7
Referências Bibliográficas

ALVES, M. C. **O método híbrido dos elementos de contorno aplicado a problemas com simetria e antissimetria**. Dissertação de Mestrado - Departamento de Engenharia Civil, Pontifícia Universidade Católica do Rio de Janeiro. Rio de Janeiro. 2002.

BOARD, J.; SCHULTEN, K. The fast multipole algorithm. **Computing in Science and Engineering**, 2, 2000. 76-79.

BREBBIA, C. A. **The boundary element method for engineers**. Londres: Pentech Press, 1978.

CHAVES, R. A. P. **O método híbrido simplificado dos elementos de contorno aplicado a problemas dependentes do tempo**. Tede de Doutorado - Departamento de Engenharia Civil, Pontifícia Universidade Católica do Rio de Janeiro. Rio de Janeiro. 2003.

CRUSE, T. A.; VANBUREN, W. Three-Dimensional Elastic Stress Analysis of a Fracture Specimen with an Edge Crack. **International Journal of Fracture Mechanics**, 7, Março 1971.

DONGARRA, J.; SULLIVAN, F. The top ten algorithms of the century. **Computing in Science and Engineering**, 2, 2000. 22-23.

DUMONT, N. A. **The hybrid boundary element method**. In: BREBBIA, C.A.; WENDLAND, W.; KUHN, G., editor, BOUNDARY ELEMENTS IX, v.1, Mathematical and Computacional Aspects. Southampton: Computational Mechanics Publications, Springer-Verlag. 1987. p. 125-138.

DUMONT, N. A. **The boundary element method revisited**. Boundary Elements and Other Mesh Reduction Methods XXXII, ed C. A. Brebbia. Southampton: WITPress. 2010a. p. 227-238.

DUMONT, N. A. From the collocation boundary element method to a meshless formulation. **Mecánica Computacional Vol XXIX**, Buenos Aires, 2010b. 4635-4659.

DUMONT, N. A.; AGUILAR, C. A. The best of two worlds: The expedite boundary element method. **Engineering Structures**, 43, 2012. 235-244.

DUMONT, N. A.; MAMANI, E. Y. Generalized Westergaard stress functions as fundamental solutions. **Computer Modeling in Engineering & Sciences**, 78(2), 2011. 109-150.

GREENGARD, L. Fast Algorithms for Classical Physics. **Science**, 265, Agosto 1994.

GREENGARD, L.; ROKHLIN, V. A fast algorithm for particle simulations. **Journal of Computational Physics**, 73, 1987. 325-348.

LIU, Y. **Fast multipole boundary element method, theory and applications in engineering**. New York: Cambridge University Press, 2009.

LIU, Y. J.; NISHIMURA, N. The fast multipole boundary element method for potential problems: A tutorial. **Engineering Analysis with Boundary Elements**, 30, 2006. 371–381.

NISHIMURA, N.; YOSHIDA, K.; KOBAYASHI, S. A fast multipole boundary integral equation method for crack problems in 3D. **Engineering Analysis with Boundary Elements**, 23, 1999. 97-105.

OLIVEIRA, M. F. F. D. **Métodos de elementos de contorno convencional, híbridos e simplificados**. Dissertação de Mestrado - Departamento de Engenharia Civil, Pontifícia Universidade Católica do Rio de Janeiro. Rio de Janeiro. 2004.

PEIRCE, A. P.; NAPIER, J. A. L. A spectral multipole method for efficient solution of large-scale boundary element models in elastostatics. **International Journal for Numerical Methods in Engineering**, 38, 1995. 4009-4034.

TREVELYAN, J. **Boundary element for engineers:** theory and applications. Southampton: Computational Mechanics Publications, 1994.

YASUDA, Y.; SAKUMA, T. An effective setting of hierarchical cell structure for the fast multipole boundary element method. **Journal of Computatuional Acoustics**, 13, 2005. 47-70.

