TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

Faculty of Electronics, Telecommunications and Information Technology

SEMESTER PROJECT

Fundamentals of Computer Aided Graphics

Class A Amplifier

Supervisors:

Conf. Dr. Ing. Mihaela Cirlugea

Student:

Onuț Elena-Sabina

Group 2023

Table of contents

I. Introduction

1. The electronic amplifier

An amplifier is an electronic device that can increase the power of a signal (a time-varying voltage or current). It is a two-port electronic circuit that uses electric power from a power supply to increase the amplitude of a signal applied to its input terminals, producing a proportionally greater amplitude signal at its output. The amount of amplification provided by an amplifier is measured by its gain: the ratio of output voltage, current, or power to input. An amplifier is a circuit that has a power gain greater than one.

An amplifier can either be a separate piece of equipment or an electrical circuit contained within another device. Amplification is fundamental to modern electronics, and amplifiers are widely used in almost all electronic equipment. Amplifiers can be categorized in different ways. One is by the frequency of the electronic signal being amplified.

For example, audio amplifiers amplify signals in the audio (sound) range of less than 20 kHz, RF amplifiers amplify frequencies in the radio frequency range between 20 kHz and 300 GHz, and servo amplifiers and instrumentation amplifiers may work with very low frequencies down to direct current.

Amplifiers can also be categorized by their physical placement in the signal chain; a preamplifier may precede other signal processing stages, for example. The first practical electrical device which could amplify was the triode vacuum tube was invented in 1906 by Lee De Forest, which led to the first amplifiers around 1912. Today most amplifiers use transistors.

2. Power amplifier classes

In electronics, power amplifier classes are letter symbols applied to different power amplifier types. The class gives a broad indication of an amplifier's characteristics and performance. The classes are related to the time period that the active amplifier device is passing current, expressed as a fraction of the period of a signal waveform applied to the input. A class A amplifier is conducting through all the period of the signal; Class B only for one-half the input period, class C for much less than half the input period. A Class D amplifier operates its output device in a switching manner; the fraction of the time that the device is conducting is adjusted so a pulse width modulation output is obtained from the stage.

Additional letter classes are defined for special purpose amplifiers, with additional active elements or particular power supply improvements; sometimes a new letter symbol is used by a manufacturer to promote its proprietary design.

3. Matlab

I.3.1. Short description

MATLAB (matrix laboratory) is a numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran.

Although MATLAB is intended primarily for numerical computing, an optional toolbox uses the MuPAD symbolic engine, allowing access to symbolic computing capabilities. An additional package, Simulink, adds graphical multi-domain simulation and Model-Based Design for dynamic and embedded systems.

I.3.2. Matlab history

Cleve Moler, the chairman of the computer science department at the University of New Mexico, started developing MATLAB in the late 1970s. He designed it to give his students access to LINPACK and EISPACK without them having to learn Fortran. It soon spread to other universities and found a strong audience within the applied mathematics community. Jack Little, an engineer, was exposed to it during a visit Moler made to Stanford University in 1983. Recognizing its commercial potential, he joined with Moler and Steve Bangert. They rewrote MATLAB in C and founded MathWorks in 1984 to continue its development. These rewritten libraries were known as JACKPAC. In 2000, MATLAB was rewritten to use a newer set of libraries for matrix manipulation, LAPACK.

MATLAB was first adopted by researchers and practitioners in control engineering, Little's specialty, but quickly spread to many other domains. It is now also used in education, in particular the teaching of linear algebra and numerical analysis, and is popular amongst scientists involved in image processing.

I.3.3. Matlab syntax

The MATLAB application is built around the MATLAB language, and most use of MATLAB involves typing MATLAB code into the Command Window (as an interactive mathematical shell), or executing text files containing MATLAB codes, including scripts and/or functions.

II. General theoretical aspects

1. Power amplifier classes

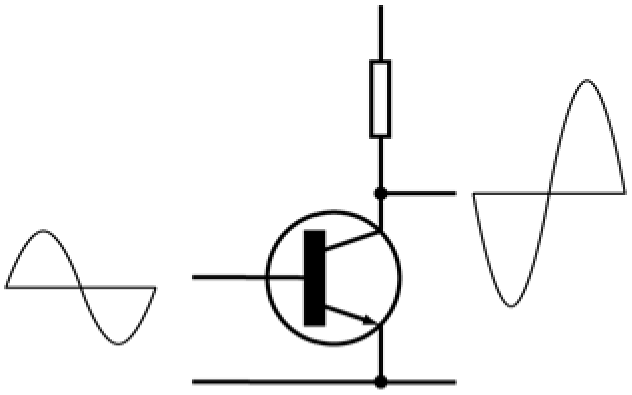
Power amplifier circuits (output stages) are classified as A, B, AB and C for linear designs—and class D and E for switching designs. The classes are based on the proportion of each input cycle (conduction angle) during which an amplifying device passes current. The image of the conduction angle derives from amplifying a sinusoidal signal. If the device is always on, the conducting angle is 360°. If it is on for only half of each cycle, the angle is 180°. The angle of flow is closely related to the amplifier power efficiency.

In the illustrations below, a bipolar junction transistor is shown as the amplifying device. However the same attributes are found with MOSFETs or vacuum tubes.

2. Class A

In a class-A amplifier, 100% of the input signal is used (conduction angle Θ = 360°). The active element remains conducting all of the time.

Amplifying devices operating in class A conduct over the entire range of the input cycle. A class-A amplifier is distinguished by the output stage devices being biased for class A operation. Subclass A2 is sometimes used to refer to vacuum-tube class-A stages that drive the grid slightly positive on signal peaks for slightly more power than normal class A (A1; where the grid is always negative). This, however, incurs higher signal distortion.


