

Resumo dos Comandos

- Lista de comandos a ser tratada

- Plot
- Plot3D
- ContourPlot
- ContourPlot3D
- ParametricPlot
- ParametricPlot3D
- StreamPlot
- VectorPlot
- Show

1 Plot

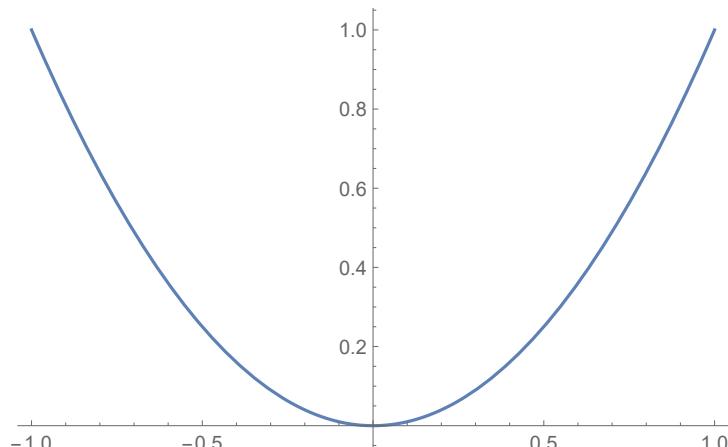
Normal recebe uma função que depende de somente uma variável seguida da definição do intervalo de amostragem desejado. Então imagine que se deseja plotar a função do segundo $f(x) = x^2$, para fazer isso basta tomar o lado direito da função e utilizar como primeiro parâmetro do Plot como segue.

```
1 Plot[x^2]
```

Porém, não é adequado parar por aqui, é necessário definir a variação de x em uma dimensão que vai abranger o “teatro” escolhido. Dessa maneira, imagine que queiramos plotar essa função de modo a abranger os valores de x que vão de -1 a 1, então aplicamos $\{x, -1, 1\}$. Como consequência, percebe-se que o programa lança valores de x abrangendo o domínio explicitado de modo a conformar a representação gráfica no menor espaço possível de visualização do gráfico para os valores de x propostos

```
1 Plot[x^2, {x, -1, 1}]
```

Da forma como está, se dermos **enter** o comando será rodado e vamos obter



Repare que sendo $x = -1$ ou $x = 1$, $f(x) = 1$, logo o máximo valor de f mostrado é 1, não precisamos a variação de y , até mesmo por que o Plot não aceitaria tal sintaxe.

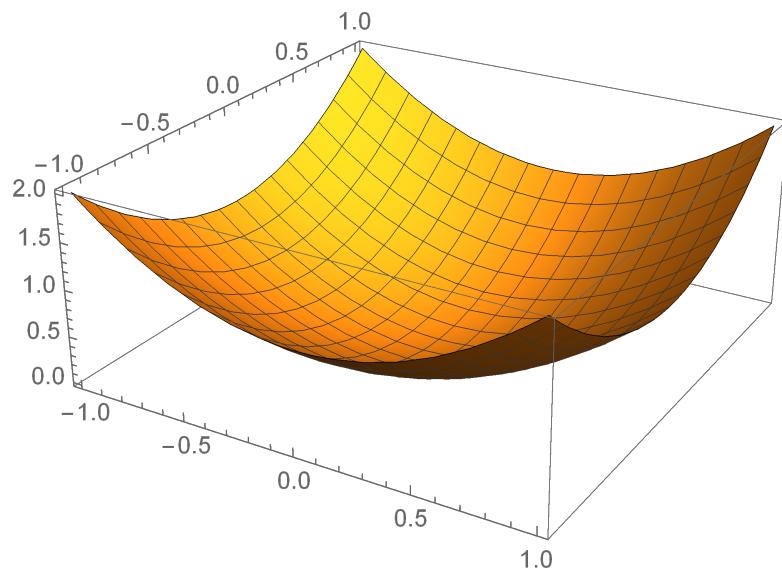
2 Plot3D

Podemos interpretar esse comando como expansão do `Plot`, porém em três dimensões. Sabendo que um ponto no espaço agora depende de x e y , consequentemente a função $f(x)$ em duas dimensões passará a ser $f(x, y)$. De forma análoga ao que foi feito para o comando anterior, imagine que queiramos plotar um paraboloide no espaço que obedeça

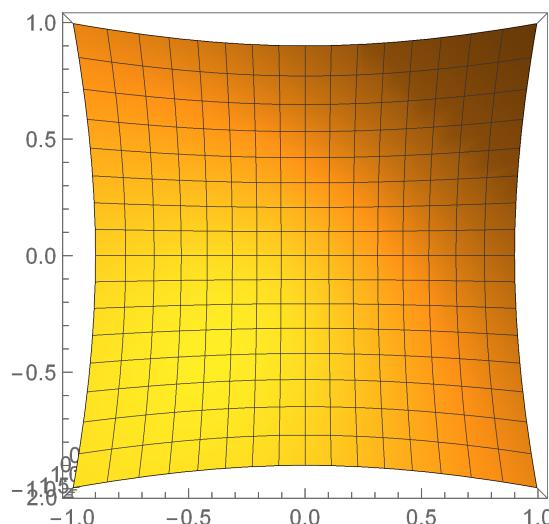
$$f(x, y) = x^2 + y^2 \quad (1)$$

Numa abordagem rápida, podemos imaginar tal superfície como a rotação de uma parábola tal como é visto em 2 em torno do eixo z no espaço, então sabendo que dependemos de mais uma variável será preciso acrescentar outro parâmetro ao `Plot3D` como segue

```
1 Plot3D [x^2+y^2, {x, -1, 1}, {y, -1, 1}]
```

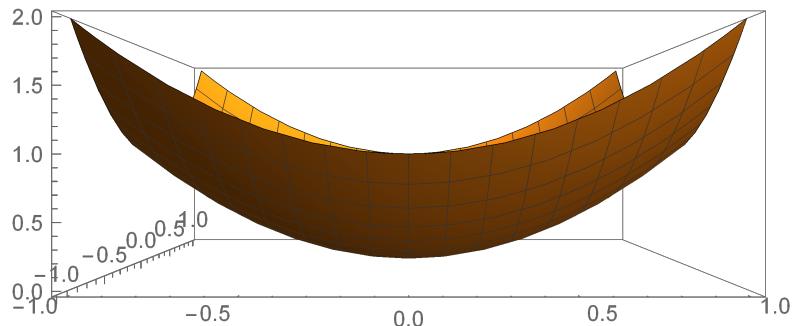


Note que se clicarmos com o botão direito do *mouse* e formos na opção *Top View* veremos



Ou seja, o intervalo de valores escolhido para x e y novamente é notado no domínio presente. Vê-se que a vista superior é um quadrado de lados valendo 2 unidades já que tanto x quanto y variam entre menos -1 e 1.

Partindo para análise em z , vamos mudar a opção de *Top View* para *Front View*. Fazendo uma análise visual vemos que a máxima altura da “caixa” que comporta o gráfico é de duas unidades. Isso ocorre pois o máximo valor que $f(x, y)$ pode assumir para o domínio escolhido ocorre quando as coordenadas são $(-1, -1)$, $(-1, 1)$, $(1, -1)$ ou $(1, 1)$ como vemos a seguir



Sabemos disso devido a simplicidade do gráfico e prevemos que a função é sempre crescente no intervalo dado.

