

## 2.3 Aplicações das Leis de Newton

### 2.3.1 Movimento tridimensional de um projétil (se despreza quaisquer efeitos do ar)

Nesta seção retomamos a análise do movimento de um projétil, visando a obter algumas características novas da trajetória.

Os resultados que já obtivemos serão, agora, generalizados para o espaço de três dimensões. (Algumas noções envolvendo o movimento de um projétil em duas dimensões já foram consideradas no capítulo anterior.)

O estudo do movimento de um projétil é uma das aplicações mais interessantes da Segunda lei de Newton: *A força resultante sobre um corpo é igual ao produto da massa do corpo pela aceleração do corpo.*

Após abandonar a boca do canhão, o projétil fica só em interação com a Terra, e a força  $\mathbf{F} = m\mathbf{g}$  é a única força que atua sobre o corpo. Então a Segunda lei de Newton (2.L.N.) reza:

$$m\mathbf{a} = m\mathbf{g} \quad (2.3-1)$$

As três componentes escalares desta equação vetorial têm a forma

$$m \, d^2x/dt^2 = 0$$

$$m \, d^2y/dt^2 = -mg \quad (2.3-2)$$

$$m \, d^2z/dt^2 = 0$$

O vetor da velocidade inicial é dado por  $\mathbf{v}_0 = v_0 \cos\alpha \cdot \mathbf{i} + v_0 \sin\alpha \cdot \mathbf{j} + 0 \cdot \mathbf{k}$ , onde  $v_{0x} = v_0 \cos\alpha$ ,  $v_{0y} = v_0 \sin\alpha$  e  $v_{0z} = 0$ . ( $\mathbf{j}$  é a direção vertical.)

As equações (2.3-2) podem ser integradas para se obter respectivamente

$$v_x = v_{0x}, \quad v_y = v_{0y} - gt, \quad v_z = v_{0z} = 0$$

Integrando novamente, obtemos as coordenadas do corpo em função do tempo:

$$x = v_{0x}t, y = v_{0y}t - gt^2/2, z = 0$$

onde as coordenadas iniciais do corpo são zero:  $x_0 = y_0 = z_0 = 0$ .

$x = v_{0x}t = t v_0 \cos\alpha$  é a componente escalar  $x$  do vetor posição  $\mathbf{r} = x \mathbf{i} + y \mathbf{j} + z \mathbf{k}$ .

Para melhorar nosso entendimento do *MuPAD*, façamos as integrações também com ajuda desse programa:

- `dglx:=ode({x''(t)=0,x(0)=0,x'(0)=v0*cos(a)},x(t)):`  
`x:=solve(dglx)[1]// componente escalar x do vetor posição`

$$t \cdot v_0 \cdot \cos(\alpha)$$

- `dgly:=ode({y''(t)=-g,y(0)=0,y'(0)=v0*sin(a)},y(t)):`  
`y:=solve(dgly)[1]//componente y do vetor posição`

$$t \cdot v_0 \cdot \sin(\alpha) - \frac{g \cdot t^2}{2}$$

- `dglz:=ode({z''(t)=0,z(0)=0,z'(0)=0},z(t)):`  
`z:=solve(dglz)[1]//componente z do vetor posição`

0

- `dglvy:=ode({vy'(t)=-g,vy(0)=v0*sin(a)},vy(t)):`  
`vy:=solve(dglvy)[1]// componente escalar y do vetor da velocidade`

$$v_0 \cdot \sin(\alpha) - g \cdot t$$

Esses resultados mostram que o corpo sempre se mantém no plano-xy, ou seja, a sua trajetória é plana. (O plano-xy coincide com o plano definido por  $\mathbf{v}_0$  e  $\mathbf{a} = \mathbf{g}$ .) O resultado  $v_x = v_{0x} = v_0 \cos\alpha$  mostra que a componente da velocidade na direção  $x$  se mantém invariante no tempo, o que é consistente com o que já vimos no capítulo anterior. No entanto, a componente  $v_y$  da velocidade decresce continuamente até atingir o valor nulo, quando o corpo atinge o ponto mais alto de sua trajetória.

O gráfico animado do Programa 4 no capítulo 2.2.4 nós havia demonstrado isso com clareza considerável. O tempo necessário para o projétil alcançar o ponto mais alto é obtido pela solução da equação  $v_y = 0$ :

- `assume (g<>0) :`
- `vy:=v0*sin(a)-g*t:`
- `tsub:=solve(vy=0,t)`

$$\left\{ \frac{v_0 \cdot \sin(\alpha)}{g} \right\}$$

- `ts:=tsub[1]//tempo de subida`

$$\frac{v_0 \cdot \sin(\alpha)}{g}$$

Depois de atingir o ponto mais alto,  $v_y$  se torna negativa com módulo crescente. A equação da trajetória do projétil é obtida por eliminação do tempo nas duas equações  $x = v_{0x} \cdot t = t \cdot v_0 \cdot \cos \alpha$  e  $y = -g \cdot t^2/2 + v_0 \cdot t \cdot \sin(\alpha)$ .