Class A Amplifier

II.2.1. Advantages of class-A amplifiers

- Class-A designs can be simpler than other classes insofar as class -AB and -B designs require two connected devices in the circuit (push–pull output), each to handle one half of the waveform whereas class A can use a single device (single-ended).
- The amplifying element is biased so the device is always conducting, the quiescent (small-signal) collector current (for transistors; drain current for FETs or anode/plate current for vacuum tubes) is close to the most linear portion of its transconductance curve.

- Because the device is never 'off' there is no "turn on" time, no problems with charge storage, and generally better high frequency performance and feedback loop stability (and usually fewer high-order harmonics).
- The point where the device comes closest to being 'off' is not at 'zero signal', so the problems of crossover distortion associated with class-AB and -B designs is avoided.
- Best for low signal levels of radio receivers due to low distortion.

II.2.2. Disadvantage of class-A amplifiers

- Class-A amplifiers are inefficient. A maximum theoretical efficiency of 25% is obtainable using usual configurations, but 50% is the maximum for a transformer or inductively coupled configuration. In a power amplifier, this not only wastes power and limits operation with batteries, but increases operating costs and requires higher-rated output devices. Inefficiency comes from the standing current, which must be roughly half the maximum output current, and a large part of the power supply voltage is present across the output device at low signal levels. If high output power is needed from a class-A circuit, the power supply and accompanying heat becomes significant. For every watt delivered to the load, the amplifier itself, at best, uses an extra watt. For high power amplifiers this means very large and expensive power supplies and heat sinks.
- Because the output devices are in full operation at all times (unlike a class A/B amplifier), they will not have as long a life unless the amplifier is specifically over-designed to take this into account, adding to the cost of maintaining or designing the amplifier.
- Class-A power amplifier designs have largely been superseded by more efficient designs, though their simplicity makes them popular with some hobbyists. There is a market for expensive high fidelity class-A amps considered a "cult item" among audiophiles mainly for their absence of crossover distortion and reduced odd-harmonic and high-order harmonic distortion. Class A power amplifiers are also used in some "boutique" guitar amplifiers due to their unique tonal quality and for reproducing vintage tones.

II.2.3. Single-ended and triode class-A amplifiers

Some hobbyists who prefer class-A amplifiers also prefer the use of thermionic valve (tube) designs instead of transistors, for several reasons:

- Single-ended output stages have an asymmetrical transfer function, meaning that even-order harmonics in the created distortion tend to not cancel out (as they do in push–pull output stages). For tubes, or FETs, most distortion is second-order harmonics, from the square law transfer characteristic, which to some produces a "warmer" and more pleasant sound.
- For those who prefer low distortion figures, the use of tubes with class A (generating little odd-harmonic distortion, as mentioned above) together with symmetrical circuits (such as push–pull output stages, or balanced low-level stages) results in the cancellation of most of the even distortion harmonics, hence the removal of most of the distortion.
- Historically, valve amplifiers were often used as a class-A power amplifier simply because valves are large and expensive; many class-A designs use only a single device.

Transistors are much less expensive than tubes so more elaborate designs that use more parts are still less expensive to manufacture than tube designs. A classic application for a pair of class-A devices is the long-tailed pair, which is exceptionally linear, and forms the basis of many more complex circuits, including many audio amplifiers and almost all op-amps.

Class-A amplifiers may be used in output stages of op-amps (although the accuracy of the bias in low cost op-amps such as the 741 may result in class A or class AB or class B performance, varying from device to device or with temperature). They are sometimes used as medium-power, low-efficiency, and high-cost audio power amplifiers. The power consumption is unrelated to the output power. At idle (no input), the power consumption is essentially the same as at high output volume. The result is low efficiency and high heat dissipation.