8
Apêndice 1

Nesse apêndice é apresentado um algoritmo para a formulação descrita no capítulo 4 em forma de pseudo-algoritmo. Também é apresentado um exemplo de arquivo de entrada de pontos fonte, campo e camadas de expansão que corresponde à distribuição de pontos apresentada na Figura 4. O algoritmo assume que os pontos fonte e campo estão distantes o suficiente, ou seja, uma estratégia de medição de distância deve ser previamente executada para a geração dos dados de entrada. Esta implementação foi desenvolvida em linguagem Maple® pelo autor e está disponível para consulta.

8.1. Algorithm

This is the Fast Multipole Algorithm for a General function (GFMM) in a 2D space. It may be considered as many expansions about source and field points as defined by the user.

One wants to evaluate the contribution of a given set of field points to a given set of source points. This contribution is defined by a function $f(z-z_0)$, which normally is a fundamental solution defined – that is, has global support –, in a given domain.

This algorithm is useful for solving the equation system, in the form $\mathbf{Ax}=\mathbf{b}$, which arises from the evaluation of the contributions aforementioned, when an iterative solver is used for this task. This kind of solver multiplies the matrix $\mathbf{A}$ by a certain trial vector $\mathbf{x}^{k-1}$. Usually it is necessary to assemble all the coefficients of this matrix in order to evaluate the multiplication. To avoid this cumbersome multiplication, the algorithm hereby presented multiplies a line from matrix $\mathbf{A}$ by the trial vector as it is formed, and, by doing so, the whole matrix is never assembled, saving space and processing time in the algorithm.

As this algorithm is used only for numerical validation of the GFMM, the trial vector is taken as $x_i = 1$, with $i = 1..nSourcePoints$.

8.1.1. Input data

n = number of terms in the Taylor expansion.

$Poles\_Field$ = Matrix with all the field poles. Each row stores the poles in a certain level of expansion.

$Poles\_Source$ = Matrix with all the source poles. Each row stores the poles in a certain level of expansion.

$Paths\_Field$ = Matrix of vectors. Each position of this matrix stores a vector informing which poles of the downward level this pole is referred to. Each position corresponds to a pole of Poles_Field down to row 2, as row 1 of Poles_Field stores the field points (level 0).

$Paths\_Source$ = Matrix of vectors. Each position of this matrix stores a vector informing which poles of the downward level this pole is referred to. Each position corresponds to a pole of Poles_Source down to row 2, as row 1 of Poles_Source stores the source points (level 0).

8.1.2. Output data

$AnalyticalResult$ = Vector with the result of matrix-vector multiplication $\mathbf{Ax}$ evaluated through the Fast Multipole Algorithm. Each component corresponds to a source point.

$AnalyticalResult$ = Vector with the result of matrix-vector multiplication $\mathbf{Ax}$ evaluated in the usual manner.

Error = Vector with the error between $Result[i]$ and $AnalyticalResult[i]$.

8.1.3. Preliminary evaluations for the algorithm

Read the input files through *Procedure 3.*

Evaluate matrix C through *Procedure 2.*

Evaluate vector *fac* through *Procedure 1.*

Evaluate vector *Q* through *Procedure 9.*

8.1.4. Execution line

After executing the procedures in the latter section, the algorithm is then evaluated as follows.

8.1.4.1. Evaluate the contributions of all field points to the highest level expansion pole

Compute the row dimension of $Poles\_Field$, as this indicates how many expansion levels exists. This information is stored in variable *DimP_F*.

Call *Procedure 6* with the following parameters: (*Paths_Field*, *Poles_Field*, *P_Fields*, *P_Matrix*, *C*, $DimP\_F-1$, $DimP\_F-1$, n, 1). In this calling line *P_Fields* is the resulting vector. It has (n+1) terms and represents the contributions of all field points to the higher expansion pole. The last parameter is set to 1 as this parameter indicates which position of *Paths_Field* this call has to consider, as there is only one pole at the higher expansion level, this is set to 1.

8.1.4.2. Evaluate the matrix-vector multiplication

Compute the row dimension of *Poles_Source*. Store in *DimP_S*.

