A NEW COMPUTER PROGRAM FOR THE SEISMIC ANALYSIS OF ARCH DAMS INCLUDING JOINT NONLINEARITY AND FLUID-STRUCTURE INTERACTION

Benedikt WEBER[1] And Hugo BACHMANN[2]

SUMMARY

Several effects are important in an earthquake analysis of an arch dam. Among the most important ones are the interaction of the dam with the reservoir and the foundation, and the non-linear behaviour of contraction joints. While interaction problems typically have to be treated in the frequency domain, nonlinear problems in the dam have to be analysed in the time domain. Algorithms for solving the interaction problem in the time domain as well as algorithms for treating the nonlinear behaviour of joints have been developed earlier. These algorithms are now being implemented in a new computer program named CODA.

The article starts with a brief review of the theoretical background. Then the main design and architecture are explained. This includes some innovative concepts like separating the model database from the algorithmic part and chaining several algorithms in a sequence with possible changes of the model between algorithms. Finally, the different software packages that build up the program are listed. Practical experience with the programming language C++ in this large software project is also given.

INTRODUCTION

In a realistic earthquake analysis of a dam, the interaction of the dam with the reservoir and the foundation, and the nonlinear behaviour of contraction joints are important effects to be considered. The interaction problems have been investigated mainly by Chopra and co-workers starting in the early sixties. These models are based on semi-analytical solutions in the frequency domain and are restricted to linear dam behaviour. The procedures have been implemented in different computer programs, the most recent one being EACD-3D-96 [Chopra 1996]. For the contraction joints, several models are given in the literature, mostly formulated as discrete joint elements. For an overview see [Hohberg 1992]. A classical implementation of a joint model is given in ADAP-88 [Mojtahedi 1992]. Also many general-purpose programs are capable of treating contact and friction and can be used to model joints. However, all these programs only consider simplified interaction models, neglecting wave effects by assuming the fluid to be incompressible and the foundation to be massless.

To combine a rigorous interaction model, like that by Chopra and co-workers, with a nonlinear joint model, it is necessary to transform the frequency-domain solution into the time domain. The classical way of the Fourier transform is unattractive because it is inefficient, both in memory and execution time. To provide a better analysis tool, new research has been performed at the Institute of Structural Engineering of the Swiss Federal Institute of Technology (ETH) in Zurich. Algorithms were developed that work efficiently in the time domain, first for the dam-reservoir interaction [Weber 1994] and later also for the two-dimensional dam-foundation interaction [Feltrin 1997]. Additionally, also a sophisticated joint element has been developed [Hohberg 1992]. However, these algorithms were all implemented in separate programs. The final step of incorporating them into a single computer program is the main theme of this paper.

[1] *Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, Email: weber@ibk.baum.ethz.ch*
[2] *Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, Email: bachmann@ibk.baum.ethz.ch*

THEORETICAL BACKGROUND

Interaction problems in the frequency domain

The dynamic behaviour of an arch dam will be influenced by its foundation rock and its reservoir. However, since these adjacent domains are of large extent they are not usually included in a finite element model. Rather, the finite element model is restricted to the dam itself and a nearby region called the nearfield. The faraway region, called the farfield, is approximately captured by transmitting boundaries. Transmitting boundaries are artificial boundaries that represent the farfield by imposing an appropriate boundary condition that lets waves pass through. Simple fixed or free boundary conditions are inappropriate because they reflect outgoing waves and neglect energy radiation into the farfield. The situation is schematically depicted in Figure 1.

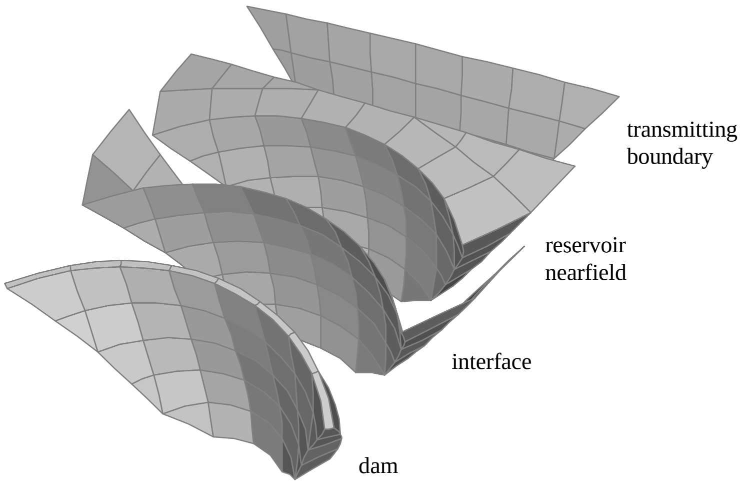

Figure 1: Arch dam with nearfield reservoir and transmitting boundary.

