LAS - A programming language and development environment for learning matrix structural analysis

P. Paultre[a], E. Lapointe[b], C. Carbonneau[c], J. Proulx[d]

[a] *Professor, Dept. of Civil Engineering, Univ. of Sherbrooke, Sherbrooke,QC , Canada ,J1K 2R1 (corresponding author). E-mail: Patrick.Paultre@usherbrooke.ca*
[b] *PhD candidate, Dept. of Civil Engineering, Univ. of Sherbrooke, Sherbrooke, QC, Canada, J1K 2R1*
[c] *Ing Ph. D., Graitec Inc., 183 rue St-Charles O., suite 300, Longueil, QC, Canada, J4H 1C8*
[d] *Professor, Dept. of Civil Engineering, Univ. of Sherbrooke, Sherbrooke, QC, Canada, J1K 2R1*

Abstract

This paper presents LAS: Language for the Analysis of Structures. It is a development environment and a programming language for learning matrix structural analysis, dynamics of structures and the finite element method. LAS is a flexible learning environment in which users must program their own solutions to solve finite element static and dynamic problems. The language includes powerful operators, conditional expressions, loop expressions, and several functions (matrix manipulations, linear algebra, direct stiffness assembly, modal analysis, time domain dynamic analysis, frequency domain dynamic analysis). The development environment includes a code editor, a matrix manager, a finite element post-processor and a Fourier-analysis tool.

Key words:
Computer-aided education, Computer analysis language, Matrix algebra, Structural analysis, Direct stiffness method, Dynamics of structures

1. Introduction

LAS, which stands for Language for the Analysis of Structure, was developed at the Université de Sherbrooke with the objective of providing civil engineering student with a complete learning environment, where they can "program" solutions to complex structural analysis problems in a smart text editor and visualize the results in a graphical post-processor. LAS is currently used at the university level in both undergraduate (Structural analysis) and graduate (Dynamics of structures, Finite Element) courses. This paper presents the teaching approach, the LAS programming language and development environment as well as some applications with examples related to static and dynamic analysis of structures.

2. Teaching approach in structural analysis

There is actually a widespread use of commercial computer programs for structural analysis and design in university curriculum. These commercial programs are so easy to use that first-year engineering students can develop 3D models without any knowledge of structural analysis theory. However, results can be misleading, as there are large numbers of potential errors in the input process for inexperienced users or students. Moreover, they act as *black-box* type programs. The user has little or no control on the solution process. Hence, there is still a need for students to understand the algorithms involved in the direct stiffness method, in the finite element method, in structural dynamics, in modal analysis, etc.

The goal of the computer program presented herein is not to create a finite element program from scratch but to provide a large and powerful set of commands that specifically handle matrix manipulations and a rich set of functions used in the solution process. Using such a tool, the engineering student can develop a complete solution for a wide range of static and dynamic structural analysis problems. For example, a basic stiffness method problem would include commands to carry out the following operations : (1) enter nodal points

and assign degrees-of-freedom for the structure; (2) enter element connectivity; (3) create element stiffness matrices; (4) assemble the global stiffness matrix; (5) create and assemble the load vectors; (6) solve the problem using any linear algebra solving method; (7) retrieve important results such as the displacements and internal forces; and (8) eventually visualize and interpret the results. The teacher can focus on the algorithms and implications of key modeling parameters, rather than on the actual calculations and matrix manipulations (inversion, Gauss elimination, eigenvalue calculations, etc.).

The CAL (Computer-Assisted Learning) programming language was developed in FORTRAN at Berkeley in the late 70s early 80s with these objectives in mind. It was a text-based environment that provided a large set of matrix-related commands, where users could program algorithms for the direct stiffness method as well as some structural-dynamics related commands, (Wilson, 1979; Wilson and Hoit, 1984). This program was distributed freely and served as basis for further development. In the late 80s, a graphical post-processor (CALGR) was developed at the Université de Sherbrooke using a graphical language that was available for DOS and based on the FORTRAN language. This was a separate program that read and graphically interpreted results calculated by an improved version of the CAL program, (Paultre et al., 1991b,a). It provided users with the ability to visualize solutions for the direct stiffness method (including specific steps of the stiffness matrix assembly) as well as results of modal and time-history analysis (vibration mode shapes, earthquake response, etc). This graphical program was ported to C++ during the 90s and made available for Windows systems. This version, called CALWIN, included a text editor with online help for all matrix and structural analysis commands, as well as a graphical post-processor, (Labbé, 2000). It was successfully used in several civil engineering departments around the world at the undergraduate and graduate levels.

As the calculation engine was being developed and new commands were added during those years, the main shortcomings became more apparent, that is the limitation of the CAL programming language itself. The function syntax, the function-based approch for matrix manupulation and the fact that the core of the program was in FORTRAN were quickly becoming outdated. In the mid-nineties, concepts were laid down for a new object-oriented environment that would be built from scratch and programmed in C++, (Carbonneau, 1994). This first attempt provided the basis for the recent developments of LAS (now in Visual C# .Net), which has become a full-fledged programming environment for static and dynamic structural analysis Lapointe (2009). The program itself and its applications are described in the following sections.

One can ask what is the need for a new programming language for matrix structural analysis when structural packages such as CALFEM for MatLab®, (Dahlblom et al., 1986) and add-on spreadsheets for Matcad® (Cedeno-Rosete, 2007) already exist. The main reasons for developing LAS is that it is (i) a complete environment that includes the code editor, the solver and the post-processor (other solutions generally do not include visualization capabilities that improve the learning experience); (ii) optimized for static and dynamic analysis; (iii) free of charge; and (iv) portable as it can fit on a USB key and executed at will without any installation procedure on the host computer.

3. The Program

LAS is a single-executable and portable software fully written in Visual C# .Net and compatible with any Windows® operating system bundled with Microsoft® .Net framework 3.5 SP1. The LAS software may be associated with the LAS document filetype (*.las) during the installation procedure which is done once when the program is first launched.

Internally, the LAS software is split in two distinct modules : a calculator and a development environment. The development environment is a multiple document interface (MDI) used to create, edit and run LAS documents. It adds data visualization, graphics and finite element post-processing capabilities to the calculator.

