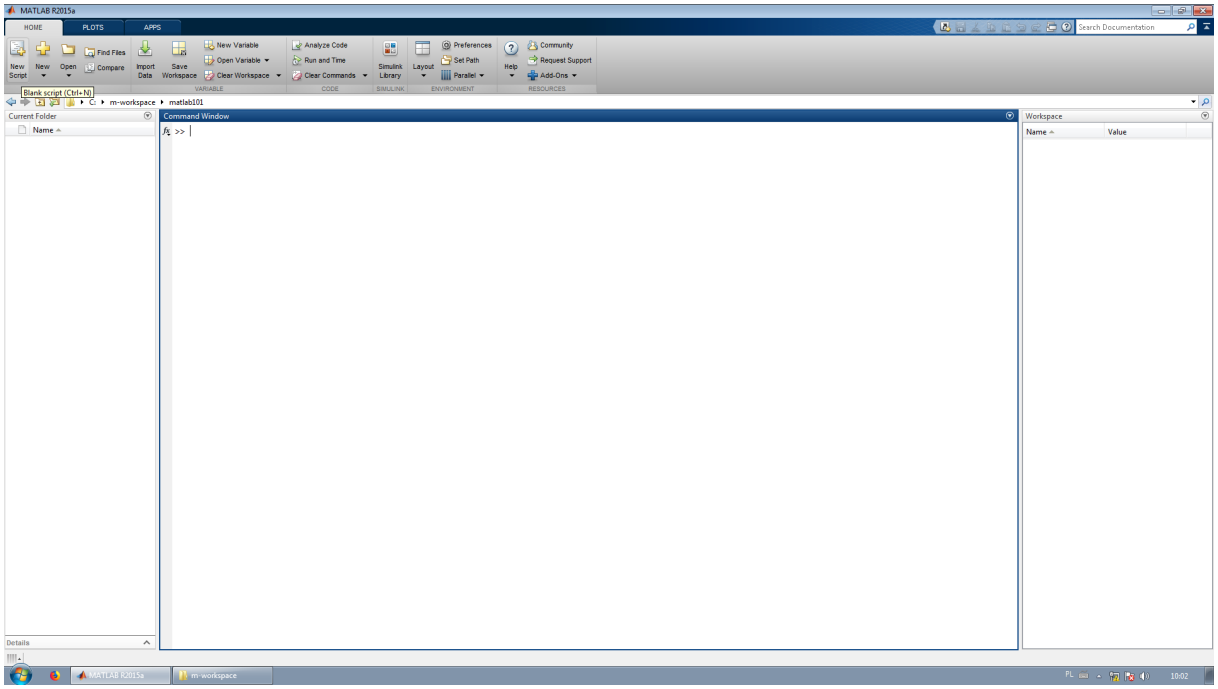
ESI: Wprowadzenie do środowiska Matlab

[Matlab_1.1] Matlab2015b i nowsze

4 marca 2019

1. **Cel ćwiczeń:** Celem ćwiczeń jest zapoznanie się studentów ze środowiskiem Matlab, które będzie stosowane w trakcie trwania kursu.

2. **Interfejs graficzny programu Matlab:** Rysunek 1 przedstawia okno główne programu. Składa się ono z trzech mniejszych okien, od prawej do lewej jest to okno z folderami i plikami przestrzeni roboczej (eksplorator plików), okno wiersza poleceń oraz okno zmiennych przestrzeni roboczej (workspace). Jeżeli po uruchomieniu programu interfejs ma inny wygląd, można go przywrócić do opisanych ustawień domyślnych używając opcji **Defualt Layout** z menu **Layout** w pasku narzędzi wstążki **Home**, tj. **Home** ⟩ **Layout** ⟩ **Default layout** Najczęściej używanymi elementami interfejsu



Rysunek 1: Okno główne programu Matlab

będą

- menu **Home** ⟩ **New Script** oraz **Home** ⟩ **New** ⟩ **Function**, służące odpowiednio do tworzenia nowych skryptów oraz funkcji. Skrypty i funkcje opisane są w dalszej części listy laboratoryjnej,
- przycisk **Editor** ⟩ **Run** lub *F5* uruchamiający wybrany skrypt
- wstążka **Publish** służące do publikacji kodu źródłowego oraz wyników jego działania w wybranej przez użytkownika formie. Temat ten zostanie omówiony w dalszej części listy laboratoryjnej.