There are several possibilities to construct transmitting boundaries, but the most rigorous one is by semi-analytical methods. In these methods, the farfield is assumed to be of constant cross-section corresponding to the transmitting boundary, and of infinite extend normal to the transmitting boundary. The differential equation can then be solved by separation of variables, where the cross-section is solved numerically and the infinite direction analytically. These models include wave propagation and the proper radiation boundary condition at infinity. This is important, for instance, when modelling compressibility effects in the water. The formulation leads to an eigenvalue problem, and from its solution the frequency-dependent impedance matrix (dynamic stiffness matrix) $\mathbf{H}(\omega)$ of the infinite domain can be calculated. This is the boundary condition of the transmitting boundary. Details are given in [Weber 1994] and [Feltrin 1997]. Because the impedance matrix is frequency-dependent, it cannot directly be used in a time-domain analysis.

Time domain approximation

The classical way of transforming $\mathbf{H}(\omega)$ into the time domain, is to apply the Fourier transform. However, $\mathbf{H}(\omega)$ is matrix-valued and the number of frequencies can be quite large, resulting in a large number of data in the transformed matrix $\mathbf{H}(t)$. The large number of data and the inefficient calculation of the interaction forces by convolution make this procedure unattractive. Therefore, the search for more efficient time-domain procedures has a long tradition. A family of algorithms has been developed at the institute of the authors and is presented elsewhere [Weber 1994, Feltrin 1997]. Here we only repeat the main ideas and the general form as far as it determines the overall structure of the program. As described in the previous section, the interaction analysis is first performed in the frequency domain. The transfer function $\mathbf{H}(\omega)$ is then approximated by a linear time-invariant system using system theory. The main step in this approximation is the singular value decomposition of the Hankel matrix. To fit into the finite element framework, the system has to be second order and symmetric. It can be expressed as

$$\mathbf{H}(\omega) = (i\omega\mathbf{B}_1 + \mathbf{B}_2)^{\mathrm{T}}(-\omega^2\mathbf{A}_0 + i\omega\mathbf{A}_1 + \mathbf{A}_2)^{-1}(i\omega\mathbf{B}_1 + \mathbf{B}_2) + i\omega\mathbf{D}_1 + \mathbf{D}_2\,.$$

$\mathbf{A}_0$, $\mathbf{A}_1$ and $\mathbf{A}_2$ are the mass, damping and stiffness matrices, respectively, of the approximate linear system. $\mathbf{B}_1$ and $\mathbf{B}_2$ are coupling matrices and $\mathbf{D}_1$ and $\mathbf{D}_2$ are the direct input matrices. In the time domain, these matrices can directly be combined with the corresponding finite element matrices of the nearfield as schematically depicted in Figure 2.

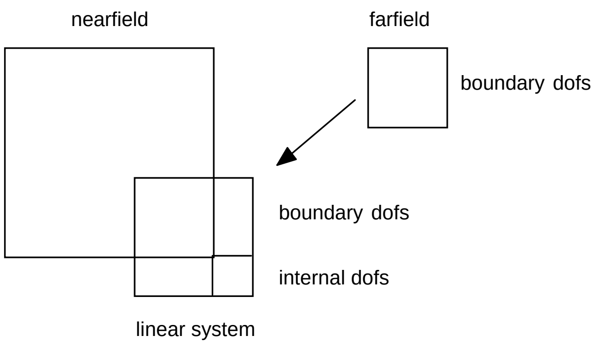

Figure 2: Linear system added to finite element matrices.