Call *Procedure 7* with the following parameters: (*Paths_Source*, *Poles_Source*, *Result*, *P_Matrix*, *C*, *DimP_S* – 1, *DimP_S* – 1, n, 1, *P_Fields*, *Q*, *fac*). *Result* is the vector which stores the evaluation of the matrix-vector multiplication. Analogously to 8.1.4.1, the 9^{th} parameter is set to 1, and *P_Fields* is the vector returned in 8.1.4.1.

8.1.4.3. Evaluate the error

Evaluate the regular matrix-vector multiplication and store it in vector *AnalyticalResult*.

For i to nSourcePoints do

$$Error[i] = \sum \left| \frac{AnalyticalResult[i] - Result[i]}{AnalyticalResult[i]} \right|$$

End do (loop i)

8.1.5. Procedures referred to in the algorithm

These are the procedures which are used in the latter section.

8.1.5.1. Procedure 1

Evaluates the *fac* vector.

$fac[1] = 1$

$fac[2] = 1$

for i from 3 to n do

$fac[i] = (i-1)\, fac[i-1]$

End do (loop i)

8.1.5.2. Procedure 2

Evaluates the *C* matrix.

for i to (n+1) do

$C[i,1] = 1$

for j to (n+1) do

$C[1,i] = 1$

$C[1, j] = C[i-1, j] + C[i, j-1]$

End do (loop j)

End do (loop i)

8.1.5.3. Procedure 3

Reads the input file which contains the poles, its expansions, and the tree structure from expansion level 1 up to the highest level in a recursive manner (Level 0 is read at Procedure 4).

Checks if the current reading level is the higher one already

If (Level > nLevels) then

Exit execution

If the highest level hasn't been read yet, then reads this level and passes onto the next one

Else

read(Level Name)

nLevelpoints := read(# of points in the current Level)

for i to nLevelpoints do

read(Pole ID in its Level)

Stores the pole in the *Poles* structure (either *Source*, or *Field*)

$Poles\left[Level, i\right] = read(\%\,\text{complex\_float})$

nSights := read(# of points in the lower level "seen" by this point)

For j to *nSights* do

$Paths[Level-1, i][j] = read(\%complex\_\text{float})$

End do (loop j)

End do (loop i)

Stores in the row's last column the number of points in this level

$Poles[Level, nPoints+1] = nLevelPoints$

Climbs up to the next expansion level

Calls Procedure 3 (Level+1)

End if

8.1.5.4. Procedure 4

Reads the input file which contains the poles, its expansions, and the tree structure using Procedure 3.

read(Flag for the number of expansion levels)

nLevels := read(# of expansion levels)

read(Flag for Level 0 of expansion, namely the field points)

nFieldPoints := read(# of field points)

for i to *nFieldPoints* do

read(Point ID)

$Poles\_Field[1,i] = read)(\%complex\_\text{float})$

End do (loop i)

$Poles[1, nFieldpoints+1] = nFieldPoints$

Calls Procedure 3 (*Level* = 2,*nFieldPoints*,*Poles_Field*,*Paths_Field*)

At the end of Procedure 3 execution, all the field points are already stored. Now for the source points

read(Flag for the number of expansion levels)

nLevels := read(# of expansion levels)

read(Flag for Level 0 of expansion, namely the source points)

nSourcePoints := read(# of source points)

for i to *nSourcePoints* do

read(Point ID)

$Poles\_Source[1,i] = read)(\%complex\_\text{float})$

End do (loop i)

$Poles[1, nSourcepoints+1] = nSourcePoints$

Calls Procedure 3 (Level = 2,nSourcePoints,Poles_Source,Paths_Source)

8.1.5.5. Procedure 5

Computes the *P* vector for an expansion of $P\left(z - z_{c^l}\right)$ about a lower level pole $z_{c^{l-1}}$. The input data for this procedure depends on the *P_Matrix* as computed both in Procedures 6 (for fields) and 7 (for sources).