Primeiramente eliminamos  $t$  da equação para  $x$ , e seguidamente substituímos o resultado na equação para  $y$ :

- `tx:=solve(x=t*v0*cos(a),t):`
- `y:=-g*t^2/2+t*v0*sin(a):`
- `subs(y,t=tx[1]):`
- `subs(%,sin(a)=cos(a)*tan(a))`

$$\left\{ \frac{x \cdot (\cos(\alpha) \cdot \tan(\alpha))}{\cos(\alpha)} - \frac{g \cdot x^2}{2 \cdot v_0^2 \cdot \cos(\alpha)^2} \text{ if } -v_0 \cdot \cos(\alpha) \neq 0 \right.$$

- `expand(%)`

$$\left\{ x \cdot \tan(\alpha) - \frac{g \cdot x^2}{2 \cdot v_0^2 \cdot \cos(\alpha)^2} \text{ if } -v_0 \cdot \cos(\alpha) \neq 0 \right.$$

Então, obtivemos a equação de uma parábola:  $y = x \cdot \operatorname{tg} \alpha - g \cdot x^2 / (2 \cdot v_0^2 \cdot \cos^2 \alpha)$ .  
(Em *MuPAD* não é possível utilizar  $\cos(a)$  em `assume`, confira `?trig`)

O tempo gasto para o projétil atingir novamente o nível  $y = 0$ , do qual foi atirado, obtemos da equação  $y(t) = 0$ . Resultado:  $t_{v\hat{o}o} = 2 \cdot t_s = 2 \cdot v_0 \cdot \operatorname{sen} \alpha / g$ .

Com este tempo obtemos para o alcance horizontal a expressão  $R = t_{v\hat{o}o} \cdot v_0 \cdot \cos \alpha$ , o seja:

$$R = 2 \cdot v_0 \cdot \operatorname{sen} \alpha / g \cdot v_0 \cdot \cos \alpha = v_0^2 \cdot \operatorname{sen}(2\alpha) / g$$

Observe que  $R$  possui seu valor *máximo* quando  $\operatorname{sen}(2\alpha) = 1$ , o que corresponde a  $2\alpha = 90^\circ$  ou  $\alpha = 45^\circ$ . Para a altura máxima do percurso obtemos  $y_{\max} = v_0^2 \operatorname{sen}^2(\alpha) / 2g$ .

Agora, se o operador do canhão não pretende alcançar a distância máxima de  $R$ , o seja  $R_{\max} = v_0^2 / g$ , ele pode acertar o alvo no ponto  $(x < R_{\max}, 0)$  com dois ângulos de tiro. (A função trigonométrica inversa "arc sen" sempre possui duas soluções possíveis.) Esses dois ângulos possíveis têm sempre uma soma de  $90^\circ$ . Eles são igualmente afastados de  $45^\circ$ . Por exemplo, um alvo na distância de 800 m pode ser acertado com um dos dois ângulos que satisfazem a equação  $R = v_0^2 \operatorname{sen}(2\alpha) / g = 800$ .

- `reset()` :

```
g:=9.8:v0:=100:
```

```
eq:=v0*v0*sin(x)-g*800://x=2 alpha; eq=0
```

```
a1:=numeric::fsolve(eq,x=1..PI);
```

```
op(a1);
```

```
eval(subs(eq,a1))//deve dar zero
```

```
[x = 2.240509096]
```

```
x = 2.240509096
```

```
-4.884981308e-14
```

A função `numeric::fsolve` determina a solução de uma equação dentro de um certo intervalo `a..b`. Com `eval(subs(eq,a1))` podemos controlar a exatidão da solução.

A solução  $x_1 = 2.24051$  rad corresponde a  $2\alpha_1 = 128.37^\circ$ , o seja,  $\alpha_1 = 64,19^\circ$ .

Se utilizarmos o intervalo  $0 \dots \text{PI}$ , então a solução será  $x = 0.9010835578$ , o que nos daria um ângulo de  $\alpha_2 = 25,81^\circ$ .

Nota-se que a soma dos dois ângulos é de  $90^\circ$ , veja a seção 2.3.2.

No seguinte gráfico traçamos as trajetórias para os ângulos ( $30^\circ$ ,  $60^\circ$ ) e  $45^\circ$ . As trajetórias com  $30^\circ$  e  $60^\circ$  cortam-se no mesmo ponto sobre o eixo  $x$ , a trajetória de  $45^\circ$  é a trajetória com o alcance máximo.

Se há traçada também a parábola-limite com a equação  $y = v_0^2/2g - gx^2/2v_0^2$ , veja mais adiante, na seção 2.3.2.

Todas as trajetórias possíveis encontram-se debaixo desta parábola-limite. Observe o método elegante que o *MuPAD* nos oferece para traçar várias curvas na mesma figura! ( $R = 565,57$  m;  $R_{\text{max}} = 653,06$  m).

