IV Simpósio Nacional
da Formação do Professor de Matemática

TikZ: UMA FERRAMENTA GRÁFICA PARA O PROFESSOR DE MATEMÁTICA

Beatriz Cabral
Tarso Caldas
Fabio Simas

SBMat
ANPMat
Associação Nacional dos Professores
de Matemática na Educação Básica

Ti*k*Z:
Uma ferramenta gráfica para o professor de matemática

Ti*k*Z: Uma ferramenta gráfica para o professor de matemática

Capa: Pablo Diego Regino
Projeto gráfico: Cinthya Maria Schneider Meneghetti

Distribuição
Associação Nacional dos Professores de Matemática na Educação Básica
https://www.anpmat.org.br / email: secretaria@anpmat.org.br

ISBN 978-65-88013-01-4

IV Simpósio Nacional
da Formação do Professor de Matemática

TikZ: UMA FERRAMENTA GRÁFICA PARA O PROFESSOR DE MATEMÁTICA

Beatriz Cabral
Tarso Caldas
Fabio Simas

1ª edição
2020
Rio de Janeiro

ANPMat

Sobre o material

Este trabalho foi desenvolvido com a orientação do Prof. Fábio Simas (Unirio) no contexto do projeto Livro Aberto de Matemática (umlivroaberto.org). Registramos os nossos agradecimentos aos professores Michel Cambrainha e Gladson Antunes, da Unirio que tiverem papel fundamental nessa realização. Agradecemos também à Obmep pela iniciativa, e à Fundação Itaú Social por viabilizá-lo financeiramente.

Este livro foi elaborado para o minicurso homônimo, de três horas e meia de duração, no IV Simpósio Nacional da Formação do Professor de Matemática, organizado pela ANPMat.

Sobre os autores

Beatriz Cabral e Tarso Caldas são estudantes de licenciatura em Matemática na Universidade Federal do Estado do Rio de Janeiro (Unirio). Beatriz e Tarso foram bolsistas de extensão no projeto Livro Aberto de Matemática, projeto coordenado pelo Prof. Fábio Simas (Unirio).

Contato

Entre em contato com os autores:

Fábio Simas: `fabio.simas@Uniriotec.br`
Beatriz Cabral: `beatriz.cabral@Uniriotec.br`
Tarso Caldas: `tarsobcaldas@gmail.com`

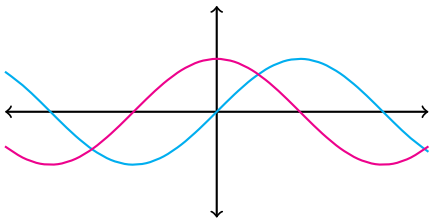
Sumário

Capítulo 1

Introdução

Elaborar listas de exercícios, provas, gabaritos, cadernos didáticos ou livros didáticos pode ser tarefas ainda mais desafiadoras sem o uso das ferramentas apropriadas. Em nossa experiência com a produção de material didático no projeto do Livro Aberto de Matemática (Obmep/Impa), conhecemos o Ti*k*Z, um pacote para LaTeX que habilita comandos para produção de figuras precisas com estética elegante a partir de comandos escritos. Acreditamos que ela pode ser uma bela aquisição para a caixa de ferramentas do Professor de Matemática.

Abaixo temos um exemplo de figura e seu respectivo código em Ti*k*Z para esboçar no plano cartesiano as funções seno e cosseno.



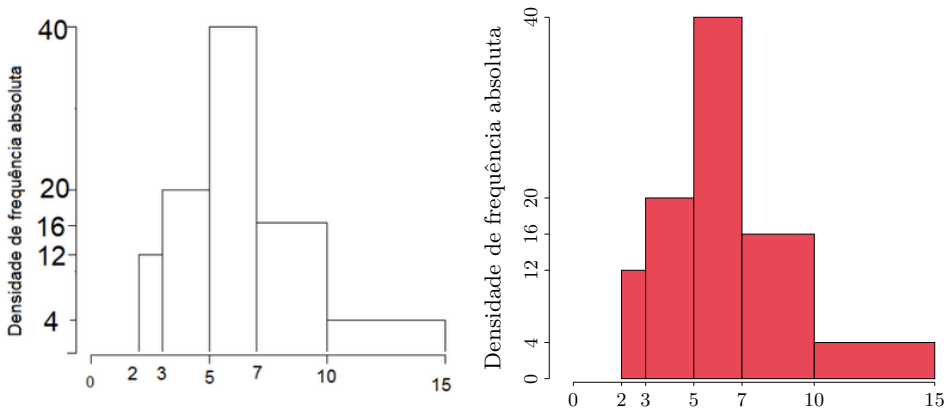
```
\draw [thick,<->] (-4,0) -- (4,0);
\draw [<->,thick] (0,-2) -- (0,2);
\draw [domain=-4:4, smooth, cyan,
thick] plot (\x,{sin(\x r)});
\draw [domain=-4:4, smooth, magenta,
thick] plot (\x,{cos(\x r)});
```

Com o Ti*k*Z, a ilustração de problemas e conceitos matemáticos torna-se menos árdua porque o autor pode imprimir rigor matemático de aspectos como proporção, simetria, distância e forma descrevendo o que espera obter diretamente no código que gera a figura. Além da precisão, com o Ti*k*Z é muito simples produzir diversas figuras seguindo o mesmo padrão editorial, como tamanho e forma de pontos, estilo de linhas, tamanho e estilo de fontes, definição precisa de cores etc.

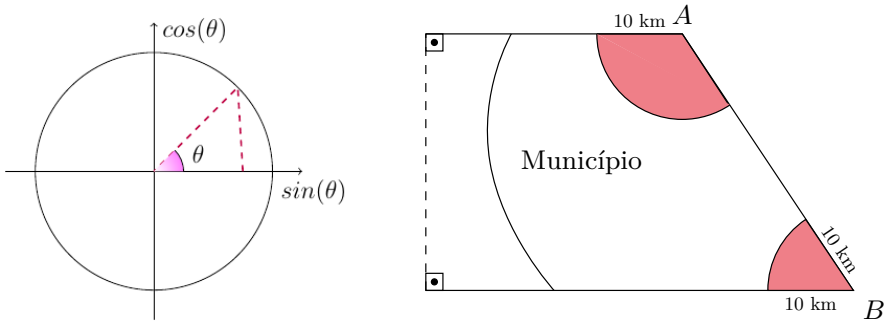
A linguagem Ti*k*Z pode ser usada para se construírem figuras fora de um arquivo em LaTeX. Contudo, o uso conjunto dessas ferramentas proporciona

Vitória - ES / UFES - IV Simpósio Nacional da Formação do Professor de Matemática - Vitória - ES / UFES - IV Simpósio Nacional d

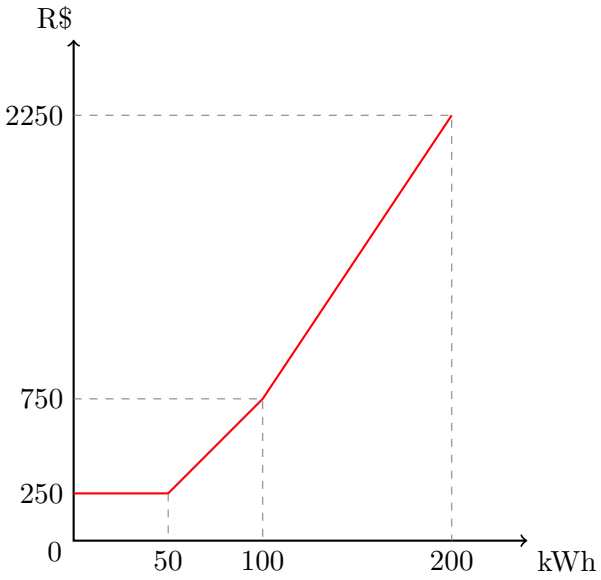
diversas facilidades. Dentre elas destacamos a possibilidade de se editar as figuras no próprio arquivo do texto, sem que seja necessário sair do ambiente do texto, e o fato de que o Ti*k*Z utiliza as mesmas definições de estilo do LaTeX, como códigos de cores, texto, tamanho de fonte, entre outras. Abaixo temos um exemplo de um histograma que foi refeito em Ti*k*Z à direita. Observe que a nova figura usa as mesmas definições de fonte deste texto.