The size of the original matrix $\mathbf{H}(\omega)$ is given by the number of degrees of freedom at the transmitting boundary. The approximated linear system has additional internal degrees of freedom. As can be seen from the above explanations, transmitting boundaries cannot be implemented solely on an element basis but require some global procedure. Each transmitting boundary constitutes a submodel that has to be analysed by itself before it is added to the complete model.

Nonlinear joints

A sophisticated joint model developed earlier is presented in [Hohberg 1992]. It is implemented as a discrete element with a force-displacement (rather than a stress-strain) constitutive law. It includes opening and closing, Coulomb friction, interlock, and tensile strength. Contact is implemented by a penalty formulation. The penalty formulation is simple to program but the parameter has to be chosen right. It has to be large enough to prevent excessive penetration, but not too large, to prevent ill-conditioning of the global stiffness matrix. The integration of the constitutive law is performed by a Runge-Kutta algorithm. This integration algorithm has been chosen for its adaptability to complicated constitutive models. However, it needs a large number of subincrements for good accuracy, especially for the large trial stresses in the radial direction typically encountered with a penalty formulation. Since friction leads to a non-associative constitutive law, the tangent stiffness matrix is not symmetric. To preserve symmetry in the global stiffness matrix, only the symmetric part was taken.

Considering these difficulties, it was decided to reformulate the joint element using an augmented Lagrangian method for contact, and a radial return algorithm for friction, neglecting the more complicated phenomena of interlock and tensile strength for the time being. The augmented Lagrangian method can be viewed as a generalisation of the penalty formulation. The solution of the contact constraints is found iteratively, where the contact force is evaluated as the force from the previous iteration corrected by a force resulting from a low penalty. The radial return algorithm is today the algorithm of choice for friction and plasticity and does not need subincrementing. In combination with the augmented Lagrangian method for contact, the consistent tangent matrix can be made symmetric [Laurson 1993].

There are other points to be considered with contact, which are not restricted to the elements. To avoid spatial oscillations in the contact surface, the element stiffness should be decoupled. This can be achieved by choosing Lagrange elements and a Lobatto quadrature. As a consequence, the adjacent brick elements need to have a mid-face node for compatibility. Another difficulty is the high-frequency noise present with contact-impact problems. To damp it out, numerical damping of the time integration scheme is mandatory. In [Hohberg 1992], also viscous damping in the joint elements was proposed but not used in the dam examples.

PROGRAM DESIGN AND IMPLEMENTATION

Design philosophy and architecture of program

The algorithms given in the previous sections are implemented in a new computer program named CODA (COde for Dam Analysis). The following is a brief description of the design philosophy and the program architecture. It explains the main components of the program and the general flow of information, starting from the input file, going through the analysis steps, and ending with the results. The different software packages that are used to build the program are described in the next section.

The design of the program was guided by various decisions and requirements. One requirement was that two, and sometimes three programmers had to be working on different parts more or less independently. Another decision was to provide an input database and an output database as the main front end and back end. The input database (model) can be constructed by a parser that reads in a text file or by a preprocessor, the output database can be printed or written to a file to be viewed by a postprocessor. This philosophy makes the program independent of a specific pre- or postprocessor. Another point is the separation of the elements and the loads into a definition part and a computational part. The design of the program was also determined by other requirements. Different algorithms will need to be processed in sequence, and different structures (substructures of the nearfield, transmitting boundaries) will be added or removed. This flexibility is typically needed when simulating the construction stages and the loading history of a dam. The present design should be contrasted with that of other dam programs, where typically a static analysis is followed by an earthquake analysis in a predetermined way and intermediate results are transferred via a file.