- `reset () :`
- `v0:=80:`
- `g:=9.8:`
- `ya:=x*tan(a)-x*x*g/(2*v0*v0*cos(a)^2):`
- `k:=PI/180:`
- `alpha:=[30*k,60*k,45*k]:`
- `y1:=subs(ya,a=alpha[i])$ i=1..3:`
- `y2:=v0*v0/(2*g)-g*x*x/(2*v0*v0):`
- `plotfunc2d(y1,y2, x=0..650)`

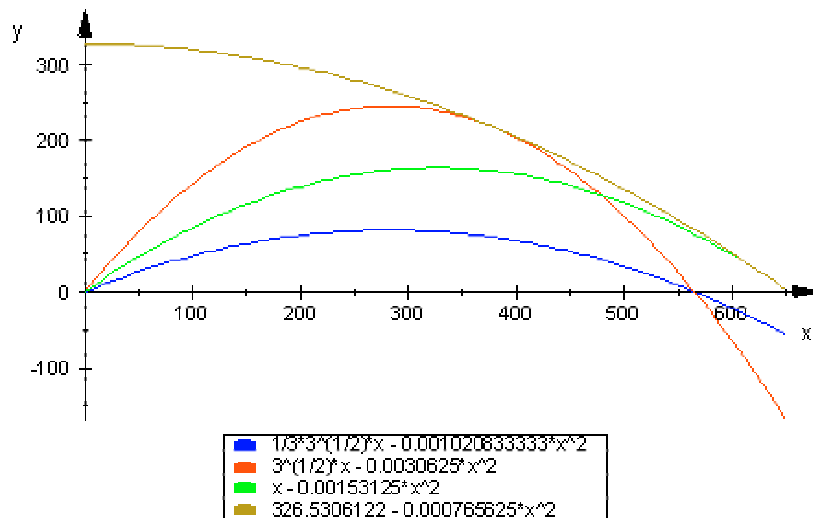


Fig. 2.3-1

Quando o alvo se encontrar num ponto  $(x_1, y_1 > 0)$ , a soma dos ângulos será  $\alpha_1 + \alpha_2 = \pi/2 + \beta$ , onde o ângulo  $\beta$  é a solução da equação  $\text{tg}\beta = y_1/x_1$ . Veja um pouco mais adiante, na seção 2.3.2.

Os ângulos  $\alpha_1$  e  $\alpha_2$  calculamos usando a seguinte (ver 2.3.2):

$$\tan(\alpha) = \frac{v_0^2}{gx_1} \pm \frac{1}{x_1} \sqrt{\frac{v_0^4}{g^2} - \frac{2v_0^2 y_1}{g} - x_1^2} \quad (2.3-3)$$

### Exemplo:

O alvo tem as coordenadas  $x_1 = 300\text{m}$ ,  $y_1 = 100\text{m}$ . O canhão atira a bala com  $v_0 = 80\text{ m/s}$ .  $g = 9.8\text{ m/s}^2$ .

Substituindo estes valores na equação (2.3-3), obtemos  $\text{tg}(\alpha_1) = 3.689325$  e  $\text{tg}(\alpha_2) = 0.664416$ .

Os ângulos são  $\alpha_1 = \text{arc tg}(\alpha_1) = 1,3061\text{ rad} = 74,8342^\circ$  e  $\alpha_2 = 0,5864\text{ rad} = 33,600^\circ$ .

A soma dos ângulos é  $1,8925\text{ rad}$  e  $\pi/2 + \beta = 1,8925$ , o seja:  $\alpha_1 + \alpha_2 = \pi/2 + \beta$ , como havíamos dito mais acima.

Aqui você tem tudo isto demonstrado por meio de *MuPAD*:

```

• reset():
v0:=80:
g:=9.8:X:=300:Y:=100:
ya:=x*tan(a)-x*x*g/(2*v0*v0*cos(a)^2):
k:=PI/180:
b:= arctan(Y/X):
alpha:=[74.834*k,PI/2+b-74.834*k]:
y1:=subs(ya,a=alpha[i])$ i=1..2:
y2:=v0*v0/(2*g)-g*x*x/(2*v0*v0):
plotfunc2d(y1,y2, x=0..350)

```

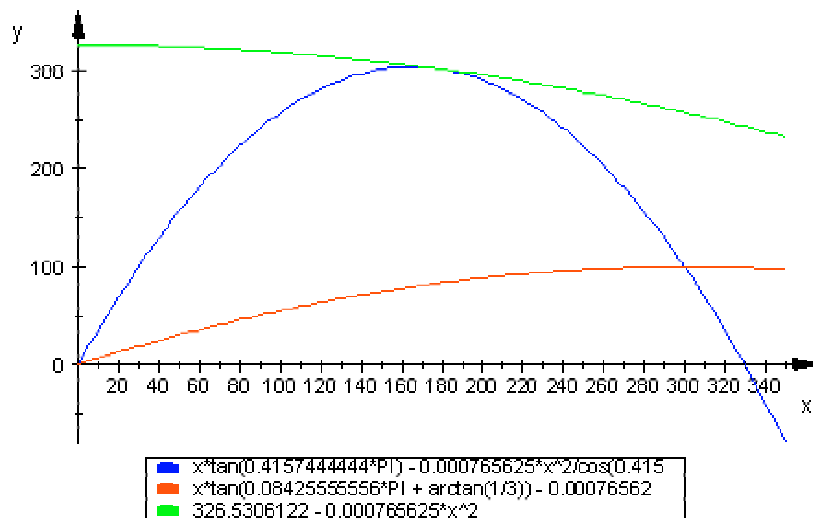


Fig. 2.3-2

### 2.3.2 Com lápis e papel

Da equação  $y = x \cdot \operatorname{tg} \alpha - g \cdot x^2 / (2 \cdot v_0^2 \cos^2 \alpha)$  podemos obter a equação da parábola-limite  $y = v_0^2 / 2g - gx^2 / 2v_0^2$ , isto é, a equação da curva que separa as regiões acessíveis das regiões não acessíveis para o projétil. Para todo valor de  $x$ , todos os pontos que estão acima de  $y = v_0^2 / 2g - gx^2 / 2v_0^2$  são inalcançáveis para a bala tirada do canhão. Para demonstrar a validade desta equação, substituímos na equação  $y = x \cdot \operatorname{tg} \alpha - g \cdot x^2 / (2 \cdot v_0^2 \cos^2 \alpha)$  o termo  $\cos^2 \alpha$  por o termo equivalente  $1 / (1 + \operatorname{tg}^2 \alpha)$ .