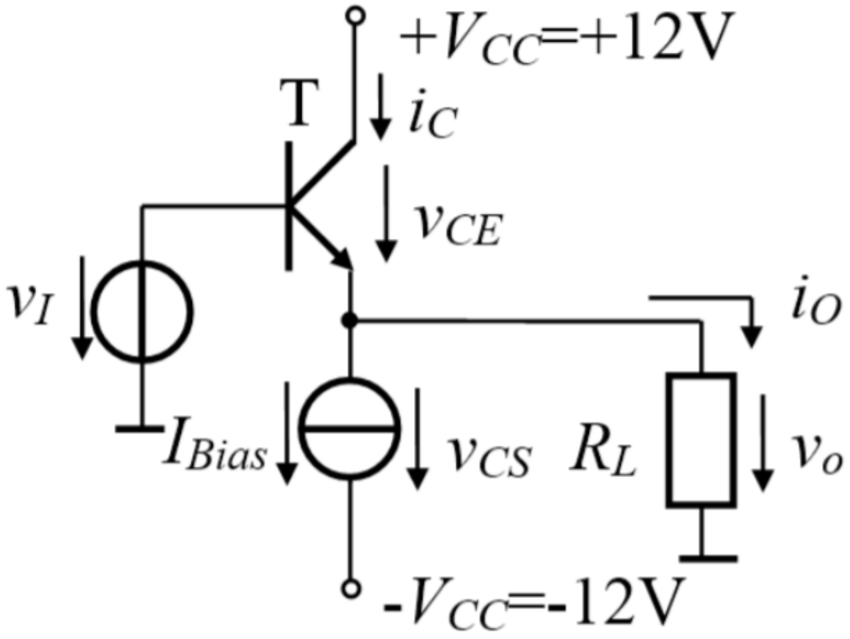
III. Experimental results

1. Problem

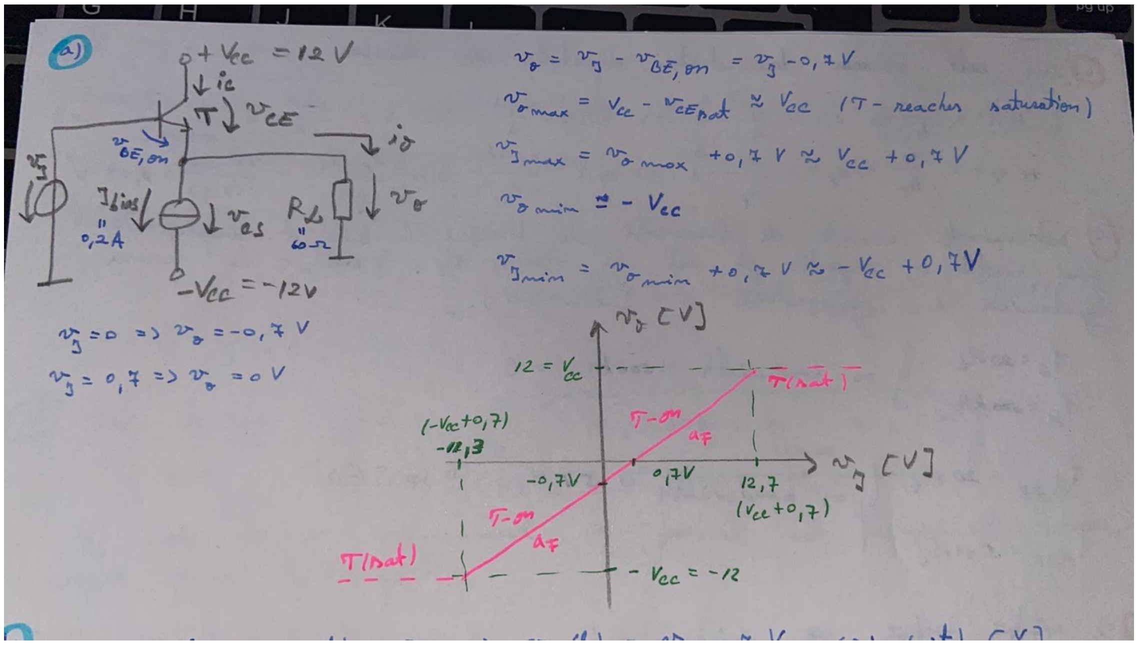
The problem that inspired me to create the GUI in Matlab is this:

For T in conduction consider vBE,on =0.7V; IBias =0.2A, RL =60Ω. a) Draw VTC vO (vI). The current source is active for vCS > 0V. Indicate the transistor state (off, aF , exc) for every region of the VTC. b) Consider vI = 0.7V+6sinωt [V]. Draw the waveforms for vO(t), iO(t), vCE(t) and iC (t). What are the peak values for all these signals? c) What is the average value of the power dissipated by the load and average efficiency for vI= 0.7V+6sinωt [V]? d) Explain why the average efficiency is less than the maximum average efficiency possible for this stage.

2. Circuit



3. The results obtained by theoretical calculation


a)

$v_I = 0 \Rightarrow v_O = -0,7\ V$

$v_I = 0,7 \Rightarrow v_O = 0\ V$

$v_O = v_I - v_{BE,on} = v_I - 0,7\ V$

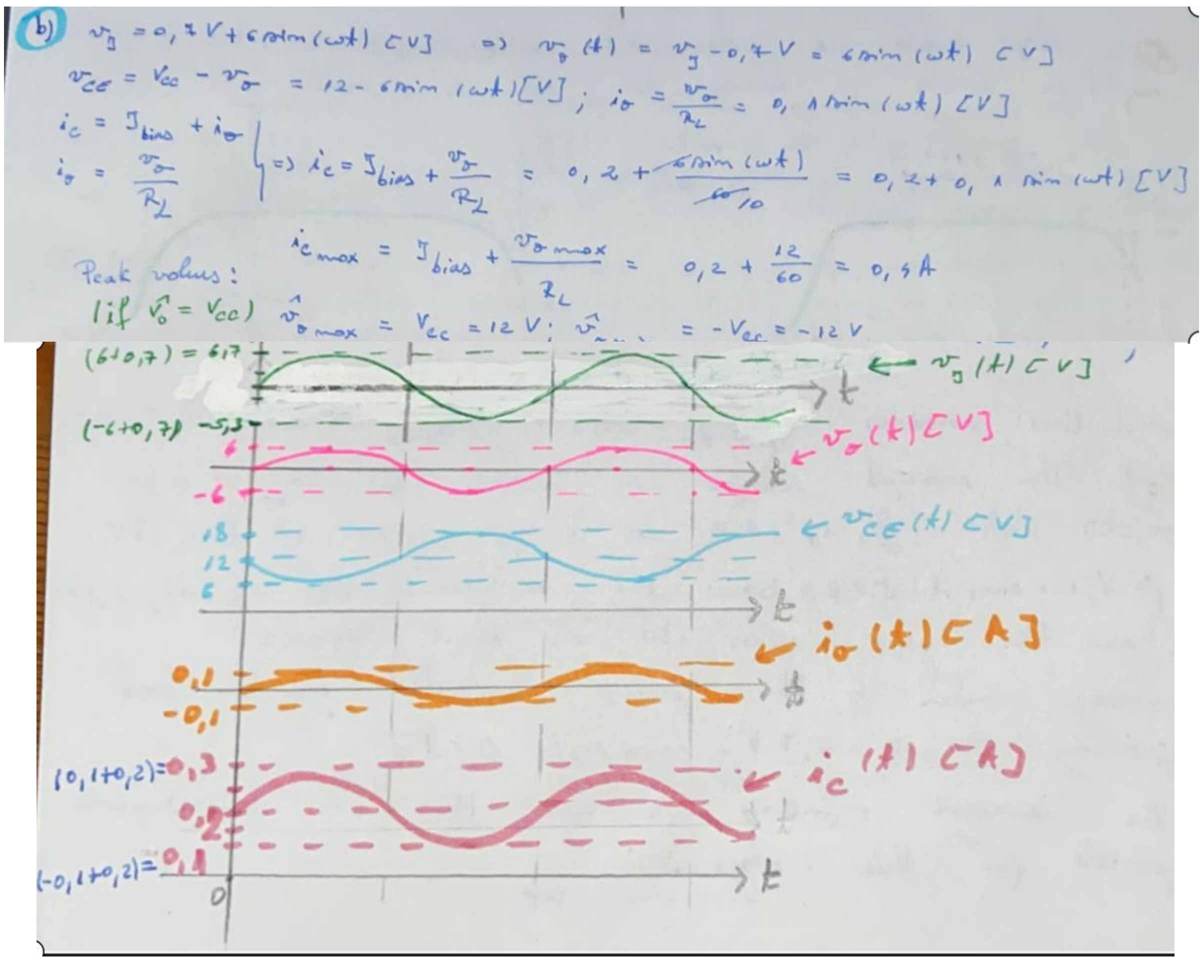
$v_{O\,max} = V_{CC} - v_{CEsat} \approx V_{CC}$ (T - reaches saturation)

