6
Referências Bibliográficas

ABNT – ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6118:2003: Projeto de estruturas de concreto – procedimento.** 2004.

ASSAN, Aloísio E.. **Métodos dos elementos finitos: primeiros passos /**. 2ºed. Campinas, SP: Editora da Unicamp, 2003. 298p.

AZEVEDO, A. F. M.. **Método dos Elementos Finitos.** Faculdade de Engenharia da Universidade do Porto. 2003.

BATHE, K. J.. **Finite element proceddures /**. New Jersey: Prentice-hall, 1996, 1037p.

BAZANT, Z. P.; CEDOLIN, L.. **Stability of structures.** 1991.

COOK, R. D.; MALKUS, D. S.; PLESHA, M. E.. **Concepts and applications of finite element analysis.** Third edition. 1989.

FARSHAD, Mehdi. **Stability of structures /**. (Developments in civil engineering: vol. 43). Elsevier Science B.V., 1994, 425p.

FELIPPA, Carlos A.. **Introdution to finite element method /**. Bouldes, Colorado. 2004.

FONSECA, J.. **Ferramentas de simulação em Mecânica: elementos finitos.** 2002.

GIANNINI, Lilian Dutra.. **Modelo de elementos finitos para estabilidade de perfis de paredes finas.** Dissertação de Mestrado PUC-Rio, Rio de Janeiro, 1990.

JAREK, Amanda.. **Modelo computacional para flambagem de placas.** Dissertação de Mestrado PUC-Rio, Rio de Janeiro, 2007.

LAGES, Eduardo Nobre.. **Formulação hierárquica-espectral de elementos finitos.** Dissertação de Mestrado PUC-Rio, Rio de Janeiro, 1992.

LEFÈVRE, Y., NEVES, C.G.C., SADOWSKI, N. CARLSON R.. **Caractérisation des vibrations d'origine électromagnetique.** Proceedings of the International Seminar on Vibrations and Acoustic Noise of Eletric Machinery, 15-16 Maio, Béthune (França), 1998, pp.111-115.

LIMA, L. R. O.. Flambagem de Colunas - Parte I. 2003. Disponível em: <http://openlink.br.inter.net/lucianolima/rm4.htm>. Acesso em: 07 abr 2007.

MARQUES, G. C. S. C. (2006) **Estudos e desenvolvimento de código computacional baseado no método dos elementos finitos para análise dinâmica não linear geométrica de sólidos bidimensionais.** Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Carlos, São Carlos, 2006.

MEIROVITCH, Leonard. **Elements of vibration analysis**, Mc-Graw Hill, 1986.

OLIVEIRA, Afonso Henrrique de.. **Avaliação de cargas críticas de estruturas planas e axissimétricas sujeitas a dano e fissuração.** Dissertação de Mestrado PUC-Rio, Rio de Janeiro, 1990.

POPOV, E. P.. **Resistência dos Materiais.** Prentice-Hall, New Jersey, 1978.

REIS, A. e CAMOTIM, D., **Estabilidade estrutural**, Mc-Graw Hill, 2000.

RIOS, Roberto Domingo. **Aplicação do método dos elementos discretos em estruturas de concreto.** 2002. Tese (Doutorado em Engenharia Civil) - Universidade Federal do Rio Grande do Sul . Orientador: Jorge Daniel Riera

SILVA, R.; SOARES, W.. **Bifurcação do equilíbrio em pórticos planos.** PUC-Rio, Departamento de Engenharia Civil, 1974.

TIMOSHENKO, S. P.; GERE, J. M.. **Mecânica dos sólidos**, Livros Técnicos e Científicos, 1984.

TIMOSHENKO, S. P.; GERE, J. M.. **Theory of elastic stability**, Second edition,

McGraw-Hill, 1961.

TIMOSHENKO, S. P.; GOODIER, J. N.. **Theory of elasticity**, Third edition, McGraw-Hill, 1970.

VENANCIO, F. F.. **Análise matricial de estruturas (estática, estabilidade, dinâmica).** Almeida Neves-Editores, LTDA, Rio de Janeiro, 1975.

WEAVER, William Jr.; JOHNSTON, Paul R.. **Finite elements for structural analysis /**. Englewood Cliffs, N. J.: Prentice-Hall, c1984. 403p.

WEAVER, William Jr.; TIMOSHENKO, S. P.; YOUNG, D. H.. **Vibration Problems in Engeineering /**. Fifth edition. Wiley Intercience. c1989.

A. Elaboração da função da parábola do diagrama tensão-deformação

Mostra-se a seguir a função da parábola do diagrama tensão-deformação para considerar o efeito do material.

