Bangladesh University of Engineering and Technology

Course No: EEE 212

Software Project Report

"Analysis of Transformer & Induction Motor Using MATLAB AppDesigner"

Prepared for:

Dr. Nahid-Al-Masood
Associate Professor
Department of EEE,BUET

Shaimur Salehin Akash
Department of EEE,BUET

Prepared by:

1906186 - Raihan Amin Rana
1906187 - Ahmad Saqlain Monsur

Section: C2
Department of Electrical & Electronics Engineering

Table of Contents

.

List of Illustrations

Introduction

Transformer and induction motor are two significant electrical machine of today's world.Using Matlab we can easily analyze various factors in the circuits of these two machines.In our project we will try to solve various mathematical problems of transformer and induction motor by using Matlab.Firstly we will find the equivalent circuit referred to high voltage side and low voltage side.In that case we use the value of short circuit test at high voltage side and open circuit voltage at low voltage side.We will also calculate voltage regulation,phase current,base value and per unit system value of a three phase transformer.There is different relationship between phase voltage and line voltage in the four types of connection of a transformer.For example in wye-delta connection,the primary line voltage is equal to root three times of the primary phase voltage whereas the secondary line voltage is equal to the secondary phase voltage.So we have shown all the four types of configuration in our project.Then we will find out various quantity of an autotransformer.We will show power rating advantage of an autotransformer.We also showed different important curves of an induction motor.We showed graphical development of an induction motor torque-speed characteristic in our project.The plot of rotor current versus speed for an induction motor,the plot of net magnetic field versus speed for the motor and the plot of rotor power factor versus speed for the motor combinely give the torque-speed characteristic curve of an induction motor.Then we showed different region of torque-speed characteristic curve of an induction motor.One can easily see braking region,motor region and generation region from our curve and thus know how to operate an induction motor according to his need.We also provided various ways of controlling the speed of an induction motor in the graph.These things can be clearly understood from our project.Finally we have shown all of the mentioned calculations in the appdesigner window of MATLAB

Theory

2.1. Short Circuit Test and Open Circuit Test:

The open-circuit test and the short-circuit test are performed on a transformer to determine the circuit parameters,efficiency and the voltage regulation without actual loaded of the transformer.In open circuit test the high voltage side of the transformer is left open i.e. the open circuit test is to be performed on the low-voltage side of the transformer.The open-circuit test is conducted to determine the core or iron losses and the no-load circuit parameters R_c and X_M of the transformer.The open circuit test is always performed on the low-voltage side of the transformer.Because if it is performed on the high voltage side,the no-load current would be inconveniently small and the applied voltage would be inconveniently large.In the short circuit test the low-voltage winding is shorted by a thick conductor.A voltmeter (V), an ammeter (A) and a wattmeter (W) are connected on the high-voltage side (primary) of the transformer.The short-circuit test is always performed on high-voltage side i.e. the low voltage winding is always short-circuited.If the test is performed on the low-voltage winding, then, the high voltage winding being short-circuited and hence, the voltage will be inconveniently low and the current would be inconveniently high.

The Exact Equivalent Circuit of a Real Transformer:

In an equivalent circuit we need to take into account all the major imperfections of real transformers. Each major imperfection is considered in turn and its effect is included in the transformer model. The easiest effect to model is the copper losses.Copper losses are resistive losses in the primary and secondary windings of the transformer core.They are modeled by placing a resistor Rp in the primary circuit of the transformer and a resistor Rs in the secondary circuit.

The leakage in the primary windings produces a voltage e_{LP} given by

$$e_{LP}(t) = NP(\frac{d\phi_{LP}}{dt})$$

The leakage in the secondary windings produces a voltage e_{LS} given by

$$e_{LS}(t) = NP(\frac{d\phi_{LS}}{dt})$$

Since much of the leakage flux path is through air and air has a constant reluctance much higher than the core reluctance,the flux, φ_{LP} is directly proportional to the primary circuit current i_p and the flux φ_{LS} is directly proportional to the secondary current i_s

$$\varphi_{LP} = PN_p i_p$$

$$\varphi_{LS} = PN_p i_p$$

Where,

P=Permeance of flux path

N_P=Number of turns on primary coil

N_S=Number of turns on secondary coil

We get, $e_{LP}(t) = N_p^2 P(\frac{di_{LP}}{dt})$

$$e_{LS}(t) = N^2 P(\frac{di_{LP}}{dt})$$

Again, $e_{LP}(t) = L_p \frac{di_p}{dt}$

$$e_{Ls}(t) = L_s \frac{di_s}{dt}$$

Therefore,the leakage flux will be modeled by primary and secondary inductors.

Where,

$L_P = N_p^2 P$, is the leakage inductance of the primary coil

$L_S = N_S^2 P$, is the leakage inductance of the secondary coil.

The magnetization current, i_m is a current proportional (in the unsaturated region) to the voltage applied to the core and lagging the applied voltage by 90 degree, so it can be modeled by a reactance X_M connected across the primary voltage source.The core-loss current, i_{h+e} is a current proportional to the voltage applied to the core that is in phase with the applied voltage, so it can be modeled by a resistance R_c connected across the primary voltage source.Both these currents are really nonlinear, so the inductance X_M and the resistance R_c are, at best, approximations of the real excitation effects.

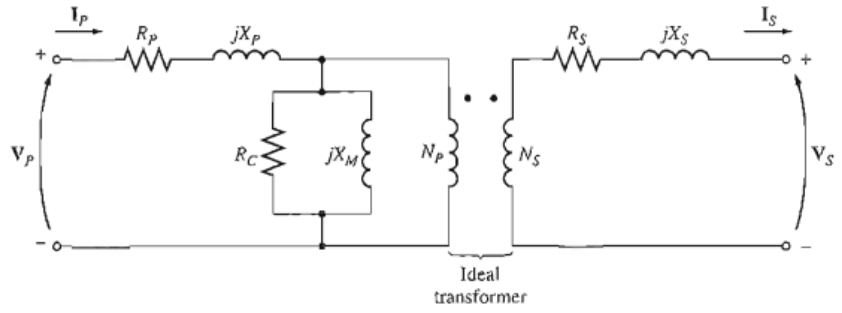


Figure 1: The model of a real transformer

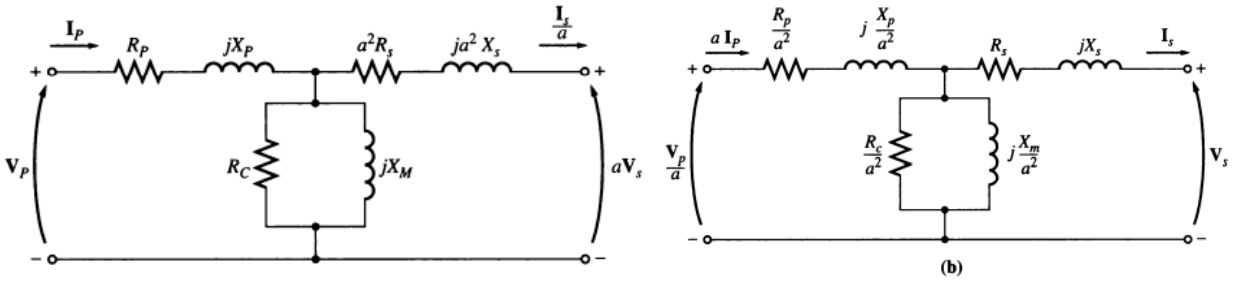


Figure 2: (a) The transformer model referred to its primary voltage level. (b) The transformer model referred to its secondary voltage level.

The resulting equivalent circuit is shown above.In this circuit, R_P is the resistance of the primary winding, X_p is the reactance due to the primary leakage inductance, R_s is the resistance of the secondary winding and X_s is the reactance due to the secondary leakage inductance.The excitation branch is modeled by the resistance R_c (hysteresis and core losses) in parallel with the reactance X_M (the magnetization current).The elements forming the excitation branch are placed inside the primary resistance R_p and reactance X_p. This is because the voltage actually applied to the core is really the input voltage less the internal voltage drops of the winding.

Although the above figure is an accurate model of a transformer,it is not a very useful one.To analyze practical circuits containing transformers, it is normally necessary to convert the entire circuit to an equivalent circuit at a single voltage level.Therefore, the equivalent circuit must be referred either to its primary side or to its secondary side.

Approximate Equivalent Circuits of a Transformer:

The transformer models shown before are often more complex than necessary in order to get good results in practical engineering applications.One of the principal complaints about them is that the excitation branch of the model adds another node to the circuit being analyzed, making the circuit solution more complex than necessary.The excitation branch has a very small current compared to the load current of the transformers. In fact, the excitation current is only about 2-3% of the full load current for typical power transformers.

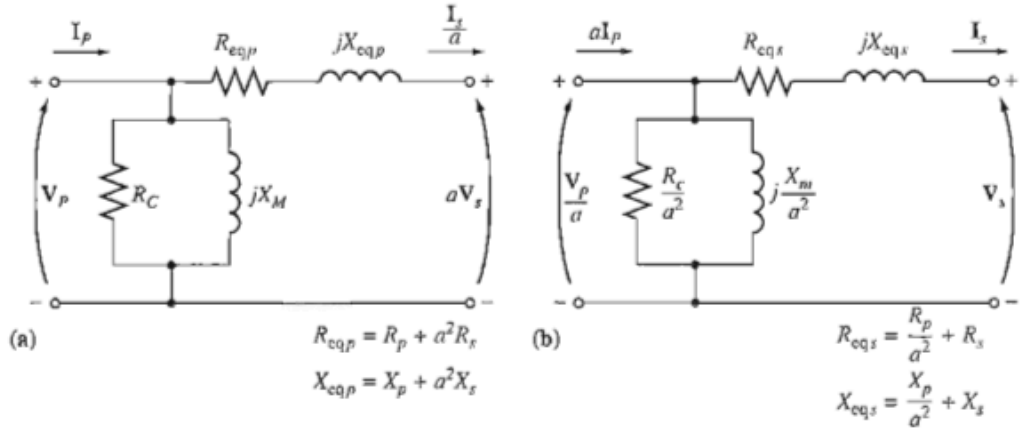


Figure : Approximate transformer models (a) Referred to the primary side; (b) referred to the secondary side; (c) with no excitation branch, referred to the primary side; (d) with no excitation branch, referred to the secondary side.

Because this is true, a simplified equivalent circuit can be produced that works almost as well as the original model. The excitation branch is simply moved to the front of the transformer, and the primary and secondary impedances are left in series with each other. These impedances are just added, creating the approximate equivalent circuits.In some applications, the excitation branch may be neglected entirely without causing serious error. In these cases, the equivalent circuit of the transformer reduces to the simple circuits.

Determining the Values of Components in the Transformer Model:

It is possible to experimentally determine the values of the inductances and resistances in the transformer model. An adequate approximation of these values can be obtained with only two tests, the open-circuit test and the short-circuit test In the open-circuit test, one transformer winding is open-circuited, and the other winding is connected to full rated line voltage.The series elements, R_p and X_p are too small in comparison to R_c and X_M to cause a significant voltage drop, so essentially all the input voltage is dropped across the excitation branch. The open-circuit test connection is shown below.Full line voltage is applied to one side of the transformer and the input voltage, input current, and input power to the transformer are measured.This measurement is normally done on the low-voltage side of the transformer, since lower voltages are easier to work with.From this information, it is possible to determine the power factor of the input current and therefore both the magnitude and the angle of the excitation impedance.

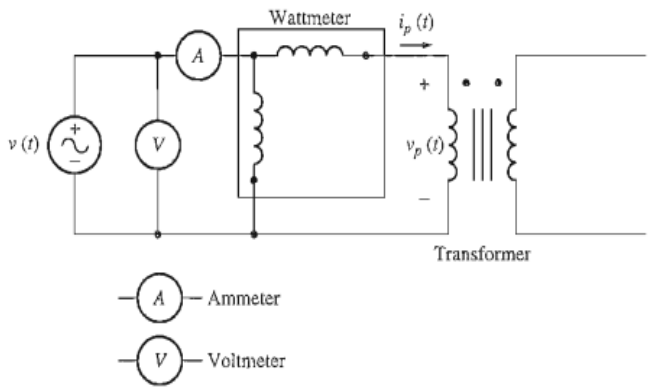


Figure 3: Connection for transformer open-circuit test.

The easiest way to calculate the values of R_c and X_M is to look first at the admittance of the excitation branch.The conductance of the core-loss resistor is given by: $G_C=1/R_C$

And the susceptance of the magnetizing inductor is given by: $B_M=1/X_M$

Since these two elements are in parallel, their admittances add, and the total excitation admittance is

$$Y_E=G_C-jB_M$$

$$Y_E=1/R_C-j(1/X_M)$$

The magnitude of the excitation admittance (referred to the side of the transformer used for the measurement) can be found from the open-circuit test voltage and current:

Magnitude of Y_E= |Ioc/Voc|

Power Factor, PF = Poc/(Voc*Ioc)

Power factor angle, $\theta= \cos^{-1}(PF)$

Hence,Y_E= (Ioc/Voc) $\angle \cos^{-1}(PF)$

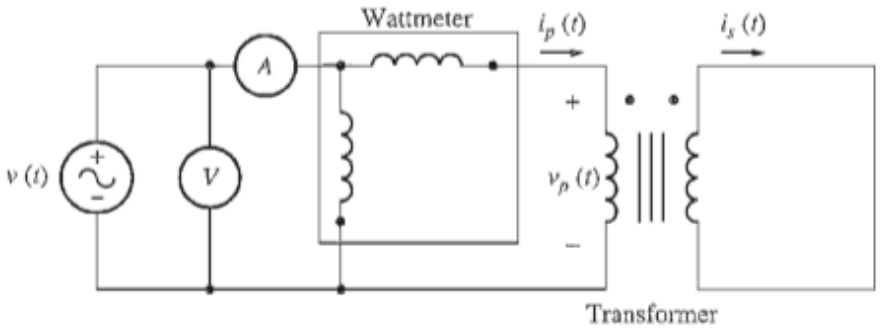


Figure 4: Connection for transformer short-circuit test.

In the short-circuit test, the low-voltage terminals of the transformer are short circuited, and the high-voltage terminals are connected to a variable voltage source.This measurement is normally done on the high-voltage side of the transformer, since currents will be lower on that side, and lower currents are easier to work with.The input voltage is adjusted until the current in the short circuited windings is equal to its rated value.The input voltage, current, and power are again measured.Since the input voltage is so low during the short-circuit test, negligible current flows through the excitation branch. If the excitation current is ignored, then all the voltage drop in the transformer can be attributed to the series elements in the circuit. The magnitude of the series impedances referred to the primary side of the transformer is

Magnitude of Z_{SE}=Absolute value of (Vsc/Isc)

Power Factor,PF=Psc/(Vsc*Isc)

Power factor angle,$\theta=\cos^{-1}(PF)$

$Z_{SE}=(Vsc/Isc)\angle\cos^{-1}(PF)$

The series impedance ZSE is equal to

$Z_{SE}=R_{eq}+jX_{eq}$

$$Z_{SE}=(R_P+a^2R_S)+j(X_P+jX_S)$$

It is possible to determine the total series impedance referred to the high voltage side by using this technique, but there is no easy way to split the series impedance into primary and secondary components. Fortunately, such separation is not necessary to solve normal problems.The open-circuit test is usually performed on the low-voltage side of the transformer, and the short-circuit test is usually performed on the high voltage side of the transformer, so R_c and X_M are usually found referred to the low-voltage side, and R_{eq} and X_{eq} are usually found referred to the high-voltage side. All of the elements must be referred to the same side (either high or low) to create the final equivalent circuit.

2.2. Three Phase Transformer:

Three-phase transformers are transformers that operate with a three-phase electrical system.The working principle of three-phase transformers is same as the single phase transformer i.e. mutual induction.Faraday‘s law of induction is also the governing law of three phase transformer.The alternating supply is given to the primary windings and it induces an emf in the secondary winding. The amount of induced emf depends upon the number of secondary turns (either can be a step-up or step-down transformer).Single-phase and three-phase transformers differ in wiring configurations.Transformers for three-phase circuits can be constructed in one of two ways.One approach is simply to take three single-phase transformers and connect them in a three-phase bank.An alternative approach is to make a three-phase transformer consisting of three sets of windings wrapped on a common core.

Configuration:

A three-phase transformer consists of three transformers, either separate or combined on one core. The primaries and secondaries of any three-phase

transformer can be independently connected in either a wye or a delta .This gives a total of four possible connections for a three-phase transformer bank

1. Wye- wye connection
2. Wye-delta connection
3. Delta-wye connection
4. Delta-delta connection

The analysis of any three-phase transformer bank is to look at a single transformer in the bank.Any single transformer in the bank behaves exactly like the single-phase transformers. The impedance, voltage regulation, efficiency, and similar calculations for three-phase transformers are done on a per-phase basis, using exactly the same techniques developed for single-phase transformers.

Wye- wye connection:

In wye- wye connection the primary line voltage is related to the primary phase voltage by

$$V_{\varphi P}=V_{LP}/\sqrt{3}$$

The secondary line voltage is related to the secondary phase voltage by

$$V_{\varphi S}=V_{LS}/\sqrt{3}$$

$$\text{Voltage Ratio}=V_{LP}/V_{LS}=a$$

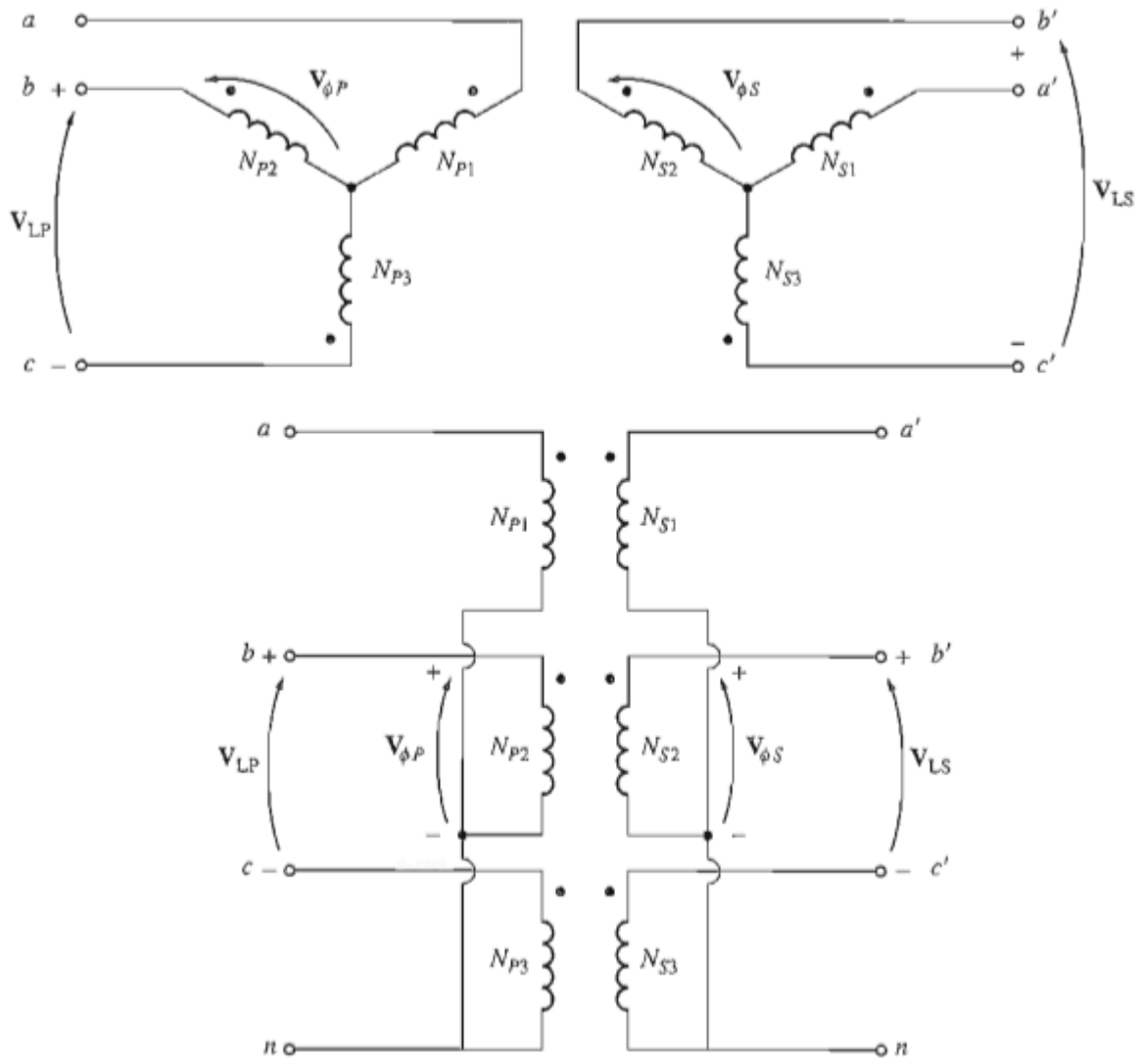


Figure 5: Wye- wye connection

Wye-delta connection:

