Politechnika Wrocławska

Aplikacje pisane w MATLABie

Zastosowanie programu MATLAB w zagadnieniach inżynierskich

Wojciech Myszka

Katedra Mechaniki, Inżynierii Materiałowej i Biomedycznej

5 stycznia 2025

HR EXCELLENCE IN RESEARCH

Politechnika Wrocławska

1. Idea
2. MATLAB Kompilator
3. MATLAB WebApp Server
4. App Designer
5. GUIDE
6. Programmatic App

Politechnika Wrocławska

Idea

Jak wykorzystać MATLABa do przygotowania aplikacji?

1. Chodzi nam o stworzenie programu, którego nie będzie się dało zmodyfikować (lub niechcący zepsuć).

   Takiego, żeby mógł go używać „każdy"

2. Skrypt (ani plik typu `.m` ani typu `.mlx`) nie będzie rozwiązaniem do końca spełniającym oczekiwania, ale na pewno będzie rozwiażaniem najprostszym

Co jeszcze mamy do dyspozycji?

Politechnika Wrocławska

Możliwe rozwiązania

1. MATLAB Coder Pozwala na generację kodu C/C++ na podstawie kodu MATLABa
   1.1 Embeded Coder — generuje kod dla systemów wbudowanych
   1.2 GPU Coder — generuje kod CUDA dla kart graficznych (i kaceleratorów NVIDIA)
   1.3 ...
2. Simulink Coder — generuje kod na podstawie modeli Simulinka
3. ...

MATLAB Coder

- Osobna aplikacja.
- Nie jest dostępna na stronach MATLAB Online (trzeba uruchomić osobny program).
- Trzeba zainstalować
- Kompiluje tylko i wyłącznie funkcje
- Jest spory problem z typami
- Ogromna liczba funkcji MATLABa jest wspierana...

Wywoływanie funkcji MATLABa z innych języków programowania I

- C++ libraries
- Java libraries
- Python libraries
- C/C++ or Fortran MEX-file functions
- FORTRAN
- C shared libraries
- .NET libraries
- COM objects
- RESTful and WSDL web services

Piszemy program w ulubionym języku programowania i wywołujemy funkcje MATLABa...

Politechnika Wrocławska

Wywoływanie funkcji MATLABa z innych języków programowania II

1. Wszystko jest dosyć skomplikowane (w C/C++ zmienne i typy trzeba deklarować).
2. Można przesyłać obiekty między programem, a przestrzenią roboczą MATLABa (ale wymaga to akcji programisty).

C/C++

- specjalnie przygotowaną aplikację kompilujemy za pomocą specjalnego kompilatora `mex`
- Najważniejszą częścią programu jest wywołanie funkcji, która uruchamia MATLABa, i nawiązuje z nim połączenie
- Poszczególne polecenia przekazywane są do MATLABa w celu ich wykonania...
- Trzeba przygotować odpowiednie środowisko pracy, ale w końcu wszystko działa.

Politechnika Wrocławska

Python I

- Jest dziś bardzo popularny i darmowy
- Ma całkiem bogaty zestaw bibliotek rozszerzających jego możliwości...
- Ale czasami współpraca z MATLABem może się przydać...

Nie jest to łatwe, ale...

1. Trzeba nauczyć się co to środowisko wirtualne Pythona (*Python virtual env*)
2. Trzeba takie utworzyć
   - tworząc kartotekę
   - i ją wypełniając treścią

```
mkdir M-p
python3 -m venv ./M-p/
```

Python II

3. Aby zacząć ze środowiska korzystać trzeba je *aktywować*

   ```
   cd M-p
   source bin/activate
   ```

   o tym, że korzystamy z wirtualnego środowiska informuje nas dodatkowy tekst

   ```
   (M-p) myszka@szczur: $
   ```

   (w tym przypadku `M-p`)

4. W kolejnym kroku instalujemy odpowiednie oprogramowanie (`matlabengine`)

   ```
   python -m pip install matlabengine
   ```

5. I teraz (teoretycznie) możemy z niego korzystać...

Python III

```
>>> import matlab.engine
>>> eng = matlab.engine.start_matlab()
>>> eng.sqrt(4.0)
2.0
```

pracę kończymy poleceniem

```
>>> eng.quit()
```

6. Można też tak:

```
>>> import matlab.engine
>>> eng = matlab.engine.start_matlab('-desktop')
```

co spowoduje, że ten sam „silnik” będzie obsługiwał sesję Pythona i MATLABa

Python IV

```
>>> a = 5.0
>>> eng.workspace['a'] = a
```

i zmienna a stanie się dostępna w MATLABie

```
>> a
a =
     5
```

ale to nie będzie ten sam obiekt, choć można jego wartość odczytać

w MATLABie piszemy

```
a = 100;
```

w Pythonie:

Python V

```
>>> a
5.0
>>> a = eng.workspace['a']
>>> a
100.0
```

7. Mamy dostęp do funkcji MATLABa

```
eng.plus(2, 3)
5
eng.gcd(100, 64)
4
```

Typy danych I

Ale wszystko nie jest tak do końca proste: **Musimy jeszcze wziać pod uwagę typy danych!**

```
>>> zz = eng.gcd(100.0, 64.0)
>>> zz
4.0
>>> eng.workspace['zz'] = zz
```

a w matlabie mamy

```
> zz
zz =
     4
```

Jezeli jednak zrobimy coś takiego:

Typy danych II

```
>>> z = eng.gcd(100, 64)
>>> z
4
>>> eng.workspace['z'] = z
```

to w MATLABie z będzie wyglądało tak:

```
>> z
z =
  int64
   4
```

tablica z Pythona przenosi się jako *cellarray* do MATLABa

```
>>> x=[1.0, 2.0, 3.0, 4.0, 5.0]
>>> eng.workspace['x']=x
```

Typy danych III

a w MATLABie

```
>> x
x =
  1×5 cell array
    {[1]}    {[2]}    {[3]}    {[4]}    {[5]}
```

Trzeba pamiętać o wykonywaniu konwersji:

```
>>> A = matlab.double([1.,2.,3.])
>>> eng.sqrt(A)
matlab.double([[1.0,1.4142135623730951,1.7320508075688772]])
>>> eng.workspace['A'] = A
```

i teraz `A` będzie tablicą odpowiedniego typu

Politechnika Wrocławska

Szczerze...

...dosyć to marudne, bo popatrzmy (`eval` pozwala wyliczyć wartość wyrażenia)

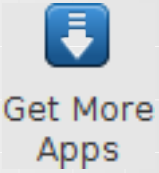
```
>>> eng.eval('a = pi',nargout=0)
a =
    3.1416
>>> eng.eval('a = pi;',nargout=0)
>>> mpi = eng.workspace['a']
>>> mpi
3.141592653589793
```

MATLAB Kompilator

Kompilator I

1. Jednym z pakietów dodatkowych jest kompilator.
2. Dosyć duży (prawie 2G).
3. Nie wszyscy go instalują.
4. Żeby zainstalować otwieramy MATLABa i przechodzimy do zakładki **APPS**

Kompilator II



5. Klikamy w Get More Apps
6. W wyszukiwarce szukamy „*Compiler*”; znajdujemy dwa narzędzia
   - Application Compiler
   - Web App Compiler
7. Wybieramy: **Application Compiler**
8. ~~Klikamy w „Sign to install” i instalujemy.~~

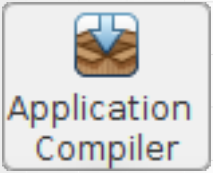
Kompilator III

Szczerze

- Zainstalowanie samego kompilatora to za mało, żeby uruchamiać aplikacje **poza środowiskiem MATLABa.**
- Potrzebny jest jeszcze tak zwany *run-time* czyli biblioteki i środowisko Java. One są w MATLABie, ale...
- ...też są wielkie (2G)
- Można to ściągnąć (dla **właściwej** wersji) z https://www.mathworks.com/products/compiler/matlab-runtime.html.
- wówczas MATLAB nie będzie potrzebny.

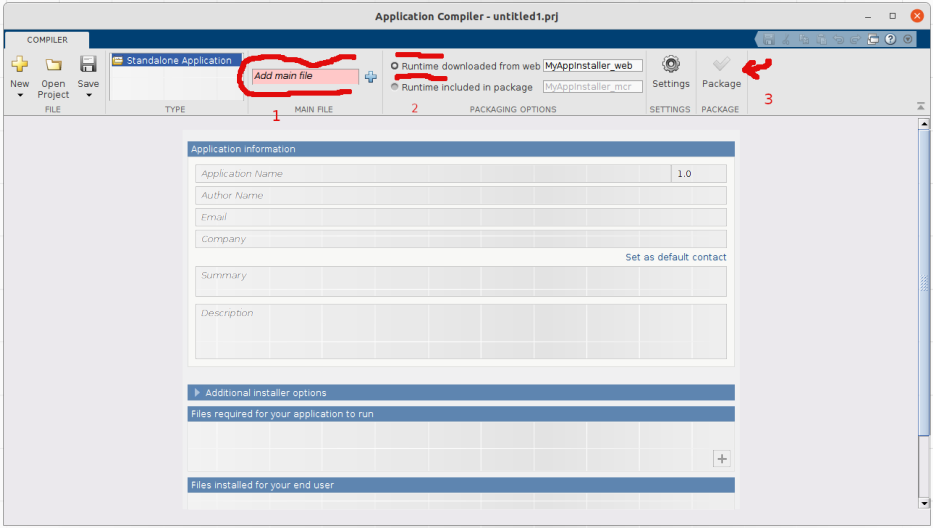
(pozwala to używać aplikacji bez zainstalowanego MATLABa)

