej książce systemów biologicznych, dążymy do tego, żeby w modelach tych istniała możliwość aktywnego eksperymentowania z modelowanym procesem, to znaczy żeby przejść od modelu typu 1 (przejawiającego wyłącznie spontaniczne działanie) do modelu typu 2 (sterowalnego).

Prześledźmy taką transformację na bazie omawianego wcześniej

dynamicznego modelu rozwoju komórek nowotworowych, na który chcielibyśmy wpływać (oczywiście hamująco) poprzez odpowiednie leczenie. Jedną z możliwych terapii jest chemoterapia, w której dożylnie podawany jest lek o nazwie *Topotecan* (TPT). Lek ten jest aktywny tylko wobec tych komórek organizmu, które są w specyficznej fazie cyklu podziału – w fazie replikacji DNA. TPT zakłóca proces replikacji DNA powodując obumieranie dzielącej się komórki. Zbudujemy model[19] w którym sygnałem wejściowym będzie wielkość dożylnych iniekcji leku TPT, a sygnałem wyjściowym liczba komórek w fazie P i w fazie Q. W tym celu uzupełnimy model opisany równaniami (4.34), (4.35) i przedstawiony na rysunku 4.28. W nowym modelu uwzględnimy niszczące oddziaływanie leku TPT na komórki rozmnażające się. Założymy, że liczba komórek w fazie P obumierających na skutek działania TPT jest proporcjonalna do zmiennego w czasie stężenia tego leku *x(t)* w osoczu. Zakładamy model liniowy (liczba ginących komórek jest proporcjonalna do stężenia leku i do liczby proliferujących komórek), a dodatkowy parametr δ jest współczynnikiem proporcjonalności. Uzupełniony model przedstawia poniższy układ równań różniczkowych (4.41), (4.42) oraz rysunek 4.31.

$$\frac{dP}{dt} = \left[\gamma - \alpha - \delta x(t)\right]P + \beta Q \tag{4.41}$$

$$\frac{dQ}{dt} = \alpha P - \beta Q \tag{4.42}$$



Rys. 4.31. Model nowotworu uwzględniający możliwość sterowania

Otrzymany model jest modelem typu 2, a jego eksploatacja wymaga definiowania przebiegu czasowego sygnału sterującego $x(t)$ – jednak wynik jest wart tego wysiłku. Program realizujący symulację tego modelu w środowisku

[19] Przedstawiany w tym skrypcie model chemoterapii neuroblastomy jest wzorowany na modelu bardziej precyzyjnym, prezentowanym m.in. w artykule *Collins C, Fister K R, Key B, Williams M: „Blasting neuroblastoma using optimal control of chemotherapy Mathematical Biosciences and Engineering", Jul; 6(3): 451-67* dostępnym także na stronie:
http://www.math.unl.edu/~bdeng1/Teaching/math943/Student%20Lectures/Molly/BlastingNeuroblastoma.pdf

MATLAB podany jest niżej jako Przykład 4.8.

Przykład 4.8 – Model komórek nowotworowych ze sterowaniem

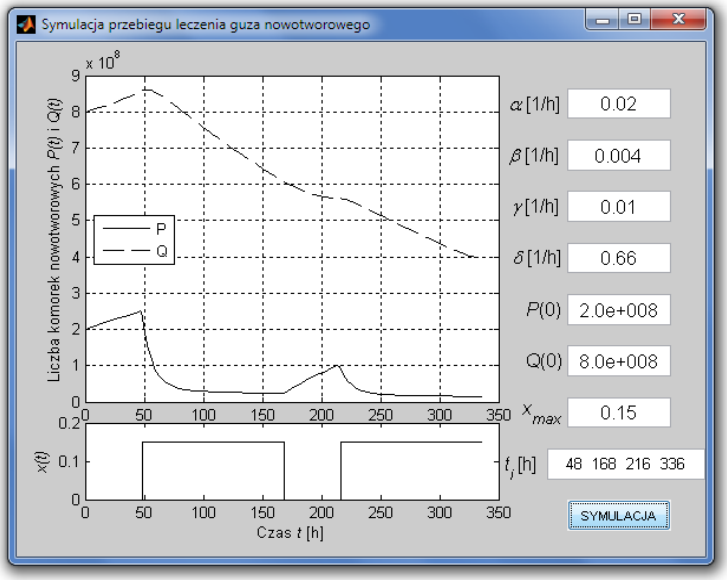
```
% Ustawienie parametrów symulacji:
czas = [0,336];  % przedział czasu [h] (2 tygodnie)
P0 = 2e8;        % początkowe liczby komórek P i Q
Q0 = 8e8;

% Symulacja:
[t,S] = ode45(@rownanie_leczonego_guza,czas,[P0, Q0]);

% Wyprowadzenie wyników w formie graficznej:
subplot(2,1,1); plot(t,S); % wykres liczebności komórek
title('Symulacja przebiegu leczenia nowotworu');
xlabel('czas [h]');
ylabel('Liczba komorek guza');
legend('Komorki proliferujace P',...
    'Komorki uspokojone Q','Location','Best');
rysuj_stezenie(czas); % wykres przebiegu dozowania leku
```

```
% W osobnym pliku rownanie_leczonego_guza.m zapisujemy
% poniższą funkcję

function dS_dt = rownanie_leczonego_guza(t, S),
% Funkcja obliczająca prawą stroną równania
% różniczkowego.
% Trzeci element wektora S (jeżeli istnieje) jest
% wartością wejściową - stężeniem leku.

% Obliczenie dozowania leku w chwili t:
if length(S) == 3, x = S(3);
else               x = stezenie_leku(t);
end

% Parametry modelu:
alfa = 0.02;      % stała uspokajania  [1/h]
beta = 0.004;     % stała powrotu do dzielenia [1/h]
gamma = 0.01;     % stała wzrostu [1/h]
delta = 0.66;     % współczynnik oddziaływania leku

% Równanie różniczkowe:
P = S(1);  Q=S(2);          % pobranie z wektora
dP_dt = (gamma-alfa-delta*x)*P + beta*Q;
dQ_dt = alfa*P - beta*Q;
dS_dt = [dP_dt; dQ_dt];     % spakowanie do wektora
```

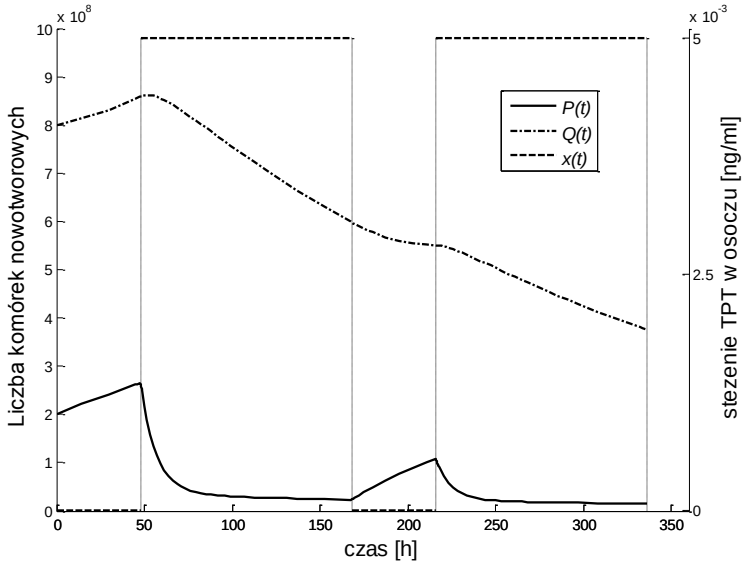
```
% W osobnym pliku rysuj_stezenie.m zapisujemy
% poniższą funkcję

function rysuj_stezenie(czas),
% Funkcja rysująca wykres stężenia leku w czasie

t=czas(1):0.01:czas(2);     % wektor punktów czasu
s = zeros(1,length(t));     % miejsce na wielkość dawki

% Pętla wyznaczania przebiegu dozowania:
for i=1:length(t),
    s(i)=stezenie_leku(t(i));
end
subplot(2,1,2); plot(t,s);  % rysowanie dozowania
xlabel('czas [h]');  ylabel('Stezenie TPT');
```

```
% W osobnym pliku stezenie_leku.m zapisujemy poniższą
% funkcję

function x = stezenie_leku(t),
% Funkcja wyznaczająca ilość leku w osoczu w chwili t.

% Przykładowy przebieg stężenia leku (czas t [h]):
stezenie_max = 0.15;  % Stężenie maksymalne
% Czasy przełączeń dawkowania
ti=24*[5,7,12,22,27,29,34,44,49,51,56];
ind=find(ti<t,1,'last');
if ind/2-floor(ind/2) > 0, x = 0;
else                       x = stezenie_max;
end
```

Wersja tego programu wyposażona w wygodny i dopracowany graficznie interfejs użytkownika dostępna jest na wiadomej stronie internetowej. Widok ekranu dostępnego dla użytkownika przy korzystaniu z tego programu przedstawiony jest na rysunku 4.32, a przykładowa ilustracja graficzna działania modelu pokazana jest na wykresach przedstawionych na rysunku 4.33.



Rys. 4.32. Okno gotowego programu służącego do symulacji przebiegu leczenia guza nowotworowego



Rys. 4.33. Wynik przykładowej symulacji przebiegu choroby w okresie dwóch tygodni (lek podawany w 3-7 i 8-14 dobie)

4.7.5. Model trzech sprzężonych procesów biologicznych. Sprzężenie szeregowe i równoległe.

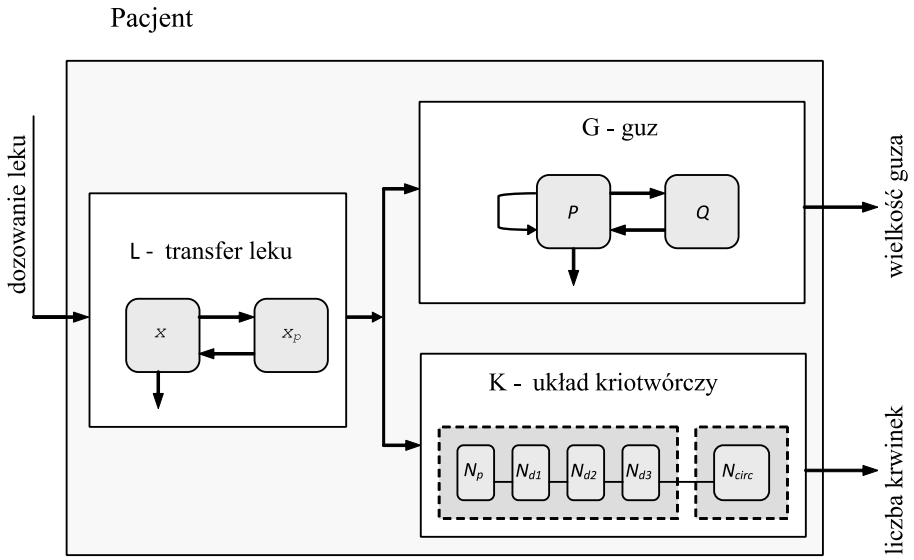
Opisany wyżej model zwalczania choroby nowotworowej przez podawanie odpowiedniego leku (TPT) w wyniku czego obserwowaliśmy zmienne w czasie stężenia tego leku *x(t)* w osoczu mógł sugerować, że aby wyleczyć pacjenta trzeba mu podawać lek jak najczęściej i w jak największych dawkach. Jest to niestety obraz wyidealizowany.

Wyżej napisano, że TPT zakłóca proces replikacji DNA powodując obumieranie dzielącej się komórki. Jest to prawda, bo na tym polega właśnie działanie tego leku. Stosowanie TPT jednak ma także skutki uboczne bo jego działanie nie jest selektywne. Niszczy zarówno komórki nowotworowe jak i inne, zdrowe, naturalnie dzielące się komórki organizmu będące w fazie replikacji DNA. To uboczne oddziaływanie najbardziej uderza w komórki o krótkiej żywotności, a zatem często się odnawiające się. Do takich należą **neutrofile** (stanowiące 45-70% białych krwinek, żyjące tylko ok. 10 godzin). Należy zauważyć, że doprowadzenie do obniżenia ich liczebności z normalnego stanu 1500/µl do poziomu poniżej 500/µl zagraża życiu pacjenta. Leczenie musi więc przebiegać w taki sposób, by maksymalnie skutecznie niszczyć raka możliwie najmniej uszkadzając przy tym układ krwiotwórczy pacjenta. Przy planowaniu takiego leczenia przydatny może być model, który spróbujemy teraz przedstawić. Model ten będzie przy okazji ilustracją opisanej wcześniej metodyki tworzenia modeli złożonych systemów poprzez łączenie modeli elementów składowych w układzie szeregowym i w układzie równoległym.

Rozważany model ograniczony będzie do trzech tylko procesów, istotnych z punktu widzenia prowadzonych tu rozważań:

- narastania guza nowotworowego (oznaczmy go skrótowo przez G),
- wytwarzania komórek krwi w szpiku kostnym (K),
- transferu (przemieszczania się) leku w organizmie (L).

Omówmy najpierw ogólną strukturę modelu, przedstawioną na rysunku 4.34.



Rys. 4.34. Ogólna struktura rozważanego modelu

Jeżeli lek jest podawany dożylnie, to najpierw pojawia się on w osoczu krwi. Stamtąd przenika do innych tkanek, także do chorych komórek guza. W istocie przenikanie leku między tkankami a osoczem ma charakter dwukierunkowy, ale nie będziemy tu wnikać w szczegóły tego procesu, poprzestając na stwierdzeniu, że osocze jest pośrednikiem w transferze leku do guza i innych tkanek. To „pośrednictwo" narzuca **szeregową** strukturę połączenia podsystemu L z G i K. Natomiast uboczne oddziaływanie leku na komórki krwi i pracę układu krwiotwórczego ma charakter **równoległy** do jego oddziaływania na guz. Przenoszone w osoczu TPT „równolegle" dociera do komórek guza i szpiku, dlatego mówimy, że podsystemy K i G są połączone równolegle. Sygnałem scalającym te dwa bloki na wejściu będzie stężenie leku w osoczu krwi pojawiające się na wejściu obu układów. Na wyjściu te dwa bloki też są związane, chociaż nie pokazano tego na rysunku 4.34, bo sygnały wyjściowe obu bloków składają się łącznie na jeden wypadkowy efekt: stan zdrowia pacjenta.

