Introduction to Ti*k*Z

Herman Jaramillo Villegas
Universidad de Medellín

January 17, 2024

1 Introduction

Ti*k*Z is a powerful package to do graphics in a LaTeX environment. It is designed to generate high-quality diagrams, figures, and illustrations. The package Ti*k*Z is part of a combination PGF/Ti*k*Z [1] which are programs to create vector graphics. [2]

Ti*k*Z was created as complement to LaTeX due to the lack tools that LaTeX had for graphics. Ti*k*Z allows the construction of shapes, lines, curves, colors, and other geometrical objects. These constructions are precise due to instructions on positions programmed for each point in the graph. The Ti*k*Z community has developed packages which facilitate the work in Ti*k*Z . For example, packages on circuits, neural networks, flowcharts, and more.

The online documentation is found in the website online documentation for Ti*k*Z and PGF Packages [3]

1.1 What are the advantages of using Ti*k*Z ?

We show an incomplete least of good attributes in Ti*k*Z .

- Ti*k*Z has a steep learning curve, but once you get used to it you want to stay there. It is **flexible** and can be reused for multiple purposes.
- **Integration with** LaTeX : This is a great advantage. It allows for the edition of articles and books with high professional look.
- Ti*k*Z is **accurate**. Since its instructions are based on accurate metrics, the figures can be done with precision.
- Ti*k*Z can help on **understanding mathematical problems**. Some times we need to draw figures to understand a mathematical problem. Sketches help in many situations but sometimes precision is needed to have a proof of concept. Ti*k*Z is good for these tasks.
- Ti*k*Z is **light**. Since the instructions are text lines, they will not weight much; still they can generate vector graphics of large size. When we share work we share the source and not the figures. Moving and storing LaTeX files is easy and cheap.
- **Vector Graphics**: This means that the generated graphics have multiresolution attributes. That is they can be scaled without loosing any resolution.

[1] https://en.wikipedia.org/wiki/PGF/TikZ
[2] https://en.wikipedia.org/wiki/Vector_graphics
[3] https://tikz.dev/

- **Extensive Libraries**: There are, in the community, a lot of developments that can be integrated into your graphics.
- Ti*k*Z has a large **community support**. Many websites such as stack overflow [4] or StackExchange [5] and many more provide help to problems.
- Ti*k*Z provides **cross-Platform compatibility**. This comes from LaTeX since it is text that runs well in any platform.
- Ti*k*Z allows **mathematical expressions** which make it into an accurate programming language.
- Some dynamical geometry tools such as GeoGebra [6] can export constructions to Ti*k*Z format.

Some of the examples shown here were based on suggestions or code from ChatGPT [7]

2 Basic Elements of Ti*k*Z

2.1 First Segment

A segment can be drawn from the following example

```
\begin{tikzpicture}
  % draw a segment
  \draw (0,0)--(1,1);
\end{tikzpicture}
```

All Ti*k*Z scripts start with `\begin{tikzpicture}` and end with `\end{tikzpicture}`. The command `\draw` indicates that something will be drawn. By default, if you use two points such as `(0,0)` and `(1,1)`, connected with `--`, a **segment** is drawn. Note: it is important not to forget the semi-colon ";". The comments in Ti*k*Z are preceded by the symbol "%".

Hard coding coordinates such as in this example is not advised. Coordinates could be used several times and if you hard coded them, you cannot recycle and need to type them again and again every time you need to use them. In the next section we introduce the statment `textbackslash coordinate` with the purpose of saving coordinates.

2.2 Instruction `\coordinate` and basic geometrical shapes

Like any programming language Ti*k*Z uses variables. One of the most important variable in Ti*k*Z is the one created by the instruction `\coordinate` . This is a like a point in geometry. Out of a point we can construct geometrical shapes. For example a segment between two points, a triangle, a pentagon, etc. We modify the previous script by defining two coordinates.

[4] https://stackoverflow.com/
[5] https://stackexchange.com/
[6] https://www.geogebra.org/?lang=en
[7] https://chat.openai.com/

```
\begin{tikzpicture}
  % draw a segment
  \coordinate (O) at (0,0);
  \coordinate (A) at (1,1);
  \draw (O) -- (A);
\end{tikzpicture}
```

Note that the coordinates are inside parenthesis. Ti*k*Z is very sensitive to syntax, any small error such as for example a semicolon ;, a dash -, a parenthesis, etc, will stop the flow of the program.

2.3 Options for the instruction \draw

The syntax for the \ option is

```
\draw[<options>] <path specification>;
```

The options are many. We illustrate a few, with the segment example.

- **color**:

```
\begin{tikzpicture}
  \coordinate (O) at (0,0);
  \coordinate (A) at (1,1);
  \draw[green] (O) -- (A);
\end{tikzpicture}
```

A green segment.

- **line width**

```
\begin{tikzpicture}
  \coordinate (O) at (0,0);
  \coordinate (A) at (1,1);
  \draw[green, ultra thick ] (O) -- (A);
\end{tikzpicture}
```

An ultra thick green segment.

```
\begin{tikzpicture}
  \coordinate (O) at (0,0);
  \coordinate (A) at (1,1);
  \draw[green, line width=4] (O) -- (A);
\end{tikzpicture}
```

A 4 points width, green segment. The default unit for the line width is `points`. We should know that

```
1 point (pt) = 0.35278 millimeters (mm)
```

We could choose other units of measure such as, for example `mm, cm, in`.

Let us use the unit `mm`

```
\begin{tikzpicture}
  \coordinate (O) at (0,0);
  \coordinate (A) at (1,1);
  \draw[green, line width=4mm] (O) -- (A);
\end{tikzpicture}
```

A 4mm thick, green segment.

- **Line Style** For line style we can use:
  - solid (this is the default)
  - dashed
  - dotted
  - dash
  - dash dot
  - dash dot dot

  Let us, for example, try the option `dash dot dot`.

```
\begin{tikzpicture}
  \coordinate (O) at (0,0);
  \coordinate (A) at (1,1);
  \draw[line width = 1, dash dot dot] (O) -- (A);
\end{tikzpicture}
```

There are other options such as `opacity`, `smooth curves`, and `pattern fills`, end of the curve (like arrows) . We will deal with them later.

3 More geometrical shapes

We can define a set of points and they are connected with the connector --. For example.

```
\begin{tikzpicture}
  \coordinate (O) at (0,0);
  \coordinate (A) at (1,1);
  \coordinate (B) at (2,1);
  \coordinate (C) at (3,0);
  \draw[line width = 1, dash dot dot] (O) -- (A) -- (B) --(C);
\end{tikzpicture}
```

We can close the loop by adding the keyword `cycle` as shown here.

