7
Referências Bibliográficas

[1] TIMOSHENKO, S. P.. **Theory of Elastic Stability**. Second edition, 1961.

[2] PRZEMIENIECKI, J. S.. **Theory of Matrix Structural Analysis**. 1968.

[3] SILVA, R.; SOARES, W.. **Bifurcação do Equilíbrio em Pórticos Planos**. PUC-Rio, Departamento de Engenharia Civil, Janeiro 1974.

[4] GALLAGHER, R. H.. **Finite Element Analysis Fundamentals**. Prentice Hall Inc, 1975.

[5] BRUSH, D. O.; ALMROTH, B. O.. **Buckling of Bars, Plates, and Shells**. 1975.

[6] BLEVINS, R. D.. **Formulas for Natural Frequency and Mode Shape** Van Nostrand Reinhold Company, 1979.

[7] WEAVER, W. J.; JOHNSTON, P. R.. **Finite Elements For Structural Analysis**. 1984.

[8] WEAVER, W. J.; JOHNSTON, P. R.. **Structural Dynamics by Finite Elements**. Prentice-hall, inc. edition, 1987.

[9] COOK, R. D.; MALKUS, D. S. ; PLESHA, M. E.. **Concepts And Applications of Finite Element Analysis** Third edition, 1989.

[10] GIANNINI, L. D.. **Modelo de elementos finitos para a estabilidade de perfis de paredes finas**. Dissertação de mestrado, Puc-Rio, Rio de Janeiro, Setembro 1990.

[11] DE OLIVEIRA, A. H. S.. **Avaliação de cargas críticas de estruturas planas e axissimétricas sujeitas a dano e fissuração** Dissertação de mestrado, Puc-Rio, Rio de Janeiro, Abril 1990.

[12] BAZANT, Z. P.; CEDOLIN, L.. **Stability of Structures**. 1991.

[13] MARTINS, M. F.. **Estratégias adaptativas com versões p e h-p do mef para problemas de elasticidade plana e placas** Dissertação de mestrado, Puc-Rio, Rio de Janeiro, 1994.

[14] SZILARD, R.. **Theories and Aplications of Plate and Analisys: Classical Numerical Engineering Methods** Prentice Hall, 1995.

[15] KOTSOVOS, M. D.; PAVLOVIC', M. N.. **Structural Concrete**. Thomas Telford Services Ltd, 1995.

[16] FUSCO, P. B.. **Técnica de armar as estruturas de concreto**. Pini, 1995.

[17] ABNT. **NBR 6118 - Projeto de Estruturas de Concreto - Procedimentos**. Rio de Janeiro, 2003.

[18] WAIDMAN, L.. **Análise dinâmica de placas delgadas utilizando elementos finitos triangulares e retangulares** Dissertação de mestrado, Universidade Estadual Paulista Júlio de Mesquita Filho - Unesp, São Paulo, Janeiro 2004.

[19] DA CUNHA, P. C.. **Comportamento crítico e pós-crítico de placas dobradas**. Dissertação de mestrado, PUC - Rio, Rio de Janeiro, Setembro 2005.

[20] KIMURA, E. A.. **Análise comparativa do dimensionamento de pilares entre as normas: Nbr 6118:1990 e nbr 6118:2003** TQS Informática Ltda, 2006.

[21] JONES, R. M.. **Buckling of Bars, Plates and Shells**. Bull Ridge, 2006.

[22] E SILVA, R. R.. **Flambagem de placas**. Rotina Computacional, Novembro 2006.

A
Trecho do programa para a obtenção da função adicional polinomial e gráficos comparativos

Este apêndice tem por finalidade apresentar um trecho demonstrativo, através do software *Maple*, de como obter a função polinomial. Além disso, apresenta gráficos comparativos para verificar as diferenças entre funções polinomiais e trigonométricas.

A.1
Trecho computacional para a obtenção da função polinomial

Para analisar internamente uma placa foi elaborada uma rotina para a obtenção da função adicional polinomial 3-21 obedecendo as condições de contorno, como será descrito a seguir.

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

Função polinomial de grau n:

```
> w:=a+b*xi+c*xi^2+d*xi^3+xi^n;
```

Condições de contorno:

```
> w1:=subs(xi=-1,w);

> w2:=subs(xi=1,w);

> w3:=subs(xi=1,diff(w,xi));

> w4:=subs(xi=-1,diff(w,xi));
```

Solução do sistema de equações:

```
> sist:=w1=0,w2=0,w3=0,w4=0:
> var:=a,b,c,d:
> S:=simplify(solve(sist, var)):
```

Função adicional polinomial:

```
> W:=subs(S,w);
```

$$W := -1/2 + 1/2\,(-1)^{1+n} + 1/4\,n + 1/4\,(-1)^{n}\,n + \left(-3/4 + 3/4\,(-1)^{n} + 1/4\,n + 1/4\,(-1)^{1+n}\,n\right)\xi + \left(-1/4\,n + 1/4\,(-1)^{1+n}\,n\right)\xi^{2} + \left(1/4 + 1/4\,(-1)^{1+n} - 1/4\,n + 1/4\,(-1)^{n}\,n\right)\xi^{3} + \xi^{n}$$

```
> a:=1:b:=1:nx:=4:ny:=5:
```

```
> W1:=subs(xi=2*x/a-1,n=nx,W);
```

Substituições das coordenadas paramétricas por x e y:

```
> W2:=subs(xi=2*y/b-1,n=ny,W);
```

Função polinomial para a Placa

```
> WW:=W1*W2;
```

$$WW := \left(1 - 2\,(2\,x-1)^{2} + (2\,x-1)^{4}\right)\left(-1 + 2\,y - 2\,(2\,y-1)^{3} + (2\,y-1)^{5}\right)$$

```
> plot([W1,subs(y=x,W2)],x=0..a,title="",legend=["Primeiro
Modo", "Segundo Modo"]);
```

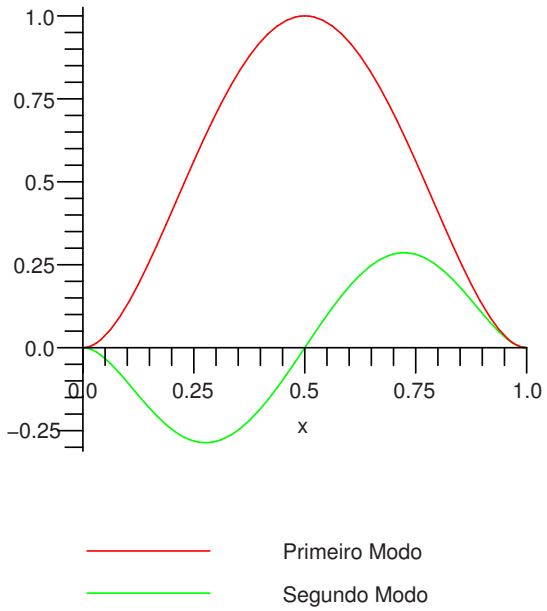

Figura A.1: Gráfico representando o primeiro e segundo modo de vibração para a função polinomial.

```
> plot3d(WW,x=0..a,y=0..b);
```

O gráfico abaixo representa uma placa engastada onde há duas funções na direção y e uma na direção y.