As figuras a seguir ilustram um pouco do que pode ser feito com o Ti*k*Z.



Este livro consiste de uma breve introdução ao tema. Nele são apresentados alguns dos comandos mais básicos pensados para que o professor seja capaz de fazer algumas das principais figuras de geometria plana, de funções e algumas outras representações gráficas como histogramas e grafos mais comuns na Educação Básica. A apresentação de um grupo de comandos é sempre seguida de atividades para que o leitor pratique.



Nosso objetivo com este curso é mostrar como fazer figuras semelhantes e até mais complexas, abordando diferentes formas para diferentes conteúdos matemáticos.

Capítulo 2

Começando uma figura

Para começar, devemos entender como o Ti*k*Z desenha os comandos na tela. A área onde o desenho será feito é tratada como um grande plano cartesiano, ou seja, cada ponto da área pode ser referenciado de forma única através de um par ordenado *(x,y)*, com a precisão decimal desejada. Assim, não será um problema caso queiramos indicar ao programa que o desenho será iniciado no ponto (1.3333, 0.999).

Neste curso abordaremos desenhos em duas dimensões, porém o Ti*k*Z possui também pacotes para desenhos em 3 dimensões que funcionam de forma bem semelhante, com coordenadas *(x,y,z)*.

Outro aspecto relevante da ferramente que vamos utilizar é que os comandos são desenhados na ordem em que aparecem no código. Isso ficará mais claro conforme construirmos nossas figuras.

Observação: nas primeiras figuras em Ti*k*Z, usar papel e caneta pode ser bastante útil! Principalmente quando o desenho tratar-se de uma figura geométrica.

- `\begin{tikzpicture}` e `\end{tikzpicture}`: iniciar e terminar uma figura em Ti*k*Z
- `\draw`: realizar desenhos de forma geral como figuras geométricas, retas e outros.
- `\node`: indicar locais onde desejamos inserir algum objeto (texto, forma ou figura, por exemplo).

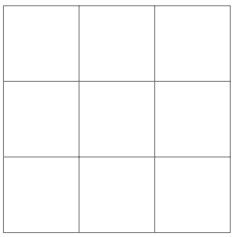
Os comandos podem possuir opções. Anteriormente acima usamos `\draw [opção]` para especificar cores, larguras, e outros detalhes que o desenho pode conter.

Observação: Lembre-se que os comandos terminam com um ponto e vírgula (;).

2.1 Malha quadriculada, segmentos de reta e texto na figura

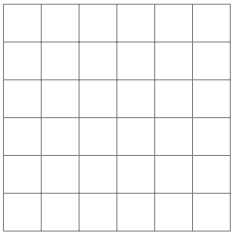
Malha quadriculada. Suponha que seja necessário facilitar o entendimento de uma figura que por exemplo represente uma função. Adicionar uma malha ao plano cartesiano para que os pontos sejam mais bem identificados é uma solução frequente, e podemos implementar de várias formas.

- `\draw[help lines] (0,0) grid (3,3);` é construída uma grade de linhas.



Por *default*, a malha é construída de forma que a marcação ocorra a cada uma unidade. Isso pode, porém, ser modificado da seguinte forma:

- `\draw[help lines, step=0.5] (0,0) grid (3,3);` nesse caso, a opção `step` faz com que a marcação da malha ocorra a cada meia unidade. O valor da opção pode ser modificado da forma que for mais conveniente.



Desenhando retas. O Ti*k*Z possui diferentes forma de se desenhar retas passando por 2 ou mais pontos que permitem trabalhar com as coordenadas de um ponto no plano, inclusive na forma polar. Abaixo veremos alguns de seus usos.

`\draw (0,0) -- (1,0);` Temos uma reta simples horizontal do ponto (0,0) até o ponto (1,0) ——

Se quisermos uma reta que passe por diferentes pontos, podemos aumentar a quantidade de pares ordenados informados. Por exemplo, o código `\draw`

`(0,0) -- (0,1) -- (1,1);` dá-nos a figura ⌜

`\draw[->] (0,0) -- (1,0);` A reta começa no ponto (0,0) e termina no ponto (1,0). Seu final possui uma seta ⟶

Esse estilo pode ser utilizado para produzir eixos de um plano cartesiano, como veremos.

Retas em coordenadas polares. A notação de pontos como um par ordenado no plano cartesiano é muito conveniente, porém ela pode ser limitada quando estamos trabalhando com figuras geométricas, onde coordenadas polares são muitas vezes necessárias.

Podemos construir retas no Ti*k*Z utilizando apenas operações geométricas. Para isso, usaremos a notação `(30:1cm)`, que significa 1cm na direção de 30 graus.

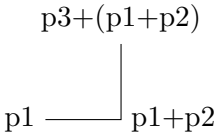
Por exemplo, `\draw [->] (0,0) -- (60:1) -- (30:1);` e `\draw [->] (0,0) -- (60:1) -- +(30:1);` resultam respectivamente em

Retas a partir de vetores. Ao desenhar uma reta podemos trabalhar também na forma vetorial, onde as coordenadas são os vetores que queremos desenhar.

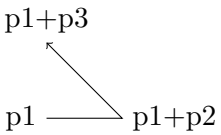
Observe que todas as somas de pontos que serão feitas a seguir consistem na soma dos valores de suas coordenadas. Então, se $a = (1, 2)$ e $b = (1, 0)$, $a + b = (2, 2)$.

Quando damos o vetor usando "`+`", desenhamos a reta saindo do ponto em que estamos até o ponto do vetor, que parte do ponto inicial. Caso queiramos criar um novo ponto de partida do vetor usamos "`++`".

`\draw (p1) -- ++(p2) -- ++(p3)`: o segmento parte de $p1$ e vai para o ponto $(p1 + p2)$. A partir desse ponto, anda então até o ponto que resulta na soma de $(p1 + p2) + p3$. No exemplo abaixo, $p1 = (1, 1)$, $p2 = (1, 0)$ e $p3 = (0, 1)$.

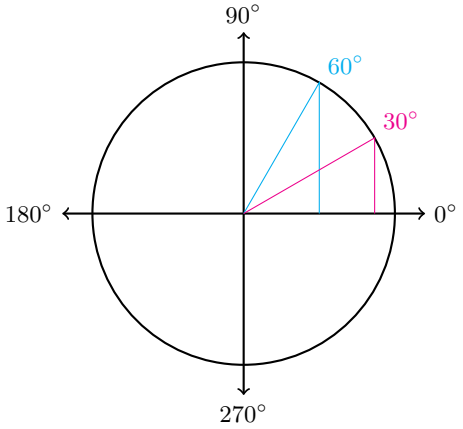


`\draw (p1) -- +(p2) -- ++(p3)`: a reta sai de $p1$ e vai para o ponto $(p1+p2)$. A partir desse ponto, então, anda até o ponto $(p1+p3)$.