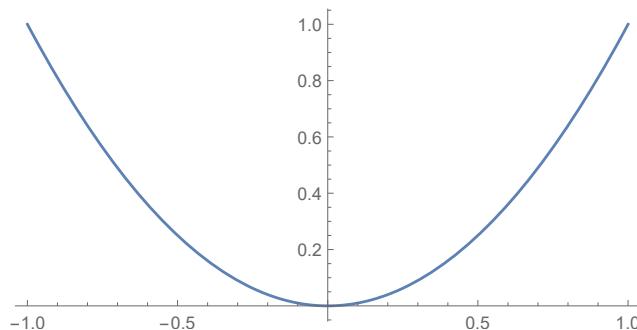
Alguns atributos de estilo podem ser passados para os *Plots* de modo a melhorar a visualização ou visando cumprir alguma meta de representação gráfica. Alguns dos comandos a seguir são interessantes de serem entendidos pelo menos no básico.

Para o *Plot* (Gráficos em 2d no geral)

- **AspectRatio**

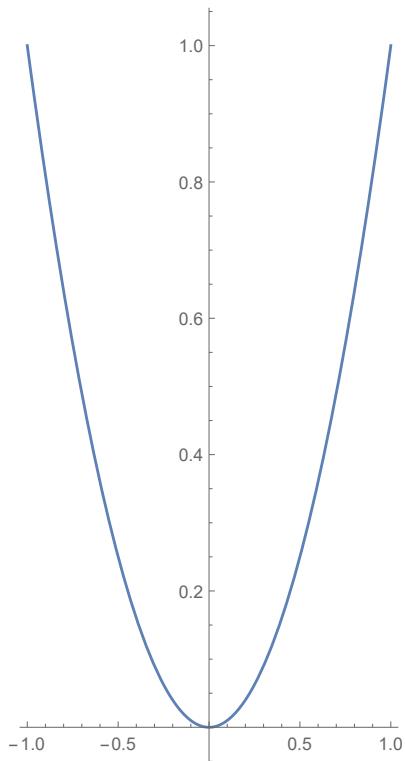
- Permite editar a proporção entre os eixos do gráfico. Imagine que quiséssemos uma proporção de 1 em x para dois em y , teríamos uma razão igual a 0.5 e podemos prever que o gráfico terá maior comprimento em x que em y como vemos a seguir com base nas linhas de comando

```
1 Plot[x^2, {x, -1, 1}, AspectRatio -> .5]
```



Trocando por 2 o valor da propriedade vemos

```
1 Plot[x^2, {x, -1, 1}, AspectRatio -> 2]
```



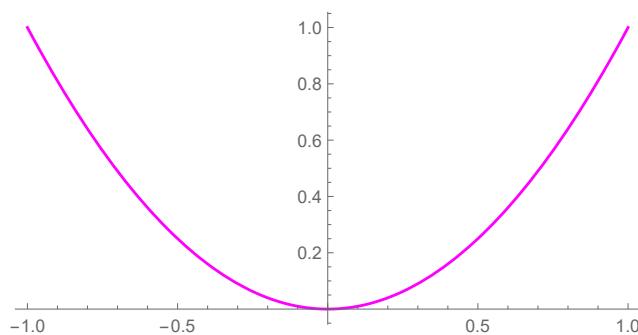
- **PlotStyle**

- Um dos atributos mais utilizados, possibilita mudar a cor do gráfico, espessura das linhas, opacidade, entre muitos outros parâmetros.

- * Mudando a cor

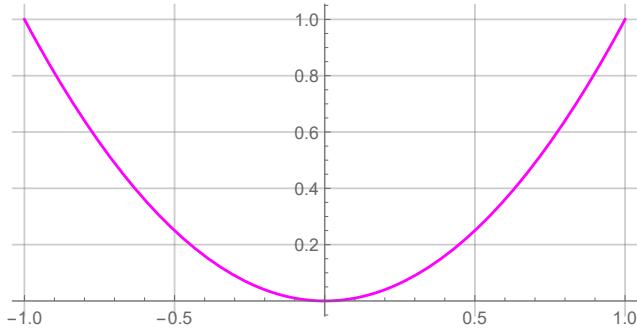
- Uma maneira aconselhável é passar diretamente o atributo de cor, como por exemplo `Magenta`, `Red`, `Blue`, `Cyan` ou utilizar o `RGBColor` para customizar com maior liberdade como segue

```
1 Plot[x^2,{x,-1,1},AspectRatio -> .5,PlotStyle -> Magenta]
```



- **GridLines**

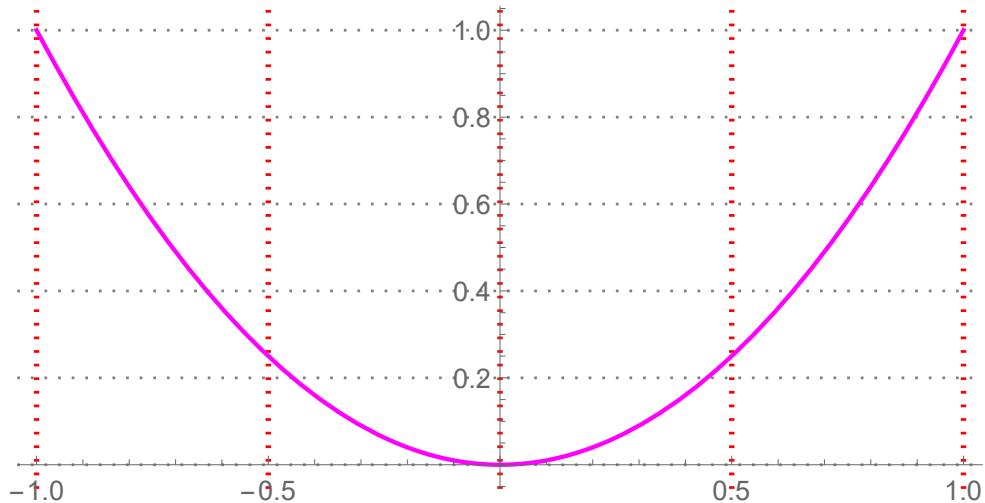
- Nos permite adicionar uma malha quadriculada ao fundo da plotagem. Esse recurso muitas vezes facilita a associação entre os eixos coordenados e nos permite ter maior precisão quantitativa para aproximações feitas visualmente. Abaixo temos alguns exemplos.



```
1 Plot[x^2,{x,-1,1},AspectRatio->.5,PlotStyle->Magenta,GridLines
->Automatic]
```

Por padrão o `GridLines` com `Automatic` renderiza linhas cheias, mas podemos alterá-las passando `GridLinesStyle -> {{Dashed}, {Dashed}}`. Isso fará com que tanto as linhas horizontais quanto as verticais se tornem tracejadas. De forma semelhante usamos o `Dotted` para deixá-las pontilhadas. O atributo de cor também pode ser passado no interior das chaves como vemos a seguir

```
1 Plot[x^2,{x,-1,1},AspectRatio->.5,PlotStyle->Magenta,GridLines
->Automatic,GridLinesStyle ->{{Red, Dotted, Thick},{Gray,
Dotted}}]]
```