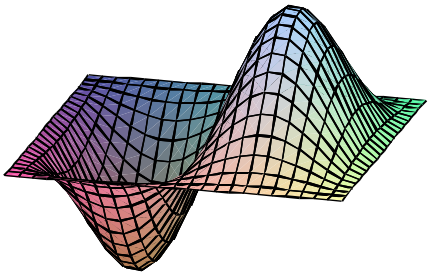

Figura A.2: Vibração da placa com a função polinomial.

Para a função trigonométrica, os gráficos bidimensional e tridimensional são A.3 e A.4 :

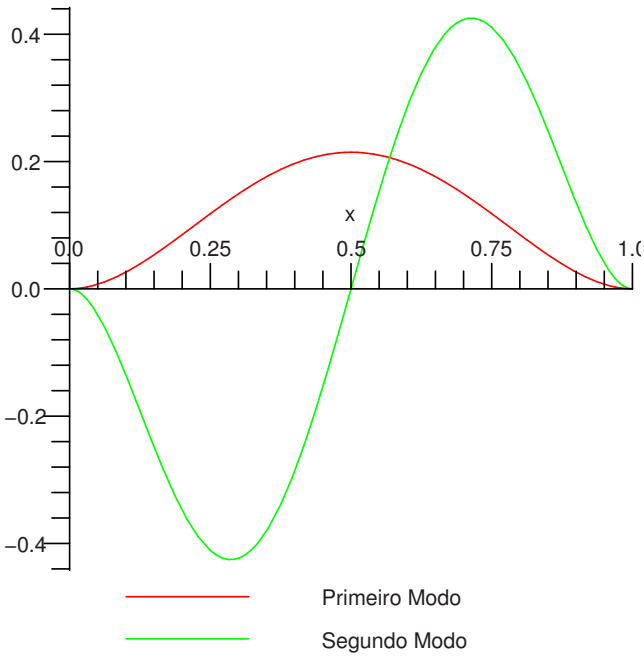

Figura A.3: Gráfico representando o primeiro e segundo modo de vibração para a função trigonométrica.

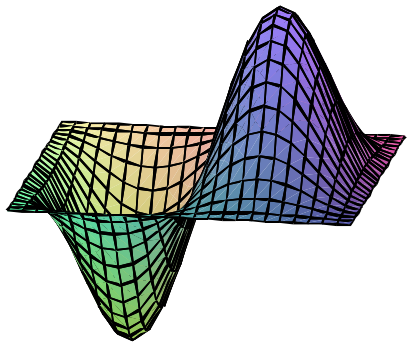

Figura A.4: Vibração da placa com uso de função trigonométrica.

A.1.1
Gráficos Comparativos entre funções polinomiais e trigonométrica

O gráfico A.5 representa a plotagem das funções polinomiais versus a trigonométrica como foi citado no Capítulo 3, com os valores igualados no centro para melhor comparação.

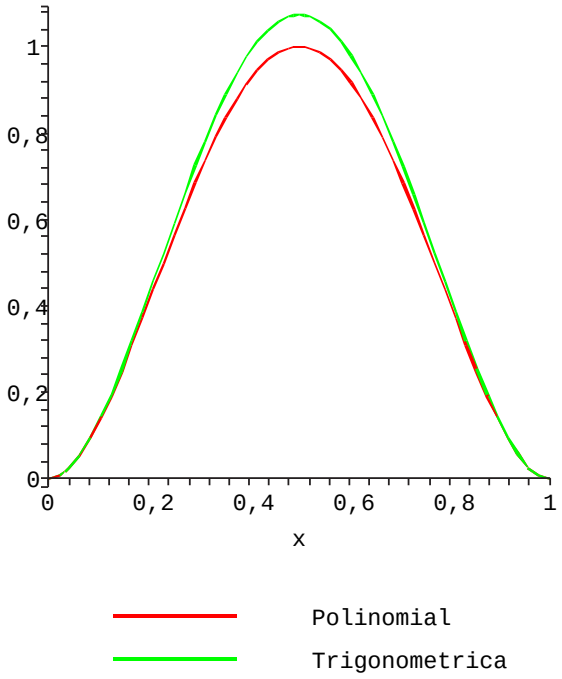

Figura A.5: Representação das Funções Polinomial e Trigonométrica em x (n=1).

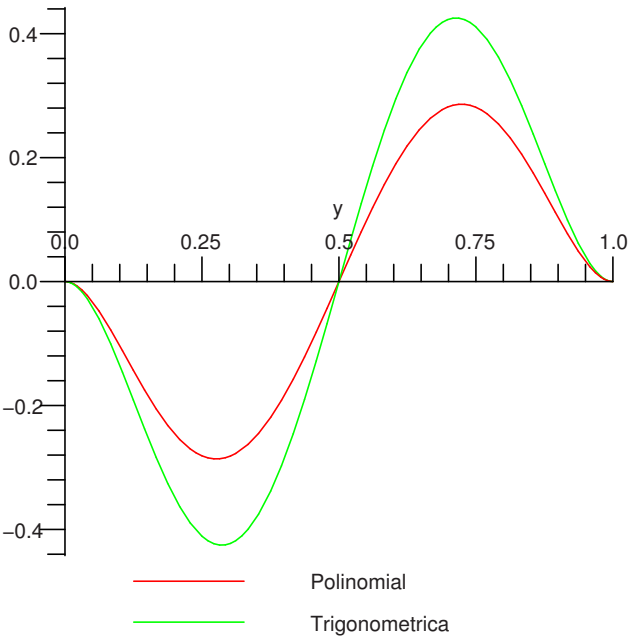

Figura A.6: Visualização das Funções Polinomial e Trigonométrica em y (n=2).

Os gráficos, A.7 e A.8, demonstram as derivadas das funções de primeira ordem das primeiras funções.

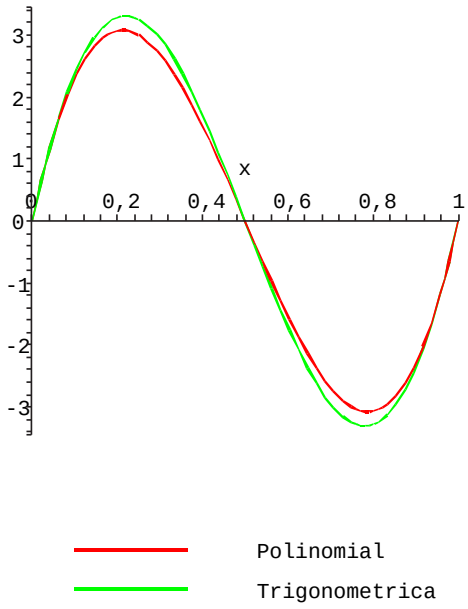

Figura A.7: Visualização das funções da primeira derivada em x.