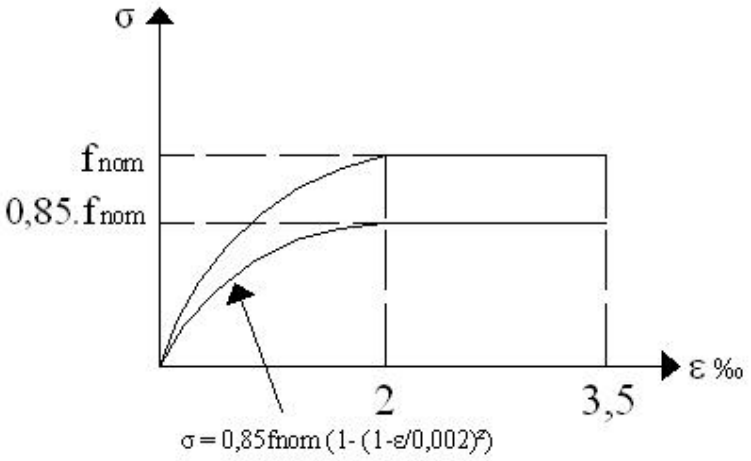

Figura A-1 – Diagrama tensão-deformação idealizado.

A parábola inicia na coordenada (0;0) e termina na coordenada (0,002;0,85f_{nom}), com isso pode-se usar a equação da parábola.

$$\sigma = a + b\varepsilon + c\varepsilon^2$$

$$\frac{d\sigma}{d\varepsilon} = 0 \therefore b + 2.c.\varepsilon = 0 \qquad \text{A-1}$$

$$\Rightarrow (0,0) \mapsto 0 = a + b.0 + c.\varepsilon^2 \rightarrow a = 0$$

$$\Rightarrow (2;0{,}85 f_{nom}) \mapsto \begin{cases} 0{,}85 f_{nom} = 0 + b.(0{,}002) + c.(0{,}002)^2 \\ 0 = b + 2.c.(0{,}002) \therefore b = -2.c.0{,}002 \end{cases}$$

$$\rightarrow 0{,}85.f_{nom} = -2.c.(0{,}002)^2 + c.(0{,}002)^2$$

$$0{,}85.f_{nom} = -c.(0{,}002)^2$$

$$c = -\frac{0{,}85 f_{nom}}{(0{,}002)^2} \rightarrow b = 2.0{,}85 f_{nom}$$

$$\sigma = 2.0{,}85.f_{nom}.\varepsilon - \frac{0{,}85 f_{nom}}{(0{,}002)^2}\varepsilon \Rightarrow \boxed{\sigma = 0{,}85.f_{nom}.\left[1 - \left(1 - \frac{\varepsilon}{0{,}002}\right)^2\right]}$$

Como se queria demonstrar.

B.

Trecho do programa com Elemento de Barber

A seguir, apresentam-se trechos do programa utilizado para obter os valores de cargas críticas, freqüências naturais, modos de flambagem, modos de vibração e carga crítica considerando dano no material.

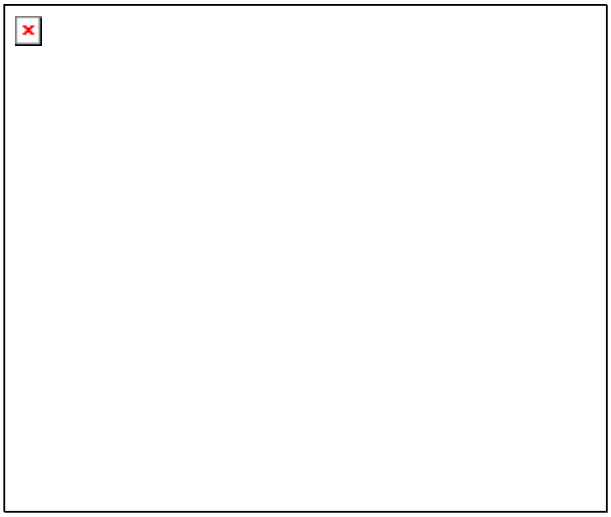

Figura B-1 – Elemento de Barber.

- **Entrada de Dados**

 # Características do Elemento e Material

> t; # Espessura

> a; b; # Dimensões apresentadas na figura acima;

> nu:=0.; # Coeficiente de Poisson

> rho;

> fnom;

> ME:=850*fnom;

> sx:=0: sy:=-1: sxy:=0:

> Ax:=2; Ay:=2; # Número de funções adicionais a serem adicionadas ao modelo

> Lx:=2; Ly:=2; # Número de funções de lado a serem adicionadas ao modelo

- **Condições de Contorno**

 # Condições de Contorno para Cada Nó (Equações Convencionais)

Obs.: Para restringir um tipo de deslocamento atribui-se um valor de uma mola cuja rigidez seja muito grande

```
> Axy:=2*Ax*Ay: #Observe que os deslocamentos são 'u' e 'v', por isto está multiplicado por 2.
> Lxy:=2*(2*Lx+2*Ly):
> TOT:=16+Axy+Lxy:
> for i to TOT do
> RestC[i]:=0;end:
> RestC[1]:= 0e20:  RestC[2]:=0e20: RestC[3]:=0e20:  RestC[4]:=0e20: # Nó 1
> RestC[5]:=0e20:  RestC[6]:=0e20:  RestC[7]:=0e20:  RestC[8]:=0e20: # Nó 2
> RestC[9]:=1e20:  RestC[10]:=1e20:RestC[11]:=1e20:RestC[12]:=1e20: # Nó 3
> RestC[13]:=1e20:RestC[14]:=1e20:RestC[15]:=1e20:RestC[16]:=1e20: # Nó 4
```

 # Condições de Contorno (Equações Adicionais Externa ou de lado)

Para prender um dos lados daremos a ele um valor de uma mola cuja rigidez seja muito grande

```
> Lado[1]:=0e20: #prende em x entre nós 1 e 2
> Lado[2]:= 1e20: #prende em x entre nós 3 e 4
> Lado[3]:=0e20: #prende em y entre nós 1 e 3
> Lado[4]:=0e20: #prende em y entre nós 2 e 4
```

 # Consideração das Condições de Contorno

```
> for i to (Lx+Ly) do
> RestC[16+Axy+i]:=Lado[1]:
> RestC[16+Axy+(Lx+Ly)+i]:=Lado[2]:
> RestC[16+Axy+2*(Lx+Ly)+i]:=Lado[3]:
> RestC[16+Axy+3*(Lx+Ly)+i]:=Lado[4]:
> end:
> TOT1:=0: #TOT1 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;
```

- **Funções Convencionais**

 # Nó 1

```
> f1:=1/8*(1-xi)*(1+eta)^2*(2-eta):
> f2:=1/8*(1-xi)^2*(2+xi)*(1+eta):
> f3:=1/8*(1-xi)^2*(1+xi)*(1+eta)*a:
> f4:=1/8*(1-xi)*(1+eta)^2*(1-eta)*b:
```

 # Nó 2

```
> f5:=1/8*(1+xi)*(1+eta)^2*(2-eta):
> f6:=1/8*(1+xi)^2*(2-xi)*(1+eta):
> f7:=-1/8*(1+xi)^2*(1-xi)*(1+eta)*a:
> f8:=1/8*(1+xi)*(1+eta)^2*(1-eta)*b:
```

 # Nó 3

```
> f9:=1/8*(1-xi)*(1-eta)^2*(2+eta):
> f10:=1/8*(1-xi)^2*(2+xi)*(1-eta):
> f11:=1/8*(1-xi)^2*(1+xi)*(1-eta)*a:
> f12:=-1/8*(1-xi)*(1-eta)^2*(1+eta)*b:
```

 # Nó 4

```
> f13:=1/8*(1+xi)*(1-eta)^2*(2+eta):
> f14:=1/8*(1+xi)^2*(2-xi)*(1-eta):
> f15:=-1/8*(1+xi)^2*(1-xi)*(1-eta)*a:
> f16:=-1/8*(1+xi)*(1-eta)^2*(1+eta)*b:
> # Função Convencional --> FC
> FC:=evalm(matrix(2,16,[f1,0,0,f4,f5,0,0,f8,f9,0,0,f12,f13,0,0,f16,0,f2,f3,0,0,f6,
f7,0,0,f10,f11,0,0,f14,f15,0]));
```

- **Funções Adicional Interna**

```
> WAx:=1/4*(nx-2)*(1+(-1)^nx)+(1/4*(nx-3)*(1-(-1)^nx))*xi+(-1/4*nx*(1+
(-1)^nx))*xi^2+(-1/4*(nx-1)*(1-(-1)^nx))*xi^3+(xi)^nx;
> WAy:=1/4*(ny-2)*(1+(-1)^ny)+(1/4*(ny-3)*(1-(-1)^ny))*eta+(-1/4*ny*(1+
(-1)^ny))*eta^2+(-1/4*(ny-1)*(1-(-1)^ny))*eta^3+(eta)^ny;
```

 # Função Adicional na direção x

```
> WAVxx:=subs(nx=NX,WAx):
> WAVx:=vector(Ax):
```

```
> for i from 1 to Ax do
> NX:=i+3;
> WAVx[i]:=WAVxx;
> od:
> WAMx:=convert(WAVx,matrix);
```

 # Função Adicional na direção y

```
> WAVyy:=subs(ny=NY,WAy):
> WAVy:=vector(Ay):
> for i from 1 to Ay do
> NY:=i+3;
> WAVy[i]:=WAVyy;
> od:
> WAMy:=convert(WAVy,matrix);
```

