U.S. DEPARTMENT OF THE INTERIOR
U.S. GEOLOGICAL SURVEY

SMSIM — Fortran Programs for Simulating Ground Motions from Earthquakes: Version 1.0

by

David M. Boore[1]

Open-File Report 96-80-A

[1]U.S. Geological Survey, MS 977, 345 Middlefield Rd., Menlo Park, CA 94025

SMSIM — Fortran Programs for Simulating Ground Motions from Earthquakes

TABLE OF CONTENTS

SMSIM — Fortran Programs for Simulating Ground Motions from Earthquakes

by

David M. Boore

INTRODUCTION

This Open-File Report is in response to requests for my programs for simulating ground motions from earthquakes. The programs are based on modifications I have made to the stochastic model first introduced by Hanks and McGuire (1981). The report contains source codes, written in Fortran, and executables that can be used on a PC. Programs are included both for time-domain and for random-vibration simulations. In addition, programs are included to produce Fourier amplitude spectra for the models used in the simulations and to convert shear velocity vs. depth into frequency-dependent amplification. The report contains an improvement in the implementation of the random-vibration method not published before.

The programs do not include extended-fault models, nor do they account for path and site effects by direct computations of wave propagation in layered media (but such path and site effects can be captured in the program by piecewise-continuous frequency- or distance-dependent functions specified by the user). Furthermore, the random-vibration calculations do not make use of the many advancements in random-vibration theory subsequent to the early work of Cartwright and Longuet-Higgins (1956).

The programs are a recent major revision of my earlier programs and therefore almost certainly contain uneradicated bugs. Although they are distributed on an "as is" basis, with no warranty of support from me, I would appreciate hearing about bugs and improvements to the codes. Please note that I have made little effort to optimize the coding of the programs or to include a user-friendly interface. Speed of execution has been sacrificed in favor of a code that is intended to be easy to understand. I will be pleased if users incorporate portions of my programs in their own applications.

Other stochastic-model codes are available and in common use. In particular, the reader is directed to the programs in Volume VIII of Herrmann (1996) and *RASCAL* by Silva and Lee (1987). I have not made a detailed comparison of my codes to these other codes and therefore cannot make any statements about the relative strengths and weaknesses of the various codes.

The programs can be obtained via anonymous ftp on *samoa.wr.usgs.gov* in directory *get*. The source code, executables, sample input and sample output have been compressed into a single self-extracting binary file with the name *SMSIM10.EXE*. After copying this file to the user's PC, the files can be extracted by typing the name *SMSIM10*. The portion of the file name with the version number ("10" in this case) will change if the program is modified.

METHOD

A description of the method is given in Boore (1983), Boore and Joyner (1984), Boore (1986), and Joyner and Boore (1988), and will not be repeated here, other than to say that the radiation from a fault is assumed to be distributed randomly over a time interval whose duration is related to the source size and possibly the distance from the source to the site. The detailed parameters used to characterize the source, path, and site effects are described later in this report.

The ground motion can be obtained via time-domain (*TD*) simulation, from which peak parameters such as peak acceleration and response spectra can be obtained (mean values of the parameters require a Monte Carlo simulation with many realizations for given input parameters). The peak parameters also can be obtained directly using random-vibration (*RV*) theory[†]. This is a much quicker way of obtaining the peak parameters, but it is not useful if time series are needed in the analysis. In addition, there are assumptions in the random-vibration theory that are not present in the time-domain simulations. For this reason, the time-domain simulations can be considered "truth" in the simulations; many simulations are needed (on the order of 50 or more), however, in order for the square-root-of-n reduction of noise to provide accurate estimates of the peak parameters. In general, I have found the random-vibration simulations to be good estimates of the ground motions in almost all cases, at greatly reduced computer time.

This report contains an improvement in my implementation of random-vibration theory. That an improvement was needed is shown in Figure 1, which compares the response spectrum computed using *RV* and *TD* simulations. The heavy line is the *TD* simulation, and the dashed line is the result from what used to be the preferred *RV* method. The results from the two methods track one another very well, except for certain period ranges where the *RV* results show discontinuous changes in value. I first noticed

[†]Some would prefer the term "random-process theory"; I have used "random-vibration theory" because many of the applications are to the vibrations of harmonic oscillators and because the term is more familiar to engineers.

these changes several years ago, but I had no explanation for them. I now understand why they occur, and I have found a way to prevent them (the improved results are given by the circles). The explanation has to do with how I treated the following integral from Cartwright and Longuet-Higgins (1956; their equation (6.8)):

$$\frac{1}{\sqrt{2}} \int_0^\infty \{1 - [1 - (1 - \epsilon^2)^{1/2} e^{-\Theta}]^N\} \Theta^{-1/2} \, d\Theta, \tag{1}$$

where ϵ is computed from the spectral moments and is a measure of the bandwidth of the spectrum, and N is the number of extrema, proportional to the square root of the ratio of the fourth and second spectral moments. This integral is the ratio of the peak and *rms* motions, which for our purposes is the fundamental piece of information provided by random-vibration theory. (Cartwright and Longuet-Higgins' equation (6.8) is an approximation to their equation (6.4); the code for computing equation (6.8) is simpler than that for equation (6.4), and judging from the comparisons with time-domain calculations in this report and other comparisons that I have made, it is an excellent approximation for the ranges of magnitudes, distances, and oscillator periods of interest in earthquake engineering. Using equation (6.4) would also require redoing the analysis of Boore and Joyner (1984) for determining the duration used to compute the *rms*— the Boore and Joyner results are based on equation (6.8)). As described in my first paper on the stochastic model (Boore, 1983), I expanded the term in square brackets using the binomial series and integrated term-by-term. This gave equation (21) in Boore (1983). The expansion assumes that N is an integer, but N is computed from spectral moments and in general is not an integer. In the calculations shown by the dashed line in Figure 1, however, the real number N was converted to an integer to determine how many terms of the series to include in the sum (equation (21) in Boore, 1983). For small N (e.g., for long-period oscillator response for short-duration earthquakes), changes by one integer lead to the discontinuous offsets seen in Figure 1. The solution to this is simple: calculate the integral in equation (1) numerically. The integral has an integrable singularity that is easily removed by the variable transformation

$$z = \Theta^{1/2},$$

and the integrand is very well behaved, having a simple shape and decaying rapidly with increasing z. Using routines from Press *et al.* (1992), the integration is very rapid. Doing the integration numerically has another advantage: before, I devised an *ad hoc* scheme for switching from what I called the "exact" solution (the summation given by equation (21) in Boore, 1983, yielding the dashed line in Figure 1) to the asymptotic expansion of the integral. This scheme is discussed on p. 82 of Joyner and Boore (1988). Now there is no need to switch from one approximation of the integral to another— one simply computes

the integral numerically at all times. Speaking of the asymptotic expansion (which is commonly used in applications of random-vibration theory), the two-term approximation is shown by the light line in Figure 1; it is clearly inadequate at long periods.

One of the most important messages from the comparisons shown in Figure 1 is how well the *RV* method works, even for excitations much shorter than the oscillator period (the $\mathbf{M} = 4.0$ source, with a stress parameter of 200 bars, has a duration of 0.24 sec). Further comparisons are shown in Figures 7, 8, 9, and 10, referred to in the discussion of input parameters.

THE PROGRAMS

PROGRAM OVERVIEW:

The set of programs are collectively called **SMSIM** (**S**tochastic **M**odel **Sim**ulation or **S**trong **M**otion **Sim**ulation, take your pick). Separate programs are included for the *RV* and the *TD* simulations, but an effort has been made to make the input and output parameter files the same for both applications. The programs include application-specific drivers (*RV_DRVR* and *TD_DRVR*) that call modules of subroutines (*SMSIM_RV* and *SMSIM_TD*); these modules in turn call two additional modules of subroutines (*RVTDSUBS* and *RECIPES*). *RVTDSUBS* contains routines that are common to both applications, and *RECIPES* contains programs from Numerical Recipes (Press *et al.*, 1992). A few of the *RECIPES* subroutines are minor modifications of the routines in Press *et al.*; the modifications are noted in the annotated list of programs.

The drivers provided in this report produce peak acceleration, peak velocity, and response spectra for a range of oscillator periods, all for a given distance and magnitude. The modules were designed so that the drivers can be easily modified to produce the ground-motion parameters for other combinations of magnitude, distance, or input parameters. (For example, recently I needed a table of response spectral values at many magnitudes and distances for a set of oscillator periods—one file per period; it was easy to generate this by modifying *RV_DRVR*.)

Programs are also given to compute Fourier spectral amplitudes corresponding to the model specified by the input-parameter file (*FAS_DRVR*) and for computing a first-order approximation to site amplification given depth-dependent velocity and density (*SITE_AMP*).

The purpose of each program and subroutine is noted in the following annotated list. Following this description are some notes about compiling and modifying the programs and descriptions of parameter input and program output.

ANNOTATED LIST OF PROGRAMS:

A short description is given of the purpose of each program; for details, see the program listings in Appendices A through E. The user may find that some of these programs are useful in other applications.

Random Vibration Programs:

- *RV_DRVR*: The front-end program for the random-vibration calculations. Interactively obtains input and output file names, whether or not response spectra are to be computed, and information needed to set the damping and periods for the response spectra. The periods can be either individual periods or a set of periods between specified limits. The program obtains input parameters from a file, and passes these parameters to the subsequent subroutine modules through common blocks (dimension statements, variable declarations, and common statements are contained in *SMSIM.FI* and are inserted into *RV_DRVR* at compile time by the use of the Fortran INCLUDE statement.) The input parameters can, of course, be overridden in customizations of the driver program. This would occur if, for example, the motion is required for many values of the stress parameter rather than the one value included in the input parameter file. The program also obtains interactively the magnitude and distance for the simulation. The program computes peak velocity, peak acceleration, and, if specified, response-spectral amplitudes, with separate calls to the main subroutine (*SMSIM_RV*). After writing the results to an output file in columnar format, the program loops back for another magnitude and distance, if desired.

- *SMSIM_RV*: The main subroutine module for the random-vibration calculations, called separately for peak velocity, peak acceleration, and response-spectral output. The routine calls subroutines to set some frequency-independent spectral parameters and calls *GET_MOTION*, which does the actual simulations. The various subroutines included in the module are:

    - *GET_MOTION*: Computes the necessary spectral moments and uses these moments in the numerical integration that provides the simulated amplitudes. The routine also computes the values based on the one- and two-term asymptotic expansions, in case the user wants to compare them to the results from the direct

integration of equation (1). These estimates, *pk_cl_1* and *pk_cl_2* (standing for peak motion from Cartwright and Longuet-Higgins formulation using 1- and 2-term asymptotic expansions), are available through a common block included in *SMSIM.FI*.

- *CL68_NUMRCL_INT*: Calls routines to compute the integral in equation (1) (equation (6.8) of Cartwright and Longuet-Higgins, thus "*CL68*").

- *CL68_INTEGRAND*: A function defining the integrand in equation (1).

- *AMOM_RV*: A function that returns a spectral moment, computed by adaptive integration. Unlike the straightforward integration of equation (1), I recommend adaptive integration (whose step sizes vary according to the requirements of the integrand) for the spectral moments; the spectral moments can have spike-like integrands, particularly for lightly-damped oscillators. The problem of fixed-increment integration is particularly critical for long-period oscillators, for which care must be taken that the frequency increment is not too coarse to approximate adequately the spectral moment.

- *DERIVS*: Subroutine needed in the adaptive-integration routine *ODEINT* described in the *RECIPES* section.

Time-Domain Programs:

- *TD_DRVR*: The front-end program for the time-domain calculations. Interactively obtains input and output file names, whether or not response spectra are to be computed, and information needed to set the damping and periods for the response spectra. The periods can be either individual periods or a set of periods between specified limits. The program also asks if sample time series are to be saved in a file. The program obtains input parameters from a file. The program also obtains interactively the magnitude and distance for the simulation. The program determines peak velocity, peak acceleration, and, if specified, response-spectral amplitudes, with one call to the main subroutine (*SMSIM_TD*). This differs from *RV_DRVR*, which made separate calls to *SMSIM_RV* in order to obtain the three main types of ground motion (the reason for the difference is that I decided that most applications would require simulations for a number of oscillator periods, which are most efficiently computed by passing a simulated time series to a subroutine that computes response spectra). After writing the results to an output file, the program loops back for another magnitude and distance, if desired.

- *SMSIM_TD*: The main subroutine module for the time-domain calculations. For each realization, it calls a routine that returns the acceleration time series, and then computes peak acceleration, peak velocity, and response spectral amplitudes for this time series. After the loop over realizations, the program computes the arithmetic average (NOT the log average) of the motions. The various subroutines included in the module are:

  - *GET_ACC*: The main computations for computing a time series are contained in this routine, which passes the time series through its argument list. The amplitudes are scaled such that the average of the squared spectral amplitudes of the random number sample, before frequency-domain filtering, is unity. This scaling differs somewhat from that of G. M. Atkinson, which was used in Atkinson and Boore (1995). In her implementation, the scaling was in terms of the average spectral amplitude rather than the average of the squared spectral amplitude. The difference between scaling leads to a systematic difference in the results, such that the motions in the tables in the Appendix of Atkinson and Boore (1995) would be reduced by about a factor of 0.89 if my scaling is used.

  - *GET_VEL*: Returns a times series that is the integral of an input time series, after detrending the input.

  - *DCDT*: A routine written by C. S. Mueller that detrends a time series.

  - *MNMAX*: Returns the minimum and maximum of an array.

  - *MEAN*: Computes the mean of an array.

  - *AVGSQ_REALFT*: Returns the average of the squared spectral amplitudes computed by REALFT, not including the values at zero frequency and the Nyquist frequency.

  - *WIND_BOX*: Returns values of a window using a raised cosine-taper at each end.

  - *WIND_EXP*: Returns values of an exponential window (see Boore, 1983, for the equation).

  - *RD_CALC*: Computes the relative displacement of the response of an oscillator to a specified motion; see the program listing for authorship.

Fourier-Amplitude Programs:

- *FAS_DRVR*: The front-end program for the calculation of Fourier amplitude spectra. The program closely follows the other drivers in obtaining input and output file names. After writing the results to an output file, the program loops back for another magnitude and distance, if desired. The program will compute Fourier amplitude spectra of ground displacement, velocity, and acceleration; it will also compute the Fourier amplitude spectra of the response of up to 10 oscillators.

- *SMSIMFAS*: The main routine for computing the Fourier acceleration spectra.

Subroutine Modules:

Many of the subroutines are common to the *RV* , *TD* , and *FAS* programs, and they have been collected into two modules, as listed below. In addition, a file with declaration and common statements is used by all of the programs.

- *SMSIM.FI*: This is the file with the declaration, dimension, and common statements.

- *RVTDSUBS*: This includes the following routines:

  - *GET_PARAMS*: Reads the input parameters from a file and computes the upper limit of integration for *RV* calculations and the number of points for the FFT calculation in *TD* simulations.

  - *WRITE_PARAMS*: Writes the input parameters to a file.

  - *SPECT_AMP*: A frequency-dependent function that computes the Fourier spectral amplitudes.

  - *CONST_AM0_GSPRD*: Computes the frequency-independent part of the Fourier spectrum, including the geometrical spreading factor.

  - *GSPRD*: A function that computes the geometrical spreading factor.

  - *BUTTRLCF*: A function that returns the response of a bidirectional high-pass Butterworth filter.

  - *SPECT_SHAPE*: A frequency-dependent function that computes the displacement spectrum, normalized to unity at zero frequency. Several spectral shapes

are built-in, and the routine can be customized to include any arbitrary shape.

- *SPECT_SCALE*: Returns parameters that control the scaling of the spectrum with source size. These include seismic moment and corner frequencies. The scalings include single-corner frequency and the Joyner (1984) and Atkinson (1993) two-corner-frequency scalings. The routine can be customized to include other spectral scalings.

- *SITE_AMP_FACTOR*: A frequency-dependent function that computes the site amplification factor.

- *DIMIN*: A frequency-dependent function that returns the spectral diminution factors, including kappa, fmax, and whole path Q.

- *Q*: A frequency-dependent function that computes the whole-path Q.

- *HARMOSCF*: A frequency-dependent function that evaluates the amplitude response of a harmonic oscillator. It is used in the *RV* calculations of response spectra.

- *DURSOURCE*: A function that returns the source duration for an earthquake with specified corner frequencies.

- *DURPATH*: A function that computes the part of the duration that depends on distance rather than earthquake size.

- *SKIP*: A simple routine that skips over a specified number of lines while reading a file.

- *GET_DATE*: Returns the system date. The routine uses Lahey Fortran system calls, but the modifications needed by the Microsoft Fortran compiler are indicated in the source code.

- *GET_TIME*: Returns the system time. The routine uses Lahey Fortran system calls, but the modifications needed by the Microsoft Fortran compiler are indicated in the source code.

- *TIME_DIFF*: Computes the difference in time obtained with two calls to *GET_TIME*; it is not compiler-dependent.

- *RECIPES*: Except for a few minor modifications indicated in the annotated list, the routines in this module are taken directly from the second edition of *Numerical Recipes* (Press *et al.*, 1992). I was not able to obtain permission to distribute the source code, either via hard-copy or via diskette, without paying a license fee. The routines have been linked into *RV_DRVR.EXE* and *TD_DRVR.EXE*, and I do have permission to use them in this way. Users of anything but these executables must obtain their own *Numerical Recipes* routines, which is excellent advice in any case. The book and the diskette that can be purchased from Cambridge University Press (address: Order Department, 110 Midland Avenue, Port Chester, New York 10573; phone: 1-800-431-1580). The module includes the following routines; annotation is given only when a modification of the original routine has been used:
    - *QMIDPNT*: This is *Numerical Recipes* routine *QTRAP*, renamed and with the word "midpnt" substituted for "trapzd".
    - *MIDPNT*:
    - *LOCATE*:
    - *ODEINT*: The program must be modified by removing the word "rkqs" from the parameter list and from the external statement.
    - *RKQS*:
    - *RKCK*:
    - *GASDEV*:
    - *RAN1*:
    - *REALFT*:
    - *FOUR1*:

Site-Amplification Programs:

Included here are other programs that may be useful to the user.

- *SITE_AMP*: This program converts a velocity and density model into a frequency-dependent site amplification, using the square root of the ratio of seismic impedances

at the source and near the surface. The surface seismic impedance is based on the shear velocity and density averaged over a depth equivalent to a quarter of a wavelength (this is how frequency enters into the computation).

- *F4RATTLE*: This program reads a file made by *SITE_AMP* and writes a file in the proper format for use by *RATTLE*, C. Mueller's program for computing the response of a stack of layers to *SH* waves.

COMPILATION AND MODIFICATION:

I used the F77L-EM/32 Fortran 77 Version 5.20 compiler from Lahey Computer Systems for all but the site-amplification programs, for which I used the Microsoft Fortran compiler, Version 5.1. The Lahey compiler was used in order to handle the large arrays in the time-domain programs. The commands to compile and link the program are as follows:

```
f77l3 rv_drvr

f77l3 smsim_rv

386link rv_drvr smsim_rv -maxdata 0 -stub runb -exe rv_drvr.exe
```

The time-domain code is assembled by substituting "td" for "rv".

The `-stub` switch binds a run-time DOS-extender into the executable file, so that the Lahey programs are not needed to run the program. As a result of the extra code, the executable file is larger than if, for example, the Microsoft Fortran version 5.1 compiler had been used (I have assembled the random-vibration code using both; although a larger file size, the execution time is shorter with the Lahey compiler).

