Estudo da Oscilação de Membranas usando Maple

D. Frenkel[1] , *L. Golebiowski*[2] *e R. Portugal*[3]

Centro Brasileiro de Pesquisas Físicas – CBPF/LAFEX
Rua Dr. Xavier Sigaud, 150
22290-180 – Rio de Janeiro, RJ – Brasil

[1]Email: frenkel@fis.puc-rio.br
[2]Email: lgolebi@netrio.com.br
[3]Present address: Department of Applied Mathematics, University of Waterloo, Waterloo, Ontario, N2L 3G1 - Canada. Email: rportuga@daisy.uwaterloo.ca or portugal@cat.cbpf.br

Contents

1 Introdução

1.1 Computação Algébrica e Ensino

Atualmente, os microcomputadores têm uma capacidade de memória e de processamento que tornou possível executar programas antigamente restritos a computadores de grande porte. Os primeiros são acessíveis a um número muito maior de pessoas. Este fato trouxe a tona a possibilidade de uso dos sistemas algébricos para o ensino da Matemática aplicada à Física e à Engenharia. Os principais sistemas, *Maple, Mathematica, Macsyma, Reduce* e *Derive*, permitem, além da realização de cálculos algébricos, manipulações gráficas, análise numérica e programação. Com estas ferramentas, dois aspectos do ensino da Matemática podem sofrer transformações.

Primeiro, o aluno não necessita passar por um extenuante treinamento de técnicas de cálculo, como por exemplo, as técnicas de integração por partes, por substituição etc., pois estas técnicas estão implementadas nestes sistemas. Além disso, outras técnicas mais elaboradas também estão implementadas e elas não são ensinadas nas Universidades pois apresentam um grau de dificuldade que as tornam inviáveis de serem usadas por seres humanos. As máquinas as utilizam sem reclamar. Estas técnicas estão disponíveis a qualquer aluno que domine um dos sistemas de computação algébrica.

Segundo, a assimilação dos conceitos matemáticos que estão por trás das técnicas passam a ser a prioridade, assumindo o lugar que lhe é devido. Uma vez diminuído o tempo que é necessário gastar com treinamento de técnicas de cálculo, o professor pode dedicar mais tempo para a transmissão dos conceitos. Porém, não se trata apenas de uma questão de tempo, os sistemas algébricos fornecem ferramentas que facilitam a explanação da parte conceitual. Muitas vezes é uma tarefa árdua explicar com giz, quadro-negro e muita gesticulação de mãos, algo que facilmente é mostrado com animações de gráficos projetados na parede da sala de aula. Os modos normais de vibração de uma membrana circular sem simetria axial podem ser explicados com giz e quadro-negro, porém somente os alunos com uma excelente capacidade de visualização abstrata entenderão.

O tema computação algébrica e ensino tem sido largamente debatido informalmente em conferências e na literatura. Especialmente com relação ao Maple podemos citar importantes referências[1][2].

O problema de oscilação de membranas é um bom exemplo onde a computação algébrica fornece várias ferramentas didáticas para a sua análise. Os principais aspectos do *Maple* podem ser usados para se obter os resultados desejados. Neste trabalho, analisamos o movimento de vários tipos de membranas tendo como objetivo a apresentação de um novo enfoque do problema que se tornou possível com o advento dos sistemas algébricos. Nos concentramos nos aspectos computacionais e didáticos, e apresentamos o desenvolvimento da teoria apenas na parte onde a computação algébrica pode ser de grande auxílio. Os aspectos mais teóricos do problema, como a dedução da equação de onda a partir das leis de Newton ou por métodos variacionais ou os detalhes da manipulação das séries de Fourier foram omitidos e eles podem se obtidos em qualquer uma das referências de Física-Matemática citadas na bibliografia[3][4][5]. Na literatura existem algumas referências que fazem análise do movimento de membranas através do uso de computação algébrica[2][6], no entanto, uma análise sistematizada parece não ter sido feita. A análise de membranas circulares requer funções que possam calcular o n-ésimo zero (n inteiro) das funções cilíndricas de Bessel e de funções correlatas. Apesar do Maple ter rotinas para o cálculo numérico de zeros (comando *fsolve*), a determinação de intervalos que contenham um determinado n-ésimo zero da função de Bessel e nenhum outro zero, não é uma tarefa simples. Este investimento está feito nos apêndices deste trabalho.

1.2 Breve Histórico do Problema da Membrana

A análise do movimento de uma membrana é bastante antiga[7]. Em 1764, Euler publicou um trabalho[4] onde ele obtém as equações para o deslocamento vertical de uma membrana retangular e circular, que são as mesmas analisadas adiante neste trabalho. A parte radial para a membrana circular recai numa equação diferencial de Bessel de ordem inteira. Esta foi a primeira vez que as funções de Bessel de ordem arbitrária foram analisadas [8]. Euler encontrou soluções para estas equações usando o método de separação de variáveis. Estas soluções representam os modos normais de oscilação. A solução geral como superposição dos modos normais teve que esperar pelos trabalhos de Fourier.

Fourier teve dificuldade de publicar seu trabalho de 1807 sobre condução de calor onde ele afirmava que qualquer função podia ser expandida em séries de seno e cosseno. Os membros da Academia de Ciências de Paris não aceitaram as novas idéias argumentando falta de rigor. Em 1819 o trabalho foi finalmente publicado[5].

[4] Novi Commu. Acad. Petrop. Sci. Petrop.,10, 1764, 246-60

[5] M'em. de l'Acad. des Sci., Paris, (2), 4, 1819/20, 185-555

Poisson imediatamente aplicou o método usado por Fourier em vários problemas físicos, entre eles o problema da membrana[6].

No começo deste século, Rayleigh[7] estudou o movimento de uma membrana em forma de um setor circular. Novamente aparecem as funções de Bessel de ordem arbitrária e em seus trabalhos, Rayleigh analisa o comportamento dos zeros para ordens crescentes da função de Bessel em consequência da diminuição do ângulo do setor. Ele obteve uma série de teoremas novos envolvendo as funções de Bessel.

A análise do movimento de membranas de diversos formatos presas nas extremidades ou não, se tornou um problema clássico da Física-Matemática devido a diversos fatores. Este problema é um excelente exemplo do uso da técnica de expansão de Fourier decorrente do método de separação de variáveis para resolução de equações diferencias parciais lineares. Além disso, dependendo do formato da membrana, diversas funções especiais são usadas, como as funções de Bessel para membrana circular e funções de Mathieu para membranas elíticas. O movimento da membrana é um exemplo do problema de autovalores de Sturm-Liouville [4]. Ele serve como aplicação de diversos teoremas gerais do problema de Sturm-Liouville como, por exemplo, os teoremas envolvendo as linhas nodais das autofunções de equações diferenciais auto-adjuntas lineares de segunda ordem. A membrana, por ser bidimensional, apresenta uma riqueza bem maior do que o problema unidimensional de oscilação de cordas.

1.3 Maple

O Maple[9][10][11][12] é uma linguagem de computação que possui quatro aspectos gerais que são:

- computação algébrica
- computação numérica
- computação gráfica
- programação

Todos estes aspectos estão integrados formando um corpo único. Por exemplo: a partir de um resultado algébrico, uma análise numérica ou gráfica pode imediatamente ser feita. Em geral, na análise de um problema várias ferramentas são necessárias. Se estas ferramentas não estiverem no mesmo *software*, o usuário enfrentará uma série de dificuldades para compatibilizar a saída de um *software* com a entrada de outro, além de ter que familiarizar-se com diferentes notações e estilos. É claro que o Maple não elimina completamente o uso de linguagens numéricas ou gráficas. Em aplicações mais elaboradas pode ser necessário usar recursos de linguagens como C ou Fortran. O Maple tem interface com estas linguagens no sentido de que um resultado algébrico encontrado no Maple pode ser convertido para a notação da linguagem C ou para o Fortran.

Os aspectos novos trazidos por esse *software* juntamente com outros sistemas algébricos são a computação algébrica e a programação simbólica. A computação algébrica é uma área que teve um forte desenvolvimento nas décadas de 60 e 70, quando foram elaborados importantes algoritmos para integração analítica e fatoração de polinômios. Estes algoritmos estão baseados na Álgebra Moderna, que guia toda a implementação do núcleo de qualquer sistema algébrico. No início do desenvolvimento desta área, uma série de tentativas foram feitas com o objetivo de implementar os recursos heurísticos provenientes da Inteligência Artificial, no entanto os resultados foram ineficientes. Os programas para integração analítica eram lentos, resolviam algumas integrais difícies porém falhavam em integrais simples que qualquer aluno de Cálculo consegue calcular.

O Maple é uma linguagem de programação simbólica. Os construtores deste sistema optaram em desenvolver um pequeno núcleo escrito na linguagem C gerenciando as operações que necessitam de maior velocidade de processamento e partir deste núcleo desenvolveram uma nova linguagem. O próprio Maple foi escrito nesta nova linguagem. Noventa e cinco por cento dos algoritmos estão escritos na linguagem Maple e estão acessíveis ao usuário. Esta opção dos seus arquitetos é muito saudável, pois uma linguagem que pode gerar todo um sistema algébrico do porte do Maple, certamente é uma boa linguagem de programação.

[6] M'em. de l'Acad. des Sci., Paris, (2), 8, 1829, 357-570

[7] Phil. Mag.(6) XXI. (1911), pp. 53-58 [Scientific Papers, VI. (1920), pp.1-5]

Na análise do problema da membrana apresentada aqui, todos estes aspectos do Maple foram usados para se obter os resultados desejados. O aspecto algébrico foi usado para desenvolver o problema passo a passo a partir da equação de onda. O aspecto gráfico foi usado para fazer a animação do movimento dos modos normais da membrana e do seu movimento quando submetida a condições iniciais. O aspecto numérico foi essencial para se trabalhar com as funções de Bessel, pois seus zeros só podem ser encontrados por métodos numéricos. As ferramentas de programação permitiram criar aplicativos para análise deste problema onde podemos, por uma simples variação de parâmetros, refazer toda uma sequência de passos.

A principal interface do Maple com o usuário é a *worksheet*. Este trabalho foi todo desenvolvido em *worksheets*[8] que permitem a visualização da dinâmica das animações. Elas permitem também que um usuário mude parâmetros e teste novas hipóteses. A versão impressa foi obtida por conversão das *worksheets* em arquivo *Latex*. Todos os gráficos e animações foram exportados automaticamente para arquivos na linguagem *postscript*, e incluídos na conversão do arquivo *dvi*[9] em arquivo *postscript*.

1.4 Resumo

A proposta deste trabalho é utilizar as ferramentas matemáticas e gráficas do Maple para estudar o problema da oscilação de vários tipos de membranas. Usando a linguagem simples de programação e os recursos gráficos do Maple, foi possível fazer uma análise detalhada do problema e gerar a animação do movimento de vários tipos de membranas.

Começamos o estudo com uma breve introdução da equação de movimento, apontando as funções matemáticas usadas no trabalho. Analisamos em seguida a resolução da equação de onda da membrana retangular e fazemos a animação dos modos normais. É feita também a análise da degenerescência e da animação com condições iniciais.

O estudo continua com a análise da membrana circular. A complexidade no trato das funções matemáticas e a quantidade de cálculo foram reduzidas com os recursos algébricos do Maple. Neste caso, também analisamos os modos normais de vibração e o movimento com condições iniciais.

Finalmente, terminamos o estudo com a membrana em formato de anel, seguindo a mesma linha de análise dos outros tipos de membrana e visualizando graficamente este tipo de oscilação usando os recursos gráficos do Maple.

O trabalho foi desenvolvido para ser usado interativamente pelo usuário que, com simples modificações de parâmetros, pode visualizar qualquer modo normal de vibração. É possível também escolher as condições iniciais e obter o movimento resultante.

[8] As *worksheets* podem ser obtidas através de *ftp* anômino no endereço *anonymous@lca1.drp.cbpf.br* no diretório *pub/membrana*.

[9] O processador Latex cria, a partir de um arquivo texto, um arquivo compilado com extensão *dvi* pronto para ser enviado para a impressora.

2 Equação de Movimento da Membrana

Vamos considerar uma membrana horizontal uniformemente distentida em todas as direções por uma tração T. A membrana pode ter qualquer forma que será especificada pelas condições de contorno. Qualquer que seja a forma, vamos analisar o caso em que toda a borda da membrana está fixa. As oscilações são sempre na direção vertical. Sob certas restrições, como por exemplo pequenas oscilações, a equação de onda que descreve o deslocamento vertical u(x, y) na ausência de forças externas é dada por:

$$c^2 \nabla^2 u = \frac{\partial^2}{\partial t^2} u$$

onde $c^2 = \frac{T}{\mu}$ e μ é a densidade de massa. Esta equação diferencial parcial pode ser resolvida pelo método de separação de variáveis, de maneira que a solução geral fica expressa na forma de uma série de Fourier generalizada. No caso da membrana retangular, a solução é uma série de Fourier tripla usual e no caso da membrana circular axialmente simétrica, a solução é uma série de Fourier-Bessel na parte espacial e Fourier usual na parte temporal. Vamos analisar o movimento oscilatório destes destes dois tipos de membranas e membranas com o formato de anel.

3 Membrana Retangular

3.1 Resolução da Equação de Onda

A membrana retangular de comprimento a e largura b presa nas bordas é caracterizada pelas seguintes condições de contorno: u(0, y, t) = 0 e u(a, y, t) = 0 para $y \leq b$ e u(x, 0, t) = 0 e u(x, b, t) = 0 para $x \leq a$. Vamos atribuir a equação diferencial deste problema à variável *wave_eq*:

```
> restart;
> wave_eq := Diff(u,x,x) + Diff(u,y,y) = Diff(u,t,t)/c^2;
```

$$wave\_eq := (\frac{\partial^2}{\partial x^2} u) + (\frac{\partial^2}{\partial y^2} u) = \frac{\frac{\partial^2}{\partial t^2} u}{c^2}$$

Vamos resolver essa equação usando o método de separação de variáveis. Vamos supor que u(x, y, t) é uma função separável nas variáveis x, y, e t. Assim, u(x, y, t) = X(x) Y(y) T(t):

```
> eq1 := expand(value(subs(u=X(x)*Y(y)*T(t),wave_eq)/(X(x)*Y(y)*T(t))));
```

$$eq1 := \frac{\frac{\partial^2}{\partial x^2} \mathrm{X}(x)}{\mathrm{X}(x)} + \frac{\frac{\partial^2}{\partial y^2} \mathrm{Y}(y)}{\mathrm{Y}(y)} = \frac{\frac{\partial^2}{\partial t^2} \mathrm{T}(t)}{\mathrm{T}(t)\, c^2}$$

Cada termo da expressão acima deve ser constante, pois cada um é função de uma variável independente. Vamos selecionar o termo dependente de x, e igualar a constante $\lambda 1$:

```
> eqx := expand(X(x)*(select(has, lhs(eq1), x) = lambda1));
```

$$eqx := \frac{\partial^2}{\partial x^2} \mathrm{X}(x) = \mathrm{X}(x)\, \lambda 1$$

A equação diferencial acima pode ter soluções oscilatórias, lineares ou exponenciais dependendo se a constante $\lambda 1$ for negativa, zero ou positiva. Para que as condições de contorno sejam satisfeitas, isto é, X(0) = 0 e X(a) = 0, temos que impor que a constante $\lambda 1$ seja negativa e que ela assuma valores discretos dados por:

```
> lambda1 := -m^2*Pi^2/a^2;
```

$$\lambda 1 := -\frac{m^2\, \pi^2}{a^2}$$

onde m é um inteiro positivo. A solução geral da equação da parte x é:

```
> sol_X := dsolve({eqx,X(0)=0}, X(x));
```

$$sol\_X := \mathrm{X}(x) = \_C2 \sin(\frac{\pi\, m\, x}{a})$$

Vamos eliminar a constante arbitrária por enquanto:

```
> assign(simplify(subs(_C1=1, _C2=1, sol_X)));
> X(x);
```

$$\sin(\frac{\pi\, m\, x}{a})$$

Vamos repetir o mesmo procedimento para o termo dependente de y:

```
> eqy := expand(Y(y)*(select(has, lhs(eq1), y) = lambda2)):
> lambda2 := -n^2*Pi^2/b^2:
> sol_Y := dsolve({eqy,Y(0)=0}, Y(y)):
> assign(simplify(subs(_C1=1, _C2=1, sol_Y)));
> Y(y);
```

$$\sin(\frac{\pi\, n\, y}{b})$$

A equação diferencial para a função dependente da variável t é obtida da seguinte forma:

```
> eqt := normal(c^2*T(t)*subs(eqx, eqy, eq1));
```

$$eqt := -\frac{c^2\, \mathrm{T}(t)\, \pi^2\, (m^2\, b^2 + n^2\, a^2)}{a^2\, b^2} = \frac{\partial^2}{\partial t^2}\, \mathrm{T}(t)$$

cuja solução geral é:

```
> sol_T := dsolve(eqt, T(t));
```

$$sol\_T := \mathrm{T}(t) = \_C1\, e^{(\frac{\sqrt{-m^2\, b^2 - n^2\, a^2}\, \pi\, c\, t}{b\, a})} + \_C2\, e^{(-\frac{\sqrt{-m^2\, b^2 - n^2\, a^2}\, \pi\, c\, t}{b\, a})}$$