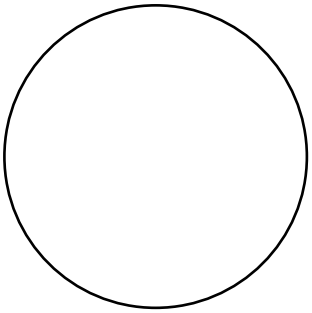
```
\begin{tikzpicture}
  \coordinate (O) at (0,0);
  \coordinate (A) at (1,1);
  \coordinate (B) at (2,1);
  \coordinate (C) at (3,0);
  \draw[line width = 1, dash dot dot] (O) -- (A) -- (B) --(C) -- cycle;
\end{tikzpicture}
```

You can draw a rectangle with two points (connecting the diagonal). For example,

```
\begin{tikzpicture}
  \coordinate (O) at (0,0);
  \coordinate (A) at (1,1);
  \draw[line width = 1, dash dot dot] (O) rectangle (A)
\end{tikzpicture}
```

Likewise you can draw a circle with the center and the radius.

```
\begin{tikzpicture}
  \coordinate (O) at (0,0);
  \draw[line width = 1]   (O) circle (2);
\end{tikzpicture}
```



Here we are hard coding the radius $R = 2$. We want to use it as a variable. We show how to do that

3.1 Defining variables in Ti*k*Z

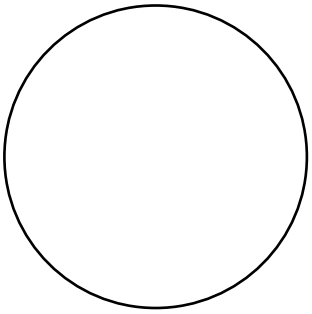
The variables in Ti*k*Z are defined through the syntax

```
\pgfmathsetmacro{\variable}{value}
```

where the name of the variable is `\variable` and the value is given by `value`.

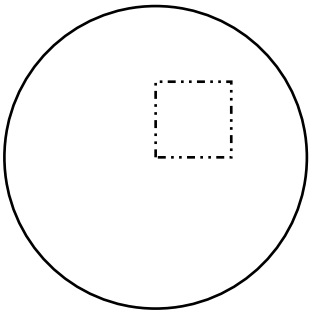
Let us see an example.

```
\begin{tikzpicture}
  \pgfmathsetmacro{\R}{2} % units of centimeters
  \coordinate (O) at (0,0);
  \coordinate (A) at (1,1);
  \draw[line width = 1]   (O) circle (\R);
\end{tikzpicture}
```



We can combine both, the rectangle and the circle as

```
\begin{tikzpicture}
  \pgfmathsetmacro{\R}{2}  % units of points
  \coordinate (O) at (0,0);
  \coordinate (A) at (1,1);
  \draw[line width = 1]   (O) circle (\R);
  \draw[line width = 1, dash dot dot] (O) rectangle (A);
\end{tikzpicture}
```



Activity # 1 There are two options that you can try for `draw`. These are

- opacity and
- pattern fills

try them on the circle surrounding the rectangle. Opacity actually works well on filled areas, still try the parameter.

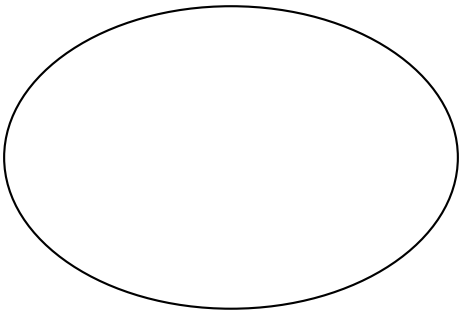
We now try to draw a smooth curve.

```
\begin{tikzpicture}
  \coordinate (O) at (0,0);
  \coordinate (A) at (1,1);
  \coordinate (B) at (2,1);
  \coordinate (C) at (3,0);
  \draw[line width = 1, dash dot dot] (O) -- (A) -- (B) --(C) ;
  \draw[red, thick, smooth] plot coordinates {(O) (A) (B) (C) (D)};
\end{tikzpicture}
```

Note that the option `smooth` requires a different syntax for the path specification. The curve is interpolated between the points using the theory of Bézier curves. [8]

We now show how to draw ellipses.

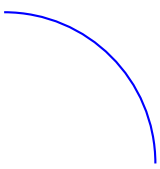
```
\begin{tikzpicture}
  % Define the ellipse parameters
    \pgfmathsetmacro{\a}{3} % Semi-major axis
    \pgfmathsetmacro{\b}{2} % Semi-minor axis

    % Draw the ellipse using calculated parameters
    \draw[black, thick] (0,0) ellipse ({\a} and {\b});
\end{tikzpicture}
```



3.2 Arcs in Ti*k*Z

Here is a simple arc:

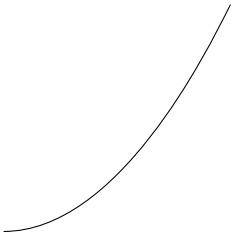
```
\begin{tikzpicture}
    \coordinate (O) at (0,0);
    \pgfmathsetmacro{\startAngle}{0}  % units of degrees
    \pgfmathsetmacro{\endAngle}{90}  % units of degrees
    \pgfmathsetmacro{\R}{2}  % units of points
    % Syntax: \draw[options] (start angle: end angle: radius)
    % Draws a quarter-circle arc
    \draw[blue, thick] (O) arc (\startAngle:\endAngle:\R);
\end{tikzpicture}
```



[8] https://en.wikipedia.org/wiki/B%C3%A9zier_curve

We should warn the reader that `(0)` is not the center of the arc, but the starting point.
Here is a way to build an arc of parabola using Ti*k*Z :

```
\begin{tikzpicture}
  \coordinate (O) at (0,0);
  \coordinate (A) at (3,3);
  \draw[] (O) parabola (A);
\end{tikzpicture}
```

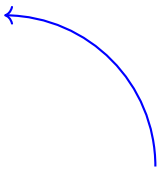


It is common that we need arrows to indicate some orientation of a curve.