3. **Paradygmat języka Matlab:** Język Matlab jest językiem skryptowym (interpretowanym), imperatywnym (obiektowym, strukturalnym), deklaracyjnym(funkcyjnym) i macierzowym o dynamicznym, słabym typowaniu [3]. Oznacza to, iż posiada wiele zalet związanych z zastosowaniem powyższych paradygmatów, jednakże posiada on także wady tychże. Do zalet języka można zaliczyć:

   - wsparcie programowania strukturalnego jednocześnie z programowaniem obiektowym, dzięki czemu możliwe jest szybkie prototypowanie;
   - połączenie imperatywnego i deklaratywnego pozwala na zaniedbanie procesu projektowania na rzecz szybkiego otrzymania wyników;
   - niezależność od platformy sprzętowej i programowej;
   - możliwość zastosowania mechanizmu refleksji;
   - możliwość zastosowania funkcji anonimowych;
   - dynamiczne typowanie pozwala na ponowne użycie zmiennych oraz zaniedbanie deklaracji;
   - macierzowość/wielowymiarowość pozwala na generalizowanie operacji na skalarach, macierzach i tensorach zwiększając ekonomię i ergonomię programowania;

   natomiast do wad:

   - możliwość programowania obiektowego i strukturalnego może prowadzić do powstania nieczytelnego, nieskalowanego, mało użytecznego kodu;
   - nieświadome łączenie paradygmatów w procesie programowania może prowadzić do powstania wolnego, nieskalowalnego kodu nie nadającego się do ponownego użycia w innych programach lub podprogramach;
   - bez typowania statycznego łatwo popełnić błędy przy ponownym wykorzystaniu tych samych zmiennych;
   - wolniejszy czas wykonania programów niż w przypadku języków kompilowanych;

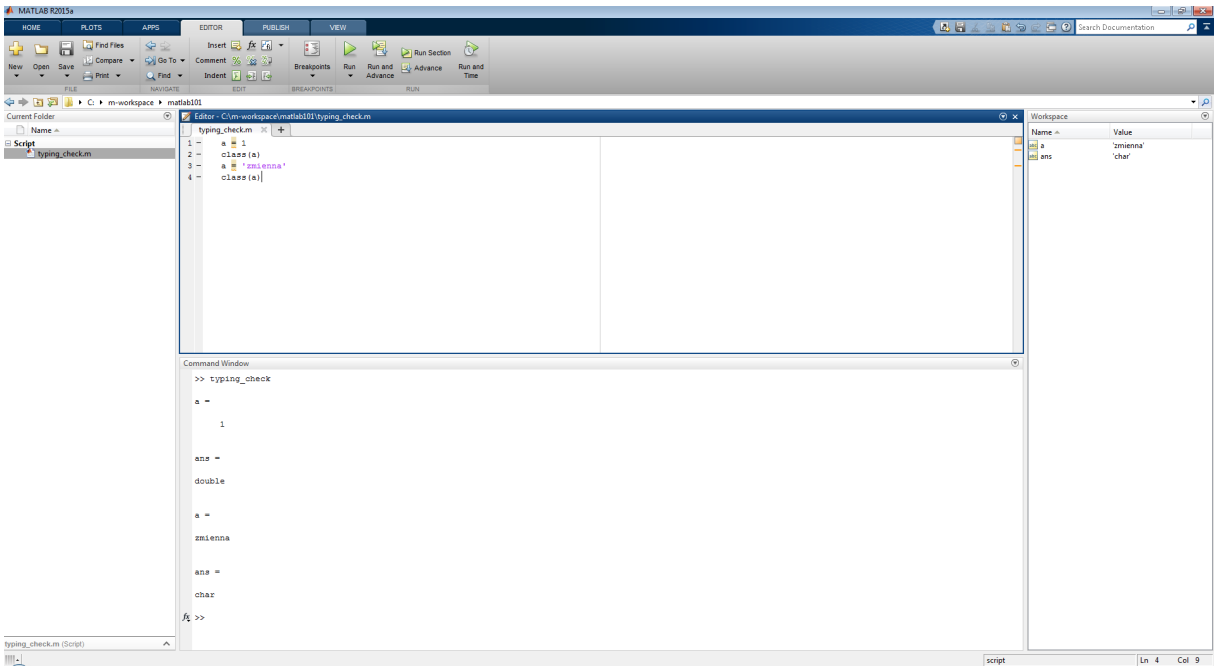
4. **Zmienne i skrypty:** Język Matlab jest językiem słabo, dynamicznie typowanym. Oznacza to, iż typ zmiennej jest automatycznie zmieniany względem kontekstu. Oznacza to również, iż nie jest konieczne ani deklarowania ani zmiennej, a jej typ jest dobierany automatycznie w trakcie inicjalizacji, patrz Listing 1. Wynik działania kolejnych przypisań wartości numerycznej oraz znakowej do zmiennej a przedstawia Rys. 2. Pomimo oczywistej zalety w postaci łatwiejszego i szybszego prototypowania programu, słabe typowanie może prowadzić do poważnych następstw, niejednokrotnie trudnych do

```
a = 1
class(a)
a = 'zmienna'
class(a)
```

Listing 1: Przykład automatycznej zmiany typu zmiennej

.