Prosta aplikacja I

1. Gdy już mamy zainstalowany pakiet klikamy w ikonkę: 

2. Otwiera się okienko

Prosta aplikacja II



Prosta aplikacja III

3. Trzeba tam dodać plik źródłowy kodu (albo plik `.m` albo `.mlx`), który chcemy zamienić na aplikację (`main file`)
4. Wykorzystałem plik z dokumentacji:

```
function magicsquare(n)
if ischar(n)
    n=str2double(n);
end
m = magic(n);
disp(m)
```

5. Funkcja każdorazowo potrzebuje wartości...

Prosta aplikacja IV

6. Warto wybrać (domyślne) *Runtime downloaded from web* (nawet jeżeli nie mamy zainstalowanego run-timea) — przyśpiesza to budowę aplikacji; będziemy ją mogli uruchomić z wnętrze MATLABa.
7. Na koniec klikamy w ikonkę Package.

Uruchomienie stworzonej aplikacji I

Do dyspozycji mamy trzy sposoby uruchomienia utworzonej aplikacji

1. **W środowisku MATLABa**
2. **Gdy mamy MATLABa zainstalowanego na komputerze**
3. **Bez MATLABa**

Omówię je pokrótce

Uruchomienie stworzonej aplikacji II

1. W bieżącej kartotece powstałą podkartoteka `magicsquare`, w której są kolejne trzy podkartoteki

```
magicsquare
|-- for_redistribution
|   `-- MyAppInstaller_web.install
|-- for_redistribution_files_only
|   |-- magicsquare
|   |-- readme.txt
|   |-- run_magicsquare.sh
|   `-- splash.png
|-- for_testing
|   |-- magicsquare
|   |-- mccExcludedFiles.log
|   |-- readme.txt
```

Uruchomienie stworzonej aplikacji III

```
|   |-- requiredMCRProducts.txt
|   |-- run_magicsquare.sh
|   |-- splash.png
|   `-- unresolvedSymbols.txt
`-- PackagingLog.html
```

2. Każda z kartotek ma inne znaczenie
3. Poziom MATLABa
   - 3.1 Przechodzimy (w MATLABie) do kartoteki `for_testing` i uruchamiamy program:

Uruchomienie stworzonej aplikacji IV

```
>> !./magickaquare 4
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

3.2 Ten wykrzyknik to prośba żeby z poziomu systemu operacyjnego uruchomić program.

3.3 4 to argument (rozmiar kwadratu magicznego)

3.4 Uruchamiany jest on w środowisku MATLABA i to on realizuje wszystkie funkcje.

4. **Poziom systemu operacyjnego** (przechodzimy do kartoteki `for_testing`) i próbujemy uruchomić aplikację

4.1 Kończy się to niepowodzeniem

Uruchomienie stworzonej aplikacji V

```
$ ./magicsquare 5
./magicsquare: error while loading shared libraries: libmwlaur
  cannot open shared object file: No such file or directory
```

4.2 Natomiast jest specjalny skrypt `run_magicsquare.sh` któremu musimy podać gdzie jest zainstalowany MATLAB/runtime

W moim przypadku MATLAB jest w kartotece '/usr/local/MATLAB/R2022b/

```
$ ./run_magicsquare.sh /usr/local/MATLAB/R2024b/ 4
-----------------------------------------
Setting up environment variables
---
LD_LIBRARY_PATH is  :/usr/local/MATLAB/MATLAB_Runtime/v910//ru
    16     2     3    13
     5    11    10     8
```

Uruchomienie stworzonej aplikacji VI