3.3 End of curves. Arrows and other

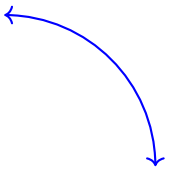
There is plenty of material about arrows in the Ti*k*Z manual[9] . We only address a few styles here.

```
\begin{tikzpicture}
    \coordinate (O) at (0,0);
    \pgfmathsetmacro{\startAngle}{0}  % units of degrees
    \pgfmathsetmacro{\endAngle}{90}  % units of degrees
    \pgfmathsetmacro{\R}{2}  % units of points
    % Syntax: \draw[options] (start angle: end angle: radius)
    % Draws a quarter-circle arc
    \draw[blue, thick, ->] (O) arc (\startAngle:\endAngle:\R);
\end{tikzpicture}
```



Note the option `->`. This is the simplest way to add an arrow at the end of a curve. There are other options which we will try here.

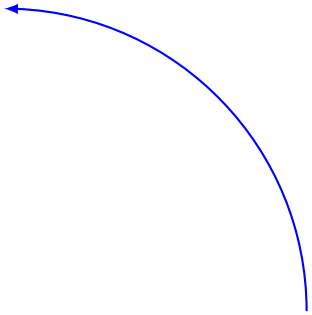
```
\begin{tikzpicture}
    \coordinate (O) at (0,0);
    \pgfmathsetmacro{\startAngle}{0}  % units of degrees
    \pgfmathsetmacro{\endAngle}{90}  % units of degrees
    \pgfmathsetmacro{\R}{2}  % units of points
    % Syntax: \draw[options] (start angle: end angle: radius)
    \draw[blue, thick, <->] (O) arc (\startAngle:\endAngle:\R);
\end{tikzpicture}
```



[9] https://tikz.dev/tikz-arrows

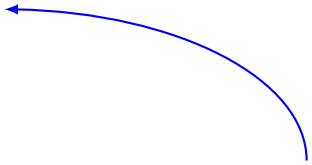
Here <-> means double arrow (at the beginning and at the end).

```
\begin{tikzpicture}[scale=2}
    \coordinate (O) at (0,0);
    \pgfmathsetmacro{\startAngle}{0}  % units of degrees
    \pgfmathsetmacro{\endAngle}{90}  % units of degrees
    \pgfmathsetmacro{\R}{2}  % units of points
    % Syntax: \draw[options] (start angle: end angle: radius)
    \draw[blue, thick, -latex] (O) arc (\startAngle:\endAngle:\R);
\end{tikzpicture}
```



Here, another type of arrow. It is called a `-latex` arrow. Note also that we used the parameter `scale=2` at the beginning of the script to make the plot larger. We could use also use `xscale=2` to stretch the figure along the x axis.

```
\begin{tikzpicture}[xscale=2}
    \coordinate (O) at (0,0);
    \pgfmathsetmacro{\startAngle}{0}  % units of degrees
    \pgfmathsetmacro{\endAngle}{90}  % units of degrees
    \pgfmathsetmacro{\R}{2}  % units of points
    % Syntax: \draw[options] (start angle: end angle: radius)
    \draw[blue, thick, -latex] (O) arc (\startAngle:\endAngle:\R);
\end{tikzpicture}
```



This trick is nice to convert circles into ellipses.
The type of arrows that we have are

- `->` : arrows at the end of the path
- `<-` : arrows at the start of the path
- `<->` : double arrow
- `-stealth` : A stealth shape arrow

The arrow tips have also options.

- **type**: the type of arrow is one of the types explained in the previous list. For example, `-latex`, `->`, etc.
- **length**: Length of the arrow head

```
\begin{tikzpicture}[xscale=2}
    \coordinate (O) at (0,0);
    \pgfmathsetmacro{\startAngle}{0}  % units of degrees
    \pgfmathsetmacro{\endAngle}{90}  % units of degrees
    \pgfmathsetmacro{\R}{2}  % units of points
    \draw[-{Latex[length=5mm, width=3mm]}, purple] (O)
      arc (\startAngle:\endAngle:\R);
\end{tikzpicture}
```

Observe that we can split a line in two (see the instruction `arc`) with no special characters needed.

To use the `Latex` directive above we need to include the `arrows.meta` library. Tha is,

```
\usetikzlibrary{arrows.meta}
```

Note that we change the length and witdth of the arrow. We changed also the color of the curve. There are plenty of more options in the Ti*k*Z manual [10] which we will not consider here. We suggest to the student to visit the manual for more on arrows and ending shapes of curves.

Activity # 2 : Make a curve with an arrow at the end. The color of the curve is black and that of the arrow is blue.

3.3.1 Make a grid

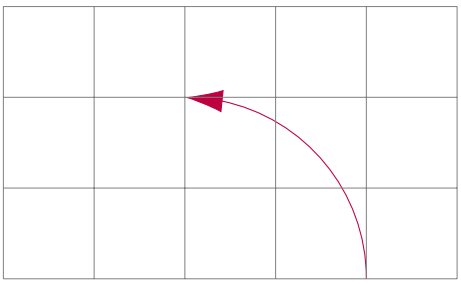
The key line here is:

```
\draw[ step=1cm, gray, very thin] (G) grid (A);
```

We draw a grid around the arc defined above.

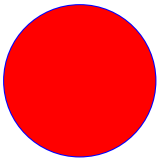
```
\begin{tikzpicture}[scale = 1.2]
    \pgfmathsetmacro{\startAngle}{0}  % units of degrees
    \pgfmathsetmacro{\endAngle}{90}  % units of degrees
    \pgfmathsetmacro{\R}{2}  % units of points
    % Syntax: \draw[options] (start angle: end angle: radius)
    \coordinate (O) at (0,0);
    \coordinate (A) at (2,0);
    \draw[-{Latex[length=5mm, width=3mm]}, purple] (A)
       arc (\startAngle:\endAngle:\R);
    \coordinate (G) at (-2,0);
    \coordinate (A) at (3,3);
    \draw[ step=1cm, gray, very thin] (G) grid (A);
\end{tikzpicture}
```

[10] https://tikz.dev/tikz-arrows



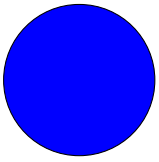
3.4 The fill and shade

```
\begin{tikzpicture}
  \pgfmathsetmacro{\R}{1}  % units of points
  \coordinate (O) at (0,0);
  \draw[thick, blue] (0,0) circle (1); % Outline of the circle
   % Filling a circle with a color
  \fill[red] (O) circle (\R); % Fill the circle with red color
\end{tikzpicture}
```