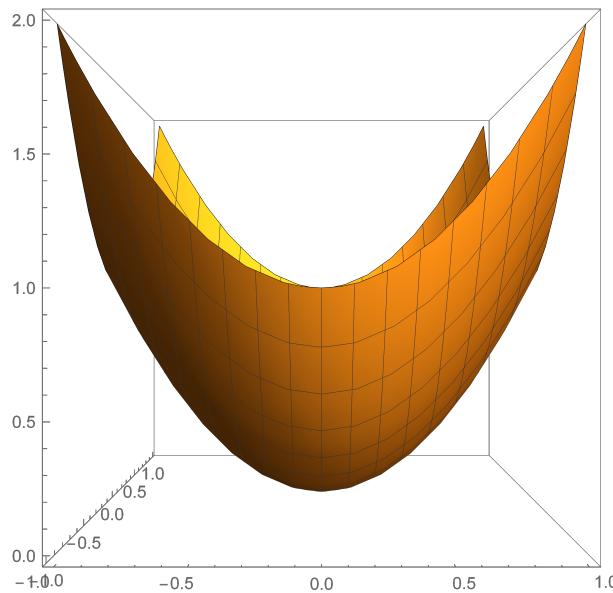
Aumentando-se a proporção do gráfico vê-se que a primeira lista de valores contendo `Red`, `Dotted` e `Thick` tornou as linhas verticais vermelhas, pontilhadas e, como foi passado o `Thick` houve uma leve aumentada na espessura, somente para melhorar a visualização do leitor.

Para o Plot3D (Gráficos em 3d no geral)

- **BoxRatios**

- Esse atributo é análogo ao **AspectRatio**, porém fazemos sua aplicação para três dimensões com base numa lista de três posições indicando a proporção em cada eixo como segue

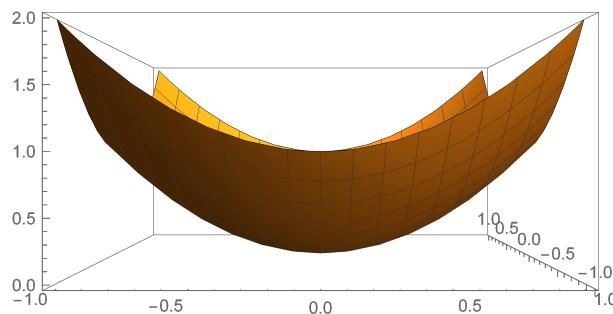
```
1 Plot3D [x^2+y^2,{x,-1,1},{y,-1,1},BoxRatios ->{1,1,1}]
```



Como é visto, a proporção do gráfico foi aumentada ocasionando um estreitamento de sua base e aumento de sua altura na escala.

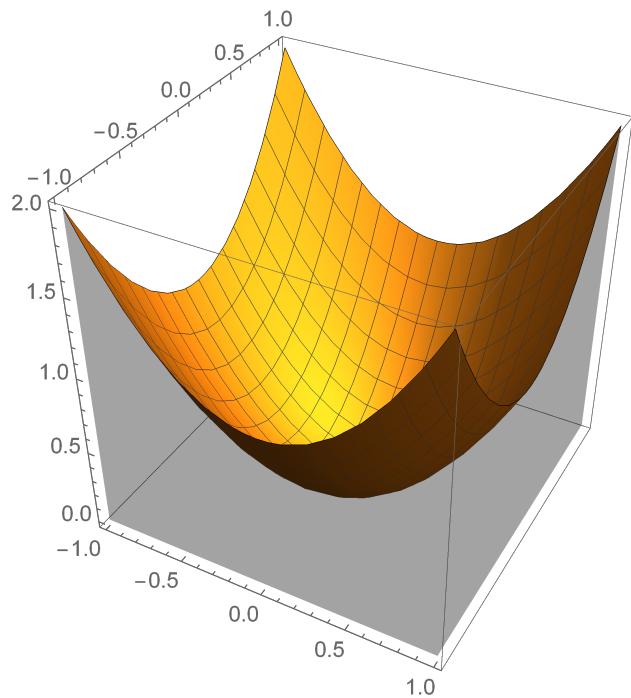
Caso quiséssemos testar fazer uma figura com o eixo z com metade da proporção, note a diferença

```
1 Plot3D [x^2+y^2,{x,-1,1},{y,-1,1},BoxRatios ->{1,1,.5}]
```



- **Filling**

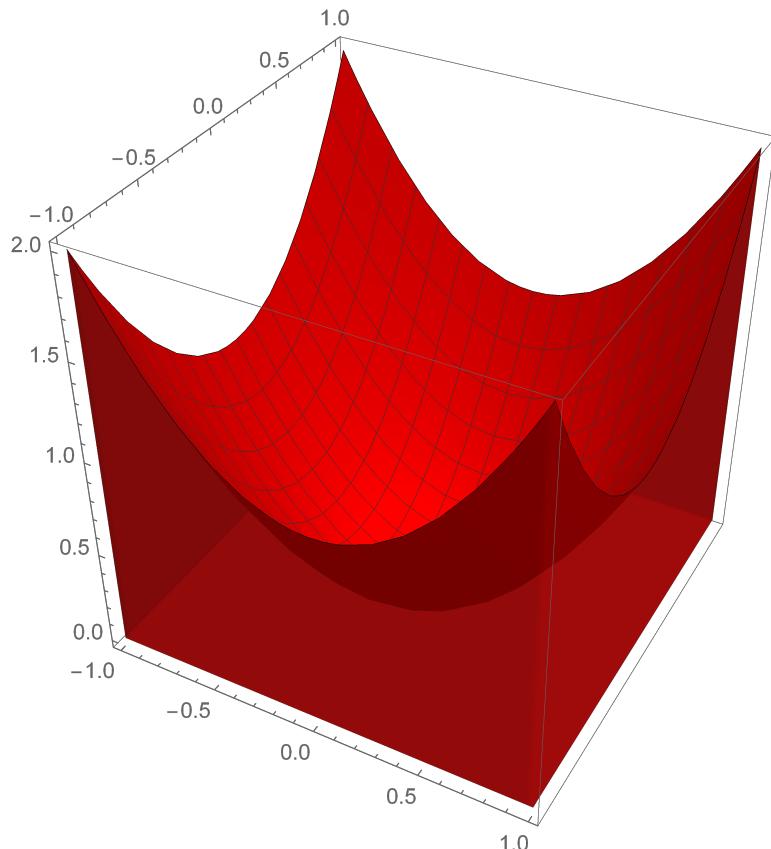
- Ferramenta bastante útil na estilização dos gráficos, nos permite criar um sombreado preenchendo a parte inferior das superfícies. Também é aplicável no Plot



```
1 Plot3D[x^2+y^2,{x,-1,1},{y,-1,1},Filling->Bottom]
```