Vamos colocar esta solução numa forma mais conveniente:

```
> simplify(evalc(sol_T),{_C1+_C2=A(m,n), _C1-_C2=B(m,n)/I});
> assign(");
```

$$\mathrm{T}(t) = \cos(\frac{\sqrt{m^2\, b^2 + n^2\, a^2}\, \pi\, c\, t}{b\, a})\, \mathrm{A}(m,\, n) + \sin(\frac{\sqrt{m^2\, b^2 + n^2\, a^2}\, \pi\, c\, t}{b\, a})\, \mathrm{B}(m,\, n)$$

onde A e B são constantes que podem depender dos autovalores m e n. Vamos definir a frequência do modo normal $< m,\, n >$:

```
> omega := (m,n) -> (m^2*b^2+n^2*a^2)^(1/2)*Pi*c/b/a;
```

$$\omega := (m,\, n) \rightarrow \frac{\sqrt{m^2\, b^2 + n^2\, a^2}\, \pi\, c}{b\, a}$$

e definir os modos normais como uma função de m e n. Existem duas possibilidades que são:

```
> normal_mode := unapply(simplify(subs(A=1, B=0, X(x)*Y(y)*T(t))),m,n);
```

$$normal\_mode := (m,\, n) \rightarrow \sin(\frac{\pi\, m\, x}{a})\, \sin(\frac{\pi\, n\, y}{b})\, \cos(\frac{\sqrt{m^2\, b^2 + n^2\, a^2}\, \pi\, c\, t}{b\, a})$$

```
> normal_mode1 := unapply(simplify(subs(A=0, B=1, X(x)*Y(y)*T(t))),m,n);
```

$$normal\_mode1 := (m,\, n) \rightarrow \sin(\frac{\pi\, m\, x}{a})\, \sin(\frac{\pi\, n\, y}{b})\, \sin(\frac{\sqrt{m^2\, b^2 + n^2\, a^2}\, \pi\, c\, t}{b\, a})$$

Uma vez que a equação diferencial do movimento da membrana é linear e homogênea, a solução geral é uma superposição de todas as soluções linearmente independentes possíveis. Assim, tomaremos o somatório sobre todos os valores de m e n:

```
> u := unapply(Sum(Sum(X(x)*Y(y)*T(t),'m'=1..infinity),'n'=1..infinity),'x',' y','t');
```

$$u := (x,\, y,\, t) \rightarrow \sum_{n=1}^{\infty} \left(\sum_{m=1}^{\infty} \sin(\frac{\pi\, m\, x}{a})\, \sin(\frac{\pi\, n\, y}{b}) \right.$$

$$\left. (\cos(\frac{\sqrt{m^2\, b^2 + n^2\, a^2}\, \pi\, c\, t}{b\, a})\, \mathrm{A}(m,\, n) + \sin(\frac{\sqrt{m^2\, b^2 + n^2\, a^2}\, \pi\, c\, t}{b\, a})\, \mathrm{B}(m,\, n)) \right)$$

As constantes arbitrárias $\mathrm{A}(m,\, n)$ e $\mathrm{B}(m,\, n)$ são determinadas a partir das condições iniciais. Geralmente elas são a posição inicial $u_0(x,\, y)$ e a velocidade inicial $v_0(x,\, y)$:

$$u_0(x,\, y) = \mathrm{u}(x,\, y,\, 0)$$

$$v_0(x,\, y) = (\frac{\partial}{\partial t}\, \mathrm{u}(x,\, y,\, t))\ \ em\ \ t\ = 0$$

As equações acima permitem determinar $\mathrm{A}(m,\, n)$ e $\mathrm{B}(m,\, n)$ por inversão da série dupla de Fourier, cujos resultados são:

$$\mathrm{A}(m,\, n) = \frac{4 \int_0^b \int_0^a \mathrm{u0}(x,\, y)\, \sin(\frac{n\, \pi\, x}{a})\, \sin(\frac{m\, \pi\, y}{b})\, dx\, dy}{a\, b}$$

e

$$\mathrm{B}(m,\,n) = \frac{4\int_0^b\int_0^a \mathrm{v0}(x,\,y)\,\sin(\frac{n\,\pi\,x}{a})\,\sin(\frac{m\,\pi\,y}{b})\,dx\,dy}{a\,b\,\omega(m,\,n)}$$

A título de segurança, vamos limpar as variáveis que não serão mais usadas, e proteger as variáveis pertinentes:

```
> unassign('wave_eq,eq1,eqx,eqy,lambda1,lambda2 ,eqt,X(x),Y(y),T(t),sol_X,sol_Y,sol_T');
> protect('omega,normal_mode,normal_mode1,u'); # protect assigned variables
> protect('x,y,t,m,n'); # protect reserved variables
```

3.2 Animação dos Modos Normais

Os modos normais podem ser animados com a função *animate3d*. Vamos construir o procedimento *animate_mode*, que anima o modo especificado pelos seus dois primeiros argumentos. Por exemplo, *animate_mode*(1,2) faz a animação do modo m=1 e n=2. Sendo um comando de animação, ele deve aceitar as mesmas opções do comando *animate3d*. Se nada for dito sobre a opção *style*, este procedimento assume que o estilo é *patch*, que é indicado para este tipo de aplicação. As outras opções seguem o *default* do comando *animate3d*.

animate_mode := **proc**(*m*::*posint*, *n*::*posint*)
  local *FRAM*, *Options*;
  global *a*, *b*, *x*, *y*, *t*, *animate3d*;
    if not type([*a*, *b*, *c*], list(*numeric*)) **then**
      ERROR(‘*The constants a, b and c must be numeric.*‘)
    elif not assigned(ω) **then** ERROR(‘*The function omega must be defined.*‘)
    elif not assigned(*normal_mode*) **then**
      ERROR(‘*The function normal_mode must be defined.*‘)
    fi;
    if nargs = 2 **then** *Options* := *'frames'* = 8, *'style* = *patch'*; *FRAM* := 8
    else
      Options := $\mathrm{args}_{3\,..\,\mathrm{nargs}}$;
      if has({*Options*}, *'frames'*) **then** *FRAM* := subs({*Options*}, *'frames'*)
      else *Options* := *Options*, *'frames'* = 8; *FRAM* := 8
      fi;
      if not has({*Options*}, *'style'*) **then** *Options* := *Options*, *'style* = *patch'* **fi**
    fi;
    plots$_{animate3d}$(normal_mode(*m*, *n*), *x* = 0 .. *a*, *y* = 0 .. *b*,
      t = 0 .. 2 × π × *FRAM*/(ω(*m*, *n*) × (*FRAM* + 1)), *Options*)
  end

Para fazer a animação de um determinado modo, temos que dar valores numéricos para os parâmetros a, b e c. Vamos supor que o comprimento é *a=1*, a largura é *b=2* e a constante *c=1* em um certo sistema de unidades:

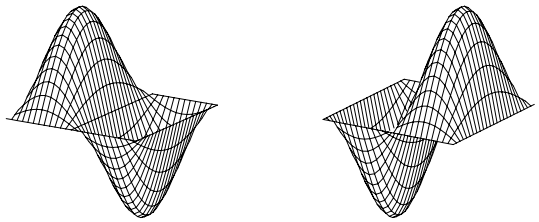
```
> a := 1; b := 2; c := 1;
```

$$a := 1$$
$$b := 2$$
$$c := 1$$

Vamos fazer a animação do modo <1,2> com as seguintes opções[10]:

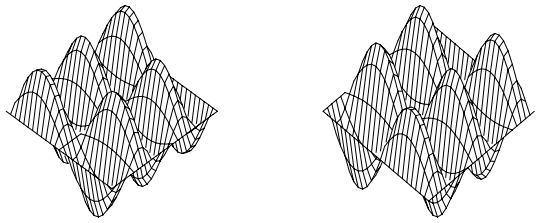
```
> animate_mode(1,2,frames=2,style=hidden,scaling=unconstrained,
> color=black,orientation=[32,78]);
```

[10] Na versão impressa, estamos mostrando a animação dos modos normais com apenas duas poses que correspondem aos extremos da oscilação. Usamos a opção *scaling* = *unconstrained* pois facilita a compreensão das características do modo normal. No entanto, a equação de onda que estamos usando aqui é válida somente para pequenas oscilações.



As linhas nodais são curvas cujos pontos se mantêm em repouso durante todo instante. Elas podem ser melhor visualizadas através da opção *style=patchcontour*. Por exemplo, vejamos a animação do modo <2,5>:

```
> animate_mode(5,2,frames=2,style=hidden,color= black,scaling=unconstrained);
```



Podemos ver que há 4 retas nodais paralelas ao eixo y e 1 reta nodal paralela ao eixo x. Esta característica pode ser generalizada para qualquer modo de vibração. O modo <*m,n*> tem *m-1* retas nodais paralelas ao eixo y e *n-1* retas nodais paralelas ao eixo x. Em geral, as linhas nodais nem sempre são retas. Abaixo, mostraremos um exemplo de uma linha nodal fechada aproximadamente circular em uma membrana quadrada. Isso ocorre quando existe degenerescência.

3.3 Análise da Degenerescência

Em um modo normal de vibração, todos os pontos da membrana vibram com a mesma frequência, de forma que cada modo normal tem um frequência característica $\omega_{m,n}$. Se a membrana for quadrada, dois modos normais diferentes podem oscilar com a mesma frequência. Por exemplo, para a membrana quadrada de largura de uma unidade, os modos <1,2> e <2,1> têm a mesma frequência:

```
> a:=1; b:=1;
```

$$a := 1$$
$$b := 1$$

```
> omega(1,2);
```

$$\sqrt{5}\,\pi$$

```
> omega(2,1);
```

$$\sqrt{5}\,\pi$$

Neste caso, qualquer combinação linear destes modos de vibração também vai oscilar na mesma frequência. Este fenômeno é chamado de ***degenerescência***. Isto ocorre sempre que um autovalor (a frequência) é degenerado, possuindo mais de um autovetor associado (o modo normal). O procedimento a seguir faz a animação da combinação linear c_1 normal_mode(m, n)+c_2 normal_mode(n, m). O procedimento deve ser chamado da forma animate_lc(mode(m, n), $c1$, $c2$):

```
animate_lc := proc(mode::function, c1::numeric, c2::numeric)
    local FRAM, Options, m, n;
    global a, b, ω, x, y, t;
      if not type([a, b], list('numeric')) then ERROR(`a and b must be numerical.`)
      elif not assigned(normal_mode) then
        ERROR(`The function normal_mode must be defined.`)
      elif a ≠ b then ERROR(`The membrane must be square`)
      elif nargs < 3 then ERROR(`The first argument is of form mode(m,n), where m an\
        d n are positive integers. The second and the third are the coefficients of the linea\
        r combination`)
      elif nops(mode) ≠ 2 then
        ERROR(`First argument is of form mode(m,n), where m and n are positive integers`)
      fi;
      m := op(1, mode);
      n := op(2, mode);
      if not (type(m, posint) and type(n, posint)) then
        ERROR(`The mode is specified by positive integers`)
      fi;
      if nargs = 3 then Options := 'frames' = 8, 'style = patch'; FRAM := 8
      else
        Options := args4 .. nargs;
        if has({Options}, 'frames') then FRAM := subs({Options}, 'frames')
        else Options := Options, 'frames' = 8; FRAM := 8
        fi;
        if not has({Options}, 'style') then Options := Options, 'style = patch' fi
      fi;
      plots_animate3d(c1 × normal_mode(m, n) + c2 × normal_mode(n, m), x = 0 .. a, y = 0 .. b,
        t = 0 .. 2 × π × FRAM/(ω(m, n) × (FRAM + 1)), Options)
    end
```

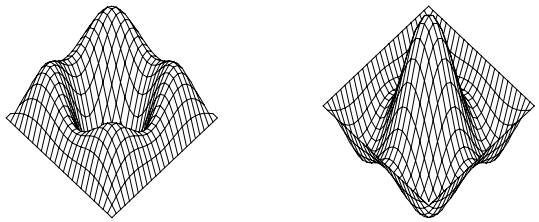
Vamos considerar uma membrana quadrada:

```
> a:=1; b:=1; c:=1;
```

$$a := 1$$
$$b := 1$$
$$c := 1$$

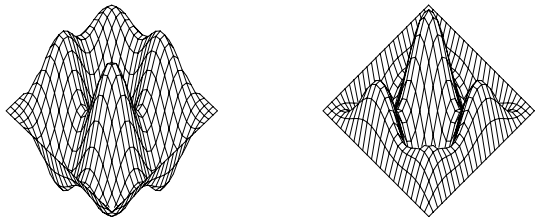
Vamos fazer a animação da combinação linear normal_mode(1, 3) + normal_mode(3, 1), onde podemos verificar que a linha nodal é uma curva fechada:

```
> animate_lc(mode(1,3),1,1,frames=2,style=hidde n,color=black,scaling=constrained);
```



A combinação linear normal_mode(1, 4) + normal_mode(4, 1) tem como linha nodal uma curva fechada e uma reta (uma das diagonais da membrana):

```
> animate_lc(mode(1,4),1,1,frames=2,style=hidde n,color=black,scaling=constrained);
```



A linha nodal é descrita pela equação

$$c_1\,\text{normal\_mode}(m,\,n) + c_2\,\text{normal\_mode}(n,\,m) = 0,$$

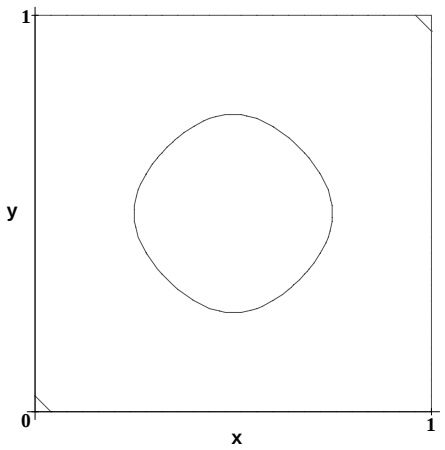
que depende das variáveis x e y. O fator temporal é comum aos dois termos e portanto é cancelado. Através do comando *implicitplot,* podemos fazer o gráfico da curva y contra x mesmo sem resolver explicitamente a equação. O procedimento *nodal_line* a seguir faz o desenho da linha nodal. Os argumentos deste procedimento são iguais aos do procedimento *animate_lc*.

```
nodal_lines := proc(mode::function, c1::numeric, c2::numeric)
    local m, n, Options;
    global a, b, x, y, t, s;
      if not type([a, b], list('numeric')) then ERROR('a and b must be numerical.')
      elif not assigned(normal_mode) then
```

```
        ERROR(`The function normal_mode must be defined.`)
      elif a ≠ b then ERROR(`The membrane must be square`)
      elif nargs < 3 then ERROR(`The first argument is of form mode(m, n), where m an\
        d n are positive integers. The second and the third are the coefficients of the linea\
        r combination`)
      elif nops(mode) ≠ 2 then
        ERROR(`First argument is of form mode(m, n), where m and n are positive integers`)
      fi;
      m := op(1, mode);
      n := op(2, mode);
      if not (type(m, posint) and type(n, posint)) then
        ERROR(`The mode is specified by positive integers`)
      fi;
      Options := args4 .. nargs;
      s := subs(t = 0, c1 × normal_mode(m, n) + c2 × normal_mode(n, m));
      plots_implicitplot(s, x = 0 .. a, y = 0 .. b, 'scaling = constrained', 'xtickmarks' = [0, a],
        'ytickmarks' = [0, b], Options)
    end
```

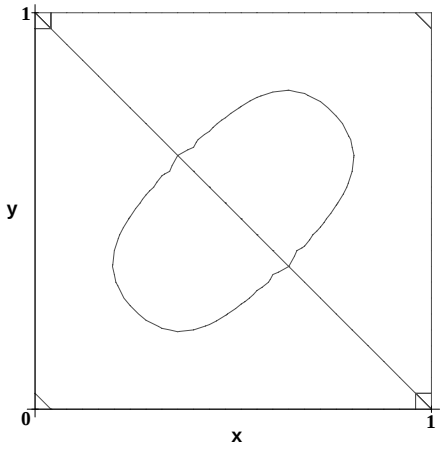
A linha nodal dos dois últimos exemplos que vimos acima são:

```
> nodal_lines(mode(1,3), 1, 1, labelfont=[HELVETICA, BOLD,20],axesfont=[TIMES, BOLD,20]);
```



e

```
> nodal_lines(mode(1,4), 1, 1,labelfont=[HELVETICA, BOLD,20],axesfont=[TIMES, BOLD,20]);
```



Este último gráfico pode ser melhorado acrescentando-se a opção $grid = [i, j]$ onde i e j são números maiores que 25 (*default* da opção *grid*). Por exemplo, nodal_line(mode(1, 4), 1, 1, $grid = [i, j]$). Os argumentos extras do procedimento *nodal_line* são repassados para o comando *implicitplot*, como é o caso do argumento $grid = [i, j]$.