The compilers are fast enough that I have used the Fortran INCLUDE statement at the end of *SMSIM_RV.FOR* and *SMSIM_TD.FOR* to bring in the subroutine modules *RVTDSUBS.FOR* and *RECIPES.FOR* at compile time. It would be more efficient, of course, to produce a library module and link this module with the application-specific programs.

Modifications are easy to make in the routines. As indicated above, changes in source shape and source scaling can be included by modifying the appropriate routines in *RVTDSUBS.FOR*; allowance has been made for an input parameter to choose any added source shaping or scaling without changing the original meanings of the input parameter. A more likely modification would be to write new drivers to produce ground motion for

other combinations of magnitude, distance, stress parameter, or oscillator periods. In this case, only the drivers *RV_DRVR.FOR* and *TD_DRVR.FOR* need be changed.

The emphasis in the output is on various peak measures of ground shaking. The time-domain simulation program does have the option of storing an acceleration and velocity time series in a file, but modifications are required if the user needs to store a suite of time series. This can be done easily by modifying the "`if (isim .eq. nacc_save)`" loop in subprogram *SMSIM_TD*.

INPUT AND OUTPUT OF SMSIM AND FAS PROGRAMS:

A sample input file is given in Figure 2. The parameters in this file do not represent any particular model that I have used in applications to either western North America (e.g., Boore, 1983, 1986; Boore *et al.*, 1992) or eastern North America (e.g., Boore and Atkinson, 1987; Boore and Joyner, 1991; Atkinson and Boore, 1995). The parameters have been chosen to illustrate the input parameters needed by the programs. **The parameters are not to be used for a particular application.** I am reluctant to give input files with the parameters used in some of my papers because my ideas concerning the appropriate parameters are evolving. For the convenience of the reader, however, I have included the input-parameter files for the Atkinson and Boore (1995) model and my current coastal California model in Appendices F and G. (I use the term "coastal California" to reflect more accurately the source of the data used in determining the parameters than the commonly used phrases "western United States" or "western North America"; this parameter file must not be used for distances beyond 100 km.) To emphasize the evolving nature of the coastal California model, I have used the date of the latest modification of the file as part of the file name.

The input-parameter file is made up of lines of text and lines containing the input parameters. The lines of text are for the convenience of the user; the programs skip over them. It is very important that the number of text lines remain the same, however, for otherwise the program will attempt to read a text line as a parameter line. In other words, only change the parameter lines! In addition, list-directed input is used. This means that all parameters must be included, even if some are not used.

The program FAS_DRVR does not need the parameters in the last several lines of input file, but these lines should be included nevertheless.

Input From Screen:

The programs ask questions of the user regarding input file name, file name for

summary listing, and file name of file containing results in columnar form, suitable for import into graphics programs. In addition, the user is asked to provide information regarding whether or not response spectra are to be computed and the damping and the periods for which the spectra will be computed. The periods can be entered individually or can be computed by the program for a specified range and number of periods (logarithmically spaced). Alternatively, the spectra can be computed at the standard set of 91 periods used by the USGS and CSMIP in their routine processing of strong-motion data. In addition, the time-domain program asks if a sample of the acceleration and velocity time series should be saved, and if so, which sample. After the results are written to various output files, the program asks if computations are to be made for another distance and magnitude, and if so, asks for the name of the file to which the output will be written in columnar format.

The numbers entered in response to a program query need not contain decimal points. For example, a period of 2.0 secs can be entered as "2.0" or "2". In addition, if the user forgets to enter the parameter on one line, the program will expect the input on the next line.

Input From File:

1. *rho, beta, prtitn, radpat, fs:* These parameters are the density (in gm/cc) and shear-wave velocity (in km/s) in the vicinity of the source, the partition factor (to partition the S wave energy into two horizontal components, usually given by $1/\sqrt{2}$; it should be consistent with the next parameter), the radiation pattern, averaged over some portion of the focal sphere (this should refer to the radiation factor of the total S-wave radiation if the partition factor is taken to be $1/\sqrt{2}$; see Boore and Boatwright, 1984, for tables of values), and the free surface factor (usually equal to 2). Note that *rho, beta* were 2.7, 3.2 and 2.8, 3.8 in the applications of Boore (1983) and Atkinson and Boore (1995), respectively. I now suggest 2.8, 3.5 for WNA and 2.8, 3.6 for ENA.

2. *source number, pf, pd:* These parameters control the shape of the spectrum. As explained in the text lines in the input-parameter file, the *source number* specifies whether the spectral shape is a single-corner spectrum (*source number* = 1), one of two possible double-corner spectra (*source number* = 2 for Joyner (1984), as modified in Boore and Joyner (1991); *source number* = 3 for Atkinson (1993)), or a different spectral shape resulting from a modification of the program (*source number* = 4). The parameters *pf* and *pd* are used if *source number* = 1 and control the spectral shape given in the following equation:

$$S(f) = 1/(1 + (f/f_c)^{pf})^{pd}, \tag{2}$$

where f_c is the corner frequency. For the usual single-corner model, $pf = 2$ and $pd = 1$. A sharper corner, preferred by some, is given by $pf = 4$ and $pd = 0.5$. In all cases, an omega-square model requires that $pf \times pd = 2$.

3. *stressc, dlsdm, fbdfa, amagc:* These parameters control the scaling of the spectral amplitudes with source size, primarily by specifying the dependence of the corner frequencies on magnitude. The parameters are not used if *source number* = 3, but in this case dummy parameters must be included in the input-parameter file. The parameters *fbdfa* and *amagc* are used if *source number* = 2, in which case *fbdfa* is the corner frequency f_b divided by f_a and *amagc* is the critical moment magnitude beyond which the scaling is no longer self-similar. If *source number* = 1, then the stress parameter is given by:

$$\Delta\sigma = stressc \times 10^{dlsdm \times (\mathbf{M} - amagc)}. \tag{3}$$

The usual case of magnitude-independent stress is given by setting $dlsdm = 0.0$ (note that "*dlsdm*" stands for derivative of log sigma with respect to magnitude). *stressc* has units of bars.

4. *nsegs, (rlow(i), slope(i), i = 1, nsegs):* These parameters control the geometrical spreading, as represented by *nsegs* segments, each starting at *rlow* with a distance-dependence of r^{slope} beyond *rlow* (in all cases, specify $rlow(1) = 1.0$). In the sample input-parameter file, the geometrical spreading is r^{-1} from 1.0 (actually, any distance less than 70 km) to 70 km, r^0 from 70 to 130 km, and $r^{-0.5}$ beyond 130 km (this is the dependence used by Atkinson and Boore, 1995).

5. *fr1, Qr1, s1, ft1, ft2, fr2, Qr2, s2:* The whole-path attenuation is given by

$$\exp(-\pi f r / Q(f) \beta), \tag{4}$$

in which the function $Q(f)$ is described by the parameters in this entry of the input-parameter file. As shown in Figure 3, $Q(f)$ is given by a piecewise continuous set of three straight lines in $\log Q$ and $\log f$ space. The first and third lines have slopes of $s1$ and $s2$ and values of $Qr1$ and $Qr2$ at reference frequencies $fr1$ and $fr2$, respectively. The first and third lines apply for $f \leq ft1$ and $f \geq ft2$, respectively, with a straight line in $\log Q, \log f$ space connecting the values of Q at the transition frequencies $ft1$ and $ft2$ (in other words, $Q(f) = Qr1(f/fr1)^{s1}$ for $f \leq ft1$ and $Q(f) = Qr1(f/fr1)^{s1}$ for $f \geq ft2$, with a connecting line between these two). I decided on this representation after much experimentation; it is the simplest way of representing a complicated $Q(f)$ function with terms that are familiar to most users. The values in the sample input-parameter file have been chosen to represent closely the $Q(f)$ function given in Boore

(1984) and in my WNA applications. (As in all of the input parameters for specific applications, this function should be confirmed or modified based on special studies; in particular, intermediate- and long-period motions at large distances can be sensitive to the location of the low-frequency branch of the $Q(f)$ function, which is not well determined from data). Note that because the decision of which line segment to use depends solely on the transition frequencies, the relative size of the reference frequencies does not matter (i.e., $fr2$ could be less than $fr1$). Two special cases should be mentioned: $Q = Q_0$ (a constant) and $Q = Q_r(f/f_r)^s$. The constant-Q case is given by specifying $s1 = 0$, $s2 = 0$, $Qr1 = Q_0$, $Qr1 = Q_0$, and any non-zero values for $fr1$, $fr2$, $ft1$, and $ft2$. The special case of a single power-law dependence is specified by $Qr1 = Q_r$, $Qr2 = Q_r$, $fr1 = f_r$, and $fr2 = f_r$, and any non-zero values for $ft1$ and $ft2$. All frequencies should have units of Hz.

6. *w_fa, w_fb:* The source duration used in the calculations is given by

$$dursource = w\_fa/fa + w\_fb/fb, \tag{5}$$

where fa and fb are the source corner frequencies. For the single corner-frequency model (*source number* = 1), $fa = fb$, so any combination of weights w_fa and w_fb can be used, as long as they add up to the desired weight. In my WNA applications, I used $w\_fa = 1.0$ and $w\_fb = 0.0$. In Atkinson and Boore (1995), $w\_fa = 0.5$ and $w\_fb = 0.0$.

7. *nknots, (rdur(i), dur(i), i = 1, nknots), slope of last segment:* These parameters are used in the specification of the path duration by a series of straight-line segments with parameters *nknots, rdur, dur, slope of last segment.* where *nknots* is the number of intersections between line segments. The meaning of these parameters is indicated in Figure 4. The values given in the figure correspond to those in the input-parameter file, which in turn were chosen to represent the duration used by Atkinson and Boore (1995). For my WNA applications I used one segment with a slope of 0.05 (i.e., $nknots = 1$, $rdur = 0.0$, $dur = 0.0$, $slope = 0.05$), but this was assumed without any special studies as was done for the Atkinson and Boore (1995) application. Atkinson (1995) finds a very different relation for western Canada; a comparable study should be done for other regions.

8. *namps, (famp(i), amp(i), i = 1, namps):* The site amplification is approximated by a series of straight-line segments in log amplification, log frequency space, connecting the values *famp, amp.* The amplification for $f \leq famp(1)$ and $f \geq famp(namps)$ is given by $amp(1)$ and $amp(namps)$, respectively. This is shown in Figure 5 (which uses the parameters in the input-parameter file). **The numbers in the data file were**

invented for the sake of illustration. Suggested values for WNA, based on my recent work (Boore and Joyner, 1996), are given in Appendix G. For no amplification, set $namps = 1$, $amp = 1.0$, and $famp$ equal to any number.

9. *fm, kappa:* The diminution function is controlled by the parameters fm and *kappa*. They are used in the following filter:

$$\exp(-\pi \times kappa \times f)/\sqrt{1+(f/fm)^8}. \tag{6}$$

In many applications only one or the other of the two parameters are desired; this is easy to implement with appropriate choices of the parameters. For example, to use only fm, specify $kappa = 0.0$; to use only *kappa*, specify a large number for fm. The units of *fm, kappa* should be consistent with that of *f*: Hz and 1/Hz.

10. *fcut, norder:* In some cases it may be desired to include a low-cut filter in the simulations. This might be the case, for example, for simulations of processed strong-motion data. The parameters $fcut$ and $norder$ control the low-cut filter and are used in *BUTTRLCF*. The filter is given by the following function:

$$1.0/(1.0+(fcut/f)^{2.0\times norder}) \tag{7}$$

(this is the response of a bidirectional filter made up of two Butterworth filters, each of order $norder$). Set $fcut = 0.0$ for no filter.

11. *zup, eps_int, amp_cutoff:* The first parameter specifies the upper limit in the integral of equation (1). I have found that 5 seems to be a good number for this limit, and there is probably no reason to change the value from that in the sample file ($zup = 10$). The second parameter specifies the error in the adaptive integration routine *ODEINT*, and the third is used as the basis for computing the upper frequency limit (fup) used in the random-vibration calculations (this is computed in subroutine *GET_PARAMS* and also in *RV_DRVR*). fup is determined such that the exponential in the equation above has a value of *amp_cutoff*. I have found the values in the sample input-parameter file to give good results, but the user should experiment to make sure that they are appropriate values. I have included these three parameters for generality, even though I do not anticipate that they will be changed.

12. *indxwind, taper, twdtmotion, eps_wind, eta_wind:* These parameters control the shape of the window applied to the random number time series in the time-domain simulations. Either a box ($indxwind = 0$) or an exponential window ($indxwind = 1$) can be used (see Boore (1983) for more discussion of the latter). *taper* is used only

in the box window; it is the fraction of the duration of motion for which a raised-cosine taper will be applied to the front and to the back of the box window (i.e., , the extent of the taper will be $taper \times (dursource + durpath)$ in both the front and the back of the window). *taper* is not used for the exponential window. The parameters *twdtmotion*, *eps_wind*, and *eta_wind* are only used for the exponential window. The meaning of the parameters is best seen by referring to Figure 6. Sample output for the box and exponential windows is given in Figures 7 through 10 for $\mathbf{M} = 4$ and 7, and $R = 10$ and 200 km. The results in the figures indicate that, in general, the response spectral amplitudes from the exponential window are closer to the random vibration results than are those from the box window. This is not surprising, because the correction factor for oscillator response proposed by Boore and Joyner (1984) and used in the *SMSIM* programs was derived empirically from calculations made using the exponential window.

13. *tsimdur, dt, tshift, seed, nruns:* These parameters deal with time-domain details. *tsimdur* is a minimum duration for which the motion will be computed and *dt* is the time spacing. The program chooses the number of time points *npts* to be a power of 2 such that the duration of the time series equals or exceeds *tsimdur* (the arrays are dimensioned such that *npts* must be less than or equal to 16,384; it is up to the user to make sure that this condition is satisfied). The program will display an error message and stop if the calculated duration (including contributions from the initial delay, the source, the path, and the window) exceeds the duration $npts \times dt$. The parameter $tshift$ produces a time shift in the start of time series; this can be useful to accommodate pre-arrival tails due to the noncausal filters used in the analysis. *seed* and *nruns* are the initial seed of the random-number generator and the number of time-domain simulations, respectively. If peak motions are desired, *nruns* should be large enough to reduce the uncertainty in the computed means of the peak motions determined from each realization. Examples of results computed for several *nruns* are given in Figures 11 and 12. In general, the uncertainty in the mean decreases as $1/\sqrt{nruns}$. Note in these figures that the time-domain and random-vibration results show some systematic disagreements for the larger earthquake. This difference is a maximum of a factor of about 1.12, and is probably related to the assumption in the random-vibration theory that the amplitudes of successive peaks are independent of one another. This is certainly not true for a long-period oscillator response. Corrections schemes for "clumping" might yield a better comparison. I tried one such scheme (due to Toro, 1985), but the comparison was not improved. In view of the aleatory uncertainty in ground-motion data and the epistemic uncertainty in the input parameters, I am willing to live with uncertainties that in general are less than 10 percent in order to take advantage of the greatly increased speed of the

random-vibration calculations compared to the time-domain calculations.

14. *remove dc from random series?:* This parameter was included in the development of the program. If not equal to 0.0, then the mean of the random number sample will be removed before windowing and transforming into the frequency domain. I suggest that the parameter be set to 0.0.

Output of SMSIM and FAS Programs:

Two or three files are produced by the *SMSIM* program, the first containing a summary of the input and the results, and the second a file with columns containing the oscillator period and frequency and response spectra amplitudes (both pseudo relative velocity, PRV, and pseudo absolute acceleration, PAA; note that the "RV" in PRV should not be confused with the use of "RV" to stand for "Random Vibration"). This latter file is in a convenient format to be imported into the graphics program that I use (CoPlot, published by CoHort Software, 1-800-728-9878). A third file is produced if it is desired to save a sample acceleration and velocity time series computed by the time-domain simulation. Sample output files are given in Figures 13, 14, and 15; a sample of the time series is given in Figure 16. The output files from *RV_DRVR* includes estimates of dominant frequency, as measured from the frequency of zero crossings (equation 27 in Boore, 1983). In addition, the output includes the parameter *eps* that measures the bandwidth of the motion ($eps = \sqrt{1-\xi^2}$, where ξ is given by equation (22) in Boore, 1983; see Cartwright and Longuet-Higgins, 1956, p. 216–217, for more discussion). Finally, the random-vibration output also includes estimates of the number of extrema (nx) and the number of zero crossings (nz). The bandwidth parameter ξ and nx and nz are related by $\xi = nz/nx$.

The response of a single-degree-of-freedom oscillator with gain of V and specified natural period (T_o) and damping (η) can be obtained by multiplying the PRV output for the specified natural period and damping by the factor $VT_0/2\pi$. This scheme can be used to simulate the response of a Wood-Anderson instrument and thereby to obtain estimates of local magnitude M_L corresponding to the ground motion. According to Uhrhammer and Collins (1990), for a Wood-Anderson instrument $V = 2080$, $T_0 = 0.8$s, and $\eta = 0.69$. Time series corresponding to the oscillator output are not returned by the programs, although they can be produced by a simple modification to the subprogram *RD_CALC* in the module *SMSIM_TD*; the modification is indicated by a comment in *RD_CALC*.

The *FAS* program creates a summary file and a file with columns of frequency, period, and Fourier spectral amplitude for the ground displacement, velocity, acceleration, and oscillator response. The output is similar to that in Figures 13 and 14.

With the units as given in the discussion of input, the output ground motion will be in cgs units.

INPUT AND OUTPUT OF SITE-AMPLIFICATION PROGRAMS:

As in the previous programs, input comes from the screen and from a parameter file.

Input From Screen:

SITE_AMP: The program first asks the user for the names of the input and output files. It then asks if the default coefficients of a linear relation between density and velocity will be used, in which case the program asks for the values specifying the end points of the line (the densities are given by the end-point values for velocities outside of the specified range). As explained below, the program looks at the parameter file and queries the user about whether the velocities are a piecewise continuous function of depth or represent a layered model. Finally, the program asks for the velocity and the density in the source region.

I have found that it is very convenient to produce the input-parameter file using a spreadsheet and printing the necessary columns to a file. With a spreadsheet it is easy to subdivide a thick constant-velocity layer into pseudo-layers. This may be necessary to obtain amplifications at closely-spaced frequencies, since the program computes frequencies only for depths included in the input file (a future version will interpolate between input depth points to produce amplification at a specified set of frequencies). Using a spreadsheet also makes it easy to include velocity models of varying complexity (I often use power laws fit to two end points, as well as linear velocity gradients).

Regarding units: velocity and depth units can by anything as long as they are consistent with one another (i.e., depth in meters should be matched with velocity in meters per second). Note that if the densities are to be computed from the default end points, the velocity units must be kilometers per second. The density units do not have to be consistent with those of depth and velocity. I strongly advise that the density units be given in grams per cubic centimeters.

F4RATTLE: The program asks for the name of input and output files (with a default of *Rattle.In* for the output file, which is what the program *RATTLE* expects). The program then asks for a series of input parameters:

1. *zdepth*: The depth at which the response is to be computed (usually 0.0 for the ground

surface).

2. *theta*: The angle of incidence, in degrees, of the incident wave (0.0 for vertical).

3. *Q*: The program asks for the attenuation parameter Q to be assigned to the layers. It assumes the same value for all layers.