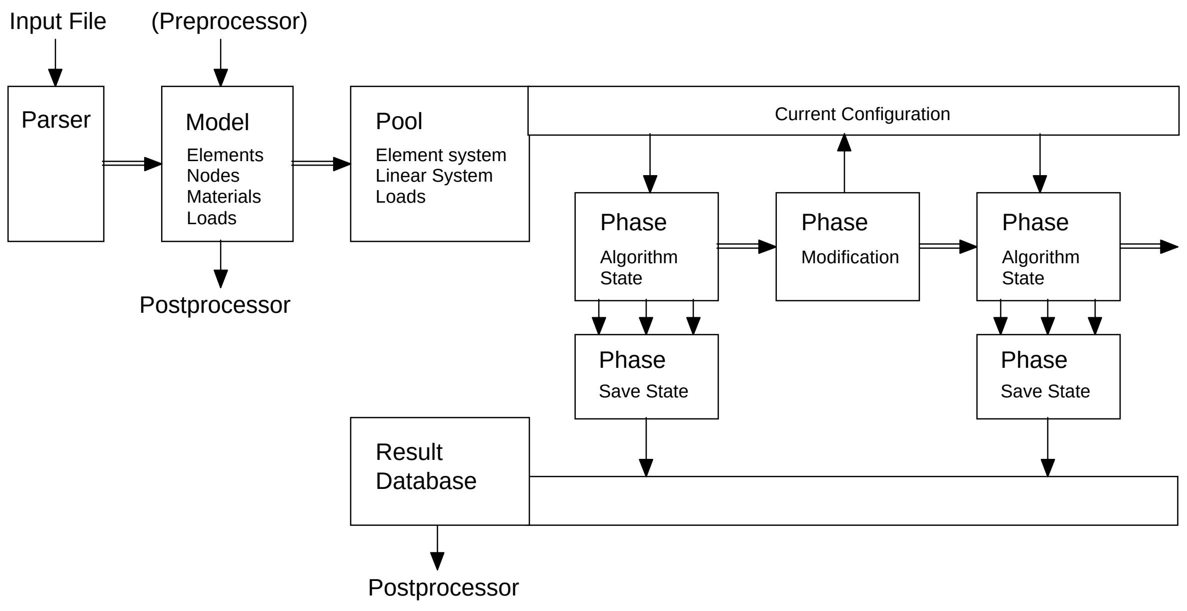

Figure 3: Architecture of CODA

Figure 3 shows the principal parts of the program with the data flow indicated by arrows. The main data flow is given by double arrows. The process starts with the parser reading the input file and constructing the model. The model is the database containing all definitions of structures (nodes, elements, materials) and loads (forces and boundary conditions). Then the data are transformed from the pure definitions of the model to computational components in the pool. For the elements, shape functions and quadrature points are calculated, and the procedures for calculating the element matrices are defined. Element loads are integrated to nodal loads. The pool has a repository of all elements and loads, and an actual configuration that defines what elements and loads are active at the time. The different analysis steps are represented by phases with an algorithm and a state. A first phase is typically a static analysis with the state being the nodal displacements. Another phase is a transient analysis with the nodal displacements, velocities, and accelerations as its state. In between, there are modification phases, which change the actual configuration by adding and removing structures and loads. For instance, after a static analysis of a dam, a modification phase would add the reservoir with the transmitting boundaries and the earthquake load. Subphases of the algorithmic phases are used to transfer selected results to the result database.

Implementational aspects

Programming language

Finite element programs have traditionally been implemented in Fortran 77. Most of the numerical part of a finite element program is commonly formulated in terms of matrices, and these are well supported in Fortran. A large portion, however, is concerned with the organisation of data and goes beyond the simple matrix scheme. Modern programming languages are more powerful and flexible, but years of experience and standard techniques make Fortran still the main programming language for finite elements. Fortran 95 and C are also used more and more, but object-oriented languages are still not widely employed in this field. Although there are quite a few examples on the Internet, most of them seem to be primarily studies in purely object-oriented design applied to finite elements rather than finite element programs implemented in an object-oriented way. Purely object-oriented design should not be considered a goal but a means of good program design, along with other concepts. The programming language C++ [Stroustrup 1997] was felt to best fit the needs of a finite element program, because it is designed to be flexible and at the same time as efficient as other high-level programming languages like C and Fortran. C++ also has now an ANSI standard describing all details of the language (there are a lot of them) and also includes the standard library, that contains a flexible and efficient implementation for containers like lists or maps (tables).