In wye-delta connection the primary line voltage is related to the primary phase voltage by

$$V_{\phi p} = \frac{V_{LP}}{\sqrt{3}}$$

The secondary line voltage is related to the secondary phase voltage by

$$V_{\varphi S}=V_{LS}$$

Voltage Ratio=$V_{LP}/V_{LS}=a\sqrt{3}$

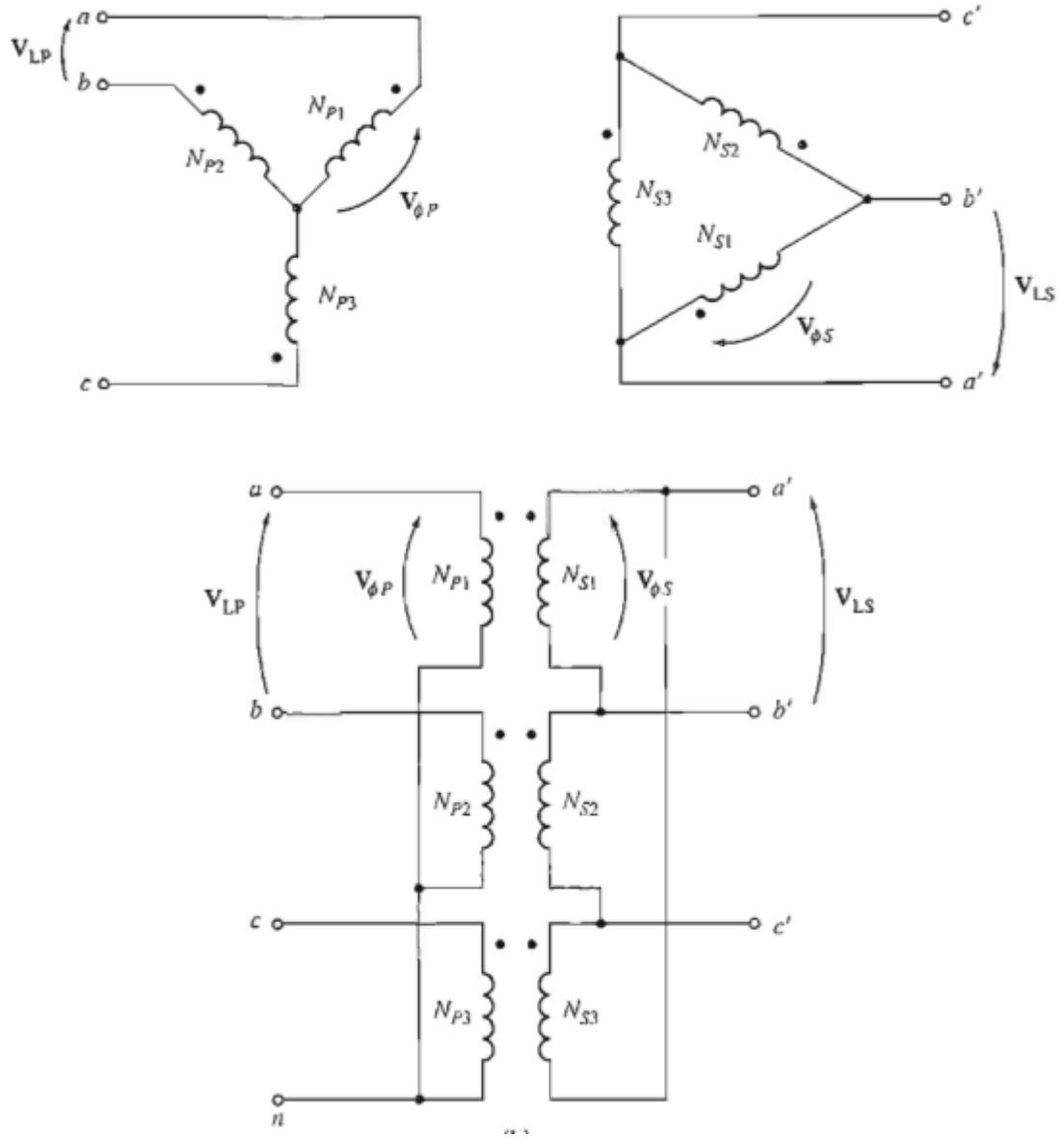


Figure 6: Wye-delta connection

Delta-wye connection:

In delta-wye connection the primary line voltage is related to the primary phase voltage by

$$V_{\varphi P}=V_{LP}$$

The secondary line voltage is related to the secondary phase voltage by

$$V_{\varphi S}=V_{LS}/\sqrt{3}$$

$$\text{Voltage Ratio}=V_{LP}/V_{LS}=a/\sqrt{3}$$

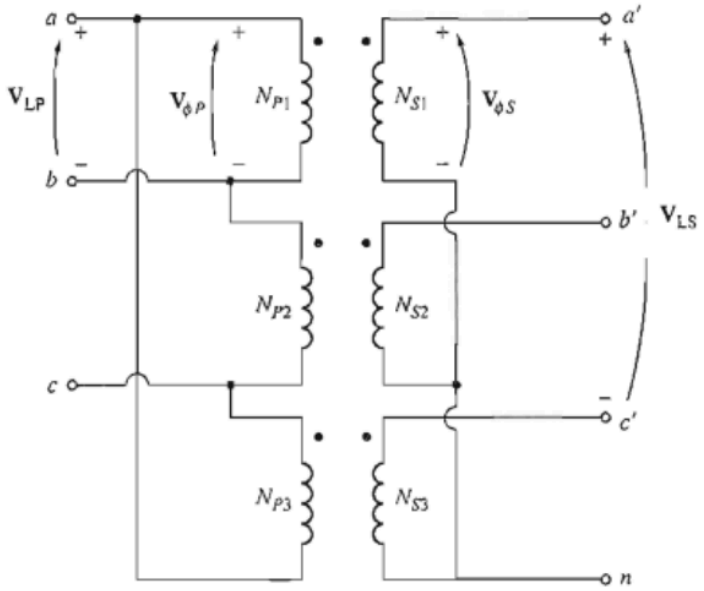


Figure 7: Delta-wye connection

Delta-delta connection:

In delta-delta connection the primary line voltage is related to the primary phase voltage by

$$V_{\varphi P}=V_{LP}$$

The secondary line voltage is related to the secondary phase voltage by

$$V_{\varphi S}=V_{LS}$$

Voltage Ratio=V_{LP}/V_{LS}=a

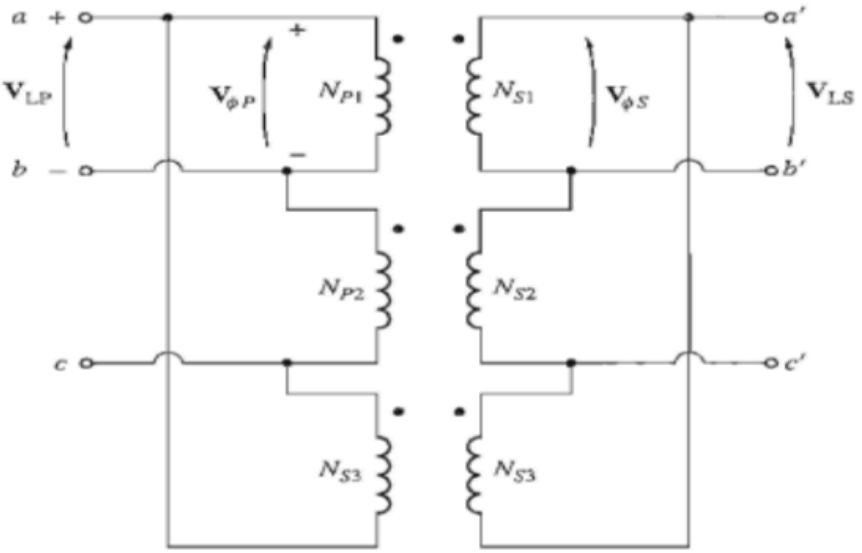


Figure 8: Delta-delta connection

2.3. Auto Transformer :

On some occasions it is desirable to change voltage levels by only a small amount.For example, it may be necessary to increase a voltage from 110 V to 120 V or from 13.2 to 13.8 kV.These small rises may be made necessary by voltage drops that occur in power systems a long way from the generators. In such circumstances, it is wasteful and excessively expensive to wind a transformer with two full windings,each rated at about the same voltage. A special-purpose transformer, called an autotransformer ; is used instead.

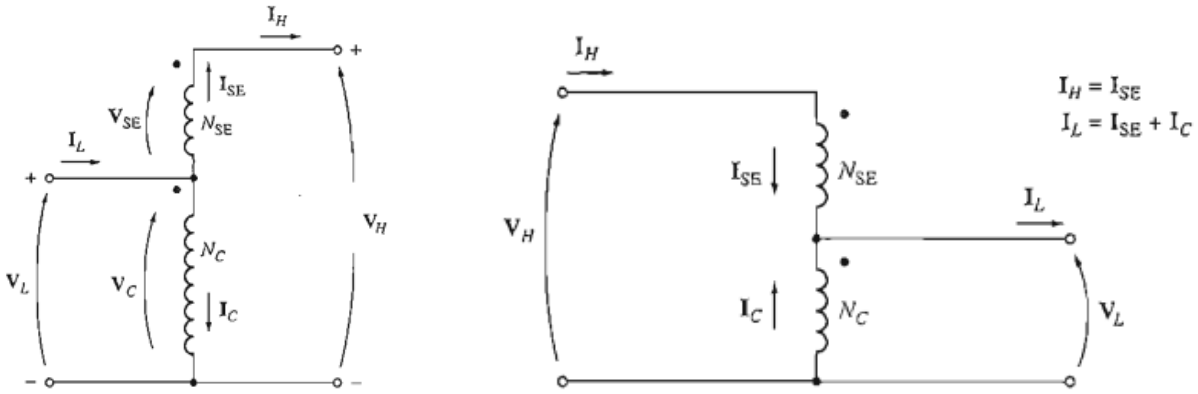


Figure 9: a) transformer with its windings reconnected as a step up auto-transformer b) transformer with its windings reconnected as a step down auto-transformer.

Here,

$$V_C/V_{SE}=N_C/N_{SE}$$

$$N_{SE}*I_{SE}=I_C*N_C$$

$$V_L=V_C$$

$$V_H=V_{SE}+V_C$$

$$I_L=I_C+I_{SE}$$

$$I_H=I_{SE}$$

Again, $V_L/V_H=N_C/(N_C+N_{SE})$

$$I_H/I_L=N_C/(N_C+N_{SE})$$

The Apparent Power Rating Advantage of Autotransformers:

The input apparent power, $S_{in}=V_L*I_L$

The output apparent power, $S_{out}=V_H*I_H$

The input apparent power=The output apparent power=Input-output apparent power=S_{IO}

The apparent power in the transformer winding,$S_W=V_C*I_C=V_{SE}*I_{SE}$

Here,

$$S_{IO}/S_w=(N_C+N_{SE})/N_{SE}$$

This ratio describes the apparent power rating advantage of an autotransformer over a conventional transformer.Here S_{IO} is the apparent power entering the primary and leaving the secondary of the transformer, while S_W is the apparent power actually traveling through the transformer's windings.The rest passes from primary to secondary without being coupled through the transformer's windings.

2.4. Understanding Torque speed characteristics of an induction motor:

The net magnetic field B_{net} in induction motor is produced by the magnetization current I_M flowing in the motor's equivalent circuit.The magnitude of the magnetization current and hence of B_{net} is directly proportional to the voltage E1.If E1 is constant,then the net magnetic field in the motor is constant.In an actual machine,E1 varies as the load changes because the stator impedances R_l and X_l cause varying voltage drops with varying load. However, these drops in the stator windings are relatively small, so E1 (and hence I_M and B_{net}) is approximately constant with changes in load. At no load, the rotor slip is very

small, and so the relative motion between the rotor and the magnetic fields is very small and the rotor frequency is also very small. Since the relative motion is small, the voltage E_R induced in the bars of the rotor is very small, and the resulting current flow I_R is small.Also, because the rotor frequency is so very small, the reactance of the rotor is nearly zero, and the maximum rotor current I_R is almost in phase with the rotor voltage E_R. The rotor current thus produces a small magnetic field B_R at an angle just slightly greater than 90 degree behind the net magnetic field B_{net}.The stator current must be quite large even at no load, since it must supply most of B_{net}.This is why induction motors have large no-load currents compared to other types of machines. The no-load current of an induction motor is usually 30-60 percent of the full-load current.

Induced torque, $T_{ind} = k B_R \times B_{net}$

Its magnitude is: $T_{ind} = k B_R B_{net} \sin\delta$

If the motor's load increases, its slip increases, and the rotor speed falls. Since the rotor speed is slower, there is now more relative motion between the rotor and the stator magnetic fields in the machine. Greater relative motion produces a stronger rotor voltage E_R which in turn produces a larger rotor current I_R. With a larger rotor current, the rotor magnetic field B_R also increases.However, the angle of the rotor current and B_R changes as well. Since the rotor slip is larger, the rotor frequency rises and the rotor's reactance increases.Therefore, the rotor current now lags further behind the rotor voltage, and the rotor magnetic field shifts with the current.The induction motor operates at a fairly high load. The rotor current has increased and that the angle δ has increased. The increase in B_R tends to increase the torque, while the increase in angle δ tends to decrease the torque (T_{ind} is proportional to sinδ, and $\delta > 90°$). Since the first effect is larger than the second one, the overall induced torque in[1]creases to supply the motor's increased load. An induction motor reaches pullout torque when the point is reached where, as the load on the shaft is increased, the sinδ term

decreases more than the B_R term increases. At that point, a further increase in load decreases T_{ind}, and the motor stops.

2.5. Understanding induction motor torque-speed characteristic:

The magnitude of the induced torque in the machine is given by

$$T_{ind}=k\, B_R\, B_{net} \sin\delta$$

Each term in this expression can be considered separately to derive the overall machine behavior.

1.The rotor magnetic field, B_R is directly proportional to the current flowing in the rotor, as long as the rotor is unsaturated. The current flow in the rotor increases with increasing slip (decreasing speed)

2.The net magnetic field,B_{net} in the motor is proportional to E1 and therefore is approximately constant.E1 actually decreases with increasing current flow, but this effect is small compared to the other two, and it will be ignored in this graphical development.

3.The angle δ between the net and rotor magnetic fields can be expressed in a very useful way.It is clear that the angle δ is just equal to the power-factor angle of the rotor plus 90 degree.

$$\delta = \theta_R + 90 \text{ degree}$$

$$\sin\delta = \sin(\theta_R + 90) = \cos\theta_R$$

$$PF_R = \cos(\tan^{-1}(X_R/R_R))$$

Since the induced torque is proportional to the product of these three terms, the torque-speed characteristic of an induction motor can be constructed from the graphical multiplication of the previous three plots.The torque-speed characteristic of an induction motor derived in this fashion is shown here.

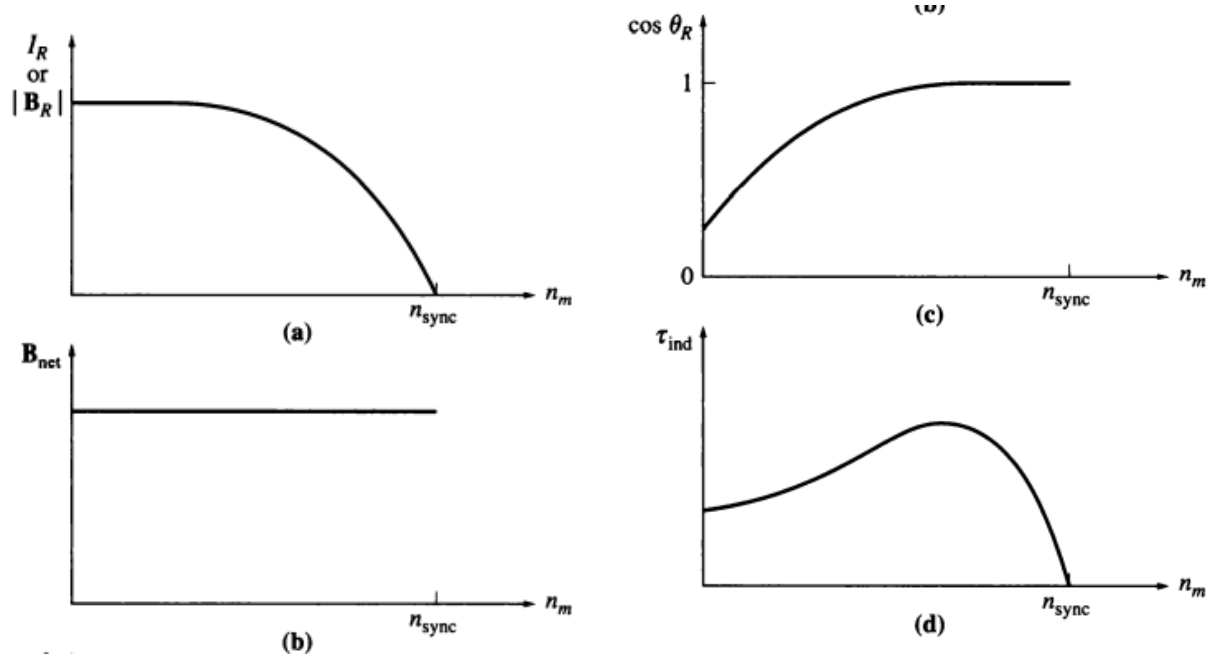
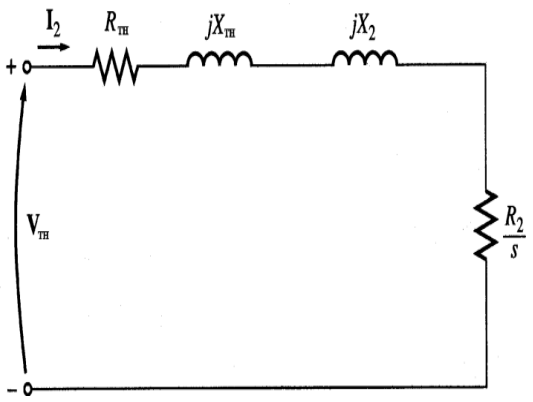


Figure 10: (a) Plot of rotor current versus speed for an induction motor; (b) plot of net magnetic field versus speed for the motor; (c) plot of rotor power factor versus speed for the motor; (d) the resulting torque-speed characteristic.

The characteristic curve can be divided roughly into three regions.The first region is the low-slip region of the curve.In the low-slip region, the motor slip increases approximately linearly with increased load, and the rotor mechanical speed decreases approximately linearly with load. In this region of operation, the rotor reactance is negligible, so the rotor power factor is approximately unity, while the rotor current increases linearly with slip. The entire normal steady-state operating range of an induction motor is included in this linear low-slip region.Thus in normal operation, an induction motor has a linear speed droop. The second region on the induction motor's curve can be called the moderate slip region. In the moderate-slip region, the rotor frequency is higher than before, and the rotor reactance is on the same order of magnitude as the

rotor resistance. In this region, the rotor current no longer increases as rapidly as before, and the power factor starts to drop. The peak torque (the pullout torque) of the motor occurs at the point where, for an incremental increase in load, the increase in the rotor current is exactly balanced by the decrease in the rotor power factor. The third region on the induction motor's curve is called the high-slip region. In the high-slip region, the induced torque actually decreases with increased load, since the increase in rotor current is completely overshadowed by the decrease in rotor power factor. For a typical induction motor, the pullout torque on the curve will be 200 to 250 percent of the rated full-load torque of the machine, and the starting torque (the torque at zero speed) will be 150 percent or so of the full-load torque. Unlike a synchronous motor, the induction motor can start with a full load attached to its shaft.