3.4 Animação com Condições Iniciais

Encontramos anteriormente a função u(x, y, t) que descreve o movimento vertical da membrana retangular de largura a e comprimento b presa nas bordas. Esta solução é uma série de Fourier dupla. Uma vez que os modos normais com pequenos valores de m e n dominam sobre os que tem grandes valores, podemos truncar o somatório duplo. O ponto de truncagem satisfatório depende das condições iniciais do problema. Vamos converter o somatório infinito num somatório finito com p^2 termos e definir a função u_p(x, y, t, p) da seguinte forma:

```
> unassign('a,b,c,A,B,u0,v0');
> # Unassign the reserved variables of this problem
> # a - width of the membrane
> # b - length of the membrane
> # c - the constant of wave equation
> # m and n - the oscillation mode
> # x,y,t - the spatial and temporal coordinates
> # A and B - the constants of Fourier expansion
> # u0 - the initial displacement of the membrane
> # v0 - the initial velocity of the membrane
> u_p := unapply(subs(infinity='p',u(x,y,t)),x,y,t,p);
```

$$u\_p := (x, y, t, p) \rightarrow \sum_{n=1}^{p} \left(\sum_{m=1}^{p} \sin(\frac{\pi\, m\, x}{a}) \sin(\frac{\pi\, n\, y}{b}) \right.$$
$$\left. (\cos(\frac{\sqrt{m^2\, b^2 + n^2\, a^2}\, \pi\, c\, t}{b\, a})\, \mathrm{A}(m, n) + \sin(\frac{\sqrt{m^2\, b^2 + n^2\, a^2}\, \pi\, c\, t}{b\, a})\, \mathrm{B}(m, n)) \right)$$

As constantes A(m, n) e B(m, n), que aparecem na função u_p, podem ser obtidas a partir dos seguintes procedimentos baseados nas expressões estabelecidas no final da seção 1.1:

```
> unprotect('A,B'):
> A := proc(m,n) global u0,v0,a,b; `A/Auv`(m,n,eval(u0),eval(v0),a,b,eval(AB_simp))
> end:
> `A/Auv` := proc(m,n,u0,v0,a,b,AB_simp)
> local x,y;
> options remember;
> AB_simp(4/a/b*int(int(u0(x,y)*sin(m*Pi*x/a)*s in(n*Pi*y/b),x=0..a),y=0..b));
> end:
> B := proc(m,n) global u0,v0,a,b; `B/Buv`(m,n,eval(u0),eval(v0),a,b,eval(AB_simp))
> end:
> `B/Buv` := proc(m,n,u0,v0,a,b,AB_simp)
> local x,y;
> options remember;
> AB_simp(4/a/b/omega(m,n)*int(int(v0(x,y)*sin( m*Pi*x/a)*sin(n*Pi*y/b),x=0..a),y=0..b))
> end:
> protect('A,B'):
```

Vamos usar a seguinte função para simplificar os coeficientes A e B:

```
> AB_simp := x -> simplify(x);
```

$$AB\_simp := simplify$$

Os procedimentos A e B chamam os procedimentos A/Auv e B/Buv que têm a opção *remember*. Isto quer dizer que se eles forem chamados mais de uma vez com os mesmos argumentos (m, n, $u0$, $v0$), os coeficientes A(m, n) e B(m, n) não serão recalculados, economizando tempo. Antes de fazer a animação com condições iniciais, vamos calcular as contantes A(m, n) e B(m, n) para m e n inteiros positivos genéricos. É mais eficiente calcular estas constantes para m e n genéricos e depois substituir valores numéricos, do que calcular diretamente para diversos valores numéricos de m e n. Vamos usar as seguintes condições iniciais:

```
> u0 := (x,y) -> 1;
```

$$u0 := 1$$

```
> v0 := (x,y) -> 0;
```

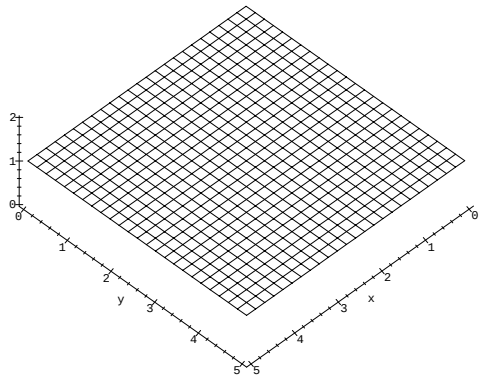
$$v0 := 0$$

```
> a:=5; b:=5; c:=1;
```

$$a := 5$$
$$b := 5$$
$$c := 1$$

O gráfico da posição inicial é:

```
> plot3d(u0(x,y),x=0..a,y=0..b,axes=framed,styl e=hidden,color=black,scaling=constrained);
```



Para calcular A(m, n) e B(m, n) com m e n inteiros positivos genéricos, vamos usar a função *assume*. Desta forma, o comando *simplify* dos procedimentos A e B poderá fazer as simplificações pertinentes:

```
> unprotect('m,n');
> assume(m,integer); assume(n,integer);
> protect('m,n');
> A(m,n);
```

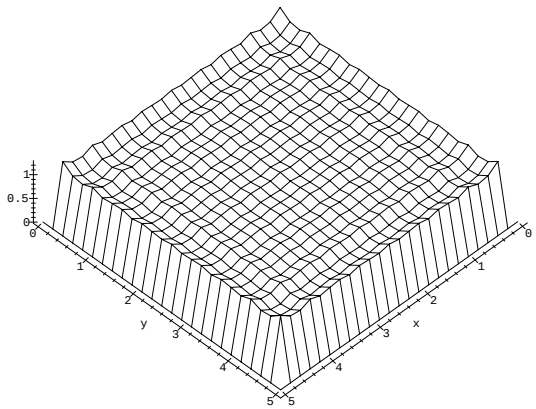
$$-4\,\frac{-(-1)^{(n\tilde{}+m\tilde{})}+(-1)^{n\tilde{}}+(-1)^{m\tilde{}}-1}{n\tilde{}\,\pi^2\,m\tilde{}}$$

```
> B(m,n);
```

$$0$$

Vamos fazer o gráfico da posição inicial da membrana desta vez descrita pela função u_p, pois a partir dele podemos achar um bom valor para a constante p, por comparação com o gráfico obtido diretamente de $u_0(x, y)$. Observe que para $p = 10$ a série de Fourier truncada aproxima razoavelmente bem a posição inicial da membrana:

```
> plot3d(value(u_p(x,y,0,30)),x=0..a,y=0..b,axe s=framed,style=hidden,
> color=black,scaling=constrained);
```



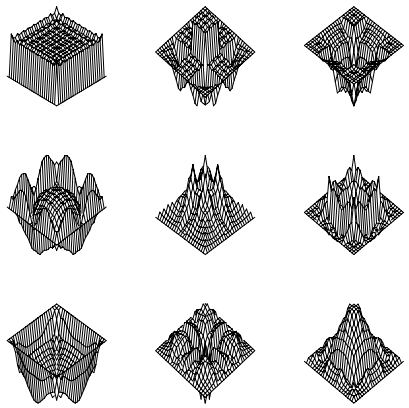
Assim, o valor $p = 10$ é satisfatório para esse tipo de condições iniciais. Note que existe uma ondulação na parte superior do gráfico. Isto é consequência do truncamento da série de Fourier dupla. Esta ondulação só desaparece a rigor, no limite quando $p \to \infty$. Outro detalhe que podemos observar é que a série truncada obedece às condições de contorno, enquanto que a condição inicial não obedece. Esta discrepância também só desaparece quando $p \to \infty$.

O procedimento a seguir faz a animação do movimento da membrana descrita pela função u_p(x, y, t, p). O primeiro argumento representa o número de oscilações dominantes, cuja frequência é a do modo mais baixo ($m = 1$ e $n = 1$). O segundo argumento é o número p que representa a truncagem do somatório duplo:

```
animate_ic := proc(nt, p::posint)
    local FRAM, Options;
    global a, b, ω, u_p, x, y, t;
      if nargs < 2 then ERROR(`This procedure has at least 2 arguments. The first one \
        is the number of oscilations. The second one is the truncation number of the doubl\
        e sum.`)
      fi;
      if not type([a, b, c], list(numeric)) then
        ERROR(`The constants a, b and c must be numerical.`)
      elif not assigned(ω) then ERROR(`The function omega must be defined.`)
      elif not assigned(u_p) then ERROR(`The function normal_mode must be defined.`)
      fi;
      if not type(8 × nt, 'posint') then ERROR(`The first argument is a integer number r\
        epresenting the number of oscillations of the membrane.`)
      fi;
      if nargs = 2 then FRAM := 8 × nt ; Options := 'frames' = FRAM, 'style = patch'
      else
        Options := args3 .. nargs ;
        if has({Options}, 'frames') then FRAM := subs({Options}, 'frames')
        else FRAM := 8 × nt ; Options := 'frames' = FRAM
        fi
      fi;
      if not has({Options}, 'style') then Options := Options, 'style = patch' fi ;
      plots_animate3d(value(u_p(x, y, t, p)), x = 0 .. a, y = 0 .. b, t = 0 .. evalf(2 × π × nt/ω(1, 1)),
        Options)
    end
```

Para as condições iniciais descritas acima, a animação do movimento da membrana com duas oscilações dominantes e $p = 30$ é:

```
> animate_ic(2,30,frames=9,style=hidden,color=b lack);
```



Vamos fazer a animação no caso em que a membrana está parada na posição horizontal e sofre um impulso de cima para baixo em seu centro:

```
> a:=2; b:=2; c:=1;
```

$$a := 2$$
$$b := 2$$
$$c := 1$$

```
> u0 := 0;
```

$$u0 := 0$$

```
> v0 := (x,y) -> -Dirac(x-a/2)*Dirac(y-b/2); # vertical impulse on center
```

$$v0 := (x,\, y) \rightarrow -\mathrm{Dirac}(x - \frac{1}{2}\,a)\,\mathrm{Dirac}(y - \frac{1}{2}\,b)$$

Como antes, vamos calcular $\mathrm{A}(m,\, n)$ e $\mathrm{B}(m,\, n)$ com m e n inteiros positivos genéricos:

```
> A(m,n);
```

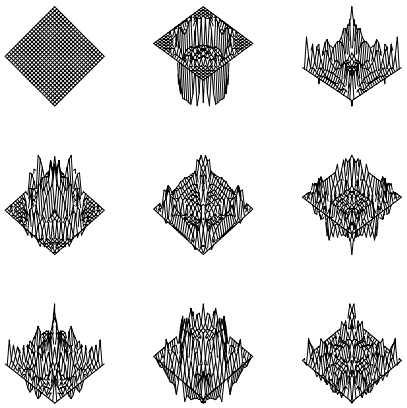
$$0$$

```
> B(m,n);
```

$$-2\,\frac{\sin(\frac{1}{2}\,m\tilde{}\,\pi)\,\sin(\frac{1}{2}\,n\tilde{}\,\pi)}{\sqrt{m\tilde{}^{\,2}+n\tilde{}^{\,2}}\,\pi}$$

Vamos fazer a animação com 2 oscilações e truncar o somatório em $p = 30$ (900 termos):

```
> animate_ic(2,30,frames=9,style=hidden,color=b lack);
```



3.5 Exercícios

1) Verifique que *normal_mode*, *normal_mode1* e u(x, y, t) satisfazem a equação de onda, as condições de contorno e as condições iniciais.

2) a) Resolva a equação de onda para uma membrana obedecendo as seguintes condições de contorno: duas extremidades opostas da membrana estão presas, e as outras duas estão livres. [Sug.: Quando a extremidade está livre, a derivada de u(x, y, t) deve se anular.]

b) Obtenha os modos normais e a solução geral obedecendo condições iniciais genéricas.

c) Faça a animação dos modos normais.

d) Faça uma análise da degenerescência.

e) Faça a animação com diversas condições iniciais.

4 Membrana Circular

4.1 Resolução da Equação de Onda

A membrana circular de raio a presa na borda é caracterizada pela seguinte condição de contorno em coordenadas cilíndricas: $\mathrm{u}(a,\,\theta,\,t)=0$, para $0\le\theta<2\,\pi$. Vamos atribuir a equação diferencial deste problema à variável *wave_eq*:

```
> restart;
> protect('r,theta,t,m,n'); # protect reserved variables
> alias(u=u(r,theta,t)):
> wave_eq := linalg[laplacian](u,[r,theta,z],coords=cylindrical)=Diff(u,t,t)/c^2;
```

$$wave\_eq := \frac{(\frac{\partial}{\partial r}\,u)+r\,(\frac{\partial^2}{\partial r^2}\,u)+\frac{\frac{\partial^2}{\partial\theta^2}\,u}{r}}{r}=\frac{\frac{\partial^2}{\partial t^2}\,u}{c^2}$$

Vamos resolver essa equação usando o método de separação de variáveis. Substituindo $\mathrm{u}(r,\,\theta,\,t)=\mathrm{R}(r)\,\Theta(\theta)\,\mathrm{T}(t)$ na variável *wave_eq* obtemos:

```
> Eq1 := expand(value(subs(u=R(r)*Theta(theta)*T(t), wave_eq)/(R(r)*Theta(theta)*T(t))));
```

$$Eq1 := \frac{\frac{\partial}{\partial r}\,\mathrm{R}(r)}{\mathrm{R}(r)\,r}+\frac{\frac{\partial^2}{\partial r^2}\,\mathrm{R}(r)}{\mathrm{R}(r)}+\frac{\frac{\partial^2}{\partial\theta^2}\,\Theta(\theta)}{\Theta(\theta)\,r^2}=\frac{\frac{\partial^2}{\partial t^2}\,\mathrm{T}(t)}{\mathrm{T}(t)\,c^2}$$

Vamos fazer agora a separação de variáveis:

```
> Eqt := expand(T(t)*c^2*( rhs(Eq1)=lambda ));
```

$$Eqt := \frac{\partial^2}{\partial t^2}\,\mathrm{T}(t)=\mathrm{T}(t)\,c^2\,\lambda$$

```
> Eqtheta := diff(Theta(theta),theta,theta) = lambda1*Theta(theta);
```

$$Eqtheta := \frac{\partial^2}{\partial\theta^2}\,\Theta(\theta)=\lambda 1\,\Theta(\theta)$$

```
> Eqr := simplify(subs(Eqt, Eqtheta, R(r)*Eq1));
```

$$Eqr := \frac{(\frac{\partial}{\partial r}\,\mathrm{R}(r))\,r+(\frac{\partial^2}{\partial r^2}\,\mathrm{R}(r))\,r^2+\lambda 1\,\mathrm{R}(r)}{r^2}=\mathrm{R}(r)\,\lambda$$

A condição de contorno $\mathrm{u}(a,\,\theta,\,t)=0$ impõe que a constante λ seja negativa pois quando ela é positiva ou nula, a equação *Eqr* não adimite soluções que se anulem para $r=a$. Além disso, a variável θ é periódica. Assim, vamos assumir que:

```
> lambda := -k(m,n)^2;
```

$$\lambda := -\mathrm{k}(m,\,n)^2$$

```
> lambda1 := -m^2;
```

$$\lambda 1 := -m^2$$

A solução geral para $\mathrm{R}(r)$ é:

```
> alias(J=BesselJ, Y=BesselY):
> sol_R := dsolve(Eqr,R(r));
```

$$sol\_R := \mathrm{R}(r)=\_C1\,\mathrm{J}(m,\,\mathrm{k}(m,\,n)\,r)+\_C2\,\mathrm{Y}(m,\,\mathrm{k}(m,\,n)\,r)$$

Como a função $\mathrm{Y}(m,\,\mathrm{k}(m,\,n)\,r)$ não é analítica em $r=0$, essa parte da solução deve ser rejeitada:

```
> assign(eval(subs(Y=0,_C1=1,_C2=1,sol_R)));
> R(r);
```

$$\mathrm{J}(m,\,\mathrm{k}(m,\,n)\,r)$$

A solução geral para $\Theta(\theta)$ é:

```
> dsolve(Eqtheta,Theta(theta));
> assign(");
```

$$\Theta(\theta)=\_C1\,\cos(m\,\theta)+\_C2\,\sin(m\,\theta)$$

```
> subs_set := {eval(subs(cos=1,sin=0,Theta(theta))) = C1,
> eval(subs(cos=0,sin=1,Theta(theta))) = C2};
```

$$subs\_set := \{\_C1 = C1,\,\_C2 = C2\}$$

```
> Theta(theta) := subs(subs_set, Theta(theta));
```

$$\Theta(\theta) := C1\,\cos(m\,\theta)+C2\,\sin(m\,\theta)$$

A solução geral para T(t) é:

```
> sol_T := dsolve(Eqt,T(t));
```

$$sol\_T := \mathrm{T}(t) = \_C1\cos(\mathrm{k}(m,\,n)\,c\,t) + \_C2\sin(\mathrm{k}(m,\,n)\,c\,t)$$

```
> C1 := eval(subs(sin=0,cos=1,rhs(sol_T)));
> C2 := eval(subs(cos=0,sin=1,rhs(sol_T)));
```

$$C1 := \_C1$$
$$C2 := \_C2$$

```
> expand(subs(C1=-I*C2,C2=I,convert(sol_T,exp)) );
> assign(");
```

$$\mathrm{T}(t) = e^{(I\,\mathrm{k}(m,\,n)\,c\,t)}$$

A título de segurança vamos limpar as variáveis que não serão mais usadas:

```
> unassign('C1,C2,sol_T,sol_R,wave_eq,Eq1,Eqr,E qtheta,Eqt,lambda,lambda1');
```

4.2 Cálculo dos Modos Normais

Vamos agora analisar os modos normais da membrana circular. Os modos com $m = 0$ não dependem da variável θ, isto é, eles são axialmente simétricos. Os modos com $m \neq 0$ dependem da variável θ e em função disto veremos exemplos tanto com $m = 0$ e com $m \neq 0$. A frequência do modo normal é:

```
> omega := (m,n) -> k(m,n)*c;
> protect('omega');
```

$$\omega := (m,\,n) \to \mathrm{k}(m,\,n)\,c$$

Para que as condições de contorno sejam satisfeitas, temos que impor que $\mathrm{R}(a) = 0$ ou seja $\mathrm{J}(m,\,\mathrm{k}(m,\,n)\,a) = 0$. Assim, $a\,\mathrm{k}(m,\,n)$ deve assumir valores que anulem a função de Bessel de ordem m:

```
> k := (m,n) -> ZerosBesselJ(m,n)/a;
> protect('k');
```

$$k := (m,\,n) \to \frac{\mathrm{j}(m,\,n)}{a}$$

A função ZerosBesselJ$(m,\,n)$ calcula o n-ésimo zero da função de Bessel de ordem m. No Apêndice A, apresentamos uma versão para esta função[11]. Por exemplo, o terceiro zero da função de Bessel de ordem 2 é:

```
> alias(j=ZerosBesselJ):
> j(2,3); # the result must be numerical
```

$$11.61984117$$

Vamos agora definir os modos normais como funçao de m e n. Se $m \neq 0$ existe uma degenerescência de quarta ordem, isto é, podemos definir os seguintes modos normais:

```
> normal_mode := unapply(R(r)*evalc(Re(T(t)))*subs(C1=1,C2=0,Theta(theta)),m,n);
```

$$normal\_mode := (m,\,n) \to \mathrm{J}(m,\,\mathrm{k}(m,\,n)\,r)\cos(\mathrm{k}(m,\,n)\,c\,t)\cos(m\,\theta)$$

```
> normal_mode1 := unapply(subs(C1=1,C2=0,R(r)*evalc(Im(T(t)))*Theta(theta)),m,n);
```

$$normal\_mode1 := (m,\,n) \to \mathrm{J}(m,\,\mathrm{k}(m,\,n)\,r)\sin(\mathrm{k}(m,\,n)\,c\,t)\cos(m\,\theta)$$

```
> normal_mode2 := unapply(subs(C1=0,C2=1,R(r)*evalc(Re(T(t)))*Theta(theta)),m,n);
```

$$normal\_mode2 := (m,\,n) \to \mathrm{J}(m,\,\mathrm{k}(m,\,n)\,r)\cos(\mathrm{k}(m,\,n)\,c\,t)\sin(m\,\theta)$$

```
> normal_mode3 := unapply(subs(C1=0,C2=1,R(r)*evalc(Im(T(t)))*Theta(theta)),m,n);
```

$$normal\_mode3 := (m,\,n) \to \mathrm{J}(m,\,\mathrm{k}(m,\,n)\,r)\sin(\mathrm{k}(m,\,n)\,c\,t)\sin(m\,\theta)$$

[11] Este procedimento deve ser carregado antes de continuar esta seção.