```
 9     7     6    12
 4    14    15     1
```

Skrypt ten informuje system operacyjny gdzie ma szukać bibliotek

4.3 W Windows jest chyba inaczej... Ale jestem pewny, że idea jest identyczna (aplikacje mają rezszerzenie `.exe`)

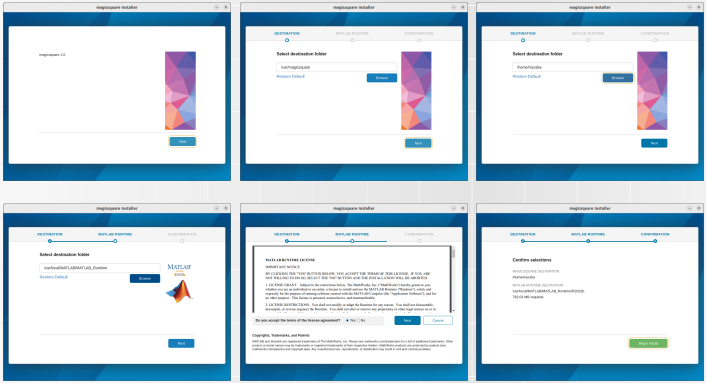
5. **Instalacja *runtajmu***

5.1 Przechodzimy do kartoteki `for_redistribution`

5.2 Plik, który się tam znajduje (`MyAppInstaller_web`) możemy przesłać na komputer, na którym **nie ma** MATLABa

Uruchomienie stworzonej aplikacji VII

5.3 Należy go tam uruchomić i odpowiedzieć na kilka prostych pytań



6. Można korzystać ze wszystkich funkcji MATLABa

Uruchomienie stworzonej aplikacji VIII

7. Po instalacji powstaje we wskazanej kartotece podkartoteka `application` z zawartością:

Szczerze

Aplikacja przygotowana w jednej wersji MATLABa **jest niekompatybilna z innymi wersjami**

> *Error: Could not find version 9.10 of the MATLAB Runtime. Attempting to load libmwmclmcrrt.so.9.10.* ***Please install the correct version of the MATLAB Runtime.*** *Contact your vendor if you do not have an installer for the MATLAB Runtime. ./magicsquare: Signal 127*

Przykład 2

Poniżej kod funkcji rysującej prosty wykres

```
function wykresik(x)
    if ischar(x)
        x=str2double(x);
    end
    t=linspace(0, 10, x);
    y = sin(t);
    plot(t,y)
    title('sin(x)')
end
```

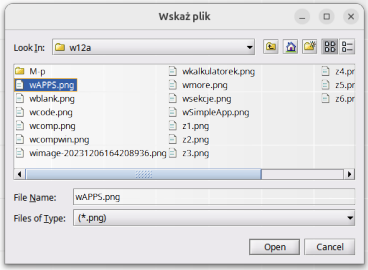
Nożna go w ramach zadania domowego przerobić na aplikację (tym razem graficzną).

Parę uwag I

1. Funkcja `uigetfile()` pozwala wybrać plik w sposób interakcyjny

   ```
   [file, location] = uigetfile(filter, title)
   ```

   otwiera okienko wyboru pliku, nadaje mu tytuł `title` i wskazuje w *bieżącej kartotece* pliki zgodne z `filter`



Parę uwag II

```
>> [file, location] = uigetfile('*.png', 'Wskaż plik')
file =
    'wAPPS.png'
location =
    '/home/myszka/Dokumenty/teksty/Dydaktyka/BIM031106/w12a/'
```

w przypadku naciśnięcia klawisza `Open`;