The main difference between C++ and C or Fortran is its organisation into classes. They are the counterparts of C functions and Fortran subroutines. Classes are structures with data and functions, which can be used as new data types. Classes are used, for instance, for vectors and matrices but also for elements and even for algorithms. Classes are used as arguments for functions and as building blocks for other classes. This leads to a modular design, allowing building up complex programs. An important aspect of modular design is abstraction. Different classes sharing the same interface represent the same abstract concept and can be used uniformly. For instance, all elements give a contribution to the global stiffness and mass matrices. The element matrices itself are calculated differently, depending on whether the element is a solid element or a fluid element, but they are all passed the same way to the global matrix. C++ has two equally important concepts to support abstraction: inheritance and templates. Inheritance is used if the real class is not known at compile time as in the example of elements. The actual type is determined at run time. Examples for template classes are matrices. The compiler knows which matrices are real and which are complex and can use this information to produce more efficient code and better type checking.

Practical experience

Starting a large project with a new programming language is exciting and also surprising sometimes. Many of the problems only show up after the code has reached a certain size. The language is relatively difficult to learn, at least if it is used with all the different features. Switching from procedural programming in Fortran to object-oriented programming in C++ requires a complete change in the way of thinking. Since programming has been started before the ANSI standard has been evolved, many features were not available at the beginning of the project, and even now that the standard is completed, compilers sometimes still lack strict conformance.

The large size of a program leads to many dependencies among the classes. If no attention is given to that fact, it can easily happen that a single change in a class requires most of the program to be recompiled. Organising the code into a hierarchy of packages that can be compiled and tested independently is, therefore, essential [Lakos 1996]. Code reuse helps keeping the code smaller but requires careful design. There is quite a long start-up time until this investment pays back. This was sometimes grossly underestimated. The gain is not so much in time but in design quality. However, in the long run, a well-designed program should take less time to maintain.

The compiler is very strict at type checking, which helps avoiding a lot of inconsistencies. Although it can take a while to get the code free of syntax errors, once compiled and linked, there are usually not many errors, and debugging takes considerably less time than in Fortran. A lot of debugging can also be avoided by providing testing programs for each class.

Teamwork development was much easier than in Fortran. Different parts of the program were developed independently by different programmers. This was possible by designing the interfaces and providing test implementations. For instance, all algorithms were developed without elements, just with a testing class implementing the interface with a simple example problem. Using descriptive names (there is no length limit) simplifies understanding, while strict type checking and testing all classes detects most inconsistencies arising from code modifications.

Overview of classes and packages

By now, the program consists of over 60,000 lines of code, constituting more than 430 classes organised in some 20 packages. As described earlier, a program of this size has to be broken up into packages that can independently be compiled and tested. To give an idea of how the packages depend on each other, Figure 4 shows a simplified version of the hierarchy. The actual packages are many more, each of the shown boxes representing 2 or 3 of them.

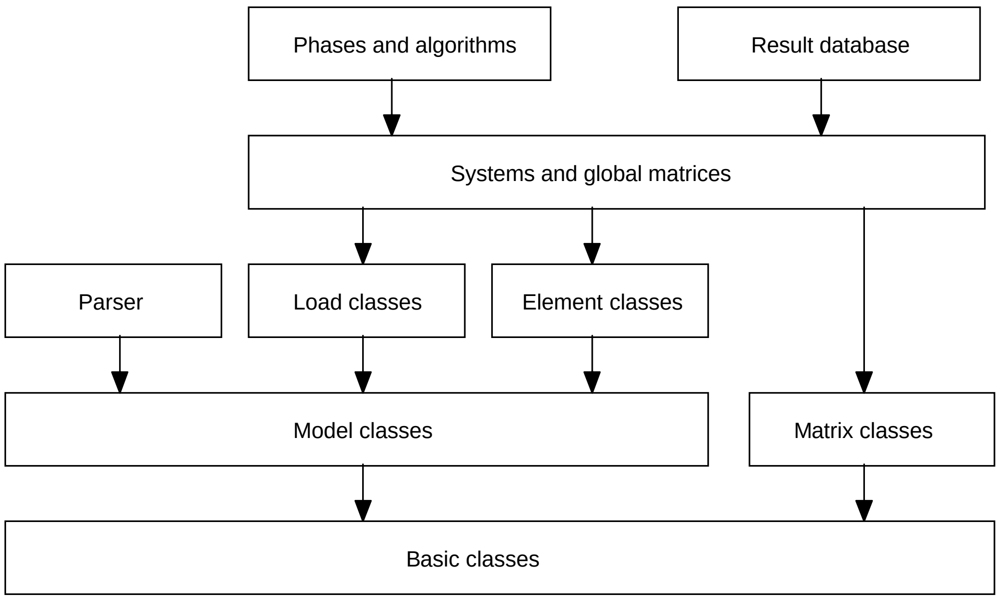

Figure 4: Packages with dependencies

It follows a brief description of the packages:

Basic classes: Basic classes define data types and utilities that are used frequently throughout the program. They include pointer classes, containers, geometry classes like plane, axis, and transformation, and also fixed-size real vectors and matrices for 1, 2, and 3 dimensions used in all classes that depend on the space dimension.