Os modos normais axialmente simétricos são os modos com $m = 0$. Neste caso, a degenerescência é dupla, já que os modos *normal_mode2* e *normal_mode3* acima não existem.

A solução geral é a superposição de todos os modos normais. A variável m pode assumir qualquer valor inteiro de 0 a ∞ e n de 1 a ∞. Porém, para as condições iniciais que estamos tratando aqui, a série de Fourier converge rapidamente, de forma que podemos truncar o somatório. As variáveis *p1* e *p2* especificam o truncamento em m e n respectivamente. Vamos separar os termos com $m = 0$ pois estes coeficientes seguem uma regra diferente dos coeficientes com m diferente de zero:

```
> alias(u=u):
> u := unapply(
> 2*Sum(factor(A1(0,n)*normal_mode(0,n) + A2(0,n)*normal_mode1(0,n)) +
> Sum(factor(A1(m,n)*normal_mode(m,n) + A2(m,n)*normal_mode1(m,n) +
> B1(m,n)*normal_mode2(m,n) + B2(m,n)*normal_mode3(m,n)),
> m=1..p1),n=1..p2), r,theta,t,p1,p2);
> protect('u');
```

$$u := (r,\, \theta,\, t,\, p1,\, p2) \to 2 \left(\sum_{n=1}^{p2} \left(\right.\right.$$

$$\mathrm{J}(0,\, \frac{\mathrm{j}(0,\, n)\, r}{a})\, (\mathrm{A1}(0,\, n) \cos(\frac{\mathrm{j}(0,\, n)\, c\, t}{a}) + \sin(\frac{\mathrm{j}(0,\, n)\, c\, t}{a})\, \mathrm{A2}(0,\, n)) + \left(\sum_{m=1}^{p1} \right.$$

$$\mathrm{J}(m,\, \frac{\mathrm{j}(m,\, n)\, r}{a})(\mathrm{A1}(m,\, n) \cos(\frac{\mathrm{j}(m,\, n)\, c\, t}{a}) \cos(m\, \theta) + \mathrm{A2}(m,\, n) \sin(\frac{\mathrm{j}(m,\, n)\, c\, t}{a}) \cos(m\, \theta)$$

$$\left. \left. + \mathrm{B1}(m,\, n) \cos(\frac{\mathrm{j}(m,\, n)\, c\, t}{a}) \sin(m\, \theta) + \sin(m\, \theta) \sin(\frac{\mathrm{j}(m,\, n)\, c\, t}{a})\, \mathrm{B2}(m,\, n)) \right) \right) \right)$$

Os coeficientes $\mathrm{A1}(m,\, n)$, $\mathrm{A2}(m,\, n)$, $\mathrm{B1}(m,\, n)$ e $\mathrm{B2}(m,\, n)$ são constantes que dependem de m e n. Quando as condições iniciais são axialmente simétricas, $\mathrm{u}(r,\, \theta,\, t)$ não depende de θ. Neste caso, temos que tomar $m = 0$, o que é equivalente a tomar $p1 = 0$. A expressão para u se reduz a:

```
> value(u(r,theta,t,0,p2));
```

$$2 \left(\sum_{n=1}^{p2} \mathrm{J}(0,\, \frac{\mathrm{j}(0,\, n)\, r}{a})\, (\mathrm{A1}(0,\, n) \cos(\frac{\mathrm{j}(0,\, n)\, c\, t}{a}) + \sin(\frac{\mathrm{j}(0,\, n)\, c\, t}{a})\, \mathrm{A2}(0,\, n)) \right)$$

Para condições iniciais genéricas:

$$u_0(r,\, \theta) = \mathrm{u}(r,\, \theta,\, 0)$$

$$v_0(r,\, \theta) = (\tfrac{\partial}{\partial t}\, \mathrm{u}(r,\, \theta,\, t)) \quad em \quad t \ = 0$$

onde $u_0(r,\, \theta)$ é a posição inicial e $v_0(r,\, \theta)$ a velocidade inicial da membrana, podemos determinar $\mathrm{A1}(m,\, n)$, $\mathrm{A2}(m,\, n)$, $\mathrm{B1}(m,\, n)$ e $\mathrm{B2}(m,\, n)$ por inversão da série dupla de Fourier cujos resultados são:

$$\mathrm{A1}(m,\, n) = \frac{\int_0^a \int_0^{2\,\pi} \mathrm{u0}(r,\, \theta) \cos(m\, \theta)\, r\, \mathrm{J}(m,\, \mathrm{k}(m,\, n)\, r)\, d\theta\, dr}{s\, a^2\, \mathrm{J}(m{+}1,\, \mathrm{k}(m,\, n)\, a)^2}$$

$$\mathrm{A2}(m,\, n) = \frac{\int_0^a \int_0^{2\,\pi} \mathrm{v0}(r,\, \theta) \cos(m\, \theta)\, r\, \mathrm{J}(m,\, \mathrm{k}(m,\, n)\, r)\, d\theta\, dr}{s\, \omega(m,\, n)\, a^2\, \mathrm{J}(m{+}1,\, \mathrm{k}(m,\, n)\, a)^2}$$

onde $s = 2$ para $m = 0$ e $s = 1$ para os outros casos. $\mathrm{B1}(m,\, n)$ e $\mathrm{B2}(m,\, n)$ são:

$$\mathrm{B1}(m,\, n) = \frac{\int_0^a \int_0^{2\,\pi} \mathrm{u0}(r,\, \theta) \sin(m\, \theta)\, r\, \mathrm{J}(m,\, \mathrm{k}(m,\, n)\, r)\, d\theta\, dr}{a^2\, \mathrm{J}(m{+}1,\, \mathrm{k}(m,\, n)\, a)^2}$$

$$\mathrm{B2}(m,\, n) = \frac{\int_0^a \int_0^{2\,\pi} \mathrm{v0}(r,\, \theta) \sin(m\, \theta)\, r\, \mathrm{J}(m,\, \mathrm{k}(m,\, n)\, r)\, d\theta\, dr}{\omega(m,\, n)\, a^2\, \mathrm{J}(m{+}1,\, \mathrm{k}(m,\, n)\, a)^2}$$

4.3 Animação dos Modos Normais

Vamos construir o procedimento *animate_mode* para a membrana circular da mesma forma que foi construído para a membrana retangular:

```
animate_mode := proc(m::nonnegint, n::posint)
    local FRAM, Options;
    global a, c, r, θ, t;
      if not type([a, c], list('numeric')) then
        ERROR(`The constants a and c must be numerical.`)
      elif not assigned(j) then ERROR(`The procedure ZerosBesselJ must be loaded.`)
      elif not assigned(ω) then ERROR(`The function omega must be defined.`)
      elif not assigned(normal_mode) then
        ERROR(`The function normal_mode must be defined.`)
      fi;
      if nargs = 1 then Options := 'frames' = 8, 'style = patch'; FRAM := 8
      else
        Options := args3 .. nargs ;
        if has({Options}, 'frames') then FRAM := subs({Options}, 'frames')
        else Options := Options, 'frames' = 8 ; FRAM := 8
        fi;
        if not has({Options}, 'style') then Options := Options, 'style = patch' fi
      fi;
      plots_animate3d([r, θ, normal_mode(m, n)], r = 0 .. a, θ = 0 .. 2 × π,
        t = 0 .. evalf(2 × π × (FRAM − 1)/(ω(m, n) × FRAM)), coords = cylindrical, Options)
    end
```

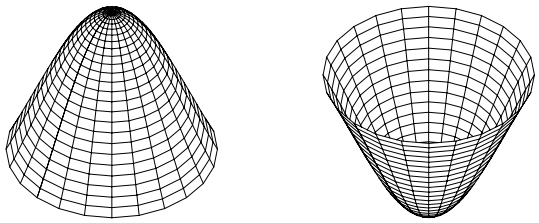
Para fazer a animação de um determinado modo normal, temos que dar valores numéricos para os parâmetros a e c. Vamos supor que o raio é a=1 e a constante c=1 em um certo sistema de unidades:

```
> a:=1; c:=1;
```

$$a := 1$$
$$c := 1$$

Primeiro, vamos analisar o movimento dos modos normais axialmente simétricos ($m = 0$). Vamos fazer a animação do modo <0,1> com as seguintes opções[12]:

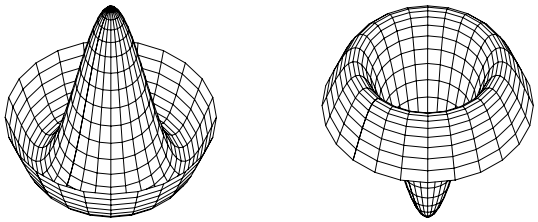
```
> animate_mode(0,1,frames=2,style=hidden,color= black,scaling=unconstrained);
```



[12] Para frisar as características do modo normal, estamos usando a opção *scaling=unconstrained*.

O primeiro modo axialmente simétrico não possui linha nodal, como era de se esperar. O segundo modo deve ter uma linha nodal:

```
> animate_mode(0,2,frames=2,style=hidden,color= black,scaling=unconstrained);
```

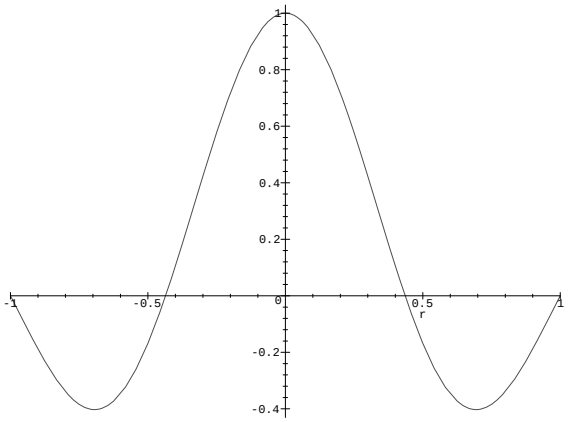


A linha nodal é um círculo cujo raio pode ser calculado da seguinte maneira. Observe que um corte vertical passando pelo centro do gráfico acima tem a seguinte forma:

```
> cut := eval( subs(t=0, normal_mode(0,2)) );
```

$$cut := \mathrm{J}(0,\, 5.520078110\, r)$$

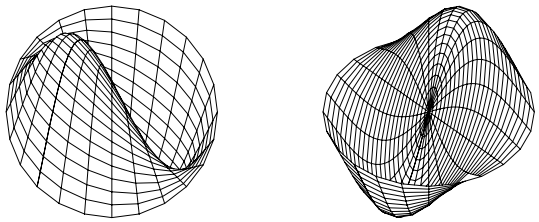
```
> plot(cut, r=-a..a);
```



A posição inicial do modo <0,2> é a figura formada pela rotação do gráfico acima em torno do eixo vertical. É fácil ver pelo gráfico que, de maneira geral, a primeira linha nodal do modo <0,n> tem o raio $r = \frac{a\,\mathrm{j}(0,\,1)}{\mathrm{j}(0,\,n)}$. No caso acima obtemos $r = .4356$.

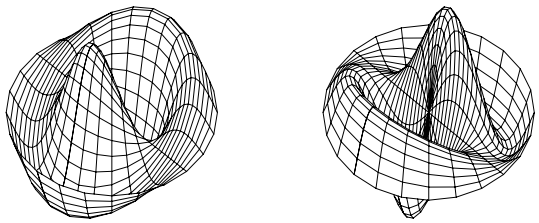
O primeiro modo normal não axialmente simétrico é o modo <1,1>:

```
> animate_mode(1,1,frames=2,style=hidden,color= black,scaling=unconstrained);
```



Neste caso, a linha nodal é uma reta passando pelo centro cujo ângulo é $\theta = \frac{\pi}{2}$. Os modos normais com $m \neq 0$ e $1 < n$ são de difícil visualização sem recursos de animação. Vejamos o modo <1,2>:

```
> animate_mode(1,2,frames=2,style=hidden,color= black,scaling=unconstrained);
```



De maneira geral, o modo $< m, n >$ tem m retas nodais passando pelo centro separadas entre si pelo ângulo $\theta = \frac{\pi}{m}$ e $n-1$ círculos de raios $r_p = \frac{a\,\mathrm{j}(0,p)}{\mathrm{j}(0,n)}$, onde p vai de 1 até $n-1$. As linhas nodais são mostradas pelo seguinte procedimento:

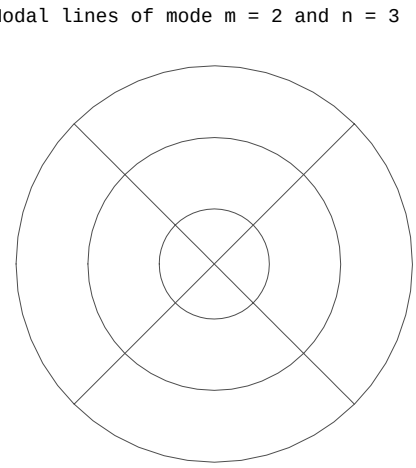
$nodal\_lines := \mathbf{proc}(m{::}posint,\ n{::}nonnegint)$
  $\mathbf{local} ci,\ sl,\ p;$
  $\mathbf{global} a,\ r;$
    $\mathbf{if}\ \mathbf{not}\ \mathrm{assigned}(j)\ \mathbf{then}\ \mathrm{ERROR}(\text{`}The\ procedure\ ZerosBesselJ\ must\ be\ loaded.\text{`})\ \mathbf{fi};$
    $ci_n := plots_{polarplot}(a,\ thickness = 2);$
    $\mathbf{for}\, p\, \mathbf{to}\, n-1\, \mathbf{do}\, ci_p := plots_{polarplot}(a \times \mathrm{j}(0,p)/\mathrm{j}(0,n))\, \mathbf{od};$
    $\mathbf{for}\, p\, \mathbf{to}\, m\, \mathbf{do}\, sl_p := plots_{polarplot}([r,\ 1/2 \times (2 \times p - 1) \times \pi/m,\ r = -a\ ..\ a])\, \mathbf{od};$
    $plots_{display}(\{\mathrm{seq}(sl_p,\ p = 1\ ..\ m),\ \mathrm{seq}(ci_p,\ p = 1\ ..\ n)\},\ axes = none,$

$scaling = constrained,\ title = `Nodal\ lines\ of\ mode\ m\ =\ `.m.`\ and\ n\ =\ `.n)$

end

Por exemplo, o modo <2,3> tem as seguintes linhas nodais:

```
> nodal_lines(2,3);
```



Observe que o círculo mais externo não deve ser contado como uma linha nodal pois representa a borda da membrana.