4. *sps*, *mx*: The final two parameters control the frequency spacing through the equation

$$delf = sps/2^{mx}.$$

They are used rather than a simple specification of the frequency spacing because the original intent of the program *RATTLE* was to provide a table of response values that could be used with a Fourier transform to yield a time series of the response (in this context, *sps* is the number of samples per second in the time series and the number of samples is 2^{mx}). The number of frequencies for which the response is calculated is given by

$$nfreq = 2^{mx-1} + 1.$$

The program *RATTLE* is currently being modified by C. Mueller, and as a result, it may be necessary to modify *F4RATTLE* to allow a different specification of the frequency spacing and number of frequency points.

Input From File:

SITE_AMP: A sample input file is given in Figure 17. The parameters in this file have been made up to illustrate the input; **they do not represent a real application**.

1. The first column is labeled "Depth", but it could also contain layer thickness. The units can be anything. If the first entry is "0.0", the program assumes that the entries will be depths and that the velocities and densities are the values for the specified depth; it asks the user to confirm this. If the first column is depth, then the program simply assumes straight-line connections between the entries. In this way a mix of linearly increasing, constant, and step changes can be included in the model. A constant parameter (velocity or density) over a depth range is entered by the depths that bound the layer, but with the same velocity or density for the two consecutive entries (e.g., as between depths 0.040 - 0.100 and 0.300–8.000 for the velocity parameter in the sample input file, with intermediate layers for better frequency resolution); similarly, a layered model can be included easily by entering the velocities and densities on either side of the interface, but with the same depth

for the two consecutive entries (e.g., at depths of 0.040 and 0.300 in the sample input file).

2. The second column contains the velocities, in any units that match the depth units.

3. The third column contains the densities. An entry of "0.0" will flag the program to use the *DENSITY* function. To compute a density from the velocity, using the relation specified interactively or the default relation built into the program (with end points of 2.5 gm/cm^3 at 0.3 km/s and 2.8 gm/cm^3 at 3.5 km/s, values that I have assumed for generic rock in WNA (see Boore and Joyner, 1996)). A nonzero entry will override the value that would have been computed by the *DENSITY* function (as has been done for a few depths in the sample input file, although it should be noted that the specified values for density of 3.0 and 4.0 gm/cc are very unrealistic).

F4RATTLE: This program uses a file made by *SITE_AMP* as input.

Output of Site-Amplification Programs:

SITE_AMP: One output file is created with columns containing the input model, the average velocities and densities, and the frequency and amplification for each input depth. A sample is given in Figure 18. The first three columns repeat the input parameters. Column 4 contains the depths at which the velocities have been specified, without the repeated depths needed for layers of constant velocity (if the input depth column corresponds to depth rather than thickness); this is why $nout \neq ndepths$. Column 5 is the travel time to the indicated depth; it is the basis for the rest of the results. Columns 6 through 8 contain a constant-velocity approximation to the input velocity model (it is constructed to give the same travel times for each depth as the continuous model) and are included as a convenience in case the approximate amplifications are to be checked using a wave-propagation program that requires a stack of constant-velocity layers. Columns 9 and 10 are the velocities and densities averaged from the surface to the specified depth. Columns 11 and 12 are the computed frequency and amplification. Note that for identification purposes, the stem of the input file name has been used to label columns 11 and 12.

One way in which I use the program *SITE_AMP* is to import the output file into my graphics program and plot the amplification vs. frequency (using log–log axes). I then pick off a set of amplifications and frequencies that will be used as input to the *SMSIM* programs.

F4RATTLE: The output is a file in the format required by *RATTLE*.

ACKNOWLEDGMENTS

I thank Bill Joyner for advice and encouragement over the years and Bob Herrmann for cross-checking of output from our programs and for reminding me that Cartwright and Longuet-Higgins' equation (6.8) is an approximation to their equation (6.4). In addition, I am grateful to Gail Atkinson for her version of my original time-domain simulation program, to Walt Silva for helpful discussions, and to Basil Margaris for provoking me into writing this report. Bill and Basil reviewed the manuscript. I also thank Stavros Anagnostopoulos and Jose Roesset for permission to use their program for computing response spectra and Chuck Mueller for permission to distribute his programs.

This work was partially supported by the Nuclear Regulatory Commission.

REFERENCES

Atkinson, G. M. (1993). Earthquake source spectra in eastern North America, *Bull. Seism. Soc. Am.* **83**, 1778–1798.

Atkinson, G.M. (1995). Attenuation and source parameters of earthquakes in the Cascadia region, **85**, 1327–1342.

Atkinson, G. M. and D. M. Boore (1995). Ground motion relations for eastern North America, *Bull. Seism. Soc. Am.* **85**, 17–30.

Boore, D. M. (1983). Stochastic simulation of high-frequency ground motions based on seismological models of the radiated spectra, *Bull. Seism. Soc. Am.* **73**, 1865–1894.

Boore, D. M. (1984). Use of seismoscope records to determine M_L and peak velocities, *Bull. Seism. Soc. Am.* **74**, 315–324.

Boore, D. M. (1986). Short-period P- and S-wave radiation from large earthquakes: implications for spectral scaling relations, *Bull. Seism. Soc. Am.* **76**, 43–64.

Boore, D. M. and G. M. Atkinson (1987). Stochastic prediction of ground motion and spectral response parameters at hard-rock sites in eastern North America, *Bull. Seism. Soc. Am.* **77**, 440–467.

Boore, D. M. and J. Boatwright (1984). Average body-wave radiation coefficients, *Bull. Seism. Soc. Am.* **74**, 1615–1621.

Boore, D. M. and W. B. Joyner (1984). A note on the use of random vibration theory to predict peak amplitudes of transient signals, *Bull. Seism. Soc. Am.* **74**, 2035–2039.

Boore, D. M. and W. B. Joyner (1991). Estimation of ground motion at deep-soil sites in eastern North America, *Bull. Seism. Soc. Am.* **81**, 2167–2185.

Boore, D.M. and W.B. Joyner (1996). Site-amplifications for generic rock sites, *Bull. Seism. Soc. Am.* **86**, (submitted)

Boore, D. M., W. B. Joyner, and L. Wennerberg (1992). Fitting the Stochastic ω^{-2} Source Model to Observed Response Spectra in Western North America: Trade-offs Between $\Delta\sigma$ and κ, Bulletin of the Seismological Society of America, Vol. 82, p. 1956-1963.

Cartwright, D. E. and M. S. Longuet-Higgins (1956). The statistical distribution of the maxima of a random function, *Proc. R. Soc. London* **237**, 212–232.

Hanks, T. C. and R. K. McGuire (1981). The character of high-frequency strong ground motion, *Bull. Seism. Soc. Am.* **71**, 2071–2095.

Herrmann, R.B. (1996). *Computer Programs in Seismology*, Dept. of Earth and Atmospheric Sciences, St. Louis University, St. Louis, Missouri.

Joyner, W. B. (1984). A scaling law for the spectra of large earthquakes, *Bull. Seism. Soc. Am.* **74**, 1167–1188.

Joyner, W. B. and D. M. Boore (1988). Measurement, characterization, and prediction of strong ground motion, in *Earthquake Engineering and Soil Dynamics II, Proc. Am. Soc. Civil Eng. Geotech. Eng. Div. Specialty Conf.*, June 27–30, 1988, Park City, Utah, 43–102.

Press, W.H., S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery (1992). *Numerical Recipes in FORTRAN: The Art of Scientific Computing*, Cambridge University Press, Cambridge, England, 963 pp.

Silva, W.J. and Lee, K. (1987). WES RASCAL code for synthesizing earthquake ground motions, *State-of-the-Art for Assessing Earthquake Hazards in the United States, Report 24*, U.S. Army Engineers Waterways Experiment Station, *Misc. Paper S-73-1.*

Toro, G.R. (1985). Stochastic model estimates of strong ground motion, Section 3 of *Seismic Hazard Methodology for Nuclear Facilities in the Eastern United States,*

Report Prepared for EPRI, Project Number P101-29.

Uhrhammer, R.A. and E.R. Collins (1990). Synthesis of Wood-Anderson seismograms from broadband digital records, *Bull. Seism. Soc. Am.* **80**, 702–716.

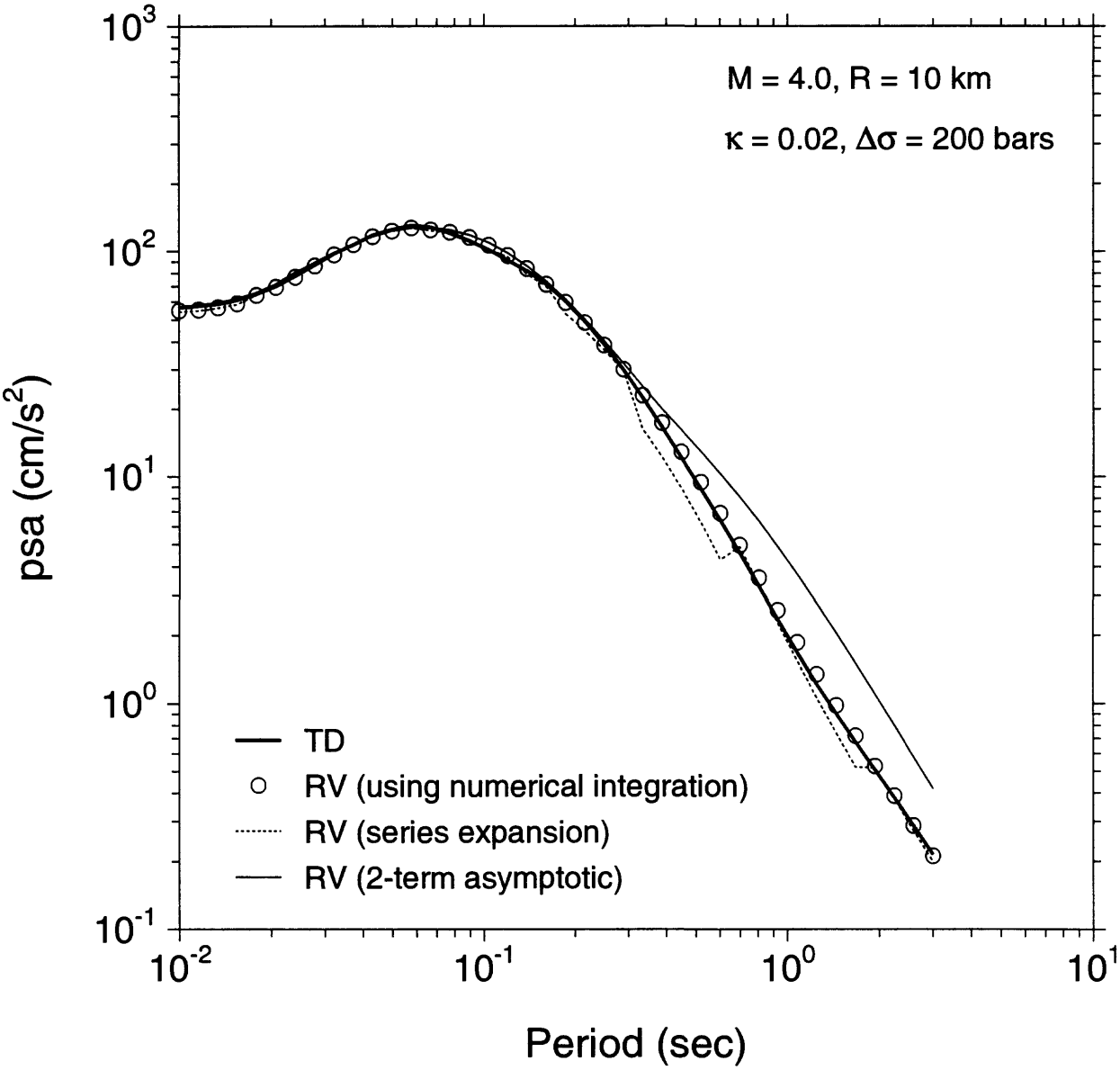

Figure 1. Response spectra computed with time-domain simulations and random-vibration simulations with various relations between the peak and rms values. The series expansion uses equation (21) in Boore (1983), and for the model parameters in this figure produces response spectra that are a discontinuous function of period. The function does not appear discontinuous in the plot because the period values are spaced too widely. The 2-term asymptotic expansion uses equation (24) in Boore (1983), and the random-vibration results computed with numerical integration use equation (1) in this report.

```
 Sample data file **** NOT FOR A PARTICULAR APPLICATION **
rho, beta, prtitn, radpat, fs:
    2.8 3.6 0.71 0.55 2.0
spectral shape: source number (1=Single Corner;2=Joyner;3=A93;4=custom),
                pf, pd (1-corner spectrum = 1/(1+(f/fc)**pf)**pd; 0.0 otherwise)
                (usual model: pf=2.0,pd=1.0; Butterworth: pf=4.0,pd=0.5)
                (Note: power of high freq decay --> pf*pd)
    1 2.0 1.0
spectral scaling: stressc, dlsdm, fbdfa, amagc
                (stress=stressc*10.0**(dlsdm*(amag-amagc))
                (fbdfa, amagc for Joyner model, usually 4.0, 7.0)
                (not used for source 3, but placeholders still needed)
    80.0 0.0 4.0 7.0
gsprd: nsegs, (rlow(i), slope(i))  (Set rlow(1) = 1.0)
    3
      1.0 -1.0
     70.0  0.0
    130.0 -0.5
q: fr1, Qr1, s1, ft1, ft2, fr2, qr2, s2
    0.1 275 -2.0 0.2 0.6 1.0 88.0 0.9
source duration: weights of 1/fa, 1/fb
    1.0 0.0
path duration: nknots, (rdur(i), dur(i)), slope of last segment
    4
    0.0 0.0
   10.0 0.0
   70.0 9.6
  130.0 7.8
    0.04
site amplification: namps, (famp(i), amp(i))
    5
    0.1   1.0
    1.0   1.5
    2.0   2.0
    5.0   2.5
    10.0  3.0
site diminution parameters: fm, akappa
    25.0 0.03
low-cut filter parameters: fcut, norder
    0.0 2
rv integration params: zup, eps_int (integration accuracy), amp_cutoff (for fup)
    10.0 0.00001  0.001
window params: indxwind(0=box,1=exp), taper(<1), twdtmotion, eps_wind, eta_wind
    1 0.05 1.0 0.2 0.05
timing stuff: tsimdur, dt, tshift, seed, nruns
    50.0  0.005 7.0 640.0 640
remove dc from random series before transforming to freq. domain (0=no;1=yes)?
    0
```

Figure 2. Sample input file for the SMSIM programs. **The file has been constructed for illustrative purposes and does not correspond to a real application.**

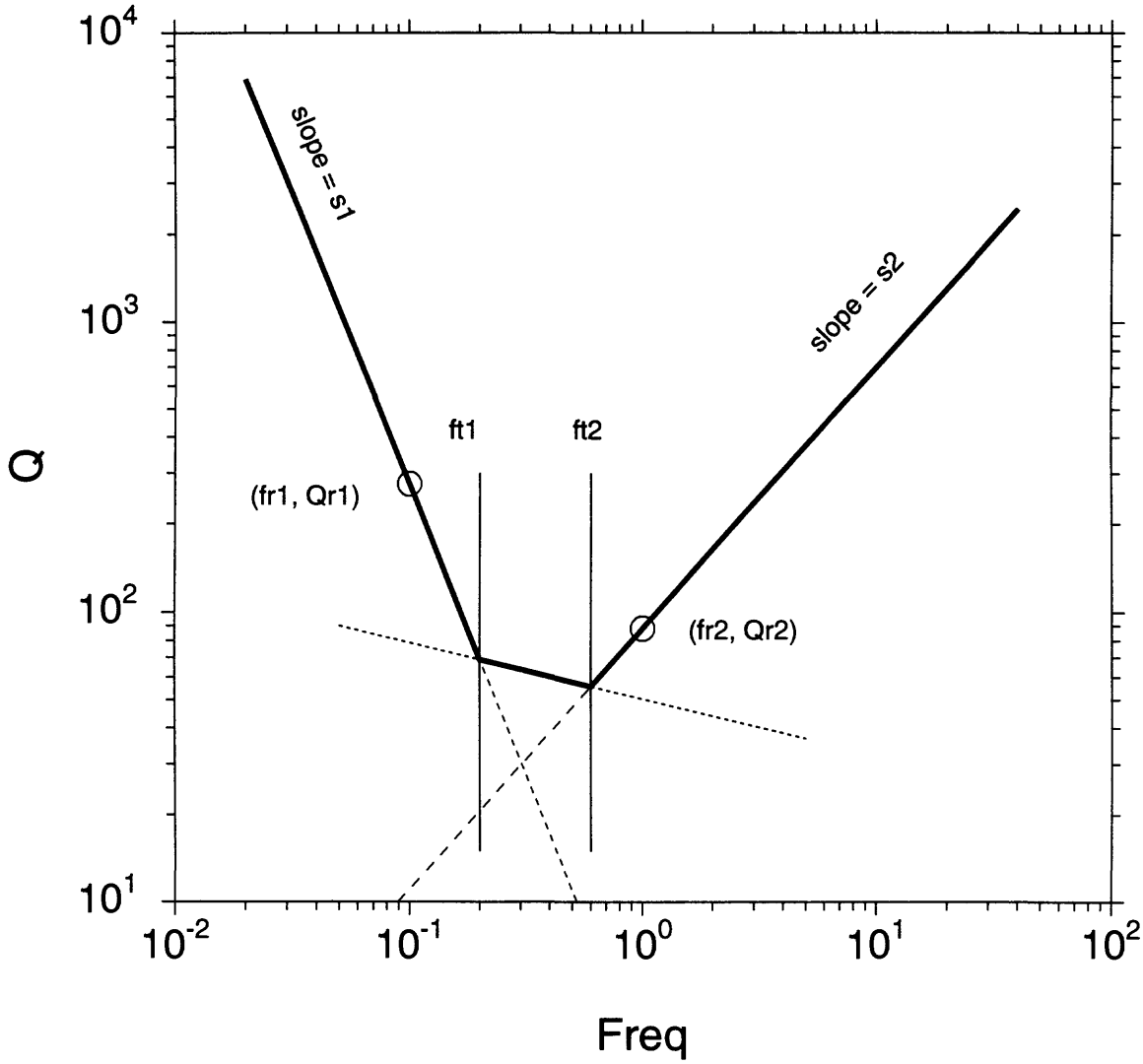

Figure 3. Illustration of the specification of $Q(f)$: it is made up of three lines in log–log space. The lines shown are those for the parameters in the sample input file, which is an approximation of the $Q(f)$ function in Boore (1984).