Figure 1 shows the user interface of the development environment with two LAS documents opened. The main window of the development environment is mainly used as document windows manager. Each created or opened LAS file is visually represented by a document window that uses a tabs-based interface to navigate through the provided LAS code editor, output viewer, matrix manager, finite element post-processor and Fourier-analysis tool. A context-sensitive toolbar below the tabs suggests appropriate user interactions.

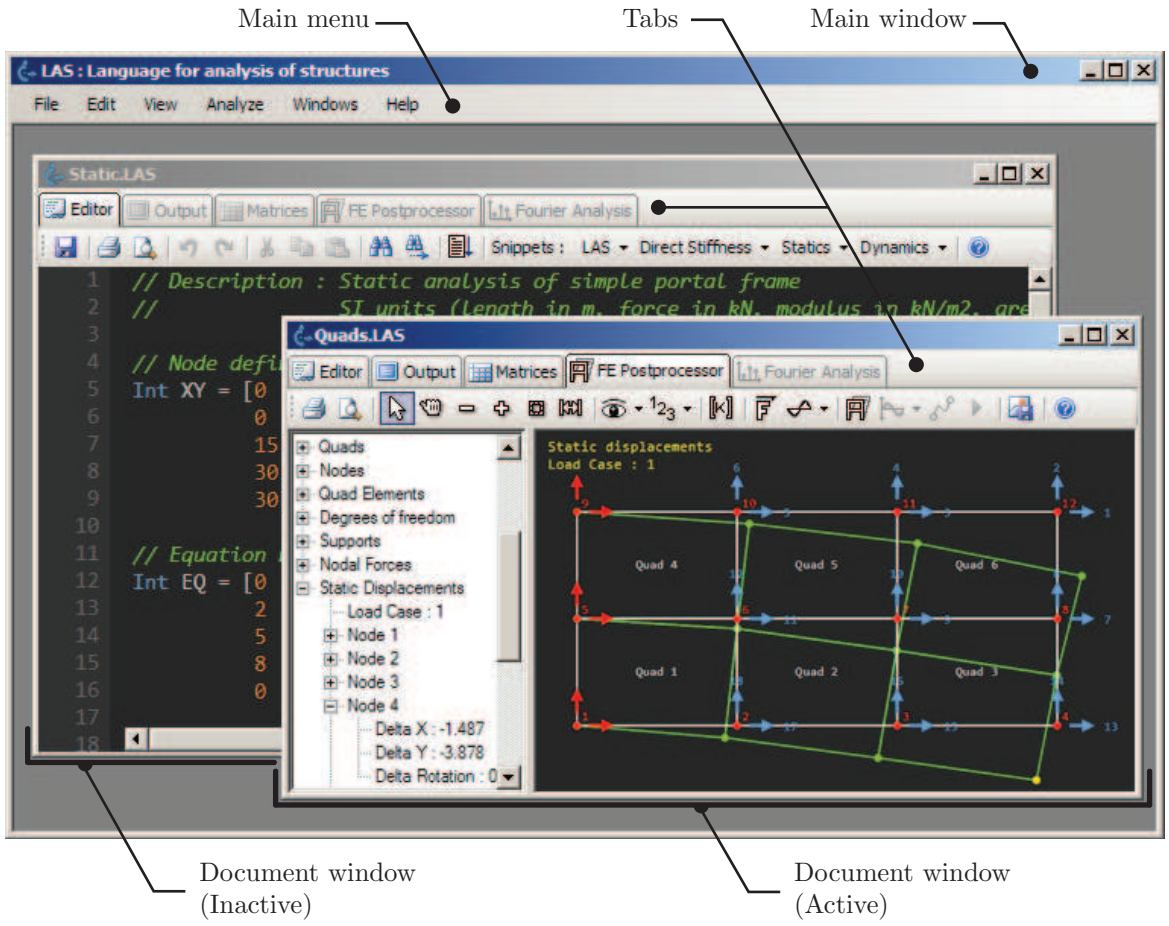

Figure 1: LAS Development Environment

Figure 2 illustrates the implementation of LAS and how the data flows through the different components of the calculator and development environments. The included code editor (or any ASCII text editor) can be used to create a LAS document. This type of document consists of a series of instructions written in the LAS programming language using ASCII characters. These instructions include commands to create matrices, and to manipulate them in order to solve a structural analysis problem. The compiler then transcodes the LAS document into an object code. To accomplish the interpretation of the text instructions into structured expressions, the compiler parses the LAS code according to the LAS syntax (defined functions, operators and statements). The resulting object code consists of a list of sequentially ordered commands. An object code command can be an operator call, a function call or a matrix variable push. The executor then carries-out each object code command. The matrix, numerical and structural algorithms are included in libraries linked to the executor. All matrices created and manipulated by the executor are stored in a dynamic array during run time of the LAS application.

The LAS Code editor consist of an ASCII text editor featuring lines numbering, syntax highlighting, word auto-completion, on-demand help, modern monospaced font and automatic snippets insertions. Snippets are predefined and reusable portions of code that allow for easier coding of complex algorithms and speed-up repetitive code writing. They are also a convenient way to help unexperienced programmers with the LAS syntax. The program also includes an output viewer, where the user can display matrices as well as debugging information.

A matrix manager lists all matrices resulting

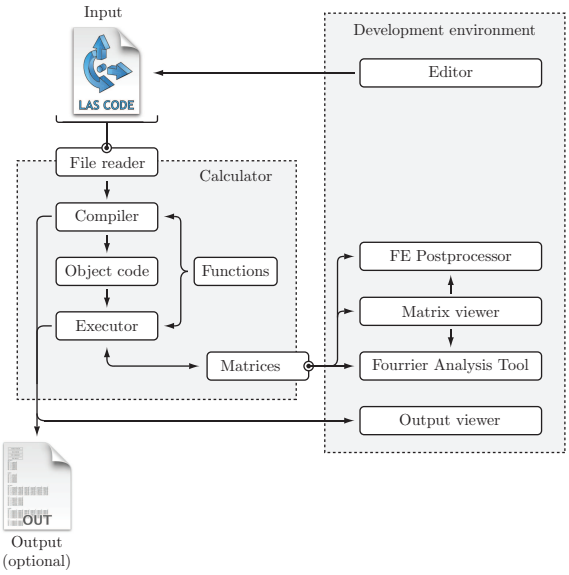

Figure 2: Flow of data through in LAS program

from the execution of an object code and lets the user assign physical significance or *usages* to those matrices (for example: displacement, forces, stiffness matrices or mode shapes). This assignment of usages is required by the graphical post-processor and the Fourier-analysis tool to select the appropriate matrices during data postprocessing and visualization. A matrix viewer and a graphics plotter are also included in the matrix manager and can be used for visual investigation of matrices content.