Um exemplo em que podemos usar essa ferramenta é para desenhar a representação de seno e cosseno no círculo trigonométrico:

```
\begin{tikzpicture}

\draw [thick](0,0) circle (1cm);

\draw [<->, thick] (-1.2,0) --
(1.2,0) node [right] {0$^{\circ}$}
node [pos=0, left] {180$^{\circ}$};

\draw [<->, thick] (0,-1.2) --
(0,1.2) node [above] {90$^{\circ}$}
node [pos=0, below] {270$^{\circ}$};

\draw [cyan] (0,0) -- +(60:1) node
[above right] {{60$^{\circ}$}} --
+(0:1/2);

\draw [magenta] (0,0) -- ++(30:1)
node [above right] {{30$^{\circ}$}}
-- +(270:1/2);

\end{tikzpicture}
```



Observação: para entender melhor a diferença entre "`+`" e "`++`", teste com algumas figuras simples qual efeito cada variação causa.

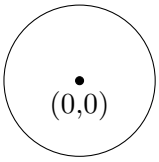
Eixos. Como muitas figuras matemáticas envolvem um plano cartesiano, veremos como fazer eixos para delimitá-lo.

- `\draw[<->] (1,0) -- (0,0) -- (0,1);` teremos os eixos da parte positiva do plano.

Inserindo textos nas figuras. Para inserir textos em figuras, vamos utilizar o comando `\node`. Assim como os outros comandos, ele possui opções registradas entre colchetes. Devemos especificar também onde o "nó" será feito.

Como exemplo, pense em um círculo de centro (0,0). Se quisermos indicar essa informação na nossa figura, teríamos o seguinte código:

```
\begin{tikzpicture}
    \draw (0,0) circle (1 cm);
    \draw [fill=black] (0,0) circle
(1.5 pt);
    \node [below] at (0,0) {(0,0)};
\end{tikzpicture}
```

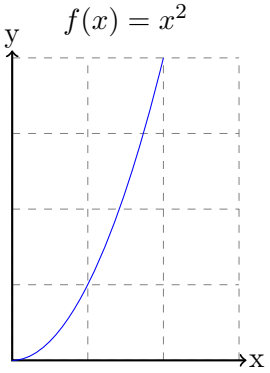


No último comando usamos o `\node` para escrever o par ordenado na figura. Devemos colocar o `{texto}` que desejamos entre as chaves e a `(posição)` entre parênteses e a `[posição relativa]` entre colchetes.

Os posicionamentos possíveis em relação a um ponto `(x,y)` são `above, below, left, right` (em cima, embaixo, esquerda, direita), ou ainda uma mistura deles como `below left` (embaixo à esquerda).

Por ser um pacote do LaTeX, os textos escritos em figuras construídas pelo Ti*k*Z terão forma semelhante. Expressões matemáticas são escritas entre `$`. Por exemplo, se quisermos escrever x^2, deveremos escrever na forma matemática `$x^2$`, como a seguir.

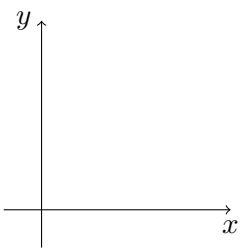
```
\begin{tikzpicture}

\draw[help lines, dashed] (0,0) grid
(3,4);
\draw [<->, thick] (0,4) -- (0,0) --
(3,0);
\draw [blue,-] (0,0) parabola (2,4);
\node [right] at (3,0) {x};
\node [above] at (0,4) {y};
\node at (1.5,4.5) {$f(x)=x^2$};

\end{tikzpicture}
```



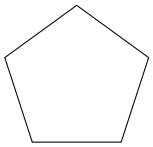
2.1.1 Atividades

Apresente os códigos correspondentes às figuras a seguir:

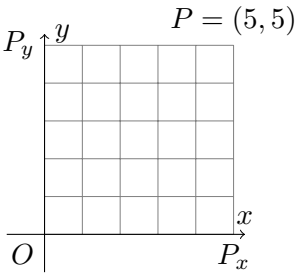
a) Sistema de eixos coordenados OXY.



b) Pentágono regular (use coordenadas polares).



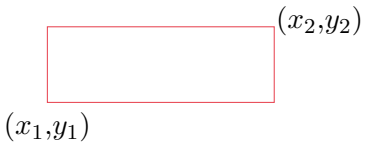
c) Sistema de eixos coordenados com malha e coordenadas de pontos.



2.2 Retângulos e círculos

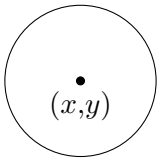
As figuras geométricas requerem atenção extra ao escrever seus comandos. É necessário estar atento ao que cada ponto significa e como cada mudança influencia no formato da figura.

Retângulos. Para desenhar retângulos, utilizamos o comando `\draw (x1,y1) rectangle (x2,y2);` que resulta no retângulo abaixo



Como podemos ver, os pares ordenados (x_1,y_1) e (x_2,y_2) representam as extremidades de uma diagonal.

Círculos. O comando `\draw (x,y) circle (r cm);` desenha um círculo de centro (x, y) e de raio r centímetros.

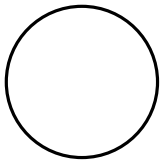


Veremos como colorir e modificar as formas descritas na próxima sessão.

2.3 Espessura das linhas, coloração e outras customizações

Espessura. Para ajustar a espessura (grossura) de uma linha, utilizamos os comandos `ultra thin, very thin, thin, thick, very thick ou ultra thick`. Com isso, a espessura irá diminuir ou aumentar.

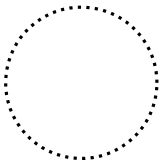
```
\draw[very thick] (0,0) circle (1 cm);
```



Estilos.

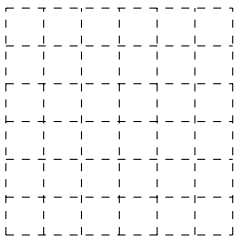
- `dotted`: deixa a linha pontilhada. Usando o exemplo acima, se `dotted` fosse passado como opção, a linha ficaria da seguinte forma :

  ```
  \draw[very thick, dotted] (0,0) circle (1 cm);
  ```

- `dashed`: deixa a linha tracejada. Usando o exemplo da malha quadriculada na sessão 4, temos o seguinte efeito:

  ```
  \draw[dashed, step=0.5] (0,0) grid (3,3);
  ```

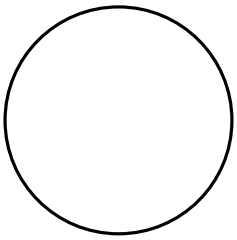
- `rounded corners`: usado para obter cantos arredondados. Em um retângulo, podemos obter bordas arredondados da seguinte forma:

  ```
  \draw[rounded corners, thick, red] (0,0) rectangle (3,1);
  ```

Escala. Caso queiramos diminuir ou aumentar a figura sem ter que mexer em todo o código, podemos usar este comando para aumentar a figura por completo.

```
\begin{tikzpicture}[scale=2]
\draw[very thick] (0,0) circle (1 cm);
\end{tikzpicture}
```



`xscale` e `yscale`: têm a mesma função, porém escalam apenas o eixo x e o eixo y, respectivamente.

Coloração.

- `fill`: preenche com alguma cor a região pretendida. Se quiséssemos preencher de azul o retângulo de bordas vermelhas que construímos acima, teríamos o seguinte código:

```
\draw[rounded corners, color=red,
fill=blue!20] (0,0) rectangle (3,1);
```

- `color`: em geral indica a cor que será usada para fazer o desenho em si. No caso acima, são as bordas vermelhas do retângulo. Pode ser utilizada tanto com formas geométricas quanto com retas e textos.
- `shade`: em português, sombreamento ou tonalidade. Podemos produzir seu efeito de duas formas:
  - `\fill` o sombreamento nos retângulos anteriores pode ser feito do azul para o vermelho da seguinte forma:

    ```
    \draw[left color=blue!60, right color=red!60] (0,0) rectangle
    (3,1);
    ```

  - `\shade`: este comando é feito especificamente para sombreamento. O mesmo retângulo teria o seguinte código:

```
\shade[left color=blue!60, right color=red!60] (0,0) rectangle
(3,1);
```

Há também algumas outras opções de sombreados e tonalidades que podem ser exploradas.