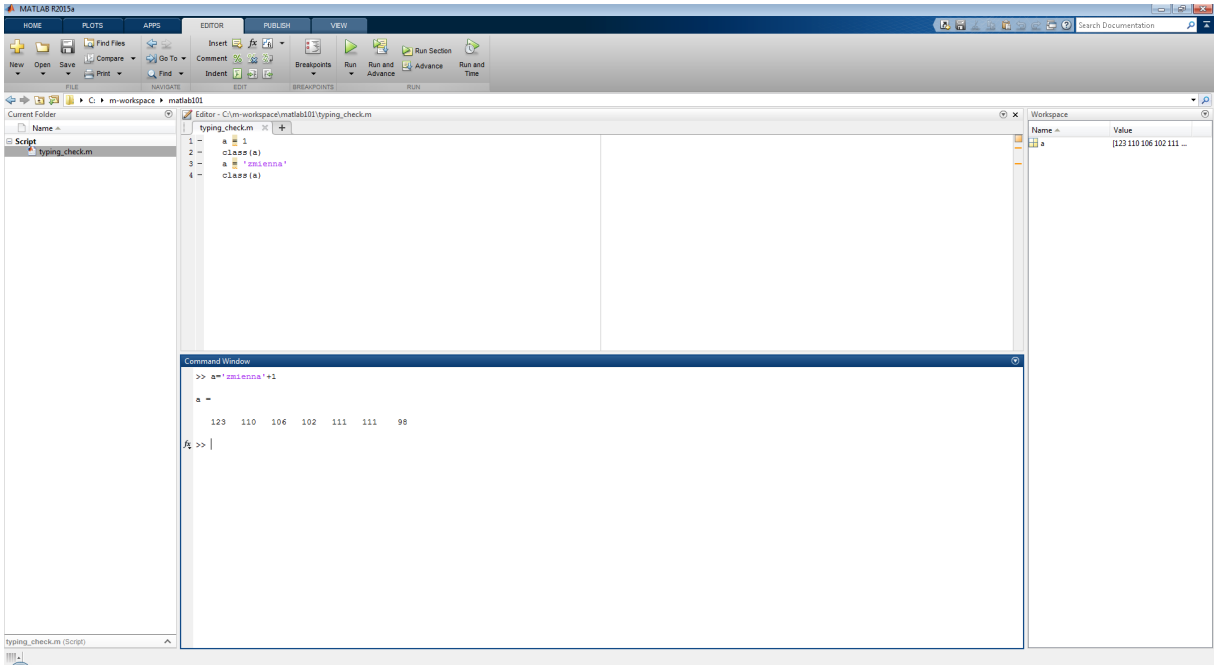
wykrycia. Za przykład może posłużyć instrukcja $a = 'zmienna' + 1$, której efekt działania przedstawia Rys. 3. Jak widać dodanie do siebie zmiennej (niejawnej) typu łańcuch znaków do zmiennej typu numerycznego powoduje zwrócenie tablicy liczb (zamiast błędu przypisania, jak można by oczekiwać w przypadku języków silnie typowanych[1]). Ostateczny typ zmiennej a z przykładu można sprawdzić używając polecenia $class(a)$. Należy zaznaczyć, iż ze względu na tablicowy charakter paradygma-



Rysunek 2: Działanie słabego typowania w praktyce

tu języka Matlab, wszystkie zmienne typów podstawowych są tożsame z tablicami zmiennych tych samych typów. Zatem zmienna typu zmiennoprzecinkowego podwójnej precyzji (double) oraz tablica/macierz o wymiarach 3×3 zmiennych tego typu będą posiadały ten sam typ, tj. double. Można tę właściwość zweryfikować poleceniem *class* wywołanym na rzecz zmiennej skalarnej oraz macierzowej. W kontekście środowiska Matlab należy rozważać wszystkie zmienne typów podstawowych jako tensory, odpowiednio zerowego, pierwszego, drugiego i wyższych rzędów [2]. Zmienne można eksportować do plików klikając prawym przyciskiem myszy na wybrane zmienne w oknie *Workspace* i wybierając opcję **Save As...**. Wszystkie zaznaczone zmienne zostaną wówczas zapisane do wybranego pliku

[1]Przy braku dynamicznego rzutowania zmiennych w jeżykach o typowaniu statycznym lub w językach z typowaniem dynamicznym, silnym (np. Python).