4.4 Animação com Condições Iniciais

```
> # Unassign the reserved variables of this problem
> # a - radius of the membrane
> # c - the constant of wave equation
> # u0 - the initial displacement of the membrane
> # v0 - the initial velocity of the membrane
> # AB_simp - simplifying function
> unassign('u0,v0,a,c,AB_simp');
```

As constantes A1(m, n), A2(m, n), B1(m, n) e B2(m, n) que aparecem na função u definida anteriormente podem ser obtidas a partir dos seguintes procedimentos baseados nas expressões teóricas previamente estabelecidas:

```
> A1:=proc(m,n) global u0,v0,a,c;
> `A1/A1uv`(m,n,eval(u0),a,c,eval(AB_simp))
> end:
> `A1/A1uv` := proc(m,n,u0,a,c,AB_simp)
> local s,r,theta;
> options remember;
> if m=0 then s:=2 else s:=1 fi;
> AB_simp(INT(r*J(m,j(m,n)/a*r)*INT(u0(r,theta) *
> cos(m*theta),theta=0..2*Pi),r=0..a)/(s*Pi*a^2 *J(m+1,j(m,n))^2));
> end:
> A2:=proc(m,n) global u0,v0,a,c;
> `A2/A2uv`(m,n,eval(v0),a,c,eval(AB_simp))
> end:
> `A2/A2uv` := proc(m,n,v0,a,c,AB_simp)
> local s,r,theta;
> options remember;
> if m=0 then s:=2 else s:=1 fi;
> AB_simp(INT(r*J(m,k(m,n)*r)*INT(v0(r,theta)*
> cos(m*theta),theta=0..2*Pi),r=0..a)/
```

```
> (s*omega(m,n)*Pi*a^2*(J(m+1,k(m,n)*a))^2));
> end:
> B1:=proc(m,n) global u0,v0,a,c;
> `B1/B1uv`(m,n,eval(u0),a,c,eval(AB_simp))
> end:
> `B1/B1uv` := proc(m,n,u0,a,c,AB_simp) local r,theta;
> options remember;
> AB_simp(INT(r*J(m,j(m,n)/a*r)*INT(u0(r,theta) *
> sin(m*theta),theta=0..2*Pi),r=0..a)/(Pi*a^2*J (m+1,j(m,n))^2));
> end:
> B2:=proc(m,n) global u0,v0,a,c;
> `B2/B2uv`(m,n,eval(v0),a,c,eval(AB_simp))
> end:
> `B2/B2uv` := proc(m,n,v0,a,c,AB_simp)
> local r,theta;
> options remember;
> AB_simp(INT(r*J(m,k(m,n)*r)*INT(v0(r,theta)*
> sin(m*theta),theta=0..2*Pi),r=0..a)/
> (omega(m,n)*Pi*a^2*(J(m+1,k(m,n)*a))^2));
> end:
> protect('A1,A2,B1,B2'):
```

Abaixo segue a definição da função INT[13]:

```
> INT:= proc() subs('int'='INT',int(args)) end:
> `evalf/INT` := proc()
> local err;
> err:=traperror(readlib(`evalf/int`)(args));
> if lasterror=`function does not evaluate to numeric` then 'INT'(args)
> elif err=lasterror then ERROR(lasterror)
> else subs('int'='INT',err)
> fi
> end:
> `print/INT` := proc() Int(args) end:
> protect('INT'):
```

Vamos usar a seguinte função para simplificar os coeficientes:

```
> AB_simp := x -> simplify(x);
```

$$AB\_simp := simplify$$

[13] Em vez de usar a função *int* usual para integração, estamos usando a função INT que contorna um *bug* do procedimento *'evalf/int'*. Este procedimento lida com integração numérica e na presente versão retorna uma mensagem de erro numa situação onde não deveria. Mostramos a seguir dois exemplos onde este fato ocorre:

```
> simplify(0.1*int(f(r),r=1..2));
Error, (in evalf/int) function does not evaluate to numeric
> sum(0.1*'i'*int(f(r),r=1..2),'i'=1..2);
Error, (in evalf/int) function does not evaluate to numeric
```

Este *bug* já foi reportado para os construtores do Maple que se protificaram em fixar em futuras versões. O procedimento INT e os procedimentos associados usam a função *traperror* para contornar o erro e retornar a integral não avaliada quando não for possível realizar a integração numérica. Para as integrais que aparecem nos exemplos analisados neste trabalho, o procedimento INT resolveu satisfatoriamente o problema.

O procedimento a seguir faz a animação do movimento da membrana descrita pela função u(r, θ, t, $p1$, $p2$). O primeiro argumento do procedimento representa o número de oscilações dominantes, cuja frequência é a do modo mais baixo ($n = 1$). O segundo argumento é o número $p1$ que representa a truncagem do somatório na variável m e o terceiro argumento é o número $p2$ que representa a truncagem do somatório na variável n:

```
animate_ic := proc(nt::numeric, p1::nonnegint, p2::posint)
    local FRAM, Options;
    global a, c, r, θ, t;
      if nargs < 3 then ERROR(`This procedure has at least 3 arguments. The first one \
        is the number of dominant oscilations. The second and the third are truncation val\
        ues of the double sum.`)
      fi;
      if not type([a, c], list(numeric)) then ERROR(`The constants a and c must be numeric.`)
      elif not assigned(j) then ERROR(`The procedure ZerosBesselJ must be loaded.`)
      elif not assigned(ω) then ERROR(`The function omega must be defined.`)
      elif not assigned(u) then ERROR(`The function normal_mode must be defined.`)
      fi;
      if not type(8 × nt, 'posint') then ERROR(`The first argument is a integer number r\
        epresenting the number of oscillations of the membrane.`)
      fi;
      if nargs = 3 then FRAM := 8 × nt ; Options := 'frames' = FRAM, 'style = patch'
      else
        Options := args4 .. nargs ;
        if has({Options}, 'frames') then FRAM := subs({Options}, 'frames')
        else FRAM := 8 × nt ; Options := 'frames' = FRAM
        fi
      fi;
      if not has({Options}, 'style') then Options := Options, 'style = patch' fi ;
      plots animate3d([r, θ, value(u(r, θ, t, p1, p2))], r = 0 .. a, θ = 0 .. 2 × π,
        t = 0 .. evalf(2 × π × nt/ω(0, 1)), coords = cylindrical, color = black, Options)
    end
```

Vamos ver dois exemplos com condições iniciais axialmente simétricas. O primeiro com velocidade inicial nula e o segundo com delocamento inicial nulo. Por motivo de simplificação, vamos calcular os coeficientes $A1$, $A2$, $B1$ e $B2$ explicitamente antes de fazer a animação, para as seguintes condições iniciais:

```
> u0 := (r,theta) -> 1;
> v0 := (r,theta) -> 0;
```

$$u0 := 1$$
$$v0 := 0$$

Ou seja, a membrana está repouso na posição $u = 1$. Tomaremos o raio da membrana igual a 5:

```
> a := 5; c := 1;
```

$$a := 5$$
$$c := 1$$

Os coeficientes A1(0, n) e A2(0, n) devem ser calculados separadamente de A1(m, n) e A2(m, n), pois estes últimos não se reduzem aos primeiros quando $m = 0$. Assim:

```
> unprotect('m');
> assume(m,integer);
> protect('m');
> A1(0,n);
```

$$\frac{1}{\mathrm{J}(1,\, \mathrm{j}(0,\, n))\, \mathrm{j}(0,\, n)}$$

```
> A1(m,n);
```

$$0$$

```
> A2(0,n);
```

$$0$$

```
> A2(m,n);
```

$$0$$

```
> B1(m,n);
```

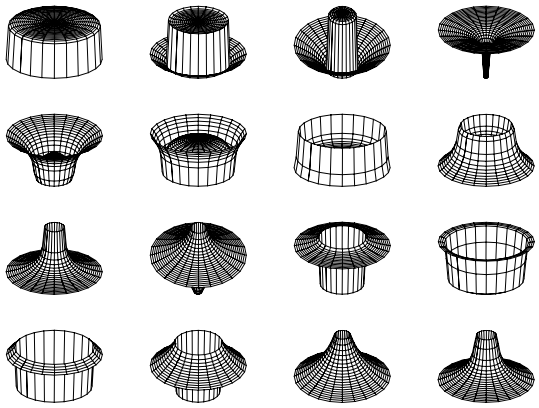
$$0$$

```
> B2(m,n);
```

$$0$$

Vamos fazer a animação com duas oscilações dominantes e com $p2 = 50$. Uma vez que a condição inicial é axialmente simétrica, vamos tomar $p1 = 0$:

```
> animate_ic(2,0,50,style=hidden,color=black);
```



Podemos ver pela animação que a onda reflete na borda da membrana e se superpõe no centro, produzindo um propagação típica de ondas, como deveria ser. A reflexão na borda inverte o deslocamento da onda já que a extremidade é fixa.

Vamos fazer a animação do movimento da membrana no caso em que ela está na posição horizontal e sofre um impulso de cima para baixo na região central. As condições iniciais são:

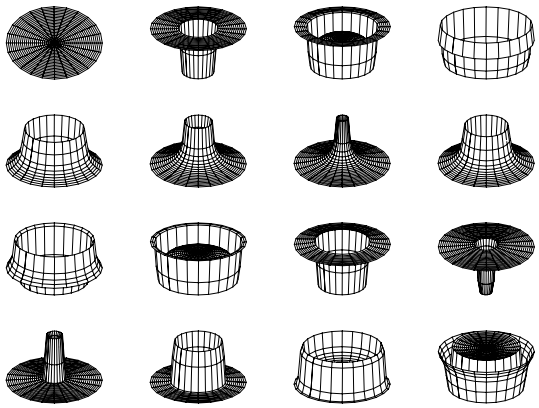
```
> u0 := (r,theta)-> 0;
> v0 := (r,theta)-> - piecewise(r<a/20,1,0);
```

$$u0 := 0$$

$$v0 := (r, \theta) \rightarrow -\text{piecewise}(r < \frac{1}{20}\, a,\ 1,\ 0)$$

Vamos fazer a animação com duas oscilações dominates, $p1 = 0$ e $p2 = 50$:

```
> animate_ic(2,0,50,style=hidden,color=black);
```



Vamos ver um exemplo onde as condições iniciais dependem de θ:

```
> u0 := (r,theta) -> piecewise(theta<Pi,1,0);
> v0 := (r,theta) -> 0;
```

$$u0 := (r, \theta) \rightarrow \text{piecewise}(\theta < \pi, 1, 0)$$

$$v0 := 0$$

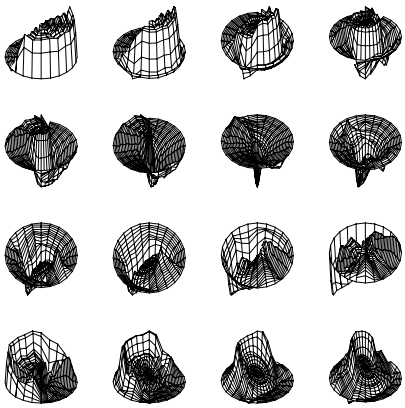
Estas condições iniciais representam uma membrana que teve metade levantada de uma unidade, como podemos ver pelo seguinte gráfico da função $u0$:

```
> plots[cylinderplot]([r,theta,u0(r,theta)],r=0 ..a,theta=0..2*Pi,style=hidden,
> color=black,scaling=constrained,orientation=[ -35,70]);
```

Plot: membto19.eps

Vamos fazer a animação com uma oscilação dominate, $p1 = 10$ e $p2 = 10$:

```
> animate_ic(1,10,10,frames=16,style=hidden,col or=black);
```



4.5 Exercícios

1) Os modos normais e o movimento com condições iniciais axialmente simétricas podem ser compreendidos a partir de gráficos bidimensionais de um corte vertical que passe pelo centro da membrana.

a) Faça a animação bidimensional dos modos normais axialmente simétricos.

b) Na animação, são mostrados em sequência vários gráficos de valores sucessivos da variável t. Neste ítem, exiba os gráficos do ítem anterior simultaneamente escolhendo uma maneira de distinguí-los, que pode ser por cor, por tonalidade ou por diferente tipos de tracejados.

c) Faça a animação bidimensional do movimento da membrana decorrente de condições iniciais que não dependam do ângulo θ, depois repita o segundo ítem para esta nova situação.

2) a) Faça o gráfico da variação da altura do ponto central da membrana contra t para diversas condições iniciais com simetria axial e sem esta simetria com velocidade inicial nula. A altura máxima de cada ciclo pode ultrapassar a altura inicial? Por quê?

b) Repita o ítem anterior para os modos normais. Quais são as diferenças com relação ao caso anterior. A altura máxima pode ultrapassar a altura inicial? Por quê?

c) Faça o gráfico da variação do volume da região sob a membrana contra t. Dê exemplos para diversos modos normais e diversos movimentos com condições iniciais com velocidade inicial nula. O volume máximo de cada ciclo pode ultrapassar o volume inicial? Por quê? Qual é a relação do volume com a energia potencial da membrana?

5 Membrana em forma de Anel Circular

5.1 Resolução da Equação de Onda

Uma membrana em forma de anel circular engastada ao longo de todo o seu contorno interno e externo deve ter o deslocamento vertical u(r, θ, t) satisfazendo a equação de onda:

$$\frac{\frac{\partial}{\partial r}u}{r} + (\frac{\partial^2}{\partial r^2}u) + \frac{\frac{\partial^2}{\partial \theta^2}u}{r^2} = \frac{\frac{\partial^2}{\partial t^2}u}{c^2}$$

Como o seu contorno permanece imóvel, o deslocamento deve satisfazer as seguintes condições de contorno:

$$\mathrm{u}(a,\, \theta,\, t) = 0$$
$$\mathrm{u}(b,\, \theta,\, t) = 0$$

onde a denomina o raio interno e b o raio externo da membrana. Utilizando o método de separação de variáveis encontramos as mesmas soluções para $\Theta(\theta)$, R(r) e T(t) da membrana circular, que são:

```
> restart;
> Theta := theta -> C1*cos(m*theta) + C2*sin(m*theta);
```

$$\Theta := \theta \rightarrow C1\cos(m\,\theta) + C2\sin(m\,\theta)$$

```
> alias(J=BesselJ,Y=BesselY):
> R := r -> C3*J(m,kappa*r) + C4*Y(m,kappa*r);
```

$$R := r \rightarrow C3\,\mathrm{J}(m,\, \kappa\, r) + C4\,\mathrm{Y}(m,\, \kappa\, r)$$

```
> T := t -> exp(I*kappa*c*t);
```

$$T := t \rightarrow e^{(I\,\kappa\,c\,t)}$$

onde *C1*, *C2*, *C3* e *C4* são constantes e m é um número inteiro não negativo. Para a membrana circular, a função Y(m, $\kappa\, r$) foi descartada porque a solução tinha que ser analítica na origem. Para este caso, esta função não deve ser descartada pois $r = 0$ não pertence ao domínio físico da membrana. Ela inclusive é necessária para se encontrar os valores das constantes *C3* e *C4* de modo que as soluções satisfaçam as condições de contorno. De fato, as condições de contorno impõem que R(a) = 0 e R(b) = 0:

```
> eqs_C3C4 := {subs(r=a,R(r))=0,subs(r=b,R(r))=0};
```

$$eqs\_C3C4 := \{C3\,\mathrm{J}(m,\, \kappa\, a) + C4\,\mathrm{Y}(m,\, \kappa\, a) = 0,\ C3\,\mathrm{J}(m,\, \kappa\, b) + C4\,\mathrm{Y}(m,\, \kappa\, b) = 0\}$$

O sistema de equações acima só possue uma solução não trivial para *C3* e *C4* se o determinante do sistema for nulo:

```
> matrix_JY := linalg[genmatrix](eqs_C3C4, {C3,C4});
```

$$matrix\_JY := \begin{bmatrix} \mathrm{J}(m,\, \kappa\, a) & \mathrm{Y}(m,\, \kappa\, a) \\ \mathrm{J}(m,\, \kappa\, b) & \mathrm{Y}(m,\, \kappa\, b) \end{bmatrix}$$

```
> eq_k := linalg[det](matrix_JY)=0;
```

$$eq\_k := \mathrm{J}(m,\, \kappa\, a)\,\mathrm{Y}(m,\, \kappa\, b) - \mathrm{Y}(m,\, \kappa\, a)\,\mathrm{J}(m,\, \kappa\, b) = 0$$

Para que a equação *eq_k* seja satizfeita, κ deve ser a n-ésima raiz da seguinte função que chamaremos de $BesselJY_m(x)$:

```
> BesselJY := (m,x) -> J(m,x*a)*Y(m,x*b)-Y(m,x*a)*J(m,x*b);
```

$$BesselJY := (m,\, x) \rightarrow \mathrm{J}(m,\, x\, a)\,\mathrm{Y}(m,\, x\, b) - \mathrm{Y}(m,\, x\, a)\,\mathrm{J}(m,\, x\, b)$$

As raízes da função $BesselJY_m(x)$ podem ser encontradas através do procedimento *ZerosBesselJY* desenvolvido no apêndice B[14]. Assim, vamos definir k(m, n) como sendo ZerosBesselJY(m, n) e em seguida substituir κ por k:

```
> alias(k=ZerosBesselJY):
```

É fácil ver que a escolha $C3 = \mathrm{Y}(m,\, \kappa\, a)$ e $C4 = -\mathrm{J}(m,\, \kappa\, a)$ satisfaz as equações *eqs_C3C4*. Sem perda de generalidade, podemos redefinir R(r) da seguinte forma:

```
> R := unapply( subs(C3=Y(m,kappa*a), C4=-J(m,kappa*a), kappa=k(m,n), R(r)), r );
```

$$R := r \rightarrow \mathrm{Y}(m,\, \mathrm{k}(m,\, n)\, a)\,\mathrm{J}(m,\, \mathrm{k}(m,\, n)\, r) - \mathrm{J}(m,\, \mathrm{k}(m,\, n)\, a)\,\mathrm{Y}(m,\, \mathrm{k}(m,\, n)\, r)$$

Da mesma forma, vamos redefinir T(t):

```
> T := unapply( subs(kappa=k(m,n), T(t)), t);
```

$$T := t \rightarrow e^{(I\,\mathrm{k}(m,\, n)\,c\,t)}$$

Vamos limpar as variáveis que não serão mais usadas e proteger as variáveis pertinentes:

```
> unassign('eqs_C3C4,matrix_JY,eq_k,wave_eq');
> protect('R,Theta,T,r,theta,t,m,n');
```

[14] Este procedimento e o procedimento *ZerosBesselJ* devem ser carregados antes de continuar esta seção.