 # Fazendo a Multiplicação das funções x por y

```
> WAxy1:=evalm(WAMx&*transpose(WAMy)):
> WAxy2:=convert(WAxy1,vector):
> WAxy:=matrix(Axy/2,1,WAxy2):#
> # Funções Adicionais --> FA
> FA:=matrix(2,Axy,0):
> n:=0:
> for j from 1 to Axy by 2 do
>   FA[1,j]:=WAxy[j-n,1]:
>   FA[2,j+1]:=WAxy[j-n,1]: n:=n+1:
> end:
```

- **<u>Funções Adicionais Externas ou de Lado</u>**

```
> L[x1]:=1/2*(1-xi):
> L[x2]:=1/2*(1+xi):
> L[y1]:=1/2*(1+eta):
> L[y2]:=1/2*(1-eta):
```

 # Função de Lado na direção x

```
> WLVx:=subs(nx=mx,WAx):
> WLVxx:=Vector(Lx):
> for i from 1 to Lx do
```

```
> mx:=i+3;
> WLVxx[i]:=WLVx;
> end:
> WLVx1:=convert(WLVxx,vector):
> WLMx:=matrix(Lx,1,WLVx1):
> # Função de Lado na direção y
> WLVy:=subs(ny=my,WAy):
> WLVyy:=Vector(Ly):
> for i from 1 to Ly do
> my:=i+3;
> WLVyy[i]:=WLVy;
> end:
> WLVy1:=convert(WLVyy,vector):
> WLMy:=matrix(Ly,1,WLVy1);
```

 # Fazendo a Multiplicação das funções x por y

```
> WLMx1:=simplify(evalm(WLMx*L[y1])):
> WLMx2:=simplify(evalm(WLMx*L[y2])):
> WLMy1:=simplify(evalm(WLMy*L[x1])):
> WLMy2:=simplify(evalm(WLMy*L[x2])):
> WLMxy:=stackmatrix(WLMx1,WLMx2,WLMy1,WLMy2);
> # Funções de Lado --> FL
> FL:=matrix(2,Lxy,0):
> n:=0:
> for j from 1 to Lxy by 2 do
>   FL[1,j]:=WLMxy[j-n,1]:
>   FL[2,j+1]:=WLMxy[j-n,1]: n:=n+1:
> end:
> print(FL);
> FTotal:=concat(FC,FA,FL):
```

- **Matriz de Rigidez Elástica**

 # Funções Convencional

 # Derivada Primeira em relação à x

```
> FCx:=matrix(1,16):
```

```
> for j to 16 do FCx[1,j]:=evalm(diff(FC[1,j],xi)/a) end:
# Derivada Primeira em relação à y
> FCy:=matrix(1,16):
> for j to 16 do FCy[1,j]:=evalm(diff(FC[2,j],eta)/b) end:
# Matriz BC
>FCxy:=matrix(1,16,[diff(f1,eta)/b,diff(f2,xi)/a,diff(f3,xi)/a,diff(f4,eta)/b,
diff(f5,eta)/b,diff(f6,xi)/a,diff(f7,xi)/a,diff(f8,eta)/b,diff(f9,eta)/b,diff(f10,xi)/a,
diff(f11,xi)/a,diff(f12,eta)/b,diff(f13,eta)/b,diff(f14,xi)/a,diff(f15,xi)/a,
diff(f16,eta)/b]):
> BC1:=matrix(3,16):
> BC1:=stackmatrix(FCx,FCy,FCxy):
# Funções Adicionais Internas
# Derivada Primeira em relação à x
> FAx:=matrix(1,Axy):
> for j to Axy do FAx[1,j]:=evalm(diff(FA[1,j],xi)/a) end:
# Derivada Primeira em relação à y
> FAy:=matrix(1,Axy):
> for j to Axy do FAy[1,j]:=evalm(diff(FA[2,j],eta)/b) end:
# Matriz BA
> for i from 1 to Axy/2 do
> dWAxy1[i,1]:=evalm(diff(WAxy[i,1],eta)/b);
> dWAxy2[i,1]:=evalm(diff(WAxy[i,1],xi)/a);
> end:
> FAxy:=matrix(1,Axy,0): n:=0:
> for j from 1 to Axy by 2 do
>  FAxy[1,j]:=dWAxy1[j-n,1];
>  FAxy[1,j+1]:=dWAxy2[j-n,1]; n:=n+1;
> end:
> BA1:=matrix(3,Axy):
> BA1:=simplify(stackmatrix(FAx,FAy,FAxy)):
# Funções de Lado
# Derivada Primeira em relação à x
> FLx:=matrix(1,Lxy):
> for j to Lxy do FLx[1,j]:=evalm(diff(FL[1,j],xi)/a) end:
```