* Aplicando o **FillingStyle**, até então não mencionado podemos obter

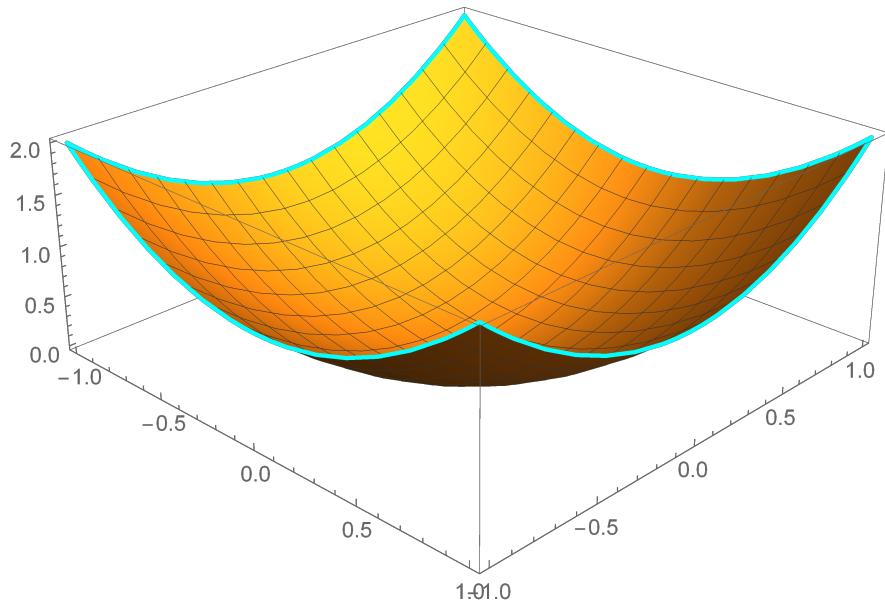
```
1 Plot3D[x^2+y^2,{x,-1,1},{y,-1,1},Filling->Bottom,
          FillingStyle->Opacity[.8],PlotStyle->Red]
```



- **BoundaryStyle**

Cria um contorno ao longo das bordas da superfície.

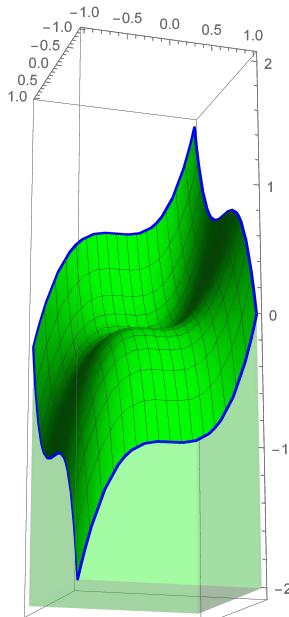
```
1 Plot3D[x^2+y^2, {x, -1, 1}, {y, -1, 1}, BoundaryStyle -> {Thick, RGBColor [0, 255, 255]}]
```



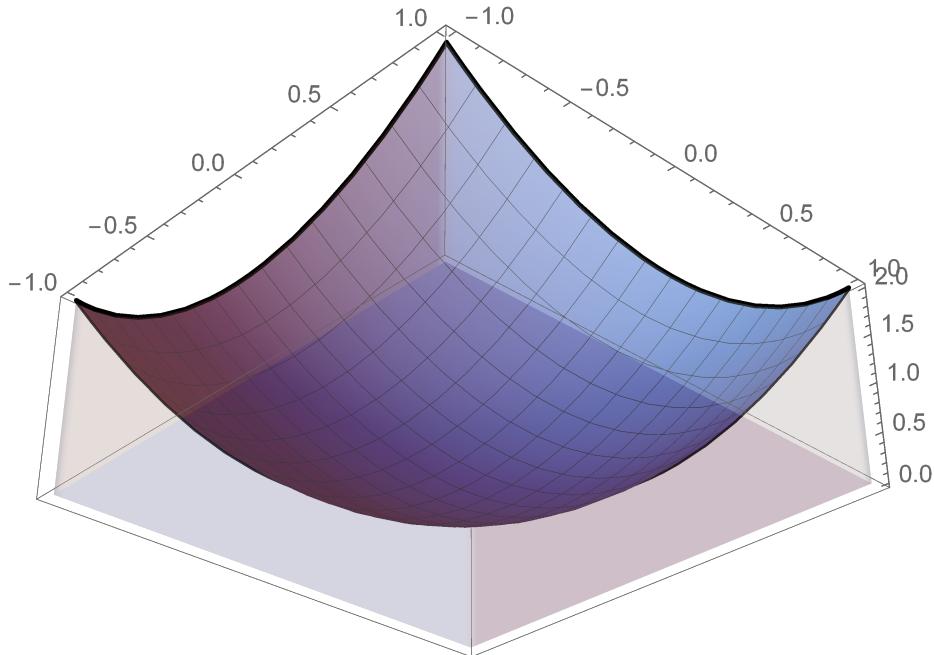
- **PlotStyle**

Vamos resgatar algumas propriedades de estilização já conhecidas mostrando alguns exemplos de superfícies

```
1 Plot3D[x^3 + y^3, {x, -1, 1}, {y, -1, 1}, PlotStyle -> {Green},  
2 BoundaryStyle -> {Thick, Blue}, Filling -> Bottom,  
3 FillingStyle -> {Opacity[.2]}, BoxRatios -> {1, 1, 3}]
```



```
1 Plot3D[x^2 + y^2, {x, -1, 1}, {y, -1, 1}, PlotStyle -> {
  LightBlue}, BoundaryStyle -> {Thick, Blue}, Filling ->
  Bottom, FillingStyle -> {Opacity[.2]}]
```

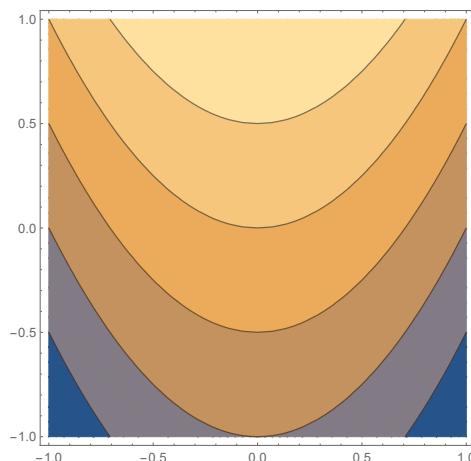


3 ContourPlot

Esse comando nos permite plotar os contornos de funções a depender de como passamos os parâmetros. Podemos imaginar que caso passarmos uma expressão seguindo a mesma estrutura do Plot vamos ter a replicação da função f ao longo do eixo y em diferentes alturas. Vamos exemplificar para tornar mais fácil.

Imagine que desejamos novamente plotar a função do segundo grau $f(x) = x^2$, só que dessa vez, ao invés de plotar somente uma curva queremos uma família de funções que seguem o mesmo molde porém transladas em y . Dessa forma, passaríamos o comando como é visto abaixo

```
1 ContourPlot[y - x^2, {x, -1, 1}, {y, -1, 1}]
```