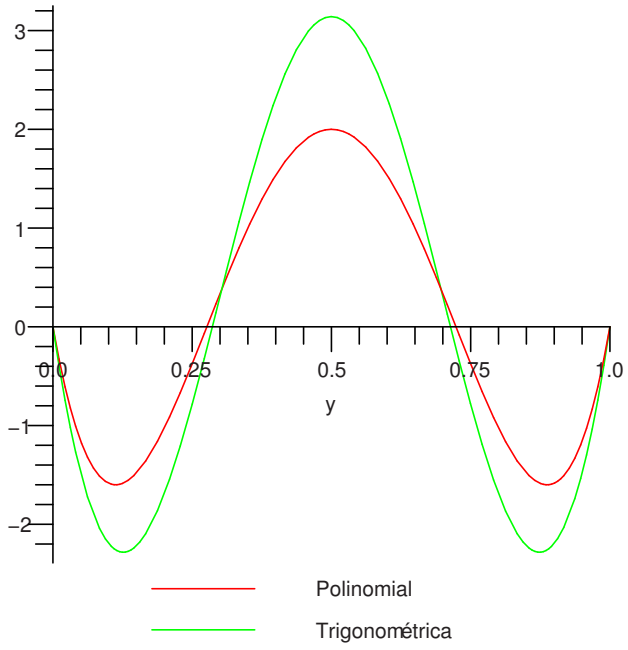

Figura A.8: Visualização da primeira derivada em y.

A comparação gráfica das derivadas de segunda ordem são representadas pelos seguintes gráficos A.9 e A.10:

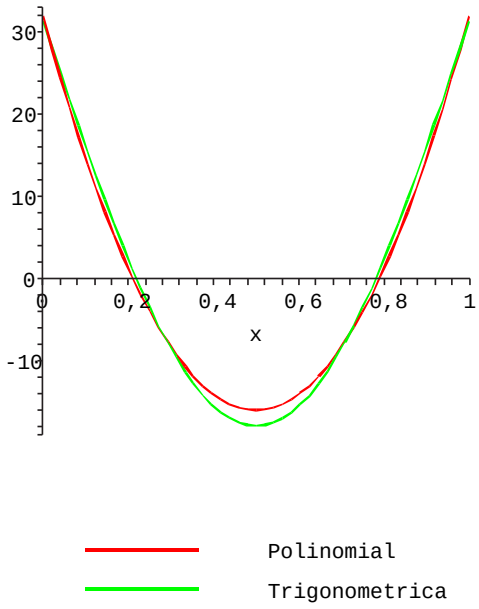

Figura A.9: Visualização da segunda derivada em x.

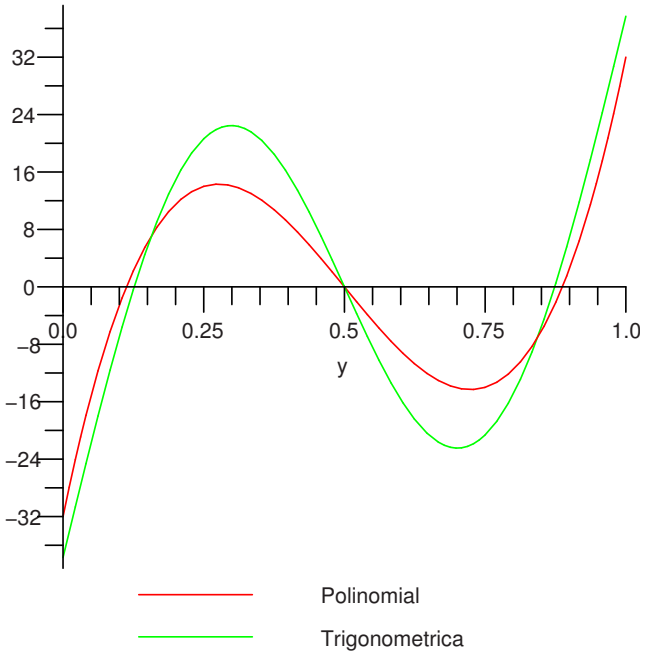

Figura A.10: Visualização da segunda derivada em y.

As funções polinomiais geram resultados num intervalo de tempo menor que as funções trigonométricas. Entretanto, as funções trigonométricas são mais estáveis e é possível usar uma quantidade grande de funções para o cálculo sem gerar erros como acontece com as funções polinomiais.

B
Trecho do modelo computacional

Este apêndice apresenta a rotina computacional elaborada para o cálculo da carga crítica de flambagem. A seguir serão mostradas a obtenção das matrizes de rigidez elástica, geométrica e de massa.

B.1
Programa para o estudo de flambagem de uma placa

Seguem os pacotes responsáveis pela resolução dos cálculos numéricos:

```
> restart:
> with(linalg):with(LinearAlgebra):
> with(student):
```

B.1.1
Propriedades geométricas e do material:

```
> t:=10:(espessura da placa em cm)
> a:=200:b:=500:(dimensões da placa em cm)
> rho:=0.00785:(Kg/m^3)
> nu:=0.3:(Coeficiente de Poisson)
> lambda:=(1-nu)/2:
> Eb:=21000:(Módulo de elasticidade do concreto em KN/cm^2)
> Ea:=Eb*(t^3/12):
    E1:=evalm(matrix(3,3,[1,nu,0,nu,1,0,0,0,lambda]) \
*((Ea)/(1-nu^2))):
```

B.1.2
Informe o número de funções adicionais para a variação da placa:

```
> Ax:=3:Ay:=3:Axy:=Ax*Ay:(número de funções adicionais a serem
adicionadas ao modelo)
```

B.1.3
Informe o número de funções adicionais para a variação dos lados da placa:

```
> Lx:=3:Ly:=3:Lxy:=4*Lx+4*Ly:(número de funções para o
contorno da placa)

> Lx1:=2*Lx:Lx2:=2*Lx:Ly1:=2*Ly:Ly2:=2*Ly:

> TOT:=16+Axy+Lxy:(número total de funções)
```

B.1.4
Forças aplicadas na placa

```
> N[x]:=10:N[y]:=0:N[xy]:=0:bz:=1:(N=Força axial aplicada;
bz=carga transversal a placa)

> Nxy:=matrix(2,2,[N[x],N[xy],N[xy],N[y]]):(matriz de \
carregamento axial)
```

B.1.5
Condições de contorno para a placa

Condições de Contorno nodais da placa (Equações Convencionais)

Para indicar uma restrição utilizaremos valor o unitário.

```
> for i to TOT do

> RestC[i]:=0:end do:
```