Matrix classes: Vectors and matrices are implemented as template classes, allowing them to be used with integer, real, or complex data. They include memory management, operators for algebraic expressions, functions for linear algebra, and printing. Most linear algebra functions, like solving linear equations or eigenvalue computations, are wrappers to Fortran subroutines from LAPACK [Anderson 1995] and ARPACK [Lehoucq 1998]. One special feature is the singular value decomposition of the Hankel matrix. With the iterative algorithms provided by ARPACK, only the defining sequence of the Hankel matrix has to be stored, and only a few singular values have to be calculated.

Parser: The parser reads the text-input file and builds up the model. Text input is free format and simply readable with key words. The parser consists of the scanner, reading the input file, and the actual parser, which does the interpretation. It has a table with real variables, built-in functions, and user-defined functions, which can be used to generate mesh data for complex dam shapes.

Model classes: The model consists of nodes, elements, materials, loads, and boundary conditions. As stated earlier, these items are merely definitions of the input. The elements and loads that perform the actual calculations are described later. Time-varying nodal loads, and element surface and volume loads are supported. A model can be split up into several structures (substructures), each having its own numbering scheme. Thus, each structure (e.g. the dam or the reservoir) can be modelled independently. Each structure can be composed of several parts, that themselves can contain parts. Parts provide a scope for element types and materials, and can be used as groups for mesh generation. In addition, set-like groups can be defined across different parts. For input check, the model can be printed or saved to a file for viewing it with a postprocessor. Currently, nodes and elements can be written to a FEMAP neutral file. Group information from the input is retained and can be used, for instance, to draw selected parts of the model. Other information, such as loads, or other postprocessors will be supported later.

Element classes: The element classes implement the computational side of elements. There are classes for local and global shape functions, Jacobian matrices and quadrature schemes. One-, two-, and three-dimensional, linear and quadratic shape functions are included. Local derivatives are constructed by automatic differentiation. The elements themselves are linear fluid, solid, and interface elements. Variable-number-of-nodes brick elements are included for compatibility with 9-node joint elements. The fluid and interface elements also have their transmitting boundary elements. These are 2D elements for a 3D problem, and 1D elements for a 2D problem. In addition to the mass, damping, and stiffness matrices, these elements have to provide matrices that account for the additional space dimension. (Joint elements and transmitting solid elements will be added later.)

Load classes: The load classes contain time-varying loads and boundary conditions. All loads and boundary conditions are defined on nodes. Element loads are integrated to nodal loads.

Systems and global matrices: In a finite element program, the algorithms work with global matrices and vectors, numbered by global degrees of freedom. They do not directly communicate with the elements, but the elements give individual contributions to global matrices and vectors. In the presence of transmitting boundaries, there are also contributions from linear systems to the global matrices and vectors. The connection of the elements and the linear systems to the global matrices and vectors is implemented by the system interface. With the system interface, an algorithm can send different requests to the elements and linear systems to give their contributions in a uniform manner. There is a global skyline matrix with an internal mapping of global degrees of freedom to equations. The equation numbers are optimised for a small profile.

Phases and algorithms: Phases are used to construct a sequence of processes. Some phases run algorithms, other do modifications of the current configuration of elements, linear systems, and loads, and others save results to the result database. Phases with algorithms also have a state, containing, for instance, the displacements. A phase has access to the state of a previous phase, to calculate, for instance, initial conditions or to save selected results. Static, transient, steady state, and eigenvalue algorithms are included. Several transient algorithms are implemented providing various forms of numerical damping (Newmark, Hilber-Hughes-Taylor, generalised-alpha). There are also several algorithms for constructing linear systems for the transmitting boundaries.