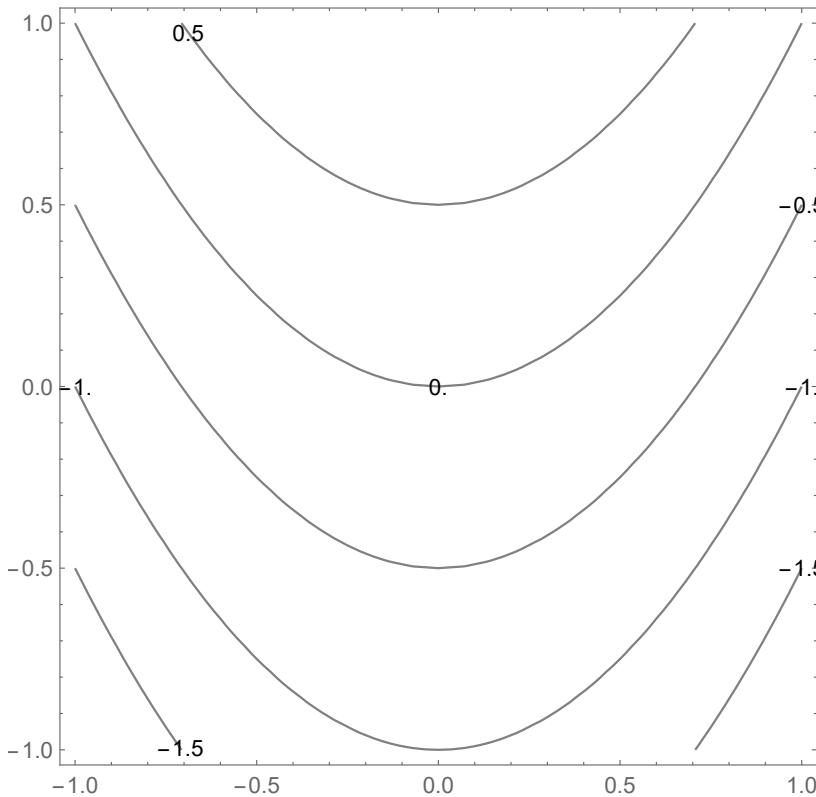


Afinal, o que foi feito para que chegássemos nessa conformação de curvas? A resposta é simples. Simplesmente assumimos que $f(x) = x^2$, trocamos $f(x)$ por y e subtraímos x^2 de ambos os lados, logo

$$y - x^2 = 0 \quad (2)$$

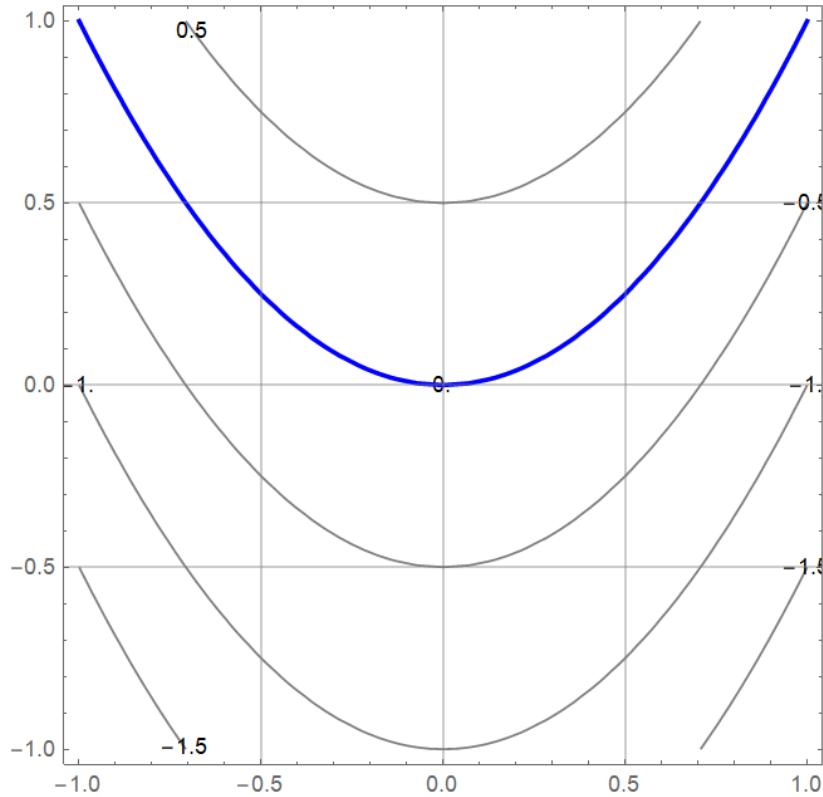
Isso significa que temos um contorno da função f cujo rótulo corresponde a 0. Ou seja, o valor nulo encontrado demonstra que a função f ao longo de toda a curva $y - x^2$ está rotulado como 0 em três dimensões (Altura constante ao longo dessa parábola). Nesse caso, como estamos tratando de rótulos um atributo interessante de ser usado é o **ContourLabels**. Ele vai mapear a altura de cada parábola na representação em duas dimensões. Vejamos

```
1 ContourPlot[y-x^2,{x,-1,1},{y,-1,1},ContourLabels-> True,
  ContourShading ->False]
```



Vemos o mapa de contornos referido. Para melhorar a visualização empregou-se o **ContourShading->False**. Note que o *label* 0. indica o contorno que foi discutida anteriormente. Para comprovarmos, usando o **ContourPlot**, vamos reforçar essa parábola com base na equação do contorno $y - x^2 = k$, onde k corresponde ao contorno 0.

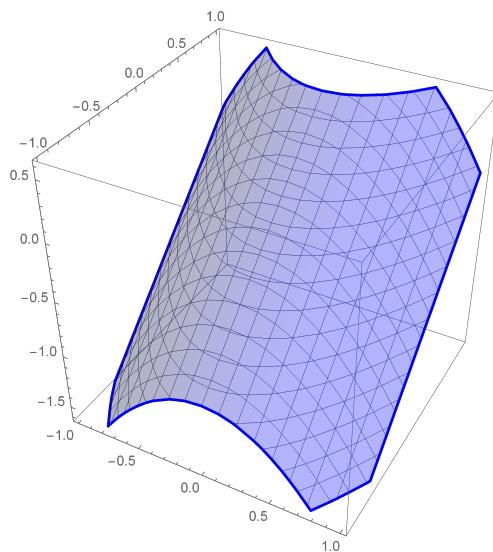
```
1 Show[ContourPlot[{y-x^2},{x,-1,1},{y,-1,1},ContourLabels->True,
  ContourShading ->False,GridLines ->Automatic],ContourPlot[
2 y-x^2==0,{x,-1,1},{y,-1,1},ContourStyle ->{Thick,Blue}]]
```



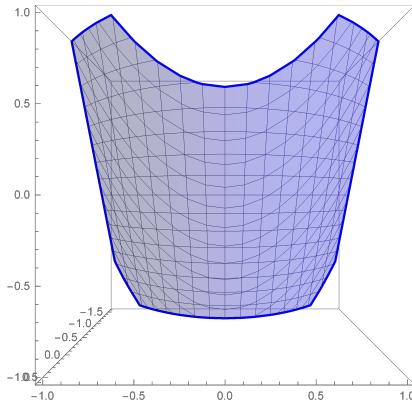
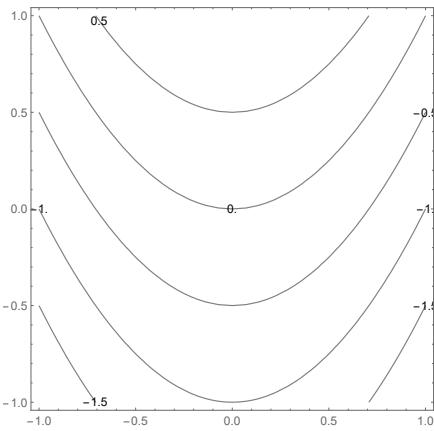
Repare que utilizou-se o `Show` para juntarmos os dois `ContourPlots`. O primeiro representa os varias contornos $y-x^2==k$, enquanto o segundo o contorno de rótulo nulo.

