

Performance Measure Calculation of Power Transformer for Exact and Approximate Models using MATLAB

Khalid Al-Olimat, Pete Jankovsky, Matt Valerio
Department of Electrical & Computer Engineering and Computer Science
Ohio Northern University

Abstract

This paper presents a MATLAB program that utilizes a Graphical User Interface (GUI) to calculate the electrical quantities and the efficiency of a power transformer. The GUI is designed in such a way to allow the user to enter the resistances and reactances of the primary and the secondary sides, the resistance and reactance that represent the core, the rating of the transformer and the transformer turn ratio. This program performs the calculations with respect to the popular exact and approximate models of transformers for both cases, referred to primary and to secondary. This allows the user to compare the obtained results and notice the effects of approximation on the values. Also, students can check the validity of their solutions of homework problems involving transformers. The program can be modified to perform calculations for magnetic circuits, ac and dc rotating machines and other engineering applications.

Introduction

Electrical engineering students at Ohio Northern University are required to take the Energy Conversion course. This course covers and exposes students to the topics of transformers and electric machines. The transformers and the electric machines are represented by their equivalent circuits and are then analyzed using basic circuit laws and theorems. This method is instructive and more helpful to students' understanding.

But in the learning process the student should not be a passive recipient of knowledge watching things happen, he should be a participant. The quantity of knowledge he obtains in any given area is of secondary importance. The primary importance is the method he develops for acquiring knowledge. The student should be made aware of why a certain thing should be done and what value it has in the overall picture. The course and its associated laboratory should help the student to generate an attitude and an inquiring frame of mind that will lead him to creative activity of his own.

This paper is a result of one of the assigned projects to students in the course where they have demonstrated that they are active participants. A MATLAB program that utilizes a GUI is written to calculate the electrical quantities and the efficiency of a power transformer. The GUI is designed in such a way to allow the user to enter the resistances and reactances of the primary and secondary sides, the resistance and reactance that represent the core, the power rating and the turn ratio of the transformer. This program performs the calculations with respect to the exact and approximate models of transformers for both referred to primary and to secondary. This allows the user to compare the obtained results and notice the effects of approximation on the values. Also, students can check the validity of their solutions of homework problems involving transformers. The program can be modified to perform calculations for magnetic circuits, ac and dc rotating machines and other engineering applications.

Transformer Overview

The major function of power transformers is to transfer electric energy from one circuit (primary) to another circuit (secondary) at a higher or a lower voltage. When used for the purpose of raising the voltage they are referred to as step-up transformers, and for lowering the voltage as step-down transformers. Transformers can be modeled as either ideal transformers or actual transformers. In the case of ideal transformer, many assumptions have to be considered. Some of these assumptions are the magnetic material is infinitely permeable and does not saturate, negligible core loss, all flux is confined within the magnetic core and each winding has no resistance. While in the case of actual transformer, these assumptions are no longer applicable.

Exact Model of Transformer

The exact model of a transformer is shown in Figure 1. The parameters involved are the primary terminal voltage, \tilde{V}_p , secondary terminal voltage, \tilde{V}_s , primary current, \tilde{I}_p , secondary current, \tilde{I}_s , excitation current, \tilde{I}_ϕ , primary resistance, R_p , secondary resistance, R_s , primary leakage reactance, X_p , secondary leakage reactance, X_s , core resistance, R_c , and magnetizing reactance, X_M . Usually it is preferred to reflect the secondary side of the transformer to its primary side or the primary side to its secondary side. This will simplify the calculation involved and results will be obtained in a faster manner.

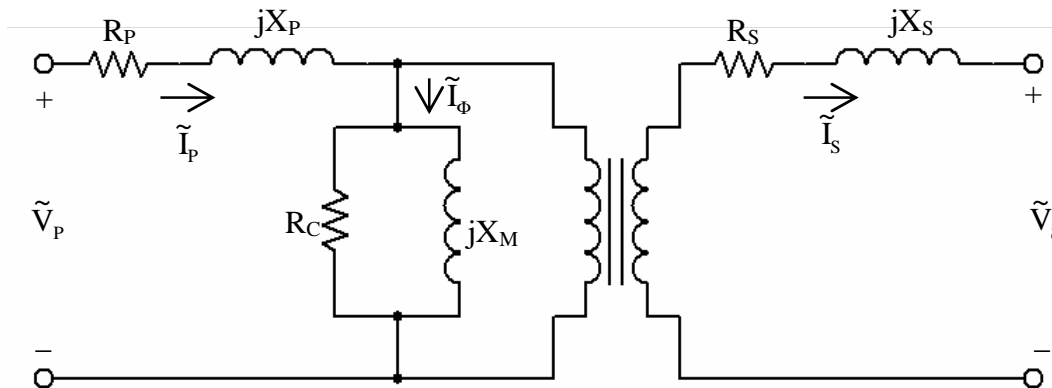


Figure 1 – Transformer Exact Model

The transformer exact model when it is referred to its primary is shown in Figure 2. Figure 3 shows the transformer exact model when it is referred to its secondary.