4. Numerical analysis language

Although the main purpose of the LAS programming language is the analysis of structures, it has been designed as a general numerical analysis language to offer flexibility and versatility in problem solving strategies. It provide a rich syntax composed of variables, operators, statements and a large set of procedures and functions capable of returning single or multiple values. While developing their own solutions to a problem, users can call any of the currently 140 built-in functions, code their own algorithms or even create user-defined functions. These user-defined functions are developed separately and can be called by other LAS codes as a standard function with a defined number of arguments and variable number of returned values. As in the C language, arguments can be passed as copy or by reference in these user-defined functions.

Code excerpt #1 shows the matrix declaration and assignation procedures. Matrix variables must be declared first and support integer and double precision floating-point formats. Sparse matrices can be defined manually while specific matrices such as diagonal, identity, zero, full and random matrices can be generated automatically. Several advanced matrix manipulation functions are available. A straightforward implementation of matrix indexing can also be used to directly assign or extract specific matrix elements or sub-matrices.

Code 1: Declaration and assignation of matrices

```
1  // Integer and double numbers
2  Int Number1 = 100
3  Dbl Number2 = 25.4
4  Dbl Number3 = 4.3E-3
5
6  // 2x3 user-defined matrix
7  Dbl Matrix1 = [25.1  12.3  34.2
8                 42.7  55.2  87.3]
9
10 // 3x3 identity matrix
11 Dbl Matrix2 = Identity(3)
```

Table 1 lists all supported operators ordered by precedence. The operators at the top of the table are evaluated first. The standard arithmetic operators are applied to whole matrices (matrix-wise operators), while those beginning with the symbol @ as well as the relational, conditional and equality operators apply to individual element (element-wise operators).

Code excerpt #2 shows how these operators can be used with matrices.

Code 2: Using operators with matrices

```
1  // Arithmetic operations with numbers
2  Dbl Number4 = 10 * ( 3 - 1.1 ) ^ 4
3
4  // Arithmetic operations with variables
5  Dbl Matrix3 = Matrix1 * Matrix2 * t(Matrix1)
```

By including If...Else...EndIf conditional expressions as well as conditional loop expressions such as For...Next, While...EndWhile and Loop...Until, the LAS programming language allows for comprehensive coding of complex and iterative algorithms.

Linear algebra functions are provided to solve the following problems: (i) linear systems of equations (using $\mathbf{LDL}^{\mathsf{T}}$ decomposition, **LU** decomposition or Gaussian elimination with backward or forward substitution procedures); (ii) standard and generalized eigenvalues problems; (iii) singular value decomposition; (iv) inversion of square matrices; and

Table 1: Precedence of operators

Category	Operators
Matrix indexing	( )
Function call	( )
Incrementation	++ −−
Unary	+ − !
Power	∧ @∧
Multiplicative	∗ / @∗ @/
Additive	+ − @+ @−
Relational	< > <= >=
Equality	== ! =
Conditional AND	&&
Conditional OR	||
Assignment	+= −= ∗= /= ∧= =

(v) computation of matrix norms, rank and determinant. Note that unlike modern optimized implementation of linear algebra algorithms such as LAPACK (Anderson et al., 1999), the vast majority of linear algebra function in LAS are non-destructive for user-defined variables and designed for an educational purpose. Code excerpt #3 shows how procedures and functions are called in the LAS language. A procedure such as Print returns no value and may accept a variable number of arguments. A function such as LDLT may return multiple values. Brackets are not required for functions that return only one value.

Code 3: Using functions and procedures

```
// Square matrix
Dbl A = [ 2 -1  3  5
         -1  2 -1  1
          3 -1  4  2
          5  1  2  5]

// LDLT Decomposition
Dbl {L,D} = LDLT(A)

// Print matrices to output
Print(A,L,D)
```

Frequency domain functions are provided: (i) to compute Fast Fourier Transforms (and their inverse) of real and complex vectors; (ii) to compute the power spectral density of a given function; and (iii) to create single-sided or double-sided amplitude and phase spectra.

5. The direct stiffness method

One of the main objectives of the LAS environment is to provide the student with tools to create their own algorithms to apply the direct stiffness method to a structural analysis problem. The user can then create the appropriate matrices for the structural system. Finite-element functions allow for the automatic generation of stiffness, mass, transformation and force-displacement matrices for truss elements, beam-column elements and 4-nodes quadrilateral isoparametric elements. The latter use a second order gaussian integration technique instead of the exact solution. These finite element functions require the definition of separate matrices for nodal coordinates, degrees-of-freedom, element connectivity, element geometric properties (area, inertia, etc.) and material properties (Young modulus, Poisson's ratio, ect.).

Assembly of the element matrices into the global stiffness and mass matrices are carried out with direct stiffness functions. These functions also allow for the assembly of beam member loads vector (such as concentrated or distributed loads) into the global nodal loads vector. The assembly process is accomplished using the location matrix that is automatically generated from the element connectivity and the specified degrees-of-freedom. Functions related to the beam element also feature rigid ends capability (which modifies the element stiffness matrices accordingly).

Figure 3 illustrates how the finite element post-processor can be used as a direct stiffness method learning tool. By selecting individual elements, students can investigate the assembly process of the global stiffness matrix. Individual terms in the global stiffness matrix that are influenced by the currently selected element are highlighted in the in the global stiffness matrix shown in the lower panel. Additional information about the selected post-processor item are listed in the left panel. Nodes, finite elements, supports, static and dynamic degrees-of-freedom, nodal loads, beam member loads and their corresponding numbering and labels can be displayed in the main (center) panel.