Checks if the current level of expansion is 1 (meaning that it will evaluate the distance between level 1 and 0, always looking down). If so:

If (nLevels = 1) then

For i to (n+1) do

$$P_j[i] = P\_Matrix[1,i]:$$

End do (Loop i)

Else

In this case, the function goes down another level

$$P\_rec = Procedure5(C, P\_Matrix, nLeves - 1, n)$$

For i to (n+1) do

$$P[i] = \sum_{j=1}^{i} C[j, i+1-j] P\_rec[j] P\_Matrix[nLevels, i+1-j]$$

End do (loop i)

Return *P*

End if

8.1.5.6. Procedure 6

Goes through the tree structure of field points from the highest level down to level 0 branch by branch. When it reaches the lowest level, calls Procedure 5 and adds this branch contribution to the other ones. Returns a vector of dimension (n+1) corresponding to the contribution of all field points to the higher level pole. Can also be understood as the result of the multiplication of matrix **A** by a vector.

Checks if *Level* 0 has been reached (*Level* 0 corresponds to the field points), if so, evaluates the contribution of this field point to the highest level pole and adds it up with the contributions of the other field points

If (*Level* = 0) then

The variable nLevels represents the total number of expansions levels

$$P = P + Procedure5(C, P\_\text{Matrix}, \text{nLevels}, \text{n})$$

Else

Variable *Dim* delivers how many poles in the lower level the current pole "sees"

$$Dim = Dimension(Paths[Level, i])$$

For j to *Dim* do

$$P\_Matrix[Level,1] = 1:$$

$$a = Poles[Level, Paths[Level,i][j]]$$

$$b = Poles[Level+1,i]$$

$$P\_Matrix[Level,2] = a - b$$

For *k* from 3 do (n+1) do

$$P\_Matrix[Level,k] = P\_Matrix[Level,k-1]*(a-b)$$

End do (loop k)

Goes down to the next level (Level-1) in the corresponding path (Paths[Level,i][j])

Calls

$$Procedure6(...,C,Level-1,nLevels,n,Paths[Level,i][j])$$

End do (loop j)

End if

8.1.5.7. Procedure 7

Goes through the tree structure from the highest level down to level 0 branch by branch. When it reaches the lowest level, calls Procedure 5 and evaluates the contributions from all the field points (Procedure 6) to this source. Returns a vector of dimension *nSourcePoints* corresponding to the contribution of all field points to each source point.

Checks if *Level* 0 has been reached (Level 0 corresponds to the source points), if so, evaluates the contribution of all the field points to one source point

If (*Level* = 0) then

P_Fields is the vector obtained from *Procedure6* and Q is the vector obtained from *Procedure9*

$$P = Procedure5(C, P\_\text{Matrix}, \text{nLevels}, \text{n})$$

$$P[cont] = Procedure8(fac, P\_Fields, P, Q, n)$$

Else

Variable *Dim* delivers how many poles in the lower level the current pole "sees"

$Dim = Dimension(Paths[Level, i])$

For j to *Dim* do

$P\_Matrix[Level, 1] = 1:$

$a = Poles[Level, Paths[Level, i][j]]$

$b = Poles[Level + 1, i]$

$P\_Matrix[Level, 2] = b - a$

For k from 3 do (n+1) do

$P\_Matrix[Level, k] = P\_Matrix[Level, k-1] * (b - a)$

End do (loop k)

Goes down to the next level (Level-1) in the corresponding path (Paths[Level,i][j])

Calls

$Procedure7(..., C, Level - 1, nLevels, n, Paths[Level, i][j], ...)$

End do (loop j)

End if

8.1.5.8. Procedure 8

Evaluates the contribution of all field points to a certain source point. Returns a vector of (n+1) terms.

$$result = \left(\sum_{i=1}^{n+1} \frac{1}{fac_i} P\_Fields_i \sum_{j=1}^{n+1} \frac{1}{fac_j} P\_Source_j Q_{i+j-1} \right)$$

8.1.5.9. Procedure 9

Evaluates the vector of the fundamental function derivatives up to (2n+1) terms.