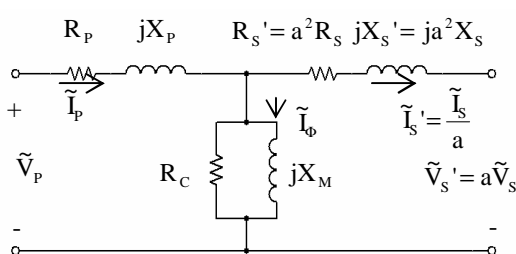


Figure 2

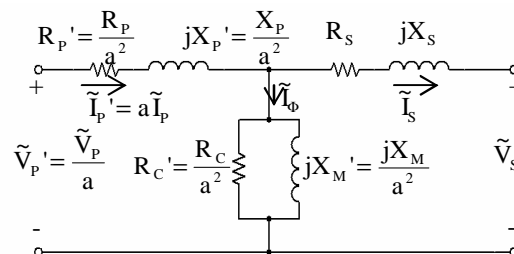


Figure 3

Approximate Models of Transformer

In engineering analysis involving the transformer as a circuit element, it is customary to adopt one of several approximate forms of the equivalent circuit of Figure 1 rather than the full circuit. The approximations chosen in a particular case depend largely on physical reasoning based on orders of magnitude of the neglected quantities. Figure 4 shows an approximate model that the shunt branch is moved to primary side and the transformer is referred to its primary. The approximate model when the shunt branch is moved to secondary side and the transformer is referred to its primary is shown in Figure 5. Figure 6 shows the model when the shunt branch is moved to primary and the transformer is referred to its secondary while Figure 7 shows the same case with the shunt branch moved to secondary.

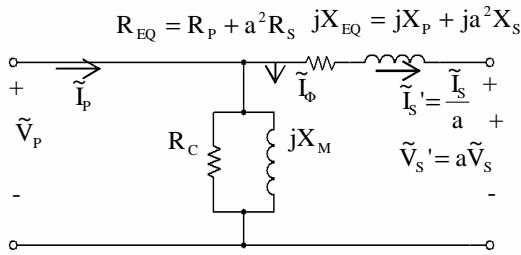


Figure 4

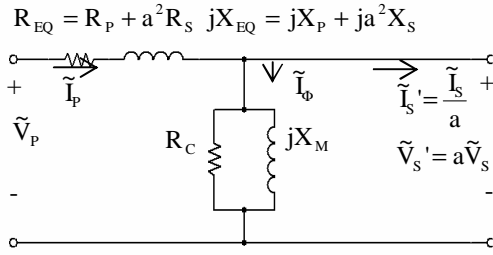


Figure 5

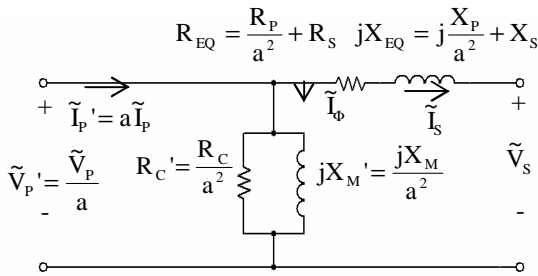


Figure 6

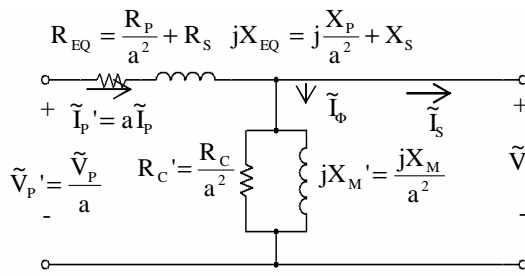


Figure 7

Also, we can use approximate models with negligible shunt branch. Figure 8 shows an approximate model for this case and the transformer is referred to its primary. The same case when transformer is referred to its secondary is shown in Figure 9.

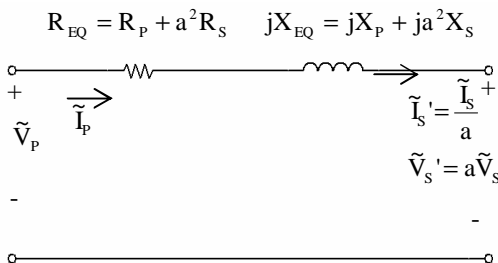


Figure 8

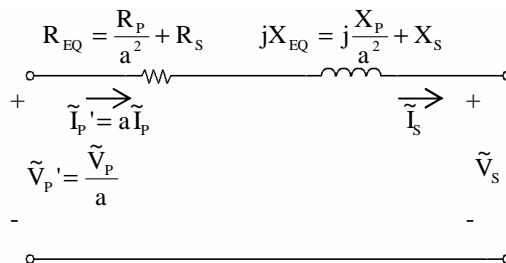


Figure 9

More approximation can be implemented on transformers through the elimination of the resistances of primary and secondary windings. Figure 10 shows this case when transformer is referred to its primary and Figure 11 shows when it is referred to its secondary.