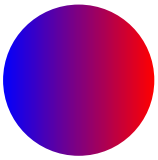


You can do both tasks (draw the border and fill) at the same time with the instruction `\filldraw`. Let us see

```
\begin{tikzpicture}
  \pgfmathsetmacro{\R}{1}  % units of points
  \coordinate (O) at (0,0);
   % Filling a circle with a color and draw boundary
  \filldraw[fill=blue, draw=black] (O) circle (\R);
\end{tikzpicture}
```



```
\begin{tikzpicture}
  \pgfmathsetmacro{\R}{1}  % units of points
  \coordinate (O) at (0,0);
   % shading a circle
  \shade[left color=blue, right color=red] (O) circle (\R);
\end{tikzpicture}
```



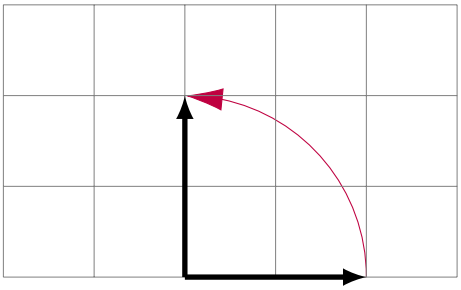
Shade works also from top to bottom, bottom to top, inner to out, or out to inner. We leave this kind of shade to the student

Activity # 3 Shade the circle above from top to bottom, bottom to top, inner to outer, and outer to inner.

3.5 Axes

Axes are easily included. Let us copy the script with the arc in a grid and add the axes to it, after a few modifications; mainly the scaling and the starting of the arc, for a more pleasant display.

```
\begin{tikzpicture}[scale = 1.2]
    \pgfmathsetmacro{\startAngle}{0}  % units of degrees
    \pgfmathsetmacro{\endAngle}{90}  % units of degrees
    \pgfmathsetmacro{\R}{2}  % units of points
    % Syntax: \draw[options] (start angle: end angle: radius)
    \coordinate (O) at (0,0);
    \coordinate (A) at (2,0);
    \draw[-{Latex[length=5mm, width=3mm]}, purple] (A)
       arc (\startAngle:\endAngle:\R);
    \coordinate (G) at (-2,0);
    \coordinate (A) at (3,3);
    \draw[ step=1cm, gray, very thin] (G) grid (A);

    \coordinate (OX) at (2,0);
    \coordinate (Oy) at (0,2);

    \draw[line width=2, -latex] (O) -- (OX);
    \draw[line width=2, -latex] (O) -- (Oy);
\end{tikzpicture}
```



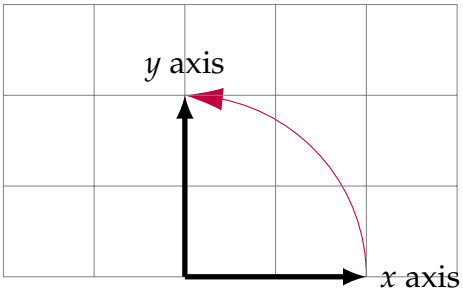
At this point we are in need to put some labels. We introduce the `node` directive.

3.6 The `node` and `\node` directives

There are two ways to use the string `node` in Ti*k*Z , one with no `\` and the other with `\`. The `node` with no backslash is used directly in the line that starts with `\draw`. Here is an example with no backslash.

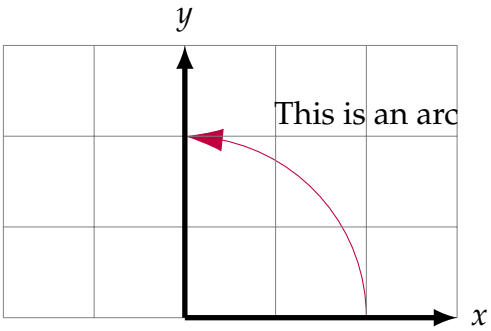
```
\begin{tikzpicture}[scale = 1.2]
    \pgfmathsetmacro{\startAngle}{0}  % units of degrees
    \pgfmathsetmacro{\endAngle}{90}  % units of degrees
    \pgfmathsetmacro{\R}{2}  % units of points
    % Syntax: \draw[options] (start angle: end angle: radius)
    \coordinate (O) at (0,0);
    \coordinate (A) at (2,0);
```

```
    \draw[-{Latex[length=5mm, width=3mm]}, purple] (A)
       arc (\startAngle:\endAngle:\R);
    \coordinate (G) at (-2,0);
    \coordinate (A) at (3,3);
    \draw[ step=1cm, gray, very thin] (G) grid (A);

    \coordinate (OX) at (2,0);
    \coordinate (Oy) at (0,2);

    \draw[line width=2, -latex] (O) -- (OX) node[right] {$x$ axis};
    \draw[line width=2, -latex] (O) -- (Oy) node[above] {$y$ axis};
\end{tikzpicture}
```



We now illustrate the use of \node ,

```
\begin{tikzpicture}[scale = 1.2]
    \pgfmathsetmacro{\startAngle}{0}  % units of degrees
    \pgfmathsetmacro{\endAngle}{90}  % units of degrees
    \pgfmathsetmacro{\R}{2}  % units of points
    % Syntax: \draw[options] (start angle: end angle: radius)
    \coordinate (O) at (0,0);
    \coordinate (A) at (2,0);
    \draw[-{Latex[length=5mm, width=3mm]}, purple] (A)
       arc (\startAngle:\endAngle:\R);
    \coordinate (G) at (-2,0);
    \coordinate (A) at (3,3);
    \draw[ step=1cm, gray, very thin] (G) grid (A);

    \coordinate (OX) at (3,0);
    \coordinate (Oy) at (0,3);

    \draw[line width=2, -latex] (O) -- (OX) node[right] {$x$};
    \draw[line width=2, -latex] (O) -- (Oy) node[above] {$y$};


    \coordinate (M) at (2,2);
    \node[above] at (M) {This is an arc };
\end{tikzpicture}
```



There are many options for `node` (with or without backslash), some of them are:

- scale: scales the node.
- rotate: rotate the node.
- below: locates the node below the reference coordinate (in the example above the coordinate is $(2, 2)$.
- above: locates the node output above the coordinate.
- left: locates the node output to the left of the coordinate.
- right : locates the node output to the right of the coordinate.
- below right: locates the node output below to the right of the coordinate.
- below left: locates the node output below and to the left of the coordinate.
- above right locates the node output above and to the right of the coordinate.
- above left: locates the node output above and to the left of the coordinate.
- xshift: This good for fine tunning the location of your node. Usually small quantities shift the node to the right or to the left. For example $-2mm$ to the left and $2mm$ to the right.
- yshift: This, like the previous, shifts the node up or down.