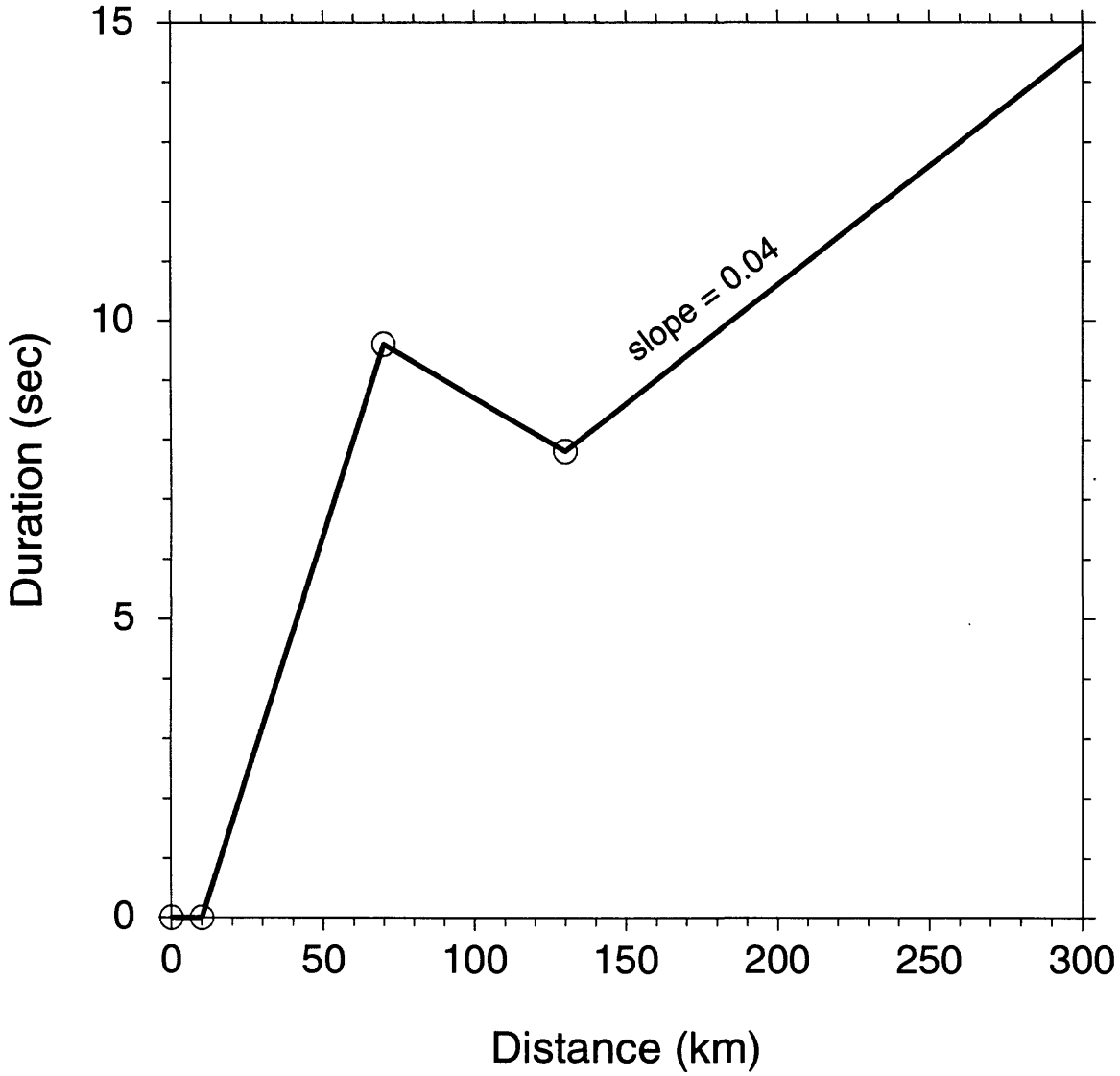

Figure 4. The duration due to the path is made up of a series of straight lines specified by distance–duration pairs (circles) and the slope of the last line. The lines shown are those for the parameters in the sample input file, and correspond to the duration in Atkinson and Boore (1995).

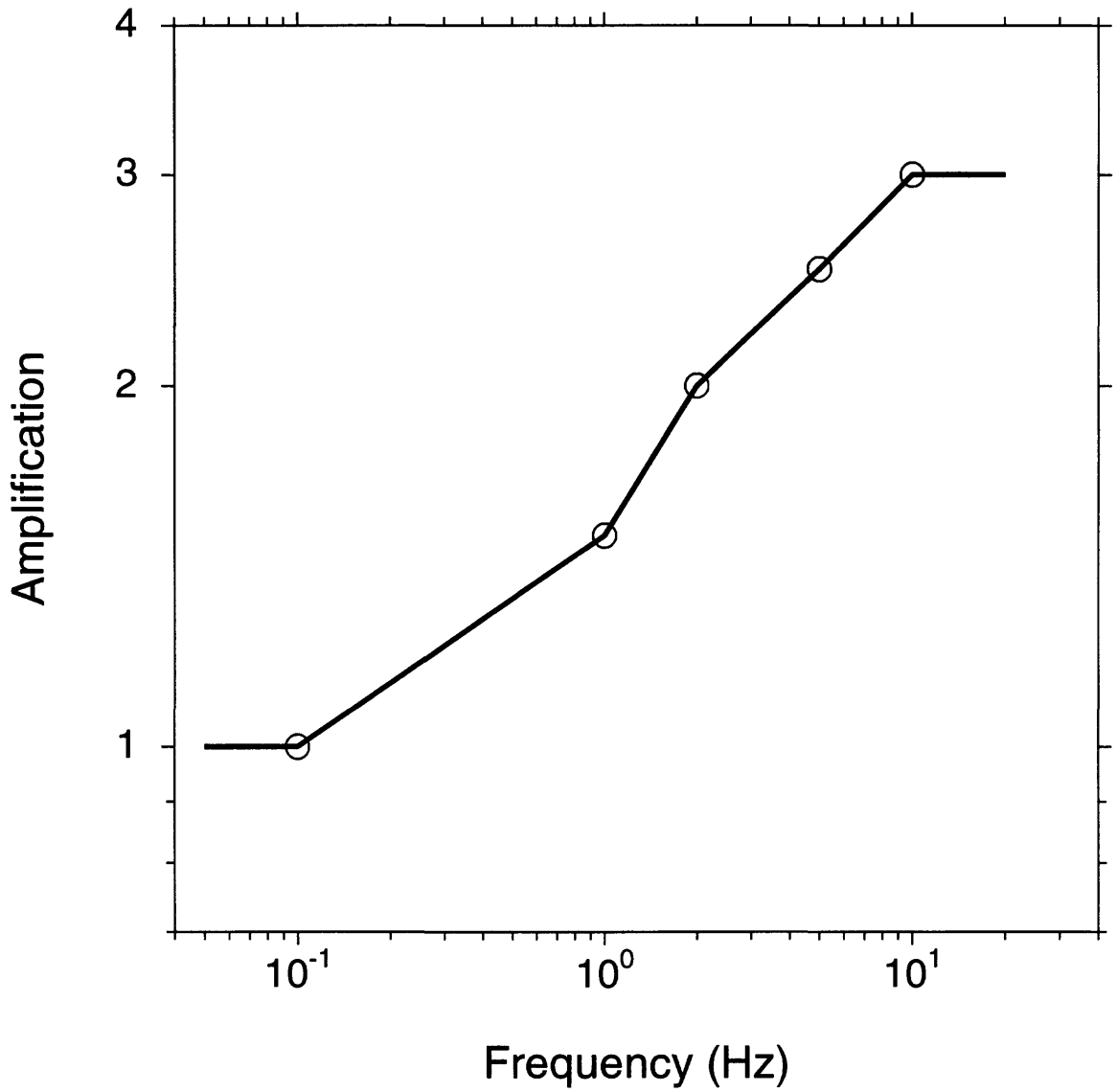

Figure 5. The site-amplification is specified by a series of straight lines in log frequency, log amplification space. The lines shown are those for the parameters in the sample input file, and are made up; **they do not correspond to any of my published applications.**

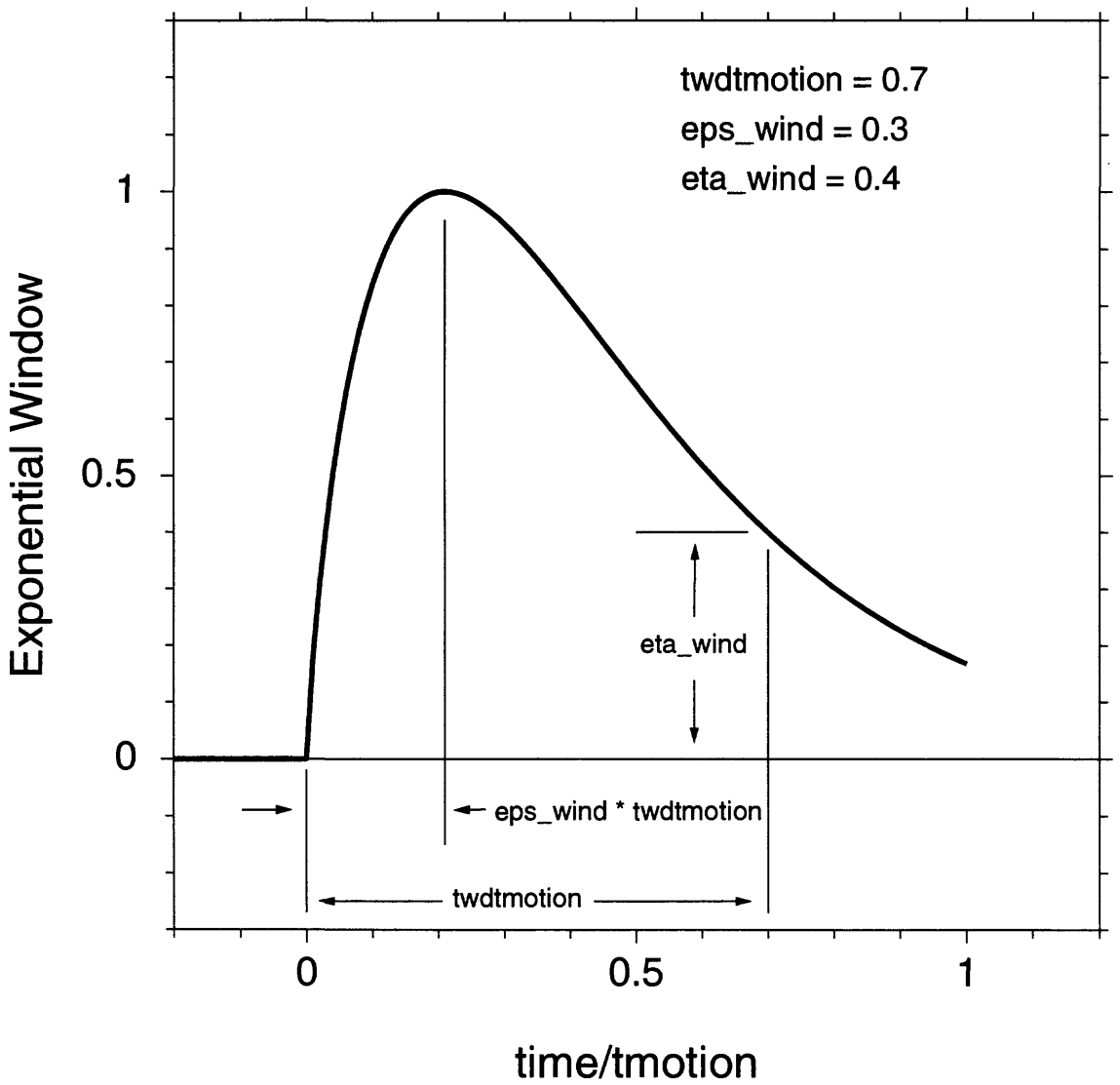

Figure 6. The exponential window is specified by parameters whose meaning is shown here. Note that the abscissa has been normalized by a quantity proportional to the duration of the motion (*tmotion*). (In the program, *tmotion* is given by doubling the source plus path durations). The parameters have been chosen to illustrate their meaning and are not those in the input file.

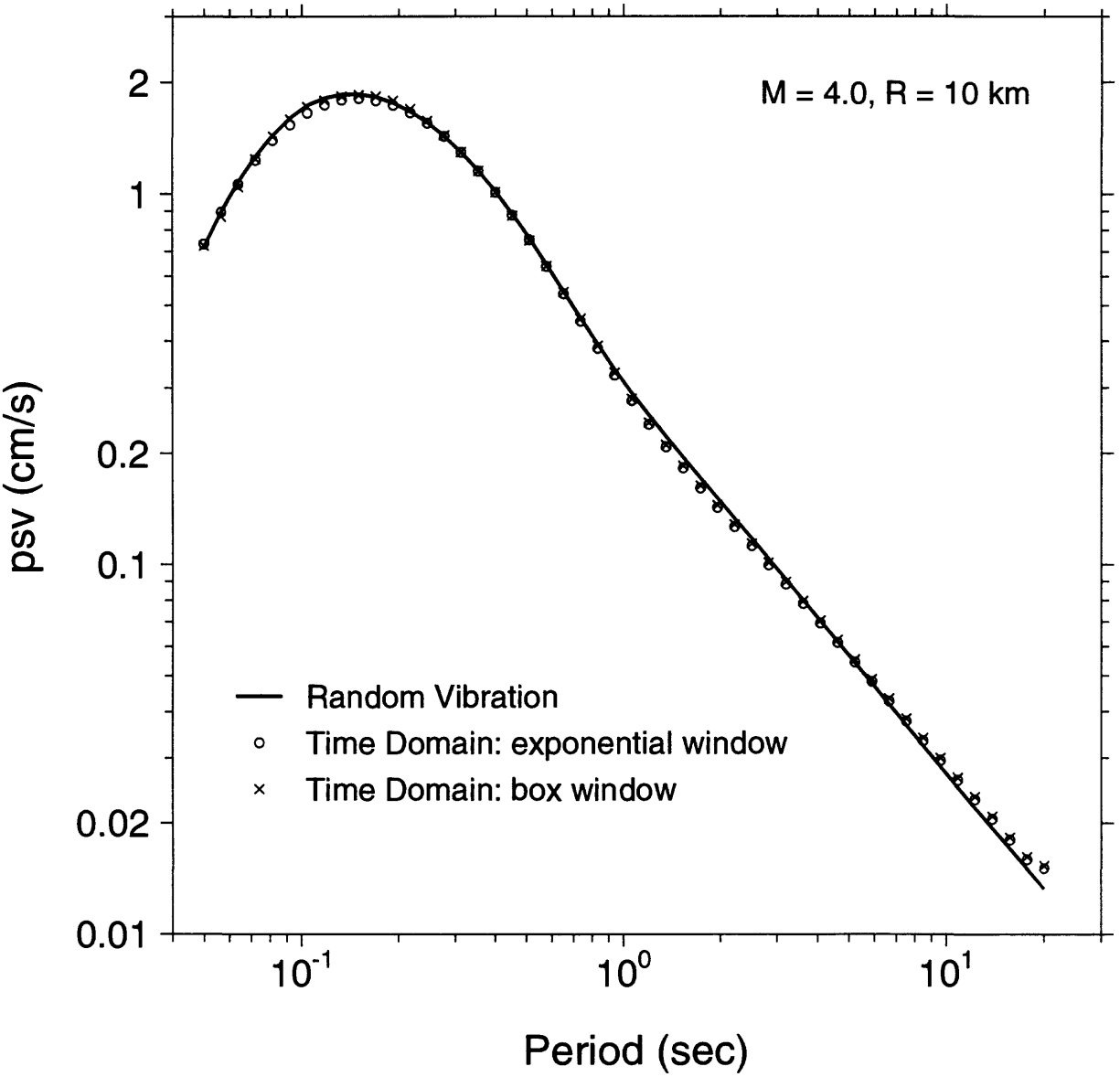

Figure 7. Comparison of simulations using box and exponential windows, with the random-vibration calculations for magnitude 4 at 10 km, using the parameters in the input-parameter file (except for the time-domain simulations, where $indxwind = 0$ for the box window).

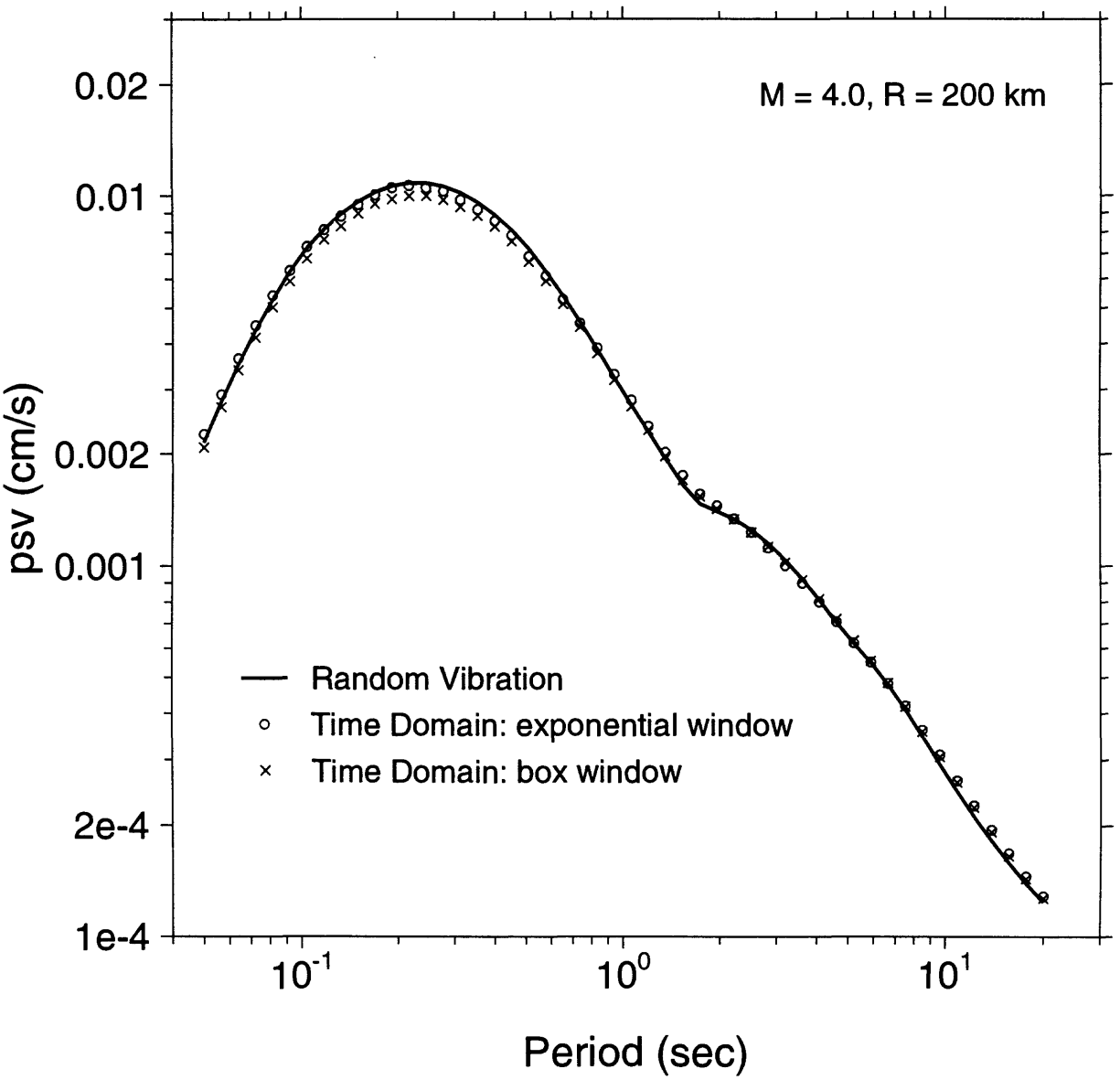

Figure 8. Comparison of simulations using box and exponential windows, with the random-vibration calculations for magnitude 4 at 200 km, using the parameters in the input-parameter file (except for the time-domain simulations, where *indxwind* = 0 for the box window).