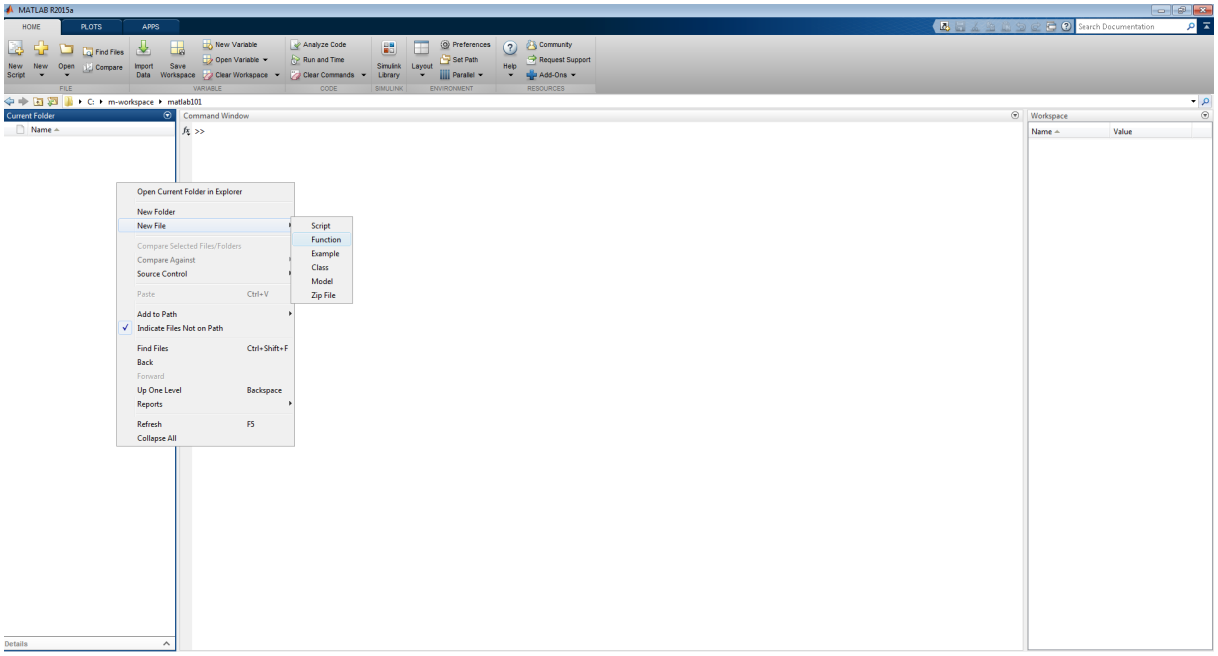
Rysunek 3: Dodawanie zmiennej numerycznej oraz łańcucha znaków

*.mat. Tak zapisane zmienne można zaimportować do przestrzeni roboczej klikając dwukrotnie w wybrany plik *.mat w oknie *Current folder*. Opcjonalnie można do tego celu wykorzystać funkcje *save* oraz *load*, patrz pomoc programu Matlab. Definiowanie zmiennych skalarnych odbywa się poprzez operator przypisania (np. $a = 4;$). Definiowanie zmiennych wektorowych, macierzowych lub skalarów wyższego rzędu wymaga zastosowania operatora nawiasu kwadratowego (np. $A = [1, 2; 3, 4]$ utworzy macierz o wymiarze 2×2). Operatory zostaną szczegółowo omówione w dalszej części laboratorium. Ponadto Matlab jest językiem skryptowym (dynamicznym), interpretowanym. Pozwala to na tworzenie skryptów składających się z wielowierszowych poleceń, wywołań funkcji lub innych skryptów. Skrypty zapisywane są w tak zwanych m-plikach o rozszerzeniu *.m. Najprostszą metodą na utworzenie nowego skryptu jest wybranie z menu menu **Home** ⟩ **New Script**. W oknie edytora (nad wierszem poleceń) pojawi się wówczas plik o nazwie Untitled.m. Przed wywołaniem skryptu przyciskiem **Editor** ⟩ **Run** lub *F5* na klawiaturze, IDE poprosi o zapisanie pliku. Nadana nazwa m-pliku będzie tożsama dla środowiska Matlab z nazwą skryptu. Odwołując się do niej można zapisany skrypt uruchomić z poziomu innego skryptu lub funkcji a także z wiersza poleceń. Prezentowane dotąd fragmenty kodu mogą być zapisywane i wywoływane w postaci skryptów. Domyślnie wynik działania każdego polecenia zwracany jest do wiersza poleceń. Aby wyłączyćz wracanie wyników do wiersza poleceń należy zakończyć polecenie średnikiem.