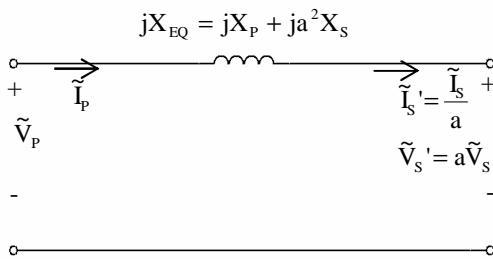


Figure 10

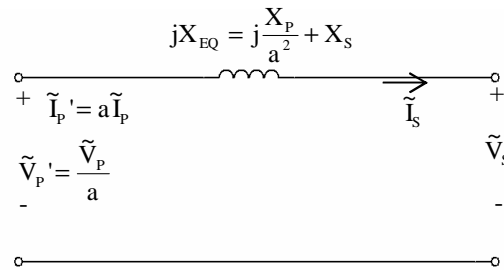


Figure 11

MATLAB Implementation

Computer simulation provides a simplification of reality due to its role in the design, analysis and evaluation of systems. A variety of software tools is available to simulate engineering applications. The most popular software is MATLAB. One difficulty students have in programming MATLAB scripts is in keeping the code organized. Developing a flow chart of the pseudo-code steps helps the students to visualize the operation of the script, which is useful for planning development and for debugging specific problems. The flow chart for the transformer model simulation is shown in Figure 12. The segments above the dashed line in the figure deal with the GUI implementation, and the segments below the dashed line deal with the calculation and result output.

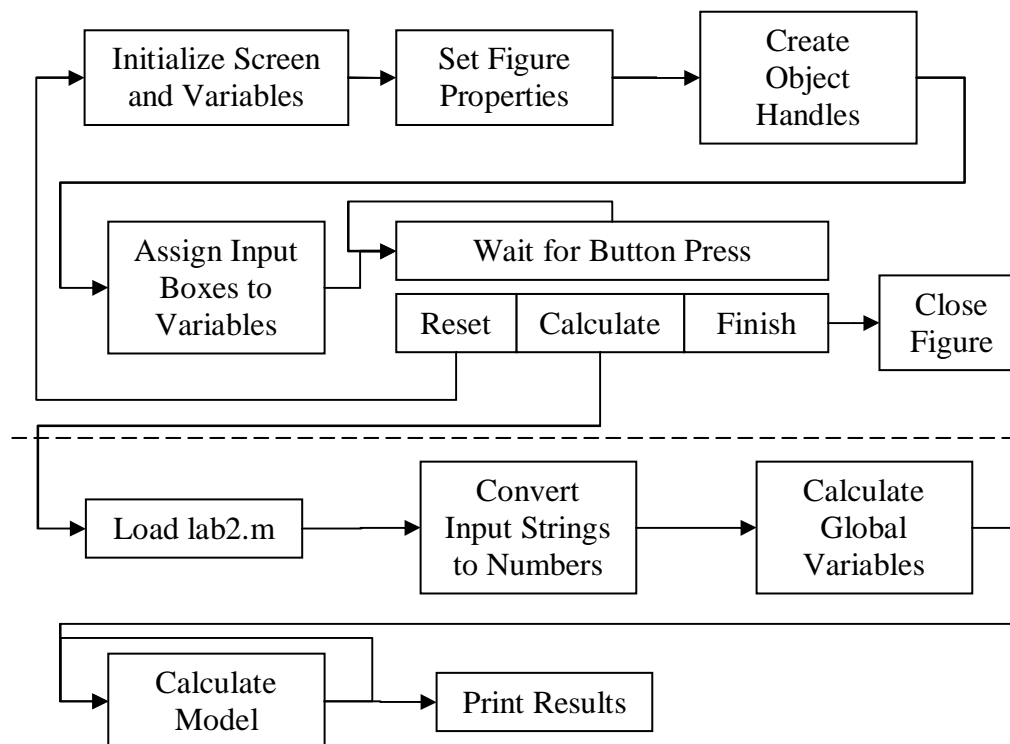


Figure 12 – Flow chart of code organization

The development of a MATLAB GUI is not straightforward for electrical engineers with little programming background. A brief discussion follows of a strategy for simple GUI implementation.

The GUI shown in Figure 13 is composed of a MATLAB *figure* containing various *handles*. Multiple *figures* can exist in a MATLAB program, so objects must specify which figure they belong to—this is designated by the *parent* property. A simple syntax for creating objects of different types is the `uicontrol` function. The ‘Style’ attribute designates which type of object will be created: A ‘text’ box contains non-editable text for display, an ‘edit’ box allows for user input of text, a ‘pushbutton’ executes a command when depressed, a ‘popup’ menu allows for input constrained by pre-defined values.