Usando esse tópico de âncora vamos falar do `ContourPlot3D`. Imagina se quiséssemos representar esse contorno com $k = 0$ no espaço. Com base no gráfico em 2d ao olhar de cima para baixo notamos que a parábola superior possui $k = 0.5$ e vai diminuindo até $k = -1.5$. Isso demonstra que a superfície da parábola está diminuindo sua altura nesse intervalo com base na interpretação desses vários valores de k (ou curvas de nível). Vejamos como fica em três dimensões

```
1 superficie = ContourPlot3D[z==y-x^2,{x,-1,1},{y,-1,1},{z,-1.6,.6},
2 ContourStyle ->{Blue,Opacity[0.3]},BoundaryStyle ->{Thick,Blue}]
```

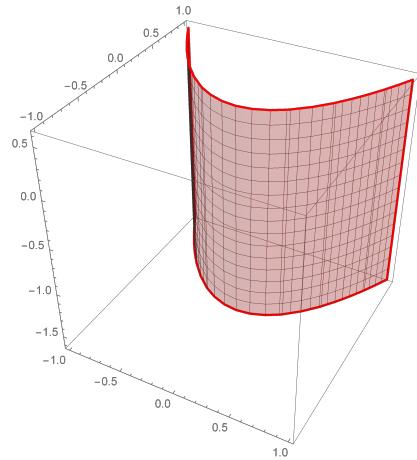
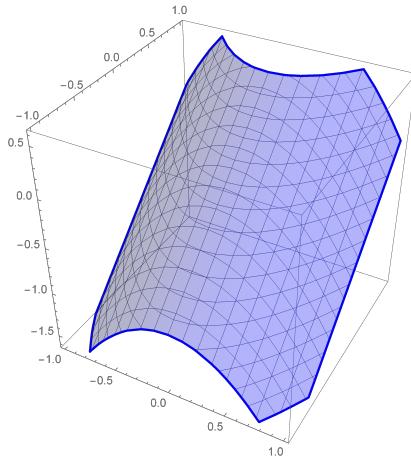


O gráfico acima revela somente os vários valores de k que agora podem ser interpretados de forma mais qualitativa. Se mudarmos a vista, podemos fazer a comparação das projeções



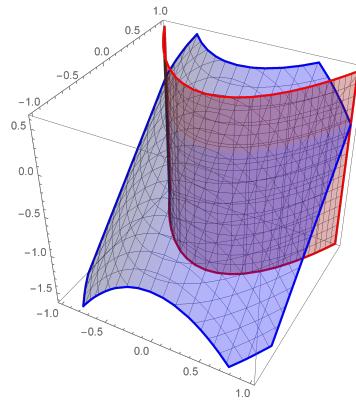
Um conceito importante está relacionado à ideia de que uma curva em duas dimensões, no espaço torna-se uma superfície. Tendo isso em mente, concluímos que a curva mais espessa identificada anteriormente para $k = 0$ em duas dimensões, agora torna-se uma superfície que não depende de z . Vamos verificar

```
1 superficie2 = ContourPlot3D [0==y-x^2,{x,-1,1},{y,-1,1},{z,-1.6,.6},
ContourStyle ->{Red,Opacity[0.3]},BoundaryStyle ->{Thick,Red}]
```



Juntando as duas

```
1 Show [superficie,superficie2]
```

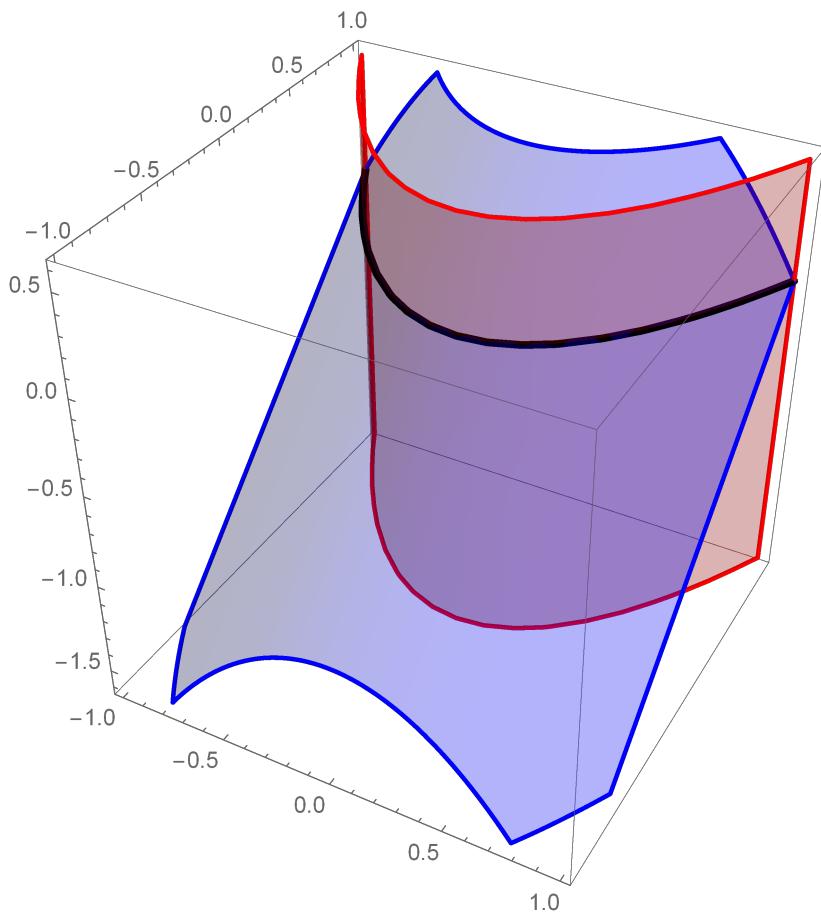


Após fazer algumas mudanças, podemos ver com ainda mais clareza

```

1 superficie = ContourPlot3D[z==y-x^2,{x,-1,1},{y,-1,1},{z,-1.6,.6},
2 ContourStyle->{Blue,Opacity[0.3]},BoundaryStyle->{Thick,Blue},
3 Mesh->None,PlotPoints ->100];
4
5 superficie2 = ContourPlot3D[0==y-x^2,{x,-1,1},{y,-1,1},{z,-1.6,.6},
6 ContourStyle->{Red,Opacity[0.3]},BoundaryStyle->{Thick,Red},
7 Mesh->None];
8
9 curve = ContourPlot3D[y-x^2==0,{x,-1,1},{y,-1,1},
10 {z,-.00001,.00001},ContourStyle->{Red,Opacity[0.3]},
11 BoundaryStyle->{Thickness [.01],Black},Mesh ->None];
12
13 Show[superficie, superficie2, curve]

```



4 ParametricPlot

Esse comando é bastante empregado como uma alternativa ao Plot ou ContourPlot. Com ele podemos fazer representações semelhantes, porém de forma mais simplificada e igualmente entendível. O primeiro posição que passamos diz respeito a uma lista de parâmetros em x , y e z , a depender se estamos tratando de duas ou três dimensões.