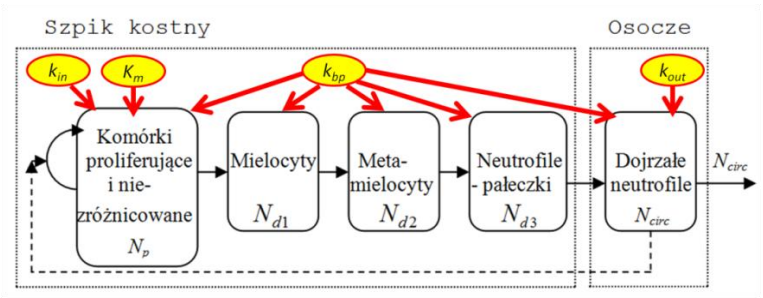
Proces narastania guza nowotworowego oznaczony na rysunku 4.34 symbolem G został opisany i zamodelowany wyżej (Przykład 4.8).

Procesy zachodzące w układzie krwiotwórczym oznaczone na rysunku 4.34 symbolem K opiszemy w następujący sposób:

Wyjściem rozważanego systemu są **neutrofile** – komórki odgrywające istotną rolę w procesach odpornościowych. To dzięki nim organizm broni się przed chorobami, dlatego liczba tych komórek cyrkulujących wraz z krwią, oznaczona dalej jako N_{circ}, jest tak ważnym parametrem. Neutrofile – podobnie jak i inne komórki krwi – są wytwarzane w szpiku kości z komórek

macierzystych. W zdrowym organizmie funkcjonuje system „samoregulacji" (nazywanej często homeostazą), dążący do stabilizacji różnych ważnych życiowo parametrów chemicznych, fizycznych i biologicznych wnętrza ciała człowieka. W odniesieniu do opisywanych tu procesów przejawia się on w tym, że gdy liczba neutrofili jest mniejsza od liczby przyjmowanej w rozważanym organizmie jako wartość normalna (wartość ta zadawana będzie w modelu za pomocą parametru K_m) – wzrasta szybkość wytwarzania nowych komórek macierzystych (ich liczbę oznaczymy N_p) i różnicowania zawiązków nowych komórek właśnie w kierunku neutrofili. I odwrotnie – gdy liczba neutrofili jest za duża, to tempo odpowiednich przemian maleje. Działa tu więc swoiste sprzężenie zwrotne, wiążące tempo wytwarzana w szpiku kostnym komórek macierzystych dających początek procesowi tworzenia neutrofili – z liczbą tych ostatnich N_{circ} aktualnie zawieszonych w osoczu krwi i krążących wraz z nim w całym organizmie. Jednak modelowaniem sprzężenia zwrotnego w systemach biologicznych zajmiemy się (na innym przykładzie) w następnym rozdziale, więc tu na odpowiednim schemacie (Rys. 4.35) obecność tego sprzężenia zwrotnego będzie jedynie zasygnalizowana linią przerywaną.

Wzrost liczy proliferujących komórek macierzystych N_p nie przekłada się od razu na liczbę krążących we krwi neutrofili N_{circ}**.** Odpowiednie procesy są dość złożone. Zawiązek określonego rodzaju komórki wytworzony z komórek macierzystych przechodzi kilka faz zanim stanie się dojrzałą komórką. W przypadku linii neutrofili tymi fazami pośrednimi są **mielocyty** (ich liczbę oznaczymy N_{d1}) **metamielocyty** (ich liczbę oznaczymy N_{d2}) oraz **neutrofile pałeczkowate** (ich liczbę oznaczymy N_{d3}). Proces przekształcania się jednej postaci komórki w kolejną nie jest natychmiastowy. Będziemy zakładali, że każdą z tych faz opisywać będzie kolejne liniowe równanie różniczkowe, ale dla uproszczenie przyjmiemy, że szybkość odpowiednich procesów jest w każdym z etapów związana z tym samym współczynnikiem oznaczonym k_{bp}. Jak widać proces działania systemu regulacji liczby komórek krwi jest rozłożony w czasie i ma swoją dynamikę.



Rys. 4.35. Schemat produkcji neutrofili

Te dwa zjawiska – etapowego charakteru tworzenia dojrzałych komórek oraz stabilizacji ich liczby ilustruje w uproszczeniu schemat przedstawiony na

rysunku 4.35. Na rysunku tym obok sygnałów N_p , N_{d1} , N_{d2} , N_{d3} , N_{circ} reprezentujących odpowiednie liczby rozważanych komórek w poszczególnych fazach cyklu rozwojowego – zaznaczono także parametry k_{in} , K_m , k_{bp}, k_{out} determinujące ten proces. Parametry te można zmieniać w trakcie eksploatacji modelu obserwując ich wpływ na badane zjawisko.

Ilościowe zależności szybkości tworzenia komórek wytwarzanych na podstawie komórek macierzystych od liczebności neutrofili krążących we można zapisać w postaci równania różniczkowego:

$$\frac{dN_p}{dt} = \left(k_{in} \frac{K_m}{K_m + N_{circ}} - k_{bp} \right) N_p \tag{4.43}$$

Natomiast proces dojrzewania komórek macierzystych w kolejnych stadiach formowania się neutrofili można zapisać w postaci zespołu równań różniczkowych:

$$\begin{aligned} \frac{dN_{d1}}{dt} &= k_{bp}\left(N_p - N_{d1}\right) \\ \frac{dN_{d2}}{dt} &= k_{bp}\left(N_{d1} - N_{d2}\right) \\ \frac{dN_{d3}}{dt} &= k_{bp}\left(N_{d2} - N_{d3}\right) \\ \frac{dN_{circ}}{dt} &= k_{bp}N_{d3} - k_{out}N_{circ} \end{aligned} \tag{4.44}$$

gdzie:

- N_p - liczba (stężenie) komórek proliferujących oraz w fazie różnicowania (w tym: mieloblasty i promielocyty),
- N_{d1}, N_{d2}, N_{d3} - liczby zawiązków neutrofili w fazach „przejściowych” – (mielocyty i metamielocyty obojętnochłonne oraz neutrofile pałeczkowate),
- N_{circ} - liczba dojrzałych neutrofili krążących w krwi obwodowej,
- k_{in} - współczynnik szybkości wytwarzania komórek macierzystych i różnicowania ich w kierunku neutrofili,
- k_{bp} - współczynnik szybkości przekształcania się zawiązków z jednej fazy w kolejną,

- k_{out} - współczynnik szybkości naturalnego obumierania neutrofili,

- K_m - parametr „połowicznego nasycenia" określający ustalony poziom liczby neutrofili.

Jest to – jak widać - model ze sprzężeniem zwrotnym stabilizującym liczbę neutrofili w krwi. W równaniu (4.43) można dostrzec zależność tempa wytwarzania nowych komórek proliferujących od liczby neutrofili w krwi. Gdy zmniejsza się liczba neutrofili N_{circ}, to wzrasta szybkość wytwarzania komórek macierzystych N_p i różnicowania ich w kierunku neutrofili – i odwrotnie.

Opisany wyżej model odpowiadał sytuacji funkcjonowania układu krwiotwórczego osoby całkowicie zdrowej. Niestety ten doskonały system biologiczny zostaje brutalnie zakłócony gdy – w celu walki z rakiem – pacjentowi podamy wzmiankowany wyżej lek zwany TPT. W trakcie takiej chemoterapii naturalny proces tworzenia nowych komórek krwi – w tym także neutrofili – jest hamowany obecnością TPT. Zjawisko to uwzględnia się w modelu wprowadzając do równania (4.43) dodatkowy czynnik w postaci prostej funkcji wymiernej. Zmodyfikowane równanie przyjmuje postać

$$\frac{dN_p}{dt} = \left(k_{in} \frac{K_m}{K_m + N_{circ}} \frac{IC_{50}}{IC_{50} + x(t)} - k_{bp} \right) N_p \tag{4.45}$$

gdzie:

- IC_{50} - parametr charakteryzujący wrażliwość mechanizmu krwiotwórczego na zmiany stężenia TPT we krwi

- $x(t)$ - zmiany stężenia we krwi TPT wywołane jego podawaniem pacjentowi w celach leczniczych.

Jak łatwo wywnioskować ze wzoru (4.45) powiększanie stężenia TPT w osoczu - reprezentowanego w modelu (dodatnimi) wartościami funkcji $x(t)$ - powoduje zmniejszenie tempa wytwarzania nowych komórek, a co za tym idzie – zmniejszanie się liczby neutrofili. Oznacza to, że lecząc raka za pomocą leku TPT – jednocześnie osłabiamy system odpornościowy pacjenta, co może go narażać na niebezpieczeństwo chorób zakaźnych. Dlatego model, który tu omawiamy, ma znaczenie nie tylko jako przykład zastosowania technik modelowania i symulacji komputerowej, ale może (po zidentyfikowaniu jego parametrów u konkretnego pacjenta) być wykorzystywany do optymalizacji terapii raka – w sensie maksymalizacji efektu terapeutycznego przy równoczesnej minimalizacji szkodliwych skutków ubocznych związanych z systemem krwiotwórczych. Zapisanie modelu w MATLABie wymaga utworzenia funkcji, w której zapisane są równania różniczkowe modelu. Elementy odpowiedniego programu przedstawione są w przykładzie 4.9.

Prezentowany model można zastosować do symulacji zmian liczebności

także innych komórek krwi (w pierwszej kolejności – płytek krwi, których niedomiar jest kolejnym zagrożeniem dla życia). Jedyną modyfikacją byłaby zmiana wartości parametrów modelu.

W dotychczas omawianych modelach jedną z wielkości wejściowych było stężenie TPT w osoczu krwi. Granicą między modelowanym obiektem (guzem, szpikiem kostnym) a jego otoczeniem był układ krwionośny pacjenta. Bardziej użyteczny będzie model, w którym ta granica nie będzie prowadzona wewnątrz człowieka lecz poza jego ciałem. Wymaga to uwzględnienia w modelu pozostałych tkanek i narządów. Ta część organizmu może mieć bowiem istotne znaczenie dla procesu rozprowadzania leku w organizmie.

Przykład 4.9 – Model wytwarzania neutrofili

```
% Ustawienie parametrów symulacji:
czas = [0,336];  % przedział czasu [h] (2 tygodnie)
% Początkowa liczebność komórek:
Np0=2000; Nd10=2300; Nd20=2400; Nd30=2100;
Ncirc0=1500;

% Symulacja:
N0 = [Np0; Nd10; Nd20; Nd30; Ncirc0]; % stan początkowy
[t,N] = ode45(@rownanie_ukladu_krwiotworczego,czas,N0);

% Wyprowadzenie wyników w formie graficznej:
subplot(2,1,1), plot(t,N(:,5)); % wykres liczebności
title(['Symulacja pracy ukladu krwiotworczego w'...
   'obecnosci TPT']);
xlabel('czas [h]');
ylabel('Liczba neutrofili w krwi {\itN_{circ}}');
rysuj_stezenie(czas); % wykres przebiegu dozowania leku

% Używane tu funkcje dozowanie_leku oraz
% rysuj_dozowanie są przedstawione w przykładzie 4.8
```

```
% W osobnym pliku rownanie_ukladu_kriotworczego.m
% zapisujemy poniższą funkcję

function dN_dt = rownanie_ukladu_krwiotworczego(t,N),
% Funkcja obliczająca prawą stronę równ. różniczkowego
% 6. element wektora N (jeżeli istnieje) jest
% wartością wejściową - stężeniem leku.

% Obliczenie dozowania leku w chwili t:
if length(N) == 6, x = N(6);
else               x = stezenie_leku(t);
end

% Parametry modelu:
kin = 0.09;  % tempo produkcji komórek macierzystych
             % [1/h]
Km = 3000;   % parametr "połowy nasycenia" Ncirc [1/ul]
IC50 = 1.5;  % parametr "połowy nasycenia" TPT [ng/ul]
kbp = 0.07;  % tempo różnicowania [1/h]
kout = 0.1;  % tempo naturalnego obumierania neutrofili
             % [1/h]

% Równania różniczkowe:
Np = N(1); Nd1 = N(2); Nd2=N(3); Nd3=N(4); Ncirc=N(5);
%pobranie z wektora
dNp_dt = (kin*Km/(Km+Ncirc)*IC50/(IC50+x))*Np - kbp*Np;
dNd1_dt = kbp*(Np-Nd1);
dNd2_dt = kbp*(Nd1-Nd2);
dNd3_dt = kbp*(Nd2-Nd3);
dNcirc_dt = kbp*Nd3 - kout*Ncirc;
dN_dt = [dNp_dt; dNd1_dt; dNd2_dt; dNd3_dt; dNcirc_dt];
```

Najbardziej uproszczony model – w farmakokinetyce nazywany jednokompartmentowym – zakłada, że szybkość wchłaniania, metabolizmu i wydalania leku jest wprost proporcjonalna do jego stężenia w kompartmencie (tkance), w którym zachodzi dany proces. W naszym przypadku, gdy rozważamy tylko jeden sposób podawania leku – poprzez iniekcję dożylną - tkanką tą jest osocze krwi. Modelem matematycznym wiążącym używany wcześniej przebieg stężenia leku TPT w osoczu krwi $x(t)$ z zewnętrznymi dostawami (iniekcjami) tego leku $x_z(t)$ jest proste równanie różniczkowe

$$\frac{dx}{dt} = -k_e\, x + x_z(t) \tag{4.46}$$

gdzie:

- k_e - współczynnik szybkości naturalnego zanikania TPT w osoczu,

- $x_z(t)$ - ilość podawanego TPT na jednostkę czasu.

Zapisanie takiego modelu w MATLABie zapewne nie będzie dla Czytelnika dużą trudnością więc nie podamy tu gotowego rozwiązania, proponując zapisanie i przetestowanie tego modelu jako pożyteczne ćwiczenie.

W praktyce częściej jest stosowany model dwukompartmentowy. Wyróżnia się w nim kompartment centralny – osocze krwi oraz kompartment obwodowy. Ten ostatni obejmuje wszystkie tkanki, do których cząsteczki leku mogą wnikać (i z niego powracać) poprzez osocze krwi. W ten sposób uwzględnia się procesy przenikania leku pomiędzy tkankami, co zbliża model do rzeczywistych zjawisk w organizmie. Model matematyczny dwukompartmentowy związany z przekazywaniem podawanego z zewnątrz leku TPT do tkanek zapiszemy równaniami:

$$\frac{dx}{dt} = -\left(k_e + k_{cp}\right)x + k_{pc}x_p + x_z(t) \tag{4.47}$$

$$\frac{dx_p}{dt} = k_{cp}\,x - k_{pc}x_p \tag{4.48}$$

gdzie:

- x - stężenie TPT w strefie centralnej – osoczu (ten sygnał był uwzględniany we wcześniejszych modelach),
- x_p - stężenie TPT w strefie peryferyjnej,
- $x_z(t)$ - dozowanie - ilość podawanego z zewnątrz TPT na jednostkę czasu,
- k_{cp} - współczynnik szybkości przenikania TPT z strefy centralnej do peryferyjnej,
- k_{pc} - współczynnik szybkości przenikania TPT z strefy peryferyjnej do centralnej.

Opisany wyżej model możemy teraz zapisać w MATLABie w postaci podanej w Przykładzie 4.10.