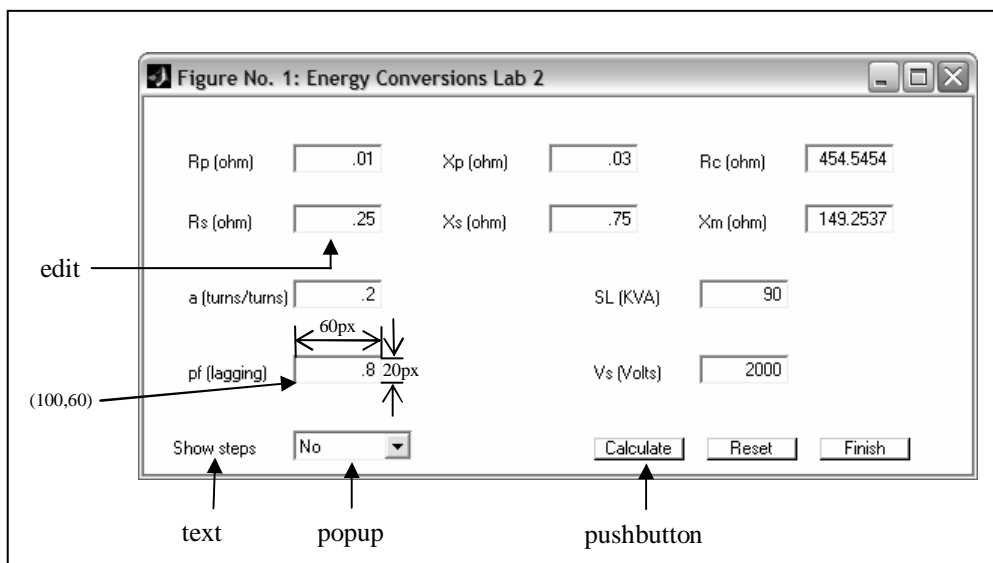


Figure 13 – Screenshot of GUI

A sample initialization of the pf input box is shown below in Figure 14. The properties of the object are set in the initialization of the object. The ‘string’ attributes contain the text that will be displayed—specifying a string value for an edit box allows for the programmer to specify default input values. The ‘dimension’ argument is a vector respectively describing the horizontal position, vertical position, horizontal size, and vertical size—expressed in units of screen pixels.

```
%input the power to variable Hpf
Hpf = uicontrol('Style', 'edit',...      %edit box
               'String', '.8',...       %default value
               'HorizontalAlignment', 'right',... %align numbers to right
               'Parent', 1,...          %child of figure 1
               'Position', [100 60 60 20]); %located at (100,60)
                                       %with width 60, height 20
```

Figure 14 – Initialization of power factor edit object

The GUI is most efficiently and effortlessly implemented if the desired layout is first drafted on paper, so that objects can be easily positioned before coding the position vectors. The initialization of the GUI input window is shown below in Figure 15. The window is spawned with the first command. Because properties have not been specified with the declaration of the figure, the `set()` command must be used to modify the attributes. The syntax is explained in the comments.

```
figure(1) %create the input figure
%set the attributes for the figure
set(1,'Color','white',... %figure background is white
     'MenuBar','none',... %don't display the menu bar
     'Position',[10 scrsz(4)-300 550 250],... %slightly offset the
                                     %figure from the top-left
     'Name','Energy Conversions Lab 2'); %set the window title
```

Figure 15 – Initialization of the figure window

Default values for newly created figure objects can also be specified, as shown below in Figure 16. These default values do not need to be specified when creating new objects, but the programmer can customize those attributes if desired. Specified values will take precedence over default values.

```
%set default attributes for the figure objects
set(0,'DefaultUicontrolBackgroundColor','white');
set(0,'DefaultUicontrolHorizontalAlignment','left');
```

Figure 16 – Default object properties

In addition to providing a suite of user controls, MATLAB extends many useful flow control constructs for writing efficient routines. Program flow does not necessarily need to move from the first line of code sequentially to the last. For example, one may use `if`, `if-else`, and `switch` statements to control the flow of the code. For example, an expression may be evaluated using an `if` statement, and if the condition is true, a group of statements is executed, otherwise it is skipped. The user defined `debugprint` function in Figure 17 illustrates this – if the value of `debug` is equal to 1, then the string `DEBUG:` is printed to the console, followed by the text value passed in `string`.

```
function debugprint(debug,string)
    if debug == 1
        disp(['DEBUG: ' string ]);
    end
```