5. **Funkcje, klasy i przestrzenie nazw:**

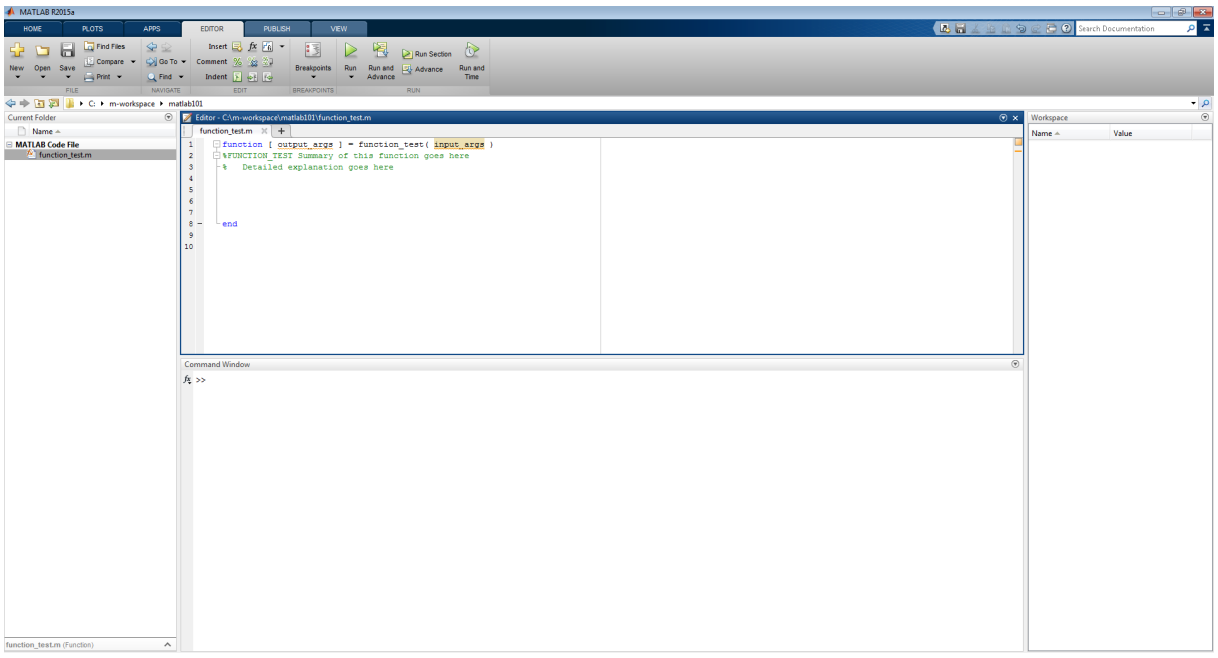
   5.1. **Funkcje:** Środowisko Matlab wspiera zarówno programowanie proceduralne oraz funkcyjne, możliwe jest także definiowanie funkcji własnych. Standardowo każda funkcja powinna być zapisana w osobnym pliku o nazwie tożsamej z nazwą funkcji oraz rozszerzeniu *.m. Najprostszą

metodą na stworzenie nowej funkcji w katalogu roboczym jest użycie prawego przycisku myszy w oknie **Current Folder** i wybranie opcji **New File** ⟩ **Function**, jak pokazano na Rysunku 4 Następnie należy nadać nazwę nowo utworzonemu plikowi, w tym przypadku *function_test.m*.



Rysunek 4: Tworzenie nowej funkcji

Po otwarciu w oknie edytora wyświetli się jego zawartość, jak pokazano na Rysunku 5. Jak

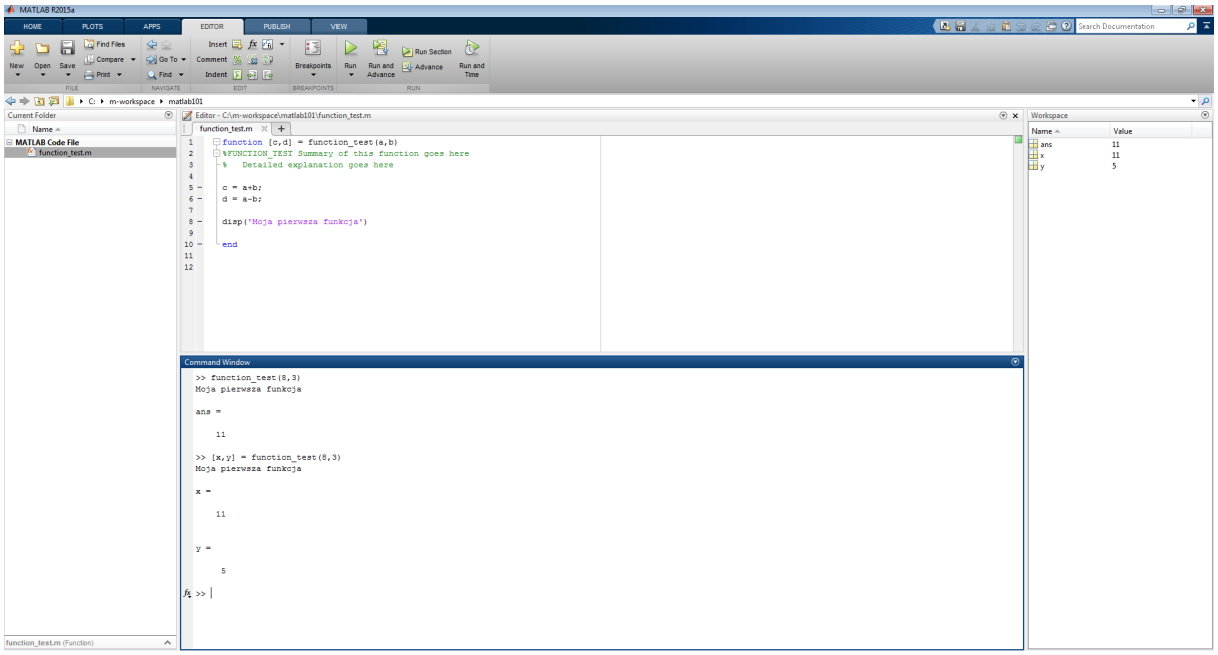


Rysunek 5: Tworzenie nowej funkcji - okno edytora

```
function [c,d] = function_test(a,b)
%FUNCTION_TEST Summary of this function goes here
%   Detailed explanation goes here

c = a+b;
d = a-b;

disp('Moja pierwsza funkcja')

end
```