2.6. The Derivation of the Induction Motor Induced-Torque Equation:



Like some other AC machines, a torque is induced in the induction motor when an electrical energy is supplied.An induction motor develops torque by inducing current to the rotor, which is proportional to the differential speed of the rotor and the rotating magnetic field in the stator.Using the Thevenin equivalent of an induction motor circuit, we can derive the relationship between induced torque and the mechanical speed of the rotor.

For a 3 Phase Induction Motor, the power converted to mechanical form is:

$$P_{conv} = \frac{3I_2^2 R_2}{S} \qquad (1)$$

So the induced torque will be:

$$\tau_{ind} = \frac{P_{conv}}{\omega_m}$$ where ω_m is the rotor frequency.

Now using the definition of slip: $s = \frac{n_{sync} - n_m}{n_{sync}}$ and line frequency $\omega_s = \frac{2\pi \times n_{sync}}{60}$

We have,

$$\tau_{ind} = \frac{P_{conv}}{(1-S)\omega_s} = \frac{3I_2^2 R_2}{S\omega_s} \quad (2)$$

Now, from the circuit, $I_2 = \frac{V_{TH}}{Z_T} = \frac{V_{TH}}{\sqrt{(R_{TH} + \frac{R_2}{s})^2 + (X_{TH} + X_2)^2}}$ (3)

So, finally we get the relation between the slip and the induced torques using equation (1), (2), & (3)

$$\tau_{ind} = \frac{1}{\omega_s} \frac{3V_{TH}^2 (\frac{R_2}{s})}{(R_{TH} + \frac{R_2}{s})^2 + (X_{TH} + X_2)^2}$$

Graph of the following equation is drawn below;

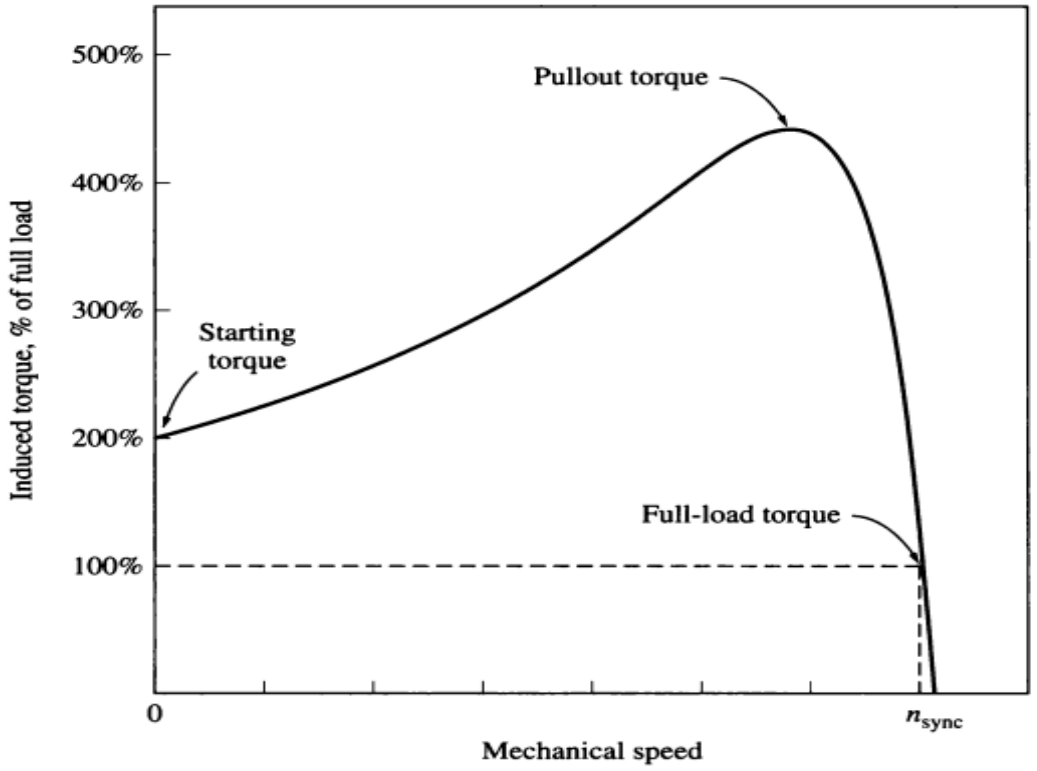


Figure 11: A typical induction motor torque-speed characteristic curve

We can make the following comments observing the graph:

1.The induced torque is zero at synchronous speed.

2. The curve is nearly linear between no-load and full load. In this range, the rotor resistance is much greater than the reactance, so the rotor current, torque increase linearly with the slip.

3. There is a maximum possible torque that can't be exceeded. This torque is called pullout torque and is 2 to 3 times the rated full-load torque.

4. The starting torque of the motor is slightly higher than its full-load torque, so the motor will start carrying any load it can supply at full load.

5. The torque of the motor for a given slip varies as the square of the applied voltage. 6. If the rotor is driven faster than synchronous speed it will run as a generator, converting mechanical power to electric power,

However, a more extended graph, including negative slip, will look like the following:

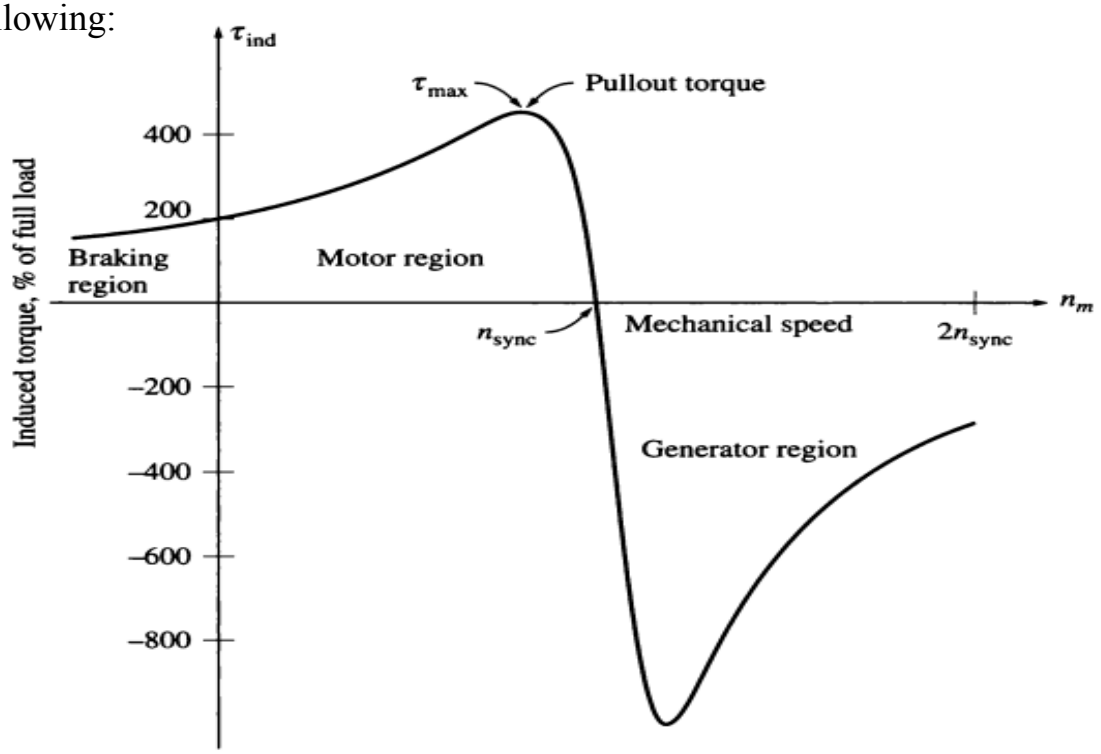


Figure 12: Induction motor torque-speed characteristic curve,showing the extended operating ranges (braking region and generator region).

Generator region:

If the rotor of the induction motor is driven faster than synchronous speed, then the direction of the induced torque in the machine reverses and the machine becomes a generator, converting mechanical power to electric power.

Braking region:

If the motor is turning backward relative to the direction of the magnetic fields, the induced torque in the machine will stop the machine very rapidly and will try to rotate it in the other direction. Since reversing the direction of magnetic field rotation is simply a matter of switching any two stator phases, this fact can be used as a way to very rapidly stop an induction motor. The act of switching two phases in order to stop the motor very rapidly is called plugging.

2.6. Various Speed Control Technique of an Induction Motor:

There are only two techniques by which the speed of an induction motor can be controlled. One is to vary the synchronous speed, which is the speed of the stator and rotor magnetic fields, since the rotor speed always remains near synchronous speed.The other technique is to vary the slip of the motor for a given load. Each of these approaches will be taken up in more detail.

The synchronous speed of an induction motor is given by

$$n_{sync}=120f_{se}/P$$

So the only ways in which the synchronous speed of the machine can be varied are (1) by changing the electrical frequency and (2) by changing the number of

poles on the machine. Slip control may be accomplished by varying either the rotor resistance or the terminal voltage of the motor.

Speed Control by Changing the Line Frequency:

If the electrical frequency applied to the stator of an induction motor is changed, the rate of rotation of its magnetic fields n_{sync} will change in direct proportion to the change in electrical frequency, and the no-load point on the torque-speed characteristic curve will change with it.The synchronous speed of the motor at rated conditions is known as the base speed. By using variable frequency control, it is possible to adjust the speed of the motor either above or below base speed. A properly designed variable-frequency induction motor drive can be very flexible. It can control the speed of an induction motor over a range from as little as 5 percent of base speed up to about twice base speed. However, it is important to maintain certain voltage and torque limits on the motor as the frequency is varied, to ensure safe operation.

When running at speeds below the base speed of the motor, it is necessary to reduce the terminal voltage applied to the stator for proper operation. The terminal voltage applied to the stator should be decreased linearly with decreasing stator frequency. This process is caned derating. If it is not done, the steel in the core of the induction motor will saturate and excessive magnetization currents will flow in the machine. To understand the necessity for derating, recall that an induction motor is basically a rotating transformer. As with any transformer, the flux in the core of an induction motor can be found from Faraday's law:

$$v(t) = -N\left(\frac{d\phi}{dt}\right)$$

If $v(t) = V_m sin(\omega t)$

$$\varphi(t) = -\frac{V_m cos\omega t}{\omega NP}$$

In the equation of flux the electrical frequency appears in the denominator of this expression. Therefore, if the electrical frequency applied to the stator decreases by 10 percent while the magnitude of the voltage applied to the stator remains constant, the flux in the core of the motor will increase by about 10 percent and the magnetization current of the motor will increase. In the unsaturated region of the motor's magnetization curve, the increase in magnetization current will also be about 10 percent. However, in the saturated region of the motor's magnetization curve, a 10 percent increase in flux requires a much larger increase in magnetization current. Induction motors are normally designed to operate near the saturation point on their magnetization curves, so the increase in flux due to a decrease in frequency will cause excessive magnetization currents to flow in the motor.

To avoid excessive magnetization currents, it is customary to decrease the applied stator voltage in direct proportion to the decrease in frequency whenever the frequency falls below the rated frequency of the motor. Since the applied voltage v appears in the numerator of the equation and the frequency appears in the denominator of the equation,the two effects counteract each other, and the magnetization current is unaffected. When the voltage applied to an induction motor is varied linearly with frequency below the base speed, the flux in the motor will remain approximately constant. Therefore, the maximum torque which the motor can supply remains fairly high. However, the maximum power rating of the motor must be decreased linearly with decreases in frequency to protect the stator circuit from overheating.

The power supplied to a three-phase induction motor is given by

$$P = \sqrt{3} V_L I_L \cos\theta$$

If the voltage V_L is decreased, then the maximum power P must also be decreased, or else the current flowing in the motor will become excessive, and the motor will overheat.

When the electrical frequency applied to the motor exceeds the rated frequency of the motor, the stator voltage is held constant at the rated value. Although saturation considerations would permit the voltage to be raised above the rated value under these circumstances, it is limited to the rated voltage to protect the winding insulation of the motor. The higher the electrical frequency above base speed, the larger the denominator of the equation becomes. Since the numerator term is held constant above rated frequency, the resulting flux in the machine decreases and the maximum torque decreases with it.In the past, the principal disadvantage of electrical frequency control as a (method of speed changing was that a dedicated generator or mechanical frequency changer was required to make it operate. This problem has disappeared with the development of modern solid-state variable-frequency motor drives. In fact, changing the line frequency with solid-state motor drives has become the method of choice for induction motor speed control. This method can be used with any induction motor, unlike the pole-changing technique, which requires a motor with special stator windings.

Speed Control by Changing the Line Voltage:

The torque developed by an induction motor is proportional to the square of the applied voltage. If a load has a torque-speed characteristic,then the speed of the motor may be controlled over a limited range by varying the line voltage. This method of speed control is sometimes used on small motors driving fans.

Speed Control by Changing the Rotor Resistance:

In wound-rotor induction motors, it is possible to change the shape of the torque speed curve by inserting extra resistances into the rotor circuit of the machine. The resulting torque-speed characteristic curves are shown below. If the torque-speed curve of the load is as shown in the figure, then changing the rotor resistance will change the operating speed of the motor. However, inserting extra resistances into the rotor circuit of an induction motor seriously reduces

the efficiency of the machine. This method of speed control is mostly of historical interest, since very few previous paragraph. wound-rotor induction motors are built anymore. When it is used, it is normally used only for short periods because of the efficiency problem mentioned in the previous paragraph.

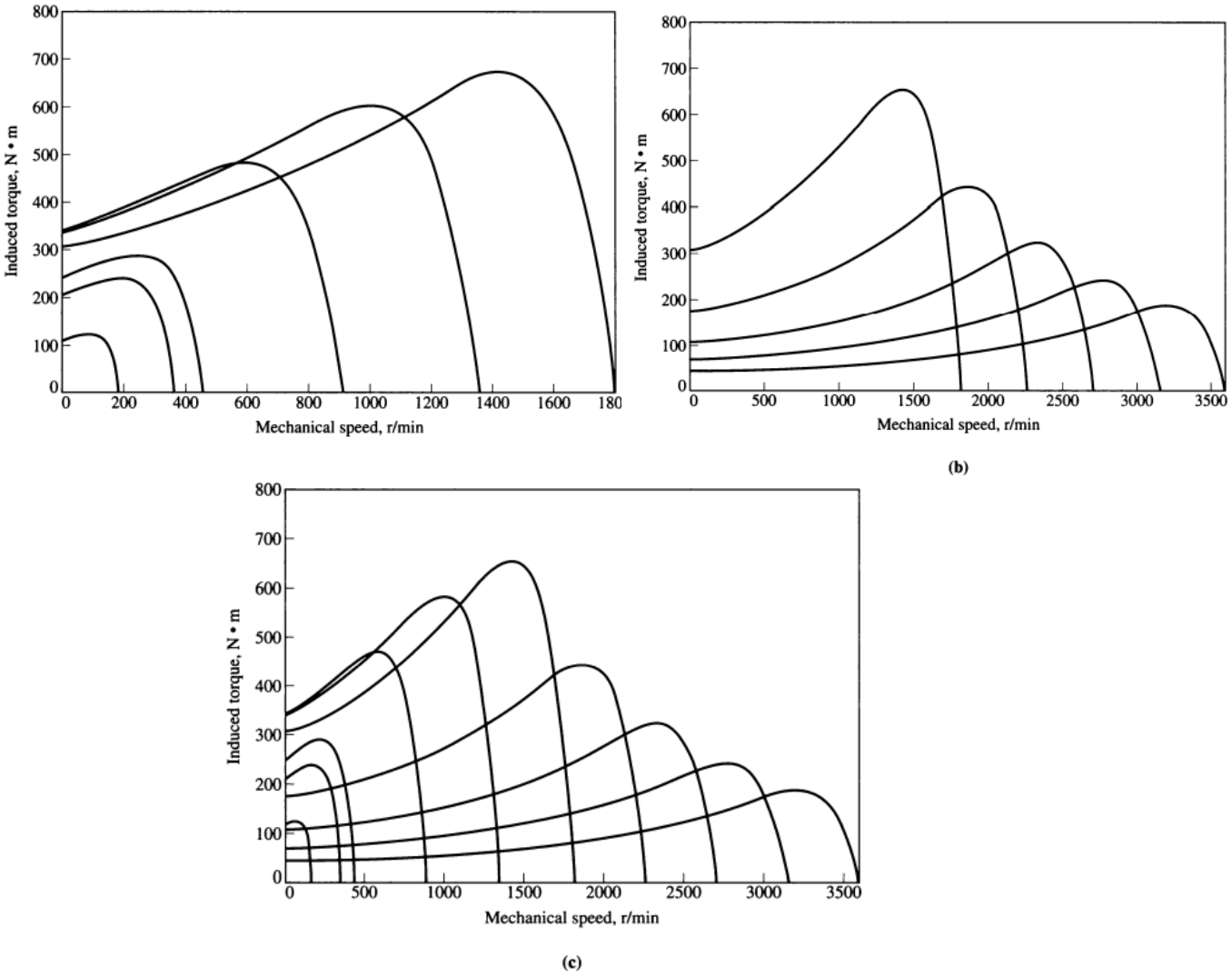


Figure 13: a) Variable-line-voltage speed control in an induction motor b)The torque-speed characteristic curves for all frequencies c) Speed control by varying the rotor resistance of a wound-rotor induction motor

App Arrangement Flow Chart

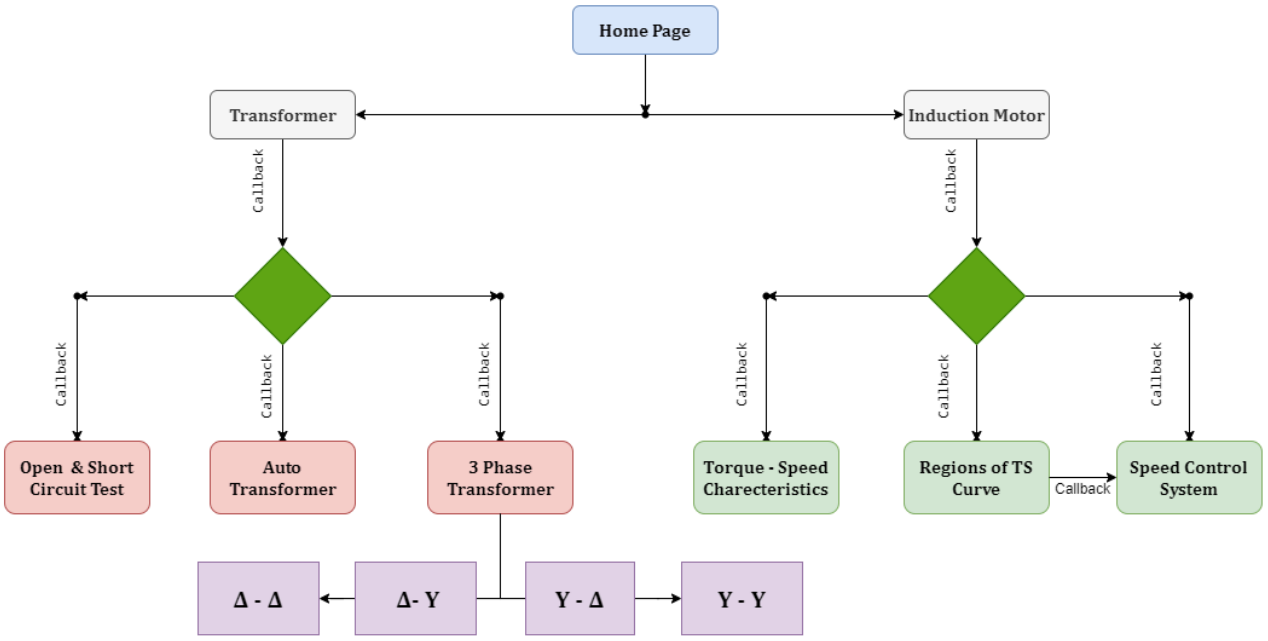


Figure 14: App arrangement flow chart

pseudocodes, Codes & Test Cases

All the codes of this project have been attached to the following GitHub Respiratory

Only the important code sections and their psudocodes/ algorithms are included here.