 # Derivada Primeira em relação à y

```
> FLy:=matrix(1,Lxy):
> for j to Lxy do FLy[1,j]:=evalm(diff(FL[2,j],eta)/b) end:
```

 # Matriz BL

```
> for i from 1 to Lxy/2 do
> dWLxy1[i,1]:=evalm(diff(WLMxy[i,1],eta)/b);
> dWLxy2[i,1]:=evalm(diff(WLMxy[i,1],xi)/a);
> end:
> FLxy:=matrix(1,Lxy,0): n:=0:
> for j from 1 to Lxy by 2 do
>  FLxy[1,j]:=dWLxy1[j-n,1];
>  FLxy[1,j+1]:=dWLxy2[j-n,1]; n:=n+1;
> end:
> BL1:=matrix(3,Lxy):
> BL1:=stackmatrix(FLx,FLy,FLxy):
> BTotal:=concat(BC1,BA1,BL1):#Junção das Matrizes "B"
```

 # <u>MATRIZ DE RIGIDEZ ELÁSTICA</u>

```
> KCe:=evalm(transpose(BC1)&*Eo&*BC1):
> for i to 16 do
> for j to 16 do
> KCe[i,j]:=value(Doubleint(KCe[i,j]*a*b*t,xi=-1..1,eta=-1..1));
> end:end:
> KAe:=evalm(transpose(BA1)&*Eo&*BA1):
> for i to Axy do
> for j to Axy do
> KAe[i,j]:=value(Doubleint(KAe[i,j]*a*b*t,xi=-1..1,eta=-1..1));
> end:end:
> KLe:=evalm(transpose(BL1)&*Eo&*BL1):
> for i to Lxy do
> for j to Lxy do
> KLe[i,j]:=value(Doubleint(KLe[i,j]*a*b*t,xi=-1..1,eta=-1..1));
> end:end:
> KCAe:=evalm(transpose(BC1)&*Eo&*BA1):
> for i to 16 do
```

```
> for j to Axy do
> KCAe[i,j]:=value(Doubleint(KCAe[i,j]*a*b*t,xi=-1..1,eta=-1..1));
> end:end:
> KCLe:=evalm(transpose(BC1)&*Eo&*BL1):
> for i to 16 do
> for j to Lxy do
> KCLe[i,j]:=value(Doubleint(KCLe[i,j]*a*b*t,xi=-1..1,eta=-1..1));
> end:end:
> KALe:=evalm(transpose(BA1)&*Eo&*BL1):
> for i to Axy do
> for j to Lxy do
> KALe[i,j]:=value(Doubleint(KALe[i,j]*a*b*t,xi=-1..1,eta=-1..1));
> end:end:
> mn:=concat(KCe,KCAe,KCLe):
> pu:=concat(transpose(KCAe),KAe,KALe):
> qr:=concat(transpose(KCLe),transpose(KALe),KLe):
> Ke:=stackmatrix(mn,pu,qr):
> for i to TOT do
>       if (RestC[i]>0)  then
>             for j to TOT do
>                   Ke[i,j]:=0;
>                   Ke[j,i]:=0; end do;
>       Ke[i,i]:=1;end if;
> end do;
```

- **Matriz de Massa**

```
> KCm:=evalm(rho*transpose(FC)&*FC):
> for i to 16 do for j to 16 do
> KCm[i,j]:=value(Doubleint(KCm[i,j]*a*b*t,xi=-1..1,eta=-1..1));
> end:end:
> KAm:=evalm(rho*transpose(FA)&*FA):
> for i to Axy do for j to Axy do
> KAm[i,j]:=value(Doubleint(KAm[i,j]*a*b*t,xi=-1..1,eta=-1..1));
> end:end:
```

```
> KLm:=evalm(rho*transpose(FL)&*FL):
> for i to Lxy do for j to Lxy do
> KLm[i,j]:=value(Doubleint(KLm[i,j]*a*b*t,xi=-1..1,eta=-1..1));
> end:end:
```

 # <u>MATRIZ DE MASSA</u>

```
> KCAm:=evalm(rho*transpose(FC)&*FA):
> for i to 16 do for j to Axy do
> KCAm[i,j]:=value(Doubleint(KCAm[i,j]*a*b*t,xi=-1..1,eta=-1..1));
> end; end;
> KCLm:=evalm(rho*transpose(FC)&*FL):
> for i to 16 do for j to Lxy do
> KCLm[i,j]:=value(Doubleint(KCLm[i,j]*a*b*t,xi=-1..1,eta=-1..1));
> end; end;
> KALm:=evalm(rho*transpose(FA)&*FL):
> for i to Axy do for j to Lxy do
> KALm[i,j]:=value(Doubleint(KALm[i,j]*a*b*t,xi=-1..1,eta=-1..1));
> end; end;
> ab:=concat(KCm,KCAm,KCLm):
> cd:=concat(transpose(KCAm),KAm,KALm):
> ef:=concat(transpose(KCLm),transpose(KALm),KLm):
> Km:=stackmatrix(ab,cd,ef):
> for i to TOT do
>       if (RestC[i]>0)  then
>             for j to TOT do
>                   Km[i,j]:=0;
>                   Km[j,i]:=0 ; end do;
>       Km[i,i]:=1; end if;
> end do;
```