Assim fazendo, obtemos a seguinte equação de segundo grau para  $\operatorname{tg}(\alpha)$ :

$$\operatorname{tg}^2(\alpha) - \frac{2v_0^2}{gx_1} \operatorname{tg}(\alpha) + \frac{2v_0^2 y_1}{gx_1^2} + 1 = 0 \quad (2.3-4)$$

A solução desta equação é a equação (2.3-3).

Agora, quando o discriminante na raiz quadrada da equação (2.3-3) for negativo, não existem raízes reais e  $\operatorname{tg}(\alpha)$  é imaginária. Isto quer dizer que não existem ângulos de tiro com  $v_0^4 / g^2 - x_1^2 - 2v_0^2 y_1 / g < 0$ . Isso tem uma interpretação física muito conveniente: Os pontos da região acima da curva com  $y = v_0^2 / 2g - gx^2 / 2v_0^2$  são inalcançáveis - como acabamos de dizer.

(Da equação  $v_0^4 / g^2 - x^2 - 2v_0^2 y / g = 0$  podemos extrair  $y$  para obter a equação da curva-limite:  $y = v_0^2 / 2g - gx^2 / 2v_0^2$ )

Falta a demonstração da relação  $\alpha_1 + \alpha_2 = \pi / 2 + \beta$ .

Vamos utilizar o teorema de *Vieta* (relações de Girard) sobre as raízes de uma equação de segundo grau na forma  $x^2 + px + q = 0$ :

1. O produto das raízes  $x_1, x_2$  é igual a  $q$
2. A soma das raízes é igual a  $-p$

(Essas relações entre raízes e coeficientes podem ser generalizadas para equações de qualquer grau.)



Dáí temos

$$\operatorname{tg}(\alpha_1) \cdot \operatorname{tg}(\alpha_2) = 1 + 2v_0^2 y_1 / g x_1^2$$

$$\operatorname{tg}(\alpha_1) + \operatorname{tg}(\alpha_2) = 2v_0^2 / g x_1$$

Vamos introduzir agora a relação geral:

$$\operatorname{tg}(\alpha_1 + \alpha_2) = \frac{\operatorname{tg}\alpha_1 + \operatorname{tg}\alpha_2}{1 - \operatorname{tg}\alpha_1 \cdot \operatorname{tg}\alpha_2}$$

para obter  $\operatorname{tg}(\alpha_1 + \alpha_2) = -x_1/y_1 = -\operatorname{cotg}(\beta) = \operatorname{tg}(\pi/2 + \beta)$ , o seja:  $\alpha_1 + \alpha_2 = \pi/2 + \beta$ .

( $\beta$  é o ângulo de inclinação da reta que passa por o origem (0,0) e por o ponto  $(x_1, y_1)$ , o que quer dizer que o coeficiente angular é  $m = \operatorname{tg}(\beta) = y_1/x_1$ .)

### 2.3.3 Mais informações sobre gráficos

Se você não quer que o *MuPAD* trace valores negativos, como na figura 2.3-2, então pode usar uma "procedure" (um 'procedimento', veja 2.2, Programa 5) onde se exclui os pontos debaixo do eixo x mediante uma estrutura condicional:

```
if <condição> then <comandoVerdade> else <comandoFalso>
```

ou em português:

```
se <condição> então <comandoVerdade> senão <comandoFalso>
```

O programa em linguagem de MuPAD tem a seguinte aparência:

```
• reset() :  
  f := proc(x) // procedimento  
  begin  
    v0:=80:  
    a:=PI/6: //0.5236 (30°)  
    y:=x*tan(a)-x^2*9.8/(2*v0^2*cos(a)^2):  
    if float(y)< 0 then  
      hold  
    else  
      y  
    end_if:  
  end_proc:
```