5.2 Animação dos Modos Normais

Vamos definir a frequência como sendo:

```
> omega := (m,n) -> k(m,n)*c;
> protect('omega'):
```

$$\omega := (m,\, n) \to \mathrm{k}(m,\, n)\, c$$

Semelhante a membrana circular, aqui existe também uma degenerescência de quarta ordem, de forma que podemos definir quatro modos normais:

```
> normal_mode := unapply(subs(C1=1,C2=0,-R(r)*Theta(theta)*evalc(Re(T(t)))),m,n);
```

$$\begin{aligned} \mathit{normal\_mode} := (m,\, n) \to {} & - \\ & (\mathrm{Y}(m,\, \mathrm{k}(m,\, n)\, a)\, \mathrm{J}(m,\, \mathrm{k}(m,\, n)\, r) - \mathrm{J}(m,\, \mathrm{k}(m,\, n)\, a)\, \mathrm{Y}(m,\, \mathrm{k}(m,\, n)\, r))\cos(m\,\theta) \\ & \cos(\mathrm{k}(m,\, n)\, c\, t) \end{aligned}$$

```
> normal_mode1 := unapply(subs(C1=1,C2=0,-R(r)*evalc(Im(T(t)))*Theta(theta)),m,n);
```

$$\begin{aligned} \mathit{normal\_mode1} := (m,\, n) \to {} & - \\ & (\mathrm{Y}(m,\, \mathrm{k}(m,\, n)\, a)\, \mathrm{J}(m,\, \mathrm{k}(m,\, n)\, r) - \mathrm{J}(m,\, \mathrm{k}(m,\, n)\, a)\, \mathrm{Y}(m,\, \mathrm{k}(m,\, n)\, r))\sin(\mathrm{k}(m,\, n)\, c\, t)\cos(m\,\theta) \end{aligned}$$

```
> normal_mode2 := unapply(subs(C1=0,C2=1,-R(r)*Theta(theta)*evalc(Re(T(t)))),m,n);
```

$$\begin{aligned} \mathit{normal\_mode2} := (m,\, n) \to {} & - \\ & (\mathrm{Y}(m,\, \mathrm{k}(m,\, n)\, a)\, \mathrm{J}(m,\, \mathrm{k}(m,\, n)\, r) - \mathrm{J}(m,\, \mathrm{k}(m,\, n)\, a)\, \mathrm{Y}(m,\, \mathrm{k}(m,\, n)\, r))\sin(m\,\theta)\cos(\mathrm{k}(m,\, n)\, c\, t) \end{aligned}$$

```
> normal_mode3 := unapply(subs(C1=0,C2=1,-R(r)*Theta(theta)*evalc(Im(T(t)))),m,n);
```

$$\begin{aligned} \mathit{normal\_mode3} := (m,\, n) \to {} & - \\ & (\mathrm{Y}(m,\, \mathrm{k}(m,\, n)\, a)\, \mathrm{J}(m,\, \mathrm{k}(m,\, n)\, r) - \mathrm{J}(m,\, \mathrm{k}(m,\, n)\, a)\, \mathrm{Y}(m,\, \mathrm{k}(m,\, n)\, r))\sin(\mathrm{k}(m,\, n)\, c\, t)\sin(m\,\theta) \end{aligned}$$

Vamos construir o procedimento *animate_mode* para a membrana em forma de anel da mesma forma que foi construido para a membrana retangular e para a membrana circular:

$$\begin{aligned} & \mathit{animate\_mode} := \mathbf{proc}(m{::}\mathit{nonnegint},\, n{::}\mathit{posint}) \\ & \quad \mathbf{local}\, \mathit{FRAM},\, \mathit{Options}; \\ & \quad \mathbf{global}\, a,\, b,\, c,\, r,\, \theta,\, t; \\ & \quad\quad \mathbf{if\ not}\ \mathrm{type}([a,\, b,\, c],\, \mathrm{list}(\mathit{'numeric'}))\ \mathbf{then} \\ & \quad\quad\quad \mathrm{ERROR}(\text{`}\mathit{The\ constants\ a,\ b\ and\ c\ must\ be\ numeric.}\text{`}) \\ & \quad\quad \mathbf{elif\ not}\ \mathrm{assigned}(k)\ \mathbf{then}\ \mathrm{ERROR}(\text{`}\mathit{The\ procedure\ ZerosBesselJY\ must\ be\ loaded.}\text{`}) \\ & \quad\quad \mathbf{elif\ not}\ \mathrm{assigned}(\omega)\ \mathbf{then}\ \mathrm{ERROR}(\text{`}\mathit{The\ function\ omega\ must\ be\ defined.}\text{`}) \\ & \quad\quad \mathbf{elif\ not}\ \mathrm{assigned}(\mathit{normal\_mode})\ \mathbf{then} \\ & \quad\quad\quad \mathrm{ERROR}(\text{`}\mathit{The\ function\ normal\_mode\ must\ be\ defined.}\text{`}) \\ & \quad\quad \mathbf{fi}; \\ & \quad\quad \mathbf{if}\ \mathrm{nargs} = 1\ \mathbf{then}\ \mathit{Options} := \mathit{'frames'} = 8,\, \mathit{'style} = \mathit{patch'};\ \mathit{FRAM} := 8 \\ & \quad\quad \mathbf{else} \\ & \quad\quad\quad \mathit{Options} := \mathrm{args}_{3\ ..\ \mathrm{nargs}}; \\ & \quad\quad\quad \mathbf{if}\ \mathrm{has}(\{\mathit{Options}\},\, \mathit{'frames'})\ \mathbf{then}\ \mathit{FRAM} := \mathrm{subs}(\{\mathit{Options}\},\, \mathit{'frames'}) \\ & \quad\quad\quad \mathbf{else}\ \mathit{Options} := \mathit{Options},\, \mathit{'frames'} = 8;\ \mathit{FRAM} := 8 \\ & \quad\quad\quad \mathbf{fi}; \\ & \quad\quad\quad \mathbf{if\ not}\ \mathrm{has}(\{\mathit{Options}\},\, \mathit{'style'})\ \mathbf{then}\ \mathit{Options} := \mathit{Options},\, \mathit{'style} = \mathit{patch'}\ \mathbf{fi} \\ & \quad\quad \mathbf{fi}; \\ & \quad\quad \mathit{plots}_{\mathit{animate3d}}([r,\, \theta,\, \mathrm{normal\_mode}(m,\, n)],\, r = a\ ..\ b,\, \theta = 0\ ..\ 2\times\pi, \\ & \quad\quad\quad t = 0\ ..\ \mathrm{evalf}(2\times\pi\times(\mathit{FRAM}-1)/(\omega(m,\, n)\times\mathit{FRAM})),\, \mathit{coords} = \mathit{cylindrical},\, \mathit{Options}) \\ & \quad \mathbf{end} \end{aligned}$$

Vamos escolher valores numéricos para os parâmetros físicos:

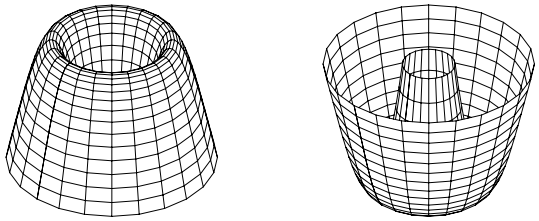
```
> a := 1; b := 4; c := 1;
```

$$\begin{aligned} a &:= 1 \\ b &:= 4 \end{aligned}$$

$$c := 1$$

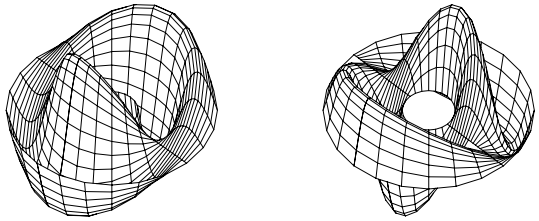
Os modos normais com $m = 0$ são axialmente simétricos, pois a dependência em θ desaparece. O modo normal mais simples com simetria axial é[15]:

```
> animate_mode(0,1,style=hidden,color=black,sca ling=unconstrained);
```



Vejamos agora um exemplo de um modo com $m \neq 0$:

```
> animate_mode(1,2,style=hidden,color=black,sca ling=unconstrained);
```



Semelhante a membrana circular, o modo $< m,\ n >$ tem m retas nodais separadas entre si pelo ângulo $\theta = \frac{\pi}{m}$ e $n - 1$ círculos. Os raios dos círculos nodais são determinados pelos zeros da equação $\mathrm{R}(r) = 0$ para $m = 0$ que são maiores que a e menores que b. A função zeros(n) encontra estes zeros para o modo $<m,n>$. O resultado é uma tabela:

$zeros := \mathbf{proc}(p::posint)$
  $\mathbf{local} z,\ A,\ B,\ R0,\ i;$
  $\mathbf{global} a,\ r,\ m,\ n;$
    $\mathbf{if}\ p = 1\ \mathbf{then}\ \mathrm{RETURN}(\mathrm{table}([]))\ \mathbf{fi};$

[15] Como já comentamos anteriormente, na versão impressa estamos mostrando a animação dos modos normais com apenas duas poses e com a opção $scaling = unconstrained$.

```
        if not assigned(R) then ERROR(`the function R(r) must be defined.`) fi;
        A := a;
        B := evalf(π/k(0, p));
        R0 := eval(subs(m = 0, n = p, eval(R(r))));
        for i to p − 1 do z_i := fsolve(R0, r = A .. A + B); A := z_i od;
        eval(z)
    end
```

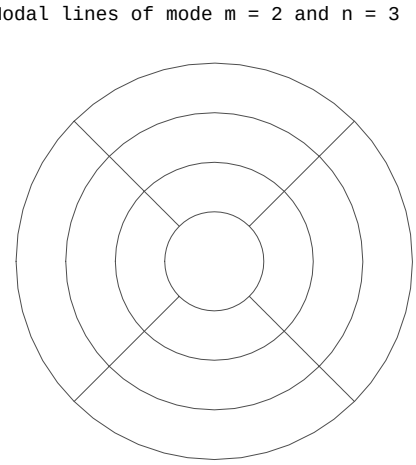
Para determinar os intervalos do comando *fsolve*, usamos a propriedade, que podemos verificar facilmente, de que os zeros de R(r) com $m = 0$ são entrelaçados com os zeros de J(0, k(0, n) r). Assim, podemos usar $\frac{\pi}{\mathrm{k}(0,\, n)}$ como tamanho do intervalo.

As linhas nodais são mostradas pelo seguinte procedimento:

```
nodal_lines := proc(m::nonnegint, n::posint)
    local ci, sl, p, radius;
      if not type(zeros, 'procedure') then ERROR(`the function zeros(n) must be defined`) fi;
      if not type([a, b], list('numeric')) then ERROR(`a and b must be numeric.`) fi;
      ci_0 := plots_polarplot(a, thickness = 2);
      ci_n := plots_polarplot(b, thickness = 2);
      radius := zeros(n);
      for p to n − 1 do ci_p := plots_polarplot(radius_p) od;
      sl_0 := NULL;
      for p to m do
        sl_p := plots_polarplot([r, 1/2 × (2 × p − 1) × π/m, r = a .. b]);
        sl_−p := plots_polarplot([r, 1/2 × (2 × p − 1) × π/m, r = −b .. −a])
      od;
      plots_display({seq(sl_p, p = −m .. m), seq(ci_p, p = 0 .. n)}, axes = none,
        scaling = constrained, title = `Nodal lines of mode m = `.m.` and n = `.n)
    end
```

Por exemplo, o modo <2,3> tem as seguintes linha nodais:

```
> a := 1:  b := 4:
> nodal_lines(2,3);
```



Observe que o círculo mais interno e o mais externo (em negrito) não contam como linha nodal, pois são a borda interna e a externa da membrana.

```
> unassign('a,b,c');
```

5.3 Animação com Condições Iniciais

A solução geral da equação de onda é a superposição de todos os modos normais com suas amplitudes ponderadas de acordo com as condições iniciais. Tendo em vista que computacionalmente não podemos fazer o somatório até infinito, será necessário truncá-lo. É claro que quanto maior for o valor do truncamento maior será a precisão, mas em compensação o tempo para gerar a animação também irá aumentar. Vamos truncar o somatório em *p1* e *p2* com relação a m e n respectivamente. Vamos separar os termos $\mathrm{A1}(0,\, n)$ e $\mathrm{A2}(0,\, n)$ pois estes coeficientes seguem uma regra diferente dos coeficientes correspondentes com m diferente de zero:

```
> u := unapply(Sum(factor(A1(0,n)*normal_mode(0,n) + A2(0,n)*normal_mode1(0,n)) +
> Sum(factor(A1(m,n)*normal_mode(m,n) + A2(m,n)*normal_mode1(m,n) +
> B1(m,n)*normal_mode2(m,n) + B2(m,n)*normal_mode3(m,n)), m=1..p1)
> ,n=1..p2),r,theta,t,p1,p2);
> protect('u');
```

$$
\begin{aligned}
u := (r,\, \theta,\, t,\, p1,\, p2) \rightarrow \sum_{n=1}^{p2} (& \\
& (-\mathrm{Y}(0,\, \mathrm{k}(0,\, n)\, a)\, \mathrm{J}(0,\, \mathrm{k}(0,\, n)\, r) + \mathrm{J}(0,\, \mathrm{k}(0,\, n)\, a)\, \mathrm{Y}(0,\, \mathrm{k}(0,\, n)\, r)) \\
& (\mathrm{A1}(0,\, n) \cos(\mathrm{k}(0,\, n)\, c\, t) + \mathrm{A2}(0,\, n) \sin(\mathrm{k}(0,\, n)\, c\, t)) + (\sum_{m=1}^{p1} (-(\\
& \mathrm{A1}(m,\, n) \cos(m\, \theta) \cos(\mathrm{k}(m,\, n)\, c\, t) + \mathrm{A2}(m,\, n) \sin(\mathrm{k}(m,\, n)\, c\, t) \cos(m\, \theta) \\
& + \mathrm{B1}(m,\, n) \sin(m\, \theta) \cos(\mathrm{k}(m,\, n)\, c\, t) + \mathrm{B2}(m,\, n) \sin(\mathrm{k}(m,\, n)\, c\, t) \sin(m\, \theta)) \\
& (\mathrm{Y}(m,\, \mathrm{k}(m,\, n)\, a)\, \mathrm{J}(m,\, \mathrm{k}(m,\, n)\, r) - \mathrm{J}(m,\, \mathrm{k}(m,\, n)\, a)\, \mathrm{Y}(m,\, \mathrm{k}(m,\, n)\, r)))))
\end{aligned}
$$

onde os coeficientes $\mathrm{A1}(m,\, n)$, $\mathrm{A2}(m,\, n)$, $\mathrm{B1}(m,\, n)$ e $\mathrm{B2}(m,\, n)$ podem ser encontrados em função das condições iniciais:

$$u_0(r,\, \theta) = \mathrm{u}(r,\, \theta,\, 0)$$

$$v_0(r,\, \theta) = (\tfrac{\partial}{\partial t}\, \mathrm{u}(r,\, \theta,\, t)) \;\; em \;\; t \;\; = 0$$

onde $u_0(r,\, \theta)$ é a posição inicial e $v_0(r,\, \theta)$ a velocidade inicial da membrana (em $t = 0$). Por inversão da série de Fourier, encontramos que $\mathrm{A1}(m,\, n)$ e $\mathrm{A2}(m,\, n)$ são dados por:

$$\mathrm{A1}(m,\, n) = -\frac{\displaystyle\int_a^b r\, \mathrm{R}(r) \int_0^{2\,\pi} u_0(r,\, \theta) \cos(m\, \theta)\, d\theta\, dr}{s\, \pi \displaystyle\int_a^b r\, \mathrm{R}(r)^2\, dr}$$

$$\mathrm{A2}(m,\, n) = -\frac{\displaystyle\int_a^b r\, \mathrm{R}(r) \int_0^{2\,\pi} v_0(r,\, \theta) \cos(m\, \theta)\, d\theta\, dr}{s\, \pi\, \omega(m,\, n) \displaystyle\int_a^b r\, \mathrm{R}(r)^2\, dr}$$

onde $s = 2$ se $m = 0$ e $s = 1$ caso contrário. $\mathrm{B1}(m,\, n)$ e $\mathrm{B2}(m,\, n)$ são dados por:

$$\mathrm{B1}(m,\, n) = -\frac{\displaystyle\int_a^b r\, \mathrm{R}(r) \int_0^{2\,\pi} u_0(r,\, \theta) \sin(m\, \theta)\, d\theta\, dr}{\pi \displaystyle\int_a^b r\, \mathrm{R}(r)^2\, dr}$$

$$\mathrm{B2}(m,\, n) = -\frac{\displaystyle\int_a^b r\, \mathrm{R}(r) \int_0^{2\,\pi} v_0(r,\, \theta) \sin(m\, \theta)\, d\theta\, dr}{\pi\, \omega(m,\, n) \displaystyle\int_a^b r\, \mathrm{R}(r)^2\, dr}$$

Os procedimentos abaixo calculam os coeficientes descritos pelas expressões acima:

```
> A1 := proc(m,n) global u0,v0,a,b,c;
> `A1/A1uv`(m,n,eval(u0),a,b,c,eval(AB_simp))
> end:
> `A1/A1uv` := proc(m,n,u0,a,b,c,AB_simp)
> local R,s;
> options remember;
> if m=0 then s:=2 else s:=1 fi;
> R:=Y(m,k(m,n)*a)*J(m,k(m,n)*r)-J(m,k(m,n)*a)* Y(m,k(m,n)*r);
> AB_simp(-1/s/Pi*INT(r*R*INT(u0(r,theta)*
> cos(m*theta),theta=0..2*Pi),r=a..b)/INT(r*R^2 ,r =a..b));
> end:
> A2 := proc(m,n) global u0,v0,a,b,c;
> `A2/A2uv`(m,n,eval(v0),a,b,c,eval(AB_simp))
> end:
> `A2/A2uv` := proc(m,n,u0,a,b,c,AB_simp)
> local R,s;
> options remember;
> if m=0 then s:=2 else s:=1 fi;
> R:=Y(m,k(m,n)*a)*J(m,k(m,n)*r)-J(m,k(m,n)*a)* Y(m,k(m,n)*r);
> AB_simp(-1/s/Pi/omega(m,n)*INT(r*R*INT(v0(r,t heta)*
> cos(m*theta),theta=0..2*Pi),r=a..b)/INT(r*R^2 ,r =a..b));
> end:
> B1 := proc(m,n) global u0,v0,a,b,c;
> `B1/B1uv`(m,n,eval(u0),a,b,c,eval(AB_simp))
> end:
> `B1/B1uv` := proc(m,n,u0,a,b,c,AB_simp)
> local R;
> options remember;
> R:=Y(m,k(m,n)*a)*J(m,k(m,n)*r)-J(m,k(m,n)*a)* Y(m,k(m,n)*r);
> AB_simp(-1/Pi*INT(r*R*INT(u0(r,theta)*
> sin(m*theta),theta=0..2*Pi),r=a..b)/INT(r*R^2 ,r =a..b));
> end:
> B2 := proc(m,n) global u0,v0,a,b,c;
> `B2/B2uv`(m,n,eval(v0),a,b,c,eval(AB_simp))
> end:
> `B2/B2uv` := proc(m,n,v0,a,b,c,AB_simp)
> local R;
> options remember;
> R:=Y(m,k(m,n)*a)*J(m,k(m,n)*r)-J(m,k(m,n)*a)* Y(m,k(m,n)*r);
> AB_simp(-1/Pi/omega(m,n)*INT(r*R*INT(v0(r,the ta)*
> sin(m*theta),theta=0..2*Pi),r=a..b)/INT(r*R^2 ,r =a..b));
> end:
> protect('A1,A2,B1,B2'):
```