Przykład 4.10 – Model transferu TPT

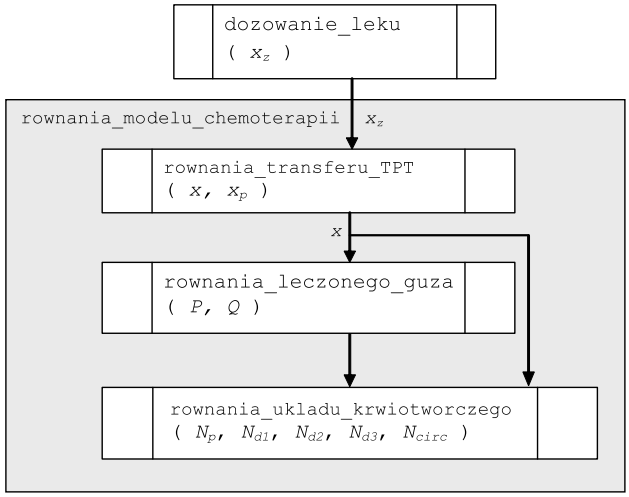
```
% Ustawienie parametrów symulacji:
czas = [0,336];  % przedział czasu [h] (2 tygodnie)
x0=0;  xp0=0;    % początkowe stężenia TPT:

% Symulacja:
[t,X] = ode15s(@rownanie_transferu_TPT,czas,[x0; xp0]);

% Wyprowadzenie wyników w formie graficznej:
subplot(2,1,1),        plot(t,X(:,1),'b',t,X(:,2),'r');
title('Symulacja transferu leku TPT');
xlabel('czas [h]');
ylabel('Stezenia leku w osoczu');
legend('{\itx(t)}','{\itx_p(t)}','Location','Best');
rysuj_dozowanie(czas); % wykres przebiegu dozowania
                       % leku
```

```
% W osobnym pliku rownanie_transferu_TPT.m
% zapisujemy poniższą funkcję

function dX_dt = rownanie_transferu_TPT(t,X),
% Funkcja obliczająca prawą stronę równ.różniczkowego

% Parametry modelu:
ke = 0.8;  % wsp. naturalnego zaniku TPT w osoczu [1/h]
kcp = 0.26; % wsp. przenikania TPT z strefy centralnej
            % do peryferyjnej [1/h]
kpc = 0.27; % wsp. przenikania TPT z strefy
            % peryferyjnej do centralnej [1/h]

xz = dozowanie_leku(t);% wyznaczenie dozowania w chwili t

x = X(1); xp = X(2); % pobranie z wektora
% równania różniczkowe
dx_dt = -(ke+kcp)*x + kpc*xp + xz;
dxp_dt = kcp*x - kpc*xp;
dX_dt = [dx_dt;dxp_dt]; % spakowanie do wektora
```

```
% W osobnym pliku dozowanie_leku.m
% zapisujemy poniższą funkcję

function xz = dozowanie_leku(t),
% Funkcja wyznaczająca wielkość podawanej dawki leku
% w chwili t
% 3 cykle po: (5 dni TPT, 2 dni odpoczynku, 5 dni TPT,
% 10 dni odpoczynku)

xz_max = 0.12;
% Czasy przełączeń dawkowania
ti=24*[5, 7, 12, 22, 27, 29, 34, 44, 49, 51, 56, 100];
ind=find(ti<t,1,'last');
if ind/2-floor(ind/2) > 0, xz = 0;
else                       xz = xz_max;
end
```

```
% W osobnym pliku rysuj_dozowanie.m
% zapisujemy poniższą funkcję

function rysuj_dozowanie(czas),
% Funkcja rysująca wykres dozowania leku w czasie

T = czas(1):0.01:czas(2);    % wektor punktów czasu
TPT= zeros(1,length(t));     % miejsce na wielkość dawki

% Pętla wyznaczania przebiegu dozowania:
for i=1:length(t),
    TPT(i) = dozowanie_leku(t(i));
end
subplot(2,1,2); plot(t,TPT); % rysowanie dozowania
xlabel('czas [h]');  ylabel('Dozowanie TPT');
```

Ogólny schemat modelu łączącego wszystkie wymienione wyżej elementy przedstawia rysunek 4.36.



Rys. 4.36. Model chemoterapii na wyższym poziomie hierarchii

Przykład 4.11 – Całościowy model chemoterapii

```
% Ustawienie parametrów symulacji:
czas = [0,1464];      % przedział czasu [h] (61 dni)

% Stan początkowy:
P0 = 2e8; Q0 = 8e8;   % liczba komórek guza

% Liczby krwinek
Np0=2000; Nd10=2300; Nd20=2400; Nd30=2100; Ncirc0=1500;
x0=0; xp0=0;          % stężenia TPT

% Symulacja:
S0 = [P0; Q0; Np0; Nd10; Nd20; Nd30; Ncirc0; x0; xp0];
[t,S] = ode45(@rownania_modelu_chemoterapii,czas,S0);

% Rysowanie wykresów:
subplot(3,1,1), plot(t,S(:,8:9));
ylabel('Stezenie TPT');
title('Symulacja calosciowego modelu chemoterapii');
subplot(3,1,2), plot(t,S(:,1:2));
ylabel('Liczebnosc komorek guza');
subplot(3,1,3), plot(t,S(:,5));
ylabel('Liczba neutrofili');
xlabel('czas [h]');
```

```
% W osobnym pliku równania_modelu_chemoterapii.m
% zapisujemy poniższą funkcję

function dS_dt = rownania_modelu_chemoterapii(t,S),
% Funkcja obliczająca prawą stronę równ. różniczkowych
% korzystająca z wcześniej zdefiniowanych podsystemów

% Równania podsystemów:
dX_dt = rownanie_transferu_TPT(t,S(8:9));
stezenie_TPT_w_osoczu = S(8);
dPQ_dt = rownanie_leczonego_guza(t,...
    [S(1:2);stezenie_TPT_w_osoczu]);
dN_dt = rownanie_ukladu_krwiotworczego(t,...
    [S(3:7);stezenie_TPT_w_osoczu]);

% Wstawienie wyniku do wektora wyjściowego:
dS_dt = [dPQ_dt; dN_dt; dX_dt];
```

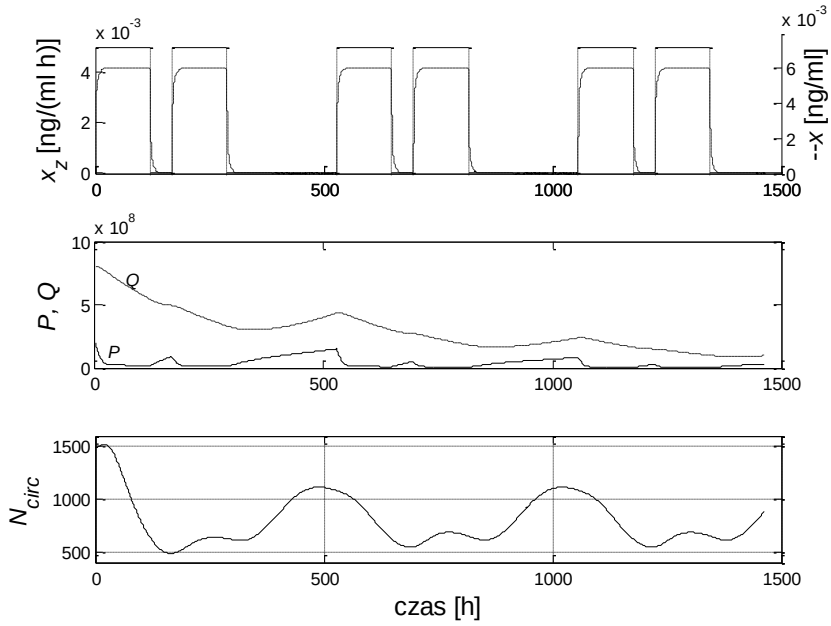
Modułowa budowa modelu ułatwia jego modyfikację. Można np.:

- dla kontroli liczby płytek krwi – rozbudować model „system krwiotwórczy" równoległy blok reprezentujący wytwarzanie trombocytów, różniący się od opisanego jedynie wartościami stałych parametrów,
- w przypadku podawania leku inną drogą – wymienić blok modelu transferu leku,
- w przypadku wspomagania terapii podawaniem glikoproteiny[20] - dodać drugi sygnał wejściowy, analogiczny blok transferu leku oraz zmodyfikować model systemu krwiotwórczego.

Model pozwala przeprowadzać symulacje przebiegu choroby przy różnych strategiach dozowania leku reprezentowanych funkcją $x_z(t)$. Pozwala to oceniać postępy kuracji oraz weryfikować, czy rozważana strategia nie prowadzi do przekroczenia zadanych ograniczeń, np. maksymalnego stężenia leku oraz minimalnego poziomu neutrofili i trombocytów.

Przykładowy wynik symulacji uzyskanej z pomocą omawianego wyżej modelu (ale w wersji wyposażonej w dobre interfejsy graficzne, dostępnej na stronie internetowej www.Tadeusiewicz.pl) przedstawiono na rysunku 4.37.

[20] Glikoproteina jest białkiem, które powoduje m.in. przyspieszenie procesu wytwarzania w szpiku komórek krwi.



Rys. 4.37. Przykładowy wynik symulacji przebiegu leczenia w trzech cyklach

Warto zaznaczyć, że modele na prezentowanym w tym rozdziale poziomie dokładności odwzorowania rzeczywistych procesów są stosowane w praktyce do wyznaczania optymalnego dozowania TPT, indywidualnie dla każdego pacjenta[21].

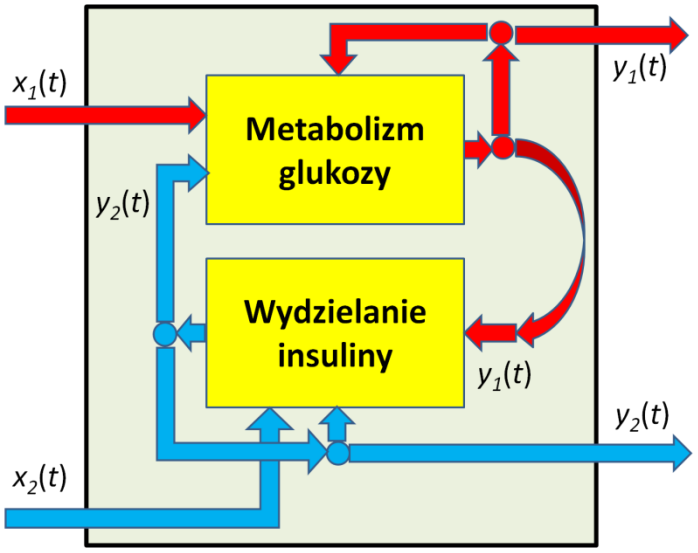
4.7.6. Model systemu ze sprzężeniem zwrotnym na przykładzie zależności glukoza - insulina

Kolejnym modelem dynamicznym typu 2 może być model metabolizmu węglowodanów prognozujący stężenie cukru we krwi. Model ten wnosi do prezentowanych tu rozważań nową jakość, ponieważ pokazuje przykład modelowania dwóch współzależnych procesów biologicznych formujących typowy układ ze sprzężeniem zwrotnym. W systemie tym określać będziemy zmienność stężenia glukozy we krwi (oznaczoną jako y_1) oraz zmienność ilości insuliny we krwi (oznaczoną jako y_2). Zarówno glukoza jak i insulina ulegają w organizmie zużyciu, co będą opisywały odpowiednie równania różniczkowe, a także są dostarczane z zewnątrz w postaci pożywienia x_1 będącego źródłem

[21] Wartości parametrów występujących w modelu można znaleźć np. we wspomnianym wcześniej artykule *Collins C, Fister K R, Key B, Williams M: „Blasting neuroblastoma using optimal control of chemotherapy Mathematical Biosciences and Engineering”.*

glukozy jak i w postaci zastrzyków insuliny x_2, które uzupełniają jej zapas w organizmie. Oczywiście wszystkie wymienione wyżej sygnały są funkcjami czasu, to znaczy mamy do czynienia z czterema przebiegami mającymi swoją reprezentację w czasie: $x_1(t)$, $x_2(t)$, $y_1(t)$, $y_2(t)$, przy czym dwa z nich można traktować jako wejściowe ($x_1(t)$ i $x_2(t)$), a dwa jako wyjściowe ($y_1(t)$ i $y_2(t)$).

Nowością, którą ten model wnosi do prowadzonych tu rozważań jest fakt, że wielkości wyjściowe oddziaływają na siebie wzajemnie. Zmiany ilości insuliny $y_1(t)$ wpływają na zmiany poziomu glukozy $y_2(t)$, a zmiany poziomu glukozy $y_1(t)$ wywołują zmiany ilości produkowanej (w trzustce) insuliny $y_1(t)$. Ilustruje to rysunek 4.38.



Rys. 4.38. Model dynamiki i wzajemnego oddziaływania glukozy i insuliny

Zamodelowanie procesów zachodzących w tym systemie pozwoli zrozumieć zależności między stężeniem glukozy i ilością insuliny we krwi a także unaoczni różnice pomiędzy osobami zdrowymi, a pacjentami chorymi na cukrzycę. Model ten może mieć także zastosowania praktyczne, ponieważ pozwala określić parametry terapii cukrzycy (m. in. dawki insuliny, czas ich podawania). Mając do dyspozycji taki model można zaprojektować też urządzenia dozujące insulinę. Model glukoza – insulina jest bardzo istotny w przypadku chorych na cukrzycę, u których wartości insuliny i glukozy muszą podlegać stałej obserwacji.

Najpierw przypomnijmy kilka faktów biologicznych, na których oprzemy potem konstrukcję modelu.

Podstawowym materiałem energetycznym dla człowieka są węglowodany pozyskiwane z owoców, warzyw i produktów zbożowych. Aby węglowodany, pobrane wraz z pożywieniem, mogły zostać użyte przez ludzkie komórki muszą ulec rozłożeniu na cukry proste: glukozę, fruktozę i galaktozę. Glukoza przenika

do krwi i jest rozprowadzana po całym organizmie. Zadaniem insuliny jest aktywowanie procesu prowadzącego do przejścia glukozy przez naczynia krwionośne do komórek. Insulina jest hormonem wytwarzanym przez trzustkę i zapewnia zwiększenie transportu glukozy do komórek, co obniża poziom cukru we krwi. Niedobór lub nieprawidłowe działanie insuliny, czego skutkiem jest hiperglikemia, jest główną przyczyną cukrzycy.