Observação: O Ti*k*Z possui as seguintes cores pré definidas: red, green, blue, cyan, magenta, yellow, black, gray, darkgray, lightgray, brown, lime, olive, orange, pink, purple, teal, violet, e white.

Essas cores podem ser customizadas para ficar mais claras ou mais escuras. Por exemplo,

. `\draw[fill=purple](0,0) circle (5 pt);` nos dá 

. `\draw[fill=purple!40](0,0) circle (5 pt);` nos dá 

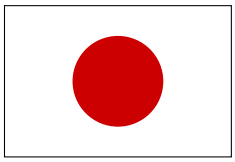
Observe que `!40` deixou a cor mais clara. Esse valor pode variar de 0 a 100 e representa a tonalidade da cor (mais clara à mais escura). Podemos também misturar duas cores, usando `cor1!80!cor2`, onde 80 substitui a porcentagem que se deseja da cor 1 em relação à cor 2, que nesse caso ficam 80% da cor 1 e 20% da cor 2.

É possível também criar suas próprias cores, que devem ser predefinidas no início do código. Veremos isso na sessão final do material.

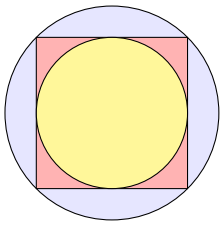
2.3.1 Atividades

Apresente os códigos correspondentes às figuras a seguir:

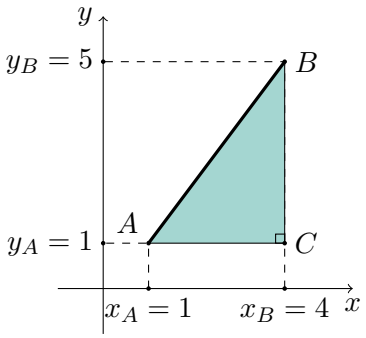
a) A bandeira do Japão. O retângulo é 3 por 2 e o círculo tem raio 0.6, no mesmo sistema de unidades. Use o vermelho red!80!black.



b) Circunferência inscrita e circunscrita a um quadrado. Use as cores blue!10, red!30 e yellow!50.



c) Comprimento do segmento de reta no sistema de coordenadas.



Capítulo 3

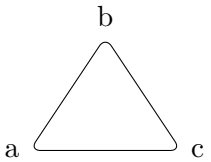
Definindo coordenadas e plotando gráficos de funções

3.1 Comando `\coordinate` e polígonos regulares

Comando `\coordinate`. Em algumas situações um mesmo ponto é utilizado várias vezes na mesma figura, e se depois de fazer a figura o resultado não for o esperado, trocar o ponto em todos os lugares em que ele foi utilizado pode ser um incômodo. Para situações como essa, utilizamos o comando `\coordinate`.

Por exemplo, suponha que queremos desenhar um triângulo cujas arestas sejam $a = (0, 0)$, $b = (1, 1.5)$ e $c = (2, 0)$ e também desejamos incluir as legendas para esses pontos. Uma maneira de se evitar reescrever as coordenadas dos pontos é nomear os pontos com comando `coordinate`.

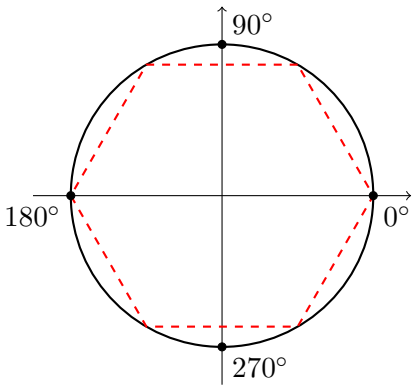
```
\begin{tikzpicture}
    \coordinate (a) at (0,0);
    \coordinate (b) at (1,1.5);
    \coordinate (c) at (2,0);
    \draw [rounded corners] (a) -- (b)
-- (c)--cycle;
    \node [left] at (a) {a};
    \node [above] at (b) {b};
    \node [right] at (c) {c};
\end{tikzpicture}
```



Esse recurso pode ser utilizado em construções menos triviais a fim de facilitar o raciocínio e a escrita do código. Note que foi usado `cycle` para construir um segmento do ponto onde se está, no caso (c), para o ponto inicial, no caso (a), formando um ciclo.

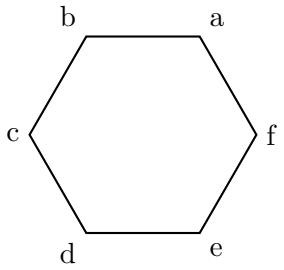
Observação: Um outro comando funciona de maneira semelhante à que vimos acima, `\node`. Ele foi utilizado anteriormente para inserir textos nas figuras, no entanto esse comando também pode ser utilizado para fixar não apenas pontos mas também objetos, como será apresentado.

Polígonos regulares. Utilizando as coordenadas polares podemos fazer qualquer polígono regular, desde que tenhamos o tamanho de cada aresta e os ângulos que são formados entre elas. Para montar um hexágono com 6 arestas de tamanho 1cm, podemos pensá-lo como sendo inscrito no círculo trigonométrico, como está representado a seguir:



Dessa forma, para descrevê-lo em comandos, iremos definir as coordenadas de seus vértices a, b, c, d, e, f utilizando `\coordinate` e sua forma polar em relação ao círculo trigonométrico.

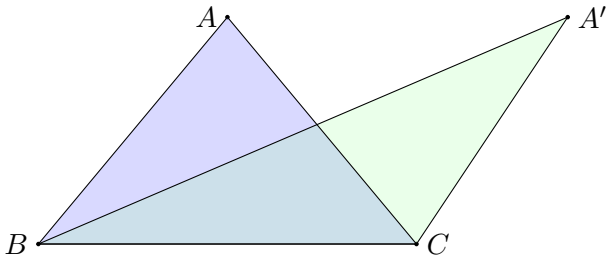
```
\begin{tikzpicture}
\coordinate (a) at (60:1);
\coordinate (b) at (120:1);
.
.
.
\coordinate (f) at (360:1);
\draw [thick](a) -- (b) --(c) -- (d) -- (e)
-- (f)--cycle;
\end{tikzpicture}
```



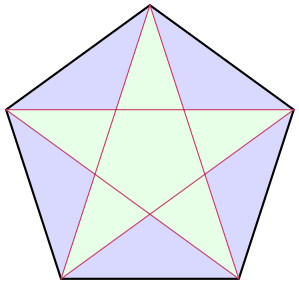
3.1.1 Atividades

Apresente os códigos correspondentes às figuras a seguir:

a) Ajustar pontos enquanto produz a figura fica muito mais fácil usando `\coordinate`. Use a opção `opacity=0.3` para tornar a figura menos opaca e mais transparente no comando `\filldraw` (desenhar e preencher).

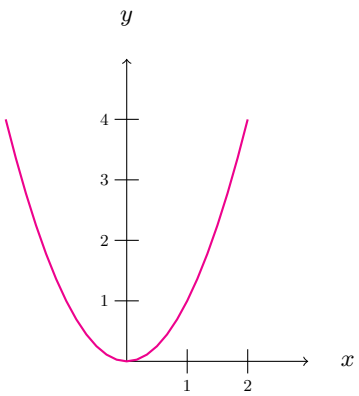


b) Estrela de cinco pontas com mesmos vértices de um pentágono regular. As cores usadas são `purple!80`, `green!10` e `blue!15`.



