Interpretação geométrica de transformações lineares

Um dos tópicos mais importantes em Álgebra Linear é a álgebra matricial. Ela aparece como representações de operadores lineares e de transformações lineares entre espaços vetoriais de dimensão finita, frequente nas aplicações. A versatilidade do Maple V é muito útil não somente para efetuar as operações matriciais, mas também para permitir uma interpretação geométrica dos efeitos dos movimentos mais importantes do plano e do espaço, através de seus recursos gráficos. Em geral, num primeiro curso de Álgebra Linear as propriedades geométricas de transformações lineares não são suficientemente compreendidas, e a utilização de Maple V pode auxiliar neste sentido. Os recursos de animação são úteis para explorar os movimentos e perceber o caráter isométrico de alguns dos mais importantes movimentos do plano e do espaço.

```
> with(linalg) : with(plots) : with(geometry) :
```

Exemplos de movimentos através da álgebra matricial

Um estudo completo de movimentos do plano e do espaço, assim como da álgebra matricial de transformações lineares, utilizando os comandos do pacote linalg de Álgebra Linear, pode ser encontrada em [B-P]. Nesta apresentação, vamos ilustrar apenas alguns exemplos.

Reflexão no plano, segundo o eixo y

É a transformação dada por :

Ry : R^2 - R^2

Ry (x , y) = (-x , y)

Exemplo: Ry (1 , 1) = (-1 , 1).

Este operador tem a sua representação matricial dada por:

```
> Ry := matrix([[-1, 0], [0, 1]]);
```

$$Ry := \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \quad \textbf{(1)}$$

Como exemplo, seja v o vetor sobre o qual será aplicada a transformação linear Ry.

```
> v := vector([1, 1]);
```

$$v := \begin{bmatrix} 1 & 1 \end{bmatrix} \quad \textbf{(2)}$$

Se q é o vetor Ry (v), o comando evalm(Ry &* v) efetua a multiplicação matricial de Ry por vetor coluna v:

```
> q := evalm(Ry &* v);
```

$$q := evalm\ (Ry\ \&*\ v) \qquad \textbf{(3)}$$

Vamos considerar P e Q , pontos do plano, tal que OP e OQ representem os vetores v e q :

```
> Q := convert(q, 'list');
```

$$Q := [evalm, Ry\ \&*\ v] \qquad \textbf{(4)}$$

```
> P := convert(v, 'list');
```

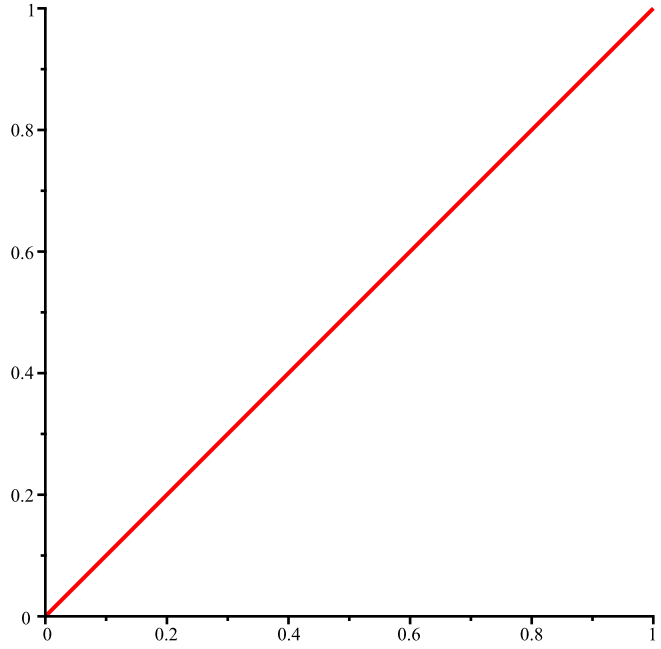
$$P := [1, 1] \qquad \textbf{(5)}$$

O comando abaixo mostra o vetor antes da aplicação de Ry e após a aplicação de Ry.