Pole_Field and *Pole_Source* are simply the poles in the higher levels of both side expansions

$Pole\_Field = Poles\_Field[RowDimension(Poles\_Field), 1]$

$Pole\_Source = Poles\_Source[RowDimension(Poles\_Source), 1]$

f is the fundamental solution for the problem

$$Q[1] = f(\mathrm{P}ole\_\mathrm{F}ield - Pole\_\mathrm{S}ource)$$

For i from 2 to (2n+1) do

$$Q[i] = \frac{df}{dx}\Big|_{x=Pole\_Field-Pole\_Source}$$

End do (loop i)

8.2. Input data example

Below it is shown an example text file to be used as input for the aforementioned algorithm. It contains a set of field and source points and also some expansion poles

```
#_FIELD_LEVELS                    #Number of layers of field points
4

FIELD_LEVEL_#0                    #Field points
35
1      1012+67I
2      1015+72I
3      1020+73I
4      1028+72I
5      1027+69I
6      1030+78I
7      1040+80I
8      1045+77I
9      1066+80I
10     1070+76I
11     1070+73I
12     1067+72I
13     1085+60I
14     1095+59I
15     1105+59I
16     1108+56I
17     1109+52I
18     1107+49I
19     1090+43I
20     1094+41I
21     1086+37I
22     1065+39I
23     1075+38I
24     1100+26I
25     1096+23I
26     1089+19I
27     1073+15I
28     1065+13I
29     1060+15I
30     1056+25I
31     1053+21I
32     1049+19I
33     1039+24I
34     1044+27I
35     1047+30I

FIELD_LEVEL_#1                    #First layer poles
10
1      1020+70I      5      1      2      3      4      5
```

```
2       1040+75I      3       6       7       8
3       1060+75I      4       9       10      11      12
4       1090+55I      2       13      14
5       1100+53I      4       15      16      17      18
6       1085+40I      3       19      20      21
7       1075+35I      2       22      23
8       1090+25I      3       24      25      26
9       1065+20I      3       27      28      29
10      1050+25I      6       30      31      32      33      34      35

FIELD_LEVEL_#2                  #Second layer poles
3
1       1050+60I      3       1       2       3
2       1080+50I      3       4       5       6
3       1070+30I      4       7       8       9       10


FIELD_LEVEL_#3                  #Highest level pole
1
1       1000+50I      3       1       2       3

#_SOURCE_LEVELS                 #Number of layers of source points
4

SOURCE_LEVEL#0                  #Source points
14
1       -80+55I
2       -85+57I
3       -90+53I
4       -95+45I
5       -100+41I
6       -105+35I
7       -107+28I
8       -100+25I
9       -100+7I
10      -110+3I
11      -110-4I
12      -100-3I
13      -70+11I
14      -75-1I

SOURCE_LEVEL#1                  #First layer poles
5
1       -85+50I       3       1       2       3
2       -95+40I       2       4       5
3       -100+30I      3       6       7       8
4       -105+1I       4       9       10      11      12
5       -75+5I 2      13      14

SOURCE_LEVEL#2                  #Second layer poles
2
1       -80+40I       3       1       2       3
2       -85+10I       2       4       5

SOURCE_LEVEL#3                  #Highest level pole
1
1       -50+30I       2       1       2

#End of File
```

9
Apêndice 2

Nesse apêndice apresenta-se um algoritmo para os procedimentos utilizados no cálculo das tabelas de integração formuladas no capítulo 5. Esta implementação foi desenvolvida em linguagem Maple® pelo autor e está disponível para consulta.

9.1.
Algorithm

This section presents a pseudo-algorithm that can be used to assemble the matrixes shown in eqs.(5.10) and (5.11).

9.1.1.
Procedure 1

Evaluates the terms of matrix $\tilde{H}_{in}$ as shown in eq. (5.10) for linear, quadratic and cubic elements.

The input data for this procedure is n, which is the number of terms of the Taylor expansion to be considered in the algorithm.