Um exemplo clássico de ser tratado nessa etapa diz respeito à parametrização de uma circunferência, que nos remete à ideia de duas dimensões. Nesse exemplo,

vamos fazer uso das coordenadas polares como forma de descrever a curva no plano. Dessa forma, da geometria analítica sabemos que as variáveis x e y assumem

$$\begin{cases} x = r \sin(t) \\ y = r \cos(t) \end{cases}$$

Para a circunferência alvo de nosso estudo vamos imaginar que ela possui raio unitário, dessa forma teremos

$$\begin{cases} x = \sin(t) \\ y = \cos(t) \end{cases}$$

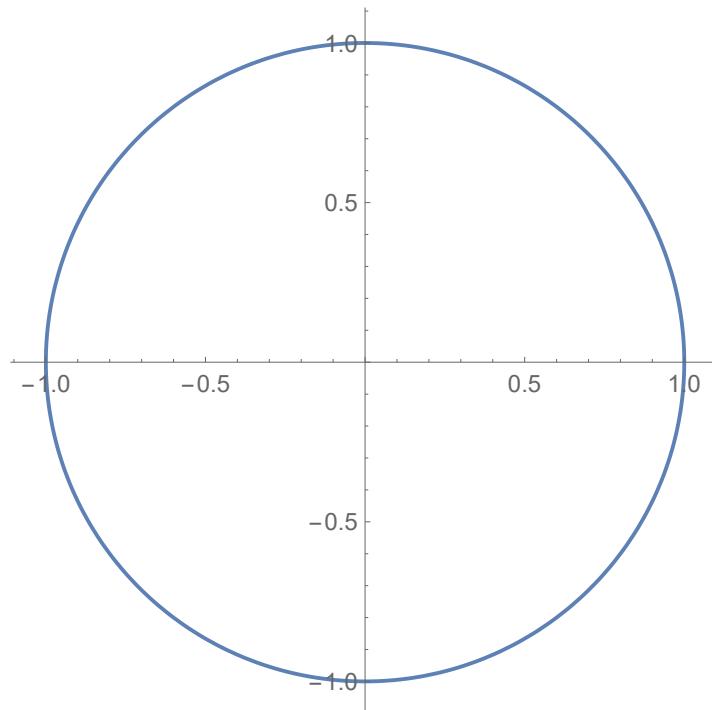
Para adequarmos à sintaxe do `ParametricPlot` mostrada abaixo

```
ParametricPlot[{fx, fy}, {t, tmin, tmáx}]
```

Vamos imaginar que a função que descreve o traçado da circunferência depende do tempo (t), assim quando $t = 0$ a curva não começou a ser traçada, mas quando $t = \pi$, metade dela já foi renderizada. Tendo esse pensamento em mente, chegamos que quando $t = 2\pi$ a representação da circunferência está completa. Dessa forma, o `ParametricPlot` alcança a seguinte forma após assumirmos que x e y dependem de t

```
1 ParametricPlot[{\$Sin[t], \$Cos[t]}, {t, 0, 2Pi}]
```

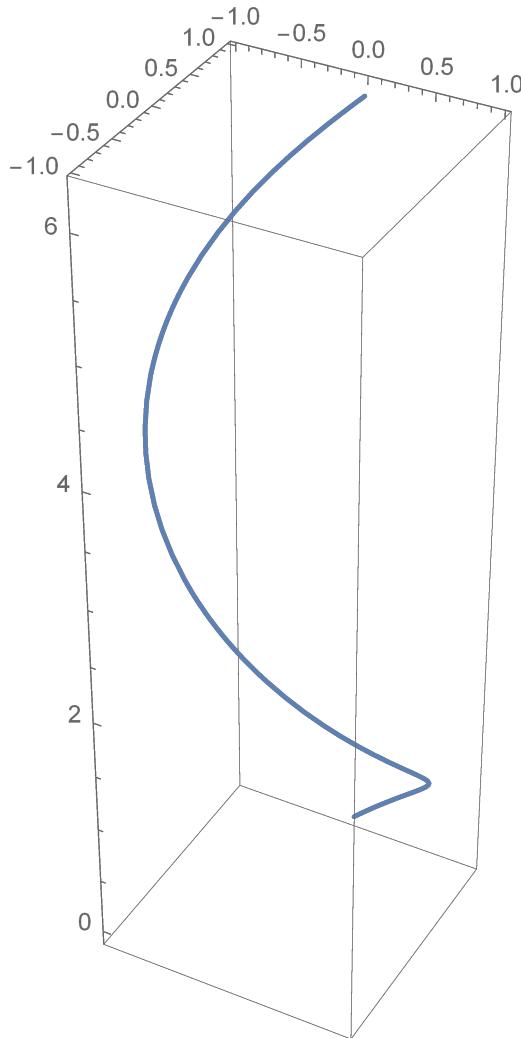
Podemos ler a linha acima como: “Substitua t no intervalo de 0 a 2π na função paramétrica de coordenadas $(\sin(t), \cos(t))$.”



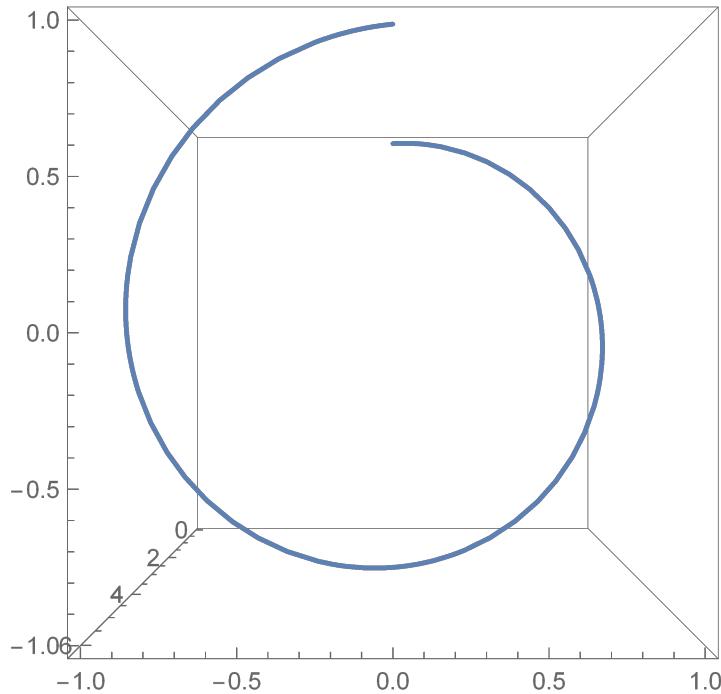
Repare que poderíamos assumir que $x = \cos(t)$ e $y = \sin(t)$ e isso só influenciaria no sentido da parametrização. Ou seja, ao invés de ser no sentido horário seria no anti-horário, mas o resultado final é o mesmo.