O procedimento INT foi definido na seção anterior. Nos exemplos desta seção, aparecem integrais no denominador. Para que elas sejam simplificadas, vamos definir AB_simp como sendo:

```
> AB_simp := x -> normal(simplify(x),expanded);
```

$$AB\_simp := x \rightarrow \mathrm{normal}(\mathrm{simplify}(x),\ expanded)$$

O procedimento a seguir faz a animação da membrana:

```
animate_ic := proc(nt, p1::nonnegint, p2::posint)
    local FRAM, Options;
    global a, b, c, r, θ, t;
      if nargs < 3 then ERROR('This procedure has at least 2 arguments. The first one \
        is the number of oscilations. The second one is the truncation number of the doubl\
        e sum.')
      elif not type([a, b, c], list('numeric')) then
        ERROR('The constants a, b and c must be numeric.')
      elif not assigned(k) then ERROR('The procedure ZerosBesselJY must be loaded.')
      elif not assigned(ω) then ERROR('The function omega must be defined.')
      fi;
      if not type(8 × nt, posint) then ERROR('The first argument is a integer number re\
        presenting the number of oscillations of the membrane.')
      fi;
      if nargs = 3 then FRAM := 8 × nt ; Options := 'frames' = FRAM, 'style = patch'
      else
        Options := args4 .. nargs ;
        if has({Options}, 'frames') then FRAM := subs({Options}, 'frames')
        else FRAM := 8 × nt ; Options := 'frames' = FRAM
        fi
      fi;
      if not has({Options}, 'style') then Options := Options, 'style = patch' fi;
      plots animate3d([r, θ, value(u(r, θ, t, p1, p2))], r = a .. b, θ = 0 .. 2 × π,
        t = 0 .. evalf(2 × π × nt/ω(0, 1)), coords = cylindrical, Options)
    end
```

Vamos usar as seguintes condições iniciais:

```
> u0 := (r,theta) -> exp(-100*(r-1.2)^2);
```

$$u0 := (r, \theta) \rightarrow e^{(-100\,(r-1.2)^2)}$$

```
> v0 := (r,theta) -> 200*(r-1.2)*exp(-100*(r-1.2)^2);
```

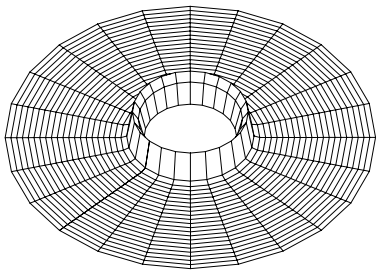
$$v0 := (r, \theta) \rightarrow 200\,(r-1.2)\,e^{(-100\,(r-1.2)^2)}$$

```
> a:=1; b:=4; c:=1;
```

$$a := 1$$
$$b := 4$$
$$c := 1$$

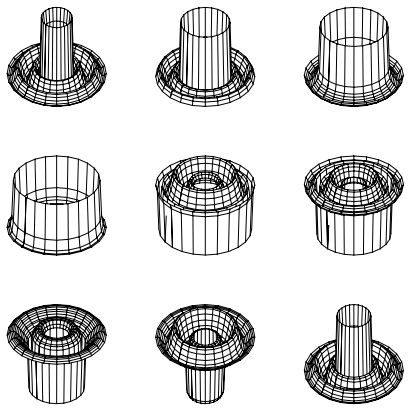
O gráfico da posição inicial é:

```
> with(plots):
> cylinderplot([r,theta,u0(r,theta)],r=a..b,the ta=0..2*Pi,scaling=constrained);
```



Estas condições iniciais correspondem a um pulso de onda que se propaga em direção ao raio externo da membrana. Vejamos a animação:

```
> unprotect('m');
> assume(m,integer);
> protect('m');
> animate_ic(1,0,30,frames=9,style=hidden,color =black);
```



Podemos ver que o pulso é refletido na borda exterior com inversão da amplitude, e se propaga em direção ao raio interno onde é novamente é refletido.

6 Conclusões e Observações Finais

Algumas observações de natureza física devem ser feitas. A equação de onda que estamos analisando é válida para pequenas oscilações. Alguns gráficos e animações que apresentamos baseavam-se em expressões que não obedecem esta restrição. Isto pode ser corrigido por uma constante multiplicativa suficientemente pequena. No entanto, tal cuidado só terá efeito se os gráficos forem feitos com a opção *constrained*.

A frequência de oscilação apresentada na animação não corresponde necessariamente a frequêcia física ω. A frequência da animação pode ser alterada clicando-se no botão de acelerar ou de retardar. Para se obter a frequêcia física real, devemos fixar um sistemas de unidades, dar os valores pertinentes para as constantes físicas, como as dimensões da membrana e a velocidade de onda c. A partir do valor da frequência ω, temos que calibrar a animação ajustando a taxa de poses (*frame rate*) que é fornecida na janela de animação para o valor correspondente a ω.

Nas seções de animação dos modos normais, escolhemos arbitrariamente um dos modos normais para animar e desconsideramos os modos normais degenerados. Porém eles podem ser facilmente compreendidos a partir do primeiro, pois eles são iguais a estes exceto por uma diferença no instante do início da animação, se a degenerescência for na variável t, ou por uma rotação de $\frac{\pi}{2}$, se a degenerescência for na variável θ (no caso da membranas circulares).

O Maple fornece um ambiente propício para a análise do problema de oscilação de membranas. As diferentes ferramentas que ele possui podem ser usadas de maneira combinada para a obtenção dos resultados. Este trabalho tem duas versões: a impressa e a virtual[16]. A versão virtual consiste nas *worksheets* onde foram desenvolvidos todos os cálculos e animações. Nesta versão é possível mudar parâmetros, executar novos exemplos e testar novas idéias. As animações podem ser melhor visualisadas, principalmente seus aspectos dinâmicos que são perdidos na versão impressa.

Este trabalho serve como um exemplo de como construir uma *worksheet* para resolução de um determinado problema. Foram tomadas uma série de precauções para evitar erros em manipulações futuras com a *worksheet*. Algumas variáveis que são usadas frequentemente foram protegidas com o comando *protect*. Os procedimentos têm várias mensagens de erro para orientar, caso um usuário faça algum uso indevido. Vale a pena tomar estas preucações pois alguns erros são de difícil detecção. Eles podem desanimar um usuário que pode desistir de usar as ferramentas discutidas aqui.

[16] As *worksheets* podem ser obtidas através de *ftp* anômino no endereço *anonymous@lca1.drp.cbpf.br* no diretório *pub/membrana*.

APÊNDICE A - Procedimento para Calcular os Zeros da Função de Bessel $J_m(x)$

O procedimento a seguir calcula os zeros da Função de Bessel $J_m(x)$. Ele foi obtido na MUG[17] de uma correspodência enviada por Dr. K.O. Geddes (4/junho/96):

```
ZerosBesselJ := proc(maxv, maxs)
    local j, incr, v, h, s;
      j := array(0 .. maxv, 1 .. maxs) ;
      incr := 4.0 ;
      for v from 0 to maxv do
        h := evalf(v + 1.9 × v^(1/3) + 1) ;
        if v = 0 then j[v,1] := fsolve(BesselJ(v, x), x, 2.0 .. 3.0)
        else j[v,1] := fsolve(BesselJ(v, x), x, 2.0 .. h)
        fi;
        for s from 2 to maxs do j[v,s] := fsolve(BesselJ(v, x), x, j[v,s-1] .. j[v,s-1] + incr) od
      od;
      RETURN(eval(j))
    end
```

Vejamos uma exemplo:

```
> set_time := time():
> ZerosBesselJ(2,3);
```

```
array(0 .. 2, 1 .. 3, [
    (0, 1) = 2.404825558
    (0, 2) = 5.520078110
    (0, 3) = 8.653727913
    (1, 1) = 3.831705970
    (1, 2) = 7.015586670
    (1, 3) = 10.17346814
    (2, 1) = 5.135622302
    (2, 2) = 8.417244140
    (2, 3) = 11.61984117
    ])
```

```
> time()-set_time;
```

5.442

Este procedimento ao ser chamado da forma $ZerosBesselJ(m,\, n)$ retorna um *array* de dimensão $n\,(m+1)$ com os n primeiros zeros da função de Bessel $J_\nu(x)$ para ν indo de 0 até m. Como foi observado por Y. Muzychka na MUG (6/junho/96), o procedimento é lento e o motivo é que ele calcula $n\,(m+1)$ zeros. Se um usuário deseja encontrar apenas um único zero de uma determinada ordem, haverá um cálculo desnecessário dos zeros da função de Bessel de outras ordens. Com uma modificação bastante simples, podemos fazer um outro procedimento que calcula apenas os zeros da função de Bessel de uma determinada ordem. Porém, não podemos evitar de maneira simples o cálculo dos $n-1$ primeiros zeros, pois como podemos observar pelo algoritmo do procedimento, o cálculo do n-ésimo zero necessita do cálculo do zero anterior que por sua vez necessita do cálculo do próximo zero anterior e assim por diante.

[17] *Maple User Group*: Trata-se de um grupo de usuários que têm seus E-mail cadastrados que trocam correspondências mediadas por um membro da *Waterloo Maple Software*. O endereço para registro e envio de correspondência é *maple-list@daisy.uwaterloo.ca.*

O procedimento modificado tem uma forma recursiva. Ele retorna apenas o n-ésimo zero de $J_m(x)$ e guarda os zeros menores na *Remember Table* (apenas para verificação):

```
ZerosBesselJ := proc(m, n::{name, nonnegint})
    local h, x;
    option remember;
      if not (type(m, numeric) and type(n, nonnegint)) then 'ZerosBesselJ(m, n)'
      elif n = 0 then if m = 0 then ERROR(`0 - th zero of Bessel(0,x) not defined`) else 0 fi
      elif n = 1 then
        if m = 0 then fsolve(BesselJ(0, x), x, 2.0 .. 3.0)
        else h := evalf(m + 1.9 × m^(1/3) + 1) ; fsolve(BesselJ(m, x), x, 2.0 .. h)
        fi
      else fsolve(BesselJ(m, x), x, ZerosBesselJ(m, n − 1) .. ZerosBesselJ(m, n − 1) + 4.0)
      fi
    end
```

Vejamos um exemplo:

```
> alias(j=ZerosBesselJ):
> alias(J=BesselJ):
> set_time := time():
> j(2,3); # terceiro zero de J2(x)
```

$$11.61984117$$

```
> time()-set_time;
```

$$.662$$

Os zeros anteriores tiveram que ser calculados e estão guardados na *Remember Table* do procedimento:

```
> op(4,eval(j));
```

$$\begin{aligned}&\text{table}([\\&\quad(2,\,1) = 5.135622302\\&\quad(2,\,3) = 11.61984117\\&\quad(2,\,2) = 8.417244140\\&\quad])\end{aligned}$$

O algoritmo deste procedimento, por ser recursivo, torna o cálculo da n-ésima raiz bastante demorado para grandes valores de n. Pior ainda: o algoritmo falha para valores de m (ordem da função de Bessel) igual ou acima de 11, pois a distância entre o primeiro e o segundo zero é maior que 4. Como o incremento escolhido no algoritmo foi de 4, o segundo zero não está no intervalo $[\mathrm{j}(m,\,n-1)\;..\;\mathrm{j}(m,\,n-1)+4.0]$. Nenhum valor fixo como incremento resolve o problema, pois a distância entre o primeiro e o segundo zero fica arbitrariamente grande quando a ordem da função de Bessel cresce.

Neste apêndice, nos propomos a corrigir estes dois problemas levantados. No que se segue, usaremos a variável m para descrever a ordem da função de Bessel (que pode assumir qualquer número real não negativo) e n para o n-ésimo zero. A recursividade do algoritmo pode ser removida para $m < \frac{5}{2}$. Isto se deve aos seguintes fatos demostrados por Schafheitlin[13]:

- Para $m \leq 1/2$ o n-ésimo zero de $J_m(x)$ está dentro do intervalo:

$$[(n + \tfrac{m}{2} - \tfrac{1}{4})\,\pi\;..\;(n + \tfrac{m}{4} - \tfrac{1}{8})\,\pi]$$

- Para $1/2 < m < 5/2$ o n-ésimo zero de $J_m(x)$ está dentro do intervalo:

$$[(n + \tfrac{m}{4} - \tfrac{1}{8})\,\pi\;..\;(n + \tfrac{m}{2} - \tfrac{1}{4})\,\pi]$$

Para $\frac{5}{2} \leq m$, a recursividade pode ser eliminada em parte devido ao seguinte resultado[14]:

- Para $1/2 < m$, os zeros de $J_m(x)$ que excedem $\frac{(2\,m+1)\,(2\,m+3)}{\pi}$ estão dentro de intervalos do tipo:

$$[(p + \tfrac{m}{2} + \tfrac{1}{2})\,\pi \;..\; (p + \tfrac{m}{2} + \tfrac{3}{4})\,\pi]$$

onde p assume valores inteiros consecutivos.

Os zeros menores que a razão $\frac{(2\,m+1)\,(2\,m+3)}{\pi}$ podem ser calculados por um algoritmo recursivo. Os zeros maiores que esta razão podem ser calculados usando-se o intervalo dado acima, porém o teorema de Schafheitlin não diz qual é a relação de p com n. Para se obter esta relação, será necessário saber o número de zeros no intervalo $[0 \;..\; \frac{(2\,m+1)\,(2\,m+3)}{\pi}]$. Neste caso, eles terão que ser calculados recursivamente.

O próximo problema é descobrir o valor adequado para o incremento no processo recursivo de cálculo. Para isto, precisamos conhecer a fórmula assintótica dos quatros primeiros zeros da função de Bessel. A fórmula assintótica do primeiro zero é[15]:

$$m + 1.8557571\,m^{(\frac{1}{3})} + 1.033150\,m^{(-\frac{1}{3})} + \mathrm{O}(m^{(-1)})\,.$$

Para se obter um intervalo onde esteja o primeiro zero, basta os dois primeiro termos da expressão acima, ou seja, precisamos da função $m + c_1\,m^{(\frac{1}{3})}$ onde c_1 tem o valor 1.8557571. O valor de c_1 foi obtido da seguinte maneira[8]: Sejam P(x, m) e Q(x, m) as seguintes funções:

```
> Digits:=5;
```

$$Digits := 5$$

```
> P := (x,m)->1/2/GAMMA(m+1/2)*Int(exp(-u)*u^(m-1/2)* evalc((1+I*u/2/x)^(m-1/2)+
> (1-I*u/2/x)^(m-1/2)),u=0..infinity,Digits,_De xp);
```

$$P := (x,\,m) \to \frac{1}{2}$$
$$\mathrm{Int}(e^{(-u)}\,u^{(m-1/2)}\,\mathrm{evalc}((1+\frac{1}{2}\,\frac{I\,u}{x})^{(m-1/2)} + (1-\frac{1}{2}\,\frac{I\,u}{x})^{(m-1/2)}),\,u = 0\;..\;\infty,\,Digits,\,\_Dexp)$$
$$\Big/\Gamma(m+\frac{1}{2})$$

```
> Q := (x,m) -> 1/2/GAMMA(m+1/2)/I*Int(exp(-u)*u^(m-1/2)* evalc((1+I*u/2/x)^(m-1/2)-
> (1-I*u/2/x)^(m-1/2)),u=0..infinity,Digits,_De xp);
```

$$Q := (x,\,m) \to -\frac{1}{2}I$$
$$\mathrm{Int}(e^{(-u)}\,u^{(m-1/2)}\,\mathrm{evalc}((1+\frac{1}{2}\,\frac{I\,u}{x})^{(m-1/2)} - (1-\frac{1}{2}\,\frac{I\,u}{x})^{(m-1/2)}),\,u = 0\;..\;\infty,\,Digits,\,\_Dexp)$$
$$\Big/\Gamma(m+\frac{1}{2})$$

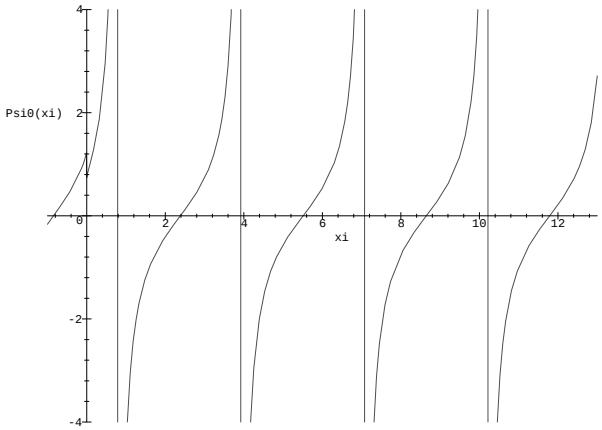
A partir de P e Q pode-se definir a seguinte função:

```
> Psi0 := xi -> tan(xi-3*Pi/4)+Q(xi,1/3)/P(xi,1/3);
```

$$\Psi 0 := \xi \to \tan(\xi - \frac{3}{4}\,\pi) + \frac{\mathrm{Q}(\xi,\,\frac{1}{3})}{\mathrm{P}(\xi,\,\frac{1}{3})}$$

cujo gráfico na região que contêm os 4 primeiros zeros é:

```
> plot('Psi0(xi)', xi=-1..13, 'Psi0(xi)'=-4..4);
```