Ze względu na źródło choroby wyróżniamy dwa rodzaje cukrzycy: typu I oraz typu II. Cukrzyca typu II jest chorobą cywilizacyjną, na którą cierpią przede wszystkim osoby starsze i otyłe. Charakteryzuje się opornością na insulinę. Cukrzyca typu II, czyli insulinoniezależna, nie wymaga zewnętrznego podawania insuliny, lecz leków zmniejszających oporność. Cukrzyca typu I, zwana cukrzycą insulinozależną, wywołana jest uszkodzeniem trzustki i charakteryzuje się zmniejszoną ilością produkcji insuliny. Osoby cierpiące na cukrzycę typu I muszą zewnętrznie aplikować insulinę lub stosować pompy insulinowe tzw. sztuczne trzustki.

Analizując powyższe rozważania można wyróżnić trzy przypadki modelu: osoba zdrowa, pacjent cierpiący na cukrzycę typu I oraz pacjent z cukrzycę typu II. Wszystkie te przypadki możemy zamodelować przy pomocy jednego opisu matematycznego z odpowiednim ustawieniem parametrów[22].

Najważniejszymi składnikami naszego modelu są sygnały $y_1(t)$ oraz $y_2(t)$ przedstawiające zawartość odpowiednio glukozy oraz insuliny we krwi. Jak widać oba te sygnały są zmienne w czasie. Głównym zadaniem układu glukoza-insulina jest zachowanie homeostazy w organizmie, co oznacza utrzymanie glukozy na stałym poziomie M, charakterystycznym dla każdego człowieka, równym ok. $100\left[\frac{mg}{dl}\right]$. W stanie równowagi, gdy $y_1=M$, występuje brak insuliny we krwi, czyli $y_2=0$. Taka sytuacja jest zauważana podczas długiego braku spożywania posiłku, głodu, czy podczas badania na czczo.

Równanie opisujące poziom glukozy we krwi przedstawione jest przy pomocy równania różniczkowego. Zależne jest zarówno od ilości spożytego pokarmu, wartości insuliny we krwi jak i parametrów charakterystycznych dla danego przypadku. Wariant górny przedstawia sytuację po spożyciu posiłku, kiedy $y_1(t) > M$, zaś wariant dolny sytuację równowagi (4.49).

$$\frac{dy_1}{dt} = \begin{cases} -a_1 y_1 y_2 + b_1 x_1 & dla\ y_1(t) > M \\ -a_1 y_1 y_2 + a_2(M - y_1) + b_1 x_1 & dla\ y_1(t) \le M \end{cases} \tag{4.49}$$

Równanie dolne przedstawia także sytuację, gdy wartość y_1 spadnie poniżej stanu równowagi M. Wtedy cukier wytwarzany jest przez wątrobę w ilości proporcjonalnej do $M - y_1$.

---

[22] Model ten oparty został na książce Stanisława Osowskiego zatytułowanej: „*Modelowanie układów dynamicznych”*, (Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 1997).

Proces utylizacji glukozy polega na tym, że jej zawartość w osoczu krwi maleje, w związku z tym odpowiedni składnik poprzedzony jest znakiem minus i jest proporcjonalny do iloczynu y_1 y_2. Sygnał wejściowy x_1 jest miarą spożytego jedzenia, a przenikanie glukozy do krwi zmienia się wykładniczo (4.50). Parametry: a_1, a_2, b_1 są charakterystyczne dla danego przypadku, a ich wartości przedstawia tabela 4.2.

$$x_1(t) = R_1 e^{-\frac{t-t_0}{\tau}} \tag{4.50}$$

Kolejnym bardzo ważnym równaniem jest zależność opisująca dynamiczne zmiany insuliny we krwi. Podobnie jak przy glukozie, zależność ta jest opisywana przy pomocy równania różniczkowego. U zdrowego człowieka insulina jest wytwarzana, gdy $y_1(t) > M$, czyli kiedy poziom cukru przekroczy poziom homeostazy. Insulina jest wytwarzana tak długo, aż zostanie osiągnięty poziom równowagi. Równanie przedstawiające charakterystykę insuliny przedstawia wzór (4.51).

$$\frac{dy_2}{dt} = \begin{cases} -a_3(y_1 - M) - a_4 y_2 + b_2 x_2 & dla\ y_1(t) > M \\ -a_4 y_2 + b_2 x_2 & dla\ y_1(t) \le M \end{cases} \tag{4.51}$$

Równanie górne przedstawia sytuację, kiedy mamy nadmiar glukozy we krwi. Wtedy wytwarzana jest insulina proporcjonalna do $y_1 - M$. Człon ujemny podobnie jak przy glukozie przedstawia proces utylizacji insuliny. Parametr x_2 jest miarą insuliny dawkowanej zewnętrznie i w przypadku zarówno osób zdrowych jak i pacjentów z cukrzycą typu II wynosi zero. W przypadku pacjentów z cukrzycą typu I konieczne jest podanie insuliny i wzór 4.52 opisuje przenikanie jej do krwi.

$$x_2(t) = R_2 e^{-\frac{t-t_0}{\tau}} \tag{4.52}$$

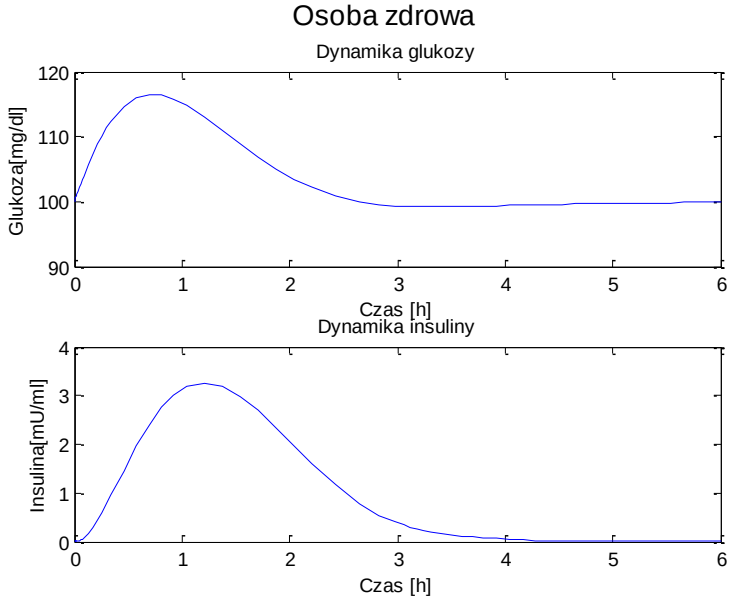
Parametry: a_3, a_4, b_2 są charakterystyczne dla danego przypadku, a ich wartości przedstawia tabela 4.2.

Tabela 4.2 Przedstawienie parametrów symulacyjnych dla przypadków: osoba zdrowa, cukrzyca typu I, cukrzyca typu II

Parametry	**Osoba zdrowa**	**Cukrzyca typu I**	**Cukrzyca typu II**
a_1	0.05	0.05	0.0001
a_2	1	1	1
a_3	0.5	0	0.5
a_4	2	2	2
b_1	1	1	1
b_2	1	1	1

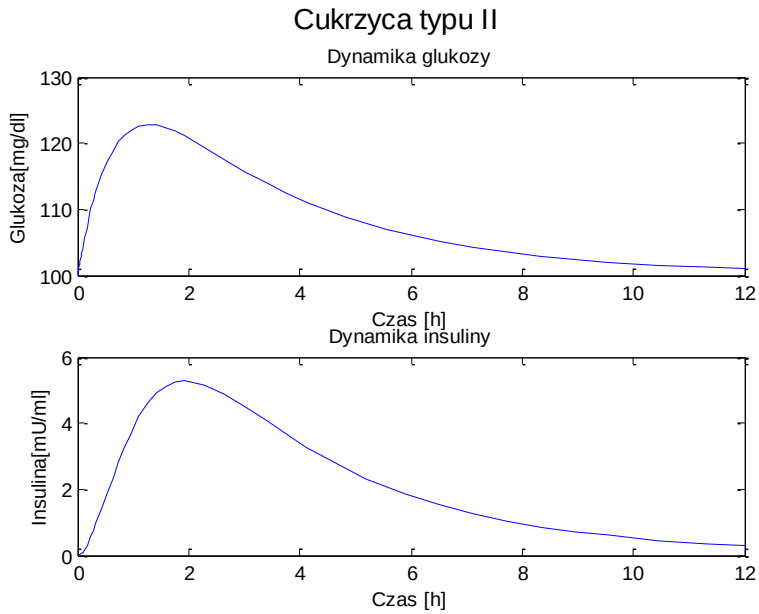
Wykresy na rysunkach 4.39 - 4.42 przedstawiają wyniki symulacji zachowania się glukozy jak i insuliny po spożyciu posiłku.

U osoby zdrowej (Rys. 4.39) wartość insuliny gwałtownie rośnie przy wzroście glukozy. Układ szybko powraca do równowagi.



Rys. 4.39. Symulacja krzywych zmian glukozy i insuliny po spożyciu posiłku dla osoby zdrowej

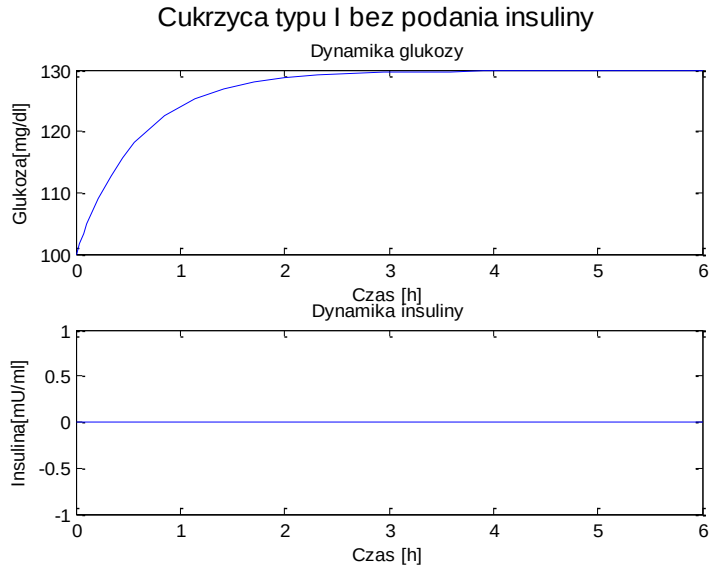
Rysunek 4.40 przedstawia reakcję pacjenta cierpiącego na cukrzycę typu II po spożyciu posiłku. Zauważalne jest zaburzenie działania jak i wydzielania insuliny. Układ dużo później powraca do równowagi.



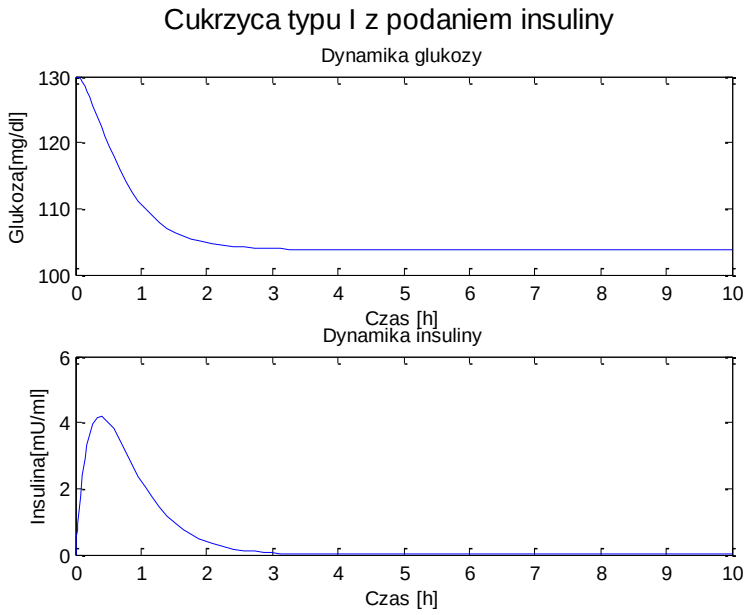
Rys. 4.40. Symulacja krzywych zmian glukozy i insuliny po spożyciu posiłku dla osoby chorej na cukrzycę typu II

Kolejny rysunek (Rys. 4.41) przedstawia krzywą zmian dla pacjenta chorego na cukrzycę typu I. Wykres przedstawia sytuację patologiczną, ponieważ przy znacznym wzroście wartości glukozy, wartość insuliny jest w ogóle nieobecna lub o bardzo małym stężeniu. Układ nie jest w stanie powrócić do stanu równowagi. Taka sytuacja jest bardzo niebezpieczna dla pacjenta, dlatego konieczne jest zewnętrzne podanie dawki insuliny.

Rysunek (Rys. 4.42) przedstawia sytuację kiedy stan początkowy jest podwyższony i następuje wstrzyknięcie zewnętrznej dawki insuliny. Powoduje to chwilowy wzrost stężenia insuliny i powrót układu do równowagi.



Rys. 4.41. Symulacja krzywych zmian glukozy i insuliny po spożyciu posiłku dla pacjenta chorego na cukrzycę typu I bez zewnętrznego podawania insuliny



Rys. 4.42. Symulacja krzywych zmian glukozy i insuliny po spożyciu posiłku dla pacjenta chorego na cukrzycę typu I wraz z zewnętrznym podawaniem insuliny

4.7.7. Symulacja zależności glukoza - insulina

W poprzednim podrozdziale został omówiony model matematyczny układu glukoza - insulina. Model ten, choć jest uproszczony, pozwala na prowadzenie analiz i może zostać wykorzystany w dydaktyce. W celu udostępnienia tego modelu Czytelnikom opracowano stosowne programy w MATLABie i przedstawiono je niżej w formie uproszczonej (Przykład 4.12, Przykład 4.13 i Przykład 4.14) oraz – jak zawsze – w formie pełnej na stronie www.Tadeusiewicz.pl .

Dla rozwiązania równania różniczkowego zwyczajnego pierwszego rzędu skorzystano z funkcji *ode15s*, która jest implementacją tak zwanych metod numerycznego całkowania równań różniczkowych zwyczajnych. Formuła rozwiązywania funkcji jest następująca:

```
[t, dane] = ode15s (@uchwyt_do_funkcji, czas, warunek_początkowy, opcjonalne_opcje);
```

Przykładowe wywołanie funkcji:

Przykład 4.12 – Przykładowe wywołanie funkcji:

```
% Symulacja dla człowieka zdrowego po spożyciu posiłku
[t1, dane] = ode15s(@symulacja, [0 t_s], x1);
```

Program modelu glukoza – insulina dzielimy na dwie części: **skrypt** – służący do ustawiania parametrów, wywoływania funkcji oraz rysowania wykresów; **funkcję** – definiującą sposób rozwiązania równania różniczkowego.

Przykład 4.13 przedstawia funkcję o nazwie *symulacja*, która jako dane wejściowe przyjmuje przedział czasowy oraz parametry dla danego przypadku symulacji.