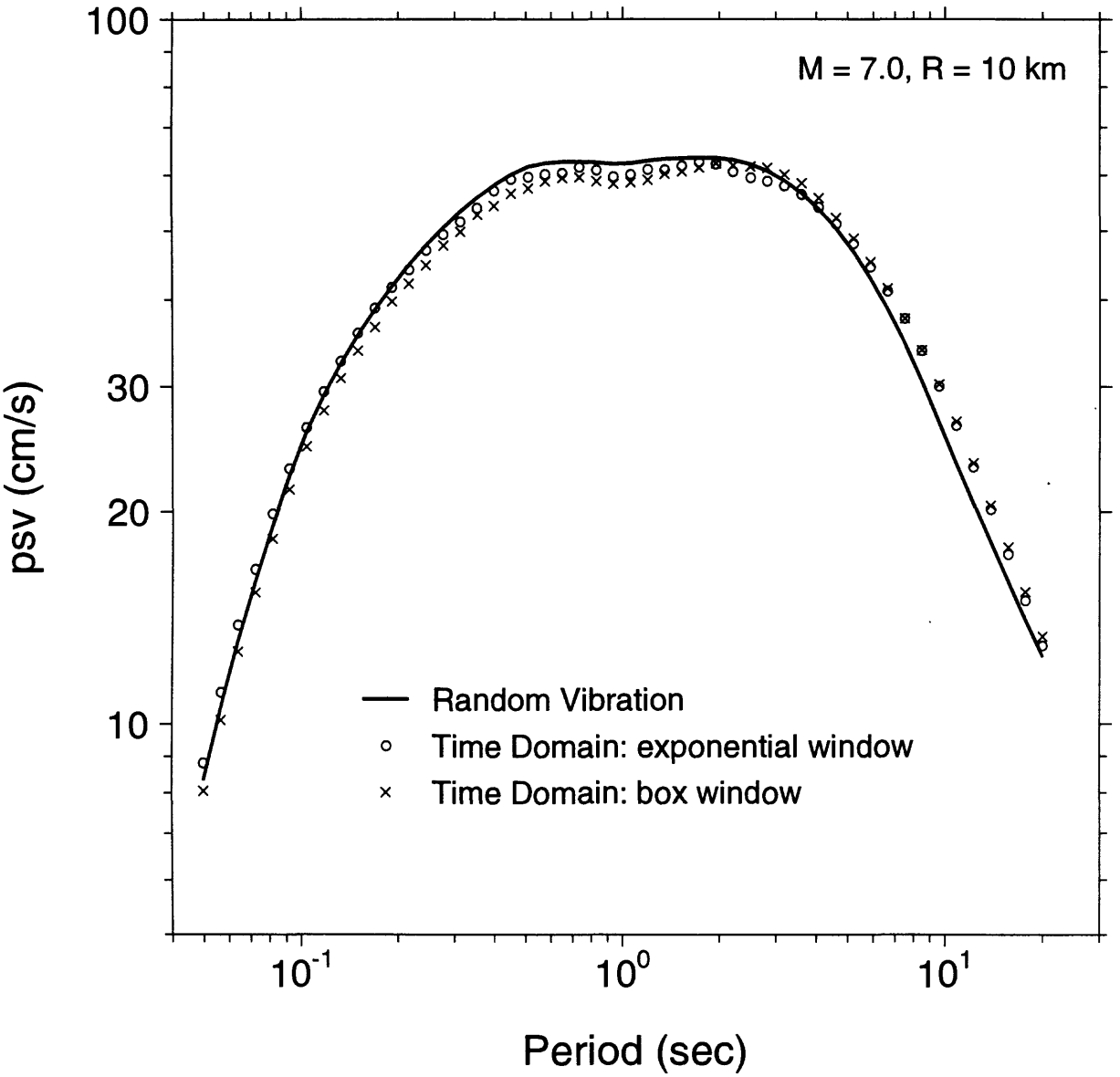

Figure 9. Comparison of simulations using box and exponential windows, with the random-vibration calculations for magnitude 7 at 10 km, using the parameters in the input-parameter file (except for the time-domain simulations, where $indxwind = 0$ for the box window).

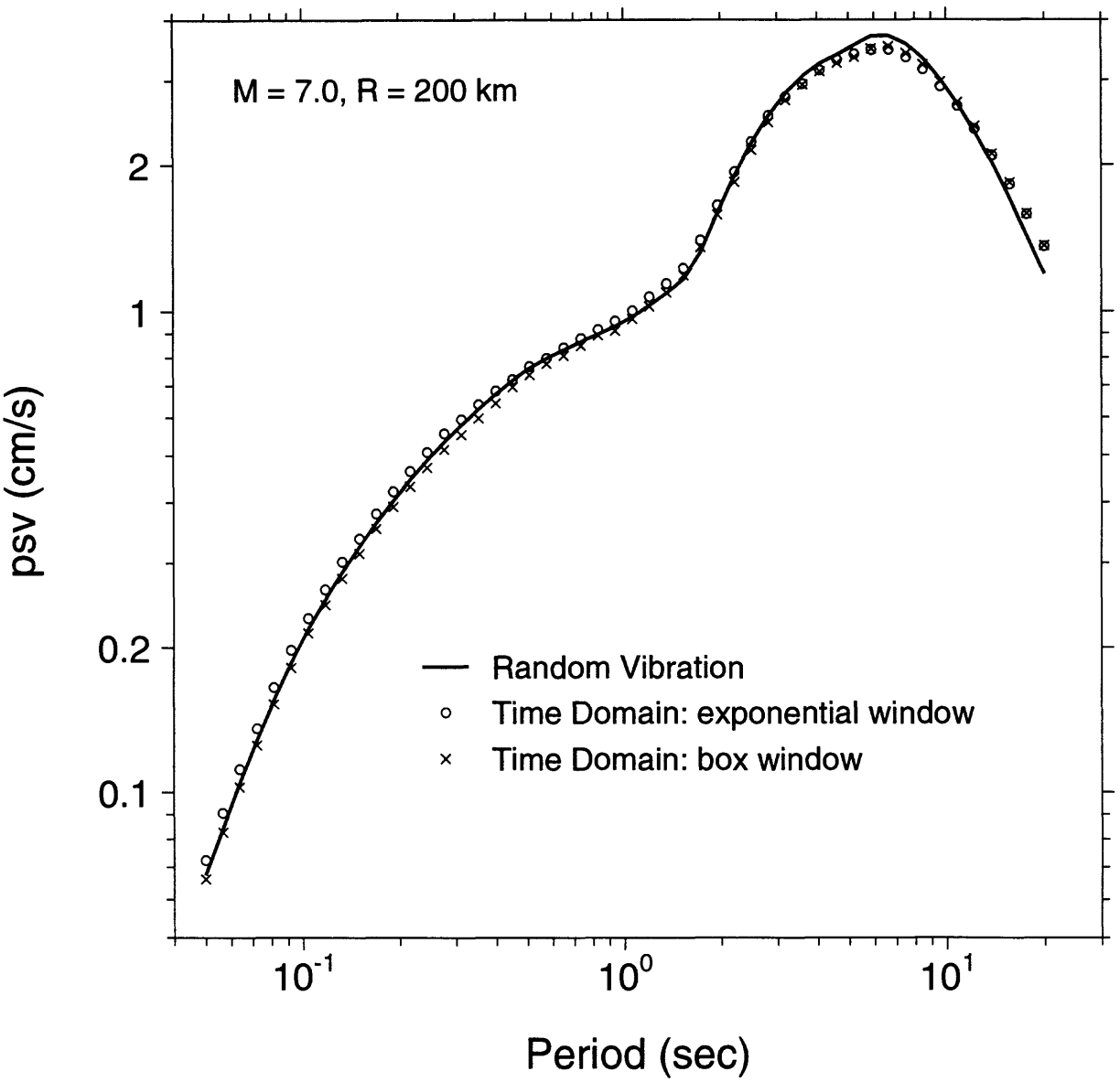

Figure 10. Comparison of simulations using box and exponential windows, with the random-vibration calculations for magnitude 7 at 200 km, using the parameters in the input-parameter file (except for the time-domain simulations, where $indxwind = 0$ for the box window and $tsimdur = 50.0$ for the exponential window).

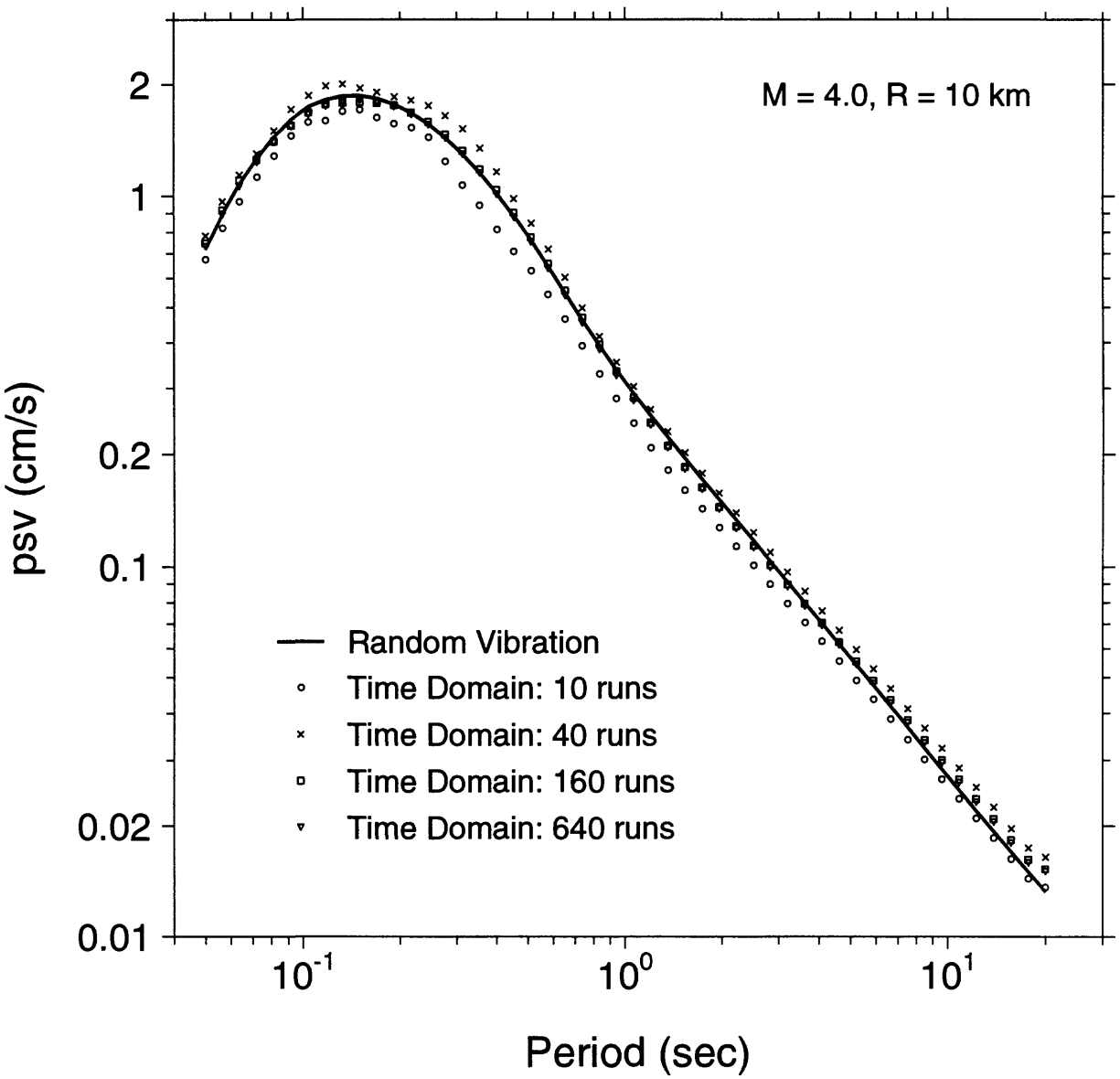

Figure 11. Comparison of simulations using the time-domain calculations with various values for *nruns*, with *seed* = *nruns* for each suite of realizations. The random-vibration results are shown for comparison. The calculations are for magnitude 4 at 10 km, using the parameters in the input-parameter file.

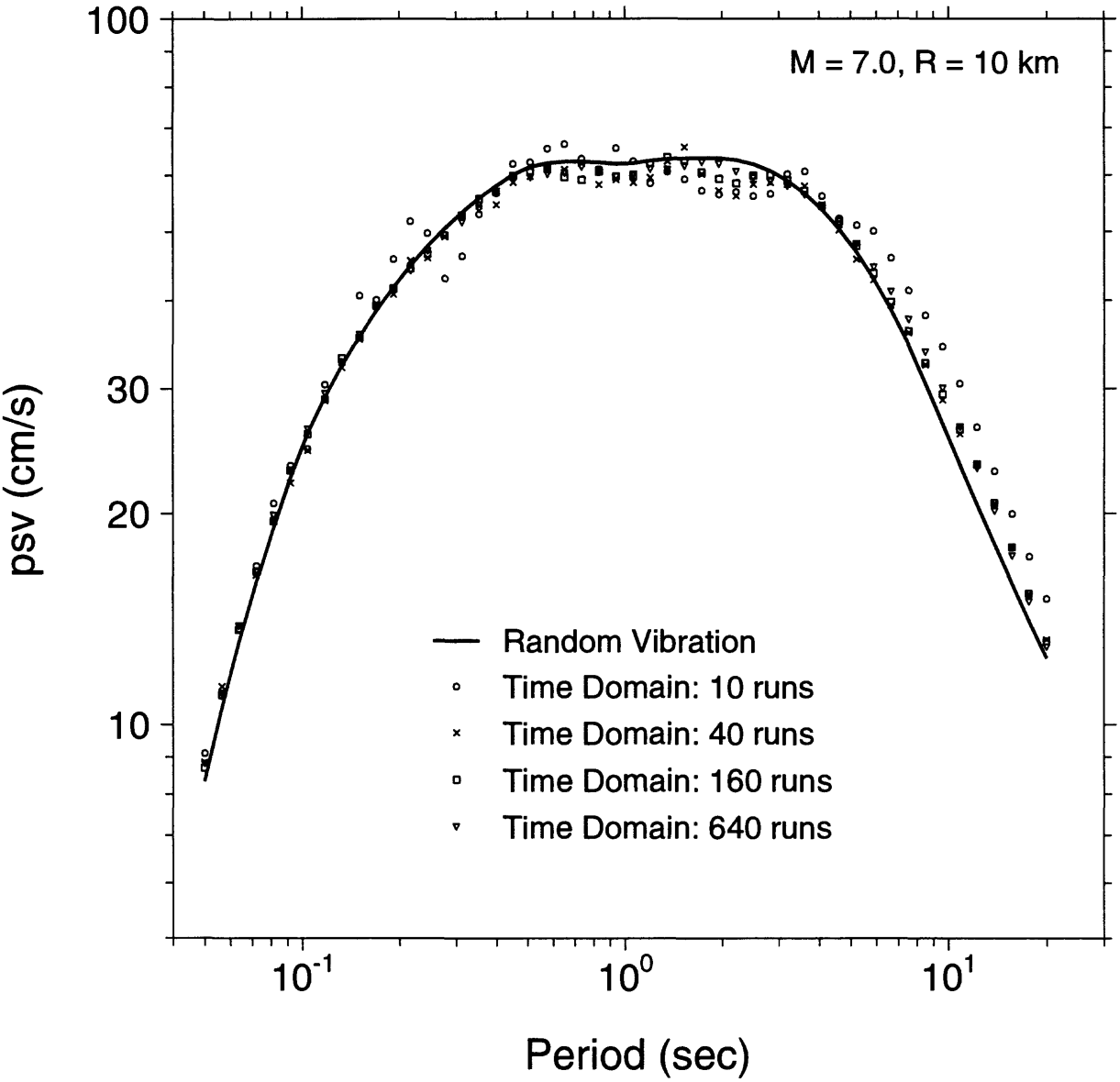

Figure 12. Comparison of simulations using the time-domain calculations with various values for *nruns*, with *seed* = *nruns* for each suite of realizations. The random-vibration results are shown for comparison. The calculations are for magnitude 7 at 10 km, using the parameters in the input-parameter file.

```
output file: sample.sum
*** Results computed using RV_DRVR ***
Date: 04/12/96
Time: 18:11:54.63
file with parameters: ofr.dat
Title:
    Sample data file **** NOT FOR A PARTICULAR APPLICATION **
rho, beta, prtitn, rtp, fs:
   2.80000       3.60000       0.710000       0.550000       2.00000
spectral shape: source number (1=Single Corner;2=Joyner;3=A93;4=custom)
pf, pd (1-corner spectrum = 1/(1+(f/fc)**pf)**pd; 0.0 otherwise)
(usual model: pf=2.0,pd=1.0; Butterworth: pf=4.0,pd=0.5)
(Note: power of high freq decay --> pf*pd)
          1    2.00000       1.00000
spectral scaling: stressc, dlsdm, fbdfa, amagc
(stress=stressc*10.0**(dlsdm*(amag-amagc))
(fbdfa, amagc for Joyner model, usually 4.0, 7.0)
(not used for srce 3, but placeholders still needed)
   80.0000       0.000000       4.00000       7.00000
gsprd: nsegs, (rlow(i), slope(i))  (Set rlow(1) = 1.0)
          3
   1.00000      -1.00000
   70.0000       0.000000
   130.000      -0.500000
q: fr1, Qr1, s1, ft1, ft2, fr2, qr2, s2
  0.100000       275.000      -2.00000       0.200000       0.600000
   1.00000       88.0000       0.900000
source duration: weights of 1/fa, 1/fb
   1.00000       0.000000
path duration: nknots, (rdur(i), dur(i), slope of last segment
          4
  0.000000       0.000000
   10.0000       0.000000
   70.0000        9.60000
   130.000        7.80000
  0.400000E-01
site amplification: namps, (famp(i), amp(i))
          5
  0.100000       1.00000
   1.00000       1.50000
   2.00000       2.00000
   5.00000       2.50000
   10.0000       3.00000
site diminution parameters: fm, akappa
   25.0000      0.300000E-01
low-cut filter parameters: fcut, norder
  0.000000              2
parameters for rv integration: zup, eps_int, amp_cutoff
   10.0000      0.100000E-04   0.100000E-02
calculated fup =  7.329E+01
fup calculated in driver =  7.329E+01

********** NEW R AND M **********
r, amag =   2.000E+02  7.000E+00
  Time Start: 18:13:07.95
  Column file: sample.col
  const=   4.757E-24
  amag, stress, fa, fb, durex=  7.000 8.00E+01 1.075E-01 1.075E-01 1.99E+01
  am0, am0b_m0fa=   3.548E+26  0.000E+00
                           pga(cm/s2)  domfreq     eps      nx      nz pk_rms
                             5.75E+00     6.12 0.8914  537.62  243.67   3.47
                  pgv(cm/s)             domfreq     eps      nx      nz pk_rms
                   1.96E+00                0.33 0.9985  243.73   13.23   2.47

Fractional oscillator damping =  0.050
 per(s)     freq prv(cm/s) paa(cm/s2)  domfreq     eps      nx      nz pk_rms
  0.100   10.000  2.08E-01   1.30E+01     8.91 0.4418  395.25  354.58   3.57
 10.000    0.100  2.89E+00   1.82E+00     0.10 0.7139    5.94    4.16   1.96
Time Stop: 18:13:08.12
Elapsed time (sec):     0.2
```

Figure 13. Sample summary file from the random-vibration program. The time-domain summary output is similar, except that it does not include estimates of dominant frequency.

Figure 13. Sample output summary file -- File SAMPLE.SUM, Page 1 of 1

per	freq	prv:sample_	paa:sample_	dom.freq.
1.000E-01	1.000E+01	2.076E-01	1.304E+01	8.908E+00
1.000E+01	1.000E-01	2.892E+00	1.817E+00	1.045E-01

Figure 14. Sample column file from the random-vibration program, for $\mathbf{M} = 7$ and $R = 200$km. The time-domain summary output is similar, except that it does not include estimates of dominant frequency.