Nó 1

```
> RestC[1]:=1:RestC[2]:=1:RestC[3]:=1:RestC[4]:=0:
```

Nó 2

```
> RestC[5]:=1:RestC[6]:=1:RestC[7]:=1:RestC[8]:=0:
```

Nó 3

```
> RestC[9]:=1:RestC[10]:=1:RestC[11]:=1:RestC[12]:=0:
```

Nó 4

```
> RestC[13]:=1:RestC[14]:=1:RestC[15]:=1:RestC[16]:=0:
```

Condições no contorno da placa (Equações adicionais de lado)

Da mesma forma que os graus de liberdade nodais, para prender algum lado insere-se um valor unitário

Lado 1 (prende em x entre nós 1 e 2)

```
> Lado1:=1:
> for i to Lx1 do ;
> RestC[16+Axy+i]:=Lado1;
> od:
```

Lado 2 (prende em x entre nós 3 e 4)

```
> Lado2:=1:
> for i to Lx2 do ;
> RestC[16+Axy+Lx1+i]:=Lado2;
> od:
```

Lado 3 (prende em y entre nós 1 e 4)

```
> Lado3:=1:
> for i to Ly1 do ;
> RestC[16+Axy+Lx1+Lx2+i]:=Lado3;
> od:
```

Lado 4 (prende em y entre nós 2 e 3)

```
> Lado4:=1:
> for i to Ly2 do ;
> RestC[16+Axy+Lx1+Lx2+Ly1+i]:=Lado4;
> od:
> TOT1:=0: (TOT! é o número de restrições igual a "zero")
> for i from 1 to TOT do
> if RestC[i]=0 then
> TOT1:=TOT1+1;
> end if;
> end;
> TOT1;
```


B.1.6
Funções de forma nodais (convencionais)

```
> C[11]:=(1-3*xi^2+2*xi^3)*(1-3*eta^2+2*eta^3):
> C[12]:=(1-3*xi^2+2*xi^3)*(eta-2*eta^2+eta^3)*b:
> C[13]:=-(xi-2*xi^2+xi^3)*(1-3*eta^2+2*eta^3)*a:
> C[14]:=(xi-2*xi^2+xi^3)*(eta-2*eta^2+eta^3)*a*b:
> C[21]:=(3*xi^2-2*xi^3)*(1-3*eta^2+2*eta^3):
> C[22]:=(3*xi^2-2*xi^3)*(eta-2*eta^2+eta^3)*b:
> C[23]:=(xi^2-xi^3)*(1-3*eta^2+2*eta^3)*a:
> C[24]:=-(xi^2-xi^3)*(eta-2*eta^2+eta^3)*a*b:
> C[31]:=(3*xi^2-2*xi^3)*(3*eta^2-2*eta^3):
> C[32]:=-(3*xi^2-2*xi^3)*(eta^2-eta^3)*b:
> C[33]:=(xi^2-xi^3)*(3*eta^2-2*eta^3)*a:
> C[34]:=(xi^2-xi^3)*(eta^2-eta^3)*a*b:
> C[41]:=(1-3*xi^2+2*xi^3)*(3*eta^2-2*eta^3):
> C[42]:=-(1-3*xi^2+2*xi^3)*(eta^2-eta^3)*b:
```

```
> C[43]:=-(xi-2*xi^2+xi^3)*(3*eta^2-2*eta^3)*a:
> C[44]:=-(xi-2*xi^2+xi^3)*(eta^2-eta^3)*a*b:
```

C[11]:=subs(xi=x/a,eta=y/b,C[11]):C[12]:=subs(xi=x/a,eta=y/b,C[12]):
C[13]:=subs(xi=x/a,eta=y/b,C[13]):C[14]:=subs(xi=x/a,eta=y/b,C[14]):
C[21]:=subs(xi=x/a,eta=y/b,C[21]):C[22]:=subs(xi=x/a,eta=y/b,C[22]):
C[23]:=subs(xi=x/a,eta=y/b,C[23]):C[24]:=subs(xi=x/a,eta=y/b,C[24]):
C[31]:=subs(xi=x/a,eta=y/b,C[31]):C[32]:=subs(xi=x/a,eta=y/b,C[32]):
C[33]:=subs(xi=x/a,eta=y/b,C[33]):C[34]:=subs(xi=x/a,eta=y/b,C[34]):
C[41]:=subs(xi=x/a,eta=y/b,C[41]):C[42]:=subs(xi=x/a,eta=y/b,C[42]):
C[43]:=subs(xi=x/a,eta=y/b,C[43]):C[44]:=subs(xi=x/a,eta=y/b,C[44]):

```
> WC1:=evalm((matrix(4,1,[C[11],C[12],C[13],C[14]]))):
> WC2:=evalm((matrix(4,1,[C[21],C[22],C[23],C[24]]))):
> WC3:=evalm((matrix(4,1,[C[31],C[32],C[33],C[34]]))):
> WC4:=evalm((matrix(4,1,[C[41],C[42],C[43],C[44]]))):
> WC:=stackmatrix(WC1,WC2,WC3,WC4):
```

Matriz de Massa para as Equações Convencionais

```
> WCM:=evalm(rho*t*WC&*transpose(WC)):
```

for i to 16 do for j to 16 do WCM[i,j]:=value(Doubleint(WCM[i,j],x = 0 .. a,y=0..b)); end:end:

```
> K[M[CC]]:=WCM:
```

Matriz de rigidez geométrica para as equações convencionais

Resolvendo a derivada em relação a x

```
> WCx1:=evalm((matrix(4,1,[C[11],C[12],C[13],C[14]]))):
> WCx2:=evalm((matrix(4,1,[C[21],C[22],C[23],C[24]]))):
> WCx3:=evalm((matrix(4,1,[C[31],C[32],C[33],C[34]]))):
> WCx4:=evalm((matrix(4,1,[C[41],C[42],C[43],C[44]]))):
```

for i to 4 do for j to 1 do WCx1[i,j] := evalm(diff(WCx1[i,j],x)) end end;

```
> dWCGx1:=WCx1:
```

for i to 4 do for j to 1 do WCx2[i,j] := evalm(diff(WCx2[i,j],x)) end end;

```
> dWCGx2:=WCx2:
```

for i to 4 do for j to 1 do WCx3[i,j] := evalm(diff(WCx3[i,j],x)) end end;

```
> dWCGx3:=WCx3:
```

for i to 4 do for j to 1 do WCx4[i,j] := evalm(diff(WCx4[i,j],x)) end end;

```
> dWCGx4:=WCx4:
```

Resolvendo a derivada em relação a y

```
> WCy1:=evalm((matrix(4,1,[C[11],C[12],C[13],C[14]]))):
> WCy2:=evalm((matrix(4,1,[C[21],C[22],C[23],C[24]]))):
> WCy3:=evalm((matrix(4,1,[C[31],C[32],C[33],C[34]]))):
> WCy4:=evalm((matrix(4,1,[C[41],C[42],C[43],C[44]]))):
```

for i to 4 do for j to 1 do WCy1[i,j] := evalm(diff(WCy1[i,j],y)) end end;

```
> dWCGy1:=WCy1:
```

for i to 4 do for j to 1 do WCy2[i,j] := evalm(diff(WCy2[i,j],y)) end end;

```
> dWCGy2:=WCy2:
```

for i to 4 do for j to 1 do WCy3[i,j] := evalm(diff(WCy3[i,j],y)) end end;

```
> dWCGy3:=WCy3:
```

for i to 4 do for j to 1 do WCy4[i,j] := evalm(diff(WCy4[i,j],y)) end end;

```
> dWCGy4:=WCy4:
```