6. Static analyses of structural systems

The solution of the linear static equilibrium equation (1), is carried out using direct stiffness functions to create and assemble the global stiffness ma-

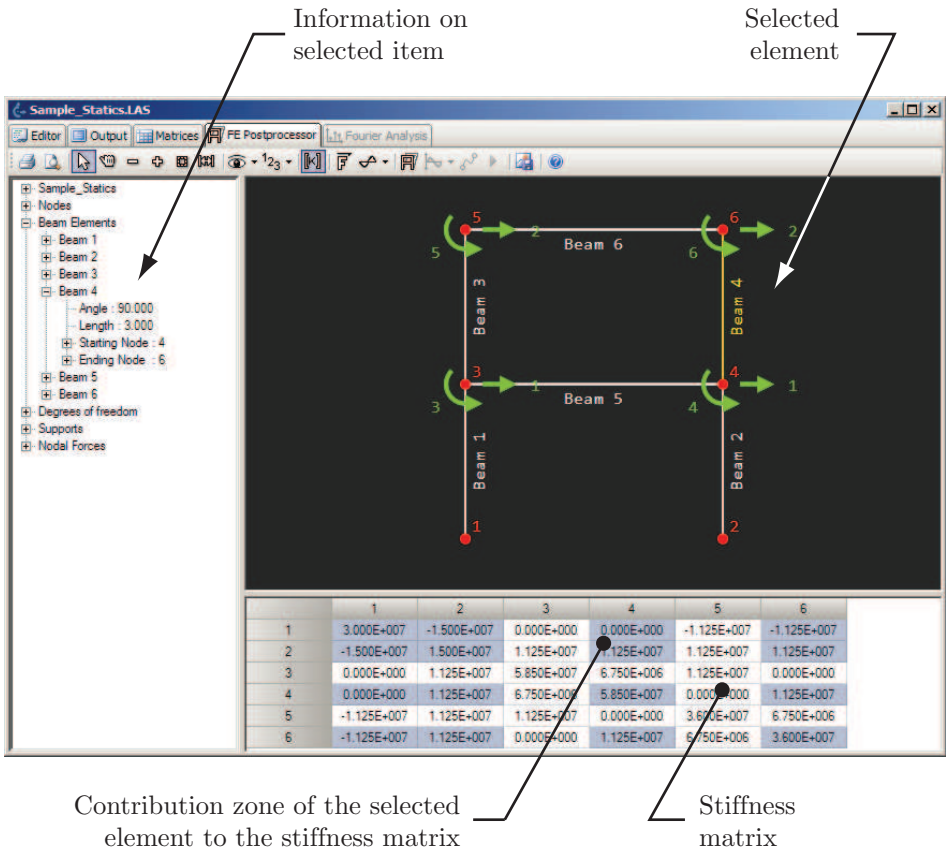

Figure 3: Visual representation of the global stiffness matrix assembly

trix, $\mathbf{K}$, and global load vector, $\mathbf{F}$, and using any of the linear algebra solving functions to determine the displacement vector, $\mathbf{u}$.

$$\mathbf{F} = \mathbf{K}\mathbf{u} \tag{1}$$

Figure 4 illustrates the internal moment diagrams of the two storeys frame subjected to earthquake static equivalent forces. The finite element post-processor can compute and display axial load diagrams, shear force diagrams and bending moment diagrams. In addition to the minimum and maximum values displayed, symbols indicate compression and tension zones for axial load diagrams, clockwise and counterclockwise shearing zones for shear force diagrams and curvature for bending moment diagrams. These symbols have an added academic value and should lead to a better understanding in the distribution of the deformation in the analyzed structure.

The finite element post-processor can illustrate the deformed shape corresponding to the calculated displacements. Linear interpolation is used for trusses and 4-Nodes quadrilateral elements while cubic interpolation is used for beam elements. Multiple loading cases are supported.

7. Dynamic analyses of structural systems

The solution of the linear dynamic equilibrium equation (2), is carried out: (i) in time domain using step-by-step integration methods applied directly to equation (2); (ii) in modal space using modal superposition; or (iii) in frequency domain using Fourier transformation.

$$\mathbf{M}\ddot{\mathbf{u}}(t) + \mathbf{C}\dot{\mathbf{u}}(t) + \mathbf{K}\mathbf{u}(t) = \mathbf{F}(t) \tag{2}$$

where $\mathbf{M}$ is the mass matrix, $\mathbf{C}$ is the damping matrix, $\dot{\mathbf{u}}(t)$ and $\ddot{\mathbf{u}}(t)$ are respectively the velocity and acceleration vectors. The modal superposition method requires the extraction of the appropriate mode shapes and corresponding natural frequencies, as well as the step-by-step integration method

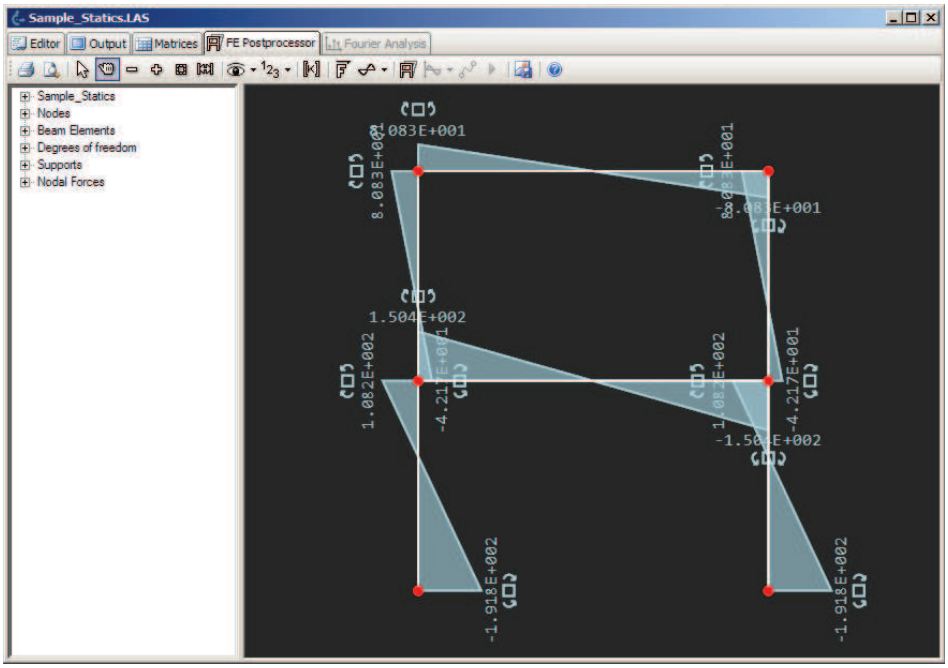

Figure 4: Internal force diagrams

when applied to uncoupled systems. Functions related to these dynamic analysis methods are presented in the following subsections.