Listing 2: Funkcja zagnieżdżona

widać utworzona w ten sposób funkcja posiada już zdefiniowany nagłówek. Aby sprawdzić działanie funkcji w środowisku należy zmodyfikować funkcję jak pokazano w Listingu 2. Funkcje



Rysunek 6: Wynik działania funkcji *function_test*

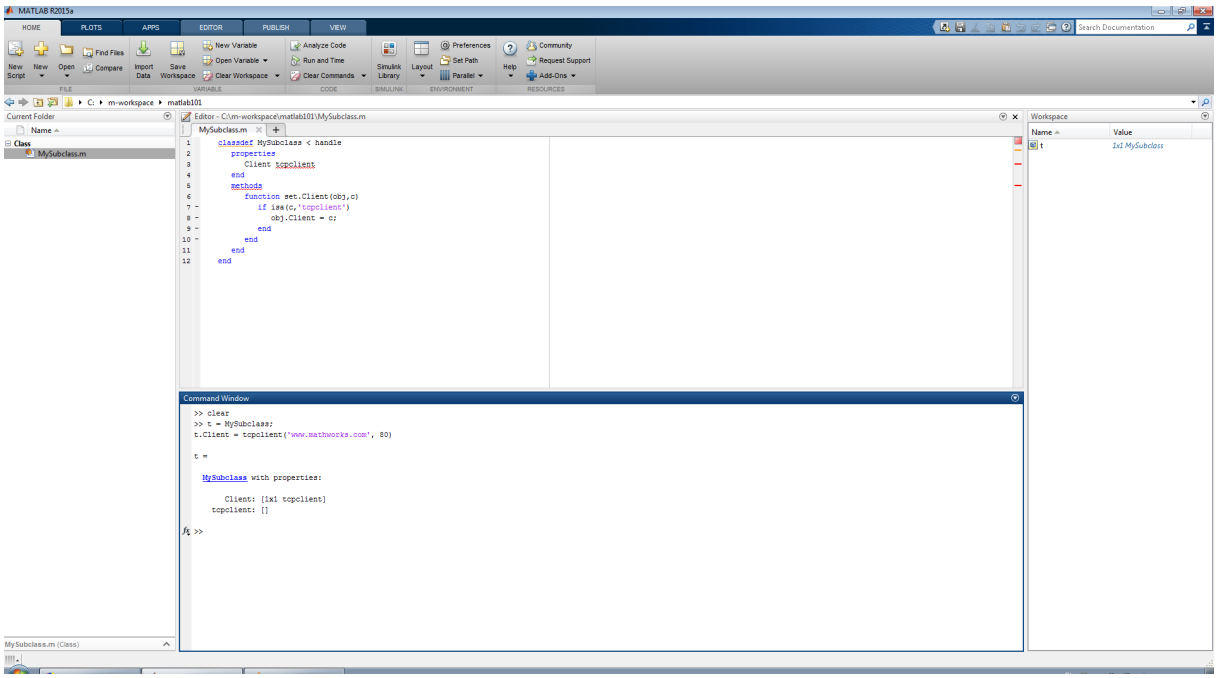
utworzone w ten sposób można wywołać z wiesza poleceń programu Matlab podając nazwę funkcji oraz jej parametry (w nawiasach okrągłych). Opcjonalnie wynik działania funkcji można przypisać do zmiennej lub zmiennych. Funkcja *function_test* zwraca dwa elementy x oraz y, przyjmuje dwa parametry a i b, $x = a + b$ natomiast $y = a - b$. Dodatkowo funkcja wywołuje funkcję wbudowaną *disp* z parametrem typu ciagu znakowego. W wyniku wywołania funkcji $function\_test(8, 3)$ program zwróci wartość 11 i przypisze ją do zmiennej domyślnej *ans*. Drugi element zostanie zaniedbany. W wyniku wywołania funkcji $[x, y] = function\_test(8, 3)$ program zwróci wartość 11 oraz przypisze ją do zmiennej x oraz wartość 3 i przypisze ją do zmiennej y. W obu przypadkach na ekranie zostanie wyświetlony komunikat podany jako parametr funkcji.

```
classdef MySubclass < handle
    properties
        Client tcpclient
    end
    methods
        function set.Client(obj,c)
            if isa(c,'tcpclient')
                obj.Client = c;
            end
        end
    end
end
```

Listing 3: klasa pochodna po klasie handle

disp. Rysunek 6 przedstawia wynik działania obu powyższych wywołań.