3.2 Gráficos de funções

Usando o eixo numerado como base, vamos plotar uma função quadrática. Assim como em outras formas utilizaremos o `\draw`, porém com alguns parâmetros diferentes. Veja abaixo:

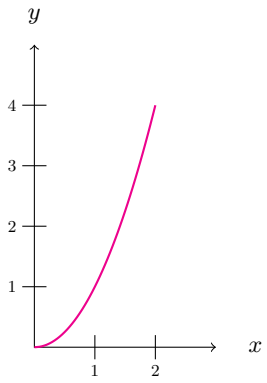



```
\draw [color=magenta,
thick, domain=-2:2] plot
(\x,{((\x)^2)});
```

Para plotar o gráfico de uma função informamos o domínio (`domain=-2:2`), a variável (`\x`) a função (`(\x)^2`).

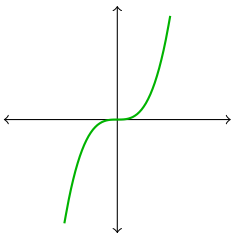
A parábola acima ficou fora do nosso plano cartesiano, por isso vamos mudar seu domínio de forma que apenas sua porção positiva seja exibida. Adicio-

namos também a opção `smooth` para que os contornos sejam suavizados.

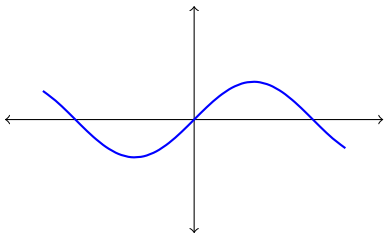



```
\draw [color=magenta, thick,
domain=0:3,smooth] plot
(\x,{((\x)^2)});
```

A função $f(x) = x^2$ é apenas um exemplo. É possível plotar inúmeros tipos de função (*Observação:* Para plotar função trigonométricas é necessário incluir o "`r`"como no exemplo). Veja os exemplos a seguir:



```
\draw [domain=-1.5:1.5, smooth,green,
thick] plot (\x,{(\x)^3})|
```

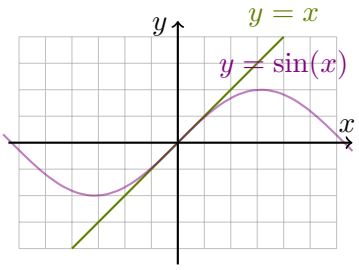


```
\draw [domain=-4:4, smooth,blue,
thick] plot (\x,{sin(\x r)});
```

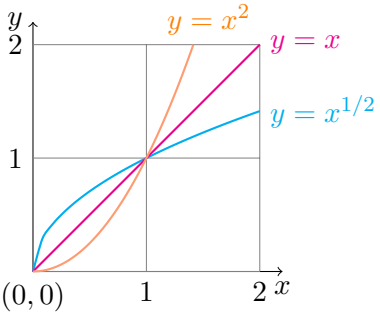
3.2.1 Atividades

Apresente os códigos correspondentes às figuras a seguir:

a) Use as opções `gray step=0.5` para a malha quadriculada.



b) Use cores diferentes para representar os gráficos das funções a seguir.



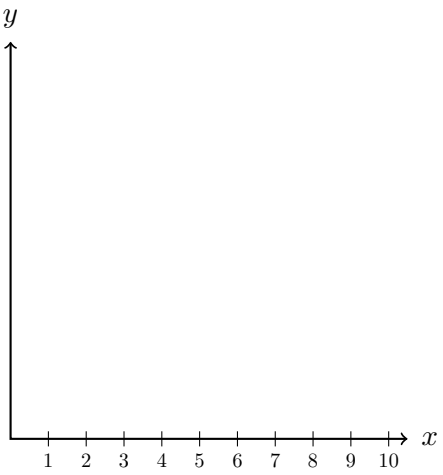
Capítulo 4

Comando `\foreach`, diagramas e marcação de ângulos

4.0.1 Comando `\foreach`

Suponha que se queira numerar os eixos de um plano cartesiano de 1 a 10. Com o que sabemos sobre Ti*k*Z até agora, faríamos da seguinte forma para numerar **apenas** o eixo x:

```
\begin{tikzpicture}
    %eixos
    \draw [<->, thick] (0,10.5) -- (0,0) -- (10.5,0);
    \node [right] at (10.6,0) {$x$};
    \node [above] at (0,10.6) {$y$};

    %traços de marcação no eixo x
    \draw (1,.2)--(1,-.2);
    \draw (2,.2)--(2,-.2);
    \draw (3,.2)--(3,-.2);
    .
    .
    .
    \draw (10,.2)--(10,-.2);

    %numeração no eixo x
    \node [below] at (1,-.2) {1};
    \node [below] at (2,-.2) {2};
    \node [below] at (3,-.2) {3};
    .
    .
    .
    \node [below] at (10,-.2) {10};
\end{tikzpicture}
```



Note que para cada unidade no eixo x precisamos de 2 linhas de comando para inserir um traço e o número em si. Se somarmos a quantidade de linhas que precisamos para numerar o eixo x com a quantidade de linhas necessárias para o eixo y, teremos no total 40 linhas só para numerar eixos!

Em figuras que necessitam de marcações de acordo com um padrão, utilizamos o comando `\foreach` (em português, "*para cada*") a fim de facilitar o trabalho. No caso acima, temos 4 marcações que repetem um padrão: os traços para marcar unidades nos eixos x e y e a escrita de seus respectivos valores. Os traços nos eixos possuem tamanhos iguais (no caso acima, `(x,0.2) -- (x,-0.2);`), e a numeração ocorre a cada unidade.

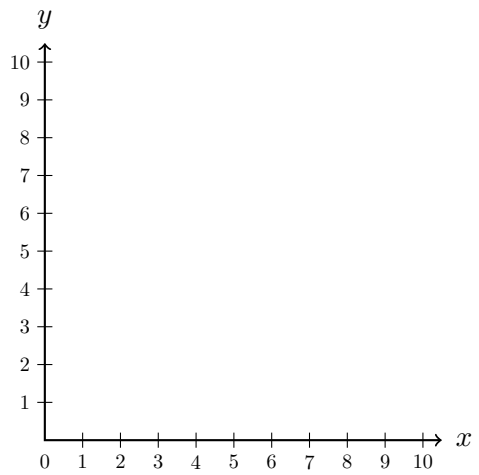
Em outras palavras, podemos descrever o tracejado como o seguinte: "*para cada unidade no eixo x, desenhe o segmento que liga os pontos* $(x, 0.2)$ *e* $(x, -0.2)$." Analogamente, escrever os valores de cada ponto do eixo é equivalente a "*para cada unidade no eixo x, escreva abaixo de seu ponto o valor do mesmo.*"

No código a seguir a variável`\x` representa uma variável que poderia ter qualquer nome. Ela assume valores diferentes ao decorrer do programa. Podemos interpretar a linha de comando abaixo como: para cada x inteiro, no intervalo de 0 a 10, escreva no ponto (x,-0.2) qual o valor de x.

```
\foreach \x in {0,1,2,...,10} \node [below] at (\x,-.2) {\x};
```

Utilizando o `\foreach` no código anterior temos:

```
\begin{tikzpicture}
   %eixos
   \draw [<->, thick] (0,10.5) --
(0,0) -- (10.5,0);
   \node [right] at (10.6,0) {$x$};
   \node [above] at (0,10.6) {$y$};
   \bide [below] at (0,-.2) {0};

   %traços de marcação no eixo x
   \foreach \x in {1,2,...,10} \draw
(\x, .2) -- (\x,-.2);

   %numeração no eixo x
   \foreach \x in {1,2,...,10} \node
[below] at (\x,-.2) {\x};

   %traços no eixo y
   \foreach \y in {1,2,...,10}  \draw
[-] (-.2,\y)--(.2,\y);

   %numeração no eixo y
   \foreach \y in {1,2,...,10} \node
[left] at (-.2,\y) {\y};
\end{tikzpicture}
```

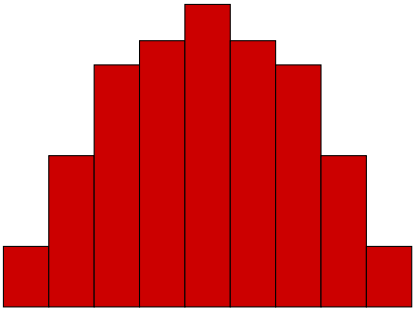


Observe que adicionamos nossas marcações também ao eixo y e temos menos linhas de código. Para cada numeração e traço em cada eixo, foi necessário apenas 1 linha. Isto significa que saímos de 40 linhas para apenas 4 usando o `\foreach`.