4.1. Home Page:

Code:

```
    % Callbacks that handle component events
      methods (Access = private)

            % Code that executes after component creation
            function startupFcn(app)
                  app.UIFigure.Name = 'EEE 212 Project';
            end

            % Button pushed function: OpenCircuitShortCircuitTestButton
            function OpenCircuitShortCircuitTestButtonPushed(app, event)
                Xer_Open_Short;
            end

            % Button pushed function: TorqueSpeedCharecteristicsButton
            function TorqueSpeedCharecteristicsButtonPushed(app, event)
                  IM_Torque_Speed;
            end

            % Button pushed function: DifferentRegionsinTSCurveButton
            function DifferentRegionsinTSCurveButtonPushed(app, event)
                  IM_TS_Region;
            end

            % Button pushed function: AutoTransformerButton
            function AutoTransformerButtonPushed(app, event)
                  Xer_Auto;
            end

            % Button pushed function:
            % VariousSpeedControlTechniquesButton
            function VariousSpeedControlTechniquesButtonPushed(app, event)
                  IM_Speed_Control;
            end

            % Button pushed function: PhaseTransformerButton
            function PhaseTransformerButtonPushed(app, event)
                  Xer_3Phase;
            end
      end
```

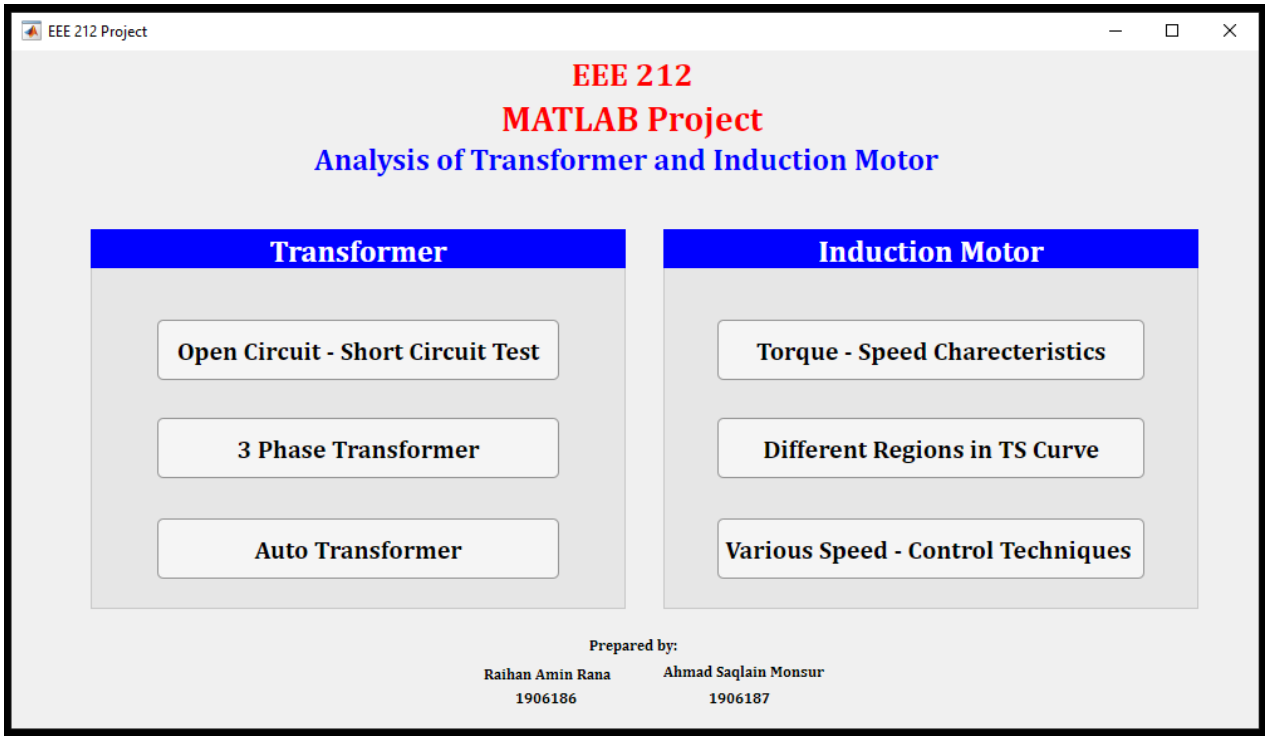


Figure 15: Home Page MATLAB App

4.2. Open & Short Circuit Test:

Code:

```
properties (Access = public)
   S = 15*1000; %rating in kVA9converted to VA or W)
   a =  10;  %turns ratio
   V_oc = 230;
   I_oc = 2.1;
   P_oc = 50;
   V_sc= 47;
   I_sc = 6.0;
   P_sc = 160; % Description
end

methods (Access = public)

        function update_primary_side(app)
              %Values from Open Circuit
              pf1 = app.P_oc/(app.V_oc*app.I_oc);
              lead_lag = -1;
              Y_E = (app.I_oc/app.V_oc)*(pf1
  +1i*lead_lag*sin(acosd(pf1)*pi/180));
              app.R_CP.Value = (app.a^2/real(Y_E))/1000;
              app.X_MP.Value = abs(app.a^2/abs(imag(Y_E)))/1000;

              %Values from Short Circuit
              pf2 = app.P_sc/(app.V_sc*app.I_sc);
              lead_lag = 1;
             Z_se = (app.V_sc/app.I_sc)*(pf2 +
  1i*lead_lag*sin(acosd(pf2)*pi/180));
              app.R_eqP.Value = real(Z_se);
              app.X_eqP.Value = imag(Z_se);
        end

        function update_secondary_side(app)
              %Values from Open Circuit
              pf1 = app.P_oc/(app.V_oc*app.I_oc);
              lead_lag = -1;
              Y_E = (app.I_oc/app.V_oc)*(pf1
  +1i*lead_lag*sin(acosd(pf1)*pi/180));
              app.R_CS.Value = 1/real(Y_E);
              app.X_MS.Value = 1/abs(imag(Y_E));

              %Values from Short Circuit
              pf2 = app.P_sc/(app.V_sc*app.I_sc);
              lead_lag = 1;
              Z_se = (app.V_sc/app.I_sc)*(pf2 +
  1i*lead_lag*sin(acosd(pf2)*pi/180));
               app.R_eqS.Value = real(Z_se)/app.a^2;
              app.X_eqS.Value = imag(Z_se)/app.a^2;

        end
    end

    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
              app.UIFigure.Name= 'Xer Open Short';
```

```
            end

            % Button pushed function: FindEquivalentCircuitButton
            function FindEquivalentCircuitButtonPushed(app, event)
                  app.S = app.TurnsRatioEditField.Value; %rating update
                  app.a = app.TurnsRatioEditField.Value;%turns ration update
                  app.V_oc = app.V_OCEditField.Value;
                  app.I_oc = app.I_OCEditField.Value;
                  app.P_oc = app.P_OCEditField.Value;
                  app.V_sc = app.V_SCEditField.Value;
                  app.I_sc = app.I_SCEditField.Value;
                  app.P_sc = app.P_SCEditField.Value;

                  update_primary_side(app);
                  update_secondary_side(app);
            end
    end
```

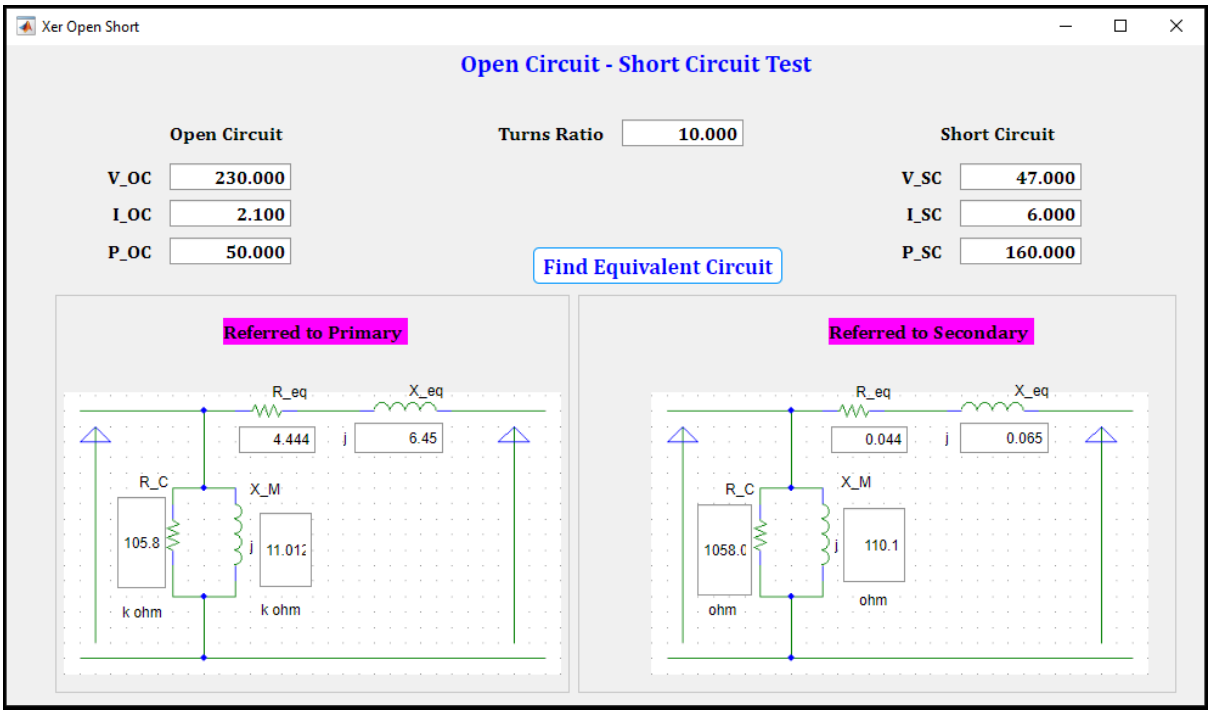


Figure 16: Open & Short Circuit Test case

4.3. Three Phase Transformer:

```
% Callbacks that handle component events
  methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
              app.UIFigure.Name = 'Xer 3 Phase';
        end

        % Button pushed function: PhaseButton
        function PhaseButtonPushed(app, event)
              Xer_3Phase_D_D;
        end

        % Button pushed function: PhaseYButton
        function PhaseYButtonPushed(app, event)
              Xer_3Phase_D_Y;
          end

        % Button pushed function: PhaseYButton_2
        function PhaseYButton_2Pushed(app, event)
              Xer_3Phase_Y_D;
        end

        % Button pushed function: PhaseYYButton
        function PhaseYYButtonPushed(app, event)
          Xer_3Phase_Y_Y;
        end
  end
```

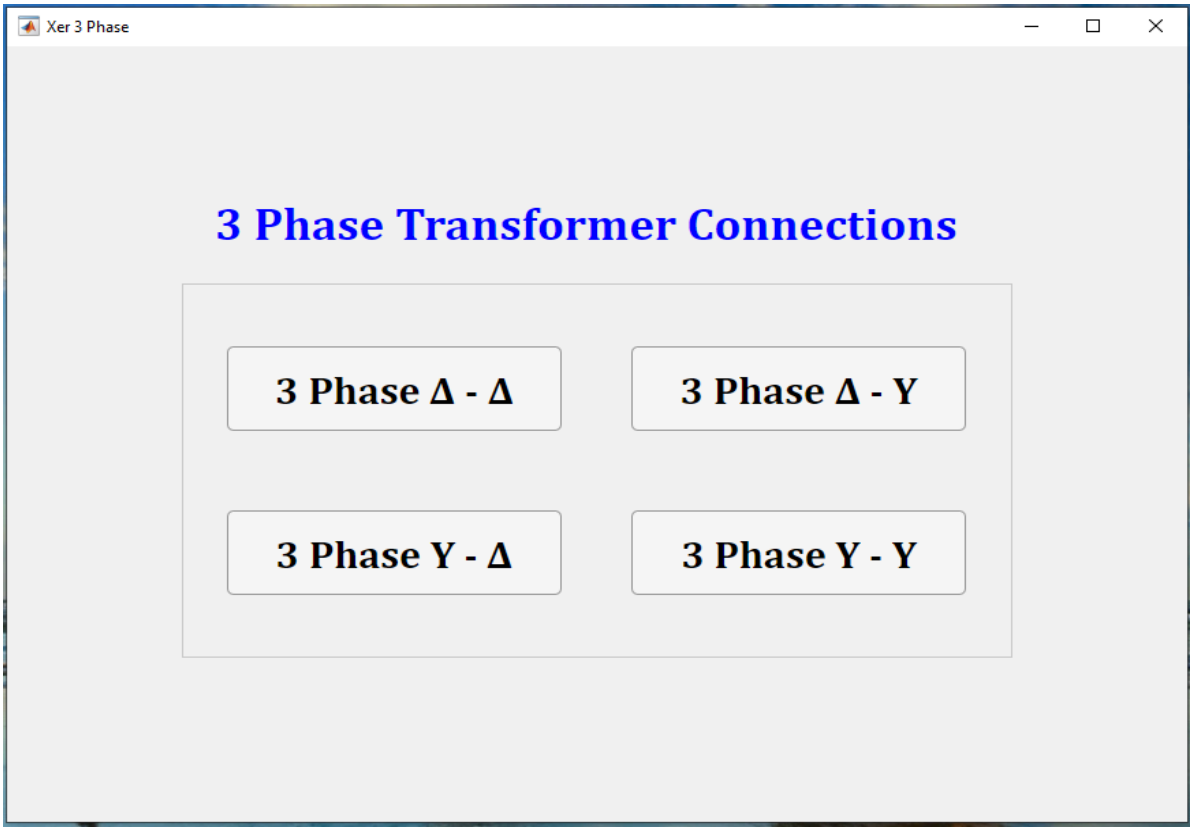


Figure 17: 3 Phase Transformer Window

a. Wye-Delta Connection:

pseudocode:

```
1. Start the app
2. Being the startup Function called,
3. Store  some default values to global variables
4. With the “Circuit Analysis” pushed callback
        if some value has provided
             Update variables
             Call the functions to show
             Show the outputs
        else
             Use default values
             Call the functions to show
             Show the outputs
        end if
5. if end callback
        End the app
   else
        Go to step 4
```

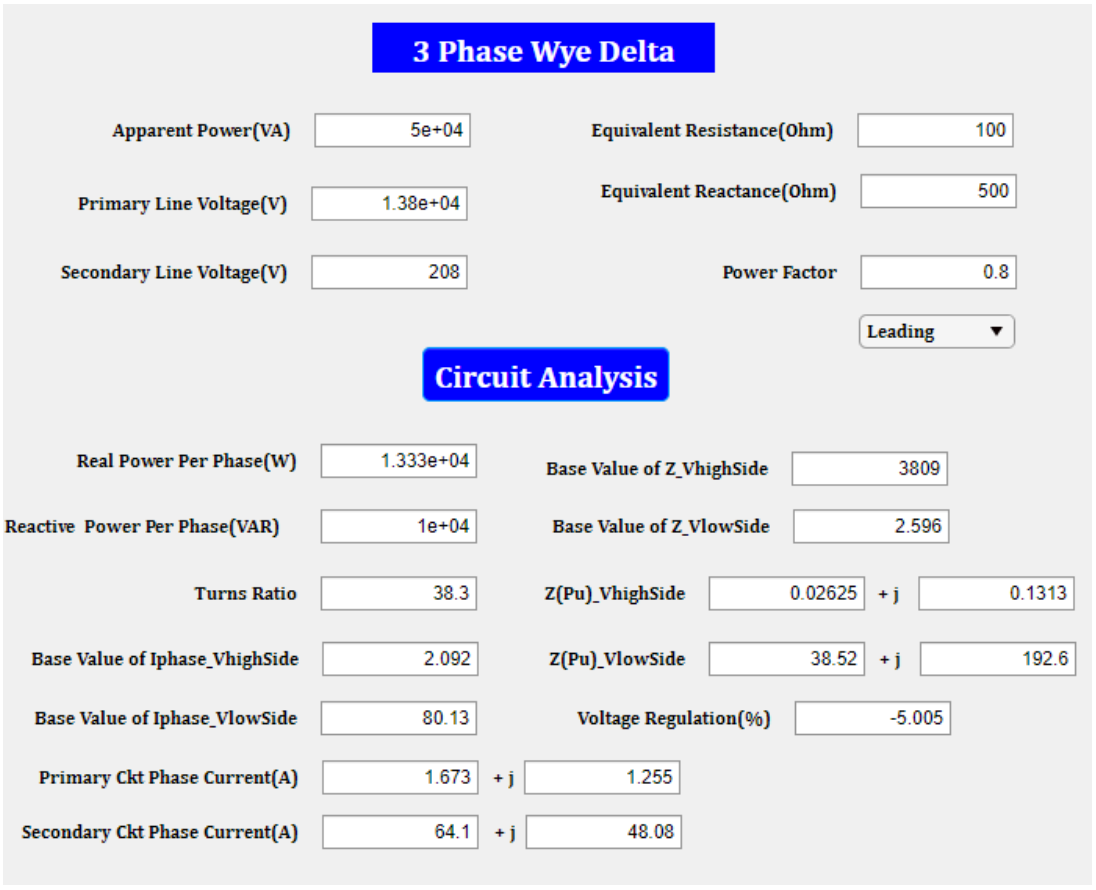
Code:

```
        % Callbacks that handle component events
    methods (Access = private)


        % Button pushed function: CircuitAnalysisButton
        function CircuitAnalysisButtonPushed(app, event)
            %Wye_delta
            %Taking Inputs
            S=app.ApparentPowerVAEditField.Value;
            Vlp=app.PrimaryLineVoltageVEditField.Value;
            Vls=app.SecondaryLineVoltageVEditField.Value;
            PF=app.PowerFactorEditField.Value;
            PFchar=app.DropDown.Value;
            R=app.EquivalentResistanceOhmEditField.Value;
            X=app.EquivalentReactanceOhmEditField.Value;
            %Real power per phase
            P=S*abs(PF);
            P_PerPhase=P/3
            app.RealPowerPerPhaseWEditField.Value=P_PerPhase;
            %Reactive power per phase
            Q=sqrt(S^2-P^2);
            Q_PerPhase=Q/3
            app.ReactivePowerPerPhaseVAREditField.Value=Q_PerPhase;
            %Turns ratio
            Vphase_primary=Vlp/sqrt(3);
            Vphase_secondary=Vls;
            a=Vphase_primary/Vphase_secondary;
            app.TurnsEditField.Value=a;
            %Base value in high voltage side
```

```
                Iphase_high_vol=S/(3*Vphase_primary);
                app.BaseValueofIphase_VhighSideEditField.Value=Iphase_high_vol;
                Zphase_high_vol=3*(Vphase_primary^2)/S;
                app.BaseValueofZ_VhighSideEditField.Value=Zphase_high_vol;
                %Base value in low voltage side
                Iphase_low_vol=S/(3*Vphase_secondary);
                app.BaseValueofIphase_VlowSideEditField.Value=Iphase_low_vol;
                Zphase_low_vol=3*(Vphase_secondary^2)/S;
                app.BaseValueofZ_VlowSideEditField.Value=Zphase_low_vol;
                %Impedence in pu in high voltage side
                Zh_real=R/Zphase_high_vol;
                Zh_imaginary=X/Zphase_high_vol;
                app.ZPu_VhighSideEditField.Value=Zh_real;
                app.EditField_2.Value=Zh_imaginary;
                %Impedence in pu in low voltage side
                Zl_real=R/Zphase_low_vol;
                Zl_imaginary=X/Zphase_low_vol;
                app.ZPu_VlowSideEditField.Value=Zl_real;
                app.EditField2_2.Value=Zl_imaginary;
                %Primary Phase current
                if PFchar=="Leading"
                   theta=acosd(PF);
                elseif PFchar=="Lagging"
                   theta=-acosd(PF);
                else
                  theta=acosd(1);
                end
```

```
                Ip=S/(3*Vphase_primary);
                Ipx=Ip*cosd(theta);
                Ipy=Ip*sind(theta);
                Ipcomplex=Ipx+Ipy*j;
                app.PrimaryCktPhaseCurrentAEditField.Value=Ipx;
                app.EditField.Value=Ipy;
                %Secondary Phase current
                Is=S/(3*Vphase_secondary);
                Isx=Is*cosd(theta);
                Isy=Is*sind(theta);
                Iscomplex=Isx+Isy*j;
                app.SecondaryCktPhaseCurrentAEditField.Value=Isx;
                app.EditField2.Value=Isy;
                %Voltage Regulation
                V_phi_P=a*Vphase_secondary+(R+X*j)*Ipcomplex;
                VR_num=abs(V_phi_P)-a*Vphase_secondary;
                VR_denom=a*Vphase_secondary;
                VR=(VR_num/VR_denom)*100;
                app.VoltageRegulationEditField.Value=VR;
            end
        end
```