. v antes de T.....: cor vermelha

. q = Ry(v) depois de Ry ...: cor azul

```
> OP := plot([[0, 0], [v[1], v[2]]], color = red) :
  OQ := plot([[0, 0], [q[1], q[2]]], color = blue) :
  display({OP, OQ}, scaling = constrained);
```



Projeção Segundo uma Reta qualquer passando pela Origem

Exemplo:

Consideremos a reta y = 3x . Um vetor direção desta reta pode ser dado por (1, 3), que é invariante por projeção sobre a reta dada. Claramente, o vetor (-3 , 1) é ortogonal ao vetor (1, 3), constitui uma base de R^2 juntamente com (1 , 3) e a projeção sobre a reta dada leva ao vetor nulo .

Temos duas bases a considerar:

Base canônica : { (1 , 0) , (0 , 1) }

Base B : { (1 , 3) , (-3 , 1) }

B C B'

A projeção terá como matriz de transformação : I] * T] * I] , onde estão envolvidas as matrizes mudança de base.

```
> restart; with(linalg) : with(plots) :
  r := 3 * x;
  a := plot( {r}, x =-1 ..1, -1 ..1, colour = RED, linestyle = 2 ) :
```

$$r := 3\,x \tag{6}$$

A matriz T representa a projeção, relativa à base B. Observamos que ela mantém inva riante o vetor (1 , 3) e anula o vetor (-3 , 1) ortogonal a ele.

```
> T := matrix( [ [1, 0], [0, 0] ]);
```

$$T := \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \tag{7}$$

mudanca_base2 : matriz de mudança da base B para a Canônica .

```
> mudanca_base2 := matrix( [ [1, -3], [3, 1] ]);
```

$$mudanca\_base2 := \begin{bmatrix} 1 & -3 \\ 3 & 1 \end{bmatrix} \tag{8}$$

mudanca_base1 : matriz de mudança da base Canônica para a B .

```
> mudanca_base1 := inverse(mudanca_base2);
```

$$mudanca\_base1 := \begin{bmatrix} \frac{1}{10} & \frac{3}{10} \\ -\frac{3}{10} & \frac{1}{10} \end{bmatrix} \tag{9}$$

Então, a matriz procurada é dada por :

```
> matriz_transf := evalm(mudanca_base2 &* T &* mudanca_base1);
```

$$matriz\_transf := \begin{bmatrix} \frac{1}{10} & \frac{3}{10} \\ \frac{3}{10} & \frac{9}{10} \end{bmatrix} \tag{10}$$

Por exemplo, consideremos o vetor (1 , 1) . Aplicando a matriz acima obtida, teremos o vetor v , que é a projeção de (1 , 1) sobre a reta dada .

```
> v1 := vector([1, 1]);
```

$$v1 := \begin{bmatrix} 1 & 1 \end{bmatrix} \quad \textbf{(11)}$$

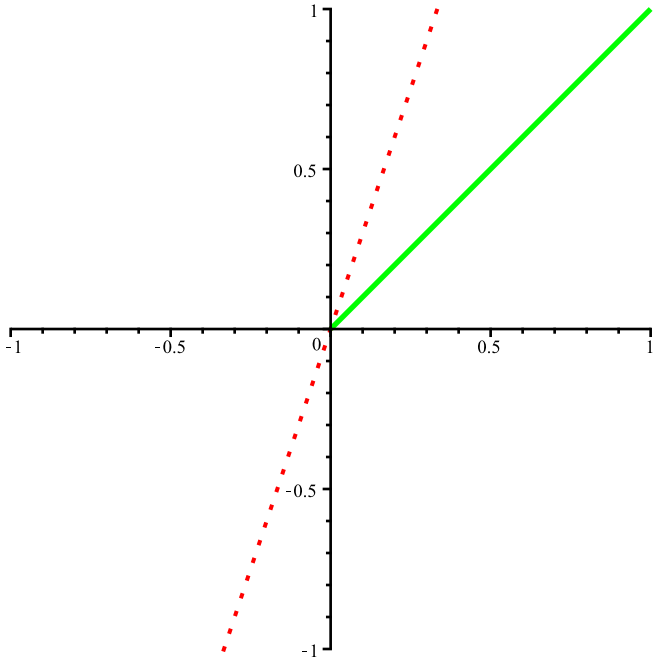
```
> v1 := vector([1, 1]);
```

$$v1 := \begin{bmatrix} 1 & 1 \end{bmatrix} \quad \textbf{(12)}$$

Podemos visualizar graficamente o exemplo, através de :

```
> b := plot([[[0, 0], [v1[1], v1[2]]], [[0, 0], [v2[1], v2[2]]]], color = [green, blue], scaling
      = constrained, thickness = 2) :
> display([b, a]);
```

Warning, data could not be converted to float Matrix



Vetor (1 , 1) ..... representado pela cor verde .