T	A	V
0.000000	2.6975E-03	0.0000E+00
0.005000	2.6958E-03	1.3483E-05
0.010000	2.6958E-03	2.6962E-05
0.015000	2.6968E-03	4.0444E-05
0.020000	2.6996E-03	5.3935E-05
0.025000	2.7032E-03	6.7442E-05
0.030000	2.7081E-03	8.0970E-05
0.035000	2.7136E-03	9.4524E-05
0.040000	2.7197E-03	1.0811E-04
0.045000	2.7263E-03	1.2172E-04
0.050000	2.7330E-03	1.3537E-04
0.055000	2.7398E-03	1.4905E-04
0.060000	2.7468E-03	1.6277E-04
0.065000	2.7537E-03	1.7652E-04
0.070000	2.7608E-03	1.9031E-04
0.075000	2.7683E-03	2.0413E-04
0.080000	2.7760E-03	2.1799E-04
0.085000	2.7843E-03	2.3189E-04
0.090000	2.7938E-03	2.4584E-04
0.095000	2.8041E-03	2.5983E-04
0.100000	2.8157E-03	2.7388E-04
0.105000	2.8288E-03	2.8799E-04
0.110000	2.8436E-03	3.0217E-04
0.115000	2.8593E-03	3.1643E-04
0.120000	2.8765E-03	3.3077E-04
0.125000	2.8949E-03	3.4520E-04
0.130000	2.9140E-03	3.5972E-04
0.135000	2.9338E-03	3.7434E-04
0.140000	2.9543E-03	3.8906E-04
0.145000	2.9750E-03	4.0388E-04
0.150000	2.9959E-03	4.1881E-04
0.155000	3.0166E-03	4.3384E-04
0.160000	3.0372E-03	4.4898E-04
0.165000	3.0577E-03	4.6421E-04
0.170000	3.0780E-03	4.7955E-04
0.175000	3.0987E-03	4.9500E-04
0.180000	3.1199E-03	5.1054E-04
0.185000	3.1408E-03	5.2619E-04
0.190000	3.1629E-03	5.4195E-04
0.195000	3.1857E-03	5.5782E-04
0.200000	3.2095E-03	5.7381E-04

Figure 15. Sample of column file with acceleration and velocity time series, for $\mathbf{M} = 7$ and $R = 200$km. This is simulation 1 of the time-domain calculations using the sample input-parameter file

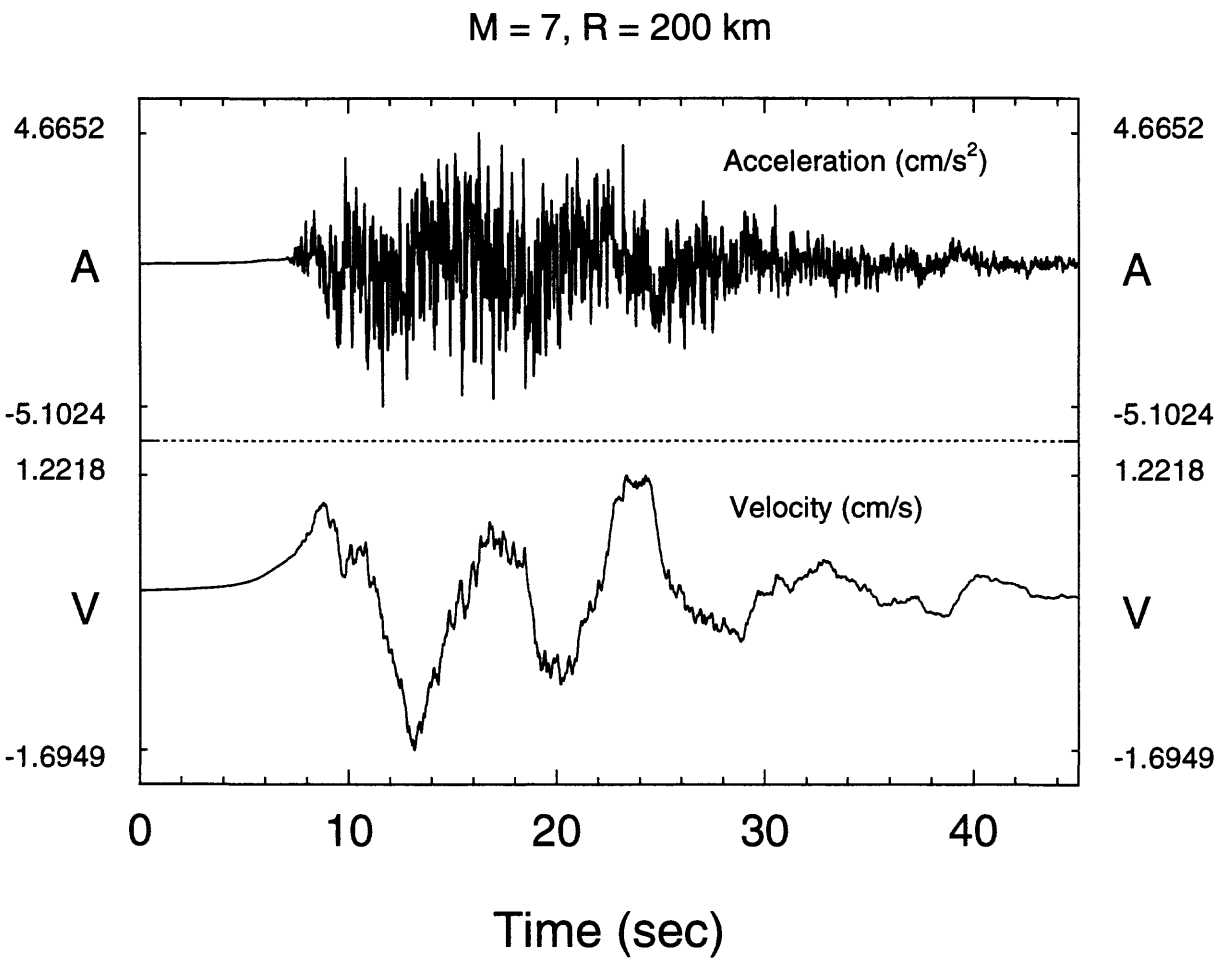

Figure 16. The time series from the column file partially shown in Figure 15.

Depth	SVel	Dens
0	0.300	2.0
0.001	0.300	2.0
0.005	0.500	0.0
0.010	0.600	0.0
0.015	0.700	0.0
0.020	0.800	0.0
0.030	0.900	0.0
0.040	1.000	0.0
0.040	1.500	0.0
0.060	1.500	0.0
0.080	1.500	0.0
0.100	1.500	3.0
0.150	1.600	0.0
0.200	1.800	0.0
0.300	2.000	0.0
0.300	3.500	0.0
1.000	3.500	4.0
8.000	3.500	4.0

Figure 17. Sample input-parameter file for the *SITE_AMP* program. **The file has been constructed for illustrative purposes and does not correspond to a real application.** Note that the velocity model is made up of a combination of constant-velocity layers and velocity gradients. For the sake of illustration, the density has been assigned specific (and unrealistic, in some cases) values for certain depths; these values override the densities assigned within the program when the input file contains 0.0 for the density. If the velocity model is a stack of constant-velocity layers, then layer thickness rather than depth could have been used, in which case the depths would not be repeated (i.e., there would be one entry per layer). A continuous velocity function should start with a depth of 0.0, as in the input-parameter file.

```
source vel & dens =   3.500E+00  2.800E+00
 Density coeffs not specified; assumed dens, vel, low, high =   2.500   .300  2.800  3.500
  ndepths =   18  nout =   15
depth_in    vel_in   dens_in   depth_out travltime thck_lyr vel_layer dens_layr    avgvel    avgdens  ofr_amp-frq  ofr_amp-amp
0.00E+00 3.000E-01 2.000E+00   1.00E-03 3.333E-03 1.00E-03 3.000E-01 2.000E+00 3.000E-01 2.000E+00   7.500E+01   4.041E+00
1.00E-03 3.000E-01 2.000E+00   5.00E-03 1.355E-02 4.00E-03 3.915E-01 2.519E+00 3.690E-01 2.196E+00   1.845E+01   3.478E+00
5.00E-03 5.000E-01 2.519E+00   1.00E-02 2.267E-02 5.00E-03 5.485E-01 2.528E+00 4.412E-01 2.327E+00   1.103E+01   3.089E+00
1.00E-02 6.000E-01 2.528E+00   1.50E-02 3.037E-02 5.00E-03 6.487E-01 2.537E+00 4.939E-01 2.380E+00   8.231E+00   2.888E+00
1.50E-02 7.000E-01 2.537E+00   2.00E-02 3.705E-02 5.00E-03 7.489E-01 2.547E+00 5.398E-01 2.409E+00   6.748E+00   2.745E+00
2.00E-02 8.000E-01 2.547E+00   3.00E-02 4.883E-02 1.00E-02 8.490E-01 2.556E+00 6.144E-01 2.443E+00   5.120E+00   2.555E+00
3.00E-02 9.000E-01 2.556E+00   4.00E-02 5.936E-02 1.00E-02 9.491E-01 2.566E+00 6.738E-01 2.464E+00   4.211E+00   2.429E+00
4.00E-02 1.000E+00 2.566E+00   6.00E-02 7.270E-02 2.00E-02 1.500E+00 2.612E+00 8.253E-01 2.491E+00   3.439E+00   2.183E+00
4.00E-02 1.500E+00 2.612E+00   8.00E-02 8.603E-02 2.00E-02 1.500E+00 2.612E+00 9.299E-01 2.510E+00   2.906E+00   2.049E+00
6.00E-02 1.500E+00 2.612E+00   1.00E-01 9.936E-02 2.00E-02 1.500E+00 3.000E+00 1.006E+00 2.550E+00   2.516E+00   1.954E+00
8.00E-02 1.500E+00 2.612E+00   1.50E-01 1.316E-01 5.00E-02 1.549E+00 2.622E+00 1.140E+00 2.614E+00   1.899E+00   1.814E+00
1.00E-01 1.500E+00 3.000E+00   2.00E-01 1.611E-01 5.00E-02 1.698E+00 2.641E+00 1.242E+00 2.617E+00   1.552E+00   1.737E+00
1.50E-01 1.600E+00 2.622E+00   3.00E-01 2.138E-01 1.00E-01 1.898E+00 2.659E+00 1.403E+00 2.625E+00   1.170E+00   1.631E+00
2.00E-01 1.800E+00 2.641E+00   1.00E+00 4.138E-01 7.00E-01 3.500E+00 4.000E+00 2.417E+00 3.000E+00   6.042E-01   1.163E+00
3.00E-01 2.000E+00 2.659E+00   8.00E+00 2.414E+00 7.00E+00 3.500E+00 4.000E+00 3.314E+00 3.829E+00   1.036E-01   8.788E-01
3.00E-01 3.500E+00 2.800E+00         .         .       .         .         .         .         .           .           .
1.00E+00 3.500E+00 4.000E+00         .         .       .         .         .         .         .           .           .
8.00E+00 3.500E+00 4.000E+00         .         .       .         .         .         .         .           .           .
```

Figure 18. Sample output for the input file in Figure 17. In this case, the density–velocity function uses a straight line between the default values of 2.5 gm/cc for 0.3 km/s and 2.8 gm/cc for 3.5 km/s (see text). This density–velocity relation is only used if $density = 0.0$ in the input. The rows of periods at the bottom of columns 4 through 12 represent null values; they are needed in order to import properly the output file into the graphics program that I use to plot the amplifications.