Result database: There are basically two different kinds of results that are commonly used. One is a complete time history of a few selected variables, which can be plotted in a x/y plot. The other is a snap shot of the whole or part of the model at selected time steps, which can be used in a geometry plot. Both needs can be satisfied by grouping the results into tables, each with its own variables and time steps. Tables intended for x/y plots can be exported to files for further processing by a spreadsheet software, tables intended for structural plots can be exported to files to be viewed by a postprocessor. (The result database is still under development.)

GENERAL REMARKS

Status of program

At the time of this writing (August 1999) most parts of the program are working, although there are still some major parts missing, among them the joint elements and the result database. No large-scale studies have been performed yet, but all classes are tested by some hundred test programs.

Acknowledgements

The authors wish to thank Dr. Glauco Feltrin, who was working on the project for a long time and has developed large parts of the program. The present work has been partially supported by the Swiss Federal Office for Water Management.

CONCLUSIONS

1. In a realistic earthquake analysis of a dam, both the interaction problems and the nonlinear joint behaviour are important. Algorithms that allow considering both of these effects have been developed and implemented in a new computer program.

2. For the program, an input database and an output database are designed that make it independent of a specific pre- or postprocessor. A parser reads text input and supports generating complex dam shapes.

3. The input database is decoupled from the computational components. The latter are kept in a repository and can be added to or removed from the current configuration, allowing changing the model between analysis steps.

4. Algorithms for static, transient, steady state, and eigenvalue analysis are provided. There are transient algorithms with improved numerical damping for reducing the high-frequency oscillations originating from joint contact. There are also several algorithms for linear system approximation used for the transmitting boundaries.

5. The programming language C++ is well suited for numerical engineering applications. However, learning the language, gaining experience with the design and maintaining a large software project is a great challenge. The iterative process of designing and redesigning can be more time-consuming than expected.

6. The new program CODA is not only a major step from a theoretical point of view, but it is also innovative in its design and implementation in the programming language C++.

REFERENCES

Anderson E., Bai Z., Bischof C., Demmel J., Dongarra J., Du Croz J., Greenbaum A., Hammarling S., McKenney A., Ostrouchov S. and Sorensen D. (1995). *LAPACK Users' Guide*, second edition. SIAM, Philadelphia.

Chung J. and Hulbert G.M. (1993). “A time integration algorithm for structural dynamics with improved numerical dissipation: the generalized-alpha method”. *Journal of Applied Mechanics*, vol. 60, 371-375.

Feltrin G. (1997). Absorbing boundaries for the time-domain analysis of dam-reservoir-foundation systems. IBK report No. 232, Birkhäuser Basel.

Hohberg J.M. (1992). A joint element for the nonlinear dynamic analysis of arch dams. IBK report No. 186, Birkhäuser Basel.

Lakos J. (1996). *Large-Scale C++ Software Design*. Addison-Wesley.

Laurson, T.A. and Simo J.C. (1993). “Algorithmic symmetrization of Coulomb frictional problems using augmented Lagrangians”. *Computer Methods in Applied Mechanics and Engineering*, vol. 108, 133-146.

Lehoucq R. B., Sorensen D. C. and Yang C. (1998). ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods. SIAM, Philadelphia.

Miranda I., Ferencz R.M. and Hughes J.R. (1989). “An improved implicit-explicit time integration method for structural dynamics”. *Earthquake Engineering and Structural Dynamics*, vol. 16, 643-653.

Mojtahedi S., Fenves G.L. and Reimer R.B. (1992). ADAP-88: a computer program for nonlinear earthquake analysis of concrete arch dams: user guide. UCB/SEMM-1992/11, Dept. of Civil Engineering, Univ. of California, Berkeley.

Stroustrup B. (1997). *The C++ Programming Language*, 3rd edition. Addison-Wesley.

Tan H. and Chopra A. K. (1996). EACD-3D-96: A computer program for three-dimensional earthquake analysis of concrete dams. UCB/SEMM-1996/06, Dept. of Civil and Environmental Engineering, Univ. of California, Berkeley.

Weber B. (1994). Rational transmitting boundaries for time-domain analysis of dam-reservoir interaction. IBK report No. 205, Birkhäuser Basel.