Defines a matrix containing the shape functions to be considered. In this work, it is used an [0,1] local interval in which the shape functions are defined.

$$TN := \begin{bmatrix} [-\xi+1, \xi] \\ [2\xi^2-3\xi+1, -4\xi^2+4\xi, 2\xi^2-\xi] \\ \left[\frac{1}{16}(-\xi+1)(9\xi^2-1), \frac{9}{16}(3\xi-1)(\xi^2-1), \frac{9}{16}(3\xi+1)(1-\xi^2), \frac{1}{16}(1+\xi)(9\xi^2-1)\right] \end{bmatrix}$$

Initializes the P vector as defined in eq. (4.5) to $n+1$ terms as

$P[1] = 1$

$P[2] = z$

for i from 3 to $n+1$ do

$P[i] = P[i-1] * z$

end do (loop i)

Initializes a matrix named Table with dimensions $3x4$. Each element of Table is a P vector of $n+1$ elements. Table is the output of this procedure. It contains all the terms of $\tilde{H}_{in}$ considering linear, quadratic and cubic elements.

Table = Matrix(3,4)

for i to 3 do

for j to 4 do

$$Table[i, j] = Vector(n+1)$$

End do (loop j)

End do (loop i)

Evaluates the geometrical interpolations of nodal values, and its derivatives, for each element and stores this information in the vectors Z and dZ. The variables z_i are the nodal points in the element – 2 for linear, 3 for quadratic, and 4 for cubic.

$$Z = \left[\sum_{i=1}^{2} TN_{1i} z_i, \sum_{i=1}^{3} TN_{2i} z_i, \sum_{i=1}^{4} TN_{3i} z_i \right]$$

$$dZ_i = \frac{\mathrm{d}Z_i}{\mathrm{d}\xi}$$

Evaluates eq. (5.10) for the three elements considered and stores it in Table. oe stands for the order of the element – which may be 1, 2 or 3. z_c is the pole about which the elements' nodes are being expanded.

for oe oe to 3 do

for j to $oe+1$ do

for i to $n+1$ do

$$Table[oe, j][i] = \int_0^1 P_i \left(Z[oe] - z_c \right) dZ_{oe} TN_{oe,j} \mathrm{d}\xi$$

End do (loop i)

End do (loop j)

End do (loop oe)

9.1.2.
Procedure 2

Evaluates the terms of matrix $\tilde{G}_{jl}$ as shown in eq. (5.11) for linear, quadratic and cubic elements.

The input data for this procedure is n, which is the number of terms of the Taylor expansion to be considered in the algorithm.

Defines a matrix containing the shape functions to be considered. In this work, it is used an [0,1] local interval in which the shape functions are defined.

$$TN := \begin{bmatrix} [-\xi+1, \xi] \\ [2\xi^2-3\xi+1, -4\xi^2+4\xi, 2\xi^2-\xi] \\ \left[\frac{1}{16}(-\xi+1)(9\xi^2-1), \frac{9}{16}(3\xi-1)(\xi^2-1), \frac{9}{16}(3\xi+1)(1-\xi^2), \frac{1}{16}(1+\xi)(9\xi^2-1)\right] \end{bmatrix}$$

Initializes the P vector as defined in eq. (4.5) to $n+1$ terms as

$P[1] = 1$

$P[2] = z$

for i from 3 to $n+1$ do

$P[i] = P[i-1] * z$

end do (loop i)

Initializes a matrix named Table with dimensions $3x4$. Each element of Table is a P vector of $n+1$ elements. Table is the output of this procedure. It contains all the terms of $\tilde{G}_{jl}$ considering linear, quadratic and cubic elements.