Figure 5 illustrates the deformed shape of the same two-storey building subjected to an earthquake ground motion. The displacement time-histories of the dynamic degrees-of-freedom and the forcing function are respectively plotted in the upper and lower part of the right panel. The deformed shape can also be animated in the main panel of the post-processor. When the animation is paused, the post-processor can be used to investigate the model (including the analysis result such as the dynamic displacements) as described above in the case of static analysis.

7.1. Modal analysis

The computation of the natural frequencies and the mode shapes of a multi-degree-of-freedom (MDOF) system is carried out by solving the following generalized eigenvalue problem:

$$\mathbf{K\Phi} = \mathbf{M\Phi\Lambda} \tag{3}$$

where $\mathbf{\Phi}$ is the eigenvectors (mode shapes) matrix and $\mathbf{\Lambda}$ is the eigenvalues (frequencies) matrix. The included QR, HQRI or Jacobi method can be used when information is required for the complete modal space (all modes). If only a limited number of natural frequencies and mode shapes are required, the subspace iteration method can be used. Additionally, the lowest and the highest frequencies and their corresponding mode shapes can be extracted with the direct and inverse iteration methods respectively. The Jacobi method can also be used to obtain rigid-body modes by solving the following standard eigenvalue problem:

$$\mathbf{K\Phi} = \mathbf{\Phi\Lambda} \tag{4}$$

It has been shown that in some cases better results can be obtained by using load-dependent Ritz vectors. Their generation can be carried out in LAS by providing a user-defined load vector to the Ritz function. In the case of earthquake loading, they can be automatically generated.

7.2. Time domain analysis

In the case of uncoupled system, the piecewise linear integration method should be used as it is an exact method for piecewise linear forcing functions. For general forcing function, the precision of the results depends on the interpolation used to represent the function. For educational purposes, Duhamel numerical integration can also be carried out using the rectangular trapezoidal or Simpson

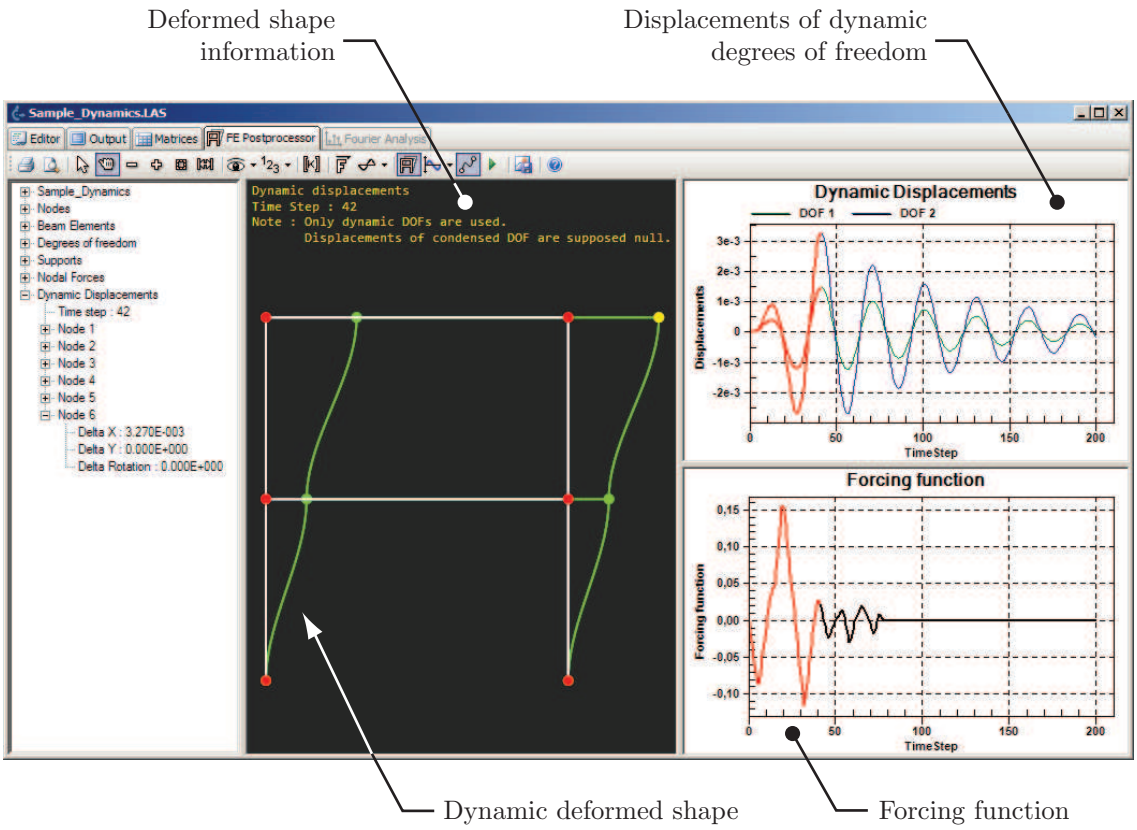

Figure 5: Deformed shape, forcing function and displacement time-histories

quadrature formula. The modal superposition can then be carried out.

Several integration schemes are available for coupled system. The central difference and Newmark linear acceleration methods are conditionally stable and should be used with caution. The Newmark constant acceleration method is unconditionally stable. The Newmark linear accelaration method does not introduce numerical damping. The family of α integration schemes such as HHT-α, WBZ-α and generalized-α methods are also included in LAS. These integration schemes are unconditionally stable, but introduce numerical damping. This numerical damping is recommended to control high-frequency responses that artefacts of the finite element model. Other integration schemes such as Houbolt, collocation and Wilson-θ methods are also included in LAS. This rich set of numerical integration method can be used effectively in a teaching environment to study precision, stability, numerical damping of different integration schemes.