$v_{I\,max} = v_{O\,max} + 0,7\ V \approx V_{CC} + 0,7\ V$

$v_{O\,min} \approx -V_{CC}$

$v_{I\,min} = v_{O\,min} + 0,7\ V \approx -V_{CC} + 0,7V$

+Vcc =12 V
ic
T
v_{CE}
$v_{BE,on}$
io
v_I
I_{bias} = 0,2A
v_{CS}
R_L = 60Ω
v_O
-Vcc = -12V
v_O [V]
12 = V_{CC}
T(sat)
T-on
AF
(-Vcc+0,7)
-11,3
-0,7V
0,7V
12,7
(Vcc+0,7)
v_I [V]
T-on
AF
T(sat)
- Vcc = -12



b) $v_I = 0,7\,V + 6\sin(\omega t)\ [V] \Rightarrow v_o(t) = v_I - 0,7\,V = 6\sin(\omega t)\ [V]$

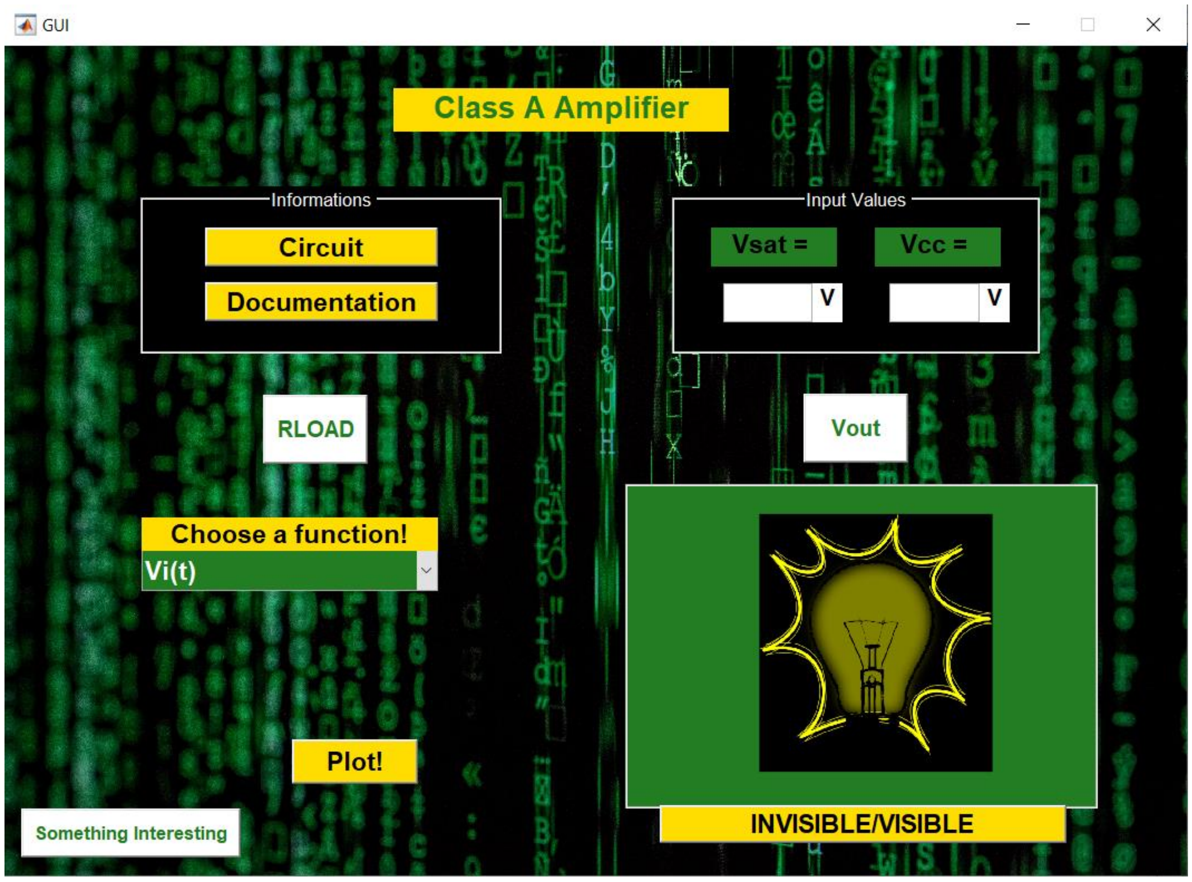
$v_{CE} = V_{CC} - v_o = 12 - 6\sin(\omega t)[V]$; $i_o = \frac{v_o}{R_L} = 0,1\sin(\omega t)\ [V]$