Activity # 4 Try all the options on `\node` above.

It is interesting that there is not a `print` statement in Ti*k*Z . However the `node` directive serves as a print for any variable in the program or any message that want to be display with the graphic. In this way the `node` (with or without backslash) helps on debugging Ti*k*Z code.

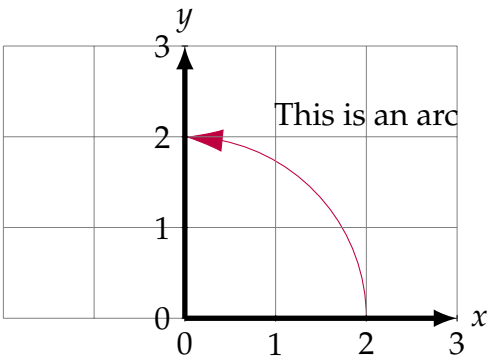
4 Control flow instructions: loops and if statements

4.1 The `\foreach` loop

We will put ticks on the figure with grid and axis.

```
\begin{tikzpicture}[scale = 1.2]
    \pgfmathsetmacro{\startAngle}{0}  % units of degrees
    \pgfmathsetmacro{\endAngle}{90}  % units of degrees
    \pgfmathsetmacro{\R}{2}  % units of points
```

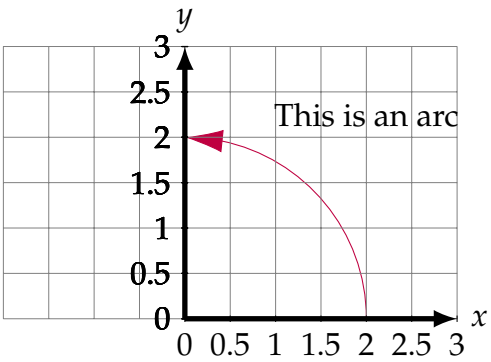
```
    % Syntax: \draw[options] (start angle: end angle: radius)
    \coordinate (O) at (0,0);
    \coordinate (A) at (2,0);
    \draw[-{Latex[length=5mm, width=3mm]}, purple] (A)
       arc (\startAngle:\endAngle:\R);
    \coordinate (G) at (-2,0);
    \coordinate (A) at (3,3);
    \draw[ step=1cm, gray, very thin] (G) grid (A);

    \coordinate (OX) at (3,0);
    \coordinate (Oy) at (0,3);

    \draw[line width=2, -latex] (O) -- (OX) node[right] {$x$};
    \draw[line width=2, -latex] (O) -- (Oy) node[above] {$y$};

    \foreach \x in {0,1,2,3}
        \draw (\x cm,1pt) -- (\x cm,-1pt) node[anchor=north]{$\x$};
    \foreach \y in {0,1,2,3}
        \draw (1pt,\y cm) -- (-1pt,\y cm) node[anchor=east]{$\y$};

    \coordinate (M) at (2,2);
    \node[above] at (M) {This is an arc };
\end{tikzpicture}
```



The \foreach loop can be extended for more general ranges. In the following example we show that a few first values will define the increment and you can fix the last position. We make the grid double as fine on each direction.

```
\begin{tikzpicture}[scale = 1.2]
    \pgfmathsetmacro{\startAngle}{0}  % units of degrees
    \pgfmathsetmacro{\endAngle}{90}  % units of degrees
    \pgfmathsetmacro{\R}{2}  % units of points
    % Syntax: \draw[options] (start angle: end angle: radius)
    \coordinate (O) at (0,0);
    \coordinate (A) at (2,0);
    \draw[-{Latex[length=5mm, width=3mm]}, purple] (A)
       arc (\startAngle:\endAngle:\R);
    \coordinate (G) at (-2,0);
    \coordinate (A) at (3,3);
    \draw[ step=0.5cm, gray, very thin] (G) grid (A);

    \coordinate (OX) at (3,0);
    \coordinate (Oy) at (0,3);
```

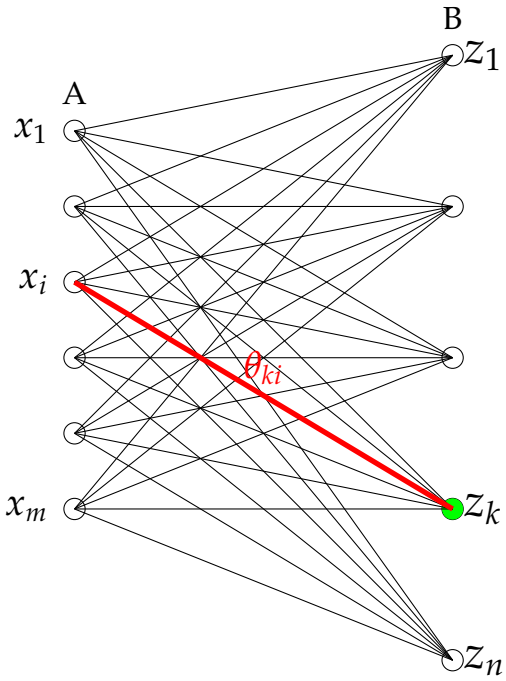
```
    \draw[line width=2, -latex] (0) -- (0X) node[right] {$x$};
    \draw[line width=2, -latex] (0) -- (0y) node[above] {$y$};

    \foreach \x in {0, 0.5, 1, 1.5, ..., 3} {
        \draw (\x cm,1pt) -- (\x cm,-1pt) node[anchor=north] {$\x$};
        \foreach \y in {0, 0.5, 1, 1.5, ..., 3} {
        \draw (1pt,\y cm) -- (-1pt,\y cm) node[anchor=east] {$\y$};
      }
    }

    \coordinate (M) at (2,2);
    \node[above] at (M) {This is an arc };
\end{tikzpicture}
```