Test Case:
Leading Power Factor:



Lagging Power Factor:

Figure 18: test cases for 3 Phase Wye - Delta

b. Wye-Wye Connection:

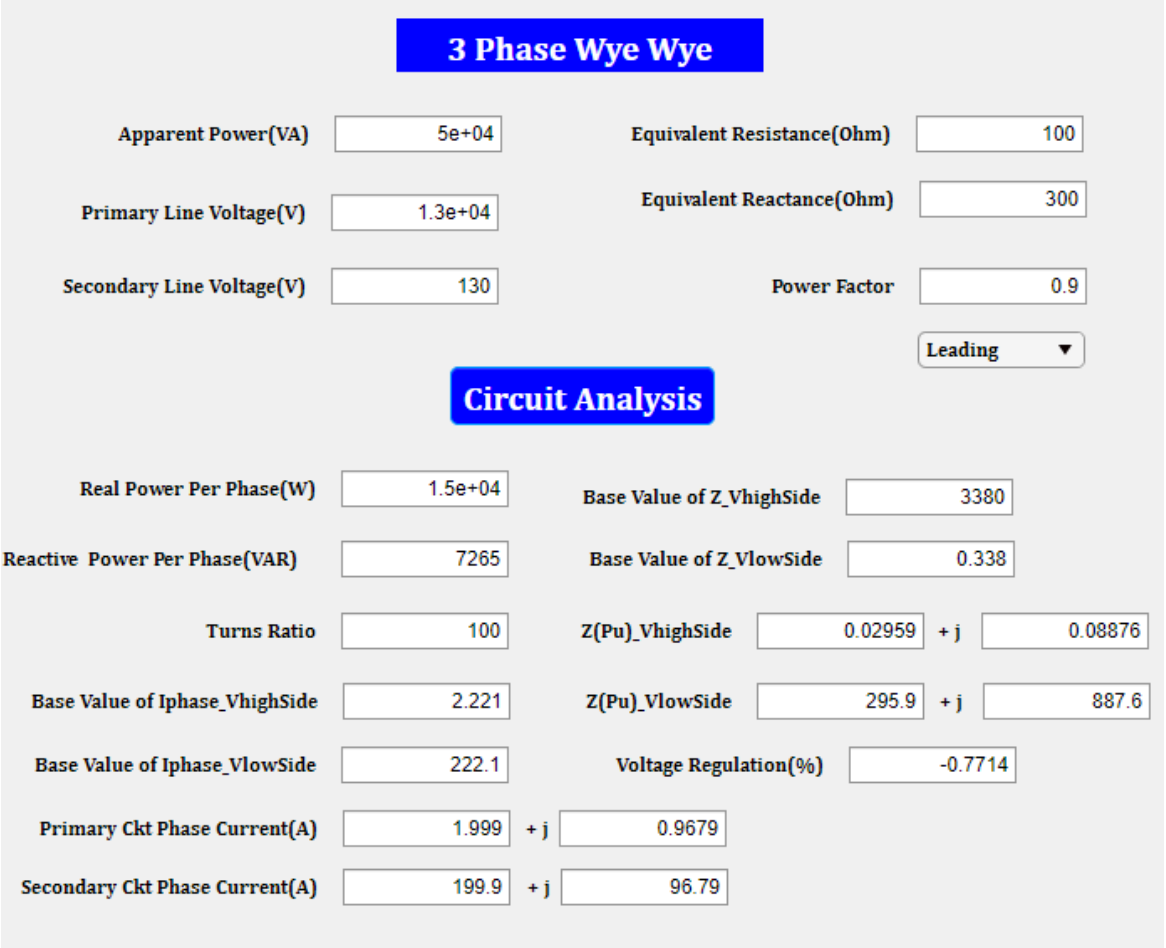
Code:

```
% Callbacks that handle component events

       methods (Access = private)


       % Button pushed function: CircuitAnalysisButton

       function CircuitAnalysisButtonPushed(app, event)
%Wye_wye
%Taking Inputs
S=app.ApparentPowerVAEditField.Value;
Vlp=app.PrimaryLineVoltageVEditField.Value;
Vls=app.SecondaryLineVoltageVEditField.Value;
PF=app.PowerFactorEditField.Value;
PFchar=app.DropDown.Value;
R=app.EquivalentResistanceOhmEditField.Value;
X=app.EquivalentReactanceOhmEditField.Value;
%Real power per phase
P=S*abs(PF);
P_PerPhase=P/3
app.RealPowerPerPhaseWEditField.Value=P_PerPhase;
%Reactive power per phase
Q=sqrt(S^2-P^2);
Q_PerPhase=Q/3
app.ReactivePowerPerPhaseVAREditField.Value=Q_PerPhase;
%Turns ratio
Vphase_primary=Vlp/sqrt(3);
Vphase_secondary=Vls/sqrt(3);
a=Vphase_primary/Vphase_secondary;
```

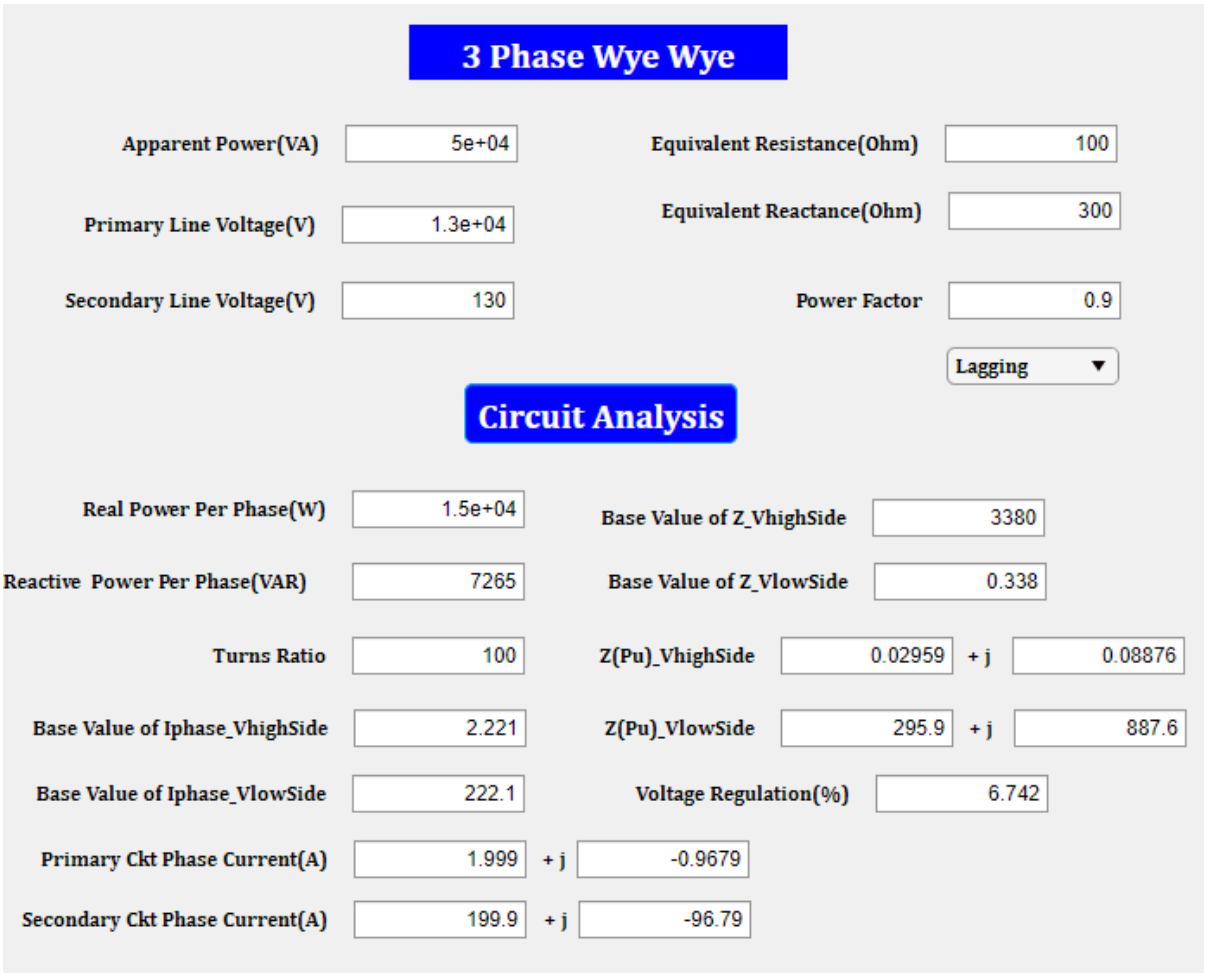
```
app.TurnsRatioEditField.Value=a;
%Base value in high voltage side
Iphase_high_vol=S/(3*Vphase_primary);
app.BaseValueofIphase_VhighSideEditField.Value=Iphase_high_vol;
Zphase_high_vol=3*(Vphase_primary^2)/S;
app.BaseValueofZ_VhighSideEditField.Value=Zphase_high_vol;
%Base value in low voltage side
Iphase_low_vol=S/(3*Vphase_secondary);
app.BaseValueofIphase_VlowSideEditField.Value=Iphase_low_vol;
Zphase_low_vol=3*(Vphase_secondary^2)/S;
app.BaseValueofZ_VlowSideEditField.Value=Zphase_low_vol;
%Impedence in pu in high voltage side
Zh_real=R/Zphase_high_vol;
Zh_imaginary=X/Zphase_high_vol;
app.ZPu_VhighSideEditField.Value=Zh_real;
app.EditField_2.Value=Zh_imaginary;
%Impedence in pu in low voltage side
Zl_real=R/Zphase_low_vol;
Zl_imaginary=X/Zphase_low_vol;
app.ZPu_VlowSideEditField.Value=Zl_real;
app.EditField2_2.Value=Zl_imaginary;
%Primary Phase current
if PFchar=="Leading"
   theta=acosd(PF);
elseif PFchar=="Lagging"
   theta=-acosd(PF);
else
```

```
  theta=acosd(1);
end
Ip=S/(3*Vphase_primary);
Ipx=Ip*cosd(theta);
Ipy=Ip*sind(theta);
Ipcomplex=Ipx+Ipy*j;
app.PrimaryCktPhaseCurrentAEditField.Value=Ipx;
app.EditField.Value=Ipy;
%Secondary Phase current
Is=S/(3*Vphase_secondary);
Isx=Is*cosd(theta);
Isy=Is*sind(theta);
Iscomplex=Isx+Isy*j;
app.SecondaryCktPhaseCurrentAEditField.Value=Isx;
app.EditField2.Value=Isy;
%Voltage Regulation
V_phi_P=a*Vphase_secondary+(R+X*j)*Ipcomplex;
VR_num=abs(V_phi_P)-a*Vphase_secondary;
VR_denom=a*Vphase_secondary;
VR=(VR_num/VR_denom)*100;
app.VoltageRegulationEditField.Value=VR;
       end
       end
```

Test Case:
Leading Power Factor:



Lagging Power Factor:



Unity Power Factor:

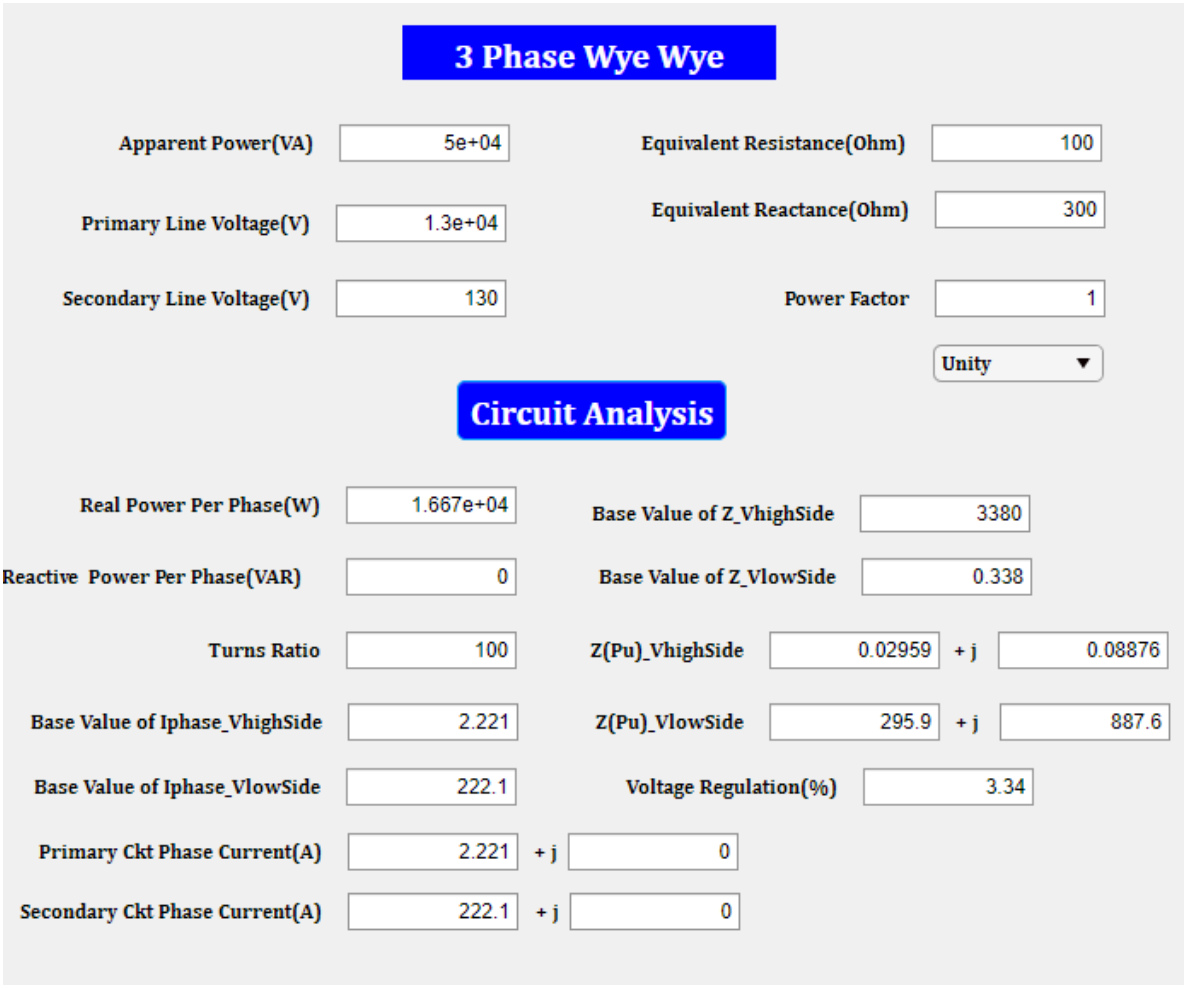


Figure 19: Test cases for 3 Phase Wye - Wye

c. Delta-Wye Connection:

Code:

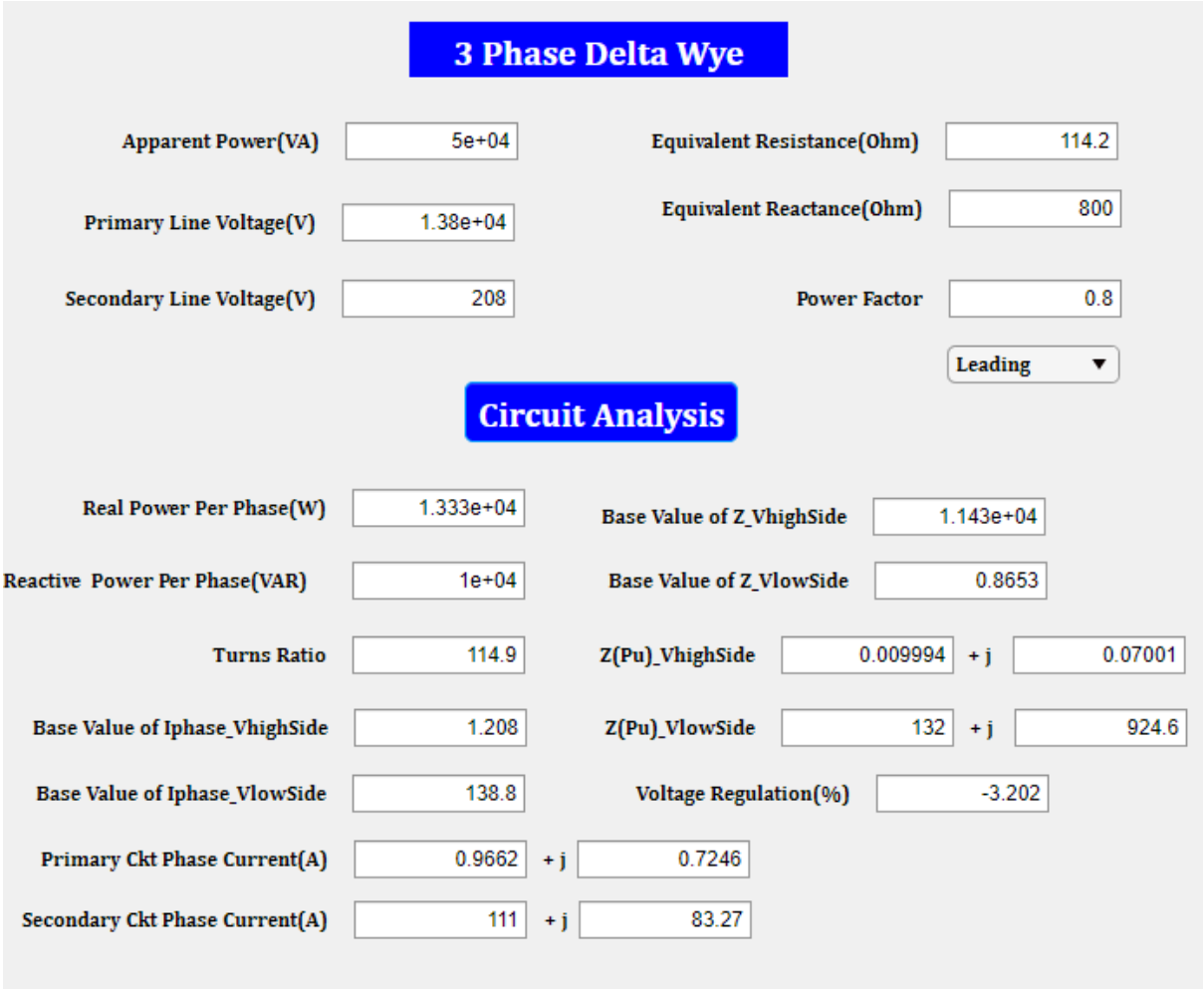
```
% Callbacks that handle component events

      methods (Access = private)


            % Button pushed function: CircuitAnalysisButton

            function CircuitAnalysisButtonPushed(app, event)

                  %Delta_wye

                  %Taking Inputs

                  S=app.ApparentPowerVAEditField.Value;

                  Vlp=app.PrimaryLineVoltageVEditField.Value;

                  Vls=app.SecondaryLineVoltageVEditField.Value;

                  PF=app.PowerFactorEditField.Value;

                  PFchar=app.DropDown.Value;

                  R=app.EquivalentResistanceOhmEditField.Value;

                  X=app.EquivalentReactanceOhmEditField.Value;

                  %Real power per phase

                  P=S*abs(PF);

                  P_PerPhase=P/3

                  app.RealPowerPerPhaseWEditField.Value=P_PerPhase;

                  %Reactive power per phase

                  Q=sqrt(S^2-P^2);

                  Q_PerPhase=Q/3

                  app.ReactivePowerPerPhaseVAREditField.Value=Q_PerPhase;


                  %Turns ratio

                  Vphase_primary=Vlp;

                  Vphase_secondary=Vls/sqrt(3);

                  a=Vphase_primary/Vphase_secondary;
```

```
                app.TurnsRatioEditField.Value=a;


                %Base value in high voltage side
                Iphase_high_vol=S/(3*Vphase_primary);
                app.BaseValueofIphase_VhighSideEditField.Value=Iphase_high_vo
                l;
                Zphase_high_vol=3*(Vphase_primary^2)/S;
                app.BaseValueofZ_VhighSideEditField.Value=Zphase_high_vol;
                %Base value in low voltage side
                Iphase_low_vol=S/(3*Vphase_secondary);
                app.BaseValueofIphase_VlowSideEditField.Value=Iphase_low_vol;
                Zphase_low_vol=3*(Vphase_secondary^2)/S;
                app.BaseValueofZ_VlowSideEditField.Value=Zphase_low_vol;


                %Impedence in pu in high voltage side
                Zh_real=R/Zphase_high_vol;
                Zh_imaginary=X/Zphase_high_vol;
                app.ZPu_VhighSideEditField.Value=Zh_real;
                app.EditField_2.Value=Zh_imaginary;


                %Impedence in pu in low voltage side
                Zl_real=R/Zphase_low_vol;
                Zl_imaginary=X/Zphase_low_vol;
                app.ZPu_VlowSideEditField.Value=Zl_real;
                app.EditField2_2.Value=Zl_imaginary;


                %Primary Phase current
```