$$\left.\begin{aligned} i_C &= I_{bias} + i_o \\ i_o &= \frac{v_o}{R_L} \end{aligned}\right\} \Rightarrow i_C = I_{bias} + \frac{v_o}{R_L} = 0,2 + \frac{\cancel{6}\sin(\omega t)}{\cancel{60}\,10} = 0,2 + 0,1\sin(\omega t)\ [V]$$

Peak values: $i_{C\,max} = I_{bias} + \frac{v_{o\,max}}{R_L} = 0,2 + \frac{12}{60} = 0,4\,A$

(if $\hat{v}_o = V_{CC}$) $\hat{v}_{o\,max} = V_{CC} = 12\,V$; $\hat{v}$ $= -V_{CC} = -12\,V$

4. The interface in Matlab



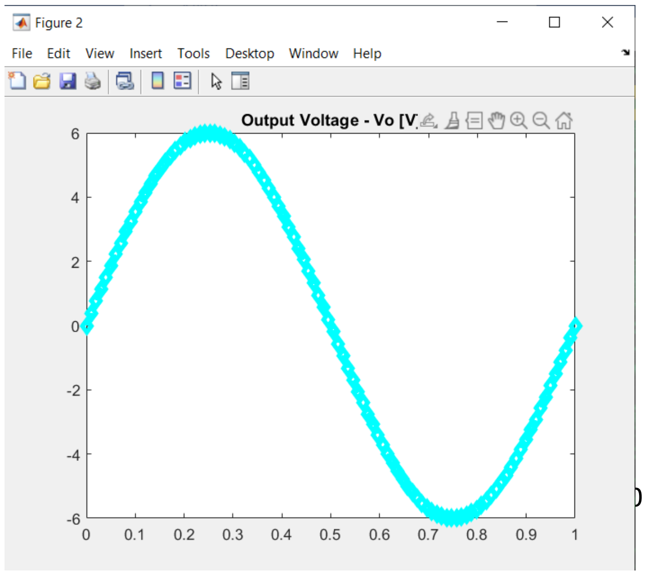
The interface contains:

- **1 popupmenu;**
- **1 background picture for the entire GUI;**
- **1 axes;**
- **2 edit buttons**, which allows us to modify the values for the saturation voltage Vsat and for Vcc;
- **3 uipanels;**
- **6 static texts;**
- **7 push buttons:**

  - **Circuit** - shows the schematics for the circuit;
  - **Documentation** – opens the pdf file regarding the theoretical documentation
  - **RLOAD** – plots the load resistance depending on the drain current and drain voltage;
  - **Vout** – plots the graphic output;
  - **Plot!** – plots a function manually selected from the list above the button(from popupmenu);
  - **INVISIBLE/VISIBLE** – makes the plotted graph or the already present image invisible or invisible;
  - **Something Interesting** – demonstrate how to control the mouse pointer from a GUI.

5. The results I obtained vs the expectations

For example, Vout from theoretical calculation is: Vi - 0.7 = 6*sin(2*pi*t) [V]. When I push the button for plotting Vout it results:

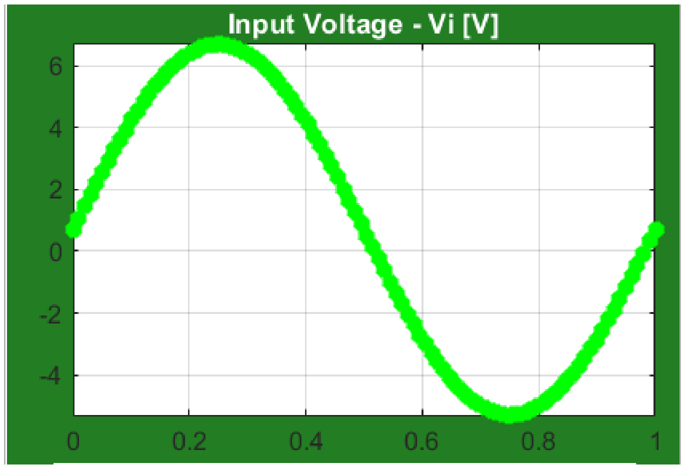


Which is correct. The signal amplitude is 6V (Vomax = 6V, Vomin = - 6V), and the shape is sinusoidal.