Concatenando as derivadas em relação a x e y

```
> BdWCx1:=concat(dWCGx1,dWCGy1):
> BdWCx2:=concat(dWCGx2,dWCGy2):
> BdWCx3:=concat(dWCGx3,dWCGy3):
> BdWCx4:=concat(dWCGx4,dWCGy4):
> BdWCxy:=stackmatrix(BdWCx1,BdWCx2,BdWCx3,BdWCx4):
> WCG:=evalm(BdWCxy&*Nxy&*transpose(BdWCxy)):
> for i to 16 do for j to 16 do
> WCG[i,j]:=value(Doubleint(WCG[i,j],x= 0 .. a,y=0..b));
> od:od:
```

```
> K[G[CC]]:=WCG:
```

Matriz de rigidez elástica para as equações convencionais

Derivada a segunda em relação a x

```
> WCxx1:=evalm((matrix(1,4,[C[11],C[12],C[13],C[14]]))):
```

for i to 1 do for j to 4 do WCxx1[i,j] := evalm(diff(WCxx1[i,j],x,x)) end end;

```
> ddWCxx1:=WCxx1:
> WCxx2:=evalm((matrix(1,4,[C[21],C[22],C[23],C[24]]))):
```

for i to 1 do for j to 4 do WCxx2[i,j] := evalm(diff(WCxx2[i,j],x,x)) end end;

```
> ddWCxx2:=WCxx2:
> WCxx3:=evalm((matrix(1,4,[C[31],C[32],C[33],C[34]]))):
```

for i to 1 do for j to 4 do WCxx3[i,j] := evalm(diff(WCxx3[i,j],x,x)) end end;

```
> ddWCxx3:=WCxx3:
> WCxx4:=evalm((matrix(1,4,[C[41],C[42],C[43],C[44]]))):
```

for i to 1 do for j to 4 do WCxx4[i,j] := evalm(diff(WCxx4[i,j],x,x)) end end;

```
> ddWCxx4:=WCxx4:
```

Derivada a segunda em relação a y

```
> WCyy1:=evalm((matrix(1,4,[C[11],C[12],C[13],C[14]]))):
```

for i to 1 do for j to 4 do WCyy1[i,j] := evalm(diff(WCyy1[i,j],y,y)) end end;

```
> ddWCyy1:=WCyy1:
> WCyy2:=evalm((matrix(1,4,[C[21],C[22],C[23],C[24]]))):
```

for i to 1 do for j to 4 do WCyy2[i,j] :=evalm(diff(WCyy2[i,j],y,y)) end end;

```
> ddWCyy2:=WCyy2:
> WCyy3:=evalm((matrix(1,4,[C[31],C[32],C[33],C[34]]))):
```

for i to 1 do for j to 4 do WCyy3[i,j] := evalm(diff(WCyy3[i,j],y,y)) end end;

```
> ddWCyy3:=WCyy3:
> WCyy4:=evalm((matrix(1,4,[C[41],C[42],C[43],C[44]]))):
```

for i to 1 do for j to 4 do WCyy4[i,j] := evalm(diff(WCyy4[i,j],y,y)) end end;

```
> ddWCyy4:=WCyy4:
```

Derivada a segunda em relação a x e y

```
> WCxy1:=evalm((matrix(1,4,[C[11],C[12],C[13],C[14]]))):
```

for i to 1 do for j to 4 do WCxy1[i,j] := evalm(diff(WCxy1[i,j],x,y)) end end;

```
> ddWCxy1:=WCxy1:
> WCxy2:=evalm((matrix(1,4,[C[21],C[22],C[23],C[24]]))):
```

for i to 1 do for j to 4 do WCxy2[i,j] := evalm(diff(WCxy2[i,j],x,y)) end end;

```
> ddWCxy2:=WCxy2:
> WCxy3:=evalm((matrix(1,4,[C[31],C[32],C[33],C[34]]))):
```

for i to 1 do for j to 4 do WCxy3[i,j] := evalm(diff(WCxy3[i,j],x,y)) end end;

```
> ddWCxy3:=WCxy3:
> WCxy4:=evalm((matrix(1,4,[C[41],C[42],C[43],C[44]]))):
```

for i to 1 do for j to 4 do WCxy4[i,j] := evalm(diff(WCxy4[i,j],x,y)) end end;

```
> ddWCxy4:=WCxy4:
```

Concatenando as derivadas x, y e xy

```
> BddWC1:=stackmatrix(ddWCxx1,ddWCyy1,2*ddWCxy1):
> BddWC2:=stackmatrix(ddWCxx2,ddWCyy2,2*ddWCxy2):
> BddWC3:=stackmatrix(ddWCxx3,ddWCyy3,2*ddWCxy3):
> BddWC4:=stackmatrix(ddWCxx4,ddWCyy4,2*ddWCxy4):
> BddWC:=concat(BddWC1,BddWC2,BddWC3,BddWC4):
> WEC:=evalm(transpose(BddWC)&*E1&*BddWC):
```

for i to 16 do for j to 16 do WEC[i,j] :=value(Doubleint(WEC[i,j],x = 0 .. a,y=0..b)) end end;

```
> K[E[CC]]:=WEC:
```

B.1.7
Funções adicionais internas

$$WAx := -1/2 + 1/2 * (-1)\hat{}(1+nx) + 1/4 * nx + 1/4 * (-1)\hat{}nx * nx + (-3/4 + 3/4 * (-1)\hat{}nx + 1/4 * nx + 1/4 * (-1)\hat{}(1+nx) * nx) * (2 * x/a - 1) + (-1/4 * nx + 1/4 * (-1)\hat{}(1+nx) * nx) * (2 * x/a - 1)\hat{}2 + (1/4 + 1/4 * (-1)\hat{}(1+nx) - 1/4 * nx + 1/4 * (-1)\hat{}nx * nx) * (2 * x/a - 1)\hat{}3 + (2 * x/a - 1)\hat{}nx : \quad \text{(B-1)}$$

```
WAy:=-1/2+1/2*(-1)^(1+ny)+1/4*ny+1/4*(-1)^ny*ny+
(-3/4+3/4*(-1)^ny+1/4*ny+1/4*(-1)^(1+ny)*ny)*(2*y/b-1)+
(-1/4*ny+1/4*(-1)^(1+ny)*ny)*(2*y/b-1)^2+(1/4+1/4*(-1)^(1+ny)-
1/4*ny+1/4*(-1)^ny*ny)*(2*y/b-1)^3+(2*y/b-1)^ny:
```