O símbolo `%` usado no código serve para indicar comentários, ou seja, linhas que não devem ser consideradas pelo programa durante a compilação. Usar comentários é útil para tornar o seu código mais claro para outras pessoas lendo. Também ajuda quando há algum erro durante a compilação e não sabemos qual é.

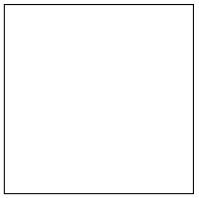
Histogramas. Outra utilidade do `\foreach` é na construção de histogramas:

```
\begin{tikzpicture}
[xscale=0.75]

\foreach \x/\y in
{0/1,1/2.5,2/4,3/4.4,4/6,
5/5.5,6/5,7/2.5,8/1} \draw
[fill=red!80!black] (\x,0)
rectangle (\x+1,\y/2);

\end{tikzpicture}
```



Neste caso introduzimos duas varáveis, `\x` e `\y`, que devem estar separadas por "`/`", tanto na definição quanto no código, e nesse caso usamos `\x` para os pontos na base do histograma e `\y` para as alturas do retângulo.

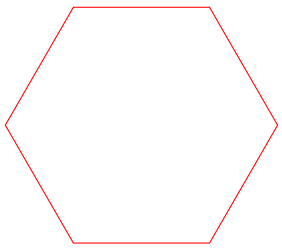
Coordenadas. É possível também criar coordenadas usando o `\foreach`. O funcionamento é um pouco diferente do normal. As coordenadas devem ser as primeiras variáveis, e em vez de usarmos `\coordinate` precisamos usar `\node [coordinate]`:



```
\begin{tikzpicture}

\foreach \x/\y in
{(1,1)/a,(0,1)/b,(0,0)/c,(1,0)/d}
\node [coordinate] (\y) at \x {};

\draw (a) -- (b) -- (c) -- (d) --
cycle;


\end{tikzpicture}
```

No código acima, quando `\x` assume o valor $(1, 1)$, a variável `\y` assume o valor b e assim sucessivamente.

Por outro lado, coordenadas polares funcionam de forma usual.



```
\begin{tikzpicture}

\foreach \x/\y in
{a/60:1,b/120:1,c/180:1,d/240:1,
e/300:1,f/360:1} \coordinate (\x) at
(\y);

\draw [red] (a) -- (b) -- (c) -- (d)
-- (e) -- (f) -- cycle;

\end{tikzpicture}
```

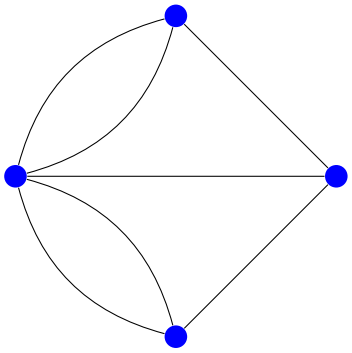
Atividade

Desenhe um decágono regular com os vértices contendo os rótulos $A_1, A_2, ..., A_{10}$ e todas as diagonais a partir de A_1.

4.0.2 Uso de `nodes` para construir diagramas

Até agora usamos `\node` para inserir textos nas figuras. No entanto, esse comando (que significa "*nó*") é utilizado de forma semelhante ao `\coordinate`. Podemos fixar não apenas pontos do plano, mas também outras formas. Aqui falaremos de círculos e retângulos.

A grande vantagem de usar `node` é o fato de poderem ter posição relativa a outros nodes que criamos. Observe o exemplo a seguir:



```
\begin{tikzpicture}[node
distance=3cm]

\tikzstyle {ponto}=[fill=blue,circle,
minimum size = 3pt]

\node [ponto] (1) {};
\node [ponto] (2) [above right of=1]
{};
\node [ponto] (3) [below right of=2]
{};
\node [ponto] (4) [below left of=3]
{};

\path
(1) edge  [bend left] (2)
(2) edge [bend left] (1)
(2) edge (3)
(3) edge (4)
(1) edge (3)
(1) edge [bend right] (4)
(4) edge [bend right] (1);

\end{tikzpicture}
```

Temos alguns elementos novos nesta figura. Primeiramente, definimos que a distância entre dois nós seja de 3 cm (ela também pode ser ajustada individualmente). Em seguida introduzimos o comando `\tikzstyle`, que nos permite definir as características que os nós terão. Nesse caso escolhemos chamá-los de "ponto" (deve estar entre chaves) com preenchimento azul.

A opção `circle` define que o ponto terá a forma de um círculo (poderíamos usar `rectangle` para desenhar um retângulo), e seu tamanho mínimo de 3pt (caso adicionemos texto, o `node` pode ter um tamanho maior do que este). Agora, todos os `nodes` criados terão as mesmas características.

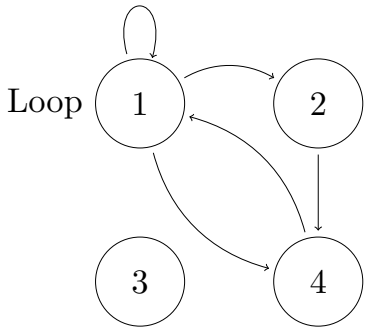
No segundo grupo de código, definimos os `(nomes)` e suas `[posições]`. Se não escolhermos nenhuma posição ele ficará automaticamente no (0,0). A partir do nó (1) colocamos nosso nó (2) acima e à direita de (1) (`above right`), (3) abaixo e à direita de (2) e (4) abaixo e à esquerda de (3).

Usando o comando `\path` podemos desenhar um caminho entre cada elemento. Para tanto devemos especificar como cada `node` liga-se com o comando `edge`. Por exemplo, se quisermos fazer um caminho entre (2) e (3) escrevemos `(1) edge (2)`. O comando `\path` pode ligar vários `nodes`, mas é necessário colocar o ponto e vírgula apenas no final de todos os caminhos

que serão desenhados.

A forma que os caminhos serão construídos é alterada de acordo com a especificação dada nas opções. Ou seja, no caso da figura acima ligamos (1) a (2), mas colocamos a opção `[bend left]` para que o caminho entre os dois faça uma curva para a esquerda.

Na figura a seguir mostramos como podemos desenhar indicação de *loops* nos nodes (que podem ser por cima, por baixo etc.). Também colocamos algumas opções no início, sendo elas `shorten`, que reduz o comprimento do caminho, podendo ser feito no início (`>=`) ou no final (`<=`) e também que todos os nossos nodes sejam direcionados (como fizemos com nossos eixos de coordenadas). Adicionamos no nosso node (1) a opção `label=left:`, que nos permite adicionar texto também fora do node.