The code in Matlab for this section:

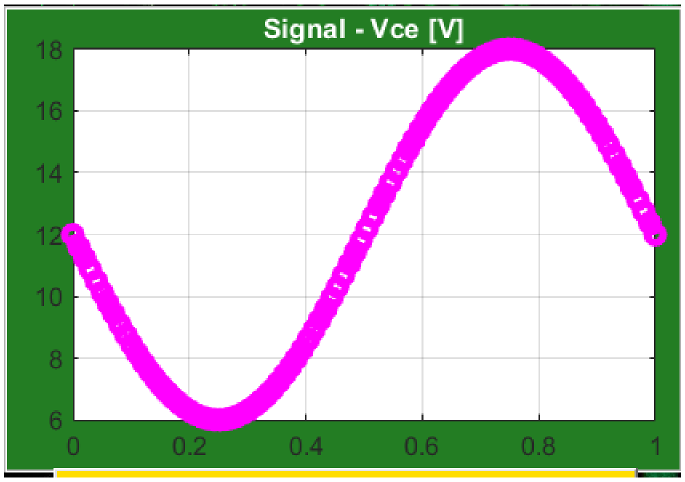
```
P = 0.87; %the power in W
Vsat = handles.edit1; %saturation voltage
Vcc = handles.edit2; %power supply
RLOAD = P*(Vcc-Vsat)^2/2 %P = Vo^2/2*RLOAD; Vo = Vcc-Vsat => RLOAD =
...

t = linspace(0,1,100);

Vi = 0.7 + 6*sin(2*pi*t);
Vo = Vi - 0.7;
Vce = Vcc - Vo;
Io = Vo/RLOAD;

figure(2)
plot(t,Vo,':cd','LineWidth',3)
title('Output Voltage - Vo [V]')
```

For the rest of the signals (those in the popupmenu) from the theoretical results I found that: Vi(t) = 0.7 + 6*sin(2*pi*t) [V]; The plot is:



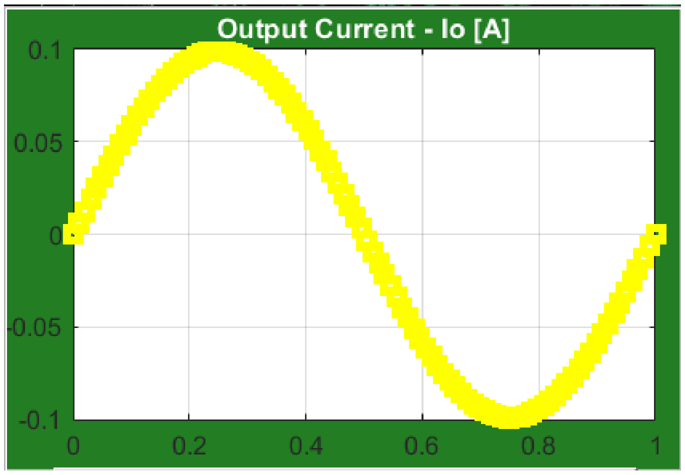
It can be seen that the amplitude of the signal, which increases somewhere close to 6.7 and for the negative part is - 6 + 0.7 = 5.3, which is correct.

For Vce(t) which is equal to: Vce (t) = Vcc - Vo depends on the value we choose for Vcc. If, for example, we choose 12V => Vce(t) = 12 - 6*sin(2*pi*t) [V].



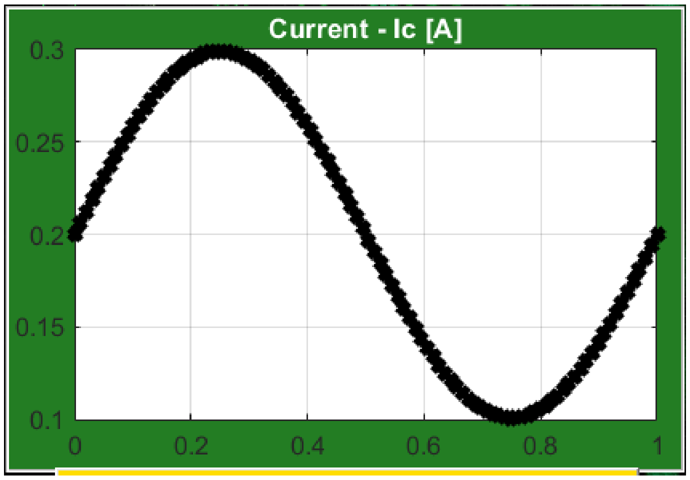
It can be seen that the max value is 6 + 12 = 18V and the min value is - 6 + 12 = 6V, which is correct.

For current Io (t), which is equal to Io (t) = Vo / RLOAD also depends on the voltages Vcc and Vsat because RLOAD depends on them. So, for example, if we choose Vsat = 0.2 V and Vcc = 12 V => Io(t) = 0.1 sin(2*pi*t) [A].



The max value for this current is 0.1 A and the min value is – 0.1 A, which is correct.

For the current Ic(t) = Ibias + Vo/RLOAD (depends on Vcc and Vsat), but for example, if Vsat = 0.2 V and Vcc = 12 V => Ic(t) = 0.2 + 0.1 sin(2*pi*t) [A].



It is clear that the result is correct, the max is 0.1 + 0.2 = 0.3 A and the min is - 0.1 + 0.2 = 0.1 A.

All these signals can be seen on theoretical results section.