Função adicional na direção x

```
> WAVx:=WAx
> WAVx:=vector(Ax):
> for i from 1 to Ax do
> nx:=i+3;
WAVx[i]:=-1/2+1/2*(-1)^(1+nx)+1/4*nx+1/4*(-1)^nx*nx+
(-3/4+3/4*(-1)^nx+1/4*nx+1/4*(-1)^(1+nx)*nx)*(2*x/a-1)+
(-1/4*nx+1/4*(-1)^(1+nx)*nx)*(2*x/a-1)^2+(1/4+
1/4*(-1)^(1+nx)-1/4*nx+1/4*(-1)^nx*nx)*(2*x/a-1)^3+(2*x/a-1)^nx;
> od:
> WAMx:=convert(WAVx,matrix):
```

Função adicional na direção y

```
> WAVy:=WAy
> WAVy:=vector(Ay):
> for i from 1 to Ay do
> ny:=i+3;
WAVy[i]:=-1/2+1/2*(-1)^(1+ny)+1/4*ny+1/4*(-1)^ny*ny+
(-3/4+3/4*(-1)^ny+1/4*ny+1/4*(-1)^(1+ny)*ny)*(2*y/b-1)+
```

(-1/4*ny+1/4*(-1)^(1+ny)*ny)*(2*y/b-1)^2+(1/4+1/4*(-1)^(1+ny)-
1/4*ny+1/4*(-1)^ny*ny)*(2*y/b-1)^3+(2*y/b-1)^ny;

```
> WAMy:=convert(WAVy,matrix):
```

Fazendo a Multiplicação das funções x por y

```
> WAxy1:=evalm(WAMx&*transpose(WAMy)):
> WAxy2:=convert(WAxy1,vector):
> WAxy:=matrix(Axy,1,WAxy2):
```

Matriz de massa para as funções adicionais internas

```
> WMA:=evalm(rho*t*WAxy&*transpose(WAxy)):
```

for i to Axy do for j to Axy do WMA[i,j] :=value(Doubleint(WMA[i,j],x = 0 ..
a,y=0..b)) end end;

```
> K[M[AA]]:=WMA:
```

Matriz de rigidez geométrica para as funções adicionais internas

Derivada a primeira em relação a x

```
> WAxy3:=WAxy1:
> WAxy4:=convert(WAxy3,vector):
> WAGx:=matrix(Axy,1,WAxy4):
```

for i to Axy do for j to 1 do WAGx[i,j] := evalm(diff(WAGx[i,j],x)) end end;

```
> dWAGx:=WAGx:
```

Derivada a primeira em relação a y

```
> WAxy5:=WAxy1:
> WAxy6:=convert(WAxy5,vector):
> WAGy:=matrix(Axy,1,WAxy6):
```

for i to Axy do for j to 1 do WAGy[i,j] := evalm(diff(WAGy[i,j],y)) end end;

```
> dWAGy:=WAGy:
```

Multipicação das derivadas em relação a x e y

```
> dWAGxy:=concat(dWAGx,dWAGy):
> WAGxy:=evalm(dWAGxy&*Nxy&*transpose(dWAGxy)):
```

for i to Axy do for j to Axy do WAGxy[i,j] :=value(Doubleint(WAGxy[i,j],x = 0 .. a,y=0..b)) end end;

```
> K[G[AA]]:=WAGxy:
```

Matriz de rigidez elástica utilizando somente funções adicionais

Derivada a segunda em relação a x

```
> WAxy7:=WAxy1:
> WAxy8:=convert(WAxy7,vector):
> WAEx:=matrix(1,Axy,WAxy8):
```

for i to 1 do for j to Axy do WAEx[i,j] := evalm(diff(WAEx[i,j],x,x)) end end;

```
> ddWAEx:=WAEx:
```

Derivada à segunda em relação a y

```
> WAxy9:=WAxy1:
> WAxy10:=convert(WAxy9,vector):
> WAEy:=matrix(1,Axy,WAxy10):
```

for i to 1 do for j to Axy do WAEy[i,j] := evalm(diff(WAEy[i,j],y,y)) end end;

```
> ddWAEy:=WAEy:
```

Derivada em relação a x e y

```
> WAxy11:=WAxy1:
> WAxy12:=convert(WAxy11,vector):
> WAExy:=matrix(1,Axy,WAxy12):
```

for i to 1 do for j to Axy do WAExy[i,j] := evalm(diff(WAExy[i,j],x,y)) end end;

```
> ddWAExy:=WAExy:
```

Unindo as funções formando um vetor

```
> ddWAEprov:=stackmatrix(ddWAEx,ddWAEy,2*ddWAExy):
> KAEC:=evalm(transpose(ddWAEprov)&*E1&*ddWAEprov):
```

for i to Axy do for j to Axy do KAEC[i,j]:=value(Doubleint(KAEC[i,j],x=0 .. a,y=0..b)) end end;

```
> K[E[AA]]:=KAEC:
```

B.1.8
Montagem das matrizes utilizando funções de contorno

Funções para variação dos lados da placa

```
> L[x1]:=(1-3*y^2/b^2+2*y^3/b^3):
> L[x2]:=(y-2*y^2/b+y^3/b^2):
> L[x3]:=(3*y^2/b^2-2*y^3/b^3):
> L[x4]:=(-y^2/b+y^3/b^2):
> L[y1]:=(1-3*x^2/a^2+2*x^3/a^3):
> L[y2]:=(x-2*x^2/a+x^3/a^2):
> L[y3]:=(3*x^2/a^2-2*x^3/a^3):
> L[y4]:=(-x^2/a+x^3/a^2):
```

Funções para o lado da placa na direção x

$$WLVx := (-1/2 + 1/2 * (-1)\hat{}(1 + mx) + 1/4 * mx + 1/4 * (-1)\hat{}mx * mx + (-3/4 + 3/4 * (-1)\hat{}mx + 1/4 * mx + 1/4 * (-1)\hat{}(1 + mx) * mx) * (2 * x/a - 1) + (-1/4 * mx + 1/4 * (-1)\hat{}(1 + mx) * mx) * (2 * x/a - 1)\hat{}2 + (1/4 + 1/4 * (-1)\hat{}(1 + mx) - 1/4 * mx + 1/4 * (-1)\hat{}mx * mx) * (2 * x/a - 1)\hat{}3 + (2 * x/a - 1)\hat{}mx) : \quad \text{(B-2)}$$

```
> WLVx:=Vector(Lx):
> for i from 1 to Lx do
> mx:=i+3;
```

WLVx[i]:=(-1/2+1/2*(-1)^(1+mx)+1/4*mx+1/4*(-1)^mx*mx+
(-3/4+3/4*(-1)^mx+1/4*mx+1/4*(-1)^(1+mx)*mx)*(2*x/a-1)+
(-1/4*mx+1/4*(-1)^(1+mx)*mx)*(2*x/a-1)^2+
(1/4+1/4*(-1)^(1+mx)-1/4*mx+1/4*(-1)^mx*mx)*
(2*x/a-1)^3+(2*x/a-1)^mx);

```
> WLVx1:=convert(WLVx,vector):
> WLMx:=matrix(Lx,1,WLVx1):
```