Przykład 4.13 – Funkcja modelu

```
function dane = symulacja(t, x)

% Parametry symulacji
% Stężenie glukozy [mg/dl]
G = x(1,1);
% Stężenie insuliny [mg/dl]
I = x(2,1);
% Parametry
a1 = x(3,1);
a3 = x(4,1);
in = x(5,1);
M = 100; % Stan równowagi
```

```
% Pożywienie przenikające do krwi po spożyciu posiłku
if(in == 0)
    g_p = 50 * exp(-t/(0.6));
else
    g_p=0;
end

% Przenikanie zewnętrznie podanej dawki insuliny
i_z = in* 30 * exp(-t/(0.3));

% Zmiana zawartości glukozy
if (G < M)
    dG_dt = -a1*G*I+1*(M-G)+1*g_p;
else
    dG_dt = -a1*G*I+1*g_p;
end

%Zmiana zawartości insuliny
if (G >= M)
    dI_dt = a3*(G-M)-2*I+i_z;
else
    dI_dt = -2*I+i_z;
end

% Obliczone wartości
% Wartości prawej strony równań różniczkowych
% równe pochodnym G i I po czasie. Zera na pozycjach
% 3-5 wskazują, że wartości parametrów a1, a3 i in są
% stałe.
dane = [dG_dt; dI_dt; 0; 0; 0];
```

Powyższy kod funkcji zapisujemy w pliku `symulacja.m`. Skrypt, który umożliwia odpowiednie wywołanie funkcji dla wszystkich przypadków opisanych w poprzednim rozdziale przedstawia przykład 4.14.

Przykład 4.14 – Parametry modelu, wywołanie funkcji modelu:

```
% Warunki początkowe - stan równowagi
G_0 = 100 % Wartość stężenia glukozy [mg/dl]
I_0 = 0;  % Wartość insuliny

% Wektor danych początkowych glukozy i insuliny oraz
% parametrów
% Parametry: a1, a3, podanie dawki insuliny
% (0-nie,1-tak)

% Parametry dla osoby zdrowej
x1 = [G_0; I_0; 0.05; 0.5; 0];
```

```
% Parametry dla osoby chorej - cukrzyca typu II
x2 = [G_0; I_0; 0.001; 0.5; 0];

% Parametry dla osoby chorej - cukrzyca typu I,
% bez podania insuliny
x3 = [G_0; I_0; 0.05; 0; 0];

% Parametry dla osoby chorej-cukrzyca typu I z podaniem
% insuliny
x4 = [130; 0; 0.05; 0; 1];

% Czas symulacji [h]
t_s = 6;

% Symulacja dla człowieka zdrowego po spożyciu posiłku
[t1, dane] = ode15s(@symulacja, [0 t_s], x1);

% Wyodrębnienie wartości glukozy oraz insuliny
G1 = dane(:,1);
I1 = dane(:,2);

% Symulacja cukrzycy typu II
[t2,dane] = ode15s(@symulacja, [0 t_s], x2);

% Wyodrębnienie wartości glukozy oraz insuliny
G2 = dane(:,1);
I2 = dane(:,2);

% Symulacja cukrzycy typu I bez podania insuliny
[t3,dane] = ode15s(@symulacja, [0 t_s], x3);

% Wyodrębnienie wartości glukozy oraz insuliny
G3 = dane(:,1);
I3 = dane(:,2);

% Czas symulacji [h]
t_s = 10;

% Symulacja cukrzycy typu I z podaniem insuliny
[t4,dane] = ode15s(@symulacja, [0 t_s], x4);

% Wyodrębnienie wartości glukozy oraz insuliny
G4 = dane(:,1);
I4 = dane(:,2);
```

Następnie prezentujemy otrzymane wyniki przy pomocy wykresów. Dla

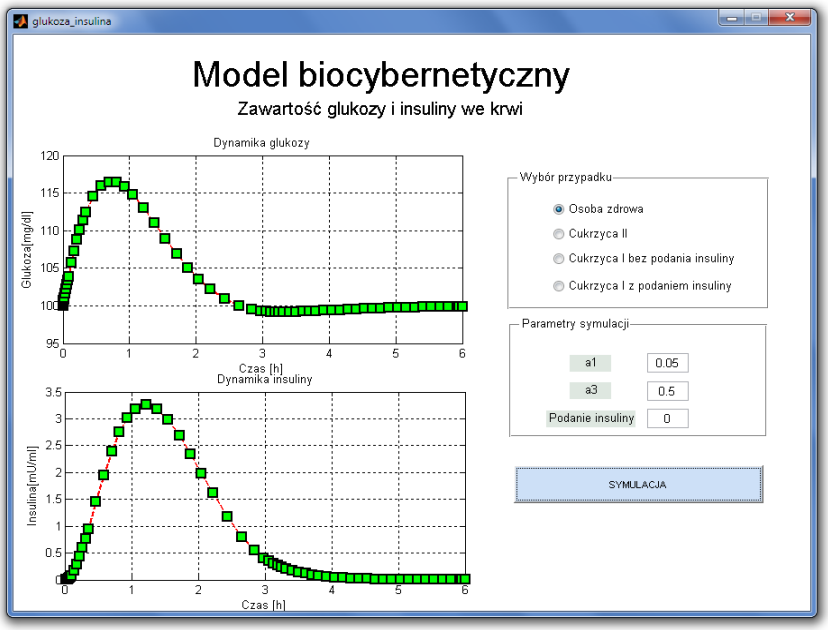
symulacji osoby zdrowej kod wykresu wygląda następująco (Przykład 4.15).

Przykład 4.15 – Wyświetlanie wyników modelu:

```
figure(1);
subplot(2,1,1);
plot(t1,G1);
title('Dynamika glukozy');
xlabel('Czas [h]');
ylabel('Glukoza [mg/dl]');
subplot(2,1,2);
plot(t1,I1);
title('Dynamika insuliny');
xlabel('Czas [h]');
ylabel('Insulina [mU/ml]');
```

Podobnie postępujemy w pozostałych przypadkach i całość skryptu zapisujemy w pliku.

Na stronie www.Tadeusiewicz.pl dostępna jest rozszerzona wersja aplikacji umożliwiająca wygodne wprowadzanie parametrów oraz przedstawiająca wykresy krzywych zmian glukozy i insuliny we krwi (Rys. 4.43).



Rys. 4.43. Okno aplikacji: Model biocybernetyczny – glukoza – insulina

Interfejs użytkownika został podzielony na 3 obszary. Pierwszy obszar zatytułowany **Wybór przypadku** umożliwia wygodne wybieranie poszczególnych sytuacji: osoba zdrowa, cukrzyca typu I, cukrzyca typu II bez

podania insuliny, cukrzyca typu II z podaniem insuliny. Obszar znajdujący się poniżej, zatytułowany **Parametry symulacji,** pozwala na ustawienie poszczególnych parametrów. Po lewej stronie umieszczono wyniki symulacji w postaci wykresów zmian stężenia glukozy i insuliny we krwi.

4.8. Metody modelowania w epidemiologii

4.8.1. Epidemia, pandemia, epidemiologia

Geneza słowa epidemia pochodzi od greckich słów *epi* i *demos*, oznaczających odpowiednio *nawiedzający* oraz *lud*. Epidemią nazwano pojawienie się zachorowań na chorobę zakaźną na określonym obszarze terytorialnym w jednym czasie. W rzeczywistości, pojęcie epidemii jest subiektywne, gdyż klasyfikuje się do niego rzadkie schorzenia o wzmożonym charakterze występowania, natomiast sezonową grypę już nie. Jedną z największych epidemii w dziejach ludzkości była dżuma, nazywana często czarną śmiercią od charakterystycznych ciemnych plam martwiczo-zgorzelinowych na skórze. Według szacunkowych danych, pochłonęła ona ponad 30% ludności Europy, a w niektórych miastach portowych zabiła nawet 80%. Ogniskiem choroby była prawdopodobnie Azja, z której ówcześni kupcy przypłynęli z nią do portów Europy. Epidemia dotarła z dużym opóźnieniem do Polski siejąc niewielkie spustoszenia na Pomorzu, Kujawach oraz Warmii i Mazurach (Rys. 4.44).

Rys. 4.44. Mapa wystąpienia epidemii dżumy w roku 1351 przy obecnych granicach terytorialnych kraju (Opracowanie własne na podstawie: http://pl.wikipedia.org/wiki/Czarna_śmierć Źródło mapy: http://upload.wikimedia.org/wikipedia/commons/f/f3/ Poland_airports_2007.svg, dostęp - listopad 2011)

W przypadku nieznanej wcześniej choroby zakaźnej oraz rozprzestrzenienia się jej na co najmniej trzy kontynenty mamy do czynienia z pandemią. Należy mieć na uwadze, że choroby nowotworowe, które występują na wszystkich kontynentach nie należy zaliczać do pandemii.

Epidemiologia z kolei, bada oddziaływanie środowiska na występowanie określonych chorób w zależności od różnych czynników jakimi są wiek, populacja, czy region świata. Przykładem może być ospa wietrzna, która występuje u młodych ludzi, a także leiszmanioza atakująca w Afryce Wschodniej i Północnej oraz na Bliskim Wschodzie.

4.8.2. Model epidemii SIS

Podstawowy model SIS (ang. *Susceptible Infected* – podatny, zainfekowany) rozprzestrzeniania się epidemii w określonej z góry populacji wyróżnia dwie kategorie osobników[23]. Grupa zdrowa, w której osobnik może zostać zainfekowany (albo zginąć) oraz grupa osobników zainfekowanych. Po przebyciu choroby, jednostka może powrócić do populacji osobników zdrowych albo ponieść śmierć (Rys. 4.45). W modelu tym stosuje się uproszczenie, w którym przyjęto, że liczba osobników umierających w każdej chwili czasowej *t* jest równa liczbie osobników przychodzących na świat.

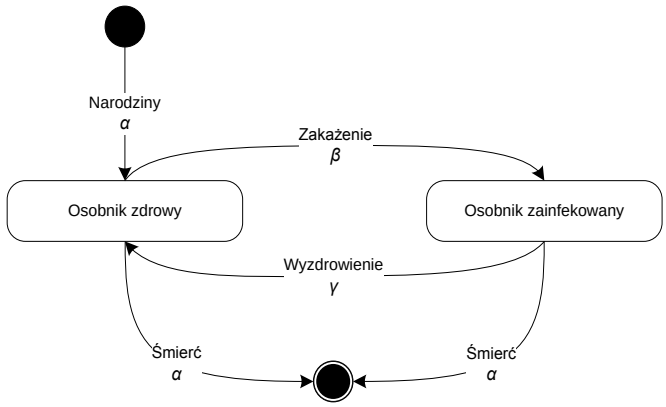
W dalszej części rozdziału oznaczenia *S(t)* oraz *I(t)* wyrażają odpowiednio proporcję osobników zdrowych (podatnych na infekcję) i zainfekowanych, zaś *N* jest liczebnością populacji. W dowolnej chwili czasu *t* spełniona jest zależność:

$$S(t) + I(t) = 1 \tag{4.53}$$

Cała populacja reprezentowana jest natomiast przez:

- *NS(t)* – liczba osobników zdrowych,
- *NI(t)* – liczba osobników zainfekowanych

[23] Wyprowadzenie modelu SIS przytoczone za Foryś U., „*Matematyka w biologii”*, (Wydawnictwa Naukowo-Techniczne, Warszawa, 2005).



Rys. 4.45. Cykl życia pojedynczego osobnika w modelu SIS

Każdy osobnik przychodzący na świat jest zdrowy. Parametrem odpowiedzialnym za narodziny (śmiertelność) jest α. Infekcja pojedynczej jednostki może nastąpić wtedy, gdy będzie miała ona kontakt z osobnikiem chorym. W modelu SIS średnią liczbę kontaktów, których wynikiem jest zakażenie, kontroluje współczynnik zachorowań β. Zatem liczba zakażeń w populacji jest proporcjonalna do iloczynu proporcji obydwu grup i wynosi $\beta NS(t)I(t)$. Ostatnim parametrem modelu jest współczynnik γ odpowiedzialny za kurację jednostek zakażonych.

W oparciu o przedstawiony diagram (Rys. 4.45), można zamodelować, w postaci układu dwóch równań różniczkowych zwyczajnych, zmianę liczebności obydwu klas w czasie:

$$\begin{cases} \dfrac{d(NS(t))}{dt} &= \alpha N - \beta NS(t)I(t) + \gamma NI(t) - \alpha NS(t) \\ \dfrac{d(NI(t))}{dt} &= \beta NS(t)I(t) - \gamma NI(t) - \alpha NI(t) \end{cases} \tag{4.54}$$

Zmiana liczby osobników w klasie zdrowej w czasie t będzie związana z zasileniem klasy pewną stałą liczbą narodzin αN, odejściem osobników do klasy zainfekowanych $-\beta NS(t)I(t)$, przyjściem osobników, które przebyły chorobę $\gamma NI(t)$ oraz pewną liczbą śmierci wyrażoną $-\alpha NS(t)$.

W sposób analogiczny, zmiana liczebności klasy osobników zainfekowanych w czasie t będzie uzależniona od zasilenia grupą osobników, które zachorowały $\beta NS(t)I(t)$, opuszczeniem osobników, które przebyły chorobę $-\gamma NI(t)$ oraz pewną liczbą śmierci wyrażoną $-\alpha NI(t)$.