Table = Matrix(3,4)

for i to 3 do

for j to 4 do

$Table[i, j] = Vector(n+1)$

End do (loop j)

End do (loop i)

Evaluates the geometrical interpolations of nodal values for each element and stores this information in the vectors Z dZ. The variables z_i are the nodal points in the element – 2 for linear, 3 for quadratic, and 4 for cubic.

$$Z = \left[\sum_{i=1}^{2} TN_{1i} z_i, \sum_{i=1}^{3} TN_{2i} z_i, \sum_{i=1}^{4} TN_{3i} z_i \right]$$

Evaluates eq. (5.11) for the three elements considered and stores it in Table. oe stands for the order of the element – which may be 1, 2 or 3. z_c is the pole about which the elements' nodes are being expanded.

for oe to 3 do

for j to $oe+1$ do

for i to $n+1$ do

$$Table[oe, j][i] = \int_0^1 P_i \left(Z[oe] - z[c] \right) TN_{oe,j} \mathrm{d}\xi$$

End do (loop i)

End do (loop j)

End do (loop oe)

9.2. Tabelas para elementos lineares e quadráticos

Nesta seção apresentam-se as tabelas de integração calculadas segundo as eqs. (5.10) e (5.11) para elementos lineares, quadráticos e cúbicos e são resultado dos algoritmos apresentados nas seções 9.1.1 e 9.1.2. Apenas são demonstrados os resultados para elementos lineares e quadráticos, pois os termos para elementos cúbicos são de mais elevada complexidade e sua visualização não é imprescindível, visto que tais funções podem ser desenvolvidas através dos algoritmos apresentados neste Apêndice.

9.2.1. Elementos Lineares

As tabelas de integração apresentadas abaixo são desenvolvidas para uma expansão em série de 3 termos ($n = 3$).

$$\tilde{H}_{in} = \begin{bmatrix} \frac{1}{2}(-\Delta_1 + \Delta_2) & \frac{1}{2}(-\Delta_1 + \Delta_2) \\ \frac{1}{6}(-2\Delta_1^{\,2} + \Delta_1\Delta_2 + \Delta_2^{\,2}) & \frac{1}{6}(-2\Delta_1^{\,2} + \Delta_1\Delta_2 + \Delta_2^{\,2}) \\ \frac{1}{12}(-3\Delta_1^{\,3} + \Delta_1^{\,2}\Delta_2 + \Delta_1\Delta_2^{\,2} + \Delta_2^{\,3}) & \frac{1}{12}(-3\Delta_1^{\,3} + \Delta_1^{\,2}\Delta_2 + \Delta_1\Delta_2^{\,2} + \Delta_2^{\,3}) \\ \frac{1}{20}(-4\Delta_1^{\,4} + \Delta_1^{\,3}\Delta_2 + \Delta_1^{\,2}\Delta_2^{\,2} + \Delta_1\Delta_2^{\,3} + \Delta_2^{\,4}) & \frac{1}{20}(-4\Delta_1^{\,4} + \Delta_1^{\,3}\Delta_2 + \Delta_1^{\,2}\Delta_2^{\,2} + \Delta_1\Delta_2^{\,3} + \Delta_2^{\,4}) \end{bmatrix}$$

$$\tilde{G}_{jl} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{6}(2\Delta_1 + \Delta_2) & \frac{1}{6}(\Delta_1 + 2\Delta_2) \\ \frac{1}{12}(3\Delta_1^{\,2} + 2\Delta_1\Delta_2 + \Delta_2^{\,2}) & \frac{1}{12}(\Delta_1^{\,2} + 2\Delta_1\Delta_2 + 3\Delta_2^{\,2}) \\ \frac{1}{20}(4\Delta_1^{\,3} + 3\Delta_1^{\,2}\Delta_2 + 2\Delta_1\Delta_2^{\,2} + \Delta_2^{\,3}) & \frac{1}{20}(\Delta_1^{\,3} + 2\Delta_1^{\,2}\Delta_2 + 3\Delta_1\Delta_2^{\,2} + 4\Delta_2^{\,3}) \end{bmatrix}$$

9.2.2. Elementos quadráticos

As tabelas de integração apresentadas abaixo são desenvolvidas para uma expansão em série de 2 termos ($n = 2$).