5.2. **Klasy:** Matlab jest językiem obiektowym. Oferuje zatem możliwość tworzenia klas oraz mechanizm polimorfii. Listing 3 pokazuje w jaki sposób można utworzyć klasę pochodną. W tym wypadku będzie to klasa dziedzicząca po *handle*. Oznacza to, iż tworzone obiekty będą wskaźnikami (kopiowanie obiektu klasy handle utworzy tak naprawdę nowy wskaźnik na ten sam obszar w pamięci). W ciele klasy zdefiniowano metodę **set.Client**(setter parametru Client). Rys. 7 pokazuje utworzenie instancji obiektu **MySubclass**



Rysunek 7: Wynik utworzenia obieku klasy MySubclass oraz inicjalizacji zmiennej wewnętrznej obiektu metodą set

```
function nested_test()

  y = 4;

  disp(y);
    function x = nested_in(z)
        y = 5;
        x = z;
    end

  q = nested_in(2);
  disp(q);
  disp(y);
end
```

Listing 4: Funkcja zagnieżdżona

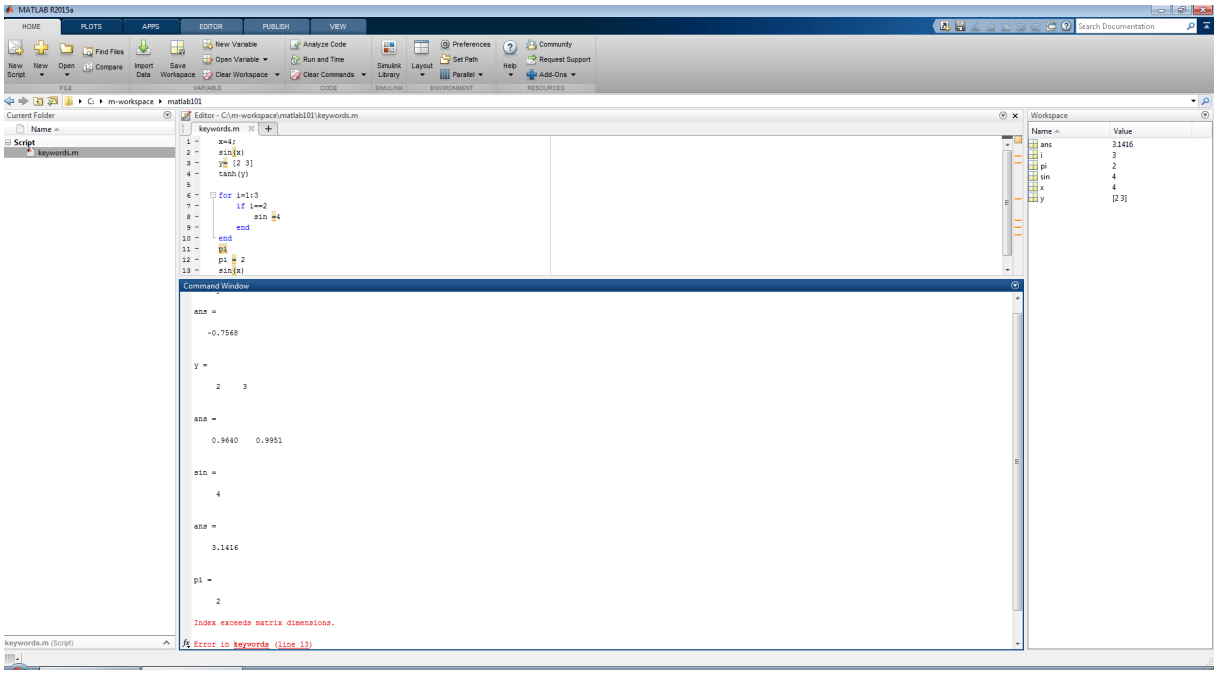
5.3. **Przestrzeń nazw:** (przestrzeń robocza, ang. Workspace). W Matlabie występuje hierarchizacja przestrzeni nazw. Najwyższą przestrzenią jest przestrzeń podstawowa (ang. Base Workspace), która przechowuje zmienne utworzone bezpośrednio w wierszu poleceń lub w skrypcie (patrz *Skrypty*). Funkcje z kolei dysponują własnymi przestrzeniami nazw. Dodatkowo każda funkcja zagnieżdżona (ang. nested function) ma dostęp do przestrzeni roboczej funkcji nadrzędnej [1], jednakże jeżeli funkcja nadrzędna nie definiuje danej zmiennej to pozostaje ona lokalna dla zadanej funkcji zagnieżdżonej. Korzystając z wiedzy zdobytej na temat funkcji należy zaimplementować funkcję *nested_test* jak pokazano na Listingu 4