Podstawiając *S(t) = 1 – I(t)* do pierwszego równania różniczkowego (4.54) następuje redukcja układu do pojedynczego równania różniczkowego zwyczajnego pierwszego rzędu – nosi ono nazwę równania Bernoulliego:

$$\frac{d(I(t))}{dt} = -\beta I^2(t) + (\beta - (\alpha + \gamma))I(t) \tag{4.55}$$

Zakładając $I(t) \neq 0$ oraz dzieląc obustronnie równanie (4.55) przez $-I^2(t)$ otrzymujemy:

$$-\frac{1}{I^2(t)}\frac{d(I(t))}{dt} = \beta - (\beta - (\alpha + \gamma))\frac{1}{I(t)} \tag{4.56}$$

Ostatnim etapem jest podstawienie (4.57), które prowadzi do nowego równania liniowego niejednorodnego (4.58):

$$z(t) = \frac{1}{I(t)}, \qquad \frac{d(z(t))}{dt} = -\frac{1}{I^2(t)}\frac{d(I(t))}{dt} \tag{4.57}$$

$$\frac{d(z(t))}{dt} = \beta - (\beta - (\alpha + \gamma))z(t) \tag{4.58}$$

W celu późniejszej analizy rozwiązania równania różniczkowego, wprowadza się współczynnik δ, który jest stosunkiem współczynnika zakażeń do sumy współczynników śmiertelności i wyzdrowień:

$$\delta = \frac{\beta}{\alpha + \gamma} \tag{4.59}$$

Analityczne rozwiązanie równania różniczkowego (4.58) jest możliwe wówczas przy wykorzystaniu na przykład metody uzmienniania stałej. Ostatecznie funkcją będącą rozwiązaniem równania różniczkowego (4.55), przy założeniu $\delta \neq 1$, jest:

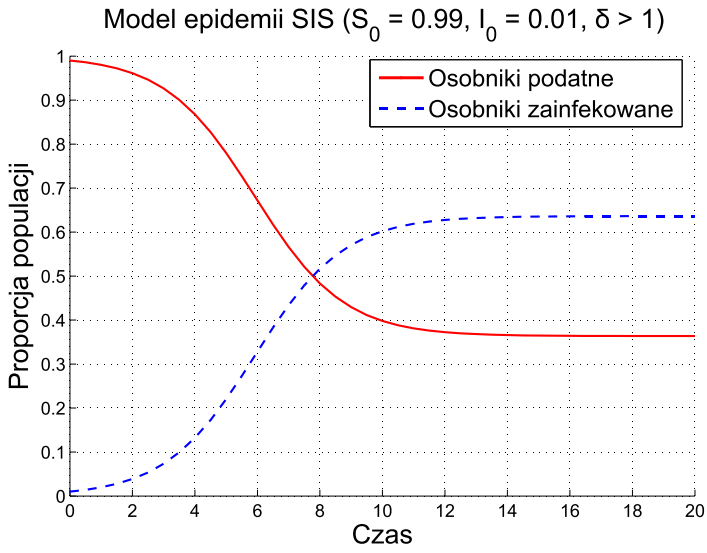
$$I(t) = \frac{I_0 e^{(\alpha+\gamma)(\delta-1)t}}{1 + I_0 \dfrac{\delta}{\delta - 1}(e^{(\alpha+\gamma)(\delta-1)t} - 1)} \tag{4.60}$$

gdzie $I_0 = I(0)$ jest początkową wartością osobników zainfekowanych. Dynamika opisanego systemu jest determinowana przez parametr δ. Dla $\delta > 1$ ($\beta > \alpha + \gamma$ w równaniu (4.58)) grupa osobników zakażonych ustala się na stałym poziomie równym:

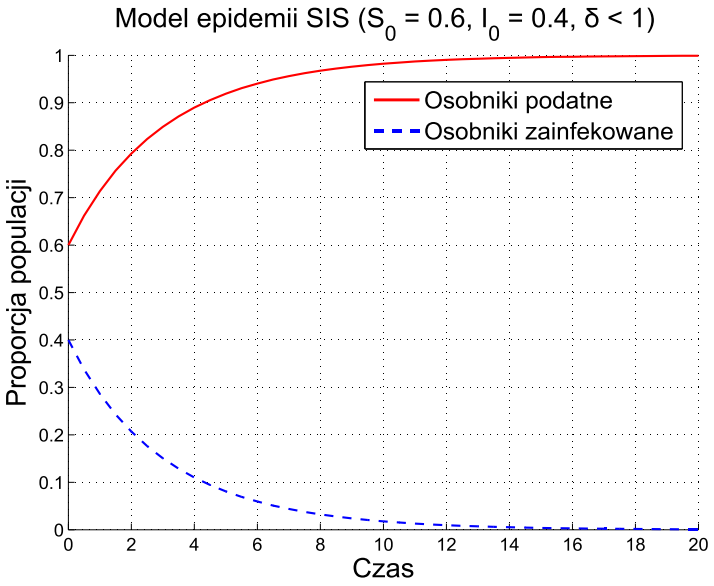
$$\lim_{t \to \infty} I(t) = 1 - \frac{1}{\delta} \tag{4.61}$$

czyli w populacji dochodzi do pandemii (Rys. 4.46). Granicę (4.60) można wyznaczyć dzieląc licznik i mianownik przez wyrażenie $e^{(\alpha+\gamma)(\delta-1)t}$. Operację wykonujemy, ponieważ mamy do czynienia z symbolem nieoznaczonym, granicy wyrażenia (4.60), typu $\left[\frac{\infty}{\infty}\right]$. W przypadku, gdy $\delta < 1$, epidemia zawsze wygasa (Rys. 4.47):

$$\lim_{t\to\infty} I(t) = 0 \tag{4.62}$$



Rys. 4.46. Porównanie przebiegów proporcji osobników podatnych na infekcję i zainfekowanych w modelu SIS dla $\delta > 1$



Rys. 4.47. Porównanie przebiegów proporcji osobników podatnych na infekcję i zainfekowanych w modelu SIS dla $\delta < 1$

Analityczne rozwiązanie przedstawionego problemu i jego analiza jest zadaniem czasochłonnym i wymaga skrupulatnych rachunków obliczeniowych. W dalszej części rozdziału przedstawione zostaną tylko numeryczne wyniki rozwiązań równań różniczkowych w zależności od doboru początkowej proporcji klas oraz parametrów modeli.

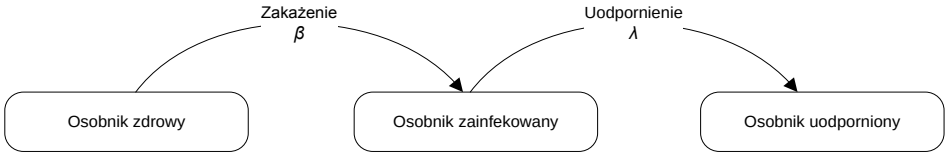
4.8.3. Model epidemii SIR

Model SIS nie uwzględnia możliwości uodpornienia się osobnika na przebytą chorobę. Zjawisko uodpornienia się jednostki zostało uwzględnione w modelu Kermacka i McKendricka[24] [25] [26]. Model taki nosi nazwę SIR (ang. *Susceptible Infected Resistant* – podatny, zainfekowany, odporny). Model ten jednak, w przeciwieństwie do modelu SIS, nie uwzględnia rozrodczości i śmiertelności osobników (Rys. 4.48).

[24] Współpracowali oni z Ronaldem Rossem (1857 - 1932) – laureatem nagrody Nobla w roku 1902 w dziedzinie fizjologii lub medycyny za odkrycie cyklu rozwoju malarii.

[25] Model SIR przytoczony za Foryś U., „*Matematyka w biologii”*, (Wydawnictwa Naukowo-Techniczne, Warszawa, 2005).

[26] Model został zaproponowany w pracy: Kermack W. O., McKendrick A.G., “*A contribution to the Mathematical Theory of Epidemics*”, Proceedings of the Royal Society of London, Vol. 115, Issue 772, pp. 700-721, 1927



Rys. 4.48. Cykl życia pojedynczego osobnika w modelu SIR

Postać równań różniczkowych opisujących model SIR przedstawia się następująco:

$$\begin{cases} \dfrac{d(NS(t))}{dt} = -\beta NS(t)I(t) \\ \dfrac{d(NI(t))}{dt} = \beta NS(t)I(t) - \lambda NI(t) \\ \dfrac{d(NR(t))}{dt} = \lambda NI(t) \end{cases} \tag{4.63}$$

Oznaczenia *S(t), I(t), β* są analogiczne jak w przedstawionym modelu SIS. Parametr λ związany jest z uodpornieniem osobników na infekcję. Trzecie równania wiążące się ze zmianą *R(t)* oznacza proporcję osobników uodpornionych (zdrowych), którzy przebyli infekcję. Proporcje każdej klasy osobników sumują się do jedności:

$$S(t) + I(t) + R(t) = 1 \tag{4.64}$$

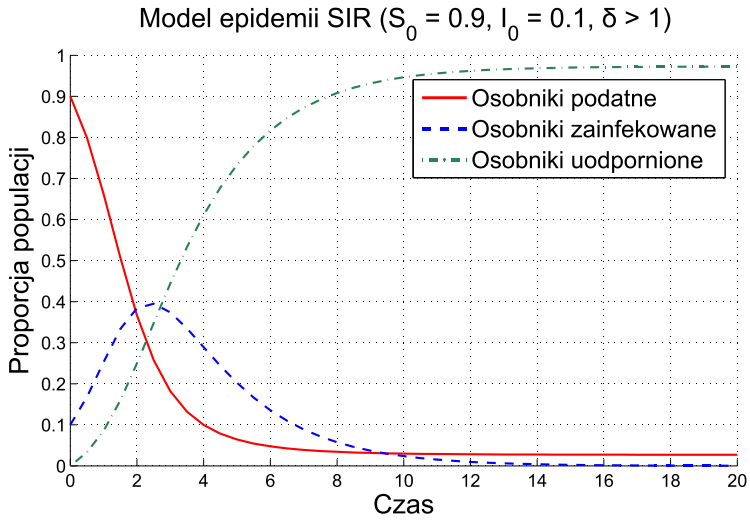
Korzystając z założeń (4.64), układ równań (4.63) po podzieleniu pierwszego i drugiego równania przez *N* może zostać zapisany w postaci:

$$\begin{cases} \dfrac{d(S(t))}{dt} = -\beta S(t)I(t) \\ \dfrac{d(I(t))}{dt} = \beta S(t)I(t) - \lambda I(t) \end{cases} \tag{4.65}$$

wraz z warunkami początkowymi $S(0) = S_0 > 0$, $I(0) = I_0 > 0$, $R(0) = 0$, $S_0 + I_0 \in [0,1]$. Dynamika modelu SIR zdeterminowana jest przez współczynnik δ, który jest stosunkiem współczynnika zakażeń do współczynnika uodpornienia (wyzdrowień):

$$\delta = \frac{\beta}{\lambda} \tag{4.66}$$

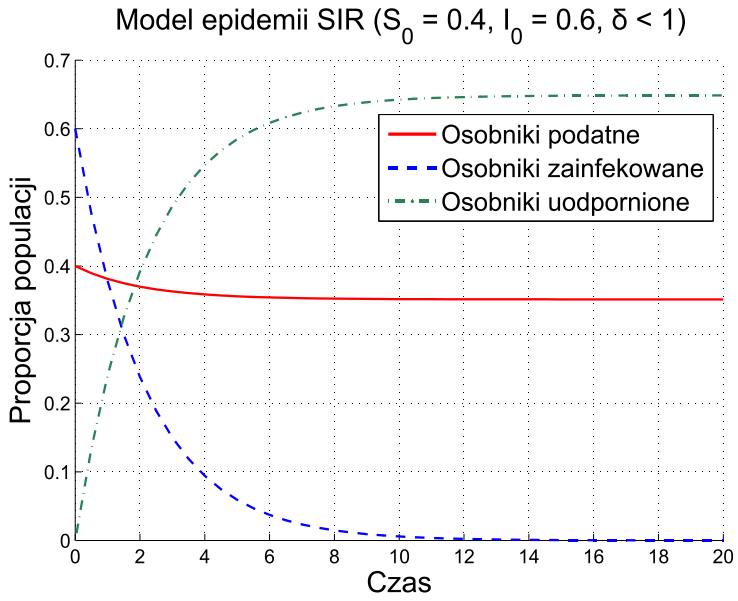
Dla $\delta > 1$ zmiana liczebności grupy zainfekowanej zależna jest od warunku początkowego S_0. Jeżeli wartość ta jest wystarczająco duża, epidemia osiąga fazę krytyczną, po czym liczba zakażonych osobników maleje do zera, a więc epidemia wygasa (Rys. 4.49).



Rys. 4.49. Porównanie przebiegów proporcji osobników podatnych na infekcję, zainfekowanych i uodpornionych w modelu SIR dla $\delta > 1$

W przypadku, gdy $\delta < 1$ epidemia również wygasa. Liczebność grupy zakażonych maleje do zera (Rys. 4.50). Grupa osobników podatnych na infekcję w obydwu przypadkach maleje do wartości x, która jest rozwiązaniem równania:

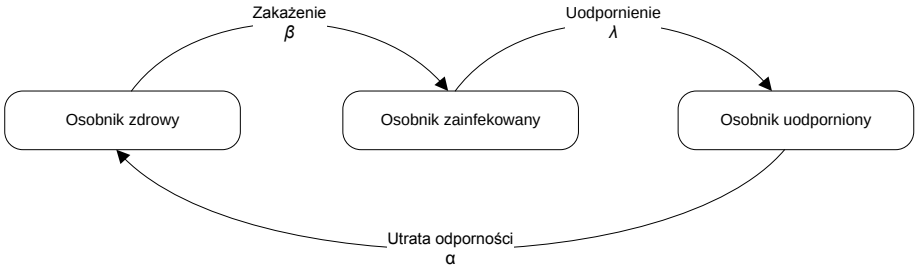
$$1 - R_0 - x - \frac{\lambda}{\beta}\ln\left(\frac{x}{S_0}\right) = 0 \tag{4.67}$$



Rys. 4.50. Porównanie przebiegów proporcji osobników podatnych na infekcję, zainfekowanych i uodpornionych w modelu SIR dla $\delta < 1$

4.8.4. Model epidemii SIRS

Model SIR nie uwzględnia możliwości ponownej infekcji osobników po przebytej chorobie. Z własnością tą mamy do czynienia w modelu SIRS (ang. *Susceptible Infected Resistant Susceptible* – podatny, zainfekowany, odporny, podatny) (Rys. 4.51).