- **<u>Matriz de Tensão</u>**

```
> sigma1:=matrix(3,1,[sx,sy,sxy]);
> S2:=matrix(2,2,[sigma1[1,1],sigma1[3,1],sigma1[3,1],sigma1[2,1]]):
>S:=matrix(4,4,[S2[1,1],S2[1,2],0,0,S2[2,1],S2[2,2],0,0,0,0,S2[1,1],S2[1,2],0,0,
S2[2,1],S2[2,2]]);
```

- **Matriz de Rigidez Geométrica**

 # Funções Convencionais

```
> FC1:=matrix(2,16):
> for i to 2 do for j to 16 do
> FC1[i,j]:=subs(xi=x/a,eta=y/b,FC[i,j]);
> end; end;
> uCx:=matrix(1,16):
> for j to 16 do uCx[1,j]:=evalm((FC1[1,j])) end:
> uCy:=matrix(1,16):
> for j to 16 do uCy[1,j]:=evalm(-diff(FC1[1,j],y)) end:
> vCx:=matrix(1,16):
> for j to 16 do vCx[1,j]:=evalm(diff(FC1[2,j],x)) end:
> vCy:=matrix(1,16):
> for j to 16 do vCy[1,j]:=evalm((FC1[2,j])) end:
> GC:=stackmatrix(uCx,vCy,vCx,uCy):
> KCg:=evalm(transpose(GC)&*S&*GC):
> for i to 16 do
> for j to 16 do
> KCg[i,j]:=value(Doubleint(KCg[i,j]*t,x=-a..a,y=-b..b));
> end:end:
```

#Funções Adicionais Interna

```
> FA1:=matrix(2,Axy):
> for i to 2 do for j to Axy do
> FA1[i,j]:=subs(xi=x/a,eta=y/b,FA[i,j]);
> end; end;
> uAx:=matrix(1,Axy):
> for j to Axy do uAx[1,j]:=evalm(diff(FA1[1,j],x)) end:
> uAy:=matrix(1,Axy):
> for j to Axy do uAy[1,j]:=evalm(diff(FA1[1,j],y)) end:
> vAx:=matrix(1,Axy):
> for j to Axy do vAx[1,j]:=evalm(diff(FA1[2,j],x)) end:
> vAy:=matrix(1,Axy):
> for j to Axy do vAy[1,j]:=evalm(diff(FA1[2,j],y)) end:
```

```
> GA:=stackmatrix(uAx,uAy,vAx,vAy):
> KAg:=evalm(transpose(GA)&*S&*GA):
> for i to Axy do
> for j to Axy do
> KAg[i,j]:=value(Doubleint(KAg[i,j]*t,x=-a..a,y=-b..b));
> end:end:
```

#Funções de Lado

```
> FL1:=matrix(2,Lxy):
> for i to 2 do for j to Lxy do
> FL1[i,j]:=subs(xi=x/a,eta=y/b,FL[i,j]);
> end; end;
> uLx:=matrix(1,Lxy):
> for j to Lxy do uLx[1,j]:=evalm(diff(FL1[1,j],x)) end:
> uLy:=matrix(1,Lxy):
> for j to Lxy do uLy[1,j]:=evalm(diff(FL1[1,j],y)) end:
> vLx:=matrix(1,Lxy):
> for j to Lxy do vLx[1,j]:=evalm(diff(FL1[2,j],x)) end:
> vLy:=matrix(1,Lxy):
> for j to Lxy do vLy[1,j]:=evalm(diff(FL1[2,j],y)) end:
> GL:=stackmatrix(uLx,uLy,vLx,vLy):
> KLg:=evalm(transpose(GL)&*S&*GL):
> for i to Lxy do
> for j to Lxy do
> KLg[i,j]:=value(Doubleint(KLg[i,j]*t,x=-a..a,y=-b..b));
> end:end:
```