Finally, we will show an example of how to construct a neural network to explain the matrix vector product. There are packages already built to draw Neural networks, we show them at the end of this notes.

```
\begin{center}
  \begin{tikzpicture}

    \foreach \t in {0,1,...,5} {
      \draw[] (0, \t) circle(4pt);
    }

    \foreach \t in {-2, 0, ..., 6} {
      \draw[] (5, \t) circle(4pt);
    }

      % node k in the arriving layer
    \coordinate (B2) at (5, 0);
    \node[right] at (B2) {\Large $z_k$};
    \coordinate (B3) at (5, 6);
    \node[right] at (B3) {\Large $z_1$};
    \coordinate (B4) at (5, -2);
    \node[right] at (B4) {\Large $z_n$};
    \fill[green] (B2) circle (4pt);
```

```
    \foreach \x in {0,1,...,5} {
      \foreach \y in {-2,0,...,6} {
        \draw[] (0, \x) -- (5, \y);
      }
    }

    \draw[line width=2, color=red] (0,3)--(B2);
    \coordinate (theta) at (2.5, 1.5);
    \node[above] at (theta) {\large $\textcolor{red}{\bf \theta_{ki}}$};
    \node[left] at (-0.2,3) {\large $x_i$};
    \node[left] at (-0.2,0) {\large $x_m$};
    \node[left] at (-0.2,5) {\large $x_1$};

    \coordinate (A) at (0,5.2);
    \coordinate (B) at (5,6.2);
    \node[above] at (A) {A};
    \node[above] at (B) {B};

  \end{tikzpicture}
\end{center}
```

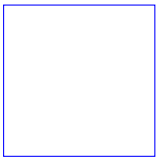


4.2 The \if statement

The following example, generated by ChatGPT, shows the use of the `if` statement in Ti*k*Z . The instruction
`\def\myvalue`
is a macro from LaTeX . It does not belong to Ti*k*Z .

```
\begin{tikzpicture}
    % Define a variable
    \def\myvalue{5}
```

```
    % Conditional statement to draw different shapes
    \ifnum\myvalue<5
        \draw[red] (0,0) circle (1);
    \else
        \draw[blue] (0,0) rectangle (2,2);
    \fi
\end{tikzpicture}
```



The script decides if drawing a circle (if the number \myvalue is smaller than 5), or a rectangle (if the number \myvalue is smaller or equal than 5). Since the number is equal to 5 a rectangle is drawn.

5 Mathematics in Ti*k*Z

Ti*k*Z is not a package to do math. However some complex programs could need some mathematical computations. We show how do do math operations with Ti*k*Z .

Here is a script with some simple math.

```
\begin{tikzpicture}
    % Define variables
    \pgfmathsetmacro{\a}{3}
    \pgfmathsetmacro{\b}{7}

    % Perform arithmetic operations
    \pgfmathsetmacro{\sum}{\a + \b}
    \pgfmathsetmacro{\difference}{\b - \a}
    \pgfmathsetmacro{\product}{\a * \b}
    \pgfmathsetmacro{\quotient}{\b / \a}

    % Display results
    \node at (0,0) {Sum: $\sum$};
    \node at (0,-0.5) {Difference: $\difference$};
    \node at (0,-1) {Product: $\product$};
    \node at (0,-1.5) {Quotient: $\quotient$};
\end{tikzpicture}
```

Sum: 10.0
Difference: 4.0
Product: 21.0
Quotient: 2.33333

The idea is not to do complicated math but use some math for complex graphics that require it.

We can do more complex math such as trigonometrical, logarithm, and exponential functions.

```
\begin{tikzpicture}
    % Trigonometric functions
```

```
    \pgfmathsetmacro{\angle}{30}
    \pgfmathsetmacro{\sine}{sin(\angle)}
    \pgfmathsetmacro{\cosine}{cos(\angle)}
    \pgfmathsetmacro{\tangent}{tan(\angle)}

    % Exponential and logarithmic functions
    \pgfmathsetmacro{\exponential}{exp(1)}
    \pgfmathsetmacro{\lnValue}{ln(5)}

    % Display results
    \node at (0,3) {Trigonometric Functions:};
    \node at (0,2.5) {$\sin(\angle) = \sine$};
    \node at (0,2) {$\cos(\angle) = \cosine$};
    \node at (0,1.5) {$\tan(\angle) = \tangent$};

    \node at (0,0.5) {Exponential and Logarithmic Functions:};
    \node at (0,0) {$e = \exponential$};
    \node at (0,-0.5) {$\ln(5) = \lnValue$};
\end{tikzpicture}
```

Trigonometric Functions:

$\sin(30) = 0.5$

$\cos(30) = 0.86603$

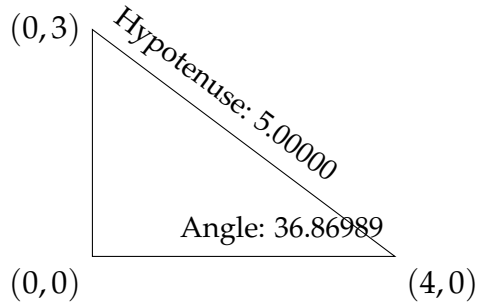
$\tan(30) = 0.57732$

Exponential and Logarithmic Functions:

$e = 2.71825$

$\ln(5) = 1.60942$

Activity # 5: Write a Ti*k*Z script that draws a right triangle with vertices $(0,0)$, $(0,4)$, $(0,3)$. Compute, using the Pythagoras theorem, the length of the hypotenuse and write this in a text above the hypotenuse tilted so that it is parallel to it. Write also the angle at the vertex $(4,0)$. In other words, write a script that shows the following figure.

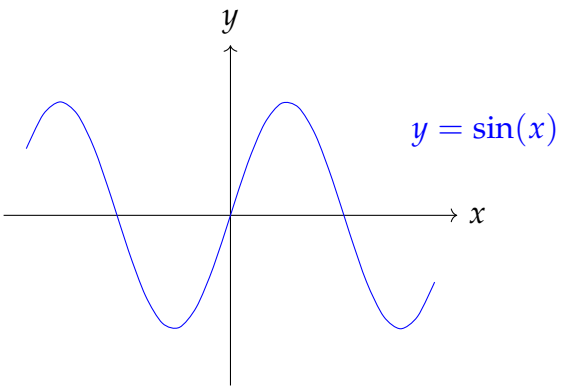


6 Graphing functions

Graphing functions is easy with Ti*k*Z . Here is a script to graph the function $y = \sin x$.

```
\begin{tikzpicture}[scale=1.5]
    % Axes
    \draw[->] (-2,0) -- (2,0) node[right] {$x$};
```

```
    \draw[->] (0,-1.5) -- (0,1.5) node[above] {$y$};

    % Sine function
    \draw[domain=-1.8:1.8,smooth,variable=\x,blue]
        plot (\x,{sin(180*\x)});

    % Equation labels
    \node[below right,blue] at (1.5,1) {$y = \sin(x)$};

\end{tikzpicture}
```



Note that we hard-coded the coordinates and other numerical parameters to make the script short. However when writing professional code this should not be done.

