AN RFQ SIMULATION CODE*

W. P. Lysenko, AT-6, MS-H829
Los Alamos National Laboratory, Los Alamos, New Mexico 87545 USA

Summary

We have developed the RFQLIB simulation system to provide a means to systematically generate the new versions of radio-frequency quadrupole (RFQ) linac simulation codes that are required by the constantly changing needs of a research environment. This integrated system simplifies keeping track of the various versions of the simulation code and makes it practical to maintain complete and up-to-date documentation. In this scheme, there is a certain standard version of the simulation code that forms a library upon which new versions are built. To generate a new version of the simulation code, the routines to be modified or added are appended to a standard command file, which contains the commands to compile the new routines and link them to the routines in the library. The library itself is rarely changed. Whenever the library is modified, however, this modification is seen by all versions of the simulation code, which actually exist as different versions of the command file. All code is written according to the rules of structured programming. Modularity is enforced by not using COMMON statements, simplifying the relation of the data flow to a hierarchy diagram. Simulation results are similar to those of the PARMTEQ code, as expected, because of the similar physical model. Different capabilities, such as those for generating beams matched in detail to the structure, are available in the new code for help in testing new ideas in designing RFQ linacs.

Introduction

We desired a new RFQ simulation code to test some new ideas for designing RFQ linacs. Frequent modifications to the code would be necessary in this application. Our code development work considered, from the very beginning, the problem of producing a software product with the required flexibility. Ease of modification and of managing the many versions that would inevitably arise were the major concern for the programming effort.

From the physics point of view, the primary concern was the ease of changing the space-charge and external-force computations in the simulations. Also, we thought it important to be able to generate particle phase-space distributions, that are matched in detail to the accelerator structure, and to verify matched conditions. This capability is useful both to check for numerical emittance-growth effects and to prevent mismatch effects from obscuring other effects under study.

We achieved our goals by forming a library file containing what we call the standard version of the simulation code. New versions of the simulation code can be generated, as required, by means of command files that contain commands to compile any new or modified subroutines and link them to the standard versions in the library file. This collection of files we call the RFQLIB system. The rules of structured programming were followed in all coding to make the inevitable modifications easy to make.

Organization of the RFQLIB System

There are four kinds of files in the RFQLIB system.

1. Library File (source or object code). This file contains the main program and all the subroutines for the standard version of the simulation code. The object code can be loaded and run but is really meant to be used as a library with the command files.

2. Command Files. These files contain source code for routines that have been changed from the library version. They also contain the commands necessary to compile the new routines and to link them to the library routines to produce an executable code.

3. Data Files. There are three types of data files providing information on the input distribution, accelerator parameters, and simulation parameters.

4. Output Programs. These programs process output from the simulation codes, usually to generate graphical output.

The standard command file is called RFQRZP*C. We use one of the available text editors as a controller to execute the commands in the command file. The standard command file contains no source code except for a dummy routine that references a routine in the library file, so that executing the commands in the file results in an executable file, called RFQRZP*X, consisting of the main program and subroutines in the library file. If a new version of the simulation code is desired, the new subroutines are appended to the standard command file. Any subroutines in the command file that have the same name as those in the library will replace those in the library when the executable code is produced. The library itself is not meant to be changed often. When a change is made to the library, however, these changes are seen by all versions of the simulation code. Different versions of the simulation code are stored as different versions of the command file.

The code structure for the standard version is shown in fig. 1. The important feature of this structure is that the external or space-charge forces can be changed without any knowledge of beam dynamics or numerical methods. This modularity is a feature of structured programming.

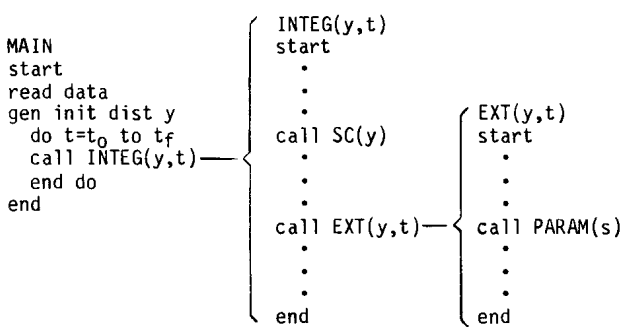

Fig. 1. High-level code structure for the standard version of the simulation code.

*This work was supported by the US Dept. of Defense, Defense Advanced Research Projects Agency, and Ballistic Missile Defense Advanced Technology Center.

Programming Aspects

Let us describe what we mean by structured programming as it is applied to this project. Originally, structured programming was a management technique that allowed a number of people to work simultaneously on a large software project. But structured programming is useful even for small one-man projects because it makes programs more understandable and easy to modify. These are precisely the qualities we desire in a code that is to be constantly modified to meet changing requirements. One way of stating the rules of structured programming is the following: Programs must be modular and the modules must be connected in a simple manner.

A simple thing to do is to consider the FORTRAN subprograms as modules of the overall program. Then, one consequence of our rules is that COMMON statements are not allowed. In a program with COMMON statements, it is impossible to determine where variables are defined or redefined without a detailed analysis of the whole program. Suppose we have a program with no COMMON statements, such as one of the simulation codes in the RFQLIB system. Then a hierarchy diagram (not shown in this paper) containing data-flow information can be extremely useful. The main program is shown at the top of the diagram and the most deeply nested subprograms are at the bottom. The dummy argument list associated with each subprogram is shown on the diagram. If any actual argument list is different from the dummy argument list, the actual argument list is shown associated with the line connecting the calling and the called subprograms. To modify one of the subprograms, only the code for the given subprogram needs to be examined. How the subprogram relates to the rest of the program can be learned from looking at the hierarchy/data-flow diagram.