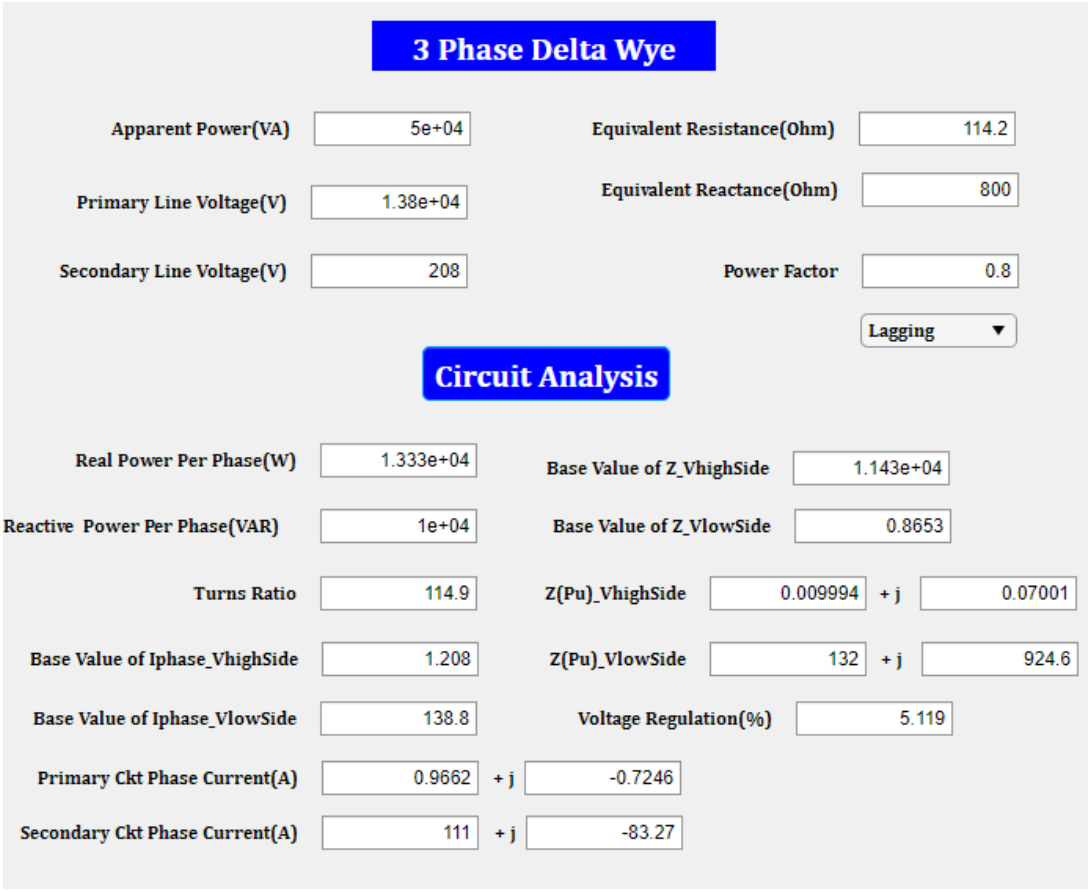
```
if PFchar=="Leading"
   theta=acosd(PF);
elseif PFchar=="Lagging"
   theta=-acosd(PF);
else
  theta=acosd(1);
end
Ip=S/(3*Vphase_primary);
Ipx=Ip*cosd(theta);
Ipy=Ip*sind(theta);
Ipcomplex=Ipx+Ipy*j;
app.PrimaryCktPhaseCurrentAEditField.Value=Ipx;
app.EditField.Value=Ipy;

%Secondary Phase current
Is=S/(3*Vphase_secondary);
Isx=Is*cosd(theta);
Isy=Is*sind(theta);
Iscomplex=Isx+Isy*j;
app.SecondaryCktPhaseCurrentAEditField.Value=Isx;
app.EditField2.Value=Isy;

%Voltage Regulation
V_phi_P=a*Vphase_secondary+(R+X*j)*Ipcomplex;
VR_num=abs(V_phi_P)-a*Vphase_secondary;
VR_denom=a*Vphase_secondary;
VR=(VR_num/VR_denom)*100;
```

```
                app.VoltageRegulationEditField.Value=VR;
            end
        end
```

Test Case:
Leading Power Factor:



Lagging Power Factor:



Unity Power Factor:

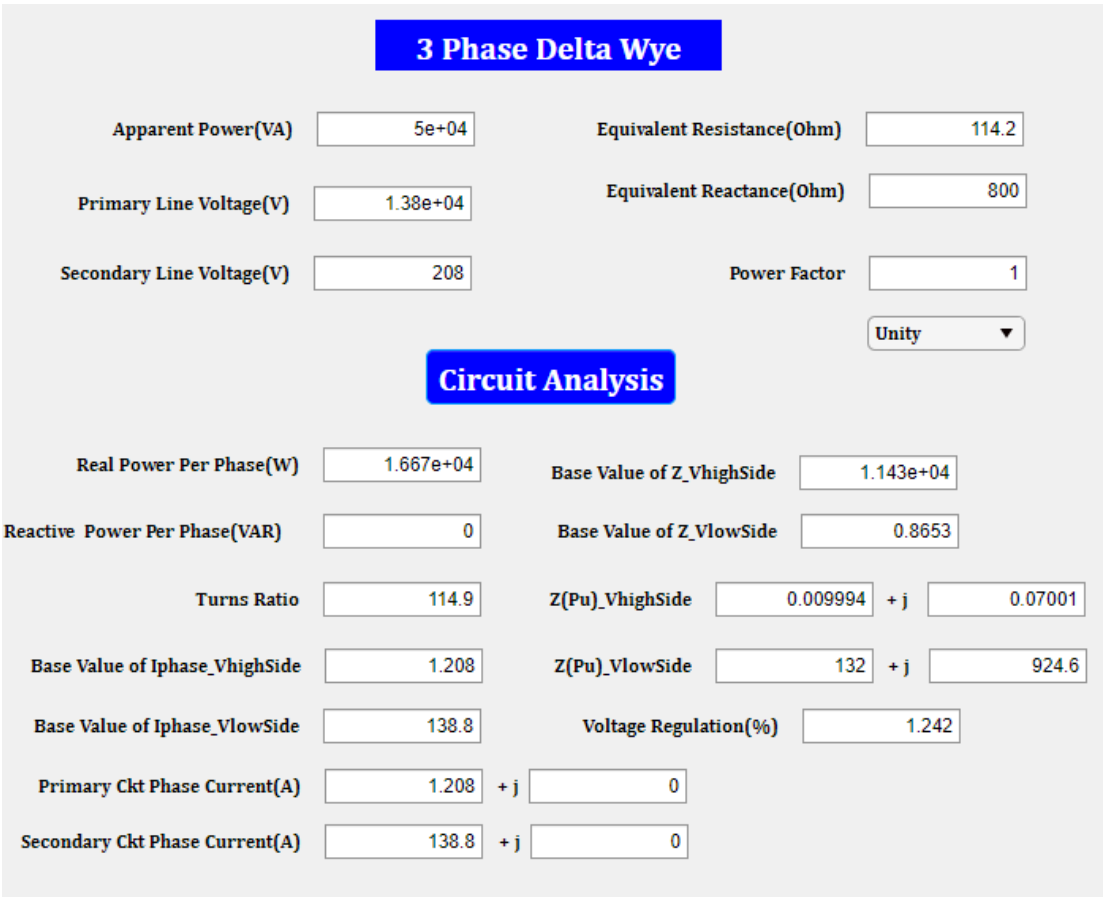


Figure 20: Test cases for 3 Phase Delta -Wye

d. Delta-Delta Connection:

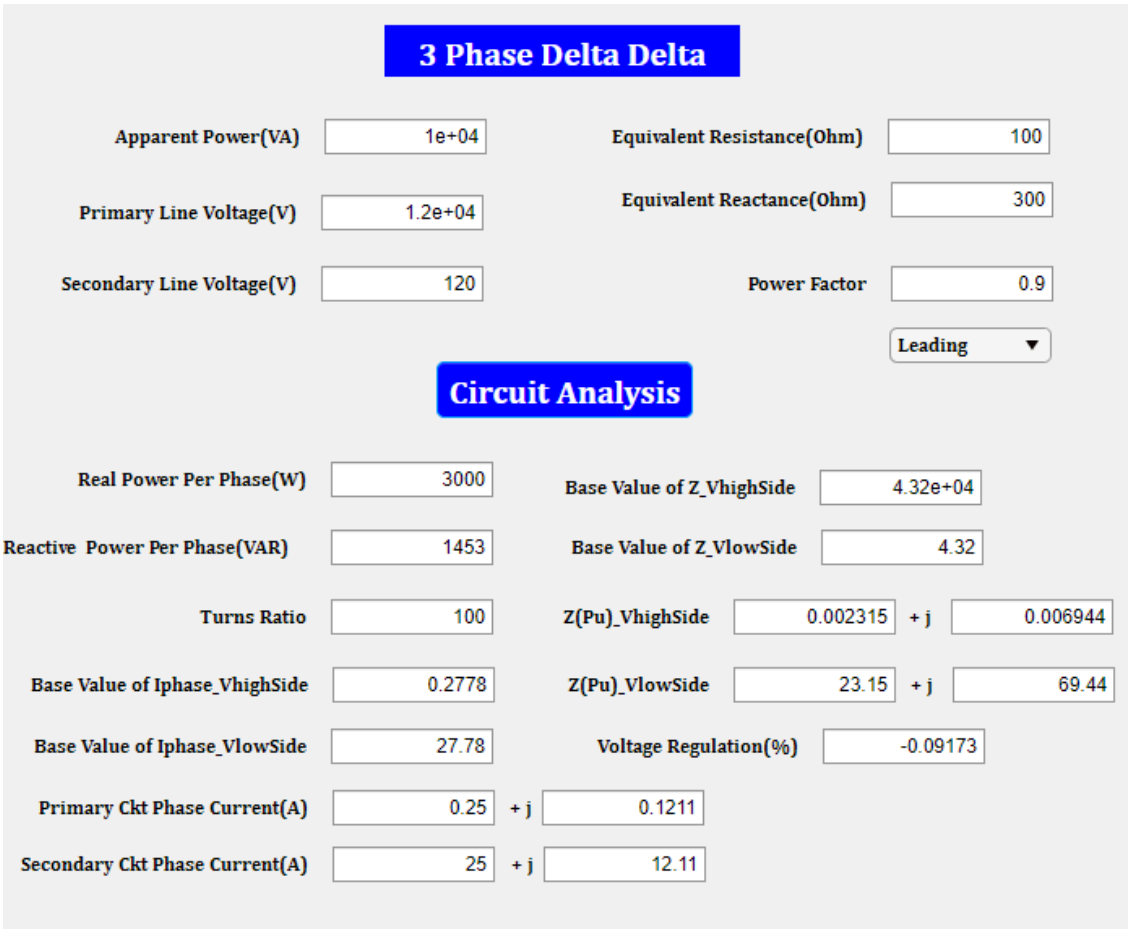
Code:

```
% Callbacks that handle component events

      methods (Access = private)


            % Button pushed function: CircuitAnalysisButton

            function CircuitAnalysisButtonPushed(app, event)

                  %Delta_Delta

                  %Taking Inputs

                  S=app.ApparentPowerVAEditField.Value;

                  Vlp=app.PrimaryLineVoltageVEditField.Value;

                  Vls=app.SecondaryLineVoltageVEditField.Value;

                  PF=app.PowerFactorEditField.Value;

                  PFchar=app.DropDown.Value;

                  R=app.EquivalentResistanceOhmEditField.Value;

                  X=app.EquivalentReactanceOhmEditField.Value;

                  %Real power per phase

                  P=S*abs(PF);

                  P_PerPhase=P/3

                  app.RealPowerPerPhaseWEditField.Value=P_PerPhase;

                  %Reactive power per phase

                  Q=sqrt(S^2-P^2);

                  Q_PerPhase=Q/3

                  app.ReactivePowerPerPhaseVAREditField.Value=Q_PerPhase;

                  %Turns ratio

                  Vphase_primary=Vlp;
```

```
                Vphase_secondary=Vls;
                a=Vphase_primary/Vphase_secondary;
                app.TurnsRatioEditField.Value=a;
                %Base value in high voltage side
                Iphase_high_vol=S/(3*Vphase_primary);
                app.BaseValueofIphase_VhighSideEditField.Value=Iphase_high_vol;
                Zphase_high_vol=3*(Vphase_primary^2)/S;
                app.BaseValueofZ_VhighSideEditField.Value=Zphase_high_vol;
                %Base value in low voltage side
                Iphase_low_vol=S/(3*Vphase_secondary);
                app.BaseValueofIphase_VlowSideEditField.Value=Iphase_low_vol;
                Zphase_low_vol=3*(Vphase_secondary^2)/S;
                app.BaseValueofZ_VlowSideEditField.Value=Zphase_low_vol;
                %Impedence in pu in high voltage side
                Zh_real=R/Zphase_high_vol;
                Zh_imaginary=X/Zphase_high_vol;
                app.ZPu_VhighSideEditField.Value=Zh_real;
                app.EditField_2.Value=Zh_imaginary;
                %Impedence in pu in low voltage side
                Zl_real=R/Zphase_low_vol;
                Zl_imaginary=X/Zphase_low_vol;
                app.ZPu_VlowSideEditField.Value=Zl_real;
                app.EditField2_2.Value=Zl_imaginary;
                %Primary Phase current
                if PFchar=="Leading"
                   theta=acosd(PF);
                elseif PFchar=="Lagging"
```

```
                    theta=-acosd(PF);
                else
                  theta=acosd(1);
                end
                Ip=S/(3*Vphase_primary);
                Ipx=Ip*cosd(theta);
                Ipy=Ip*sind(theta);
                Ipcomplex=Ipx+Ipy*j;
                app.PrimaryCktPhaseCurrentAEditField.Value=Ipx;
                app.EditField.Value=Ipy;
                %Secondary Phase current
                Is=S/(3*Vphase_secondary);
                Isx=Is*cosd(theta);
                Isy=Is*sind(theta);
                Iscomplex=Isx+Isy*j;
                app.SecondaryCktPhaseCurrentAEditField.Value=Isx;
                app.EditField2.Value=Isy;
                %Voltage Regulation
                V_phi_P=a*Vphase_secondary+(R+X*j)*Ipcomplex;
                VR_num=abs(V_phi_P)-a*Vphase_secondary;
                VR_denom=a*Vphase_secondary;
                VR=(VR_num/VR_denom)*100;
                app.VoltageRegulationEditField.Value=VR;
            end
        end
```

Test Case:
Leading Power Factor:



Unity Power Factor:

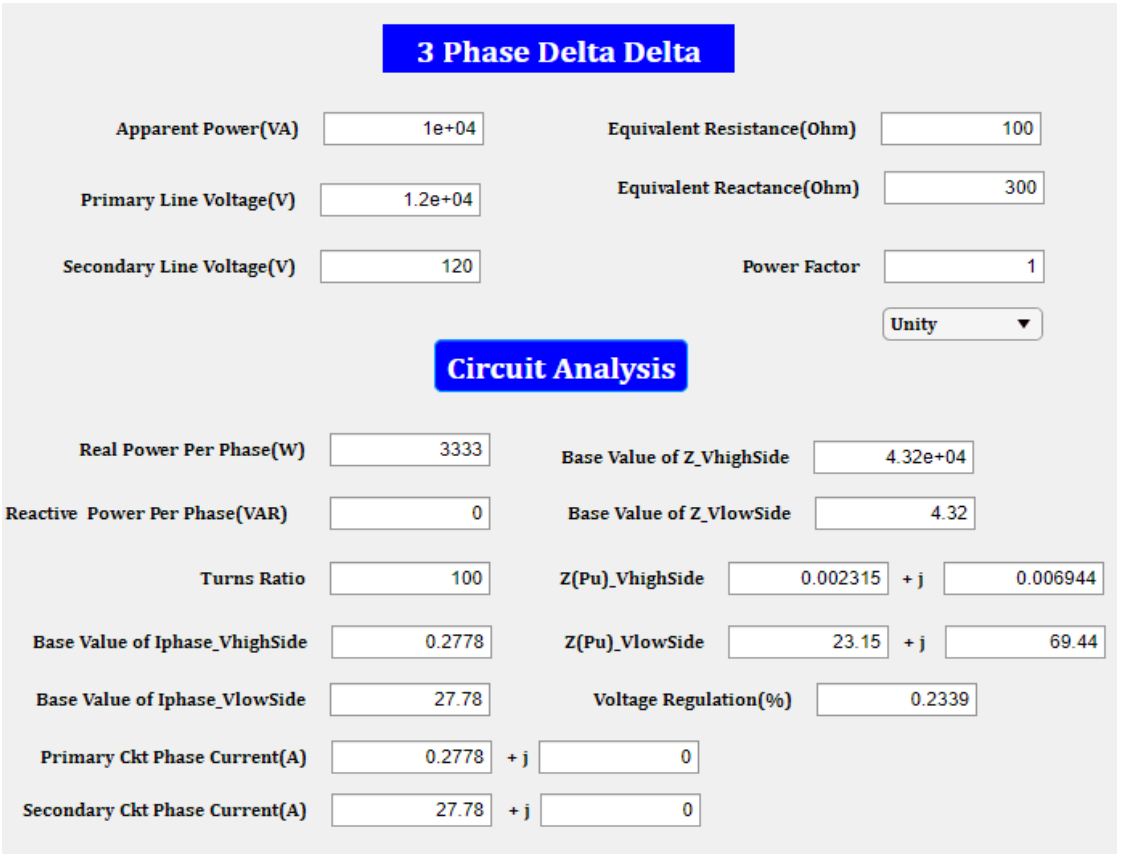


Figure 21: Test cases for 3 Phase Delta - Delta

4.4. Auto-Transformer:

Code:

```
    % Callbacks that handle component events
      methods (Access = private)

            % Code that executes after component creation
            function startupFcn(app)
                  app.UIFigure.Name = 'Xer_Auto';
            end

            % Button pushed function: CircuitAnalysisButton
            function CircuitAnalysisButtonPushed(app, event)
                  %Auto_Transformer
                  %Vlow
                  Vl=app.VoltageinCommonWindingVEditField.Value;
                  app.VoltageofVlowSideVEditField.Value=Vl;
                  Nc=app.NumberofTurninCommonWindingEditField.Value;
                  Nse=app.NumberofTurninSeriesWindingEditField.Value;

                  %Vhigh
                  Vh=Vl*(Nse+Nc)/Nc;
                  app.VoltageofVhighSideVEditField.Value=Vh;

                  %Ilow and Ihigh
                  Ic=app.CurrentinCommonWindingAEditField.Value;
                  Ise=(Nc*Ic)/Nse;
                  Il=Ic+Ise;
                  Ih=Ise;
                  app.CurrentofVhighSideAEditField.Value=Ih;
                  app.CurrentofVlowSideAEditField.Value=Il;
```

```
                %Apparent Output Power
                Sw=app.ApparentPowerofWindingVAEditField.Value;
                So=Sw*(Nse+Nc)/Nse;
                app.OuputApparentPowerVAEditField.Value=So;

                %Rating Advantage
                Rating=So/Sw;
                app.PowerRatingAdvantageEditField.Value=Rating;
            end
        end
```