6.1 Plots using \addplot and axis

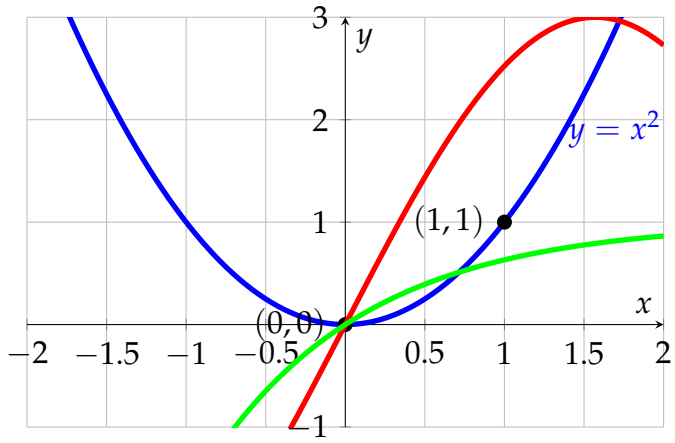
There is an environment

```
\begin{axis}
  do things here
\end{axis}
```

which is quite useful for plotting functions. Here is an example of how this works. It is flexible to include several 2D graphs.

```
\begin{tikzpicture}
    \begin{axis}[
        axis lines=middle,
        xlabel=$x$,
        ylabel=$y$,
        xmin=-2, xmax=2,
        ymin=-1, ymax=3,
        grid=both,
        width=10cm,
        height=7cm,
        samples=100 % Number of samples for smooth curve
    ]

    % Plot the function
     % Function: y = x^2
    \addplot[blue, domain=-2:2, line width=2] {x^2}
        node[pos=0.8, below right, xshift=-10pt] {$y=x^2$};
    % Mark some key points
```

```
    \node[label={180:{$(0,0)$}},circle,fill,inner sep=2pt]
         at (axis cs:0,0) {};
    \node[label={180:{$(1,1)$}},circle,fill,inner sep=2pt]
         at (axis cs:1,1) {};



    % Function: y=3 sin(x*180/pi)
    \addplot[red, line width=2,  domain=-2:2] {3*sin(x*180/pi)};
    % Function: y=1-exp(-x)
    \addplot[green, line width=2, domain=-2:2] {1-exp(-x)};

    \end{axis}
\end{tikzpicture}
```

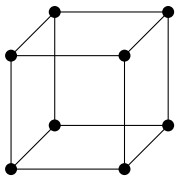


7 Three dimensional graphs

The following example, generated with ChatGPT shows a figure using 3D coordinates.

```
\begin{tikzpicture}[scale=1.5]

  % Define coordinates
  \coordinate (A) at (0,0,0);
  \coordinate (B) at (1,0,0);
  \coordinate (C) at (1,1,0);
  \coordinate (D) at (0,1,0);
  \coordinate (E) at (0,0,1);
  \coordinate (F) at (1,0,1);
  \coordinate (G) at (1,1,1);
  \coordinate (H) at (0,1,1);

% Draw the edges of the cube
  \draw (A) -- (B) -- (C) -- (D) -- cycle;
  \draw (E) -- (F) -- (G) -- (H) -- cycle;
  \draw (A) -- (E);
  \draw (B) -- (F);
  \draw (C) -- (G);
  \draw (D) -- (H);
```

```
% Show vertices
  \foreach \vertex in {A,B,C,D,E,F,G,H}
      \fill [black] (\vertex) circle (1.5pt);

\end{tikzpicture}
```

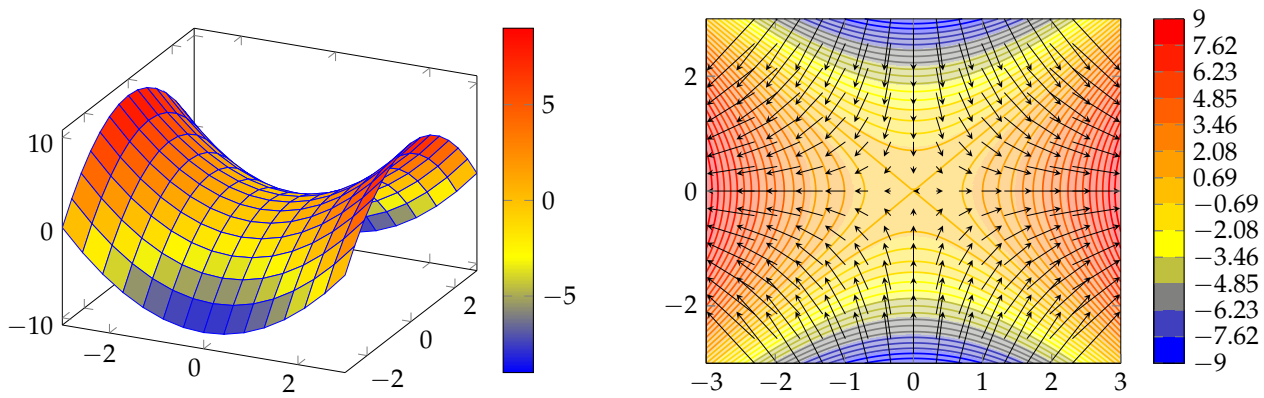


7.1 Collaboration with GNUPLOT

The following script from my multidimensional calculus book creates a surface, the contours and the field lines.

```
\begin{tikzpicture}[scale=0.8]
\begin{axis}[colorbar]
\addplot3
[surf,faceted color=blue,
samples=15,
domain=-3:3,y domain=-3:3]
{x^2 - y^2};
\end{axis}
\end{tikzpicture}
\qquad
\begin{tikzpicture}[scale=0.8]
  \begin{axis}[domain=-3:3,view={0}{90},
    colormap={CM}{
  samples of colormap=(13 of hot)},
  colormap access=piecewise constant,
  colorbar ,
  colorbar style={%
    ytick=data,
    }
  ]

    \addplot3[surf,shader=interp, opacity=0.4] {x^2-y^2};
    \addplot3[contour gnuplot={number=45, labels=false},thick,
        samples=100] {x^2-y^2};
      \addplot3[
        quiver = {
          u = {2*x},
          v = {-2*y},
          scale arrows = 0.15
        },
        -stealth,
        domain = -2.3:2.3,
        domain y = -3.0:3.0,
        samples=15
```

```
    ] {0};
  \end{axis}
\end{tikzpicture}
```



There are several important things that should be considered here.