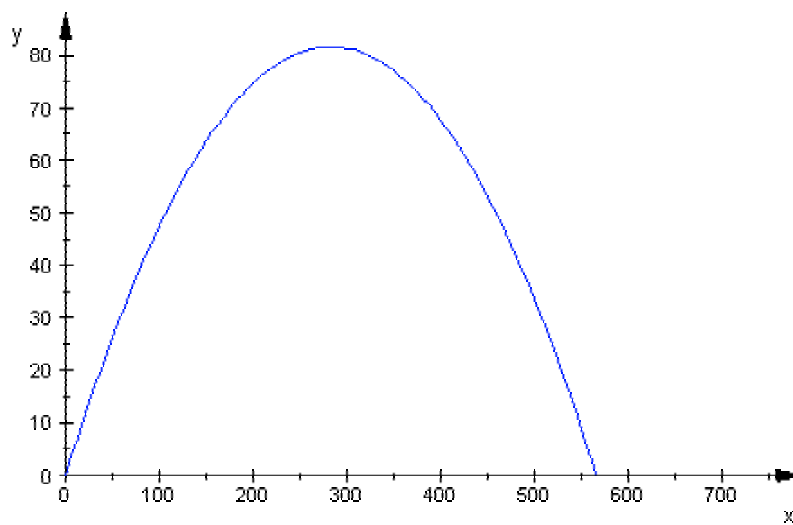


Fig. 2.3-3

Para fazer uma comparação, o MuPAD pede que um número seja do tipo ponto flutuante. Na linha `if float(y) < 0 then` temos a função `float` que transforma o objeto `y` num número em ponto flutuante (se a variável `a` for um número flutuante, por exemplo `a := 0.5236`, poderíamos fazer a comparação sem utilizar a função `float`, escreveríamos simplesmente `if y < 0 then`).

Tecnicamente, o símbolo `<` (menor que) é um operador lógico que estamos utilizando para expressar uma condição: **se `y` é menor do que 0, então** a linha `plot(plot::Function2d(f, x = 0 .. 750))` não será executada (`hold`), **senão** (else) a condição é falsa e o 'procedimento' `f` desenhará o ponto `(y,x)`.

Com o seguinte programa pode-se fazer um **gráfico animado em três dimensões** de uma bala lançada desde o ponto `(0,0,0)`. (Dê um duplo clique com o botão esquerdo do mouse no referencial tridimensional que o MuPAD produz e exibe na tela. O gráfico se deixa também girar: clique com o botão esquerdo do mouse nele e mova-o. Observe os muitos botões na barra superior e clique neles. No lado inferior, na direita, você pode conferir as posições da câmara virtual "instalada" no programa de animação: Camera Position X, Y, Z.

A função gráfica `plot::Curve3d([x(t), y(t), z(t)], t=tmin..tmax)`

permite visualizar curvas parametrizadas num sistema de referência espacial.

As equações paramétricas da parábola de tiro no espaço são

$$\begin{aligned}x &= t v_0 \cos\alpha \cos\beta \\y &= t v_0 \cos\alpha \sin\beta \\z &= t v_0 \sin\alpha - gt^2/2\end{aligned}\quad (2.3-5)$$

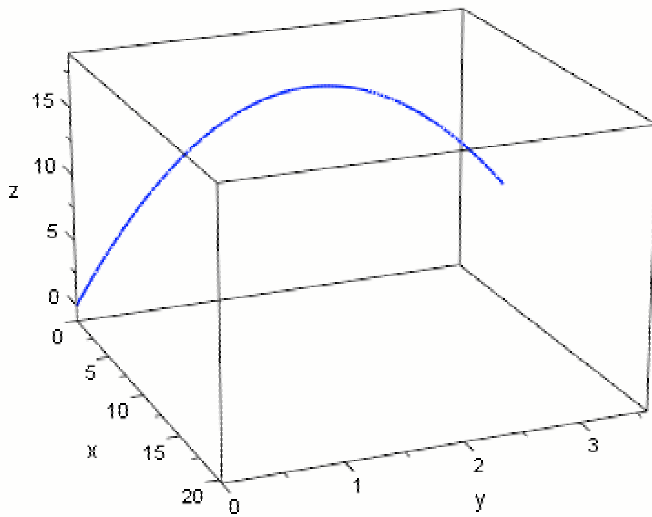


Fig. 2.3-4

```

• reset() :
a:=75*PI/180:
b:=10*PI/180:
g:=9.81:
v0:=20:
x:=t*v0*cos(a)*cos(b):
y:=t*v0*cos(a)*sin(b):
z:=t*v0*sin(a)-g*t*t/2:
curva:=plot::Curve3d([x(t),y(t),z(t)],t=0..u,u=0..4):
plot(curva)

```

Para observar a influência do valor de  $b$  sobre a trajetória, repetimos o gráfico com dois valores para  $b$ . Para  $b = 0$  obtemos um movimento no plano- $xz$ .

```

• reset() :
  a:=75*PI/180:
  b:=20*PI/180:
  b1:=30*PI/180:
  g:=9.81:
  v0:=20:
  x0:=t*v0*cos(a)*cos(b):
  y0:=t*v0*cos(a)*sin(b):
  z0:=t*v0*sin(a)-g*t*t/2:
  x:=t*v0*cos(a)*cos(b1):
  y:=t*v0*cos(a)*sin(b1):
  z:=t*v0*sin(a)-g*t*t/2:
  curva0:=plot::Curve3d([x0(t),y0(t),z0(t)],t=0..4,
  Color=RGB::Red):
  curva1:=plot::Curve3d([x(t),y(t),z(t)],t=0..4):
  plot(curva0,curva1)