```
>> [file, location] = uigetfile('*.png', 'Wskaż plik')
file =
     0
location =
     0
```

w przypadku naciśnięcia klawisza `Cancel`.

MATLAB WebApp Server

Politechnika Wrocławska

Aplikacje webowe I

1. Osobny podsystem MATLABa pozwalający umieszczać na stronach WWW aplikacje stworzone w:
   - MATLABie

     i
   - Simulinku
2. Napisane jest, że aplikacje z różnych wersji MATLABa mogą być serwowane na jednym serwerze. (Ale trzeba instalować wszystkie niezbędne run-time'y)
3. Odwiedzający stronę mogą uruchamiać aplikacje bez potrzeby instalowania oprogramowania: wszystko odbywa się w przeglądarce (i na serwerze).

Politechnika Wrocławska

Aplikacje webowe II

4. Po stronie serwera potrzebne będzie:
   - 4.1 MATLAB Run-Time
   - 4.2 60 G dysku
   - 4.3 1 G RAM (minimum)
   - 4.4 1 rdzeń na 4 uruchomione aplikacje (minimum 2 rdzenie)
5. Obsługiwane przeglądarki
   - Google Chrome™,
   - Safari,
   - Firefox®,
   - Microsoft Edge®.
6. Procedura
   - 6.1 Tworzymy aplikację (AppDesigner) lub kodując
   - 6.2 Używamy aplikacji **Web App Compiler**

Aplikacje webowe III

6.3 Transferujemy na serwer

7. (Nie zajmujemy się tym)

App Designer

Politechnika Wrocławska

App Designer

- Osobna aplikacja (Design App 

)
- Komponenty układa się graficznie na panelu roboczym
- Trzeba dodawać kod, gdy wymagane są interakcje między elementami
- Potrzebny jest MATLAB Compiler i Run-Time żeby aplikację uruchamiać (poza MATLABem)

Prosty przykład I

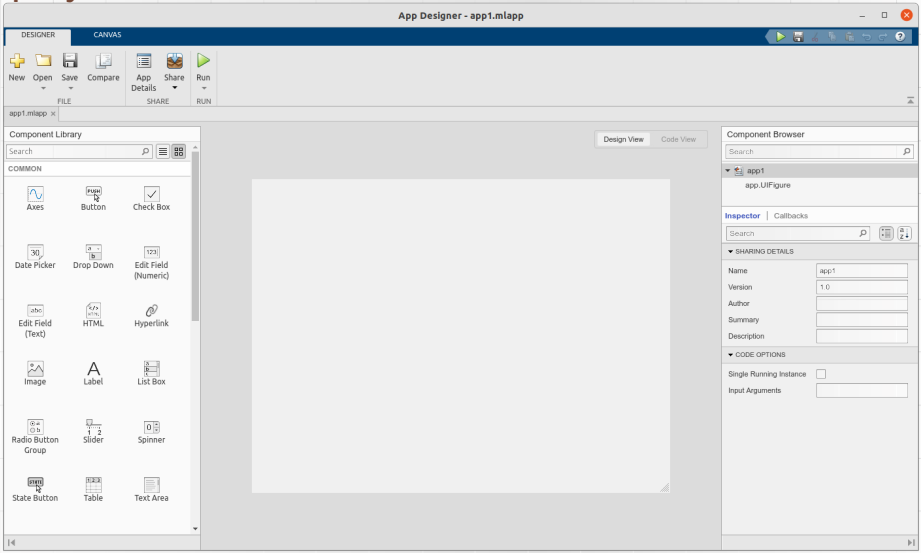
- Chcemy napisać aplikację, która sumuje dwie liczby.
- Będziemy potrzebowali pół pozwalających na wprowadzanie danych.
- Będziemy potrzebowali pola, w którym pojawi się wynik.
- Będziemy potrzebowali jakiegoś narzędzia, które powie, że wprowadzone dane są gotowe i można wykonać obliczenia.

Zaczynamy od projektu interfejsu (czyli GUI — *Graphical User Interface*)