Test Case:

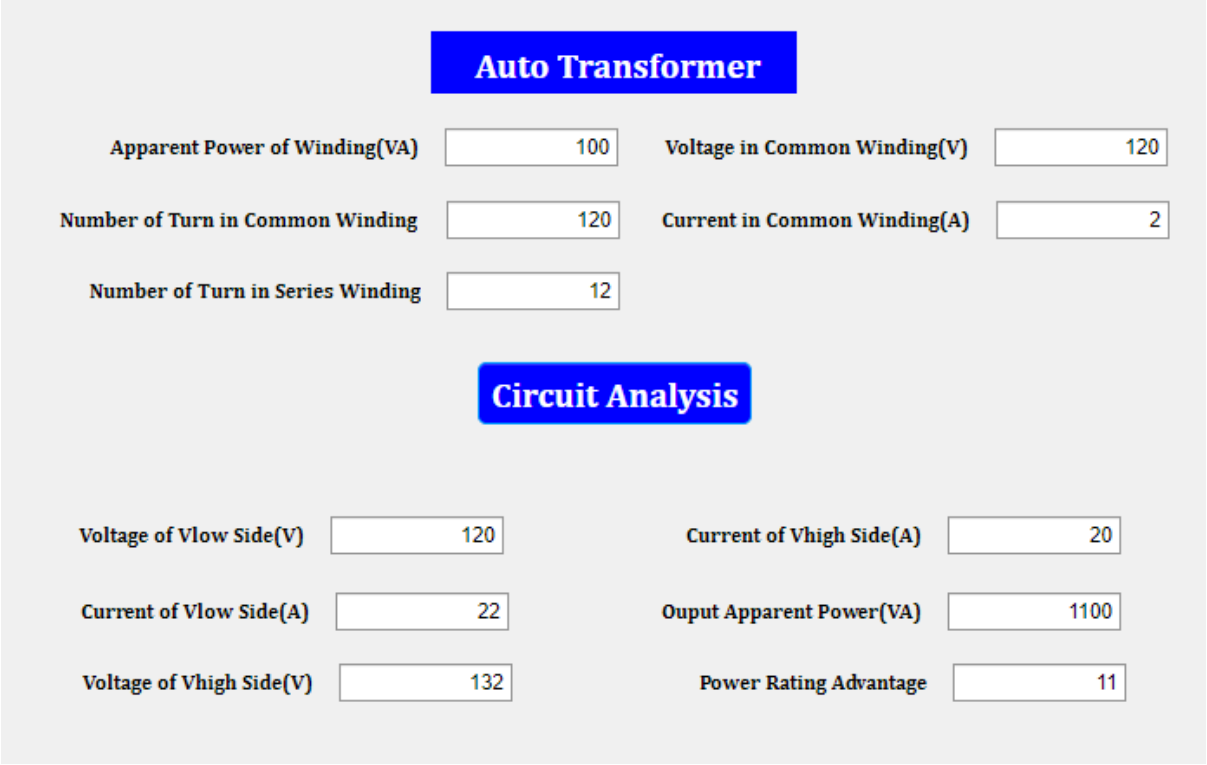


Figure 22: Test cases for Auto Transformer

4.5. Torque-Speed Characteristics:

pseudocode:

```
1. Start the app
2. Being the startup Function called,
   Store  some default values to global variables
```

```
3. With the “Plot button” pushed callback
        if some value has provided
              Update variables
              Call the functions to plot
                    Plot 4 graphs
        else
              Use default values
              Call the functions to plot
                    Plot 4 graphs
        end if
4. With the “Inspect TS Curve” being pushed
        Open “IM_TS_Region” app
5. if end callback
        End the app
   else
        Go to step 3
```

Code:

```
properties(Access = public)
        n_sync = 1800; %synchronus speed in rev/min
        Frequency = 50;
        Pole = 2;
        V_TH = 255.2;
        R_TH = 0.590;
        X_TH = 1.106;
  end
  methods (Access = private)

        function plot_BR(app,UIAxes, V_p, R_1, X_1, R_2, X_2, X_M)
              s= [0: 0.01: 1];
              n_m = (1-s)*app.n_sync;

              V_th = V_p*(1i*X_M)/(R_1 + 1i*(X_1 +X_M));
              for i = 1: length(n_m)
                  Z(i) = app.R_TH + 1i*app.X_TH + R_2/s(i) + 1i*X_2;
              I_2(i) = V_th/Z(i);
                  BR(i) = abs(I_2(i));
              end


             plot(UIAxes, n_m, BR);
        end

        function plot_PF(app,UIAxes,R_2, X_2)
              s= [0: 0.01: 1];
              n_m = (1-s)*app.n_sync;
              PF = cos(atan(s*X_2/R_2));
             plot(UIAxes,n_m,PF);
        end

        function plot_BN(app,UIAxes, V_p, R_1, X_1, R_2, X_2, X_M)
              s= [0: 0.01: 1];
              n_m = (1-s)*app.n_sync;
```

```
                V_th = V_p*(1i*X_M)/(R_1 + 1i*(X_1 +X_M));
                Z = app.R_TH + 1i*app.X_TH + (R_2/s(2)) + 1i*X_2;
                I_2 = V_th/Z;
                BN = abs(I_2*((R_2/s(2)) + 1i*X_2))*ones(1, length(n_m));

               plot(UIAxes, n_m, BN);
        end

        function plot_T(app, UIAxes, R_2, X_2)
               s= [0: 0.01: 1];
               n_m = (1-s)*app.n_sync;
               w_sync = 2*pi*app.n_sync/60;
               Tau = zeros(1, length(n_m));
               for i = 1: length(n_m)
                   Tau(i) = (3*(app.V_TH)^2*R_2)/(w_sync*s(i)*((app.R_TH
+(R_2/s(i)))^2 + (app.X_TH +X_2)^2));

                    if n_m(i) == app.n_sync

                      Tau(i) = 0;
                    end
               end

               plot(UIAxes, n_m, Tau);
        end

    end


    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
              app.UIFigure.Name = 'IM_Torque_Speed';
        end

        % Button pushed function: PlotButton
        function PlotButtonPushed(app, event)
               a_eff = app.a_effEditField.Value;
             app.Frequency = app.FrequencyKnob.Value;
              app.Pole = app.PoleSpinner.Value;
             app.n_sync = 120*app.Frequency/app.Pole;

              V_p = app.V_pEditField.Value/sqrt(3);
              R_1 = app.R_1EditField.Value;
              X_1 = app.X_1EditField.Value;
            R_C = app.R_CEditField.Value;
              X_M = app.X_MEditField.Value;
              R_2 = (a_eff^2)*(app.R_2EditField.Value);
              X_2 = (a_eff^2)*(app.X_2EditField.Value);

         app.V_TH = V_p*X_M/sqrt((R_1)^2+(X_1 + X_M)^2);
               Z_TH = 1i*X_M*(R_1 + 1i*X_1)/(R_1 + 1i*(X_1 + X_M));

               app.R_TH = real(Z_TH);
               app.X_TH = imag(Z_TH);
```

```
            plot_BR(app,app.UIAxes_BR, V_p, R_1, X_1, R_2, X_2, X_M);
            plot_PF(app,app.UIAxes_PF,R_2, X_2);
            plot_BN(app,app.UIAxes_BN, V_p, R_1, X_1, R_2, X_2, X_M);
            plot_T(app, app.UIAxes_T,R_2,X_2);

        end

        % Button pushed function: InspectTSCurveButton
        function InspectTSCurveButtonPushed(app, event)
            IM_TS_Region();
        end

        % Value changing function: FrequencyKnob
        function FrequencyKnobValueChanging(app, event)
            app.Frequencyvalue.Value = event.Value;

        end

        % Value changed function: Frequencyvalue
        function FrequencyvalueValueChanged(app, event)
            app.FrequencyKnob.Value = app.Frequencyvalue.Value;

        end
    end
```

Test Case:

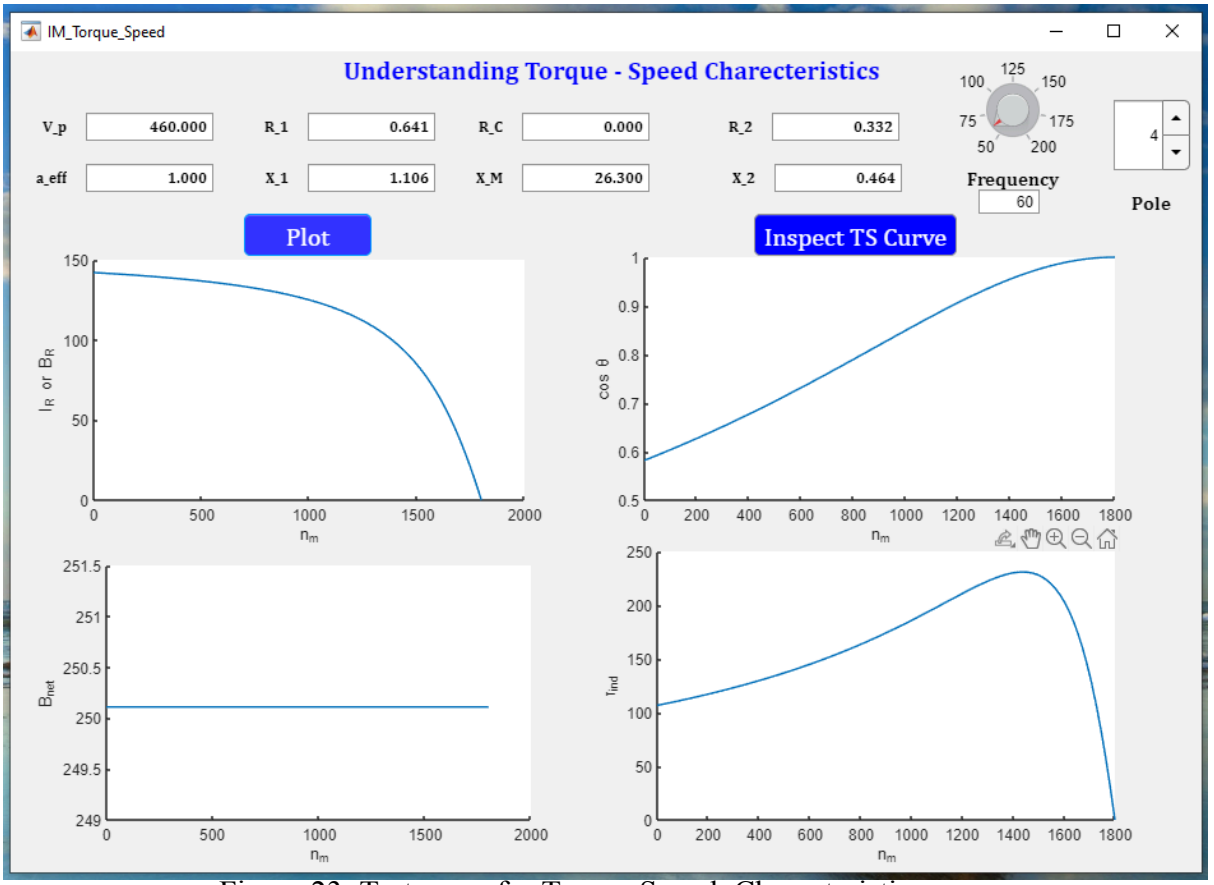


Figure 23: Test cases for Torque Speed Characteristics

4.6. Different TS Regions

pseudocode:

```
1. Start the app
2. With the Startup Function being called
        Store some default values to global variables
3. if the “Plot button” being pushed
        if some value have been given
              Update the variables
              Plot the graph
        else
             Use default values
             Plot the graph
        end if

   elseif “R_2 Slider is pushed”
        Update the variables
        Plot the graph
   end if

4. if “FrequencyKnob is rotated”
        Go to step 3
5. if “Pole Spinner value changed
        Go to step 3
6. With the end button pushed
        end the app
```

Codes:

```
    properties (Access = private)
            tempapp % Description
            V_TH;
            R_TH;
            X_TH;
            Frequency;
            Pole;
      end

      methods (Access = private)

            function plot_TS(app, Axes, R_2,X_2)
                  cla(Axes);
                  app.Frequency = app.FrequencyKnob.Value;
                  app.Pole = app.PoleSpinner.Value;
                   n_sync = 120*app.Frequency/app.Pole;
                   s = [-1:0.01:1.5];
                   n_m = (1-s)*n_sync;
                   w_sync = 2*pi*n_sync/60;

                   Tau = zeros(1, length(n_m));
                   for i = 1: length(n_m)
                       Tau(i) = (3*(app.V_TH)^2*R_2/(w_sync*s(i)*((app.R_TH
      +(R_2/s(i)))^2 + (app.X_TH + X_2)^2)));
                     if n_m(i) == n_sync
                           Tau(i) = 0;
                   end
                end
                  plot(Axes, n_m, Tau);
                  hold(Axes, 'on');

                   s_max = (R_2)/ (sqrt((app.R_TH)^2 + (app.X_TH + X_2)^2));
                   n_mmax = (1- s_max)*n_sync;
                   Tau_max = (3*(app.V_TH)^2)/(2*w_sync*(app.R_TH
      +sqrt((app.R_TH)^2 + (app.X_TH + X_2)^2)));
                  plot(Axes, n_m, Tau_max*ones(1, length(n_m)), '--');
                  plot(Axes, n_mmax, Tau_max, 'o');
                  plot(Axes, n_mmax*ones(1, length(n_m)),Tau, '--');
                  hold(Axes, 'off');

            end
      end


      % Callbacks that handle component events
      methods (Access = private)

            % Code that executes after component creation
            function startupFcn(app)
                  app.UIFigure.Name = 'IM_TS_Region'; %changing the app name

                   %importing values from IM_Torque_Speed app

                  app.tempapp = IM_Torque_Speed;

                  app.V_TH = app.tempapp.V_TH;
                  app.R_TH = app.tempapp.R_TH;
```

```
                app.X_TH = app.tempapp.X_TH;

                app.V_THEditField.Value = app.V_TH;
                app.R_THEditField.Value = app.R_TH;
                app.X_THEditField.Value = app.X_TH;
                delete(app.tempapp.UIFigure); %deleting the instance of
      IM_Torque_Speed app
            end

            % Value changing function: FrequencyKnob
            function FrequencyKnobValueChanging(app, event)
                app.Frequencyvalue.Value = event.Value;

            end

            % Value changed function: Frequencyvalue
            function FrequencyvalueValueChanged(app, event)
                app.FrequencyKnob.Value = app.Frequencyvalue.Value;

            end

            % Callback function: PlotButton, R_2timesR_0Slider
            function PlotButtonPushed(app, event)

                app.V_TH = app.V_THEditField.Value;
                app.R_TH = app.R_THEditField.Value;
                app.X_TH = app.X_THEditField.Value;
                 R_2 = app.R_2EditField.Value*app.R_2timesR_0Slider.Value;
                 X_2 = app.X_2EditField.Value;

                plot_TS(app,app.UIAxes, R_2, X_2);


                app.GeneratorRegionLabel.Visible = 'on';
                app.MotorRegionLabel.Visible = 'on';
                app.BrakingRegionLabel.Visible = 'on';
            end
      end
```

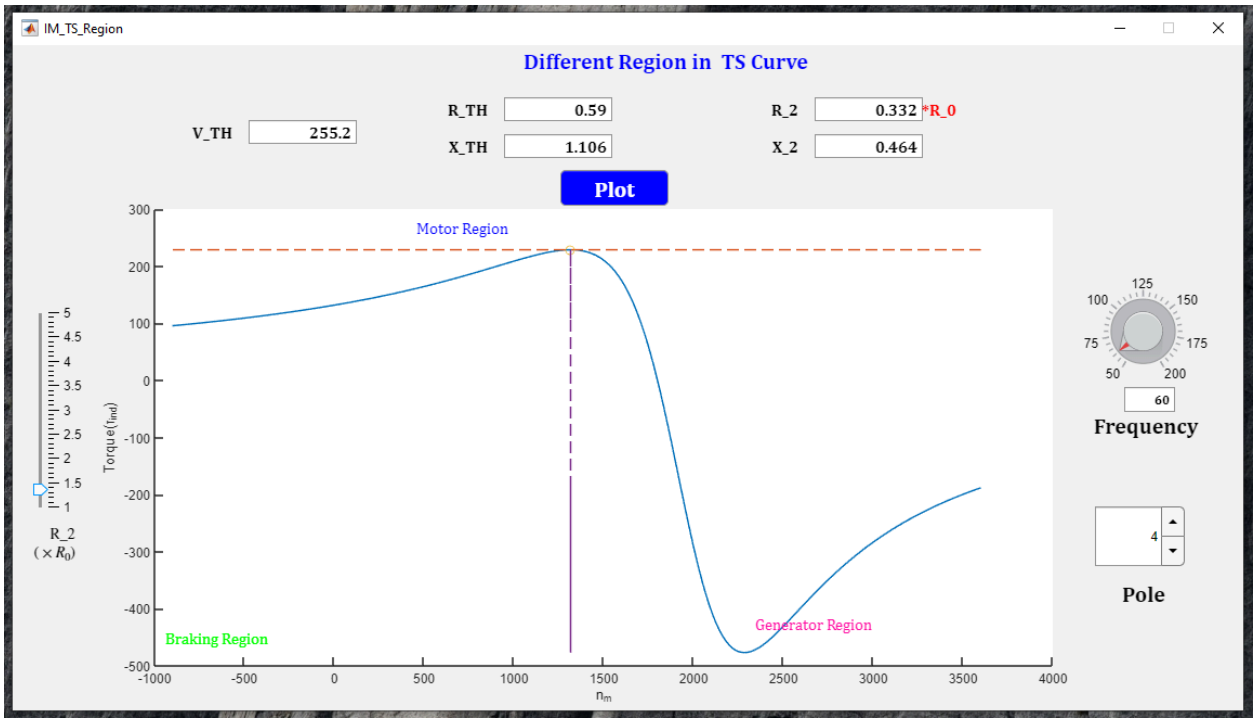


Figure 24: Test cases for Different Regions in TS Curve

4.7. Speed - Control Techniques:

Codes:

```
    properties (Access = public)
            % Description
            tempapp;
            V_TH;
            R_TH;
            X_TH;
            Frequency;
            Pole = 4;
            state = 0;
      end

      methods (Access = public)

            function plot_single_graph(app)
                  cla(app.UIAxes);
                  V_TH = app.V_THEditField.Value;
                  R_TH = app.R_THEditField.Value;
                  X_TH = app.X_THEditField.Value;
                  R_2 = app.R_2EditField.Value;
                  X_2 = app.X_2EditField.Value;
                   app.Frequency = app.FrequencyKnob.Value;
                  Frequency = app.Frequency;
                  Pole = app.Pole;

                  %plotting a single graph
```