 # <u>MATRIZ GEOMÉTRICA</u>

```
> KCAg:=evalm(transpose(GC)&*S&*GA):
> for i to 16 do
> for j to Axy do
> KCAg[i,j]:=value(Doubleint(KCAg[i,j]*t,x=-a..a,y=-b..b));
> end:end:
> KCLg:=evalm(transpose(GC)&*S&*GL):
> for i to 16 do
> for j to Lxy do
```

```
> KCLg[i,j]:=value(Doubleint(KCLg[i,j]*t,x=-a..a,y=-b..b));
> end:end:
> KALg:=evalm(transpose(GA)&*S&*GL):
> for i to Axy do
> for j to Lxy do
> KALg[i,j]:=value(Doubleint(KALg[i,j]*t,x=-a..a,y=-b..b));
> end:end:
> gh:=concat(KCg,KCAg,KCLg):
> ij:=concat(transpose(KCAg),KAg,KALg):
> kl:=concat(transpose(KCLg),transpose(KALg),KLg):
> Kg:=stackmatrix(gh,ij,kl):
> for i to TOT do
>       if (RestC[i]>0)  then
>             for j to TOT do
>                   Kg[i,j]:=0;
>                   Kg[j,i]:=0 ; end do;
>       Kg[i,i]:=1;end if;
> end do;
```

- **Carga Crítica**

```
> M:=evalm(-inverse(Ke)&*(Kg)):
> Pc1:=([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;
```

- **Modos de Flambagem**

 # Cálculo do Autovetor

```
> xd:=([eigenvectors(M)]):
> TOT11:=TOT1+1: #Somei 1 pq tem o autovalor '-1'
> for i from 1 to TOT11 do
>       Digits:=6:
>       if (xd[i][1]=evalf(Pc11)) then
>             xd1:=xd[i];
>       end if:
> end:
> xd11:=xd1[3][1]:
> ModFun:=evalm(FTotal&*xd11);
```

 # 1º Modo na direção 'u'

```
> Modou:=subs(xi=x/(a),eta=y/(b),ModFun[1]):
> plot3d(Modou,x=-a..a,y=-b..b,axes=boxed);
```

 # 1º Modo na direção 'v'

```
> Modov:=subs(xi=x/(a),eta=y/(b),ModFun[2]):
> plot3d(Modov,x=-a..a,y=-b..b,axes=boxed);
```

- **Freqüência Natural**

 # Freqüências Numéricas

```
> interface(displayprecision=25): Digits:=25:
> Freq:=evalm(inverse(Ke)&*Km):
> Freq1:=sort([eigenvalues(Freq)]);
> Freq11:=0:
> for i from 1 to TOT do
>       if (Freq1[i]<>1.) then
>       if ((Freq1[i])>(Freq11)) then
>             Freq11:=evalf(Freq1[i])
>       end if; end if;
> end;
> FrequenciaModo1:=1/sqrt((Freq11));
> Freq12:=0:
> for i from 1 to TOT do
>       if (Freq1[i]<>Freq11) then
>       if (Freq1[i]<>1.) then
```

```
>       if ((Freq1[i])>(Freq12)) then
>               Freq12:=evalf(Freq1[i])
>       end if; end if; end if;
> end;
> FrequenciaModo2:=1/sqrt((Freq12));
> Freq13:=0:
> for i from 1 to TOT do
>       if (Freq1[i]<>Freq11) then
>       if (Freq1[i]<>Freq12) then
>       if (Freq1[i]<>1.) then
>       if ((Freq1[i])>(Freq13)) then
>               Freq13:=evalf(Freq1[i])
>       end if; end if; end if; end if;
> end;
> FrequenciaModo3:=1/sqrt((Freq13));
```

- **# Modos de Vibração**

```
> xdF:=([eigenvectors(Freq)]):
# 1º Modo
> TOT11:=TOT1+1: #Somei 1 pq tem o autovalor '-1'
> for i from 1 to TOT11 do
>       Digits:=4:
>       if (xdF[i][1]=evalf(Freq11)) then
>               xd1F1:=xdF[i];
>       end if:
> end:
> xd11F1:=xd1F1[3][1]:
> ModFunF1:=evalm(FTotal&*xd11F1):
# 1º Modo direção 'u'
> ModouF1:=subs(xi=x/(a),eta=y/(b),ModFunF1[1]):
> plot3d(ModouF1,x=-a..a,y=-b..b,axes=boxed);
# 1º Modo direção 'v'
> ModovF1:=subs(xi=x/(a),eta=y/(b),ModFunF1[2]):
> plot3d(ModovF1,x=-a..a,y=-b..b,axes=boxed);
```

```
# 2º Modo
> TOT11:=TOT1+1: #Somei 1 pq tem o autovalor '-1'
> for i from 1 to TOT11 do
>       Digits:=5:
>       if (xdF[i][1]=evalf(Freq12)) then
>             xd1F2:=xdF[i];
>       end if:
> end:
> xd11F2:=xd1F2[3][1]:
> ModFunF2:=evalm(FTotal&*xd11F2):
# 2º Modo direção 'u'
> ModouF2:=subs(xi=x/(a),eta=y/(b),ModFunF2[1]):
> plot3d(ModouF2,x=-a..a,y=-b..b,axes=boxed);
# 2º Modo direção 'v'
> ModovF2:=subs(xi=x/(a),eta=y/(b),ModFunF2[2]):
> plot3d(ModovF2,x=-a..a,y=-b..b,axes=boxed);
# 3º Modo
> TOT11:=TOT1+1: #Somei 1 pq tem o autovalor '-1'
> for i from 1 to TOT11 do
>       Digits:=5:
>       if (xdF[i][1]=evalf(Freq13)) then
>             xd1F3:=xdF[i];
>       end if:
> end:
> xd11F3:=xd1F 3[3][1]:
> ModFunF3:=evalm(FTotal&*xd11F3):
# 3º Modo direção 'u'
> ModouF3:=subs(xi=x/(a),eta=y/(b),ModFunF3[1]):
> plot3d(ModouF3,x=-a..a,y=-b..b,axes=boxed);
# 3º Modo direção 'v'
> ModovF3:=subs(xi=x/(a),eta=y/(b),ModFunF3[2]):
> plot3d(ModovF3,x=-a..a,y=-b..b,axes=boxed);
```