Figure 17 – Example of an `if` statement

A `switch` statement may be used to execute a group of statements when a condition takes on a range of possible values. For example, depending on the model number being calculated, the

section of code would look like Figure 18. If `num` is equal to 1, then the code to calculate the Exact Model is executed, and none of the code in the other case statements is executed.

```
switch num
    case 1
        disp('Calculating [Exact Model - Referred to primary]...');
        % calculation code here
    case 2
        disp('Calculating [Exact Model - Referred to secondary]...');
        % calculation code here
    case 3
        % calculation code here
    % other case statements here
end
```

Figure 18 – Example of a switch statement

To increase code efficiency and eliminate unnecessary typing, functions may be created to take certain input values, perform some calculation, and return other output values. For routine actions that are performed multiple times in a program, functions serve an important purpose. Functions may be easily implemented in MATLAB by making a M-file with the function name (e.g. `name_of_function.m`) and typing a function declaration on the first line of the file. For example, a file named `complexp.m` contains two lines as shown in Figure 19.

```
function c = complexp(r,theta)
    c = r * exp(j*theta);
```

Figure 19 – Function example

This file creates a function called `complexp` that accepts two input arguments, `r` and `theta`. A return value, `c`, is specified, and is programmed to return a complex number given the magnitude, `r`, and the angle, `theta`.

In any other M-file, this function may be called, and the return value may be used in another calculation. For example, an expression such as `z = complexp(10,30)+complexp(20,-20)` could be used to add two vectors and store the result in a variable called `z`.

Each function must be stored in a separate M-file with the function name in the filename so that MATLAB can locate it when necessary.

Output Formatting

MATLAB provides a select number of output routines that are useful for displaying numerical and string data on the console. The `disp` function accepts one argument and displays its value on the console. It is useful for displaying status messages, for example, but does not display numbers in the best format.

Formatted printing may be accomplished by using `fprintf` to display to the console and `sprintf` to return a formatted string. The `fprintf` function accepts a variable number of arguments, the first being a format string, and the others being values to be placed in the format string. For example, if a string and two numbers needed to be displayed on the console, the following code (Figure 20) could be used:

```
s = 'Textbook Title';
a = 1;
b = 2;
fprintf('%s - Page %d of %d',s, a, b);
```

Figure 20 – fprintf example

The format string, `'%s - Page %d of %d'` contains three argument delimiters – one `%s` to insert a string, and two `%d` to insert two numbers. The arguments, `s`, `a`, and `b` contain the values that will be inserted into the format string to produce the final output string. For this example, the string `'Textbook Title - Page 1 of 2'` will be printed to the console. Many other options may be used in the format string, such as specifying the number of decimal places before and after the decimal point, whether to display in scientific or engineering notation, and others. The MATLAB help files for `fprintf` and `sprintf` document all options and show many useful examples.

The command used to generate one line is shown in Figure 21 where `A` is a two-dimensional array containing each calculated value for each model number. A formatted output of all 10 transformer models is shown in Figure 22.

```
fprintf('%2d (%5.3f @ %+3.3f) (%5.3f @ %+3.3f) ...
(%5.3f @ %+3.3f) (%5.3f @ %+3.3f) (%5.3f @ %+3.3f) ...
%5.3f %5.3f %3.3f%%',i, ...
abs(A(i,1)),angle(A(i,1))*180/pi, ...
abs(A(i,2)),angle(A(i,2))*180/pi, ...
abs(A(i,3)),angle(A(i,3))*180/pi, ...
abs(A(i,4)),angle(A(i,4))*180/pi, ...
abs(A(i,5)),angle(A(i,5))*180/pi, ...
A(i,6), A(i,7), A(i,8)*100 );
```

Figure 21 – Code used

```
*****
                Calculation Results                *
      (Note: All angles are in degrees)            *
*****
# Vp (V)      Vs (V)      Ip (A)      Is (A)      Iphi (A)      Pin (W)      Pout (W)      Efficiency
-----
1 (411.780 @ +1.127) (2000.000 @ +0.000) (224.999 @ -36.869) (45.000 @ -36.870) (0.003 @ +72.394) 73012.858 72000.000 98.613%
2 (411.780 @ +1.127) (2000.000 @ +0.000) (224.999 @ -36.869) (45.000 @ -36.870) (0.001 @ +72.394) 73012.858 72000.000 98.613%
3 (400.000 @ -0.000) (2000.000 @ +0.000) (224.999 @ -36.869) (45.000 @ -36.870) (0.003 @ +71.822) 72000.392 72000.000 99.999%
4 (411.780 @ +1.127) (2000.000 @ +0.000) (224.999 @ -36.869) (45.000 @ -36.870) (0.003 @ +71.822) 73012.844 72000.000 98.613%
5 (411.780 @ +1.127) (2000.000 @ +0.000) (224.999 @ -36.869) (45.000 @ -36.870) (0.001 @ +72.949) 73012.873 72000.000 98.613%
6 (411.780 @ +1.127) (2000.000 @ +0.000) (224.999 @ -36.869) (45.000 @ -36.870) (0.001 @ +71.822) 73012.844 72000.000 98.613%
7 (411.780 @ +1.127) (2000.000 @ +0.000) (225.000 @ -36.870) (45.000 @ -36.870) (0.000 @ +0.000) 73012.500 72000.000 98.613%
8 (411.780 @ +1.127) (2000.000 @ +0.000) (225.000 @ -36.870) (45.000 @ -36.870) (0.000 @ +0.000) 73012.500 72000.000 98.613%
9 (410.880 @ -1.130) (2000.000 @ +0.000) (225.000 @ -36.870) (45.000 @ -36.870) (0.000 @ +0.000) 75037.500 72000.000 95.952%
10 (410.880 @ -1.130) (2000.000 @ +0.000) (225.000 @ -36.870) (45.000 @ -36.870) (0.000 @ +0.000) 75037.500 72000.000 95.952%
```