Vetor(2/5 , 6/5) ..... representado pela cor azul .

Reta y = 3x ..... representada pela cor vermelha .

Podemos obter a imagem de qualquer vetor do plano pela transformação, repetindo o procedimento.

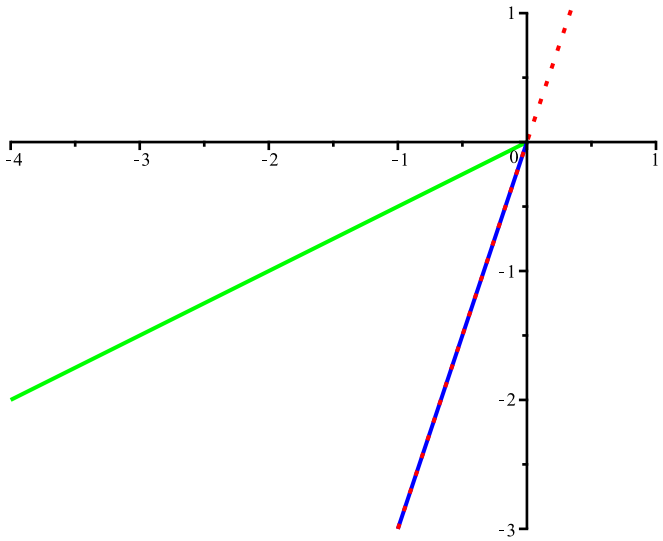
```
> w1 := vector([-4,-2]);
```

$$w1 := \begin{bmatrix} -4 & -2 \end{bmatrix} \quad \textbf{(13)}$$

```
> w2 := evalm(matriz_transf &* w1);
```

$$w2 := \begin{bmatrix} -1 & -3 \end{bmatrix} \quad \textbf{(14)}$$

```
> b := plot([[[0, 0], [w1[1], w1[2]]], [[0, 0], [w2[1], w2[2]]]], color = [green, blue], scaling
      = constrained) :
> display([b, a]);
```



Exemplo de utilização de recursos de animação no Estudo de Movimentos: Cisalhamento

Iremos apresentar uma pequena amostra dos recursos gráficos e de animação que o Maple V possui, aplicados no estudo de operadores lineares do plano. A técnica da álgebra matricial para representar os movimentos continua sendo a base teórica fundamental. A programação da animação foi realizada pela bolsista do REENGE Karina Moreno Passos. Os recursos gráficos são excelentes para compreender as propriedades geométricas das matrizes de movimentos.

Descrição

Esta animação gráfica consiste em cisalhar um quadrado , sendo que :

- O quadrado é definido no procedimento , ou seja , a figura é pré-fixada .

- A animação é obtida através de quadros (frames) e o fator de cisalhamento (FC) adotado é 2 .

Detalhes do Procedimento

. Nome do Procedimento : CisalhaQuadr2D( )

. Chamada do procedimento : CisalhaQuadr2D( );