- We need to include the following LaTeX packages:

```
\usepackage{tikz-3dplot}
\usetikzlibrary{decorations.markings,arrows}
\pgfplotsset{compat=newest}
```

- We are using the GNUPLOT package. See for example the instruction:

```
\addplot3[contour gnuplot={number=45, labels=false},thick,
    samples=100]
```

This implies that we need to run the program twice. The fist time that we run it, some data is created. That is, we will get the new files

```
input_contourtmp0.dat
input_contourtmp0.script
```

The `data` is needed for GNUPLOT and the `script` contains the instructions needed by GNUPLOT. Here `input` is the name of the source file. We now need to run the GNUPLOT script with the command:

```
:ClassNotes>gnuplot input_contourtmp0.script
```

(here ClassNotes is my working directory for this document) Then we run the LaTeX compilation instruction

```
:ClassNotes>pdflatex input
```

8 Macros in Ti*k*Z

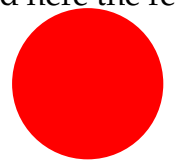
The following macro (created by ChatGPT) in Ti*k*Z is defined before \begin{docucument}

```
\newcommand{\coloredcircle}[2]{ % Takes two arguments: color and radius
  \begin{tikzpicture}
    \fill[#1] (0,0) circle (#2);
  \end{tikzpicture}
```

Here is the use of it

```
\coloredcircle{red}{1cm}
```

and here the result.



Indeed these are macros in LaTeX . Macros in LaTeX have the name \newcommand followed by a body enclosed by braces { }. Here is a couple of macros that I use to write the words LaTeX and Ti*k*Z .

```
% Define a custom command for TikZ
\newcommand{\myTikZ}{Ti\textit{k}Z }
\newcommand{\myLaTeX}{\LaTeX \space  }
```

Activity # 6 Write the two previous macros in your LaTeX document and execute the document to see that they work as expected. Assume that you need to write a sentence (centered) many times in different parts of your work and you do not want to type that sentence every time (or copy/paste). Write a macro in LaTeX that writes the sentence "I need to do this many times".

Call the macro with the name \myTask and the execution will result in

I need to do this many times

.

9 Example of Ti*k*Z packages

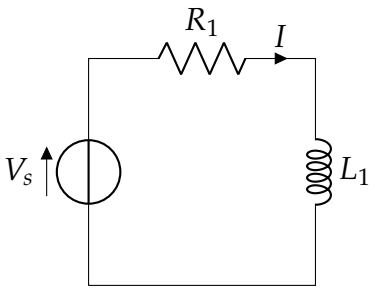
There are many Ti*k*Z packages out there. We will only illustrate a couple of them One is circuitikz and the other is neuralnetwork We can use the lines

```
\usepackage{circuitikz}
\usepackage{neuralnetwork}
```

in the preamble.

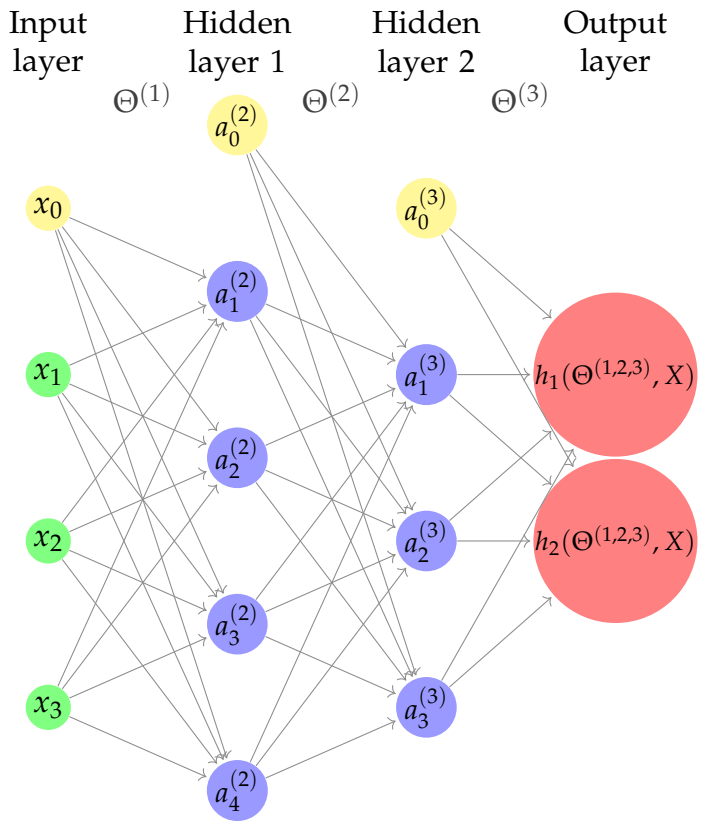
Here a simple example of a circuit generated with ChatGPT.

```
\begin{circuitikz}
    \draw (0,0) to[V, v=$V_s$] (0,3)  % Voltage source
    to[R, l=$R_1$, i=$I$] (3,3)  % Resistor
    to[L, l=$L_1$] (3,0)  % Inductor
    -- (0,0);  % Back to initial point
\end{circuitikz}
```



Here a simple example of a neural network from the Jaramillo-Rüger Machine Learning book.

```
\begin{neuralnetwork}[height=5, nodespacing=2.2cm]
  \newcommand{\x}[2]{$x_{#2}$}
  \newcommand{\y}[2]{\footnotesize $h_#2(\Theta^{(1,2,3)},X)$}
  \newcommand{\hfirst}[2]{\small $a^{(2)}_#2$}
  \newcommand{\hsecond}[2]{\small $a^{(3)}_#2$}
  \inputlayer[count=3, bias=true, title=Input\\layer, text=\x]
  \hiddenlayer[count=4, bias=true, title=Hidden\\layer 1, text=\hfirst]
  \linklayers[title=$\Theta^{(1)}$]
  \hiddenlayer[count=3, bias=true, title=Hidden\\layer 2, text=\hsecond]
  \linklayers[title=$\Theta^{(2)}$]
  \outputlayer[count=2, title=Output\\layer, text=\y]
  \linklayers[title=$\Theta^{(3)}$]
\end{neuralnetwork}
```



Activity # 7 Study the neural network shown above. Add one more hidden layer.