Figure 22 – Formatted output for all 10 transformer models.

Conclusion

This paper has presented a result of an assigned project in the energy conversion course at Ohio Northern University. It has shown all steps of writing a MATLAB code to calculate the electric quantities and the efficiency of a given transformer. This code can be modified to be suitable for other engineering applications.

Appendix A: Contents of "lab2.m"

```
% Energy Conversion Lab 2
% Get the user input from the GUI (use the get function to extract the
% string values from the handle structure, then convert the strings to
% numbers

pf = str2num(get(Hpf,'String'));
Vs = str2num(get(HVs,'String'));
a = str2num(get(Ha,'String'));
PowerLoad = str2num(get(HSL,'String'));
PowerLoad = PowerLoad *10^3;
Xs = str2num(get(HXs,'String'));
Xm = str2num(get(HXm,'String'));
Rs = str2num(get(HRs,'String'));
Rp = str2num(get(HRp,'String'));
Xp = str2num(get(HXp,'String'));
Rc = str2num(get(HRc,'String'));
ReferredTo=get(HReferredTo,'Value');

%General Calculations
Gc=1/Rc;
Bm=1/Xm;

[PowerLoad(1),PowerLoad(2)]=pol2cart(-acos(pf),PowerLoad);
PowerLoad=complex(PowerLoad(1),PowerLoad(2));

Is=PowerLoad/Vs;
IsPrime=Is/a;
VsPrime=a*Vs;

XsPrime=a^2*Xs;
XpPrime=Xp/a^2;
RsPrime=a^2*Rs;
RpPrime=Rp/a^2;

GcPrime=a^2/Rc;
BmPrime=a^2/Xm;

switch ReferredTo
    case 1 %Referred to Primary
        Xeq=Xp+XsPrime;
        Req=Rp+RsPrime;

        %Model #7
        Iphi7=0;
        Vp7=VsPrime+j*IsPrime*Xeq;
        Ip7=IsPrime;

        %Model #5
        Iphi5=0;
        Vp5=VsPrime+IsPrime*(Req+j*Xeq);
        Ip5=IsPrime;

        %Model #1
        Vp1=VsPrime+IsPrime*(Req+j*Xeq);
        Iphi1=(Gc-j*Bm)*VsPrime;
        Ip1=IsPrime+Iphi1;

        %Model #2
        Iphi2=(Gc-j*Bm)*VsPrime;
```

```

Ip2=IsPrime+Iphi2;
Vp2=VsPrime+Ip2*(Req+j*Xeq);

%Exact ModelP
VeP= VsPrime+IsPrime*(RsPrime+j*XsPrime);
IphiEP=VeP*(Gc-j*Bm);
IpEP=IsPrime+IphiEP;
VpEP=VeP+IpEP*(Rp+j*Xp);

case 2 %Referred to Secondary
Xeq=XpPrime+Xs;
Req=RpPrime+Rs;

%Model #8
Iphi8=0;
Ip8=Is/a;
VpPrime=Vs+Is*Xeq*j;
Vp8=a*VpPrime;

%Model #6
Iphi6=0;
Ip6=Is/a;
VpPrime=Vs+Is*(Req+j*Xeq);
Vp6=a*VpPrime;

%Model #4
Iphi4=Vs*(GcPrime-j*BmPrime);
IpPrime=Is+Iphi4;
Ip4=IpPrime/a;
VpPrime=Vs+IpPrime*(Req+j*Xeq);
Vp4=a*VpPrime;

%Model #3
VpPrime=Vs+Is*(Req+j*Xeq);
Iphi3=VpPrime*(GcPrime-j*BmPrime);
IpPrime=Is+Iphi3;
Ip3=IpPrime/a;
Vp3=a*VpPrime;

%Exact Models
VeS=Vs+Is*(Rs+j*Xeq);
IphiES=VeS*(GcPrime-j*BmPrime);
IpPrime=Is+IphiES;
VpPrime=VeS+IpPrime*(RpPrime+j*XpPrime);
IpES=IpPrime/a;
VpES=a*VpPrime;
end

switch ReferredTo
case 1 %Referred to Primary
Model=[1;2;5;7;0];
Vp=cat(1,Vp1,Vp2,Vp5,Vp7,VpEP); %put all the voltages together so conversion
is easier
Ip=cat(1,Ip1,Ip2,Ip5,Ip7,IpEP);
Iphi=cat(1,Iphi1,Iphi2,Iphi5,Iphi7,IphiEP);
Vs=cat(1,Vs,Vs,Vs,Vs,Vs);
Is=cat(1,Is,Is,Is,Is,Is);

Pp=real(Vp.*conj(Ip));
Ps=real(Vs.*conj(Is));
eff=Ps./Pp*100;