- **Redução da Matriz de Rigidez Elástica**

```
> interface(displayprecision=25): Digits:=25:
# Usando o Módulo de Elasticidade Tangente--->>>Et
> sigma1:=0.85*(fnom)*(1-(1-epsilon1/0.002)^2);
> MEt1:=diff(sigma1,epsilon1);
> if (sx<>0) and (sy=0) then
>         epsilon1:=abs(sx)/(t*ME); end if:
> if (sy<>0) and (sx=0) then
>         epsilon1:=abs(sy)/(t*ME); end if:
> MEt:=evalf(MEt1); # Módulo de Elasticidade Longitudinal em kN/cm²
> #Matriz Constitutiva para o concreto fissurado
> Et:=evalm(matrix(3,3,[1,nu,0,nu,1,0,0,0,lambda])*(MEt/(1-nu^2)));
# Matriz de Rigidez Elástica Reduzida
> KCet:=evalm(transpose(BC1)&*Et&*BC1):
> for i to 16 do
> for j to 16 do
> KCet[i,j]:=value(Doubleint(KCet[i,j]*a*b*t,xi=-1..1,eta=-1..1));
 > end:end:
> KAet:=evalm(transpose(BA1)&*Et&*BA1):
> for i to Axy do
> for j to Axy do
> KAet[i,j]:=value(Doubleint(KAet[i,j]*a*b*t,xi=-1..1,eta=-1..1));
 > end:end:
> KLet:=evalm(transpose(BL1)&*Et&*BL1):
> for i to Lxy do
> for j to Lxy do
> KLet[i,j]:=value(Doubleint(KLet[i,j]*a*b*t,xi=-1..1,eta=-1..1));
 > end:end:
> KCAet:=evalm(transpose(BC1)&*Et&*BA1):
> for i to 16 do
> for j to Axy do
> KCAet[i,j]:=value(Doubleint(KCAet[i,j]*a*b*t,xi=-1..1,eta=-1..1));
 > end:end:
```

```
> KCLet:=evalm(transpose(BC1)&*Et&*BL1):
> for i to 16 do
> for j to Lxy do
> KCLet[i,j]:=value(Doubleint(KCLet[i,j]*a*b*t,xi=-1..1,eta=-1..1));
 > end:end:
> KALet:=evalm(transpose(BA1)&*Et&*BL1):
> for i to Axy do
> for j to Lxy do
> KALet[i,j]:=value(Doubleint(KALet[i,j]*a*b*t,xi=-1..1,eta=-1..1));
 > end:end:
> mnt:=concat(KCet,KCAet,KCLet):
> put:=concat(transpose(KCAet),KAet,KALet):
> qrt:=concat(transpose(KCLet),transpose(KALet),KLet):
> Ket:=stackmatrix(mnt,put,qrt):
 > for i to TOT do
 >       if (RestC[i]>0)  then
 >               for j to TOT do
 >                       Ket[i,j]:=0;
 >                       Ket[j,i]:=0 ; end do;
 >       Ket[i,i]:=1;end if;
 > end do;
```

 # Carga Crítica Reduzida

```
> Kered:=evalm(Ke-Ket): Kd:=evalm(Kg-Kered):
> Md:=evalm(-inverse(Ke)&*(Kd)):
> Pc1d:=sort([eigenvalues(Md)]);
> Pc11d:=0:
> for i from 1 to TOT do
>       if (Pc1d[i]<>-1.) then
>       if (abs(Pc1d[i])>abs(Pc11d)) then
>               Pc11d:=evalf(Pc1d[i])
>       end if; end if;
> end;
> Pc11d;
> Pcd:=1/Pc11d;
```