6. **Słowa kluczowe, funkcje wbudowane oraz predefiniowane stałe:** W języku Matlab, podobnie jak w innych językach programowania występują słowa kluczowe. Zaliczyć do nich należy instrukcje warunkową, pętle, instrukcje końca bloku (funkcji, pętli lub instrukcji warunkowej). Należy pamiętać, iż słów kluczowych nie można wykorzystywać jako nazw zmiennych lub funkcji. Oprócz słów kluczowych, Matlab posiada również szereg funkcji wbudowanych oraz predefiniowanych stałych. Środowisko Matlab powstało na potrzeby obliczeń matematycznych i inżynierskich, w związku z tym funkcje matematyczne są już zaimplementowane i dostępne dla użytkownika w formie funkcji wbudowanych. Podobnie predefiniowane stałe matematyczne. W odróżnieniu od słów kluczowych wbudowane funkcje i stałe nie mają mechanizmu sprawdzania przypisania w trakcie przetwarzania programu, możliwe zatem jest przypisanie wartości innej niż zdefiniowana do predefiniowanej stałej. Dodatkowo można do funkcji przypisać wartość stałą. Należy o tym pamiętać w trakcie pisania programu, ponieważ z reguły wywoła to niepożądany efekt i w konsekwencji może doprowadzić do błędnych wyników prowadzonych eksperymentów, bez łatwej możliwości wykrycia przyczyny. Listing 5 przedstawia słowa kluczowe, stałe oraz funkcje wbudowane a także konsekwencje błędnych przypisań. Jak widać na Rys. 8 w języku Matlab możliwe jest przypisanie do referencji do funkcji *sin* wartości skalarnej. Podobnie można nadpisać funkcję wbudowaną inną funkcją używając operatora uchwytu @, np.

```
x=4;
sin(x)
y= [2  3]
tanh(y)

for  i=1:3
     if  i==2
          sin  =4
     end
end
pi
pi  =  2
sin(x)
```

Listing 5: Słowa kluczowe

$sin = @cos$. Po wywołaniu powyższego, $sin(x)$ będzie zwracać $cos(x)$.



Rysunek 8: Słowa kluczowe i konsekwencje przypisań do stałych

7. **Operatory:** Język Matlab posiada kilka przydatnych operatorów. Oprócz podstawowych operatorów arytmetycznych $(+,-,*,/)$, należy również zapamiętać

   7.1. \ operator lewostronnego mnożenia przez odwrotność $A\backslash B = inv(A) * B$

   7.2. ^operator potęgowania

   7.3. [] operator definiowania wektora lub macierzy $A = [1, 2, 3; 4, 6, 7]$

7.4. : operator zakresu $[1:4]$, używany również jako operator wyboru całego zakresu $A(:,1)$

7.5. $'$ operator transpozycji macierzy

7.6. $\sim$ operator negacji $A \sim= B$

8. **Produkt Hadamarda i operacje pokrewne:** Produkt Hadamarda lub inaczej produkt Schura to operacja na macierzach wykorzystująca mnożenie element przez element, tj.

$$A \otimes B = A_{i,j} B_{i,j} \forall i, j \tag{1}$$

w środowisku Matlab aby wykonać takie działanie należy operator mnożenia poprzedzić operatorem kropki $A. * B$. Operator kropki będzie miał analogiczne działanie dla operatorów potęgowania, lewo- oraz prawostronnego mnożenia przez odwrotność.

9. **Zadania::**

9.1. wykorzystując polecenie help oraz doc sprawdzić różnicę w działaniu operatorów / oraz \, jak pokazano na Listingu 6.

```
help /
help \
doc \
doc /
```

Listing 6: Użycie poleceń help oraz doc

9.2. obliczyć pole koła o promieniu $r = 4$ wykorzystując predefiniowaną stałą π

9.3. zdefiniować 20-elementowy wektor kolejnych liczb nieparzystych wykorzystując operator zakresu, sprawdzić działanie operatora zakresu wykorzystując polecenie doc

9.4. korzystając z polecenia help sprawdzic działanie funkcji *ezplot* oraz narysować wykres $sin^2(x)$

9.5. zdefiniować przestrzeń liniową 100 elementową z zakresu $\langle -2\pi, 2\pi \rangle$ używając operatora zakresu lub funkcji *linspace*, wyznaczyć wartości funkcji $\frac{e^x - e^{-x}}{e^x + e^{-x}}$ w każdym z punktów przestrzeni liniowej a następnie narysować wykres tej funkcji używając funkcji *plot*

9.6. korzystając z polecenia *reshape* stworzyć macierz o wymiarach 3×4 i wypełnić ją kolejnymi wartościami zaczynając od 10

9.7. utworzyć 100-elementowy wektor losowych liczb naturalnych dodatnich a następnie zapisać go w postaci macierzy o liczbie wierszy równej liczbie unikalnych liczb w wektorze oraz 100 kolumnach a następnie dla każdego elementu oryginalnego wektora zakodować jego wartość w postaci 1 w polu o indeksie równym wartości tego elementu

Literatura

[1] Nested functions. `https://www.mathworks.com/help/matlab/matlab_prog/nested-functions.html`.

[2] T.S. Blyth and E.F. Robertson. *Basic Linear Algebra*. Springer Undergraduate Mathematics Series. Springer London, 2013.

[3] P. Van-Roy and S. Haridi. *Concepts, Techniques, and Models of Computer Programming*. Concepts, Techniques, and Models of Computer Programming. Prentice-Hall, 2004.