APPENDICES: SOURCE LISTINGS AND INPUT-PARAMETER FILES

```
* ---------------- BEGIN RV_DRVR --------------------------
      Program RV_Drvr

*  Obtains input parameters and calls random vibration simulation

* Dates: 06/01/95 - Written by D.M. Boore; Modified from TD_DRVR
*        06/09/95 - Bells and whistles still being added
*        08/11/95 - Add frequency column to output
*        08/11/95 - Changed places where parameter-file and column-file names
*                   are requested
*        08/18/95 - Added call to Write_Params
*        10/17/95 - Added a flag to Get_Params and Write_Params to tell if using
*                   time domain or random vibration procedure
*        11/14/95 - Combined smsim.fi and rv.fi
*        11/16/95 - Minor changes to i/o; moved calculation of fup to get_params
*        12/06/95 - Added "patience" message on screen.
*        12/08/95 - Added switch for 91 standard periods.
*        12/14/95 - Print out frequency and duration of excitation
*        12/14/95 - Repeated computation of fup in this driver.
*        12/17/95 - Changed location of asking for col file and reordered
*                   some output
*        12/31/95 - Modified some of the *.sum output
*        01/03/96 - Added pk_rms_cl_eq68 to *.sum output
*        01/05/96 - Added column file stem name to column headings
*        02/06/96 - Minor formatting improvements
*        04/12/96 - changed names psv, psa to prv, paa

      character fname_params*80, fname_out*80, buf*80
      character colprv_head*12, colpaa_head*12
      character prvcalc*1, indvidper*1, f_col*80, message*80,
     :  standard_periods*1,
     :  date_begin*10, time_begin*11, time_start*11, time_stop*11
      logical tdflag, fparam_exist

      REAL PER(91)
      DATA PER/0.040,0.042,0.044,0.046,0.048,
     * 0.050,0.055,0.060,0.065,0.070,
     * 0.075,0.080,0.085,0.090,0.095,0.10,0.11,0.12,0.13,0.14,
     * 0.15,0.16,0.17,0.18,0.19,0.20,0.22,0.24,0.26,0.28,
     * 0.30,0.32,0.34,0.36,0.38,0.40,0.42,0.44,0.46,0.48,
     * 0.50,0.55,0.60,0.65,0.70,0.75,0.80,0.85,0.90,0.95,
     * 1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,
     * 2.0,2.2,2.4,2.6,2.8,3.0,3.2,3.4,3.6,3.8,
     * 4.0,4.2,4.4,4.6,4.8,5.0,5.5,6.0,6.5,7.0,
     * 7.5,8.0,8.5,9.0,9.5,10.0,11.0,12.0,13.0,14.0,
     * 15.0/

      include 'smsim.fi'

      pi = 4.0 * atan(1.0)
      twopi = 2.0 * pi

      write(*, '(a\)')
     :    ' Enter a message, if desired: '
      message = ' '
      read(*, '(a)') message

333   continue
      fname_params = ' '
      write(*, '(a\)')
     :    ' Enter name of file with parameters (cr to quit): '
      read(*, '(a)') fname_params
      if (fname_params(1:4) .eq. '    ') stop
      inquire(file=fname_params, exist=fparam_exist)
      if (.not. fparam_exist) then
        write(*,'(a)') ' ******* FILE DOES NOT EXIST ******* '
        go to 333
      end if

      fname_out = ' '
      write(*, '(a\)')
     :    ' Enter name of summary file: '
      read(*, '(a)') fname_out
      nout = 10
      open(unit=nout,file=fname_out,status='unknown')
      write(nout, '(a)') message
      write(nout, '(2a)') ' output file: ',
     :   fname_out(1:20)

      write(nout, '(a)')
     :  ' *** Results computed using RV_DRVR ***'

      call get_date(date_begin)
      write(nout, '(2a)') ' Date: ', date_begin
      call get_time(time_begin)
      write(nout, '(2a)') ' Time: ', time_begin

      write(nout, '(2a)') ' file with parameters: ',
     :   fname_params(1:20)

      write(*, '(a\)') ' Compute prv (y/n): '
      prvcalc = 'n'
      read(*, '(a)') prvcalc
      if (prvcalc .eq. 'y' .or. prvcalc .eq. 'Y') then
        write(*, '(a\)')
     :  ' Enter individual periods(y/n)?: '
        indvidper = ' '
        read(*, '(a)') indvidper
        if (indvidper .eq. 'y' .or. indvidper .eq. 'Y') then
          write(*, '(a\)')
     :       ' Enter fractional damping, nperiods: '
          read(*, *) damp, nper
          do i = 1, nper
            write(*, '(a\)') '    Enter oscillator period: '
            read(*,*) per(i)
          end do
        else
          write(*, '(a\)') ' Use standard set of 91 periods (y/n)? '
          standard_periods = ' '
          read(*, '(a)') standard_periods
          nper = 91
          if (standard_periods .eq. 'y' .or.
     :        standard_periods .eq. 'Y') then
            nper = 91
            write(*, '(a\)') ' Enter fractional damping: '
            read(*, *) damp
          else
            write(*, '(a\)')
     :       ' Enter fractional damping, perlow, perhigh, nper: '
            read(*,*) damp, perlow, perhigh, nper
            do i = 1, nper
              per(i) =
     :          perlow * (perhigh/perlow)**(float(i-1)/float(nper-1))
            end do
          end if
        end if
      else
        nper = 0
      end if

      tdflag = .false.  ! controls extent of input file read and written
      call get_params( fname_params, tdflag )
```

```
      call write_params( nout, tdflag)

* compute fup (this is also done in rvtdsubs, but if the driver is changed
* such that there is a loop over fm or akappa, then fup should be recomputed
* within the loop; I have repeated the calculation here as a reminder
* in case the driver is changed):
      if( akappa .eq. 0.0) then
        fup = fm/amp_cutoff**0.25
      else
        fup =
     :   amin1(fm/amp_cutoff**0.25, -alog(amp_cutoff)/(pi*akappa))
      end if
      write(nout, '(a,1pe10.3)') ' fup calculated in driver = ', fup

100   continue

      buf = ' '
      write(*, '(a\)') ' dist (cr to quit): '
      read(*, '(a)') buf
      if (buf(1:4) .eq. '    ') go to 999
      read(buf, '(f10.0)') r

      write(*, '(a\)') ' amag: '
      read(*, *) amag

      new_mr = .true.

      if (prvcalc .eq. 'y' .or. prvcalc .eq. 'Y') then
        write(*, '(a\)')
     :       ' Enter name of column file for prv results: '
        f_col = ' '
        read( *, '(a)') f_col
        ncol = 50
        open(unit=ncol, file=f_col, status='unknown')
        colprv_head = 'prv:________'
        colprv_head(5:index(f_col,'.')+3) = f_col(1:index(f_col,'.')-1)
        colpaa_head = 'paa:________'
        colpaa_head(5:index(f_col,'.')+3) = f_col(1:index(f_col,'.')-1)
        write(ncol, '(t8,a, t17,a, t22,a, t35,a, t48,a)')
     :   'per', 'freq', colprv_head, colpaa_head, 'dom.freq.'

      end if

      call get_time(time_start)

      iaorins = 1
      idva = 1
      write(*,'(a)')
     : '  Patience! Computing peak velocity'
      call smsim_rv(pgvsim)
      freq20pgv = freq20
      ane_pgv = ane
      anz_pgv = anz
      eps_pgv = eps_rv
      pk_rms_pgv = pk_rms_cl_eq68

      idva = 2
      write(*,'(a)')
     : '  Patience! Computing peak acceleration'
      call smsim_rv(pgasim)
      freq20pga = freq20
      ane_pga = ane
      anz_pga = anz
      eps_pga = eps_rv
      pk_rms_pga = pk_rms_cl_eq68

      write(nout,*)
      write(nout, '(a)') ' ********** NEW R AND M ********** '
      write(nout, '(a, 1p2(1x,e10.3))')
     :     ' r, amag = ', r, amag
      write(nout, '(2x,2a)') ' Time Start: ', time_start
      write(nout, '(2x,2a)') ' Column file: ', f_col(1:30)
      write(nout, '(2x,a,1x,1pe10.3)') ' const= ', const
      write(nout, '(2x,a,f6.3,1pe9.2,2e10.3,e9.2)')
     :  ' amag, stress, fa, fb, durex= ',
     :    amag, stress, fa, fb, durex
      write(nout, '(2x,a,1p2(1x,e10.3))') ' am0, am0b_m0fa= ',
     :                                    am0, am0b_m0
      write(nout,'( t31,a, t43,a, t55,a,
     :              t63,a, t71,a, t74,a)')
     :            'pga(cm/s2)',
     :            'domfreq', 'eps', 'nx', 'nz', 'pk_rms'
      write(nout,
     :     '( 1p, t31,e10.2, 0p, t43,f7.2, t51,f6.4, t58,f7.2,
     :       t66,f7.2, t74,f6.2)')
     :     pgasim, freq20pga, eps_pga, ane_pga, anz_pga, pk_rms_pga
      write(nout,'( 2x, t21,a, t43,a, t54,a,
     :              t63,a, t71,a, t74,a)')
     :            'pgv(cm/s)',
     :            'domfreq', 'eps', 'nx', 'nz', 'pk_rms'
      write(nout,
     :     '( 1p, t21,e9.2, 0p, t43,f7.2, t51,f6.4, t58,f7.2,
     :       t66,f7.2, t74,f6.2)')
     :     pgvsim, freq20pgv, eps_pgv, ane_pgv, anz_pgv, pk_rms_pgv

      write(*,*)

      if (prvcalc .eq. 'y' .or. prvcalc .eq. 'Y') then
        write(nout,'(2x,a,f6.3)')
     :       ' Fractional oscillator damping = ', damp
        write(nout,'( t5,a, t16,a, t21,a, t31,a, t43,a, t54,a,
     :              t63,a, t71,a, t74,a)')
     :            'per(s)', 'freq', 'prv(cm/s)', 'paa(cm/s2)',
     :            'domfreq', 'eps', 'nx', 'nz', 'pk_rms'
        iaorins = 2
        do i = 1, nper
          write(*,'(a,i3,a,i3,a)')
     :    '+ Patience! Computing response spectra number ',
     :     i, ' out of ', nper, ' periods.'

          perosc = per(i)
          call smsim_rv(prvsim)
          freq20prv = freq20
          ane_prv = ane
          anz_prv = anz
          eps_prv = eps_rv
          pk_rms_prv = pk_rms_cl_eq68

          write(nout,
     :      '( t4,f7.3, t13,f7.3, 1p, t21,e9.2, t31,e10.2,
     :               0p, t43,f7.2,     t51,f6.4, t58,f7.2,
     :        t66,f7.2, t74,f6.2)')
     :      perosc, 1.0/perosc, prvsim, (twopi/perosc)*prvsim,
     :      freq20prv,
     :      eps_prv, ane_prv, anz_prv, pk_rms_prv

          write(ncol,'(1p2e10.3,2e13.3,e10.3)')
     :      perosc, 1.0/perosc, prvsim, (twopi/perosc)*prvsim,
     :      freq20prv
        end do
        close(unit=ncol)
      end if
```

```
      call get_time(time_stop)
      write(nout, '(2x,2a)') ' Time Stop: ', time_stop
      call time_diff(time_start, time_stop, time_elapsed)
      write(nout, '(2x,a,1x,f6.1)')
     :       ' Elapsed time (sec): ', time_elapsed

      write(*,*)
      write(*, '(a\)')
     : ' Compute results for another r and M (y/n;cr=quit)? '
      read(*, '(a)') buf
      if (buf(1:4) .eq. '    ') go to 999
      if (buf(1:1) .eq. 'n' .or. buf(1:1) .eq. 'N') go to 999

      goto 100

999   continue
      close(unit=nout)
      stop
      end
* ----------------- END RV_DRVR --------------------------
```

```
*---------------- BEGIN SMSIM_RV ----------------------------
      subroutine smsim_rv(gmsim)
* Dates: 06/07/95 - Written by D.M. Boore
*        06/08/95 - Pass amag, r through common block in smsim.fi
*        11/14/95 - Combined smsim.fi and rv.fi
*        12/14/95 - Pass freq20, durex through common block in smsim.fi

* Uses random-vibration stochastic model to compute ground motion
*  (SMSIM = Stochastic Model Simulation)

* Dates:  05/10/95 - Written by D. M. Boore, based on RVIB

      include 'smsim.fi'

      pi = 4.0*atan(1.0)
      twopi = 2.0 * pi

* Set spectral parameters:
      call spect_scale()   ! Be sure to call Get_Params in driver

* Get frequency-independent factor:
      call const_am0_gsprd()

* Set parameters related to type of output:
      fosc = 1.0
      if (iaorins .eq. 2) then
        fosc = 1.0/perosc
      end if
*---------------------------------------
* Assume that instrument response is relative to ground
* displacement.  In this case idva = 0, and to make sure
* that this is so, I include the following statement:

      if (iaorins .gt. 1) idva = 0
*---------------------------------------

      call get_motion(gmsim)

      return
      end
*---------------- END SMSIM_RV ------------------------------

*---------------- BEGIN GET_MOTION --------------------------
      subroutine get_motion(gmsim)

* Dates: 06/06/95 - written by D.M. Boore, adapted from main.for, which see for
*                   code for printing out much more info.
*        06/07/95 - Changed to two term asymptotic expression for output.  The
*                   'exact' causes problems.
*        06/08/95 - Changed anz to have a minimum value of 1.33 (as mentioned
*                   by Toro in section 3, vol. 3, appendices B & C, Seismic
*                   Hazard Methodology for Nuclear Facilities in the Eastern
*                   United States, EPRI, April 30, 1985, eq. B-35, p. B-80.)
*        06/08/95 - Changed names of variables such as 'anclee' to more
*                   meaningful names, and also put the sqrt 2 into the proper
*                   place rather than carrying it around with rms (in
*                   'afact' in subroutine main on the VAX).
*        06/09/95 - Changed method of computing 'exact' solution to a numerical
*                   integration.
*        10/17/95 - Eliminated exact series evaluation, as well as the switch
*                   from asymptotic to "exact"; I also eliminated the commented
*                   statements that included Toro's "clumping" correction...
*                   the source code for that has been retained in smsmclmp.for.
*        12/14/95 - Pass freq20, durex through common block in smsim.fi
*        12/19/95 - Added zup to exact_numrcl (as of 1/3/96, cl68_numrcl_int)
*        12/25/95 - Add Herrmann's integration of C&L-H eq. 6.4
*        12/26/95 - Pass eps_rv and ane through smsim.fi common rather than
*                   through the parameter list
*        12/28/95 - Changed variable names to indicate that equation 6.8 of
*                   Cartwright and Longuet-Higgins is being used.
*        12/30/95 - Remove Herrmann's integration of C&L-H eq. 6.4

      include 'smsim.fi'

      character e_or_a*1

      real amom0,amom1,amom2,amom4,anz,ane

c
c compute moments and frequencies, bandwidth factors, rms.
c
C
      amom0=amom_rv(0)
      amom1=amom_rv(1)
      amom2=amom_rv(2)
      amom4=amom_rv(4)

      arg=1.0-amom1*amom1/amom0/amom2
      if(abs(arg).lt.1.0e-20) arg=1.0e-20
      deltay=sqrt(arg)                    ! see MAIN, used in V-M or T&U calcs
      xi = amom2/sqrt(amom0*amom4)
      arg=(1.-xi*xi)
      if(abs(arg).lt.1.0e-20) arg=1.0e-20
      eps_rv=sqrt(arg)
      freq20=sqrt(amom2/amom0)/(2.0*pi)
      freq42=sqrt(amom4/amom2)/(2.0*pi)
c
c because the acceleration is a transient, the durations for
c computing rms and for determining n may be different.
c durex Duration of excitation) is used for computing N.
c The rms is computed using
c trms, which is determined as durex for
c regular time series and durex + tosc * ( rf/(rf+avib) ) for the
c oscillator output, where tosc is the time for an oscillator
c to decay to 1/e and rf = (fosc * durex)**3. avib is an adjustable
c parameter, currently set to 1/3. The reason for using the oscillator
c duration is that the random vibration theory gives the ratio
c of peak to rms. If the spectral energy is spread over a number of
c cycles as in an oscillator, the "local" rms is smaller than
c if contained just within durex.
c In effect, we are forced to apply a fixup in an
c attempt to get around this basic limitation.
c
      avib = 1./3.
c determine duration of excitation.
      durex = dursource(w_fa, fa, w_fb, fb)+
     :  durpath(r, nknots, rdur, dur, slast)
c determine duration of rms
      trms = durex
      if (iaorins .eq. 2 .or. iaorins .eq. 3 ) then
         rf = (fosc * durex)**3
         tosc = 1.0/(twopi*damp*fosc)
         trms = trms + tosc * ( rf/(rf+avib) )
      end if
      rms=sqrt(amom0/trms)
      ane = 2.0*freq42 * durex
      anz = 2.0*freq20 * durex
      if (ane .le. 1.0) ane = 1.002
      if (anz .le. 1.33) anz = 1.33

c factor of 2.0 because consider positive maxima
c and negative minima.
```

```
c ane is an estimate of the total number of extrema.
c anz ( also=ane*sqrt(1.0-eps_rv*eps_rv) ) is an estimate of the
c number of positive and negative zero crossings.
c
c
c compute Cartwright and Longuet-Higgins estimates of peak/rms:
c
      pk_rms_cl_1 = sqrt(2.0*alog(anz))             ! 'cl' = Cart. & L-H
      pk_rms_cl_2 = pk_rms_cl_1+0.5772/pk_rms_cl_1
      pk_rms_cl_eq68 = cl68_numrcl_int( ane, xi, zup)

      e_or_a = 'e'                  ! might be used in a print statement

      pk_cl_eq68 = rms * pk_rms_cl_eq68    ! eq. 6.8 of C&L-H
      pk_cl_1 = rms * pk_rms_cl_1          ! 1 term asymptotic
      pk_cl_2 = rms * pk_rms_cl_2          ! 2 term asymptotic

      gmsim = pk_cl_eq68

c
c that is all
c
      return
      end
*---------------- END GET_MOTION ----------------------------

*---------------- BEGIN CL68_NUMRCL_INT ---------------------------
      function cl68_numrcl_int( an_in, xi_in, zup)
* Numerical integration of eq. 6.8 in Cartwright and Longuet-Higgins
* Dates: 06/09/95 - Written by D.M. Boore, and tested using CHK_INT.
*                   I also plotted the integrand for typical values
*                   of an, xi, and found that it decays strongly to zero
*                   by a value of 5 for the variable.  I use 10 as an upper
*                   limit, which should be much more than enough.  The
*                   integration routines are such, however, that I could
*                   probably use a much larger m=number with little extra
*                   time.
*        12/19/95 - Added zup to exact_numrcl
*        12/28/95 - Name changed from exact_numrcl to cl68_numrcl
*        01/03/96 - Name changed from cl68_numrcl to cl68_numrcl_int

      external cl68_integrand

      common /clint/ xi, an

      an = an_in
      xi = xi_in
      zlow = 0.0
      call qmidpnt(cl68_integrand,zlow,zup,result)
      cl68_numrcl_int = result/sqrt(2.0)

      return
      end
*---------------- END CL68_NUMRCL_INT ----------------------------

*---------------- BEGIN CL68_INTEGRAND ----------------------------
      function cl68_integrand(z)
* Dates: 06/09/95 - Written by D.M. Boore.  See 7/11/82 notes for
*                   stochastic model, with 6/9/95 addition that uses
*                   a variable transformation to remove the sqrt
*                   singularity at the origin.
*        01/03/95 - Name changed from cl_int to cl68_integrand

      common /clint/ xi, an
      cl68_integrand = 2.0*(1.0-(1.0-xi*exp(-z**2))**an)
      return
      end
*---------------- END CL68_INTEGRAND ----------------------------

*---------------- BEGIN AMOM_RV ----------------------------
      function amom_rv(i)
c Dates: 06/06/95 - Written by D.M. Boore, patterned after AMOMI, which
*                   see for more detailed history.
*        11/14/95 - Obtain eps_int from get_params and pass through common

      external derivs

      include 'smsim.fi'

      h1 = 0.1
      hmin = 0.0
      imom = i       ! keep param i in parameter list rather than imom;
                     ! imom is passed through the rv common block

      result = 0.0
      call odeint(result, 1, 0.0, fup, eps_int, h1, hmin,
     :            nok, nbad, derivs)


      amom_rv = 2.0 * result

      return
      end
*---------------- END AMOM_RV ----------------------------

*---------------- BEGIN DERIVS ----------------------------
* Dates: 06/07/95 - Written by D.M. Boore
      subroutine derivs(freq, y, dydf)

      include 'smsim.fi'

      f = freq
      if (freq .eq. 0.0) f = 0.001
      w = twopi * f

      a = spect_amp(f)

      if(imom .eq. 0) then
        dydf=a*a
      else
        dydf=a*a*w**imom
      end if

      return
      end
*---------------- END DERIVS ----------------------------

      include 'rvtdsubs.for'
      include 'recipes.for'
```

```
* -------------- BEGIN TD_DRVR --------------------------------
      Program TD_Drvr

*  Obtains input parameters and calls time domain simulation

* Dates: 06/01/95 - Written by D. M. Boore;
*                   Renamed and slightly modified PSV_DRVR
*        06/09/95 - Pass r, amag through common rather than parameter lists
*        06/12/95 - Added optional writing of acc, vel time series to a file
*        08/11/95 - Added frequency column to psv output
*        08/11/95 - Changed places where parameter-file and column-file names
*                   are requested
*        08/18/95 - Added call to Write_Params
*        10/17/95 - Added flag to get_params and write_params that tells
*                   whether are dealing with time domain or rv.
*        11/14/95 - Modified output slightly
*        12/08/95 - Added switch to use 91 standard periods
*        12/14/95 - Print out frequency and duration of excitation
*        12/17/95 - Changed location of asking for col file name and
*                   reorder some output
*        12/28/95 - Used new_mr to correct bug related to use of loop over
*                   amag,r
*        12/31/95 - Modified some of the *.sum output
*        01/05/96 - Added column file stem name to column headings
*        01/22/96 - Added total duration, npts to output
*        02/06/96 - Minor formatting improvements
*        04/12/96 - changed names psv, psa to prv, paa; changed
*                   format of time in accvel file to fixed format.

      character colprv_head*12, colpaa_head*12
      character fname_params*80, fname_out*80, buf*80
      character fname_accvel*80, save_av*1
      character prvcalc*1, indvidper*1, f_col*80, message*80,
     :  standard_periods*1,
     :  date_begin*10, time_begin*11, time_start*11, time_stop*11
      logical tdflag, fparam_exist

      real prvsim(100),
     :     acc_save(16400), vel_save(16400)

      REAL PER(91)
      DATA PER/0.040,0.042,0.044,0.046,0.048,
     * 0.050,0.055,0.060,0.065,0.070,
     * 0.075,0.080,0.085,0.090,0.095,0.10,0.11,0.12,0.13,0.14,
     * 0.15,0.16,0.17,0.18,0.19,0.20,0.22,0.24,0.26,0.28,
     * 0.30,0.32,0.34,0.36,0.38,0.40,0.42,0.44,0.46,0.48,
     * 0.50,0.55,0.60,0.65,0.70,0.75,0.80,0.85,0.90,0.95,
     * 1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,
     * 2.0,2.2,2.4,2.6,2.8,3.0,3.2,3.4,3.6,3.8,
     * 4.0,4.2,4.4,4.6,4.8,5.0,5.5,6.0,6.5,7.0,
     * 7.5,8.0,8.5,9.0,9.5,10.0,11.0,12.0,13.0,14.0,
     * 15.0/