Funções para o Lado da Placa na direção y

WLVy:=(-1/2+1/2*(-1)^(1+my)+1/4*my+1/4*(-1)^my*my+
(-3/4+3/4*(-1)^my+1/4*my+1/4*(-1)^(1+my)*my)*(2*y/b-1)+
(-1/4*my+1/4*(-1)^(1+my)*my)*(2*y/b-1)^2+(1/4+1/4*(-1)^(1+my)
-1/4*my+1/4*(-1)^my*my)*(2*y/b-1)^3+(2*y/b-1)^my):

```
> WLVy:=Vector(Ly):
> for i from 1 to Ly do
> my:=i+3;
```

WLVy[i]:=(-1/2+1/2*(-1)^(1+my)+1/4*my+1/4*(-1)^my*my+
(-3/4+3/4*(-1)^my+1/4*my+1/4*(-1)^(1+my)*my)*(2*y/b-1)+
(-1/4*my+1/4*(-1)^(1+my)*my)*(2*y/b-1)^2+(1/4+1/4*(-1)^(1+my)
-1/4*my+1/4*(-1)^my*my)*(2*y/b-1)^3+(2*y/b-1)^my);

```
> od:
> WLVy1:=convert(WLVy,vector):
> WLMy:=matrix(Ly,1,WLVy1):
```

Multiplicação para a Função do lado com a de Variação

```
> WLMx1:=evalm((WLMx)*(L[x1])):
> WLMx2:=evalm(WLMx*(L[x2])):
> WLMx3:=evalm(WLMx*(L[x3])):
> WLMx4:=evalm(WLMx*(L[x4])):
> WLMy1:=evalm(WLMy*(L[y1])):
> WLMy2:=evalm(WLMy*(L[y2])):
> WLMy3:=evalm(WLMy*(L[y3])):
```

> WLMy4:=evalm(WLMy*(L[y4])):

$$WLMxy := stackmatrix(WLMx1, WLMx2, WLMx3, WLMx4, WLMy1, WLMy2, WLMy3, WLMy4) : \quad \text{(B-3)}$$

Matriz de massa utilizando as funções de lado

> KMLxy:=evalm(rho*t*WLMxy&*transpose(WLMxy)):

for i to Lxy do for j to Lxy do KMLxy[i,j] :=value(Doubleint(KMLxy[i,j],x = 0 .. a,y=0..b)) end end;

> K[M[LL]]:=KMLxy:

Matriz de rigidez geométrica utilizando as funções de lado

Derivada a primeira em relação a x

WLGx:=stackmatrix(WLMx1,WLMx2,WLMx3,WLMx4, WLMy1,WLMy2,WLMy3,WLMy4): for i to Lxy do for j to 1 do WLGx[i,j] := evalm(diff(WLGx[i,j],x)) end end;

> dWLGx:=WLGx:

WLGy:=stackmatrix(WLMx1,WLMx2,WLMx3,WLMx4, WLMy1,WLMy2,WLMy3,WLMy4):

Derivada à primeira em relação a y

for i to Lxy do for j to 1 do WLGy[i,j] := evalm(diff(WLGy[i,j],y)) end end;

> dWLGy:=WLGy:

> dWGL:=concat(dWLGx,dWLGy):

> KGLxy:=evalm(dWGL&*Nxy&*transpose(dWGL)):

for i to Lxy do for j to Lxy do KGLxy[i,j] :=value(Doubleint(KGLxy[i,j],x = 0 .. a,y=0..b)) end end;

> K[G[LL]]:=KGLxy:

Matriz de rigidez elástica utilizando as funções de lado

Derivada a segunda em relação à x

```
> WLExx:=WLMxy:
> WLEVxx:=convert(WLExx,vector):
> ddWMLxx:=matrix(1,Lxy,WLEVxx):
```

for i to 1 do for j to Lxy do ddWMLxx[i,j] := evalm(diff(ddWMLxx[i,j],x,x)) end end:

```
> ddWLxx:=ddWMLxx:
```

Derivada à segunda em relação à y

```
> WLEyy:=WLMxy:
> WLEVyy:=convert(WLEyy,vector):
> ddWMLyy:=matrix(1,Lxy,WLEVyy):
```

for i to 1 do for j to Lxy do ddWMLyy[i,j] := evalm(diff(ddWMLyy[i,j],y,y)) end end;

```
> ddWLyy:=ddWMLyy:
```

Derivada de segunda ordem em relação a x e y

```
> WLExy:=WLMxy:
> WLEVxy:=convert(WLExy,vector):
> ddWMLxy:=matrix(1,Lxy,WLEVxy):
```

for i to 1 do for j to Lxy do ddWMLxy[i,j] := evalm(diff(ddWMLxy[i,j],x,y)) end end;

```
> ddWLxy:=ddWMLxy:
> BddWMLxy:=stackmatrix(ddWLxx,ddWLyy,2*ddWLxy):
> KELxy:=evalm(transpose(BddWMLxy)&*E1&*(BddWMLxy)):
```

for i to Lxy do for j to Lxy do KELxy[i,j]:=value(Doubleint(KELxy[i,j],x=0 .. a,y=0..b)) end end;

```
> K[E[LL]]:=KELxy:
```

B.1.9
Utilizando as funções: convencionais, adicionais internas e de contorno

Para a matriz de massa

Utilizando as funções convencionais e adicionais

```
> WMCA:=WC:
> WMAC:=WAxy:
> KWMCA:=evalm(rho*t*WMCA&*transpose(WMAC)):
```

for i to 16 do for j to Axy do KWMCA[i,j] :=value(Doubleint(KWMCA[i,j],x = 0 .. a,y=0..b)) end end;

```
> K[M[CA]]:=KWMCA:
```

Utilizando as funções convencionais e de lado

```
> WMCL:=WC:
> WMLC:=WLMxy:
> KWMCL:=evalm(rho*t*WMCL&*transpose(WMLC)):
```

for i to 16 do for j to Lxy do KWMCL[i,j]:=value(Doubleint(KWMCL[i,j],x = 0 .. a,y=0..b)) end end;

```
> K[M[CL]]:=KWMCL:
```

Utilizando as funções adicionais e de lado

```
> WMAL:=WAxy:
> WMLA:=WLMxy:
> KWMAL:=evalm(rho*t*WMAL&*transpose(WMLA)):
```

for i to Axy do for j to Lxy do KWMAL[i,j]:=value(Doubleint(KWMAL[i,j],x = 0 .. a,y=0..b)) end end;

```
> K[M[AL]]:=KWMAL:
```