7.3. Frequency domain analysis

Dynamic analysis in the frequency domain can be carried out with LAS. The forcing functions are first transformed in the frequency domain, using the Fast Fourier Transform, and then multiplied by the frequency response function of the system under consideration. The resulting frequency domain solution can be transformed back in the time domain using inverse Fast Fourier Transform.

LAS also includes specific functions to carry out Response Spectrum Superposition (RSS). The spectral accelerations corresponding to the natural frequencies of a MDOF system can be interpolated from a given response of design spectra. The modal responses can then be combined in LAS using the Square Root of the Sum of the Squares (SRSS) method or the Complete Quadratic Combination (CQC) method. The latter method is damping dependant and will provide a more accurate combination for systems with closely-spaced natural frequencies.

Working in the frequency domain can be chal-

lenging for students because the amplitude and phase representation of a signal is not as intuitive as a spatial coordinate system in time domain. The Fourier-analysis tool in LAS can be used to visualize the Fast Fourier Transform or Fourier series of any discrete signal. In structural engineering, this tool would most likely be used to visually investigate the frequency content of a force function or of a response history.

Figure 6 illustrates the Fourier analysis of a force function. The interface of the Fourier tool in LAS is composed of four graph panels arranged in quadrants. The original signal is always plotted in the time domain in the upper left quadrant. The student can select which plots are displayed in the lower quadrants. These include the single-sided amplitude and phase spectrums, the double-sided amplitude and phase spectrums, the power spectral density and the real and complex components of a Fast Fourier Transform. The student can also select individual harmonic components and the corresponding sine wave of the Fast Fourier Transform or of the Fourier series is plotted in the upper right quadrant. The Fourier tool can also be used to display and animate the superposition of sine waves up to the specific harmonic (the fifth harmonic in the case shown in Figure 6) in order to reconstruct the original signal. This process gives a physical significance to the Fourier series in signal decomposition.

8. The LAS set of commands

Figure 7 provides a list (or *cheatsheet*) of the complete set of commands in the LAS programming environment. The **Matrix** group of commands is at the core of this language and provides instructions to create and manipulate matrices of real or integer numbers. In a typical structural analysis problem, students would first use these commands to create a model (nodes, elements, degrees-of-freedom). The **Finite Element Method** commands would then be used to create elements matrices and to assemble them. The **Linear Algebra** commands would then be used in static or dynamic analysis to solve equations 1 and 2 expressed above. In a dynamic problem, **Steb-by-step**, **Frequency-domain** and/or **Dynamics** commands would be used to compute the response of a system subjected to a dynamic load. The **Math** group of commands provides the basic mathematical operators and functions that can be used in structural analysis. LAS also provides **Loop** and **Conditional** expressions to develop iterative algorithms such as the direct or inverse iteration method used to extract mode shapes. Finally, students can create **User-defined** functions to create new algorithms for specific problems.

9. Static and dynamic analyses examples

The complete LAS codes required to carry out the analyses of the two-storey building illustrated in the previous figures are presented below.

The complete listing for static analysis (presented in figure 9) shows the commands used for direct stiffness assembly (lines 5-30), displacement solving (lines 34-38) and internal forces computation (lines 40-51).

The complete listing for dynamic analyses (presented in figure 9) shows the commands used for the creation of the dynamic system matrices (lines 8-12 and 27-29), the modal analysis (lines 16-23) and the linear time history analysis (lines 31-52). Rotational degrees-of-freedom are reduced by static condensation and a lumped-mass approach is used. The Rayleigh damping matrix is generated using the natural frequencies computed by modal analysis. Since the model geometry and the assembly of the stiffness matrix were already coded for the static analysis example, they are simply input at the beginning of the dynamic analysis (line 4). The input command is a powerful command that allows a file to be separated into different pieces.

10. Conclusion

The LAS program was developed for civil and mechanical engineering as a tool to learn static and dynamic structural analysis. It is the result of more than twenty years of teaching and has evolved from a text-based set of matrix related commands to a full-fledged programming environment with a graphical post-processor that provides insight to the procedures involved in the direct stiffness method, the finite element method, as well as time and frequency domain dynamic analysis. Students can quickly write an algorithm without having advanced programming skills (the language is straightforward and intuitive) but also without using a commercial analysis program in *black-box* mode. After a semester in structural analysis, students have a deeper understanding of the direct