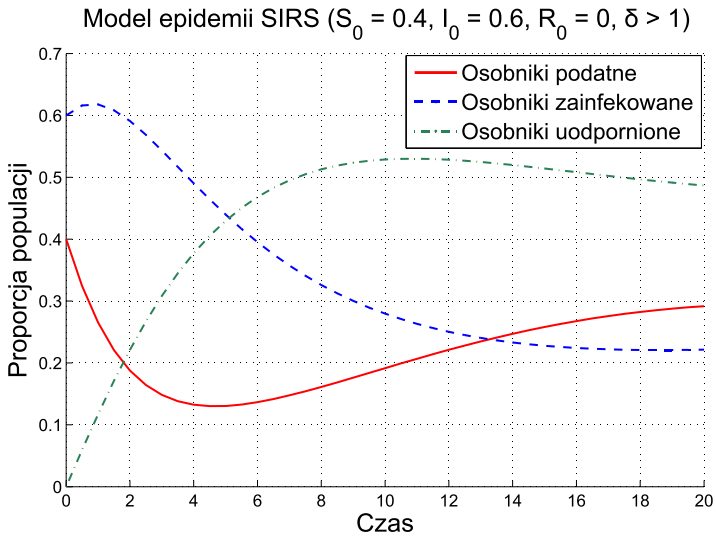
Rys. 4.51. Cykl życia pojedynczego osobnika w modelu SIRS

Układ równań różniczkowych opisujących model SIRS jest zbliżony do modelu SIR:

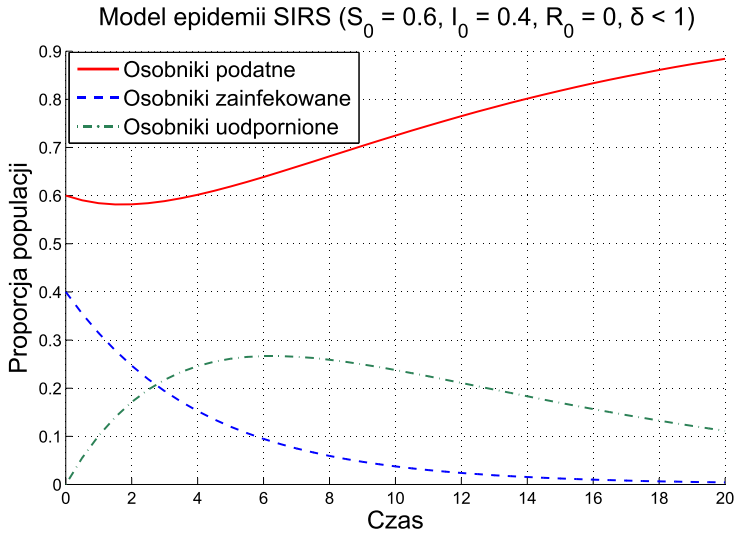
$$\begin{cases} \dfrac{d(NS(t))}{dt} &= -\beta NS(t)I(t) + \alpha NR(t) \\ \dfrac{d(NI(t))}{dt} &= \beta NS(t)I(t) - \lambda NI(t) \\ \dfrac{d(NR(t))}{dt} &= \lambda NI(t) - \alpha NR(t) \end{cases} \tag{4.68}$$

Utratą odporności na infekcję kontroluje współczynnik α. Pierwsze i trzecie równanie zostało uzupełnione o człon $\alpha NR(t)$ odpowiedzialny za przejście grupy osobników z klasy uodpornionej do klasy zdrowej (podatnej).

Gdy $\alpha = 0$, mamy do czynienia z modelem Kermacka-McKendricka. Model SIRS wykazuje nieco bardziej złożone zachowania niż poprzednie przypadki. Tym razem dynamikę determinują trzy współczynniki (α, β, λ), jednak rozwój epidemii zależy od wartości parametru $\delta = \dfrac{\beta}{\lambda}$. Gdy $\delta > 1$, liczebności każdej klasy zmierzają do stałej, niezerowej wartości, a więc epidemia nie wygasa (Rys. 4.52). W przypadku gdy $\delta < 1$, epidemia wygasa, jednak wszystkie osobniki pozostają w rezultacie podatne (Rys. 4.53).



Rys. 4.52. Porównanie przebiegów proporcji osobników podatnych na infekcję, zainfekowanych i uodpornionych w modelu SIRS dla $\delta > 1$



Rys. 4.53. Porównanie przebiegów proporcji osobników podatnych na infekcję, zainfekowanych i uodpornionych w modelu SIRS dla $\delta < 1$

4.8.5. Komputerowa symulacja modeli epidemiologicznych

Komputerowa symulacja nie pozwala zaprezentować zmian populacji w dowolnej chwili czasu *t*. Dlatego też, w przypadku symulacji przedstawionych równań różniczkowych możliwe jest ich rozwiązanie i przedstawienie tylko w dyskretnych chwilach czasu. Numeryczne symulacje poszczególnych modeli zawarte są odpowiednio w skryptach:

- `epidemia_sis.m` oraz `model_matematyczny_sis.m`
- `epidemia_sir.m` oraz `model_matematyczny_sir.m`
- `epidemia_sirs.m` oraz `model_matematyczny_sirs.m`

Przeglądając polecenia zawarte w skryptach `epidemia_*.m` warto zwrócić uwagę na fragmenty odpowiedzialne za numeryczne rozwiązanie równania różniczkowego z zadanym krokiem początkowym *h* (w tym przypadku wywołanie funkcji `ode45()`) oraz wyznaczenie proporcji osobników należących do poszczególnych klas.

Skrypty `model_matematyczny_*.m` zawierają parametry symulacji (współczynniki: śmiertelności, zakażenia i wyzdrowienia) oraz równania różniczkowe poszczególnych modeli.

Przykład 4.16 – Model epidemii SIS

```
% Dane wejściowe modelu:
t0 = 0;  % czas początkowy symulacji
tk = 10; % czas końcowy symulacji
h = 0.1; % krok różniczkowania
I0 = 0.7;% proporcja osobników zainfekowanych w t0

% Rozwiązanie równania różniczkowego metodą przybliżoną
% Dormand-Prince
[wektor_czasu, osobniki_zainfekowane] = ...
   ode45(@model_matematyczny_sis, [t0, tk], I0, h);

% Wyznaczenie liczby osobników podatnych
osobniki_podatne = 1 - osobniki_zainfekowane;

% Przebiegi liczby osobników zainfekowanych i podatnych
hold on;
plot(wektor_czasu, osobniki_podatne, 'r-');
plot(wektor_czasu, osobniki_zainfekowane, 'b--');

% Ustawienie legendy, tytułu, opisów osi oraz
% uwidocznienie siatki
legend('Osobniki podatne', 'Osobniki zainfekowane');
title(['Model epidemii SIS (S_0 = 0.99,'...
'I_0 = 0.01, \delta > 1)']);
xlabel('Czas'); ylabel('Proporcja populacji'); grid on
```

```
% W osobnym pliku model_matematyczny_sis.m zapisujemy
% poniższą funkcję

% Równanie modelu SIS - osobniki zainfekowane
function [I] = model_matematyczny_sis(t, arg)
alfa = 0.3;        % współczynnik śmiertelności
beta = 0.2;        % współczynnik zakażeń
gamma = 0.3;       % współczynnik wyzdrowień

% Równanie różniczkowe (4.55)
I = -beta*arg^2 + (beta - (alfa + gamma))*arg;
```

Przykład 4.17 – Model epidemii SIR

```
% Dane wejściowe modelu:
t0 = 0;   % czas początkowy symulacji
tk = 10;  % czas końcowy symulacji
h = 0.5;  % krok różniczkowania (początkowy)
S0 = 0.4; % proporcja osobników podatnych w t0
I0 = 0.6; % proporcja osobników zainfekowanych w t0

% Rozwiązanie równania różniczkowego metodą przybliżoną
% Dormand-Prince
[wektor_czasu, rezultat] = ...
ode45(@model_matematyczny_sir, [t0, tk], [S0, I0], h);

% wyznaczenie proporcji osobników podatnych,
% zainfekowanych i uodpornionych
osobniki_podatne = rezultat(:, 1);
osobniki_zainfekowane = rezultat(:, 2);
osobniki_uodpornione = 1-rezultat(:, 1)-rezultat(:, 1);

% przebiegi proporcji osobników podatnych,
% zainfekowanych  i uodpornionych w zależności od
% czasu
hold on;
plot(wektor_czasu, osobniki_podatne, 'r-');
plot(wektor_czasu, osobniki_zainfekowane, 'b-');
plot(wektor_czasu, osobniki_uodpornione, 'g-');

% ustawienie legendy, tytułu, opisów osi oraz
% uwidocznienie siatki
legend('Osobniki podatne', ...
   'Osobniki zainfekowane', 'Osobniki uodpornione');
title(['Model epidemii SIR (S_0 = 0.4,'...
' I_0 = 0.6, \delta < 1)']);
xlabel('Czas'); ylabel('Proporcja populacji'); grid on
```

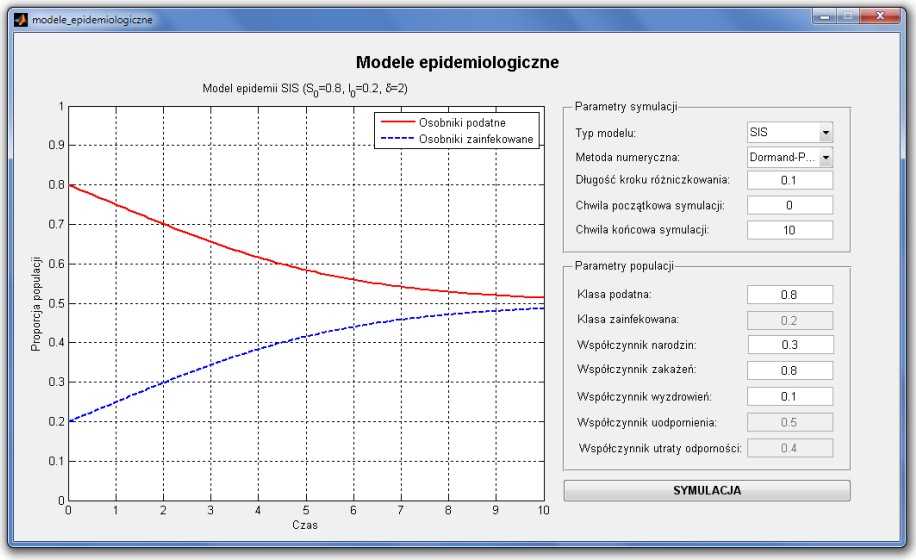
```
% W osobnym pliku model_matematyczny_sir.m zapisujemy
% poniższą funkcję

% Równania modelu SIR - osobniki podatne i zainfekowane
function [rezultat] = model_matematyczny_sir(t, arg)
beta = 0.1;     % współczynnik zakażeń
lambda = 0.5;  % współczynnik (uodpornień) wyzdrowień

% układ równań różniczkowych (4.65)
rezultat(1) = -beta*arg(1)*arg(2);
rezultat(2) = beta*arg(1)*arg(2) - lambda*arg(2);
rezultat = rezultat';
```

Przykład 4.18 – Model epidemii SIRS

```
% Dane wejściowe modelu:
t0 = 0;   % czas początkowy symulacji
tk = 20;  % czas końcowy symulacji
h = 0.5;  % krok różniczkowania (początkowy)
S0 = 0.6; % proporcja osobników podatnych w t0
I0 = 0.4; % proporcja osobników zainfekowanych w t0
R0 = 0.0; % proporcja osobników uodpornionych w t0

% Rozwiązanie równania różniczkowego metodą przybliżoną
% Dormand-Prince
[wektor_czasu, rezultat] = ode45(...
 @model_matematyczny_sirs, [t0, tk], [S0, I0, R0], h);

% wyznaczenie proporcji osobników podatnych,
% zainfekowanych i uodpornionych
osobniki_podatne = rezultat(:, 1);
osobniki_zainfekowane = rezultat(:, 2);
osobniki_uodpornione =  rezultat(:, 3);

% przebiegi proporcji osobników podatnych,
% zainfekowanych i uodpornionych w zależności od czasu
hold on;
plot(wektor_czasu, osobniki_podatne, 'r-');
plot(wektor_czasu, osobniki_zainfekowane, 'b-');
plot(wektor_czasu, osobniki_uodpornione, 'g-');

% ustawienie legendy, tytułu, opisów osi oraz
% uwidocznienie siatki
legend('Osobniki podatne', ...
    'Osobniki zainfekowane', 'Osobniki uodpornione');
title(['Model epidemii SIRS (S_0 = 0.6,'...
   'I_0 = 0.4, R_0 = 0, \delta < 1)']);
xlabel('Czas');
ylabel('Proporcja populacji');
grid on
```

```
% W osobnym pliku model_matematyczny_sirs.m zapisujemy
% poniższą funkcję

function [rezultat] = model_matematyczny_sirs(t, arg)
beta = 0.1;       % współczynnik zakażeń
lambda = 0.3;     % współczynnik (uodpornień) wyzdrowień
alfa = 0.1;       % współczynnik utraty odporności

rezultat(1) = -beta*arg(1)*arg(2) + alfa*arg(3);
rezultat(2) = beta*arg(1)*arg(2) - lambda*arg(2);
rezultat(3) = lambda*arg(2) - alfa*arg(3);
rezultat = rezultat';
```

Zainteresowany czytelnik, oprócz wykonania własnych doświadczeń w oparciu o M-pliki może wykorzystać specjalnie przygotowany do tego program z graficznym interfejsem użytkownika `modele_epidemiologiczne.m` dostępny na stronie www.Tadeusiewicz.pl. Pozwala on na przeprowadzanie symulacji wybranego modelu o zadanych parametrach symulacji oraz populacji, które należy wybrać oraz wpisać w odpowiednich polach edycji (Rys. 4.54).



Rys. 4.54. Gotowe oprogramowanie pozwalające dokonywać własnych symulacji rozrostu epidemii przy wybranych parametrach symulacji

BIBLIOGRAFIA

[1] L. Edelstein-Keshet, *Mathematical Models in Biology*, SIAM, Philadelphia 2005.

[2] R. Tadeusiewicz, *Problemy biocybernetyki*, PWN, Warszawa 1993.

[3] W. Marczewski, R. Tadeusiewicz, *Antropomotoryka biocybernetyczna*, Wydawnictwo AWF, Kraków, 1993.

[4] J.M. Smith, *Matematyka w biologii*, Wiedza Powszechna, Warszawa, 1974.

[5] W. Sawyer, *W poszukiwaniu modelu matematycznego*, Wiedza Powszechna, Warszawa, 1975.

[6] J. Jankowska, M. Jankowski, *Przegląd metod i algorytmów numerycznych. Część 1*, Wydawnictwa-Naukowo Techniczne, Warszawa, 1981.

[7] D.G. Zill, M.R. Cullen, *Differential Equations with Boundary-Value Problems*, Brooks Cole, 2008.

[8] P. Blanchard, R.L. Devaney, G.R. Hall, *Differential Equations*, Brooks Cole, 2002.

[9] J. Povstenko, *Wprowadzenie do metod numerycznych*, Akademicka Oficyna Wydawnicza EXIT, Warszawa, 2005.

[10] Z. Fortuna, B. Macukow, J. Wąsowski, *Metody numeryczne*, Wydawnictwo Naukowo-Techniczne, Warszawa, 2005.

[11] B. Bożek, *Metody obliczeniowe i ich komputerowa realizacja*, Uczelniane Wydawnictwa Naukowo-Dydaktyczne, Kraków, 2005.

[12] D. Kincaid, W. Cheney, *Analiza numeryczna*, Wydawnictwo Naukowo-Techniczne, Warszawa, 2006.

[13] A.K. Kaw, E.E. Kalu, D. Nguyen, *Numerical Methods with Applications: Abridged*,2008, http://numericalmethods.eng.usf.edu/topics/textbook_index.html

[14] S. Osowski, *Modelowanie układów*, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 1997.

[15] S. Osowski, *Modelowanie i symulacja układów i procesów dynamicznych*, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 2007

[16] J. Rewera, *System kompleksowego wspomagania terapii cukrzycy i chorób metabolicznych*, rozprawa doktorska, promotor: Ryszard Tadeusiewicz, Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie, Kraków, 2010.