The code for all this functions is:

```
P = 0.87; %the power in W
Vsat = handles.edit1; %saturation voltage
Vcc = handles.edit2; %power supply
RLOAD = P*(Vcc-Vsat)^2/2  %P = Vo^2/2*RLOAD; Vo = Vcc-Vsat => RLOAD =
...

t = linspace(0,1,100);
```

```
Vi = 0.7 + 6*sin(2*pi*t);
Vo = Vi - 0.7;
Vce = Vcc - Vo;
Io = Vo/RLOAD;
Ic = 0.2 + Io;

sel = handles.sel;
switch sel;
case 1
plot(t, Vi,'-.g*','LineWidth',3)
title('Input Voltage - Vi [V]', 'color','white');
case 2
plot(t, Vce,'--mo','LineWidth',3)
title('Signal - Vce [V]', 'color','white');
case 3
plot(t, Io,':ys','LineWidth',3)
title('Output Current - Io [A]', 'color','white');
case 4
plot(t, Ic,'-kx','LineWidth',3)
title('Current - Ic [A]', 'color','white');
end
grid on;
```

IV. Conclusions

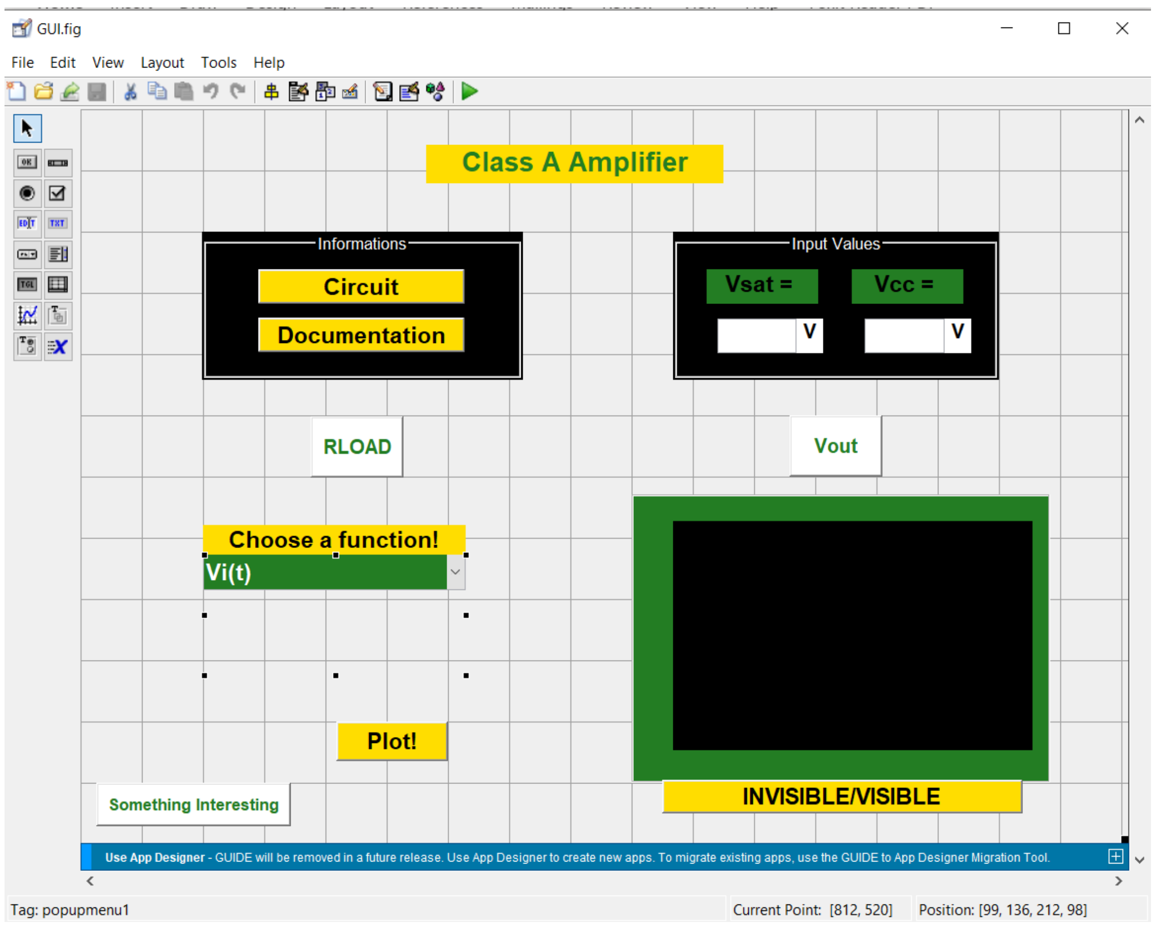
For this project “Class A amplifer” I chose to make a nice Matlab interface which shows the plots for the RLOAD, Vout, Vi, Vc, Io and Ic. The values for the 2 input voltages (Vsat and Vcc) can be changed from the edit boxes. The interface contains also action buttons which allow us to plot the signals. I want to improve my GUI and I put on it the option for make the plot on the axes visible and invisible, and I put a background image on it. The results I obtained through the GUI created meet my expectations, being correct and respecting the theoretical calculation. And just for fun, I want to show you how to manipulate the mouse pointer. The last push button (**Something Interesting**) opens another GUI (I wrote a specific function for this GUI) where there is a push button to close it, but you can not do that because the mouse pointer will go somewhere at random when you try to press it, so to close the figure, use the window ordinary x in the upper right corner.

V. References

http://en.wikipedia.org/wiki/MATLAB
https://en.wikipedia.org/wiki/Power_amplifier_classes
https://en.wikipedia.org/wiki/Amplifier
http://www.bel.utcluj.ro/dce/didactic/fec/13_power_amplifiers_ClassA.pdf
https://www.mathworks.com/matlabcentral/answers/166021-hide-image-in-gui-using-visible-on-off

VI. Appendix

Here is the GUI figure (where I edited it)



Use App Designer - GUIDE will be removed in a future release. Use App Designer to create new apps. To migrate existing apps, use the GUIDE to App Designer Migration Tool.