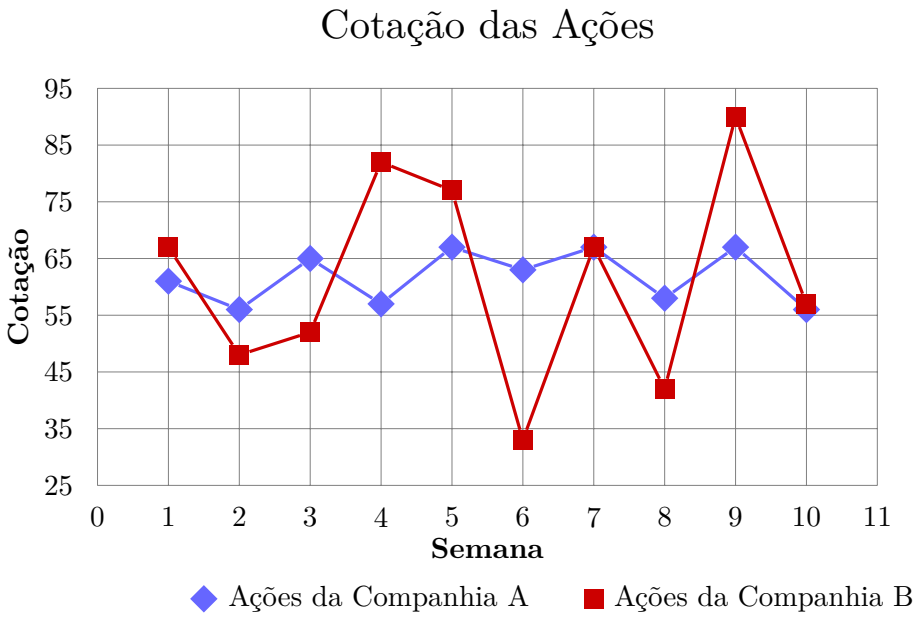



```
\begin{tikzpicture}[node
distance=2cm,->, shorten >=0.1cm,
shorten <= 0.1cm]

\tikzstyle {bola}=[circle, draw,
minimum size = 1cm]

\tikzstyle {quadrado}=[rectangle,
draw, minimum size=1cm]

\node [bola] (1) [label=left:Loop]
{1};
\node [bola] (2) [right of=1] {2};
\node [bola] (3) [below of=1] {3};
\node [bola] (4) [below of=2] {4};

\path
(1) edge [bend right] (4)
(4) edge [bend right] (1)
(1) edge [bend left](2)
(1) edge [loop above] (1)
(2) edge (4)
;
\end{tikzpicture}
```

Há muito mais possibilidades de opções e comandos para nodes, essas são apenas algumas delas que escolhemos para demonstrar o poder desta ferramenta.

Atividade

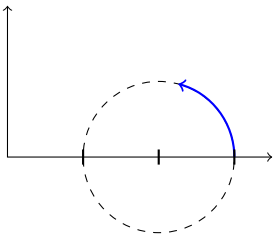
Usando `nodes`, desenhe o seguinte gráfico (utilize aproximações para cada ponto):



4.1 Comandos \arc, \clip e a marcação de ângulos

Comando \arc. Observe a seguir como o arco azul é descrito em Ti*k*Z. Indicamos primeiro a partir de qual ponto vamos iniciar o desenho. Informamos o ângulo inicial, o ângulo final e o raio da curva separados por dois pontos. A seta do desenho mostra em qual sentido a curva foi feita para que possamos entender o processo.

```
\begin{tikzpicture}
\draw [blue,->](3,0) arc (0:75:1cm);
\end{tikzpicture}
```



Nesse caso, a curva azul começa no ponto (3,0), possui ângulo inicial 0°, ângulo final de 75° e raio igual a 1 centímetro. Visualizar o arco sobre o plano cartesiano pode ajudar a entender como cada um desses parâmetros altera a curva. Para tanto, temos tracejado o círculo de raio 1cm, cujo centro está a essa mesma distância do ponto de partida da curva.

O centro (a, b) da circunferência tracejada na figura anterior pode ser encontrado para qualquer arco. Considere então a curva dada por `\draw (x,y) arc (inicial:final:raio)`. Seu início é no ponto `(x,y)`, o ângulo inicial $\alpha =$ `inicial`, ângulo final $\theta =$ `final` e de raio r igual a `raio`. Portanto o centro (a, b) será dado por:

$$(a, b) = (x - r\cos\alpha, y - r\sin\theta)$$

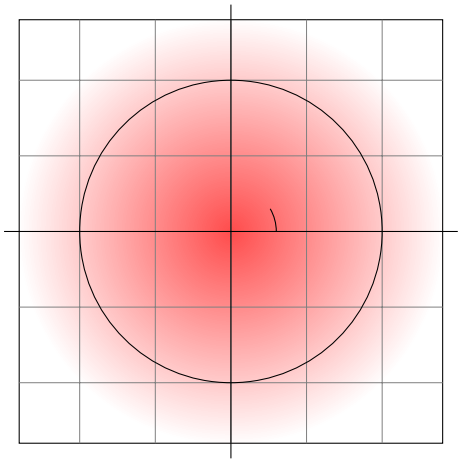
O ponto final da curva (a ponta da seta azul acima) é o par ordenado (x', y') dado por:

$$(x', y') = (x - r\cos\alpha + r\cos\theta, y - r\sin\alpha + r\sin\theta)$$

Comando \clip. Para dar ênfase ou apenas delimitar uma região de uma figura, usamos o comando `\clip`.

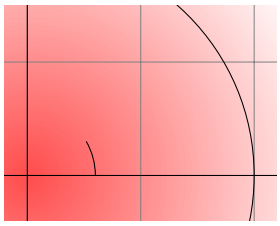
Este exemplo foi adaptado do manual *The TikZ and PGF Packages*, de Till Tantau disponível em http://www.texample.net/media/pgf/builds/pgfmanualCVS2012-11-04.pdf.

Considere a figura:



```
\begin{tikzpicture}
    \draw[inner color=red!70,
outer color=white] (-1.4,-1.4)
rectangle (1.4,1.4);
    \draw[step=.5cm,gray,very
thin] (-1.4,-1.4) grid
(1.4,1.4);
    \draw (-1.5,0) -- (1.5,0);
    \draw (0,-1.5) -- (0,1.5);
    \draw (0,0) circle (1cm);
    \draw (3mm,0mm) arc
(0:30:3mm);
\end{tikzpicture}
```

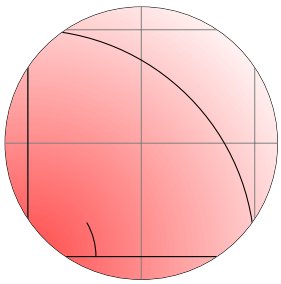
Vamos inserir no início `\clip (x1,y1) rectangle (x2,y2);`. Ele funciona de forma parecida ao comando `\draw`, exceto que não produz nenhum desenho e sim delimita uma região. Esse retângulo vai funcionar como uma janela, ou seja, será exibida apenas a região da figura que estiver dentro desta janela. O `\clip` terá efeito em todo o código abaixo dele.



```
\begin{tikzpicture}
    \clip (-0.1,-0.2) rectangle
(1.1,0.75);
    .
    .
    .
    \draw (3mm,0mm) arc
(0:30:3mm);
\end{tikzpicture}
```

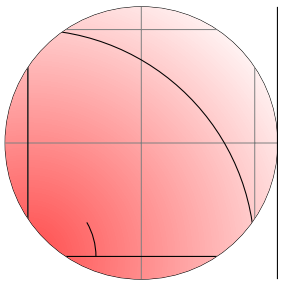
Tal delimitação contribui para enfatizar um trecho da figura. No caso acima, foi possível visualizar melhor o pequeno arco dentro da circunferência.