. Parâmetros para a chamada do procedimento : nenhum

. Bibliotecas do Maple utilizadas : < linalg e < plots

. Linguagem de programação utilizada : linguagem própria do Maple

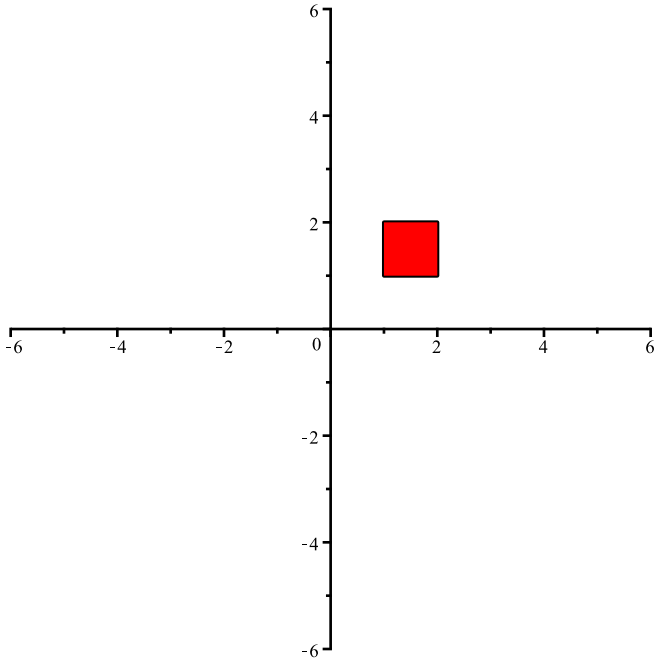
. Número de frames : 5

Definição do código de procedimento

```
> CisalhaQuadr2D := proc( )
    local fig, u1, u2, u3, u4, v1, v2, v3, v4, T, G, i;
    restart; with(linalg) : with(plots) :
    fig := array(1 ..1, 1 ..5);
    u1 := vector([1, 1]);
    u2 := vector([1, 2]);
    u3 := vector([2, 2]);
    u4 := vector([2, 1]);
    v1 := vector([1, 1]);
    v2 := vector([1, 2]);
    v3 := vector([2, 2]);
    v4 := vector([2, 1]);
    G := 0;
    i := 1;
    while (G ≤ 2)
    do
    fig[1, i] := PLOT(POLYGONS([[v1[1], v1[2]], [v2[1], v2[2]], [v3[1], v3[2]], [v4[1],
        v4[2]], [v1[1], v1[2]]], COLOUR(RGB, 1, 0, 0)));
    G := G + 1/2;
    i := i + 1;
    T := matrix([[1, G], [0, 1]]);
    v1 := evalm(T &* u1);
    v2 := evalm(T &* u2);
```

```
v3 := evalm(T &* u3);

v4 := evalm(T &* u4);

od:

display([seq(fig[1,j], j = 1..5)], insequence = true, view = [-6..6, -6..6], scaling
    = constrained);

end:

CisalhaQuadr2D( );
```

```
Warning, (in CisalhaQuadr2D) `j` is implicitly declared local
Warning, the restart command only works at the top level. It cannot
be executed within a procedure, or from a file being read by the read
statement.
```



Exemplos de animação de figuras com Movimentos de rotação

Vamos apresentar aqui um exemplo de utilização do movimento de rotação, para animar pontos, polígonos e curvas parametrizadas.

```
> restart; with(linalg) : with(plots) :
```

A matriz de rotação de ângulo θ radianos medido positivamente no sentido anti-horário é dada por

```
> matrix(2, 2, [cos(theta), -sin(theta), sin(theta), cos(theta)]);
```

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \quad \textbf{(15)}$$

que é uma matriz ortogonal

```
> Rot := theta -> matrix(2, 2, [cos(theta), -sin(theta), sin(theta), cos(theta)]);
```

$$Rot := \theta \mapsto matrix(2, 2, [\cos(\theta), -\sin(\theta), \sin(\theta), \cos(\theta)]) \quad \textbf{(16)}$$

Veja os resultados:

```
> Rot(theta), Rot(alpha), Rot(Pi/3);
```

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}, \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}, \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & \frac{1}{2} \end{bmatrix} \quad \textbf{(17)}$$

Agora, uma função que a cada ponto P do plano associa o ponto Q = RotP(P,t) que é o ponto P rotacionado de ângulo t em torno da origem:

```
> RotP := (P, t) -> convert(evalm(Rot(t) &* P), 'list');
```

$$RotP := (P, t) \mapsto convert(evalm(Rot(t) \;\&*\; P), 'list') \quad \textbf{(18)}$$

A conversão para 'list' foi feita para apresentar os pontos em formato de 'list' e não de 'array' como seria o resultado de evalm . Matrizes e vetores são 'array's.

Vejamos agora o resultado da função:

```
> RotP([1, 0], theta), RotP([0, 1], theta);
```

$$[\cos(\theta), \sin(\theta)], [-\sin(\theta), \cos(\theta)] \quad \textbf{(19)}$$

```
> RotP([x, y], Pi/3);
```

$$\left[\frac{x}{2} - \frac{\sqrt{3}\,y}{2}, \frac{\sqrt{3}\,x}{2} + \frac{y}{2}\right] \quad \textbf{(20)}$$

Vamos agora aos desenhos:

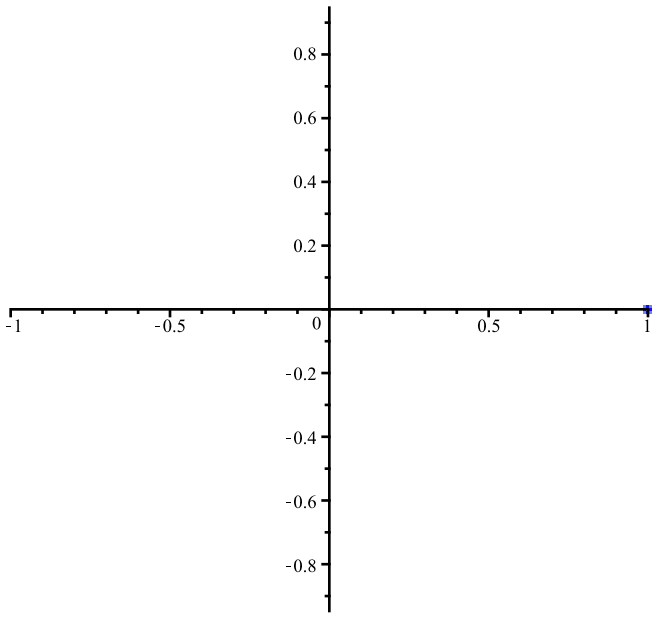
Iniciemos com uma rotina para desenhar um ponto escolhido P girar em torno da origem:

```
> P := [1, 0];
```

$$P := [1, 0] \quad \textbf{(21)}$$

```
> N := 10 : Angulo := 2 * Pi : # N+1 = número de quadros
  for i from 0 to N do
  t := Angulo/N * i :
  quadro[i] := pointplot( RotP(P, t), color = blue, symbol = circle) :
```

```
od: # formando os quadros

display( [seq(quadro[i], i = 0 ..N ) ], insequence = true, scaling = constrained);
    # animação de um ponto
```

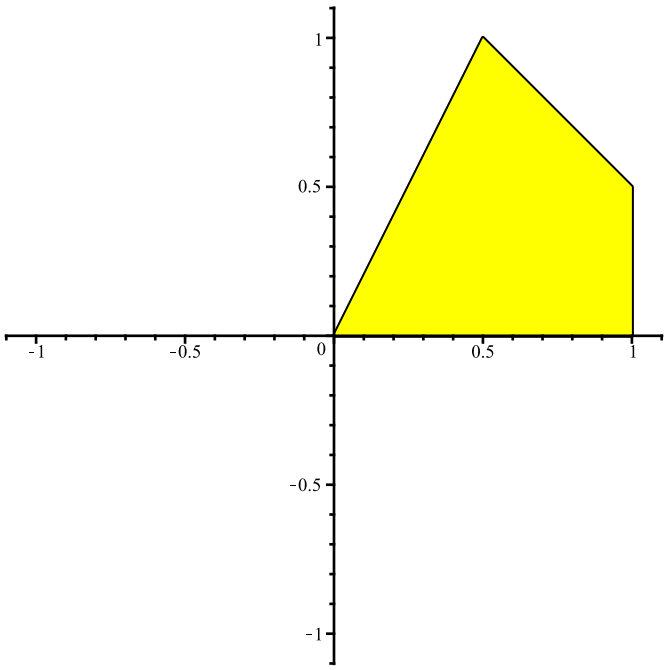


Agora definimos os vértices P1, P2, P3, P4 de um polígono:

```
> P1 := [0, 0] : P2 := [1, 0] : P3 := [1, 1/2] : P4 := [1/2, 1] :
```