Tag: popupmenu1

Current Point: [812, 520]

Position: [99, 136, 212, 98]

Here is the entire code from my program:

```
function varargout = GUI(varargin)
% GUI MATLAB code for GUI.fig
%      GUI, by itself, creates a new GUI or raises the existing
%      singleton*.
%
%      H = GUI returns the handle to a new GUI or the handle to
%      the existing singleton*.
%
%      GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI.M with the given input
arguments.
%
%      GUI('Property','Value',...) creates a new GUI or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before GUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to GUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help GUI

% Last Modified by GUIDE v2.5 31-Dec-2020 11:55:55

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
```

```
% End initialization code - DO NOT EDIT


% --- Executes just before GUI is made visible.
function GUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI (see VARARGIN)

w = imread('unnamed.png');
imshow(w, 'InitialMagnification', 120);

% Choose default command line output for GUI
handles.output = hObject;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          FOR THE BACKGROUND IMAGE               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% create an axes that spans the whole gui
ah = axes('unit', 'normalized', 'position', [0 0 1 1]);
% import the background image and show it on the axes
bg = imread('background_for_gui.jpg'); imagesc(bg);
% prevent plotting over the background and turn the axis off
set(ah,'handlevisibility','off','visible','off')
% making sure the background is behind all the other uicontrols
uistack(ah, 'bottom');

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes GUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = GUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1
as a double

newstrval = get(hObject,'String');
newval = str2double(newstrval);
handles.edit1 = newval;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2
as a double

newstrval = get(hObject,'String');
newval = str2double(newstrval);
handles.edit2 = newval;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 figure ('Name','Image','NumberTitle','off');
 img=imread('circuit.png');
 image(img);
 axis off;

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

open('Documentation.docx')

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

P = 0.87; %the power in W
Vsat = handles.edit1; %saturation voltage
Vcc = handles.edit2; %power supply
Vd = linspace(Vsat,2*Vcc,50);
RLOAD = P*(Vcc-Vsat)^2/2 %P = Vo^2/2*RLOAD; Vo = Vcc-Vsat => RLOAD =
...
Id = (2*Vcc - Vd - Vsat)/RLOAD;

figure(1)
plot(Vd,Id,'-g^',...
'LineWidth',2,...
'MarkerEdgeColor','k',...
'MarkerFaceColor','y',...
'MarkerSize',5)
xlabel('Drain Voltage (V_{DS})')
ylabel('Drain Current (I_{DS})')
title("Load Line for RLOAD = " + RLOAD + " \Omega")

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
P = 0.87; %the power in W
Vsat = handles.edit1; %saturation voltage
Vcc = handles.edit2; %power supply
RLOAD = P*(Vcc-Vsat)^2/2  %P = Vo^2/2*RLOAD; Vo = Vcc-Vsat => RLOAD =
...

t = linspace(0,1,100);

Vi = 0.7 + 6*sin(2*pi*t);
Vo = Vi - 0.7;
Vce = Vcc - Vo;
Io = Vo/RLOAD;

figure(2)
plot(t,Vo,':cd','LineWidth',3)
title('Output Voltage - Vo [V]')

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1

handles.sel = get(hObject,'Value');
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
P = 0.87; %the power in W
Vsat = handles.edit1; %saturation voltage
Vcc = handles.edit2; %power supply
RLOAD = P*(Vcc-Vsat)^2/2  %P = Vo^2/2*RLOAD; Vo = Vcc-Vsat => RLOAD =
...

t = linspace(0,1,100);

Vi = 0.7 + 6*sin(2*pi*t);
Vo = Vi - 0.7;
Vce = Vcc - Vo;
Io = Vo/RLOAD;
Ic = 0.2 + Io;

sel = handles.sel;
switch sel;
case 1
plot(t, Vi,'-.g*','LineWidth',3)
title('Input Voltage - Vi [V]', 'color','white');
case 2
plot(t, Vce,'--mo','LineWidth',3)
title('Signal - Vce [V]', 'color','white');
case 3
plot(t, Io,':ys','LineWidth',3)
title('Output Current - Io [A]', 'color','white');
case 4
plot(t, Ic,'-kx','LineWidth',3)
title('Current - Ic [A]', 'color','white');
end
grid on;


% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

switch get(handles.axes1,'visible')
    case 'on'
       set(handles.axes1,'visible','off') %hide the current axes
       set(get(handles.axes1,'children'),'visible','off') %hide the
current axes contents
    case 'off'
       set(handles.axes1,'visible','on') %make visible the current
axes
       set(get(handles.axes1,'children'),'visible','on') %make visible
the current axes contents
end
```

```
% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Demonstrate how to control the mouse pointer from a GUI.
% Just for fun, show how to manipulate the mouse pointer.  To close
the
% figure, use the regular window x in the upper right corner.

S.fh = figure('units','pixels',...
              'position',[300 300 300 160],...
              'menubar','none',...
              'name','THE-FUN-GUI',...
              'numbertitle','off',...
              'resize','off');
S.UN = get(0,'units');  % We need to temporarily change this.
S.pb = uicontrol('style','push',...
                 'units','pix',...
                 'position',[20 30 260 100],...
                 'string', 'Push Me To Close Figure',...
                 'fontsize',12,'fontweight','bold',...
                 'callback',{@pb_call,S});
set(S.fh,'WindowButtonMotionFcn',{@fh_wbmf,S})

function [] = fh_wbmf(varargin)
S = varargin{3};  % Get the structure.
set(0,'units','normalized') % Temporary, see below.
% Move the mouse pointer to a random position.
set(0, 'PointerLocation', [rand rand])
set(0,'units',S.UN)  % Undo change to user's system.  Good courtesy.
```