Within a subprogram, a consequence of our rules is that GO TO statements are not allowed. (Even better is to eliminate statement labels.) The reason is that a section of code containing a statement label cannot be understood without looking at all the code in the given subprogram to see if it contains a GO TO statement pointing to the statement label. The reason for the rules again is the same: To understand a section of code (necessary if modifications are required), it should not be necessary to look in detail at all the rest of the code. In our project, we found that the higher level rule (no COMMON) was more important than the lower level rule (no GO TO).

Physics and Numerical Aspects of the Standard Version

The standard version of the simulation code is a particle-in-cell code computing space charge on a uniform r-z mesh. The boundary conditions for the Poisson solver are a conducting circular cylinder at a given radius and periodic boundary conditions in the axial direction with period $\beta\lambda$. Time is the independent variable. The particle coordinates and momenta and the position and velocity of the synchronous particle are the dependent variables, using x-, y-, z-coordinates. The equations are integrated using a first-order, explicit, symplectic integrator.

There are certain features we built into the simulation code that we considered important for our RFQ design work. The ability to create beams matched in detail to the accelerator structure was accomplished by facilities for adiabatic deformation as described in the example below. This feature required the introduction of nonphysical forces into our model of the RFQ.

Simulation Examples

Comparison with PARMTEQ

We ran a simulation of the accelerator test stand (ATS) RFQ linac at Los Alamos[1] to compare the new code with PARMTEQ, the standard RFQ simulation code at Los Alamos. We found very good agreement. The transmission factor for the standard-design input beam was 92% for the PARMTEQ run and 94% for the new code. Emittances differed by less than 5%, a value within the statistical accuracy of representing the phase-space distribution by a finite number of particles. The fundamental difference between the two codes is that PARMTEQ has free-space boundary conditions and the new code has a conducting wall. The presence of the conducting wall decreased the axial space-charge field by several per cent at most. This was not a large-enough effect to affect the transmission factor or final emittance.

Matched and Acceptance Beams

The generation of phase-space distributions matched in detail to a periodic structure can be done by adiabatic deformation.[2] The previous work did not include space-charge forces. The new code has provisions for using this method to generate high-brightness beams matched to an RFQ structure. Such a matched beam has been generated for the ATS RFQ matched to a point 1 m from the input. This was a 100-mA beam with a normalized rms transverse emittance (area in $x\text{-}p_x$ phase space, divided by πmc) of 0.2 mm•mrad. When traced through an exactly periodic structure corresponding to the 1-m point of the RFQ, the matched beam is very nearly periodic. When we consider rms beam sizes that are one rf period apart, the beam-size fluctuations are less than 2%.

In the presence of space charge, the acceptance of an accelerator is not unique. We can define an acceptance distribution to be one that, if injected into the accelerator, results in a beam well matched over the whole length of the accelerator. Because the parameters at the beginning of the accelerator are rapidly changing, obtaining a beam matched to a periodic structure corresponding to the RFQ entrance does not work well. We obtained a good acceptance distribution by taking the beam matched to the 1-m point of the RFQ and tracing it backward to the accelerator entrance. The 2D phase-space projections of this acceptance beam are shown in fig. 2. Also shown for comparison is the standard input beam used in the RFQ design process.

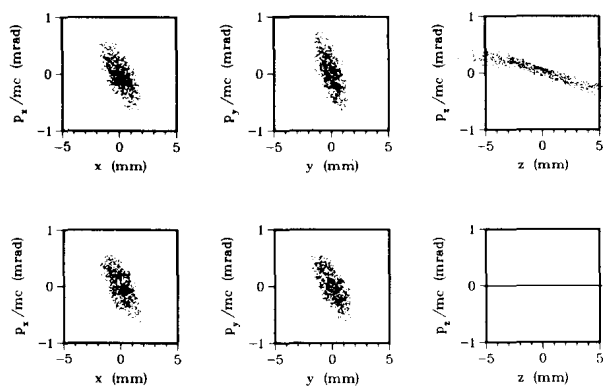

Fig. 2. Phase-space projections for the acceptance beam (top row) and the standard input beam for the ATS RFQ (bottom row).

The acceptance beam has a transmission factor of 100% (no particles lost) and practically no emittance growth. Of course, creating such beams is not practical in the laboratory. We believe the utility of such computations is in designing the low-energy part of RFQ linacs. We can vary the RFQ parameters and compute the acceptance distributions for the various cases, choosing the case for which the acceptance distribution is closest to a physically attainable distribution.

Discussion

The new RFQLIB simulation system has been designed to be easily extended to include new versions of the code as new requirements arise. We believe this technique to be very successful. Storing a standard version of the simulation code as a library simplifies the maintenance of the various versions of the code. Because of the integrated nature of this system, a single manual documenting the whole system has been written.[3]

The capability of generating matched distributions by adiabatic deformation is an important feature of the standard version of the simulation code. This allows us to factor out input beam effects from accelerator-structure characteristics. For example, we have learned that the ATS RFQ design can transmit a 100-mA, 0.2 mm•mrad emittance beam with no transmission loss, provided the input distribution is properly chosen.

References

1. E. A. Wadlinger, J. A. Farrell, and H. O. Dogliani, "A High-Brightness Negative Hydrogen Linear Accelerator," 7th Conf on Application of Accelerators in Research and Industry, Denton, Texas, November 7-10, 1982.

2. W. P. Lysenko, "Matching Bunched Beams to Alternating Gradient Focusing Systems," IEEE Trans, Nucl. Sci. 28, 2516 (1981).

3. W. P. Lysenko, "The RFQ Simulation Code Library RFQLIB and Associated Codes," Los Alamos National Laboratory unpublished report ATN-84-1 (January 1984).