Para a matriz de rigidez geométrica

Utilizando as funções convencionais e adicionais

```
> WGCA:=BdWCxy:
> WGAC:=dWAGxy:
> KGCA:=evalm(WGCA&*Nxy&*transpose(WGAC)):
```

for i to 16 do for j to Axy do KGCA[i,j] :=value(Doubleint(KGCA[i,j],x = 0 .. a,y=0..b)) end end;

```
> K[G[CA]]:=KGCA:
```

Utilizando as funções convencionais e de lado

```
> WGCL:=BdWCxy:
> WGLC:=dWGL:
> KGCL:=evalm(WGCL&*Nxy&*transpose(WGLC)):
```

for i to 16 do for j to Lxy do KGCL[i,j] :=value(Doubleint(KGCL[i,j],x = 0 .. a,y=0..b)) end end;

```
> K[G[CL]]:=KGCL:
```

Utilizando as funções adicionais e de lado

```
> WGAL:=dWAGxy:
> WGLA:=dWGL:
> KGAL:=evalm(WGAL&*Nxy&*transpose(WGLA)):
```

for i to Axy do for j to Lxy do KGAL[i,j] :=value(Doubleint(KGAL[i,j],x = 0 .. a,y=0..b)) end end;

```
> K[G[AL]]:=KGAL:
```

Matriz de rigidez elástica

Utilizando as funções convencionais e adicionais

```
> WECA:=BddWC:
> WEAC:=ddWAEprov:
> KECA:=evalm(transpose(WECA)&*E1&*WEAC):
```

for i to 16 do for j to Axy do KECA[i,j]:=value(Doubleint(KECA[i,j],x=0 .. a,y=0..b)) end end;

```
> K[E[CA]]:=KECA:
```

Utilizando as funções convencionais e de lado

```
> WECL:=BddWC:
> WELC:=BddWMLxy:
> KECL:=evalm(transpose(WECL)&*E1&*WELC):
```

for i to 16 do for j to Lxy do KECL[i,j]:=value(Doubleint(KECL[i,j],x=0 .. a,y=0..b)) end end;

```
> K[E[CL]]:=KECL:
```

Utilizando as funções adicionais e a de lado

```
> WEAL:=ddWAEprov:
> WELA:=BddWMLxy:
> KEAL:=evalm(transpose(WEAL)&*E1&*WELA):
```

for i to Axy do for j to Lxy do KEAL[i,j]:=value(Doubleint(KEAL[i,j],x=0 .. a,y=0..b)) end end;

```
> K[E[AL]]:=KEAL:
```

B.1.10
Matriz de massa final

```
> ab:=concat(K[M[CC]],K[M[CA]],K[M[CL]]):
> cd:=concat(transpose(K[M[CA]]),K[M[AA]],K[M[AL]]):
> ef:=concat(transpose(K[M[CL]]),transpose(K[M[AL]]),K[M[LL]]):
> K[Massa]:=(stackmatrix(ab,cd,ef)):
> for i to TOT do
> if (RestC[i]>0) then
> for j to TOT do
> K[Massa][i,j]:=0;
> K[Massa][j,i]:=0 ; end do;
> K[Massa][i,i]:=1;end if;
> end do;
```

B.1.11
Matriz de rigidez geométrica final

```
> gh:=concat(K[G[CC]],K[G[CA]],K[G[CL]]):
> ij:=concat(transpose(K[G[CA]]),(K[G[AA]]),K[G[AL]]):
> kl:=concat(transpose(K[G[CL]]),transpose(K[G[AL]]),K[G[LL]]):
> K[Geométrica]:=(stackmatrix(gh,ij,kl)):
> for i to TOT do
> if (RestC[i]>0) then
> for j to TOT do
> K[Geométrica][i,j]:=0;
> K[Geométrica][j,i]:=0 ; end do;
> K[Geométrica][i,i]:=1;end if;
> end do;
```

B.1.12
Matriz de rigidez elástica final

```
> mn:=concat(K[E[CC]],K[E[CA]],K[E[CL]]):
> pu:=concat(transpose(K[E[CA]]),K[E[AA]],K[E[AL]]):
> qr:=concat(transpose(K[E[CL]]),transpose(K[E[AL]]),K[E[LL]]):
> K[Elástica]:=(stackmatrix(mn,pu,qr)):
```

Agora estamos fazendo a verificação de quais nós e lados estão presos

```
> for i to TOT do
> if (RestC[i]>0) then
> for j to TOT do
> K[Elástica][i,j]:=0;
> K[Elástica][j,i]:=0 ; end do;
> K[Elástica][i,i]:=1;end if;
> end do;
```

B.1.13
Carga crítica de flambagem

```
> M:=evalm(-inverse(K[Elástica])&*(K[Geométrica])):
> Pc1:=sort([eigenvalues(M)]):
> Pc11:=0:
> for i from 1 to TOT do
> if (Pc1[i]<>-1.) then
> if (abs(Pc1[i])>abs(Pc11)) then
> Pc11:=evalf(Pc1[i])
> end if; end if;
> end;
> Pc11:
> Pc:=1/Pc11;
```

Segunda carga crítica

```
> Pc0:=Pc1[38]:
```

```
> Pc2:=1/Pc0:
```

Terceira carga crítica

```
> Pc03:=Pc1[39]:
> Pc3:=1/Pc03:
```

Cálculo do Autovetor referente a carga crítica

```
> xd:=sort([eigenvectors(M)]):
> TOT2:=TOT1+1: (Foi somado 1 por causa do autovalor '-1')
> Digits:=6:
> for i from 1 to TOT2 do
> if (xd[i][1]=evalf(Pc11)) then
> xd1:=xd[i];
> end if;
> end;
> print(xd1);
> xd2:=sort([eigenvectors(M)]):
> Digits:=5:
> for i from 1 to TOT2 do
> if (xd2[i][1]=evalf(Pc0)) then
> xd3:=xd2[i];
> end if; end;
> xd4:=sort([eigenvectors(M)]):
> Digits:=5:
> for i from 1 to TOT2 do
> if (xd4[i][1]=evalf(Pc03)) then
> xd5:=xd4[i];
> end if; end;
> fmplt:=stackmatrix(WC,WAxy,WLMxy):
```

Modos de Flambagem

```
> xd11:=xd1[3][1];
> ModFun:=evalm(transpose(fmplt)&*xd11):
> Modo:=ModFun[1]:
> plot3d(Modo,x=0..a,y=0..b);
> xd22:=xd3[3][1]:
> ModFun2:=evalm(transpose(fmplt)&*xd22):
> Modo2:=ModFun2[1]:
> xd33:=xd5[3][1]:
> ModFun3:=evalm(transpose(fmplt)&*xd33):
> Modo3:=ModFun3[1]:
```

B.1.14
Freqüências naturais de uma placa

```
> FREQUENCIA:=[(eigenvalues(K[Elástica],K[Massa]))]:
```

FREQUENCIA2:=sort(map(FREQUENCIA->
FREQUENCIA^(1/2),FREQUENCIA)):