Podemos utilizar os comando `\clip` e `\draw` ao mesmo tempo. Por exemplo, podemos usar `\draw` com a opçãp `clip` e vice-versa (`\draw[clip]` ou `\clip[draw]`).



```
\begin{tikzpicture}
    \clip[draw] (0.5,0.5)
circle (.6cm);
    .
    .
    .
    \draw (3mm,0mm) arc
(0:30:3mm);
\end{tikzpicture}
```

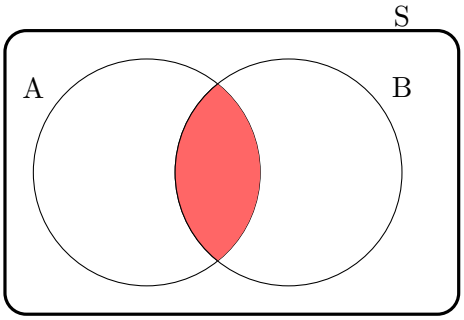
O comando `\clip` faz com que todos os códigos abaixo dele sejam exibidos apenas se estiverem na região delimitada por ele, portanto, se quisermos adicionar outros códigos fora do `\clip` devemos colocá-los no ambiente do Ti*k*Z chamado `scope`.



```
\begin{tikzpicture}
    \begin{scope}
        \clip[draw] (0.5,0.5)
circle (.6cm);
        .
        .
        .
        \draw (3mm,0mm) arc
(0:30:3mm);
    \end{scope}
    \draw (1.1,-0.1) --
(1.1,1.1);
\end{tikzpicture}
```

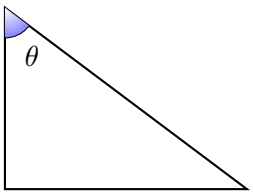
Um outro exemplo bom de uso do `clip` é para marcar interseções em diagramas de Venn:

```
\begin{tikzpicture}

\draw [very thick,rounded
corners=8pt, samples=200]
(0,-0) -- (0,5) -- (8,5) --
(8,0) -- cycle;

\node at (7,4) {B};
\node at (.5,4) {A};
\node at (7,5.25) {S};
\draw [samples=200] (5,2.5)
circle (2cm);

\clip [draw, samples=200]
(2.5,2.5) circle (2cm);
\draw [fill= red!60,
samples=200] (5,2.5) circle
(2cm);

\end{tikzpicture}
```



Marcação de ângulos. O comando `\clip` será muito útil para tarefas como colorir regiões específicas e fazer indicação de ângulos, como veremos agora.

Para marcar ângulos em figuras, podemos usar um arco, porém é mais simples usar o `clip` de um círculo delimitado por nossa figura. Na figura a seguir, o ângulo θ está destacado



O código usado na construção é:

```
\draw [clip, thick] (0,3) -- (4,0) -- (0,0) -- cycle;
```

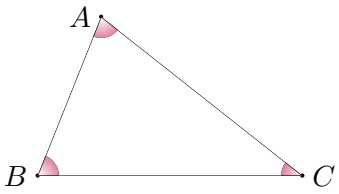
Para marcar o ângulo, construímos o círculo de centro `(0,3)` e raio `.5cm`. No entanto, apenas a porção interior ao triângulo é mostrada. Isso foi possível graças ao `\clip`. Com eles delimitamos a região do triângulo de modo que apenas o que está contido nessa região seja mostrado.

.
.
.

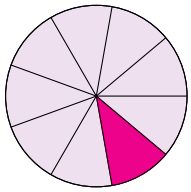
```
\clip (0,3) -- (4,0) -- (0,0);
\draw [inner color=white, outer color=blue!50] (0,3) circle (.5cm);
```

4.1.1 Atividades

a) Faça a figura a seguir usando os comandos `\clip` e `\coordinate`.



b) Use os comandos `\foreach` e `\clip` para construir a figura a seguir.



Capítulo 5

Outras utilidades

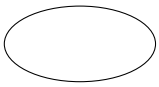
Definindo cores. Além de poder misturar tonalidades de cores como por exemplo em `[color=red!20]`, é possível criar cores prenomeadas com código RGB. Por exemplo, para colorir o círculo a seguir de verde-oliva, no preâmbulo do código definimos a cor "`meuverde`" e informamos o código RGB da cor.

```
    \definecolor{meuverde}{RGB}{128,128,0}
%verde-oliva

    \begin{tikzpicture}
        \draw[fill=meuverde, ultra
thick] (0,0) circle (.5cm);
    \end{tikzpicture}
```

Outros códigos RGB podem ser encontrados na internet sem dificuldade.

Observação: é possível criar sua própria paleta de cores que permita daltônicos diferenciar as tonalidades sem ambiguidade. O material possui alguns estudos, explicações e os códigos em RGB das cores sugeridas pelo mesmo. Pode ser acessado em http://mkweb.bcgsc.ca/colorblind/.

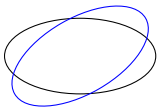
Elipses. Semelhante ao comando para se desenhar círculos, para uma elipse de centro (x, y) e raio horizontal rx e raio vertical ry, usamos `\draw (x,y) ellipse (rx cm and ry cm);`.



Rotação de textos e figuras. Se quisermos girar um texto para seguir uma determinada reta, ou então rotacionar uma figura inteira, podemos usar o comando `rotate`. Usando a Elispse anterior como exemplo, temos

Vitória - ES / UFES - IV Simpósio Nacional da Formação do Professor de Matemática - Vitória - ES / UFES - IV Simpósio Nacional d

Em **preto**: `\draw (0,0) ellipse (1cm and .5cm);`
Em **azul**: `\draw [rotate=30, blue] (0,0) ellipse (1cm and .5cm);`



Opção shift. Se uma figura ou apenas uma parte dela, como nodes ou outros objetos, não estiverem na posição que queremos, podemos usar a opção `shift` para ajustar sua posição. Isso pode ser feito com a opção `shift={(x,y)}`, sendo `(x,y)` as dimensões do deslocamento. Por exemplo, se escrevemos `shift={(2,3)}`, o nosso objeto irá se deslocar 2 centímetros para cima e 3 para a direita. Caso se queira mover apenas horizontalmente ou verticalmente podemos usar `xshift=2` e `yshift=3`. Diferente do `shift` e dos comandos usuais, que têm suas medidas em centímetros, `xshift` e `yshift` são medidos em pt (aproximadamente 0.3527mm). Mas podemos alterar para centímetros especificando na dimensão. Por exemplo: `xshift=1cm` irá mexer o objeto desejado em 1cm para a direita.

Opção every node/.style={opção}. Quando queremos alterar as propriedades de todos nodes da figura, este comando é muito útil, já que os nodes não acompanham as mesmas alterações que fazemos nas opções

Por exemplo, se usamos `scale`, o que será escalado serão apenas as retas, formas ou caminhos (funções, polígonos etc.), não dos nodes. Entretanto, `every node/.style={scale=2}` irá duplicar o tamanho de todos os nodes. Portanto, se quisermos que os nodes acompanhem a escalonagem da figura devemos adicionar esta opção. Também serve se quisermos que todos os nossos nodes sigam determinadas propriedades (como as exemplificadas nos diagramas).

Lista de símbolos matemáticos

Aqui estão listados alguns exemplos de símbolos e operadores matemáticos (é importante para usar a maioria dos símbolos matemáticos os pacotes `amsmath` e `amssymb` no preâmbulo do texto em LaTeX).

Δ	`\Delta`	$\cong$	`\cong`
δ	`\delta`	$\perp$	`\perp`
$\subset$	`\subset`	$\varnothing$	`\varnothing`
$\in$	`\in`	$\iff$	`\iff`
$\cup$	`\cup`	$\mathbb{R}$	`\mathbb{R}`
$\neq$	`\neq`	$\sin$	`\sin`

Para a lista completa, recomendamos o *site*:

https://oeis.org/wiki/List_of_LaTeX_mathematical_symbols

Observação: Lembre-se que os símbolos devem ser escritos entre `$ $`.