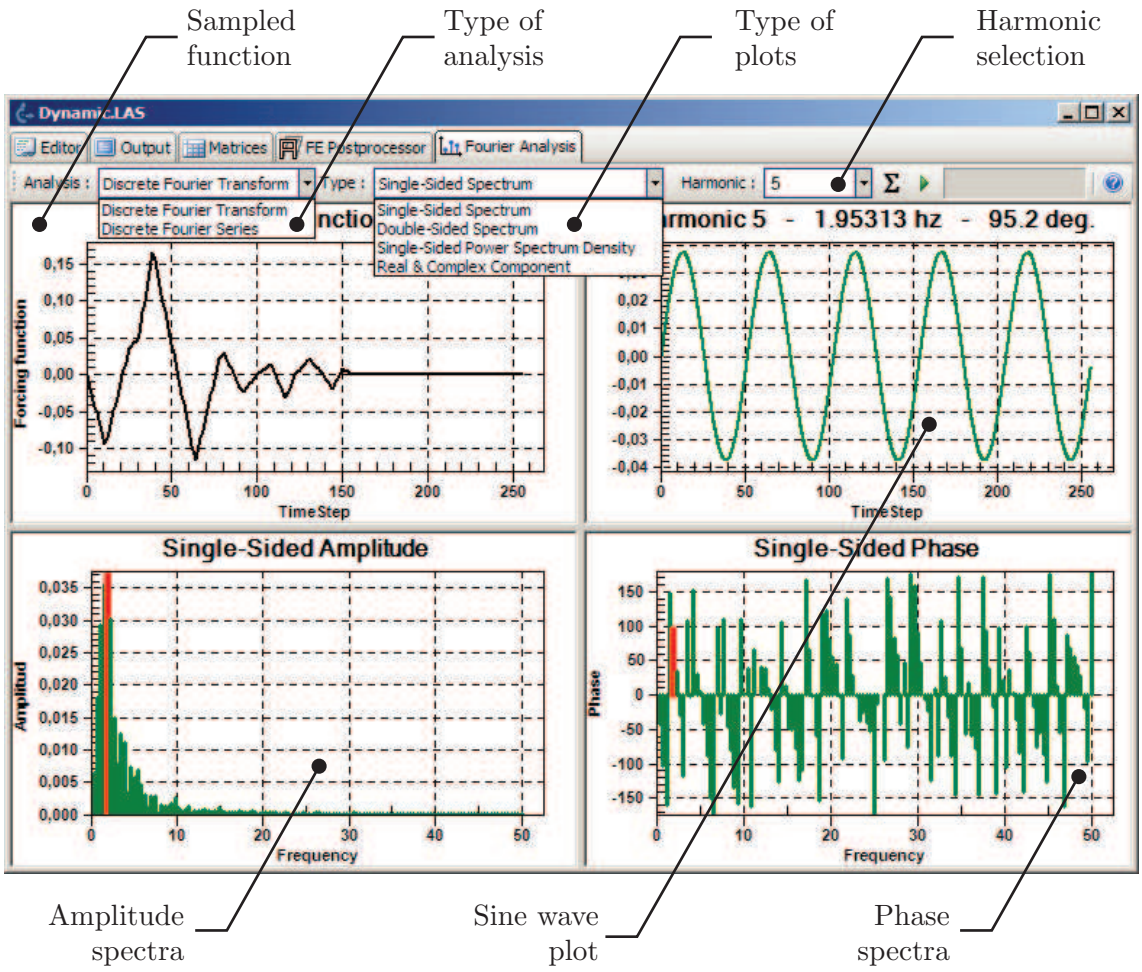

Figure 6: Fourier-analysis tool in LAS

stiffness method, and at the graduate level, it is extensively used in structural dynamics and finite element courses. The use of the program, however, is not restricted to teaching as it can most effectively be used in research for developing numerical algorithm or in the laboratory. The program is currently used at Université de Sherbrooke and can be freely downloaded at `www.civil.usherbrooke.ca/ppaultre/Software.html`

References

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D., 1999. LAPACK Users' Guide, 3rd Edition. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Carbonneau, C., April 1994. LAS: Langage d'analyse des structure. Master's thesis, Université de Sherbrooke.

Cedeno-Rosete, R., 2007. Stiffness matrix structural analysis educational package by mathcad. Computer Applications in Engineering Education 15 (2), 107 – 12.

Dahlblom, O., Peterson, A., Petersson, H., 1986. Calfem - a program for computer-aided learning of the finite element method. Engineering computations 3 (2), 155 – 160.

Labbé, A., December 2000. Développement de logiciel d'analyse structurale à interface graphique : Wmnphi et calwin. Master's thesis, Université de Sherbrooke.

Lapointe, É., December 2009. LAS - Un environnement de dévelopement destiné à l'apprentissage assisté par ordinateur de l'analyse matricielle des structures. Master's thesis, Université de Sherbrooke.

Paultre, P., Leger, P., Proulx, J., 1991a. Computer-aided education in structural dynamics. Journal of Computing in Civil Engineering 5 (4), 374 – 390.

Paultre, P., Proulx, J., Leger, P., 1991b. Cal/cgi - an application of graphics for matrix structural analysis education. Computers & Graphics 15 (1), 131 – 5.

Wilson, E. L., April 1979. Cal - a computer analysis language for teaching structural analysis. Computers and Structures 10 (1-2), 127 – 132.

Wilson, E. L., Hoit, M. I., 1984. Computer adaptive language for the development of structural analysis programs. Computers and Structures 19 (3), 321 – 338.

General

Execution :
Input Break
End Time
Stop

Comments :
// *or* \\

Matrix

Declaration :
Dbl
Int

Generation :
Diagonal
Init
Identity
Pi
Random
Zero

Properties :
ColMax
ColMin
Cols
Colsum
Max
Min
Norm_Infinity
Norm_One
Norm_Two
Product
Rank
RowMax
RowMin
Rows
RowSum
Sum

Manipulation :
Col
Combine
Delete
Determinant | Det
Diagonal
Export
Interpolation
Invert | Inv
Print
Resize
Rotate
Row
StoreDiag
SwitchCol
SwitchRow
Transpose | t

Math

Trigonometry :
ACos
ASin
ATan
Cos
Sin
Tan

Base-Power :
Exp
Ln
Log
Sqrt

Hyperbolic :
Sinh
Cosh
Tanh

Number :
Abs
Even
Odd

Angle :
Deg
Rad

Rounding :
Ceil
Floor

Linear algebra

Gaussian elimination :
Backward_Substitution
Forward_Substitution
Row_Reduce_From_Buttom
Row_Reduce_From_Top
Solve

LU Decomposition :
LU
SolveLU

LDLT Decomposition :
LDLT
SolveLDLT

Singular value decomposition :
SVD

Eigenvalues problem :
HQRI
Jacobi
Direct_Iteration
Inverse_Iteration
Subspace_Iteration

Finite Element Method

Generation :
Gen_Elements
Gen_Equations
Gen_Nodes

Direct stiffness method :
Assemble
Internal_Forces

Beam element :
Beam_Make_FD
Beam_Make_K
Beam_Make_KG
Beam_Make_LM
Beam_Make_M
Beam_Make_T
Beam_Make_T_Rigid
Beam_Rigid_Joint

Quadilateral element :
Quad_Make_K
Quad_Make_LM
Quad_Make_M

Truss element :
Truss_Make_FD
Truss_Make_K
Truss_Make_KG
Truss_Make_LM
Truss_Make_M
Truss_Make_T

Conditional expressions

If(*condition*)
|
Else
|
EndIf

If(*condition*)
|
EndIf

Step-by-step integration schemes

Uncoupled systems :
Duhamel_Rectangle
Duhamel_Simpson
Duhamel_Triangle
Piecewise_Linear

Coupled systems :
Average_Acceleration
Centered_Difference
Collocation
Generalized_Alpha
HHT_Alpha
Houbolt
Linear_Acceleration
WBZ_Alpha
Wilson_Theta