Pelo gráfico podemos determinar os intervalos para achar os zeros com a função *fsolve*:

```
> map( rng -> fsolve('Psi0(xi)'=0,xi=rng), [2..3, 5..6, 8..9, 11..12]);
```

$$[2.3834, 5.5102, 8.6474, 11.787]$$

O primeiro zero positivo de $\Psi_0(\xi)$ está relacionado com c_1 através da seguinte seguinte expressão $\frac{(2\,c_1)^{(\frac{3}{2})}}{3}$. O valor do primeiro zero é aproximadamente 2.38, de forma que $c_1 = 1.85$. O segundo, terceiro e quarto zeros da função de Bessel também obedecem ao mesmo tipo de fórmula assintótica sendo que os coeficientes c_2, c_3 e c_4 são obtidos a partir do segundo, terceiro e quarto zeros da função $\Psi_0(\xi)$, análogo a forma pela qual foi obtido c_1. Os valores do zeros subsequentes de $\Psi_0(\xi)$ são aproximadamente 5.51, 8.65 e 11.8. Os valores correspondentes de c são $c_2 = 3.24$, $c_3 = 4.38$ e $c_4 = 5.39$, como podemos ver pelo cálculo abaixo:

```
> map(z -> evalf((3*z)^(2/3)/2), ");
```

$$[1.8558, 3.2447, 4.3817, 5.3868]$$

```
> Digits := 10;
```

O intervalo que contém o primeiro zero da função de Bessel e o que contém o segundo zero podem ser obtidos a partir das seguintes fórmulas assintóticas respectivamente:

```
> h1 := m -> evalf(m+1.85*m^(1/3));
```

$$h1 := m \rightarrow \text{evalf}(m + 1.85\, m^{1/3})$$

```
> h2 := m -> evalf(m+3.24*m^(1/3));
```

$$h2 := m \rightarrow \text{evalf}(m + 3.24\, m^{1/3})$$

O próximo zero pode ser obtido da seguinte maneira: Tomamos como incremento a distância entre o primeiro e o segundo zero. O intervalo do comando *fsolve* para o cálculo do terceiro zero pode ser:

$$\text{ZerosBesselJ}(m,\, 2)\ ..\ \text{ZerosBesselJ}(m,\, 2) + \mathit{incremento}.$$

Assim, temos que garantir que o terceiro zero está dentro deste intervalo e que o quarto zero não está. Definindo h_3 e h_4 da mesma forma que h_1 e h_2 com os valores de c_3 e c_4, é fácil verificar que $h_3(m) - h_2(m) < \mathit{incremento}$ e que $\mathit{incremento} < h_4(m) - h_2(m)$, onde $\mathit{incremento} = h_2(m) - h_1(m)$ como afirmamos anteriormente. Este argumento pode ser repetido para os próximos zeros pois quando $n \rightarrow \infty$ o *incremento* e a distância entre os zeros tendem a π.

Levando em conta estes dados, podemos escrever o procedimento *ZerosBesselJ* da seguinte forma:

ZerosBesselJ := **proc**(*m*::*algebraic*, *n*::*algebraic*, *fdig*)
if 3 < nargs **or** nargs < 2 **then** ERROR(‘*wrong number of arguments*‘) **fi**;
if $n = \infty$ **or** $n = -\infty$ **or not** type(*n*, *realcons*) **then** RETURN('procname(*m*, *n*)') **fi**;
if not type(*n*, *nonnegint*) **then** ERROR(‘*2nd argument must be a positive integer*‘) **fi**;

```
        if m = ∞ or m = -∞ or not type(m, realcons) then RETURN('procname(m, n)') fi;
        if evalf(m) < 0 then ERROR(`negative order not implemented`) fi;
        if n = 0 then
            if m = 0 then ERROR(`0 - th zero of Bessel(0, x) not defined`) else RETURN(0) fi
        fi;
        _EnvZerosBesselJ := 'false';
        _Envfdig := NULL;
        if nargs = 3 then
            if fdig = 'fulldigits' then _Envfdig := 'fulldigits'
            else ERROR(`3rd argument must be fulldigits`)
            fi
        fi;
        evalf('ZerosBesselJ'(m, n))
    end

evalf/ZerosBesselJ := proc(mm, n)
    local m, a, b, qmax, q1, q2, i, r, x;
        m := evalf(mm);
        if not type(m, 'numeric') then RETURN('ZerosBesselJ'(m, n)) fi;
        if n = ∞ or n = -∞ or not type(n, realcons) then RETURN('ZerosBesselJ'(m, n)) fi;
        if not type(n, nonnegint) then ERROR(`2nd argument must be a positive integer`) fi;
        if evalf(m - 1/2) < 0 then
            a := evalf((n + 1/2 × m - 1/4) × π);
            b := evalf((n + 1/4 × m - 1/8) × π);
            fsolve(BesselJ(m, x), x, a .. b, _Envfdig)
        elif evalf(m - .5) = 0 then evalf(n × π)
        elif evalf(m - 5/2) < 0 then
            a := evalf((n + 1/4 × m - 1/8) × π);
            b := evalf((n + 1/2 × m - 1/4) × π);
            fsolve(BesselJ(m, x), x, a .. b, _Envfdig)
        elif n = 1 then
            a := evalf(m + 1.85 × m^(1/3)); b := a + 1.; fsolve(BesselJ(m, x), x, a .. b, _Envfdig)
        elif n = 2 then
            a := evalf(m + 3.24 × m^(1/3)); b := a + 2.2; fsolve(BesselJ(m, x), x, a .. b, _Envfdig)
        elif _EnvZerosBesselJ then fsolve(BesselJ(m, x), x, _Enva .. _Envb, _Envfdig)
        else
            qmax := evalf((2 × m + 1) × (2 × m + 3)/π);
            q1 := evalf('ZerosBesselJ'(m, 1));
            q2 := evalf('ZerosBesselJ'(m, 2));
            _EnvZerosBesselJ := 'true';
            for i from 3 to n while q2 < qmax do
                _Enva := q2; _Envb := 2 × q2 - q1; q1 := q2; q2 := evalf('ZerosBesselJ'(m, i))
            od;
            if i = n + 1 then q2
            else
                r := round(q2/π - 1/2 × m + 1/2 - i);
                a := evalf((n + r + 1/2 × m + 1/2) × π);
                b := evalf(a + 1/4 × π);
                fsolve(BesselJ(m, x), x, a .. b, _Envfdig)
            fi
```

fi

end

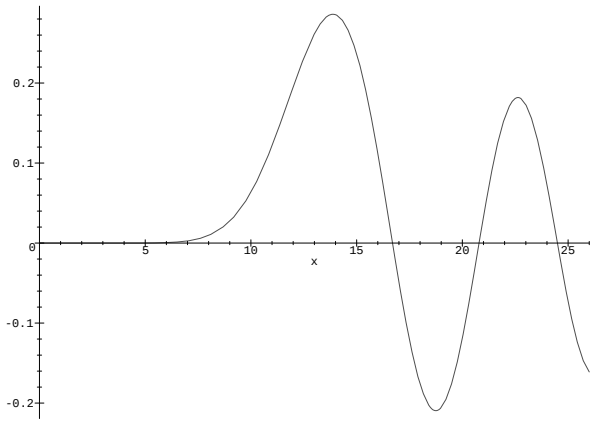
Vejamos um exemplo:

```
> seq(j(12,n),n=1..3);
```

$$16.69824993, 20.78990636, 24.49488504$$

Podemos confirmar os resultados pelo gráfico abaixo:

```
> plot(J(12,x),x=0..26);
```



APÊNDICE B - Zeros da função: $JY_m(x) = J_m(a\,x)\,Y_m(b\,x) - J_m(b\,x)\,Y_m(a\,x)$

Neste apêndice usaremos as convenções do apêndice A. Considere a seguinte função:

```
> alias(J=BesselJ,Y=BesselY,JY=BesselJY):
> BesselJY := (m,x) -> BesselJ(m,a*x)*BesselY(m,b*x)-BesselJ(m,b*x)*BesselY(m,a*x);
```

$$JY := (m,\,x) \to \mathrm{J}(m,\,a\,x)\,\mathrm{Y}(m,\,b\,x) - \mathrm{J}(m,\,b\,x)\,\mathrm{Y}(m,\,a\,x)$$

Queremos construir um procedimento que ache os zeros da função $\mathrm{JY}(m,\,x)$. Para encontrar a n-ésima raiz através do comando *fsolve*, temos que determinar um intervalo que contenha somente a esta raiz. Nenhuma outra raiz pode estar neste intervalo. Vamos definir a função

```
> tanJY := (m,x) -> Y(m,x)/J(m,x);
```

$$tanJY := (m,\,x) \to \frac{\mathrm{Y}(m,\,x)}{\mathrm{J}(m,\,x)}$$

Esta função é semelhante a função $\tan(x)$. A função $\tan(x)$ tem descontinuidades nos pontos $\frac{n\,\pi}{2}$, onde n é um número inteiro. A função $\mathrm{tanJY}(m,\,x)$ tem as singularidades nas raízes da funçao de Bessel $J_m(x)$. A função $\mathrm{JY}(m,\,n)$ pode ser escrita na forma:

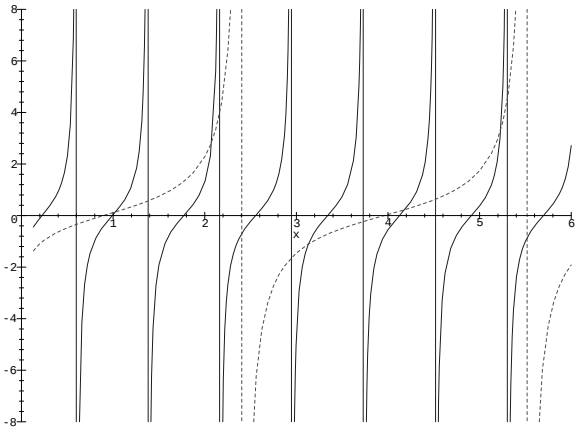
$$\mathrm{JY}(m,\,x) = \mathrm{J}(m,\,a\,x)\,\mathrm{J}(m,\,b\,x)\,(\mathrm{tanJY}(m,\,b\,x) - \mathrm{tanJY}(m,\,a\,x))$$

Assim, as raizes de $\mathrm{JY}(m,\,x)$ são dadas pelos valores x tais que $\mathrm{tanJY}(m,\,a\,x) = \mathrm{tanJY}(m,\,b\,x)$. Vamos fazer o gráfico de $\mathrm{tanJY}(m,\,a\,x)$ e $\mathrm{tanJY}(m,\,b\,x)$ simultaneamente para $a = 1$ e $b = 4$:

```
> a:=1; b:=4; m:=0;
```

$$a := 1$$
$$b := 4$$
$$m := 0$$

```
> g1 := plot(tanJY(m,a*x),x=0..6,-8..8,color=red,linestyle=2):
> g2 := plot(tanJY(m,b*x),x=0..6,-8..8,color=blue,thickness=2):
> plots[display]({g1,g2});
```



Podemos ver que os zeros de $\mathrm{JY}(m,\,x)$ estão sempre em intervalos do tipo $[\frac{\mathrm{j}(m,\,n)}{b}\;..\;\frac{\mathrm{j}(m,\,n+1)}{b}]$. Porém, se o intervalo contém algum número do tipo $\frac{\mathrm{j}(m,\,p)}{a}$ onde p é um número inteiro, então não há nenhuma raiz de $\mathrm{JY}(m,\,x)$ neste intervalo. No exemplo acima, o intervalo $[2.15\;..\;2.92]$ possui um número da forma $\frac{\mathrm{j}(m,\,n)}{a}$, cujo valor é aproximadamente 2.4 (primeira linha tracejada vertical). Este intervalo não possui nenhuma raiz de $\mathrm{JY}(m,\,n)$. Este fato se repete para todo valor de x. Baseado neste fato, podemos construir o seguinte procedimento:

```
ZerosBesselJY := proc(m::algebraic, n::algebraic)
    global a, b, j;
      if 3 < nargs or nargs < 2 then ERROR(`wrong number of arguments`) fi;
      if n = ∞ or n = -∞ or not type(n, realcons) then RETURN('procname(m, n)') fi;
      if not type(n, nonnegint) then ERROR(`2nd argument must be a positive integer`) fi;
      if m = ∞ or m = -∞ or not type(m, realcons) then RETURN('procname(m, n)') fi;
      if evalf(m) < 0 then ERROR(`negative order not implemented`) fi;
      if not assigned(j) then ERROR(`The function ZerosBesselJ must be loaded`) fi;
      if not type([a, b], list('numeric')) then ERROR(`a and b must be numeric.`) fi;
      _Envfdig1 := NULL;
      if nargs = 3 then
        if args_3 = 'fulldigits' then _Envfdig1 := 'fulldigits'
        else ERROR(`third argument must be fulldigits`)
        fi
      fi;
      _EnvZerosBesselJY := 'false';
      evalf('ZerosBesselJY'(m, n))
    end

evalf/ZerosBesselJY := proc(mm, n)
    local m, M, N, i, x, q;
    global a, b;
      m := evalf(mm);
      if n = ∞ or n = -∞ or not type(n, realcons) then RETURN('ZerosBesselJY'(m, n)) fi;
      if not type(n, nonnegint) then ERROR(`2nd argument must be a positive integer`) fi;
      if m = ∞ or m = -∞ or not type(m, realcons) then RETURN('ZerosBesselJY'(m, n)) fi;
      if not type([a, b], list('numeric')) then ERROR(`a and b must be numeric.`) fi;
      if _EnvZerosBesselJY then
        RETURN(fsolve(BesselJY(m, x), x, _Envq1 .. _Envq2, _Envfdig1))
      fi;
      N := 1;
      M := 1;
      for i while N ≤ n do
        _Envq1 := j(m, i)/b;
        _Envq2 := j(m, i + 1)/b;
        q := j(m, M)/a;
        if _Envq1 - q < 0 and q - _Envq2 < 0 then M := M + 1
        else _EnvZerosBesselJY := 'true'; evalf('ZerosBesselJY'(m, i)); N := N + 1
        fi
      od;
      evalf('ZerosBesselJY'(m, n))
    end
```

Vejamos um exemplo:

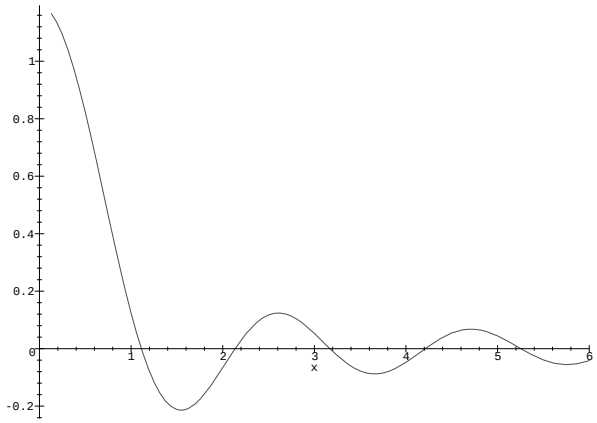
```
> a:=1; b:=4;
```

$$a := 1$$
$$b := 4$$

```
> seq(ZerosBesselJY(1,i),i=1..5);
```

$$1.111876398, 2.134230377, 3.169739710, 4.210407336, 5.253487501$$

```
> plot(JY(1,x),x=0..6);
```



References

[1] R. Lopez, MapleTech, **3** (1996) 54.

[2] M. Rybowicz and J. P. Massias, MapleTech, **3** (1996) 69.

[3] E. Butkov, Mathematical Physics, Addison-Wesley, 1968.

[4] R. Courant and D. Hilbert, Methods of Mathematical Physics, John Wiley & Sons, 1953.

[5] H. Sagan, Boundary and Eigenvalue Problems in Mathematical Physics, John Wiley & Sons, New York, 1961.

[6] E. Kreyszig and E. J. Normington, Maple Computer Algebra Manual for Advanced Engineering Mathematics, John Wiley & Sons, New York, 1994.

[7] M. Kline, Mathematical Thought from Ancient to Modern Times, Oxford University Press, New York, 1972.

[8] G. N. Watson, A Treatise on the Theory of Bessel Functions, Cambridge University Press, 1966.

[9] K. M. Heal et al., Maple V Learning Guide, Springer-Verlag, New York, 1996.

[10] M. B. Monagan et al., Maple V Programming Guide, Springer-Verlag, New York, 1996.

[11] R. Portugal, Introdução ao Maple[18], monografia CBPF-MO-003/96, Centro Brasileiro de Pesquisas Físicas, 1996.

[12] R. Portugal, Introdução a Programação em Maple[19], série Notas de Aula, vol. 1, editora CBPF, Rio de Janeiro, 1996.

[13] P. Schafheitlin, Journal für Math. CXIV (1894) 31.

[14] P. Schafheitlin, Journal für Math. CXXII (1900) 299.

[15] M. Abramowitz and I. A. Stegun, Handbook of Mathematical Functions, Dover, New York, 1972.

[18] Endereço de contato para obter esta monografia: valeria@cat.cbpf.br.
[19] Endereço de contato para obter este livro: myriam@novell.cat.cbpf.br.