[17] J. Koleszyńska, *Modelowanie... zjadania cukierków*. Rozdział w książce: Inżynieria biomedyczna : księga współczesnej wiedzy tajemnej w wersji przystępnej i przyjemnej. (red.):Ryszard Tadeusiewicz. AGH Uczelniane Wydawnictwa Naukowo–Dydaktyczne, Kraków,2008.

SKOROWIDZ WAŻNIEJSZYCH OZNACZEŃ I POJĘĆ UŻYWANYCH W KSIĄŻCE

D.1. Definicje wybranych terminów związanych z modelowaniem

Błąd modelu – dowolny błąd polegający na tym, że wyniki uzyskiwane z modelu są znacząco inne, niż wyniki obserwowane w rzeczywistym obiekcie.

Błąd modelu konceptualnego – błąd występujący w przypadku niewłaściwego sformułowania koncepcji modelu reprezentującego analizowany obiekt, proces albo zagadnienie.

Błąd modelu matematycznego– błąd występujący w przypadku niewłaściwego doboru lub nieprawidłowego zastosowania formuł matematycznych odwzorowujących w sposób formalny elementy modelu konceptualnego.

Błąd modelu symulacyjnego– błąd występujący w przypadku niewłaściwego zaprogramowania na komputerze prawidłowego modelu matematycznego wywiedzionego z poprawnego konceptualnego.

Dyskretyzacja– proces przekształcania modelu matematycznego ciągłego wyrażonego przez równania różniczkowe zwyczajne (lub cząstkowe) do modelu matematycznego dyskretnego definiowanego przez zespół dyskretnych struktur danych oraz równania różnicowe.

Identyfikacja bierna – sposób identyfikacji modelu polegający wyłącznie na biernej obserwacji obiektu. Stosuje się ją, gdy niemożliwa jest identyfikacja czynna. Identyfikacja bierna nie zakłóca normalnego funkcjonowania obiektu, więc nie wiąże się z kosztami, ale dane zebrane podczas takiej identyfikacji w niewielkim stopniu odpowiadają potrzebom modelowania i wymagają starannej analizy i obróbki – zwykle głównie statystycznej, co jednak może spowodować pojawienie się kosztów z tym związanych i także składających się na cały koszt procesu modelowania.

Identyfikacja czynna –sposób identyfikacji modelu polegający na wykonywaniu na nim celowych eksperymentów zmierzających do wykrycia i pomierzenia wybranych cech modelowanego obiektu. Identyfikacja czynna przynosi najwięcej wartościowych informacji na temat modelowanego obiektu, ale wymaga głębokiego ingerowania w normalne funkcjonowanie obiektu i z tego powodu często jest niemożliwa do przeprowadzenia. Niemożliwość stosowania identyfikacji czynnej może wynikać ze względów pragmatycznych (nie każdy eksperyment da się przeprowadzić w praktyce ze względu na ograniczone możliwości sterowania rozważanego obiektu) ze względów ekonomicznych (identyfikacja czynna rodzi koszty bezpośrednie – eksperyment kosztuje – oraz pośrednie – obiekt poddany eksperymentom nie pracuje normalnie, co

może rodzić koszty na zasadzie utraty oczekiwanych zysków), a także ze względów moralnych (identyfikacja czynna cech obiektów biologicznych może wymagać wykonywania okrutnych doświadczeń na zwierzętach lub całkowicie nie akceptowalnych eksperymentów na ludziach).

Identyfikacja modelu – zbiór czynności wykonywanych na modelowanym obiekcie w tym celu, aby określić jego mierzalne cechy, będące potem podstawą ustalenia parametrów modelu.

Kalibracja– proces polegający na dopasowaniu modelu do procesu rzeczywistego poprzez porównanie wyników symulacji z danymi doświadczalnymi (uzyskanymi w trakcie identyfikacji) bądź danymi teoretycznymi(zwykle literaturowymi). Zwykle w trakcie kalibracji parametry modelu otrzymują konkretne wartości, które dobiera się w taki sposób, by odpowiednio dobrane kryteria jakości modelu osiągały najkorzystniejsze wartości.

Kod programowy– zestaw instrukcji w języku maszynowym, lub języku wysokiego poziomu (na przykład w środowisku Matlab), zawierających implementację struktur danych i równań algebraicznych w postaci nadającej się do prowadzenia obliczeń komputerowych (w postaci numerycznej).

Konceptualny model – patrz **Model konceptualny**

Konstrukcja modelu– proces przekształcania modelu konceptualnego, poprzez model matematyczny ciągły, dyskretny do modelu symulacyjnego (numerycznego wyrażonego przy pomocy kodu programowego jednego z języków programowania).

Kryteria jakości modelu – ilościowe lub opisowe cechy modelu, które twórca modelu stara się zmaksymalizować, jeśli wyrażają pozytywne cechy modelu (na przykład stopień wierności modelu w stosunku do oryginału) względnie stara się zminimalizować jeśli stanowią miarę niedoskonałości modelu (na przykład błąd).

Metoda elementów skończonych (ang. *Finite Element Method* -FEM) – metoda dyskretyzacji przestrzennej lub przestrzenno-czasowej obiektu modelowania. Wynikiem dyskretyzacji jest układ równań algebraicznych rozwiązywanych przy pomocy metod macierzowych. Jest to jedna z najczęściej stosowanych metod modelowania obiektów w których istotny jest aspekt przestrzenny (w przypadku systemów biologicznych dotyczy to głównie biomechaniki).

Metoda odwrotna (ang. *Inverse Method*) – metoda kalibracji parametrów

modelu numerycznego procesu opartej na procesie minimalizacji funkcji celu zdefiniowanej jako MSE (RMSE).

Metoda różnic skończonych (ang. *FiniteDifference Method* – FDM) – metoda dyskretyzacji modeli ciągłych.

Model– zbiór koncepcji sformułowanych przy pomocy opisu teoretycznego, równań matematycznych lub kodu programowego.

Model konceptualny– model obejmujący ogólne założenia i definicje analizowanego procesu. Stanowi podstawę do opracowania modelu matematycznego bądź numerycznego.

Model matematyczny– zbiór równań i nierówności algebraicznych opisujących modelowany obiekt w obszarze i zakresie wyznaczonym przez model konceptualny.

Model różnic skończonych (ang. *FiniteDifference Model*) – rodzaj modelu matematycznego obejmującego definicję obiektu z wykorzystaniem równań różnicowych.

Model symulacyjny – program odwzorowujący w komputerze model matematyczny obiektu. Model symulacyjny umożliwia zwykle przeprowadzanie eksperymentów numerycznych zastępujących w wielu zastosowaniach badanie rzeczywistego obiektu.

Modelowanie– proces formułowania definicji modelu systemu lub procesu. Modelowanie obejmuje zwykle trzy następujące po sobie procesy: tworzenie modelu konceptualnego, dobieranie modelu matematycznego i konstruowanie modelu symulacyjnego.

Obiekt modelowania –rzeczywisty system, proces lub zjawisko do którego chcemy zastosować proces modelowania.

Otoczenie –wszystkie systemy, procesy lub zjawiska, które świadomie i celowo pozostawiamy poza obrębem modelowanego systemu. Jeśli otoczenie oddziałuje na obiekt modelowania to oddziaływania te najczęściej klasyfikujemy jako zakłócenia.

Parametry modelu– zespół umieszczonych w modelu stałych o odpowiednio dobieranych wartościach. Stałe te najczęściej reprezentują w modelu wartości wielkości fizycznych dających się zmierzyć w rzeczywistym obiekcie lub wartości współczynników matematycznych wprowadzonych arbitralnie do struktury modelu, których wartości ustalane są w trakcie

kalibracji modelu. Ustalenie wartości parametrów modelu zamienia model jednoznacznie identyfikujących proces rzeczywisty.

Postprocessing– analiza wyników symulacji komputerowej – zwykle sygnałów wyjściowych modelu.

Preprocessing– proces przygotowania danych do obliczeń symulacyjnych (na przykład normalizacja danych). Dotyczy zwykle sygnałów wejściowych modelu.

Scenariusz wykorzystania modelu – przewidywany na etapie tworzenia modelu konceptualnego sposób zastosowania modelu oraz wyników symulacji komputerowej. Zazwyczaj przewiduje się wykorzystanie modelu jako źródła dodatkowej inspiracji naukowej (*experiment in computo*), bierze się pod uwagę możliwości praktycznego wykorzystania samego modelu lub wyników uzyskanych w wyniku jego badań symulacyjnych, a ponadto często scenariusz przewiduje, że model może być wykorzystany jako narzędzie dydaktyczne.

Selekcja kodu programowego– proces analizy środowisk programistycznych pozwalających wybrać rozwiązanie najodpowiedniejsze do symulowanych zagadnień.

Struktura modelu – odpowiednio dobrana konstrukcja matematyczna wyrażająca w kategoriach formalnych wiedzę badacza na temat tego, czym jest modelowany obiekt i jakie prawa nim rządzą.

Wejście modelu– zbiór sygnałów wejście modelu, stanowiących jednocześnie zespół danych do obliczeń symulacyjnych.

Weryfikacja modelu– proces oceny zgodności kodu programowego modelu symulacyjnego z bazowym modelem matematycznym.

Wyjście modelu – sygnał lub zbiór sygnałów określających wynik obliczeń przeprowadzanych na modelu matematycznym lub uzyskiwanych w wyniku symulacji komputerowych.

Zakłócenia – sygnały docierające do rozważanego obiektu modelowania które wpływają na jego zachowanie (w szczególności na sygnały wyjściowe), ale nie podlegają bezpośredniej kontroli ze strony czynników wpływających na zachowanie obiektu (w szczególności generujących sygnały wejściowe).

Założenia – przyjmowane na początku procesu tworzenia modelu ustalenia

biorące pod uwagę właściwości modelowanego obiektu oraz cel i zakres modelowania, a także przewidywane scenariusze wykorzystania modelu.

D.2. Skorowidz słów kluczowych Matlaba

„Cell mode" – tryb tzw. celek pozwalający na blokowe uruchamianie kodu m-skryptu. Tworzy się go poprzez stosowanie podwójnego znaku komentarza `%%`.

„Commandhistory" – zakładka interfejsu środowiska MATLAB w której wyświetlane są zapamiętane komendy wydawane w linii poleceń.

„Commandwindow" – okno poleceń środowiska MATLAB, w którym wpisuje się komendy i gdzie wyświetlane są rezultaty.

„Current folder" – zakładka interfejsu środowiska MATLAB, udostępniająca przeglądarkę plików oraz dostęp do dodatkowych narzędzi (np. importu danych).

„Import wizard" – narzędzie importu danych udostępniające wygodny graficzny interfejs użytkownika (GUI).

Indeksowanie typu wiersz kolumna – sposób dostępu do elementów tablicy polegający na podaniu numerów wierszy i kolumn.

Indeksowanie liniowe – sposób dostępu do elementów tablicy polegający na podaniu numeru elementu w tablicy.

Indeksowanie logiczne – sposób dostępu do elementów tablicy pozwalający na wybór z tablicy elementów spełniających określony warunek logiczny.

Komentarze – sposób wpisywania dodatkowych informacji w M-plikach, polegający na umieszczeniu znaku `%` na początku linii. Komentarze nie są uruchamiane tak jak pozostałe komendy MATLABa, ale pełnią rolę informacyjną (np. dodatkowy opis i wyjaśnienia).

M-pliki – są to pliki tekstowe z rozszerzeniem `*.m`, zawierające zapisane polecenia MATLABa, pozwalające odtworzyć wykonywane wcześniej obliczenia.

M-skrypt – zapamiętane komendy MATLABa wraz z kolejnością ich

wykonania, zapisane w M-pliku tekstowym.

M-funkcja – podobnie jak m-skrypty, są to pliki tekstowe zawierające polecenia MATLABa. To co odróżnia m-funkcje od m-skryptów to składnia oraz posiadanie osobnej przestrzeni roboczej dla zmiennych.

Operacje tablicowe – obliczenia wykonywane przez MATLABa niezależnie dla każdego elementu tablicy. Cechą charakterystyczną środowiska jest brak konieczności stosowania instrukcji pętli programowych tak jak np. w języku C. W przypadku obliczeń matematycznych (np. mnożenie, dzielenie) stosuje się znak kropki przez operatorem - przykładowo `.*`

Operacje macierzowe – obliczenia wykonywane przez środowisko na tablicach z uwzględnieniem zasad rachunku macierzowego (tablica traktowana jako macierz dwuwymiarowa).

Operacje matematyczne – są to obliczenia wykonywane przez MATLABa niezależnie dla każdego z elementów tablicy. Rezultatem jest tablica o takim samym rozmiarze jak tablica wejściowa.

Operacje statystyczne – są to obliczenia wykonywane przez MATLABa na elementach tablicy, przy czym wiersze tablicy traktowane są jako obserwacje natomiast kolumny jako elementy wielowymiarowej zmiennej losowej.

„Plot tools" – narzędzia do tworzenia i interaktywnej edycji wykresów.

Przestrzeń robocza – zaalokowana przez środowisko MATLAB pamięć operacyjna na zmienne i dane. Wyróżniamy główną przestrzeń roboczą oraz lokalne przestrzenie robocze funkcji.

Publikowanie rezultatów – narzędzie do automatycznego tworzenia raportów z rezultatów wykonania m-skryptu.

Struktury – typ danych środowiska posiadający ustaloną wewnętrzną reprezentację, składającą się z rekordów oraz pól.

Tablica numeryczna – podstawowa jednostka danych w środowisku MATLAB, zazwyczaj o liczbie wymiarów równej 2. Do tworzenia tablic stosuje się nawias prostokątny `[]`. Dane w tablicach muszą być jednorodne, tzn. wszystkie elementy tablicy są tego samego typu.

Tablica komórkowa (ang. cellarrays) – tablice ogólnego przeznaczenia, pozwalające zapamiętać dowolne typy danych. Do tworzenia tablic

stosuje się nawias klamrowy `{}`.

Typ danych – sposób reprezentacji numerycznej zmiennych w środowisku. Domyślny typ danych w środowisku MATLAB to liczby zmiennoprzecinkowe podwójnej precyzji.

Uchwyty do funkcji (ang. functionhandles) – są to referencje do m-funkcji.

„Variableeditor” – edytor tablic pozwalający na podgląd i edycję zmiennych MATLABa.

„Workspacebrowser” – zakładka interfejsu w której wyświetlane są informacje na temat zmiennych dostępnych w aktywnej przestrzeni roboczej MATLABa.

Zmienna– konstrukcja programistyczna posiadająca symboliczną nazwę i reprezentująca określoną wartość (lub tablicę wartości) przechowywaną w pamięci operacyjnej komputera. Poprzez zmienną możliwe jest odwoływanie się do zapamiętanych danych. W środowisku MATLAB zarządzanie pamięcią danych odbywa się automatycznie (ang.garbagecollection). Każda zmienna jest tablicą (pojedyncza liczba to tablica o rozmiarze 1x1).