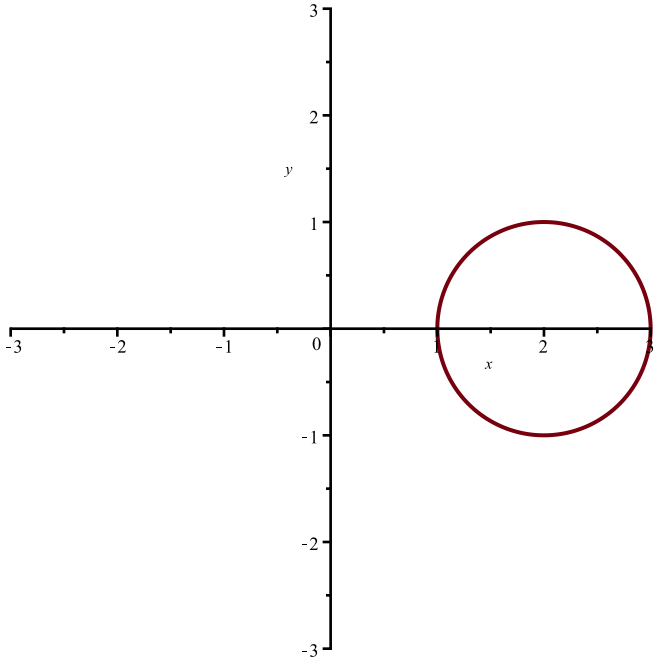
Agora, vamos girar esse polígono de ângulo Abertura em torno da origem:

```
> N := 10 : Abertura := 2 * Pi :

  for i from 0 to N do

  t := Abertura / N * i :

  quadro[i] := polygonplot( [RotP(P1, t), RotP(P2, t), RotP(P3, t), RotP(P4, t) ], color
      = yellow) :

  od:

  display( [seq(quadro[i], i = 0 ..N ) ], insequence = true, scaling = constrained);
```



Agora, vamos girar uma curva parametrizada em torno da origem:

```
> t := 't': #limpando a variável t

  plot( [ 2 + cos(t), sin(t), t = 0 ..2 * Pi ], x = -3 ..3, y = -3 ..3, scaling = constrained);
```




```
> alpha := t → [2 + cos(t), sin(t) ];
```

$$\alpha := t \mapsto [2 + \cos(t), \sin(t)] \quad \textbf{(22)}$$

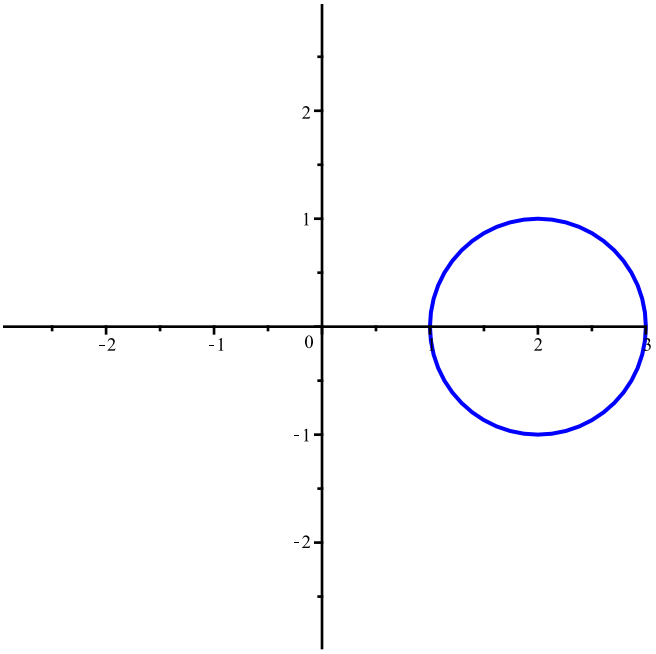
```
> Rotalpha := (t, theta) → RotP(alpha(t), theta);
```

$$Rotalpha := (t, \theta) \mapsto RotP(\alpha(t), \theta) \quad \textbf{(23)}$$

```
> op(Rotalpha(u, v));
```

$$\cos(v)\ (2 + \cos(u)) - \sin(v)\ \sin(u),\ \sin(v)\ (2 + \cos(u)) + \cos(v)\ \sin(u) \qquad \textbf{(24)}$$

```
> animate( [op(Rotalpha(t, theta))], t = 0 ..2 * Pi], theta = 0 ..2 * Pi, scaling = constrained, color
      = blue);
```



Outros Tópicos e Comentários Finais

O pacote de Álgebra Linear do Maple V é bastante completo e útil. Nesta apresentação, não podemos apresentar exemplos de todos os seus recursos, mas além dos exemplos mostrados acima, em [B-P] temos preparado material sobre a linguagem básica de Álgebra Linear, teoria geral de transformações lineares, auto-valores e problemas de diagonalização, produto interno e operadores geométricos. Um anexo com as aplicações também está em preparação. Um ponto a ser observado também na utilização de Softwares é a sua limitação, envolvendo problemas de aproximação nos resultados, ou ainda a interpretação das respostas que o Software nos apresenta de acordo com o problema que nós queremos resolver. Acreditamos que temos um vasto campo de estudo para ser explorado.