```

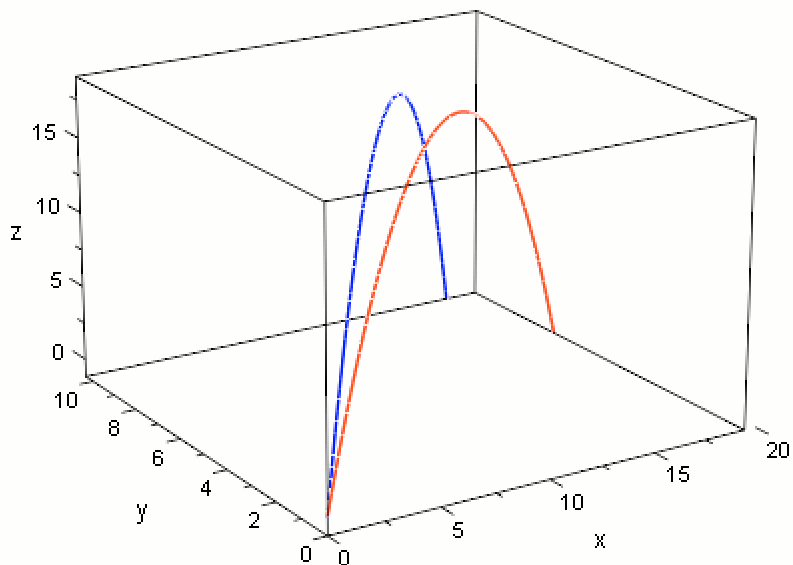


Fig. 2.3-5

Da forma análoga, podemos variar o valor do ângulo  $a$  para observar o seu efeito sobre a forma da trajetória.

```
a:=60*PI/180:
```

```
b:=20*PI/180:
```

```
a1:=30*PI/180:// a + a1 = 90 !
```

```
g:=9.81:
```

```
v0:=50:
```

```
x0:=t*v0*cos(a)*cos(b):
```

```
y0:=t*v0*cos(a)*sin(b):
```

```
z0:=t*v0*sin(a)-g*t*t/2:
```

```
x:=t*v0*cos(a1)*cos(b):
```

```
y:=t*v0*cos(a1)*sin(b):
```

```
z:=t*v0*sin(a1)-g*t*t/2:
```

```

curva0:=plot::Curve3d([x0(t),y0(t),z0(t)],t=0..8.8,
Color=RGB::Red):
curva1:=plot::Curve3d([x(t),y(t),z(t)],t=0..5):
plot(curva0,curva1)

```

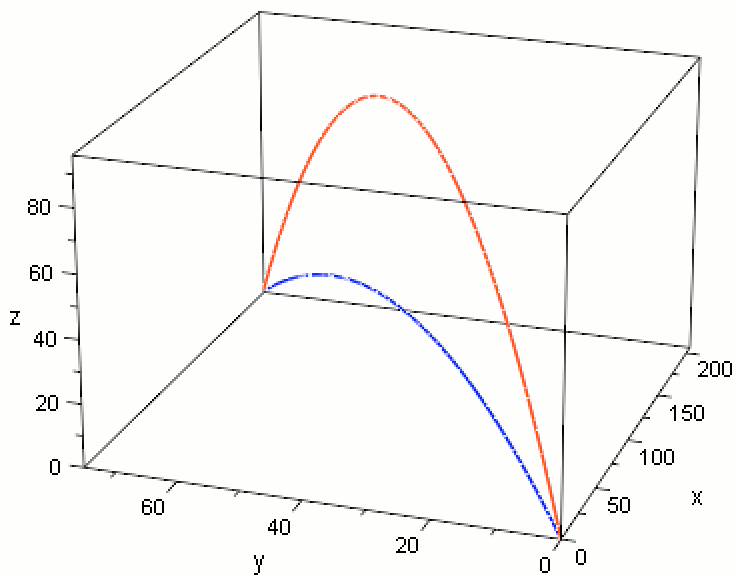


Fig. 2.3-6

### 2.3.4 Curvas e superfícies parametrizadas

Nos seguintes problemas vamos falar sobre a parametrização de curvas e superfícies e da representação delas utilizando MuPAD.

1. Descreva o movimento de uma partícula cujas coordenadas no instante  $t$  são dadas pelas equações  $x = \cos t$ ,  $y = \sin t$ .
2. Dê uma parametrização da elipse  $4x^2 + 9y^2 = 1$ .

3. Descreva em palavras o movimento dado parametricamente por  $x = \cos t$ ,  $y = \sin t$ ,  $z = t$  e trace o gráfico.
4. Ache equações paramétricas para a reta pelo ponto (1,3,5) e paralela ao vetor  $3\mathbf{i} + 5\mathbf{j} + 6\mathbf{k}$ .

O parâmetro, que usualmente denotamos por  $t$ , possa não ter significado físico. É útil de qualquer forma pensar nele como sendo o tempo.

### Soluções:

1. Como  $(\cos t)^2 + (\sin t)^2 = 1$  temos  $x^2 + y^2 = 1$ .  
Isto é, a qualquer tempo  $t$  a partícula está num ponto  $(x,y)$  em algum lugar no círculo  $x^2 + y^2 = 1$ .
2. Como  $(2x)^2 + (3y)^2 = 1$  ajustamos a parametrização do círculo no problema 1. Substituindo  $x$  por  $2x$  e  $y$  por  $3y$  dá as equações  $3x = \cos t$ ,  $3y = \sin t$ . Assim uma parametrização da elipse é  $x = \cos t/3$ ,  $y = \sin t/3$ ,  $0 \leq t \leq 2\pi$
3. As coordenadas- $x$  e  $y$  da partícula são as mesmas que no problema 1, que dá movimento circular no plano- $xy$ , ao passo que a coordenada- $z$  cresce sempre. Assim, a partícula traça uma espiral ascendente, como uma mola. Esta curva chama-se uma *hélice*.