1. Uruchamiamy App Designer i wybieramy projekt *Blank App*
2. Zaczynamy w widoku **Design View** gdzie wybieramy z biblioteki (po lewej stronie) potrzebne elementy

Politechnika Wrocławska

Prosty przykład II

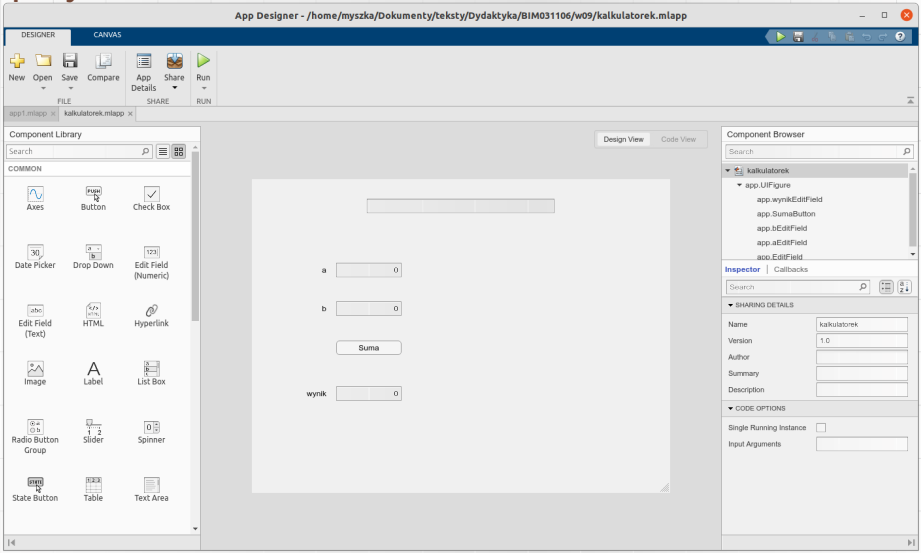


Politechnika Wrocławska

Prosty przykład III

3. Będziemy potrzebowali:
   - *Edit Field (Numeric)* — trzy razy
   - *Edit Field (Text)* — raz
   - klawisz (*Button*) do inicjacji obliczeń
4. Elementy przeciągamy i rozmieszczamy na płaszczyźnie roboczej. Pole tekstowe będzie użyte do wypisania prostego tekstu.
5. Każdy element powinien mieć unikatową nazwę — przez nią będziemy odwoływać się do obiektów (widzimy je w okienku *Component Browser*)

Prosty przykład IV

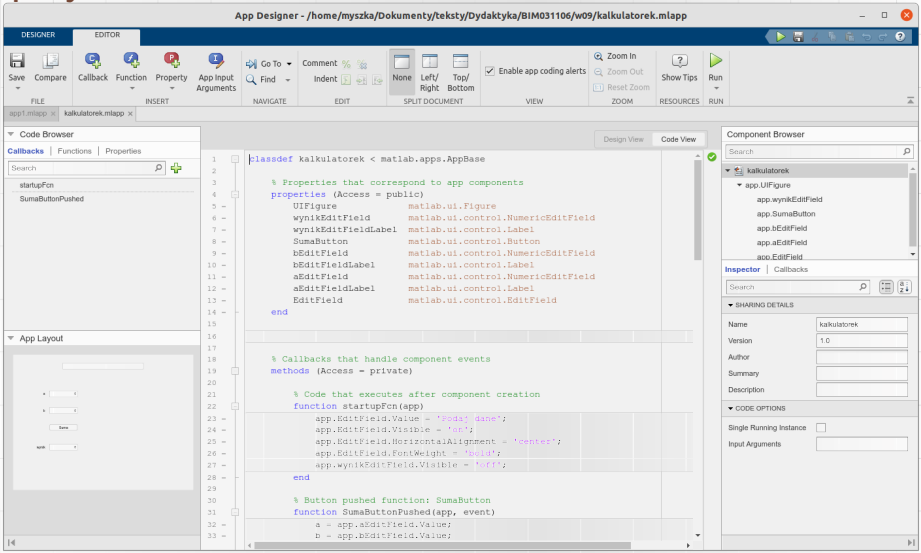


Prosty przykład V

6. Teraz musimy rozpocząć programowanie.
7. Przełączamy widok na Code View. Zobaczymy tam już sporo wstawionego kodu

Politechnika Wrocławska

Prosty przykład VI



Prosty przykład VII

8. Większość kodu (wstawiona automatycznie) jest zabezpieczona przed edycją.

Politechnika Wrocławska

Idea programowania I

- Idea programowania opiera się na funkcjach *callback*.
- Funkcje uruchamiane są w efekcie interakcji użytkownika z elementami interfejsu:
  - po uruchomieniu aplikacji (`startupFCN`)
  - naciśnięcie klawisza,
  - przesunięcie suwaka,
  - ...
- Każdy element ma zestaw swoich akcji, na które można programistycznie reagować.
- Jednym z ważniejszych „wydarzeń" jest uruchomienie aplikacji.
- Kodowanie polega na wybraniu elementu z okienka w prawym górnym rogu i wybraniem prawym klawiszem myszy *funkcji callback*
  - wskazując na aplikację (korzeń drzewa) wybieramy CallBack i Add startupFcn callback

Idea programowania II

- odpowiednia funkcja zostanie wpisana do kodu i będziemy mogli/musieli wpisać kod wpisywany na początku aplikacji; ja wpisałem to:

```
% Code that executes after component creation
function startupFcn(app)
    app.EditField.Value = 'Podaj dane';
    app.EditField.Visible = 'on';
    app.EditField.HorizontalAlignment = 'center';
    app.EditField.FontWeight = 'bold';
    app.wynikEditField.Visible = 'off';
end
```

Politechnika Wrocławska

Idea programowania III

`app.EditField` to pole tekstowe bez nazwy

`app.wynikEditField` to pole numeryczne, gdzie będzie wynik

Jak widać kod to zmiana atrybutów/właściwości tych pól

`Value` — to wartość wyświetlana w polu

`Visible` — informacja czy pole ma być widocznevarargout

- Następny fragment kodu to działania wykonywane po naciśnięciu klawisza opisanego **Wynik**

Idea programowania IV

```
% Button pushed function: SumaButton
function SumaButtonPushed(app, event)
    a = app.aEditField.Value;
    b = app.bEditField.Value;
    suma = a + b;
    app.wynikEditField.Value = suma;
    app.wynikEditField.Visible = 'on';
end
```

`app.aEditField.Value` to wartość wpisana przez użytkownika w polu **a**

`app.wynikEditField.Valuse` to wartość, która pojawi się w polu wyniku (rezultat operacji sumowania)

Budowa samodzielnej aplikacji

Identycznie jak w przypadku źródła typu `.m`/`.mlx`. Jako źródło podajemy utworzony przez Design App plik projektu o rozszerzeniu `.mlapp`.

Politechnika Wrocławska

GUIDE

GUIDE

1. GUIDE to GUI Development Environment
2. GUI to Graphical User Interface
3. Jest to jeszcze jedna metoda tworzenia aplikacji graficznych (starsza niż App Designer).
4. Zajmiemy się nią jak zostanie czasu.

Programmatic App

Politechnika Wrocławska

Programmatic App

- Kolejnym sposobem tworzenia aplikacji jest coś co w MATLABie nazywa *Programmatic App*
- Skoro każdy wykres to okienko graficzne...
- ...w prostych sytuacjach niewiele trzeba dodać, żeby wprowadzić możliwość interakcji...

Szczerze

- są dwa sposoby na budowanie aplikacji.
  - pierwszy oparty na funkcji `uifigure`
  - drugi oparty na funkcji `fgure` (i mocno powiązany z GUIDE)
- nie jest to specjalnie skomplikowane, ale...
- ...o ile w App Designer elementy przenosiło się myszą...
- ...tu trzeba wszystko „na piechotę' zaplanować i rozmieścić

Politechnika Wrocławska

uifigure

1. W aplikacjach można umieszczać
   - wykresy w układzie kartezjańskim
   - wykresy w układzie kołowym
   - mapy
2. Dostępne są następujące elemantu GUI
   - klawisz
   - przyciski i przełączniki
   - pola wyboru
   - listy rozwijalne
   - pola wprowadzania danych
   - obrazki (mapy bitowe)
   - etykiety
   - listy wyboru
   - suwaki
   - pokrętła
   - ...

Przykład I

1. Nie silę się na coś oryginalnego, podążam za przykładem z dokumentacji
2. Tworzymy skrypt o nazwie takiej jaka będzie nazwa funkcji.

```
function simpleApp
% SIMPLEAPP Interactively explore plotting functions
%   Choose the function used to plot the sample data to see
%   the differences between surface plots, mesh plots,
%   and waterfall plots
```

3. Komentarz to będzie Help
4. zasadnicza najważniejsza część

```
% Create figure window
fig = uifigure;
fig.Name = "Moja apka";
```

Przykład II

5. tu właśnie korzystamy z podstawowej funkcji `uifigure`, która tworzy okienko graficzne i udostępnia mechanizmy pozwalające na sterowanie pracą.
6. Aby łatwiej rozmieszczać graficzne elementy UI (*User interface*), trzeba stworzyć siatkę

```
% Manage app layout
gl = uigridlayout(fig,[2 2]);
gl.RowHeight = {30,'1x'};
gl.ColumnWidth = {'fit','1x'};
```

7. czyli w obiekcie `fig` dwa wiersze dwie kolumny (2x2)
8. Jest drobny kłopot z rozmiarami; można je podawać jako
   - liczbę (wartość w pikselach)
   - `fit` rozmiar dopasowuje się do zawartości automatycznie

Przykład III

- wartość względna złożona z liczby i znaku `x`, na przykład `2x` (*dwa iks*)

Jeżeli mamy cztery wiersze i ich rozmiary (wysokość) podamy jako `['fit',200,'2x','1x]` to

- pierwszy będzie miał wysokość zależną od zawartości,
- drugi 200 pikseli
- trzeci i czwarty zajmą całą resztę miejsca, ale trzeci zawsze będzie dwa razy wyższy niż czwarty

użycie *n*`x` ma sens kiedy zmieniamy rozmiar okienka