```

```

Vp(:,2)=0; %expand the matrix
Ip(:,2)=0;

i=1;
while (i<6) %step through the matrices
    [Vp(i,2),Vp(i,1)]=cart2pol(real(Vp(i,1)),imag(Vp(i,1))); %convert the
rectangular coordinates to polar
    Vp(i,2)=Vp(i,2)*180/pi; %convert radians to decimal
    [Ip(i,2),Ip(i,1)]=cart2pol(real(Ip(i,1)),imag(Ip(i,1)));
    Ip(i,2)=Ip(i,2)*180/pi;
    [Vs(i,2),Vs(i,1)]=cart2pol(real(Vs(i,1)),imag(Vs(i,1))); %convert the
rectangular coordinates to polar
    Vs(i,2)=Vs(i,2)*180/pi; %convert radians to decimal
    [Is(i,2),Is(i,1)]=cart2pol(real(Is(i,1)),imag(Is(i,1)));
    Is(i,2)=Is(i,2)*180/pi;
    [Iphi(i,2),Iphi(i,1)]=cart2pol(real(Iphi(i,1)),imag(Iphi(i,1)));
    Iphi(i,2)=Iphi(i,2)*180/pi;
    i=i+1;
end

Results=(cat(2,Model,Vp,Ip,Vs,Is,Iphi,Pp,Ps,eff)); %merge the results into one
table

format short g;
fprintf('\n\rModels 1,2,5,7, and Exact as referred to Primary follow:\n\r')
fprintf('columns: Model#   |Vp|   <Vp   |Ip|   <Ip   |Vs|   <Vs   |Is|   <Is
|Iphi|  <Iphi  Pp   Ps   Efficiency\n')
disp(Results)

case 2 %Referred to Secondary
Model=[3;4;6;8;0];
Vp=cat(1,Vp3,Vp4,Vp6,Vp8,VpES); %put all the voltages together so conversion
is easier
Ip=cat(1,Ip3,Ip4,Ip6,Ip8,IpES);
Iphi=cat(1,Iphi3,Iphi4,Iphi6,Iphi8,IphiES);
Vs=cat(1,Vs,Vs,Vs,Vs,Vs);
Is=cat(1,Is,Is,Is,Is,Is);

Pp=real(Vp.*conj(Ip));
Ps=real(Vs.*conj(Is));
eff=Ps./Pp*100;

Vp(:,2)=0; %expand the matrix
Ip(:,2)=0;

i=1;
while (i<6) %step through the matrices
    [Vp(i,2),Vp(i,1)]=cart2pol(real(Vp(i,1)),imag(Vp(i,1))); %convert the
rectangular coordinates to polar
    Vp(i,2)=Vp(i,2)*180/pi; %convert radians to decimal
    [Ip(i,2),Ip(i,1)]=cart2pol(real(Ip(i,1)),imag(Ip(i,1)));
    Ip(i,2)=Ip(i,2)*180/pi;
    [Vs(i,2),Vs(i,1)]=cart2pol(real(Vs(i,1)),imag(Vs(i,1))); %convert the
rectangular coordinates to polar
    Vs(i,2)=Vs(i,2)*180/pi; %convert radians to decimal
    [Is(i,2),Is(i,1)]=cart2pol(real(Is(i,1)),imag(Is(i,1)));
    Is(i,2)=Is(i,2)*180/pi;
    [Iphi(i,2),Iphi(i,1)]=cart2pol(real(Iphi(i,1)),imag(Iphi(i,1)));
    Iphi(i,2)=Iphi(i,2)*180/pi;

    i=i+1;
end

```

```

        Results=(cat(2,Model,Vp,Ip,Vs,Is,Iphi,Pp,Ps,eff)); %merge the results into one
table

        format short g;
        fprintf('\n\rModels 3,4,6,8, and Exact as referred to Secondary follow:\n\r')
        fprintf('columns: Model#   |Vp|   <Vp   |Ip|   <Ip   |Vs|   <Vs   |Is|   <Is
|Iphi|   <Iphi   Pp   Ps   Efficiency\n')
        disp(Results)

end