- `reset()` :

`x:=cos(t) :`

`y:=sin(t) :`

`z:=t :`

`curva:=plot::Curve3d([x(t),y(t),z(t)],t=0..u,u=0..5*PI) :`

`plot(curva)`



Uma partícula eletricamente carregada sob o efeito de um campo magnético uniforme percorre uma trajetória helicoidal com velocidade de módulo constante. A partícula combina um movimento circular uniforme no plano-xy com uma velocidade também uniforme na direção z. A posição da partícula varia com o tempo na forma

$$\mathbf{r}(t) = r \cos \omega t \mathbf{i} + r \sin \omega t \mathbf{j} + v_z t \mathbf{k},$$

Onde  $r$ ,  $\omega$  e  $v_z$  são constantes positivas.

*MuPAD* determina os seguintes expressões para velocidade e aceleração em função do tempo:

- $x := t \rightarrow r \cdot \cos(\omega \cdot t) :$
- $y := t \rightarrow r \cdot \sin(\omega \cdot t) :$
- $z := t \rightarrow v_z \cdot t :$
- $pos := t \rightarrow (x(t), y(t), z(t)) :$
- $v := t \rightarrow (x'(t), y'(t), z'(t)) :$
- $a := t \rightarrow (x''(t), y''(t), z''(t)) :$
- $vx := v(t) [1] ;$
- $vy := v(t) [2] ;$
- $vz := v(t) [3] ;$
- $ax := a(t) [1] ;$
- $ay := a(t) [2] ;$
- $az := a(t) [3] :$

$$-r \cdot \omega \cdot \sin(\omega \cdot t)$$

$$r \cdot \omega \cdot \cos(\omega \cdot t)$$

$$v_z$$

$$-r \cdot \omega^2 \cdot \cos(t \cdot \omega)$$

$$-r \cdot \omega^2 \cdot \sin(t \cdot \omega)$$

A figura 2.3-7 mostra a hélice:

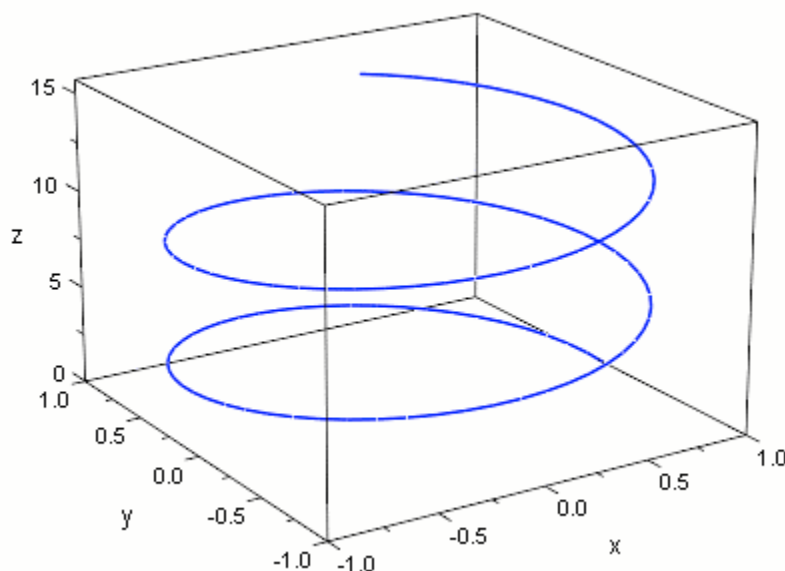


Fig. 2.3-7

1. Imaginemos uma partícula no ponto (1,3,5) no tempo  $t = 0$  e que se move por um deslocamento  $3\mathbf{i} + 5\mathbf{j} + 6\mathbf{k}$  a cada unidade de tempo. Quando  $t = 0$ ,  $x = 1$  e  $x$  cresce por 3 unidades de tempo. Assim no tempo  $t$  a coordenada- $x$  da partícula é dada por  $x = 1 + 3t$ . Analogamente, a coordenada- $y$  começa em  $y = 3$  e cresce numa taxa de 5 unidades para cada unidade de tempo. A coordenada- $z$  começa em  $z = 5$  e cresce por 6 unidades a cada unidade de tempo. Assim as equações paramétricas da reta são  $x = 1+3t$ ,  $y = 3+5t$ ,  $z = 5+6t$ . Observe que a parametrização de uma reta dada acima expressa as coordenadas  $x$ ,  $y$ ,  $z$  como funções lineares do parâmetro  $t$ .

`reset() :`

`x:=1+3*t:`

`y:=3+5*t:`

```

z:=5+6*t:

reta:=plot::Curve3d([x(t),y(t),z(t)],t=0..u,u=0..2*
PI):

plot(reta)

```

## Comentário:

Acima parametrizamos um círculo no 2-espaco usando as equações

$x = \cos t$ ,  $y = \sin t$ .

No 3-espaco, o mesmo círculo no plano-xy tem equações paramétricas

$x = \cos t$ ,  $y = \sin t$ ,  $z = 0$ .

Acrescentamos a equação  $z = 0$  para especificar que o círculo está no plano-xy.