```
% Create UI components
lbl = uilabel(gl);
dd = uidropdown(gl);
ax = uiaxes(gl);
```

Przykład IV

9. Powyższe tworzy trzy komponenty
   - etykietę (napis)
   - listę rozwijaną
   - miejsce na wykres
10. Zwracam uwagę na hierarchię:
    10.1 na samej „górze” jest `fig` (*figure*)
    10.2 dzieckiem fig jest `gl` (*gridlayout*)
    10.3 dziećmi `gl` są `lbl`, `dd`, `ax`
11. Teraz rozmieszczamy elementy

```
% Lay out UI components
% Position label
lbl.Layout.Row = 1;
lbl.Layout.Column = 1;
% Position drop-down
```

Przykład V

```
dd.Layout.Row = 1;
dd.Layout.Column = 2;
% Position axes
ax.Layout.Row = 2;
ax.Layout.Column = [1 2];
```

- etykieta będzie w pierwszym wierszu, pierwszej kolumnie
- menu w pierwszym wierszu, drugiej kolumnie
- wykres zajmie dolną część (obie kolumny)

12. Wypełniamy treścią

Przykład VI

```
% Configure UI component appearance
lbl.Text = "Rodzaj wykresu:";
dd.Items = ["Surf" "Mesh" "Waterfall"];
dd.Value = "Surf";
surf(ax,peaks);
```

- pojawia się tekst etykiety
- pozycje menu
- wartość domyślna (`surf`)
- dodatkowo rysowany jest wykres `surf`

13. Na sam koniec musimy zdefiniować funkcję *callback*

```
% Assign callback function to drop-down
dd.ValueChangedFcn = {@changePlotType,ax};
```

Przykład VII

- w przypadku interakcji z każdą pozycją, która może być zmieniana musi być powiązana funkcja, która na zmiany reaguje; w naszym przypadku może zmieniać się wybór z lisy (`dd`) więc w odpowiednim parametrze wpisujemy
- pierwszy element *cell array* to wskaźnik na funkcję
- drugi parametr to informacja gdzie ma się zmieniać wykres

14. Gdy użytkownik wybierze pozycję z menu zostanie wywołana funkcja z trzema parametrami: pierwszy to źródło, druki wybrana pozycja i trzeci — parametr użytkownika
15. I to jest koniec głównej funkcji

```
end
```

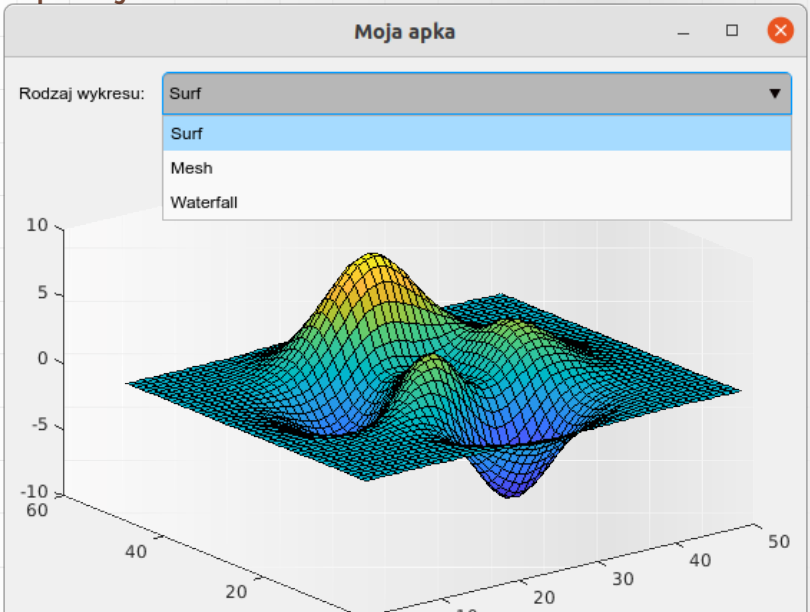
16. Teraz musimy zdefiniować funkcję związaną z menu rozwijanym

Przykład VIII

```
% Program app behavior
function changePlotType(src,event,ax)
type = event.Value;
switch type
    case "Surf"
        surf(ax,peaks);
    case "Mesh"
        mesh(ax,peaks);
    case "Waterfall"
        waterfall(ax,peaks);
end
end
```

- element struktury `event.Value` zawiera wybór użytkownika i używany jest do wskazania wykresu, który ma wypełnić dolną część obrazka.

Kompilacja


Moja apka
Rodzaj wykresu:
Surf
Surf
Mesh
Waterfall
10
5
0
-5
-10
60
40
20
50
40
30
20