Para o **ParametricPlot3D** temos somente o acréscimo de uma nova posição na função paramétrica. Dessa forma, mudamos de $\{f_x, f_y\}$ para $\{f_x, f_y, f_z\}$. Vamos usar exemplo bastante análogo ao anterior, porém enquanto a circunferência é traçada em x e y vamos fazê-la ganhar altitude em z . Vejamos o exemplo

```
1 ParametricPlot3D[{Sin[t], Cos[t], t}, {t, 0, 2Pi}]
```



Mas o que houve nesse caso? Simples, ao adicionar o parâmetro t na terceira posição da função paramétrica ocasionamos o deslocamento dos pontos em z como dito anteriormente. Dessa forma, vê-se que não só o ângulo t vai de 0 a 2π nas funções \sin e \cos , mas a altura do gráfico também. Isso ocasiona na cota 6 mostrada em z (valor próximo dos $2\pi \approx 6,28\dots$) gerando a conformação de espiral mostrada. Como sabemos que f_x e f_y não foram modificadas se alterarmos a vista de *Default* para *Top*



veremos que a circunferência ainda existe na vista superior

Vamos supor agora que queremos mostrar dois pontos. Um estará no início da curva em espiral – quando $t = 0$ – e outra ao final ($t = 2\pi$). A ideia é simples, basta substituirmos esses valores de t na função paramétrica e vamos obter

$$\begin{cases} p_1 = (\sin 0, \cos 0, 0) \Rightarrow (0, 1, 0) \\ p_2 = (\sin 2\pi, \cos 2\pi, 2\pi) \Rightarrow (0, 1, 2\pi) \end{cases}$$

Para representarmos p_1 e p_2 graficamente vamos usar o comando **Graphics3D**. Como vamos plotar pontos usamos o comando **Point** dentro dele da seguinte forma

Graphics3D[Point[]]

Para aumentarmos o tamanho dos pontos na representação aplicamos o **PointSize**

Graphics3D[{PointSize[Large], Point[]}]

Podemos escolher as cores também. Vamos de **Magenta** nesse caso

Graphics3D[{PointSize[Large], Red, Point[]}]

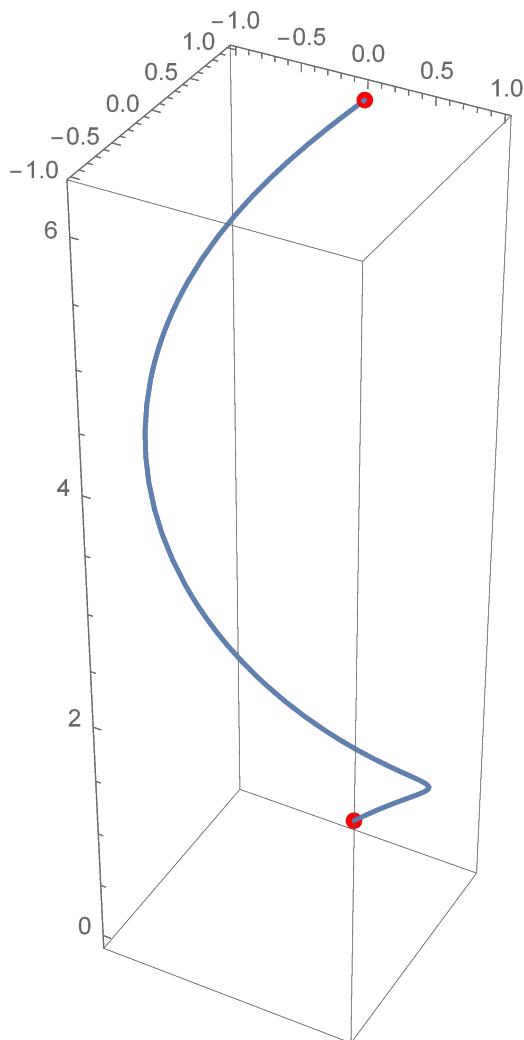
Agora, dentro do **Point** basta acrescentarmos uma lista de pontos com a estrutura

$\{p_1, p_2, \dots, p_n\}$

No nosso caso $p_1 = \{0, 1, 0\}$ e $p_2 = \{0, 1, 2\pi\}$, assim o **ParametricPlot3D** assumirá a forma

Graphics3D[{PointSize[Large], Point[\{\{0,1,0\}, \{0,1,2\pi\}\}]}]

Escrevendo as seguintes linhas de código, obtemos



```

1 a = ParametricPlot3D[{Sin[t], Cos[t], t}, {t, 0, 2Pi}]
2 b = Graphics3D[{PointSize[Large], Red, Point[{{0, 1, 0}, {0, 1, 2Pi}}]}]
3 Show[a, b]

```

Vamos acrescentar outro comando dentro do **Graphics** ainda não mostrado, o **Line**. A sintaxe é semelhante ao **Point**, nós passamos uma lista de pontos e a de forma automática é feita a interligação entre eles com base na ordem em que são escritos dentro da lista. Como só serão passados dois pontos (**p1** e **p2**), ao digitar

```

1 c = Graphics3D[{PointSize[Large], Red, Thick, Line[{{0, 1, 0}, {0, 1, 2Pi}}]}]

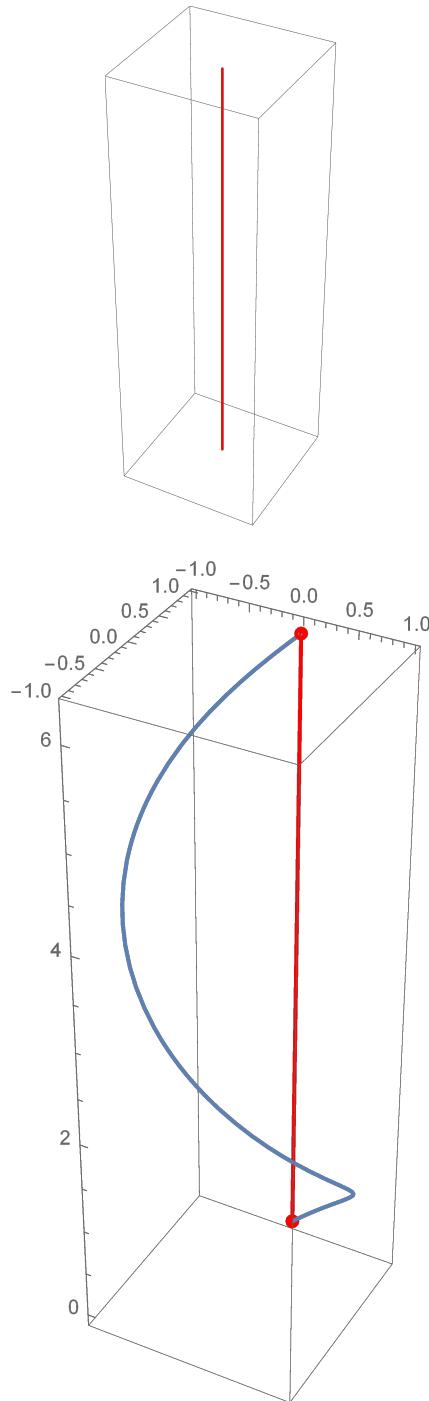
```

Juntando **a**, **b** e **c** no **Show**

```

1 Show[a, b, c]

```



4.1 Outro exemplo com o ParametricPlot3D

Vamos fazer um paraboloide análogo ao que foi feito com o `ContourPlot`. Basta estendermos a ideia do `ParametricPlot` para duas dimensões alterando a terceira posição da função (f_z).

Imagine que o paraboloide é uma série de círculos superpostos de modo que o raio e altura de cada um deles no empilhamento dependam de r e t . Nesse caso, t terá o mesmo papel visto anteriormente. Fará o ângulo interno variar de 0 a 2π , enquanto r elevará a altura de cada circunferência enquanto aumenta seu raio segundo r^2

Levando essa ideia para o `ParametricPlot3D`, podemos escrever

$$\begin{cases} x = r \sin t \\ y = r \cos t \\ z = r^2 \end{cases}$$

```
1 ParametricPlot3D[{r Sin[t], r Cos[t], r^2}, {t, 0, 2Pi}, {r, 0, 4}]
```

Podemos interpretar a linha acima como: “A partir da variação de t , renderize varias circunferências com o raio e altura variando segundo r (de 0 a 4 para o raio e de 0 a 16 para a altura)”