Se quiséssemos um círculo no plano  $z = 3$  usaríamos as equações

$x = \cos t$ ,  $y = \sin t$ ,  $z = 3$

Suponha agora que deixamos  $z$  variar livremente, bem como  $t$ . Obtemos círculos em cada plano horizontal, formando um *cilindro*.

Assim, precisamos de dois parâmetros,  $u$  e  $v$ , para parametrizar o cilindro.

O MuPAD possui a função `plot::Surface` com a qual podemos traçar facilmente a superfície dum cilindro:

- `reset()` :
 

```

x:=2*cos(u):: r = 2
y:=2*sin(u):
z:=v:

s := plot::Surface([x, y, z], u = 0 .. 2*PI, v = -2
.. 2):

plot(s, Scaling= Constrained)

```

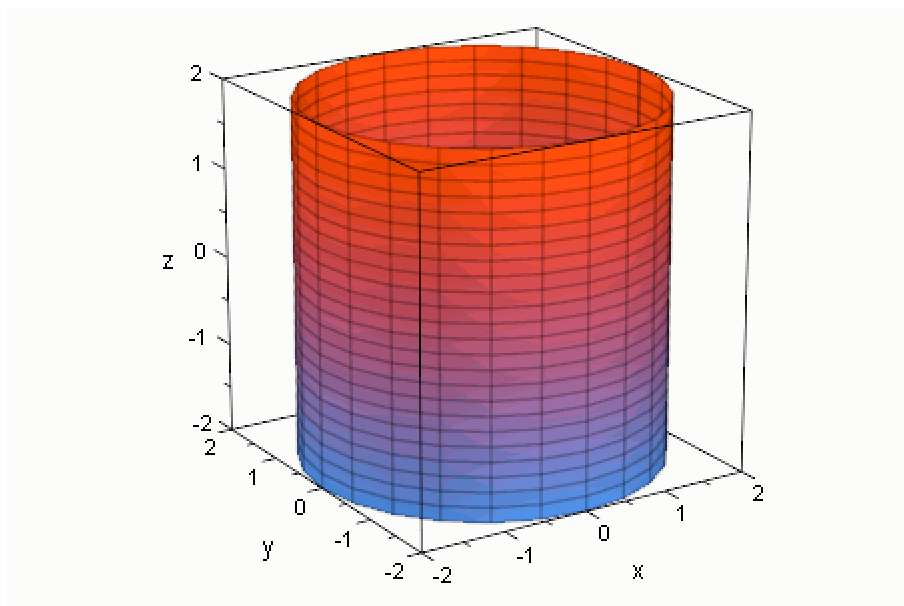


Fig. 2.3-8

A representação de uma esfera de raio 1 vemos na seguinte figura 2.3-9.

A esfera de raio 1 e centro na origem pode ser representada implicitamente pela equação  $x^2 + y^2 + z^2 = 1$  ou explicitamente pelas equações

$$z = \sqrt{1 - x^2 - y^2}$$

$$z = -\sqrt{1 - x^2 - y^2}$$

ou parametricamente por

$$x = \text{sen}\alpha \cos\beta ,$$

$$y = \text{sen}\alpha \text{sen}\beta ,$$

$$z = \text{cos}\alpha$$

$$0 \leq \alpha \leq \pi$$

$$0 \leq \beta \leq 2\pi$$

```
• reset():  
x:=cos(u)*sin(v):  
y:=sin(u)*sin(v):  
z:=cos(v):  
esfera := plot::Surface([x, y, z], u = 0 .. 2*PI, v  
= 0 .. PI):  
plot(esfera, Scaling= Constrained)
```

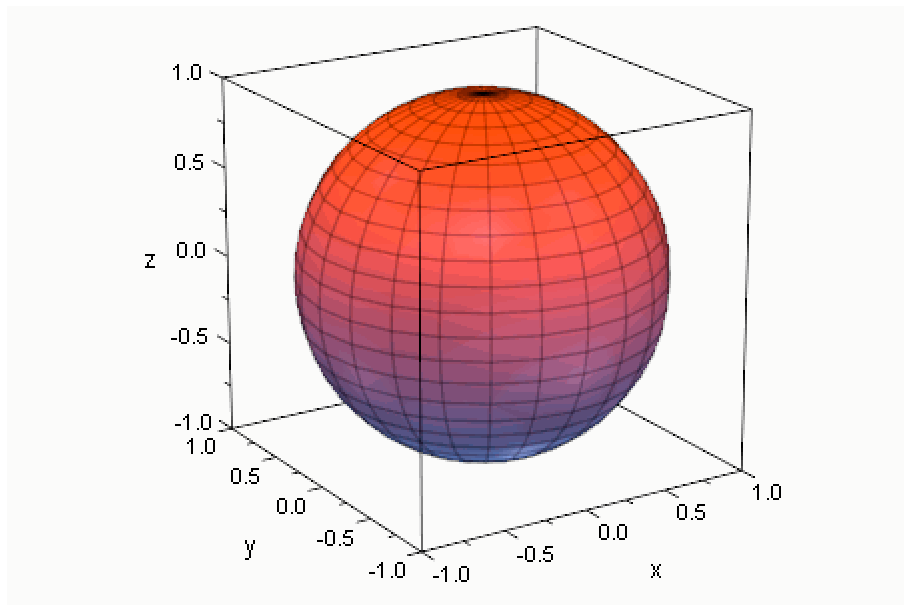


Fig. 2.3-9