```

Appendix B: Contents of “mygui.m”

```

% Energy Conversion Lab 2
% GUI code as each function is edited, the code from lab2.m is executed

clear;

%Get the screen size (so the program will be displayed nicely :)
scrsz = get(0,'ScreenSize');

figure(1)
%set the attributes for the figure
set(1,'Color','white',...
    'MenuBar','none',...
    'Position',[10 scrsz(4)-300 550 250],...
    'Name','Energy Conversions Lab 2');

%set default attributes for the figure objects
set(0,'DefaultUicontrolBackgroundColor','white');
set(0,'DefaultUicontrolHorizontalAlignment','left');

%input the refer to in a pull-down menu
HReferredTo = uicontrol('Style','popup',...
    'String','Primary|Secondary',...
    'Parent',1,...
    'Position',[100 10 80 20]);

uicontrol('Style','text',...
    'String','Referred to',...
    'Parent',1,...
    'Position',[30 10 60 15]);

uicontrol('Style','pushbutton',...
    'String','Calculate',...
    'Parent',1,...
    'Position',[300 10 60 15],...
    'Callback','lab2');

uicontrol('Style','pushbutton',...
    'String','Reset',...
    'Parent',1,...
    'Position',[375 10 60 15],...
    'Callback','mygui');

uicontrol('Style','pushbutton',...
    'String','Finish',...
    'Parent',1,...
    'Position',[450 10 60 15],...
    'Callback','close all');

```

```

uicontrol('Style', 'text',...
    'String', 'pf (lagging)',...
    'Parent', 1,...
    'Position', [30 60 60 15]);

%input the power factor in an edit box
Hpf = uicontrol('Style', 'edit',...
    'String', '.8',...
    'HorizontalAlignment', 'right',...
    'Parent', 1,...
    'Position', [100 60 60 20]);

uicontrol('Style', 'text',...
    'String', 'Vs (Volts)',...
    'Parent', 1,...
    'Position', [300 60 60 15]);

HVVs = uicontrol('Style', 'edit',...
    'String', '2000',...
    'Parent', 1,...
    'HorizontalAlignment', 'right',...
    'Position', [370 60 60 20]);

uicontrol('Style', 'text',...
    'String', 'a (turns/turns)',...
    'Parent', 1,...
    'Position', [30 110 80 15]);

Ha = uicontrol('Style', 'edit',...
    'String', '.2',...
    'HorizontalAlignment', 'right',...
    'Parent', 1,...
    'Position', [100 110 60 20]);

uicontrol('Style', 'text',...
    'String', 'SL (KVA)',...
    'Parent', 1,...
    'Position', [300 110 60 15]);

HSL = uicontrol('Style', 'edit',...
    'String', '90',...
    'HorizontalAlignment', 'right',...
    'Parent', 1,...
    'Position', [370 110 60 20]);

uicontrol('Style', 'text',...
    'String', 'Rs (ohm)',...
    'Parent', 1,...
    'Position', [30 160 60 15]);

HRs = uicontrol('Style', 'edit',...
    'String', '.25',...
    'HorizontalAlignment', 'right',...
    'Parent', 1,...
    'Position', [100 160 60 20]);

uicontrol('Style', 'text',...
    'String', 'Xs (ohm)',...
    'Parent', 1,...
    'Position', [200 160 60 15]);

HXs = uicontrol('Style', 'edit',...

```

```

    'String', '.75',...
    'HorizontalAlignment', 'right',...
    'Parent', 1,...
    'Position', [270 160 60 20]);

uicontrol('Style', 'text',...
    'String', 'Xm (ohm)',...
    'Parent', 1,...
    'Position', [370 160 60 15]);

HXm = uicontrol('Style', 'edit',...
    'String', '149.253731343',...
    'HorizontalAlignment', 'right',...
    'Parent', 1,...
    'Position', [440 160 60 20]);

uicontrol('Style', 'text',...
    'String', 'Rp (ohm)',...
    'Parent', 1,...
    'Position', [30 200 60 15]);

HRp = uicontrol('Style', 'edit',...
    'String', '.01',...
    'Parent', 1,...
    'HorizontalAlignment', 'right',...
    'Position', [100 200 60 20]);

uicontrol('Style', 'text',...
    'String', 'Xp (ohm)',...
    'Parent', 1,...
    'Position', [200 200 60 15]);

HXp = uicontrol('Style', 'edit',...
    'String', '.03',...
    'HorizontalAlignment', 'right',...
    'Parent', 1,...
    'Position', [270 200 60 20]);

uicontrol('Style', 'text',...
    'String', 'Rc (ohm)',...
    'Parent', 1,...
    'Position', [370 200 60 15]);

HRC = uicontrol('Style', 'edit',...
    'String', '454.54545454545',...
    'HorizontalAlignment', 'right',...
    'Parent', 1,...
    'Position', [440 200 60 20]);

```