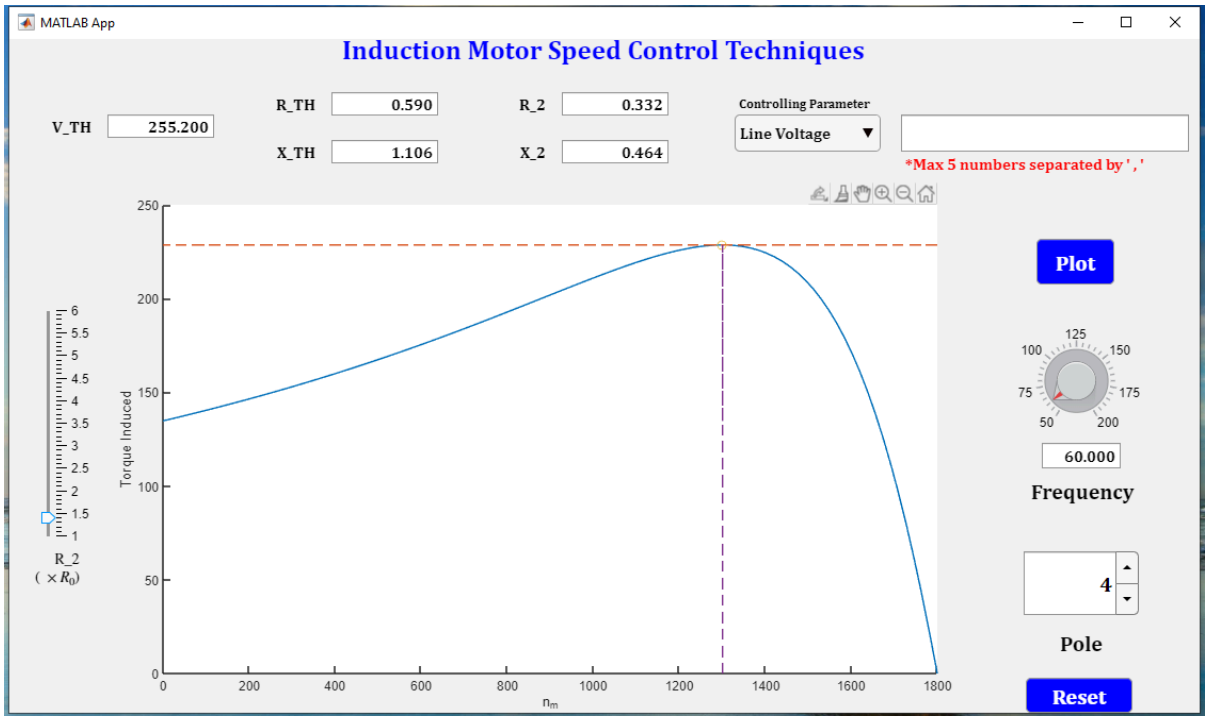
```
                n_sync = 120*Frequency/Pole;
                s= [0: 0.01: 1];
                n_m = (1-s)*n_sync;
                 w_sync = 2*pi*n_sync/60;
                Tau = zeros(1, length(n_m));
                for i = 1: length(n_m)
                    Tau(i) = (3*V_TH^2*R_2)/(w_sync*s(i)*((R_TH
    +(R_2/s(i)))^2 + (X_TH +X_2)^2));
                if n_m(i) == n_sync
                        Tau(i) =0;
                end
                end

                plot(app.UIAxes, n_m, Tau);
                hold(app.UIAxes, 'on');
                 s_max = R_2/ sqrt(app.R_TH^2 + (app.X_TH + X_2)^2);
                 n_mmax = (1- s_max)*n_sync;
                 Tau_max = (3*(V_TH)^2)/(2*w_sync*(R_TH +sqrt((R_TH)^2 +
    (X_TH + X_2)^2)));
                plot(app.UIAxes,n_m, Tau_max*ones(1, length(n_m)), '--');
                plot(app.UIAxes, n_mmax, Tau_max, 'o');
                plot(app.UIAxes, n_mmax*ones(1, length(n_m)),Tau, '--');
                hold(app.UIAxes, 'off');
          end
          end

          methods (Access = private)

          function plot_graph_VR(app, V_TH, R_TH, X_TH, R_2, X_2,
    legend_value);
                app.Frequency = app.FrequencyKnob.Value;
                Frequency = app.Frequency;
                 Pole = app.Pole;

                 %plotting a single graph
                 n_sync = 120*Frequency/Pole;
                 s= [0: 0.01: 1];
                 n_m = (1-s)*n_sync;
                 w_sync = 2*pi*n_sync/60;
                 Tau = zeros(1, length(n_m));
                 for i = 1: length(n_m)
                     Tau(i) = (3*V_TH^2*R_2)/(w_sync*s(i)*((R_TH
    +(R_2/s(i)))^2 + (X_TH +X_2)^2));
                 if n_m(i) == n_sync
                         Tau(i) =0;
                 end
                 end

                plot(app.UIAxes, n_m, Tau, "DisplayName",legend_value);
          end

          function plot_graph_Fre(app, V_TH, R_TH, X_TH, R_2, X_2,
    Frequency, legend_value)
                 Pole = app.Pole;

                 %plotting a single graph
                 n_sync = 120*Frequency/Pole;
```

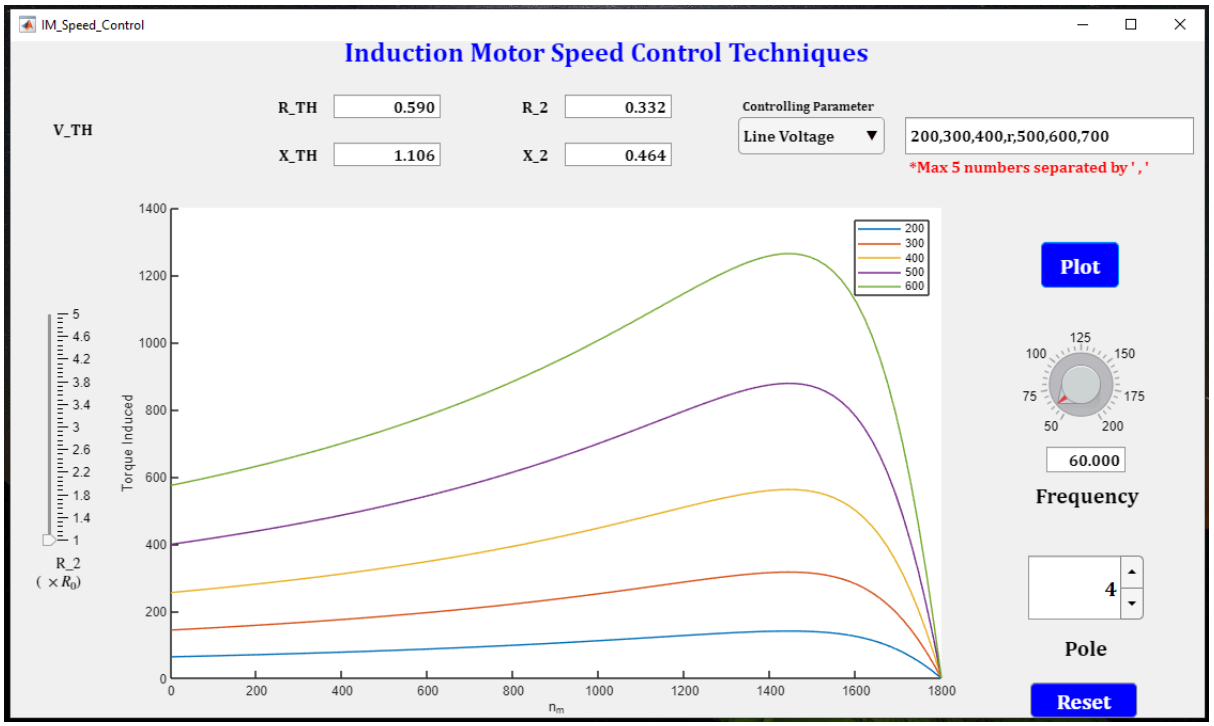
```
                s= [0: 0.01: 1];
                n_m = (1-s)*n_sync;
                w_sync = 2*pi*n_sync/60;
                Tau = zeros(1, length(n_m));
                for i = 1: length(n_m)
                    Tau(i) = (3*V_TH^2*R_2)/(w_sync*s(i)*((R_TH
    +(R_2/s(i)))^2 + (X_TH +X_2)^2));
                if n_m(i) == n_sync
                        Tau(i) =0;
                end
                end

                plot(app.UIAxes, n_m, Tau, "DisplayName", legend_value);
        end
        end


        % Callbacks that handle component events
        methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
              app.UIFigure.Name = 'IM_Speed_Control';
              app.tempapp = IM_Torque_Speed;

              app.V_TH = app.tempapp.V_TH;
              app.R_TH = app.tempapp.R_TH;
              app.X_TH = app.tempapp.X_TH;
              app.V_THEditField.Value = app.V_TH;
              app.R_THEditField.Value = app.R_TH;
              app.X_THEditField.Value =app.X_TH;

              delete(app.tempapp);
        end

        % Value changing function: FrequencyKnob
        function FrequencyKnobValueChanging(app, event)
              app.Frequency = event.Value;
              app.frequencyvalue.Value = event.Value;
        end

        % Button pushed function: PlotButton
        function PlotButtonPushed(app, event)
              cla(app.UIAxes);
              if app.param_list.Value == ""
                  app.state = 0;
              else
                  app.state = 1;
              end


              if app.state ==1
                  value = app.DropDown.Value;
              switch value
                      case 'Line Voltage'
                        app.V_THEditField.Visible = 'off';
                           str = app.param_list.Value;
```

```
                            str(~isstrprop(str,'digit')) = ' '; %replace
    non-numeric characters with empty space
                            V_TH = str2double(strsplit(strtrim(str)));
                            R_2 = app.R_2EditField.Value;
                            X_2 = app.X_2EditField.Value;
                            n = length(V_TH);
                            if n >= 5
                             for i = 1:5
                                 if length(V_TH) ~=0
                                        plot_graph_VR(app, V_TH(i),
    app.R_TH, app.X_TH, R_2, X_2, num2str(V_TH(i)));
                                        hold(app.UIAxes,'on');
                                    end
                             end
                            elseif n == 0
                            else
                             for i = 1:n
                                   if length(V_TH) ~=0
                                        plot_graph_VR(app, V_TH(i),
    app.R_TH, app.X_TH, R_2, X_2, num2str(V_TH(i)));
                                        hold(app.UIAxes,'on');

                                    end
                             end
                           end
                            legend(app.UIAxes);
                            hold(app.UIAxes, "off");

                        case 'Frequency'
                            app.FrequencyKnob.Visible = 'off';
                            str = app.param_list.Value;
                            str(~isstrprop(str,'digit')) = ' '; %replace
    non-numeric characters with empty space
                            Frequency = str2double(strsplit(strtrim(str)));
                            R_2 = app.R_2EditField.Value;
                            X_2 = app.X_2EditField.Value;
                            n = length(Frequency);
                            if n >= 5
                             for i = 1:5
                                    plot_graph_Fre(app, app.V_TH, app.R_TH,
    app.X_TH, R_2, X_2, Frequency(i), num2str(Frequency(i)));
                                    hold(app.UIAxes,'on');
                             end
                            elseif n == 0

                            else
                             for i = 1:n
                                    plot_graph_Fre(app, app.V_TH, app.R_TH,
    app.X_TH, R_2, X_2, Frequency(i), num2str(Frequency(i)));
                                    hold(app.UIAxes,'on');
                                end
                            end
                            legend(app.UIAxes);
                            hold(app.UIAxes, "off");
                        case 'Load Resistance'
                            app.R_2EditField.Visible = 'off';
                              str = app.param_list.Value;
```

```
                            R_2 = str2num(str);
                            X_2 = app.X_2EditField.Value;
                            n = length(R_2);
                            if n >= 5
                             for i = 1:5
                                    plot_graph_VR(app, app.V_TH, app.R_TH,
app.X_TH, R_2(i), X_2, num2str(R_2(i)));
                                    hold(app.UIAxes,'on');
                             end
                            elseif n == 0

                            else
                             for i = 1:n
                                    plot_graph_VR(app, app.V_TH, app.R_TH,
app.X_TH, R_2(i), X_2,num2str(R_2(i)));
                                    hold(app.UIAxes,'on');
                             end
                            end
                            legend(app.UIAxes);
                            hold(app.UIAxes, "off");
                end



                elseif app.state == 0
                    plot_single_graph(app);

                end
        end

        % Callback function: FrequencyKnob, R_2timesR_0Slider
        function R_2timesR_0SliderValueChanging(app, event)
                R_2 = app.R_2EditField.Value*app.R_2timesR_0Slider.Value;
               cla(app.UIAxes);
                 V_TH = app.V_THEditField.Value;
                R_TH = app.R_THEditField.Value;
                X_TH = app.X_THEditField.Value;
                X_2 = app.X_2EditField.Value;
               app.Frequency = app.FrequencyKnob.Value;
               Frequency = app.Frequency;
                Pole = app.Pole;

                %plotting a single graph
                n_sync = 120*Frequency/Pole;
                s= [0: 0.01: 1];
                n_m = (1-s)*n_sync;
             w_sync = 2*pi*n_sync/60;
                Tau = zeros(1, length(n_m));
                for i = 1: length(n_m)
                    Tau(i) = (3*V_TH^2*R_2)/(w_sync*s(i)*((R_TH
+(R_2/s(i)))^2 + (X_TH +X_2)^2));
                if n_m(i) == n_sync
                        Tau(i) =0;
                end
                end
```

```
                plot(app.UIAxes, n_m, Tau);
                hold(app.UIAxes, 'on');
                 s_max = R_2/ sqrt(app.R_TH^2 + (app.X_TH + X_2)^2);
                 n_mmax = (1- s_max)*n_sync;
                 Tau_max = (3*(V_TH)^2)/(2*w_sync*(R_TH +sqrt((R_TH)^2 +
    (X_TH + X_2)^2)));
                plot(app.UIAxes,n_m, Tau_max*ones(1, length(n_m)), '--');
                plot(app.UIAxes, n_mmax, Tau_max, 'o');
                plot(app.UIAxes, n_mmax*ones(1, length(n_m)),Tau, '--');
                 hold(app.UIAxes, 'off');

            end

            % Value changed function: frequencyvalue
            function frequencyvalueValueChanged(app, event)
                app.FrequencyKnob.Value = app.frequencyvalue.Value;
            end

            % Value changed function: PoleSpinner
            function PoleSpinnerValueChanged(app, event)
                app.Pole = app.PoleSpinner.Value;
            end

            % Button pushed function: ResetButton
            function ResetButtonPushed(app, event)
                app.R_2EditField.Visible = 'on';
                app.FrequencyKnob.Visible = 'on';
                 app.V_THEditField.Visible = 'on';
                app.param_list.Value = '';
                app.R_2EditField.Value = 0.332;
                app.X_2EditField.Value = 0.464;
                app.frequencyvalue.Value = 60;
                app.FrequencyKnob.Value = app.frequencyvalue.Value;

                cla(app.UIAxes);
                legend(app.UIAxes, 'off');
                startupFcn(app);

            end
    end
```


MATLAB App
Induction Motor Speed Control Techniques
V_TH
255.200
R_TH
0.590
X_TH
1.106
R_2
0.332
X_2
0.464
Controlling Parameter
Line Voltage
*Max 5 numbers separated by ' , '
Plot
60.000
Frequency
4
Pole
Reset
Torque Induced
R_2
(× R0)
nm


IM_Speed_Control
Induction Motor Speed Control Techniques
V_TH
R_TH
0.590
X_TH
1.106
R_2
0.332
X_2
0.464
Controlling Parameter
Line Voltage
200,300,400,r,500,600,700
*Max 5 numbers separated by ' , '
Plot
60.000
Frequency
4
Pole
Reset
Torque Induced
R_2
(× R0)
nm

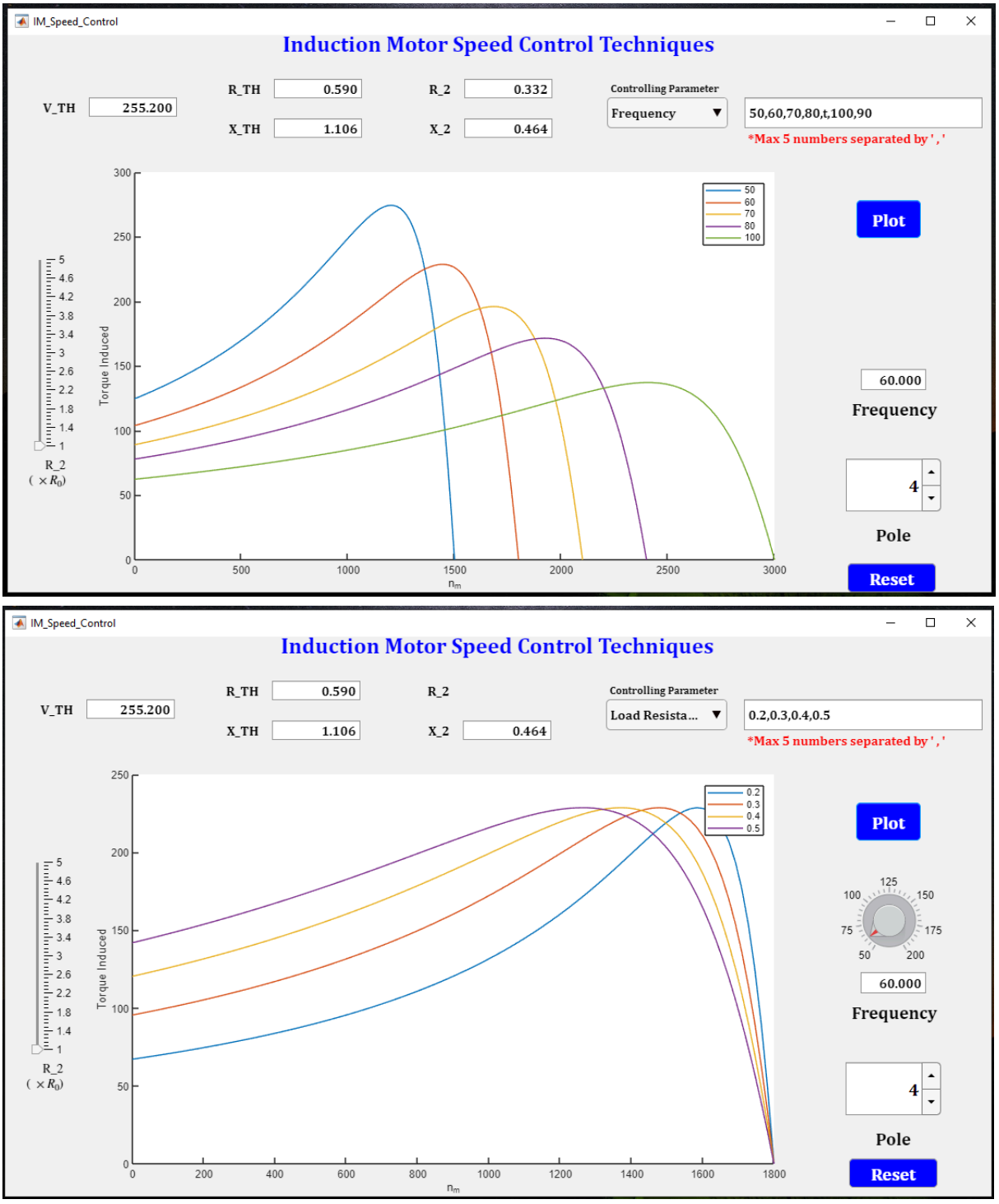


Figure 25: Test cases for Speed Control Techniques

Application of the Project

There could be several use of the app.For example

1. This app can be used as a teaching aid for both the teachers and students.
2. This app can be also adopted for industrial used with further modification.
3. This app can also be used for research purpose.

Discussion

Despite some obstacles, we were able to complete the project effectively.Still some improvement can be made to the final project. For example, while dealing with 3 Phase Transformer, we cannot display complex current in a single numeric field. We had to display the real and imaginary parts of current using two separate fields. Also while taking input we were unable to solve additional circuits of these electrical machines due to a lack of theoretical understanding. We also faced difficulties while taking array input for Speed Control Techniques. Continuously updating the variables, plotting and adding legend programmatically make the code more challenging. Our project would be more appealing if we could use Simulink..

List of References

[1] S.Chapman, Electric Machinery and Power System Fundamentals ,5th Edition,2012

[2] MATLAB AppDesigner <https://www.mathworks.com/products/matlab/app-designer.html >

[3]Electric machinery. I. Fitzgerald, A. E. (Arthur Eugene), 1909- Electric machinery. II. Title.TK2181.F5 2014