$$\tilde{H}_{i1} = \begin{bmatrix} \frac{1}{6}(-3\Delta_1 + 4\Delta_2 - \Delta_3) \\ \frac{1}{30}(-10\Delta_1^{\,2} + 6\Delta_1\Delta_2 - \Delta_1\Delta_3 + 8\Delta_2^{\,2} - 2\Delta_2\Delta_3 - \Delta_3^{\,2}) \\ \frac{1}{420}(-105\Delta_1^{\,3} - 9\Delta_3^{\,3} + 44\Delta_1^{\,2}\Delta_2 + 48\Delta_1\Delta_2^{\,2} + 64\Delta_2^{\,3} - 16\Delta_1\Delta_2\Delta_3 - 9\Delta_1^{\,2}\Delta_3 + 3\Delta_1\Delta_3^{\,2} - 16\Delta_2^{\,2}\Delta_3 - 4\Delta_2\Delta_3^{\,2}) \end{bmatrix}$$

$$\tilde{H}_{i2} = \begin{bmatrix} \frac{2}{3}(-\Delta_1 + \Delta_3) \\ \frac{1}{15}(-3\Delta_1^{\,2} - 4\Delta_1\Delta_2 + 4\Delta_2\Delta_3 + 3\Delta_3^{\,2}) \\ \frac{1}{420}(-44\Delta_1^{\,3} + 44\Delta_3^{\,3} - 48\Delta_1^{\,2}\Delta_2 - 64\Delta_1\Delta_2^{\,2} + 12\Delta_1^{\,2}\Delta_3 - 12\Delta_1\Delta_3^{\,2} + 64\Delta_2^{\,2}\Delta_3 + 48\Delta_2\Delta_3^{\,2}) \end{bmatrix}$$

$$\tilde{H}_{i3} =$$

$$\begin{bmatrix} \frac{1}{6}\left(\Delta_1 - 4\Delta_2 + 3\Delta_3\right) \\ \frac{1}{30}\left(\Delta_1^{\,2} + 2\Delta_1\Delta_2 - 8\Delta_2^{\,2} + \Delta_1\Delta_3 - 6\Delta_2\Delta_3 + 10\Delta_3^{\,2}\right) \\ \frac{1}{420}\left(9\Delta_1^{\,3} + 105\Delta_3^{\,3} + 4\Delta_1^{\,2}\Delta_2 + 16\Delta_1\Delta_2^{\,2} - 64\Delta_2^{\,3} + 16\Delta_1\Delta_2\Delta_3 - 3\Delta_1^{\,2}\Delta_3 + 9\Delta_1\Delta_3^{\,2} - 48\Delta_2^{\,2}\Delta_3 - 44\Delta_2\Delta_3^{\,2}\right) \end{bmatrix}$$

$$\tilde{G}_{j1} = \begin{bmatrix} \frac{1}{6} \\ \frac{1}{30}\left(4\Delta_1 + 2\Delta_2 - \Delta_3\right) \\ \frac{1}{420}\left(39\Delta_1^{\,2} + 40\Delta_1\Delta_2 - 6\Delta_1\Delta_3 + 16\Delta_2^{\,2} - 16\Delta_2\Delta_3 - 3\Delta_3^{\,2}\right) \end{bmatrix}$$

$$\tilde{G}_{j3} = \begin{bmatrix} \frac{1}{6} \\ \frac{1}{30}\left(-\Delta_1 + 2\Delta_2 + 4\Delta_3\right) \\ \frac{1}{420}\left(-3\Delta_1^{\,2} - 16\Delta_1\Delta_2 + 16\Delta_2^{\,2} - 6\Delta_1\Delta_3 + 40\Delta_2\Delta_3 + 39\Delta_3^{\,2}\right) \end{bmatrix}$$

$$\tilde{G}_{j2} = \begin{bmatrix} \frac{2}{3} \\ \frac{1}{15}\left(\Delta_1 + 8\Delta_2 + \Delta_3\right) \\ \frac{1}{105}\left(5\Delta_1^{\,2} + 8\Delta_1\Delta_2 + 48\Delta_2^{\,2} - 4\Delta_1\Delta_3 + 8\Delta_2\Delta_3 + 5\Delta_3^{\,2}\right) \end{bmatrix}$$