Dynamics

Damping :
Caughey
Rayleigh

Coordinates reduction :
Condense
Ritz

Combinaison methods :
CQC
SRSS

Varia :
Exact_displacement
Function
Pseudo_Spectrums

User-defined function

User_Defined_Function
Get_Argument_ByCopy
Get_Argument_ByReference
Return

Frequency domain analysis

Fast Fourier Transform :
iFFT_Real or iFFT_Complex
FFT_Real or FFT_Complex

Spectral analysis :
FFT_Spectrum_Single_Sided
FFT_Spectrum_Double_Sided
FFT_Spectrum_Power_Density
FFT_Spectrum_Complex_Components

Frequency response :
FSolve

Loop expressions

For(*declaration* ; *condition* ; *incrementation*)
|
Next

While(*condition*)
|
EndWhile

Loop
|
Until(*condition*)

Figure 7: LAS programming language cheat sheet

```
// CONTENT OF FILE : Sample_Statics.LAS

// PART A : Direct Stiffness Assembly

// Coordinates of nodes  (6x2)
Dbl XY = [ 0 0 ; 5 0 ; 0 3 ; 5 3 ; 0 6 ; 5 6]

// Beam connectivity (6x2)
Dbl Beam_EL = [1 3 ; 2 4 ; 3 5 ; 4 6 ; 3 4 ; 5 6]

// Equation matrix  (6x3) : DOF of each node (tx,ty,rot)
Int EQ = [0 0 0 ; 0 0 0 ; 1 0 3 ; 1 0 4 ; 2 0 5 ; 2 0 6 ]

// Location Matrix (6x6)
Int Beam_LM = Beam_Make_LM( Beam_EL, EQ)

// Properties matrix [A As I E Nu Dn]
// 300mm x 300mm beams & columns
Dbl Properties = [ 0.3*0.3 0 0.3^4/12 25E9 0 0 ]

// Global Stiffness matrix for each element
Dbl KColumn = Beam_Make_K(1,XY,Beam_EL,Properties ,"G")
Dbl KBeam   = Beam_Make_K(5,XY,Beam_EL,Properties ,"G")

// Global stiffness matrix assembly
Dbl K = Zero(6)

// Add the element stiffness matrices to the global stiffness matrix
Assemble( K, KColumn , Beam_LM, [1,4] )
Assemble( K, KBeam   , Beam_LM, [5,6] )

// PART B : Static Analysis

// Nodal Force Vector :
Dbl F = [118;82;0;0;0;0]

// Solving displacement F = Ku
Dbl U = Solve(K,F)

// Force-Displacement matrices
Dbl FDColumn = Beam_Make_FD(1,XY,Beam_EL,Properties )
Dbl FDBeam   = Beam_Make_FD(5,XY,Beam_EL,Properties )

// Internal forces
Dbl Beam_IF= Zero(6,4)
Beam_IF(1,1) = Beam_Internal_Forces(FDColumn,U,Beam_LM,Beam_EL,1)
Beam_IF(1,2) = Beam_Internal_Forces(FDColumn,U,Beam_LM,Beam_EL,2)
Beam_IF(1,3) = Beam_Internal_Forces(FDColumn,U,Beam_LM,Beam_EL,3)
Beam_IF(1,4) = Beam_Internal_Forces(FDColumn,U,Beam_LM,Beam_EL,4)
Beam_IF(1,5) = Beam_Internal_Forces(FDBeam,U,Beam_LM,Beam_EL,5)
Beam_IF(1,6) = Beam_Internal_Forces(FDBeam,U,Beam_LM,Beam_EL,6)
```

Figure 8: Code for static analysis including direct stiffness assembly

```
// CONTENT OF FILE : Sample_Statics.LAS

// PART A : Direct Stiffness Assembly

// Coordinates of nodes  (6x2)
Dbl XY = [ 0 0 ; 5 0 ; 0 3 ; 5 3 ; 0 6 ; 5 6]

// Beam connectivity (6x2)
Dbl Beam_EL = [1 3 ; 2 4 ; 3 5 ; 4 6 ; 3 4 ; 5 6]

// Equation matrix  (6x3) : DOF of each node (tx,ty,rot)
Int EQ = [0 0 0 ; 0 0 0 ; 1 0 3 ; 1 0 4 ; 2 0 5 ; 2 0 6 ]

// Location Matrix (6x6)
Int Beam_LM = Beam_Make_LM( Beam_EL, EQ)

// Properties matrix [A As I E Nu Dn]
// 300mm x 300mm beams & columns
Dbl Properties = [ 0.3*0.3 0 0.3^4/12 25E9 0 0 ]

// Global Stiffness matrix for each element
Dbl KColumn = Beam_Make_K(1,XY,Beam_EL,Properties ,"G")
Dbl KBeam   = Beam_Make_K(5,XY,Beam_EL,Properties ,"G")

// Global stiffness matrix assembly
Dbl K = Zero(6)

// Add the element stiffness matrices to the global stiffness matrix
Assemble( K, KColumn , Beam_LM, [1,4] )
Assemble( K, KBeam   , Beam_LM, [5,6] )

// PART B : Static Analysis

// Nodal Force Vector :
Dbl F = [118;82;0;0;0;0]

// Solving displacement F = Ku
Dbl U = Solve(K,F)

// Force-Displacement matrices
Dbl FDColumn = Beam_Make_FD(1,XY,Beam_EL,Properties )
Dbl FDBeam   = Beam_Make_FD(5,XY,Beam_EL,Properties )

// Internal forces
Dbl Beam_IF= Zero(6,4)
Beam_IF(1,1) = Beam_Internal_Forces(FDColumn,U,Beam_LM,Beam_EL,1)
Beam_IF(1,2) = Beam_Internal_Forces(FDColumn,U,Beam_LM,Beam_EL,2)
Beam_IF(1,3) = Beam_Internal_Forces(FDColumn,U,Beam_LM,Beam_EL,3)
Beam_IF(1,4) = Beam_Internal_Forces(FDColumn,U,Beam_LM,Beam_EL,4)
Beam_IF(1,5) = Beam_Internal_Forces(FDBeam,U,Beam_LM,Beam_EL,5)
Beam_IF(1,6) = Beam_Internal_Forces(FDBeam,U,Beam_LM,Beam_EL,6)
```

Figure 9: Code for dynamic analyses