      include 'smsim.fi'

      pi = 4.0 * atan(1.0)
      twopi = 2.0 * pi

      write(*, '(a\)')
     :    ' Enter a message, if desired: '
      message = ' '
      read(*, '(a)') message

333   continue
      fname_params = ' '
      write(*, '(a\)')
     :    ' Enter name of file with parameters (cr to quit): '
      read(*, '(a)') fname_params
      if (fname_params(1:4) .eq. '    ') stop
      inquire(file=fname_params, exist=fparam_exist)
      if (.not. fparam_exist) then
        write(*,'(a)') ' ******* FILE DOES NOT EXIST ******* '
        go to 333
      end if

      fname_out = ' '
      write(*, '(a\)')
     :    ' Enter name of summary file: '
      read(*, '(a)') fname_out
      nout = 10
      open(unit=nout,file=fname_out,status='unknown')
      write(nout, '(a)') message
      write(nout, '(2a)') ' summary file: ',
     :   fname_out(1:20)

      write(nout, '(a)')
     :    ' *** Results computed using TD_DRVR ***'

      call get_date(date_begin)
      write(nout, '(2a)') ' Date: ', date_begin
      call get_time(time_begin)
      write(nout, '(2a)') ' Time: ', time_begin

      write(nout, '(2a)') ' file with parameters: ',
     :   fname_params(1:20)

      write(*, '(a\)') ' Compute prv (y/n): '
      prvcalc = 'n'
      read(*, '(a)') prvcalc
      if (prvcalc .eq. 'y' .or. prvcalc .eq. 'Y') then
        write(*, '(a\)')
     :  ' Enter individual periods(y/n)?: '
        indvidper = ' '
        read(*, '(a)') indvidper
        if (indvidper .eq. 'y' .or. indvidper .eq. 'Y') then
          write(*, '(a\)')
     :       ' Enter fractional damping, nperiods: '
          read(*, *) damp, nper
          do i = 1, nper
            write(*, '(a\)') '    Enter oscillator period: '
            read(*,*) per(i)
          end do
        else
          write(*, '(a\)') ' Use standard set of 91 periods (y/n)? '
          standard_periods = ' '
          read(*, '(a)') standard_periods
          nper = 91
          if (standard_periods .eq. 'y' .or.
     :        standard_periods .eq. 'Y') then
            nper = 91
            Write(*, '(a\)') ' Enter fractional damping: '
            read(*, *) damp
          else
            write(*, '(a\)')
     :       ' Enter fractional damping, perlow, perhigh, nper: '
            read(*,*) damp, perlow, perhigh, nper
            do i = 1, nper
              per(i) =
     :          perlow * (perhigh/perlow)**(float(i-1)/float(nper-1))
            end do
          end if
        end if
```

```
      else
        nper = 0
      end if

      tdflag = .true.  ! controls extent of the input file read and written
      call get_params( fname_params, tdflag )
      call write_params(nout, tdflag)

      write(nout, *)
      write(nout, '(a,i6, f8.5, f6.1)')
     :  ' npts, dt, total duration = ', npts, dt, npts*dt
      write(nout, *)

  100 continue

      buf = ' '
      write(*, '(a\)') ' dist (cr to quit): '
      read(*, '(a)') buf
      if (buf(1:4) .eq. '    ') go to 999
      read(buf, '(f10.0)') r

      write(*, '(a\)') ' amag: '
      read(*, *) amag

      new_mr = .true.

      if (prvcalc .eq. 'y' .or. prvcalc .eq. 'Y') then
        write(*, '(a\)')
     :  ' Enter name of column file to contain response spectra: '
        f_col = ' '
        read( *, '(a)') f_col
        ncol = 50
        open(unit=ncol, file=f_col, status='unknown')
        colprv_head = 'prv:_______'
        colprv_head(5:index(f_col,'.')+3) = f_col(1:index(f_col,'.')-1)
        colpaa_head = 'paa:_______'
        colpaa_head(5:index(f_col,'.')+3) = f_col(1:index(f_col,'.')-1)
        write(ncol, '(t8,a, t17,a, t22,a, t35,a)')
     :   'per', 'freq', colprv_head, colpaa_head
      end if

      write(*, '(a\)') ' Save acc & vel time series? (y/n): '
      save_av = 'n'
      read(*, '(a)') save_av
      if (save_av .eq. 'y' .or. save_av .eq. 'Y') then
        write(*, '(a\)')
     : '    Save which simulation (give sim number)? '
        read(*, *) nacc_save
        write(*, '(a\)')
     : '    Enter file name for time series (cr=accvel.col): '
        fname_accvel = ' '
        read(*, '(a)') fname_accvel
        if (fname_accvel(1:3) .eq. '   ') then
          fname_accvel = 'accvel.col'
        end if
      else
        nacc_save = 0
      end if

      call get_time(time_start)

      call smsim_td(per, nper,
     :   prvsim, pgasim, pgvsim, prvcalc,
     :   nacc_save, acc_save, vel_save)

      if (save_av .eq. 'y' .or. save_av .eq. 'Y') then
        nuacc = 11
        open(unit=nuacc,file=fname_accvel,status='unknown')
        write(nuacc, '(t12,a,t24,a,t36,a)') 'T', 'A', 'V'
        do i = 1, npts
          write(nuacc, '(1x,f11.6,1p2(1x,e11.4))')
     :             float(i-1)*dt, acc_save(i), vel_save(i)
        end do
        close(unit=nuacc)
      end if

      write(nout, *)
      write(nout, '(a)') ' ********** NEW R AND M **********'
      write(nout, '(a, 1p2(1x,e10.3))')
     :     ' r, amag = ', r, amag
      write(nout, '(2x,2a)') ' Time Start: ', time_start
      write(nout, '(2x,2a)') ' Column file: ', f_col(1:30)
      write(nout, '(2x,a,1x,1pe10.3)') ' const= ', const
      write(nout, '(2x,a,f6.3,1pe9.2,2e10.3,e9.2)')
     :  ' amag, stress, fa, fb, durex= ',
     :    amag, stress, fa, fb, durex
      write(nout, '(2x,a,1p2(1x,e10.3))') ' am0, am0b_m0fa= ',
     :                                    am0, am0b_m0
      write(nout,'( t21,a, t31,a)') 'pgv(cm/s)', 'pga(cm/s2)'
      write(nout,'(1p,t21,e9.2, t31,e10.2)') pgvsim, pgasim

      write(*,*)

      if (prvcalc .eq. 'y' .or. prvcalc .eq. 'Y') then
        write(nout,'(2x,a,f6.3)')
     :       ' Fractional oscillator damping = ', damp
        write(nout,'( t5,a, t16,a, t21,a, t31,a)')
     :            'per(s)', 'freq', 'prv(cm/s)', 'paa(cm/s2)'
        do i = 1, nper
          write(nout,
     :      '( t4,f7.3, t13,f7.3, 1p, t21,e9.2, t31,e10.2)')
     :      per(i), 1.0/per(i), prvsim(i), (twopi/per(i))*prvsim(i)
          write(ncol,'(1p2e10.3,2e13.3,e10.3)')
     :      per(i), 1.0/per(i), prvsim(i), (twopi/per(i))*prvsim(i)
        end do
        close(unit=ncol)
      end if

      call get_time(time_stop)
      write(nout, '(2x,2a)') ' Time Stop: ', time_stop
      call time_diff(time_start, time_stop, time_elapsed)
      write(nout, '(2x,a,1x,f6.1)')
     :     ' Elapsed time (sec): ', time_elapsed

      write(*,*)
      write(*, '(a\)')
     : ' Compute results for another r and M (y/n;cr=quit)? '
      read(*, '(a)') buf
      if (buf(1:4) .eq. '    ') go to 999
      if (buf(1:1) .eq. 'n' .or. buf(1:1) .eq. 'N') go to 999

      goto 100

  999 continue
      close(unit=nout)
      stop
      end
* -------------- END TD_DRVR --------------------------------
```

```
*---------------- BEGIN SMSIM_TD -----------------------------
      subroutine smsim_td(per, nper,
     :    prvsim, pgasim, pgvsim, prvcalc,
     :    nacc_save, acc_save, vel_save)

* Uses time-domain stochastic model to compute ground motion
*  (SMSIM = Stochastic Model Simulation)

* Dates:  05/10/95 - Written by D. M. Boore
*         05/30/95 - Modified by Basil Margaris to allow choice of window
*         06/02/95 - Further renamed (from psvsimnu) and modified.
*         06/09/95 - Get r, amag from common, not parameter list
*         06/12/95 - Removed writing of acc, vel to a file; this should
*                    be done in the front-end driver program.
*         08/08/95 - Removed dc from noise segment before going to
*                    the frequency domain (in get_acc).
*         08/08/95 - Changed things so that remove dc from noise sample before
*                    applying the window (in get_acc).
*         08/18/95 - Added subroutine Write_Params
*         10/17/95 - Added flag to Get_Params and Write_Params
*         10/17/95 - Remove dc or not from random series, depending on a flag
*                    that is passed through the include statement
*         11/14/95 - Broke out numerical recipes and subroutines used
*                    by both rv and td into separate collections
*                    of Fortran subroutines
*         12/14/95 - Assigned durex the source plus path duration
*         04/12/96 - Changed names psv, sd to prv, rd
*         04/15/96 - Changed "rd_spect" to "rd_calc"

      real acc(16400), vel(16400), avgprv(120),
     :     per(*), prvsim(*), acc_save(*), vel_save(*)
      character prvcalc*1

      include 'smsim.fi'

      pi = 4.0 * atan(1.0)
      twopi = 2.0 * pi

      do i = 1, nper
        avgprv(i) = 0.
      end do
      avgpga = 0.
      avgpgv = 0.

      iseed = -abs(seed)

      write(*,*)
      do isim = 1, nruns
         write(*,'(3(a,i3),a)')
     :    '+ Patience! Computing accelerogram # ', isim,
     :    ' of ', nruns,
     :    ' and rs at ', nper,' periods.'

         call get_acc(acc)

         call get_vel(acc, npts, dt, vel)

         if (isim .eq. nacc_save) then
           do i = 1, npts
             acc_save(i) = acc(i)
             vel_save(i) = vel(i)
           end do
         end if

* Compute pga, pgv:
         call mnmax(acc, 1, npts, 1, amin, amax)
         pga = amax1(abs(amin), abs(amax))

         call mnmax(vel, 1, npts, 1, vmin, vmax)
         pgv = amax1(abs(vmin), abs(vmax))


         avgpga = avgpga + pga
         avgpgv = avgpgv + pgv

         if (prvcalc .eq. 'y' .or. prvcalc .eq. 'Y') then
* Compute prv:
           do i = 1, nper
             omega = twopi/per(i)
             call rd_calc(acc, npts, omega, damp, dt, rd)
             prv = omega * rd
             avgprv(i) = avgprv(i) + prv
           end do                              ! iper
         end if

      end do                                 ! isim

* Compute averages
      pgasim = avgpga/float(nruns)
      pgvsim = avgpgv/float(nruns)
      if (prvcalc .eq. 'y' .or. prvcalc .eq. 'Y') then
        do i = 1, nper
          prvsim(i) = avgprv(i)/float(nruns)
        end do
      end if

      return
      end
*---------------- END SMSIM_TD -------------------------------

*---------------- BEGIN GET_ACC ------------------------------
      subroutine get_acc(acc)
* Dates:  06/07/95 - Written by D.M. Boore
*         06/08/95 - Normalize by sqrt(avg square amp)
*         06/09/95 - Get r, amag from common, not parameter list
*         08/08/95 - Removed dc from noise segment before going to
*                    the frequency domain (in get_acc).
*         08/08/95 - Changed things so that remove dc from noise sample before
*                    applying the window (in get_acc).
*         10/17/95 - Remove dc or not from random series, depending on a flag
*                    (irmvdc) that is passed through the include statement
*         11/14/95 - Added call to const_am0_gsprd
*         11/16/95 - Replaced 'shape = ...' statement with call to spect_amp
*         12/06/95 - Major simplification in exponential window, removing
*                    tapers (the window itself provides tapers) and correcting
*                    a bug so that the window can be used for a duration
*                    other than the duration used to determine the window
*                    parameters.
*         12/22/95 - Removed rmean real part of work only
*         01/22/96 - Use REALFT rather than FORK

      real work(16400)
      real acc(*)
      include 'smsim.fi'

* Set spectral parameters:
      call spect_scale()  ! call this now because need corner frequencies
*                           for window durations

* Fill an array with the proper duration of windowed Gaussian noise:
c
```

```
      do i = 1, npts
          acc(i)=0.
          work(i) = 0.
      end do

*  Modifications in order to use another window 5/30/1995

      if(indxwind .eq. 0) then        ! BOX WINDOW

* Figure out indices for a noise sample of the proper duration:
        nstart = tshift/dt
        durex = dursource(w_fa, fa, w_fb, fb)+
     :          durpath(r, nknots, rdur, dur, slast)
        nmotion = durex/dt
* Increase duration of motion by tapers (figure 1/2 front and back):
        ntaper = ifix(taper*nmotion)  ! taper is fractional, not percent
        nmotion = nmotion + ntaper
        nstop = nstart + nmotion
        if (nstop .gt. npts) then
          write(*,'(2x,a,i5,a,i5,a)')
     :       ' *** FATAL ERROR: NSTOP (',
     :                                  nstop,
     :       ') > NPTS (',
     :                                   npts,
     :       '); QUITTING!'
          stop
        end if

* Generate Gaussian white noise with zero mean, unit variance:
        do i = nstart, nstop
           work(i) = gasdev(iseed)
        end do

        if (irmvdc .eq. 0) then
          rmean = 0.0
        else
          call mean(work, nstart, nstop, rmean)
        end if

        do i = nstart, nstop
           work(i) = wind_box(i, nstart, nstop, ntaper)
     :       * (work(i) - rmean)
        end do

      else                            ! EXPONENTIAL WINDOW

* Figure out indices for a noise sample of the proper duration:
        nstart = tshift/dt
        durex = dursource(w_fa, fa, w_fb, fb)+
     :          durpath(r, nknots, rdur, dur, slast)
        nmotion = 2.0 * durex/dt   ! doubled following Boore (1983, p. 1869)
        tmotion = nmotion * dt
        nstop = nstart + nmotion
        if (nstop .gt. npts) then
          write(*,'(2x,a,i5,a,i5,a)')
     :       ' *** FATAL ERROR: NSTOP (',
     :                                  nstop,
     :       ') > NPTS (',
     :                                   npts,
     :       '); QUITTING!'
          stop
        end if

* Generate Gaussian white noise with zero mean, unit variance:
        tw = tmotion * twdtmotion
        do i = nstart, nstop
          work(i) = gasdev(iseed)
        end do

        if (irmvdc .eq. 0) then
          rmean = 0.0
        else
          call mean(work, nstart, nstop, rmean)
        end if

        do i = nstart, nstop
          t = float(i-nstart) * dt
          work(i) = wind_exp( t, tw, eps_wind, eta_wind, new_mr)
     :    * (work(i) - rmean)
        end do

      end if

* Transform to frequency domain:
      call realft(work,npts,+1)

* Find sqrt of squared spectral amplitude for normalization:
      call avgsq_realft(work,npts,avgsqamp)
      sqrt_avgsqamp = sqrt(avgsqamp)

* Get frequency independent parameter:
      call const_am0_gsprd()     ! returns freq_indep_factor through common

      iaorins = 1              ! ground motion
      idva = 2                  ! acceleration

      df = 1.0/(float(npts)*dt)
      do i = 2,npts/2
          f = (i-1) * df
          sfact = spect_amp(f)/ sqrt_avgsqamp
          work(2*i-1) = sfact * work(2*i-1)
          work(2*i)   = sfact * work(2*i)
      end do
      work(1) = 0.0               ! OK for acceleration spectrum, but not for
                                  ! displacement spectrum
      fnyq = 1.0/(2.0*dt)
      work(2) = spect_amp(fnyq) * work(2)/ sqrt_avgsqamp

* Transform back to time domain:
      call realft(work,npts,-1)

* Apply FFT normalization:
      afact = 2 * df
      do i = 1, npts
         acc(i) = afact * work(i)
      end do

      return
      end
*----------------- END GET_ACC ----------------------------

*----------------- BEGIN GET_VEL ----------------------------
* Dates: 06/07/95 - Written by D.M. Boore
*        02/02/96 - Modified to use double precision for
*                   the accumulator, following Converse's BAP routine.

      subroutine get_vel(acc, npts, dt, vel)

      real vel(*), acc(*)
      double precision cum, hdt

* remove trend first
```

```
      call dcdt(acc, dt, npts, 1, npts, .false., .true.)

* compute velocity (assume vel = 0 for first point)

      hdt = dble(dt) * dble(0.5)
      cum = 0.0
      vel(1) = sngl(cum)
      do j=2,npts
        cum = cum + dble(acc(j) + acc(j-1)) * hdt
        vel(j) = sngl(cum)
      end do

* high pass filter the velocity (may want to allow this in a future version;
* as it is, the acceleration time series can be filtered, so there is no need
* to do it again).

      return
      end
*---------------- END GET_VEL ----------------------------

*---------------- BEGIN DCDT -----------------------------
* Dates:  - Written by C.S. Mueller
      SUBROUTINE DCDT (Y,DT,NPTS,INDX1,INDX2,LDC,LDT)
c+
c  DCDT - Fits DC or trend between indices INDX1 and INDX2.
C         Then removes DC or detrends whole trace.
c         Y is real, DT = delta t.
c         If remove DC, LDC = .TRUE.
c         IF detrend, LDT = .TRUE.
c-
      real Y(1)
      logical LDC,LDT
c
c...Fit DC and trend between indices INDX1 and INDX2.
100   NSUM = INDX2-INDX1+1
      SUMX = 0.0
      SUMX2 = 0.0
      SUMY = 0.0
      SUMXY = 0.0
      DO 200 I=INDX1,INDX2
         XSUBI = (I-1)*DT
         SUMXY = SUMXY+XSUBI*Y(I)
         SUMX = SUMX+XSUBI
         SUMX2 = SUMX2+XSUBI*XSUBI
200      SUMY = SUMY+Y(I)
C
C... Remove DC.
300   IF (LDC) THEN
        AVY = SUMY/NSUM
        DO 360 I=1,NPTS
360        Y(I) = Y(I)-AVY
        RETURN
      END IF
C
C... Detrend. See Draper and Smith, p. 10.
400   IF (LDT) THEN
        BXY = (SUMXY-SUMX*SUMY/NSUM)/(SUMX2-SUMX*SUMX/NSUM)
        AXY = (SUMY-BXY*SUMX)/NSUM
        QXY = DT*BXY
        DO 450 I=1,NPTS
450        Y(I) = Y(I)-(AXY+(I-1)*QXY)
        RETURN
      END IF
C
      STOP
      END
*---------------- END DCDT -----------------------------

*---------------- BEGIN MNMAX ----------------------------
      subroutine mnmax(a,nstrt,nstop,ninc,amin,amax)
c
c author: D. M. Boore
c last change: 9/7/84
c
      dimension a(1)
      amax = a( nstrt)
      amin=amax
      do 10 i=nstrt,nstop,ninc
      if(a(i)-amax) 15,15,20
20    amax=a(i)
      go to 10
15    if(a(i)-amin) 25,10,10
25    amin=a(i)
10    continue
      return
      end
*---------------- END MNMAX ----------------------------

*---------------- BEGIN MEAN -----------------------------
      subroutine mean(work, nstart, nstop, rmean)
*  Dates: 01/22/96 - Written by D. Boore
      real work(*)
* find mean of the array:
      sum = 0.0
      do i = nstart, nstop
        sum = sum + work(i)
      end do
      rmean = sum/float(nstop-nstart+1)
      return
      end
*---------------- END MEAN -----------------------------

*------------------ BEGIN AVGSQ_REALFT -----------------------------
* Dates: 01/22/96 - Written by D.M. Boore
      subroutine avgsq_realft( s, npts, avgsqamp)
      real s(*)
      sum=0.
      do j = 2, npts/2                  !don't include the dc or Nyquist values
        sum=sum + s(2*j-1)**2 + s(2*j)**2 ! odd, even = real, imag spect
      end do
      avgsqamp = sum/float(npts/2 - 1)
      return
      end
*  ------------------- END AVGSQ_REALFT -----------------------------

*  ------------------- BEGIN WIND_BOX -----------------------------
      function wind_box( i, nstart, nstop, ntaper)
c
c applies cosine tapered window.
c unit amplitude assumed
c
c written by D. M. Boore
c
c latest revision: 9/26/95
      real wind_box
      wind_box = 0.0
      if ( i .lt. nstart .or. i. gt. nstop) return
      wind_box = 1.0
      if ( i .ge. nstart+ntaper .and. i .le. nstop-ntaper ) return
c
      pi = 4.0 * atan(1.0)
c
```

```
      dum1 = (nstop+nstart)/2.0
      dum2 = (nstop-nstart-ntaper)/2.0
c
      wind_box = 0.5 * (1.0 - sin( pi*
     *  ( abs(float(i)-dum1) - dum2 ) /float(ntaper) ) )
      return
      end
*  ------------------- END WIND_BOX -----------------------------

*  ------------------- BEGIN WIND_EXP ---------------------------
      function  wind_exp( t, tw, eps_wind, eta_wind, new_mr)
c
c     apply Sargoni and Hart (1974) window, with parameters
c     tw, eps (fraction of tw to reach peak), and
c     eta ( fraction of peak ampl. at tw).  See Boore (BSSA, 1983,
*     p. 1869).  Note that t can be larger than tw.

* Dates:
*         05/30/95 - Initially written by Dave Boore, based on a routine
*                    by G. Atkinson but with significant structural
*                    changes and the taper to be a raised cosine taper
*         12/06/95 - Removed the taper and simplified the parameters in the
*                    calling list.  Also added a switch so that the window
*                    shape coefficients are only computed once.  This assumes
*                    that the shape coefficients will be the same for
*                    any run of the program... a good assumption.
*         12/28/95 - Used new_mr, set from driver, to control computation
*                    of b,c,a; before I used "firstcall", set in this
*                    subprogram, but this gave an error if the driver
*                    looped over multiple values of m and r.

      real wind_exp
      logical new_mr
      save a, b, c

      if (new_mr) then
        b = -eps_wind * alog(eta_wind)/
     :      (1. + eps_wind*(alog(eps_wind)-1.))
        c = b/(eps_wind*tw)
        a = (exp(1.0)/(eps_wind*tw))**b
        new_mr = .false.
      end if

      wind_exp = 0.
      if( t .lt. 0.0) return
c
c     Apply Sargoni and Hart window.
c
      wind_exp = a*t**b * exp(-c*t)

      return
      end
*------------------- END WIND_EXP ---------------------------

*----------------- BEGIN RD_CALC -----------------------------
      subroutine rd_calc(acc,na,omega,damp,dt,rd)
* This is a modified version of "Quake.For", originally
* written by J.M. Roesset in 1971 and modified by
* Stavros A. Anagnostopoulos, Oct. 1986.  The formulation is that of
* Nigam and Jennings (BSSA, v. 59, 909-922, 1969).  This modification
* eliminates the computation of the relative velocity and absolute
* acceleration; it returns only the relative displacement.
* Dates: 05/06/95 - Modified by David M. Boore
*        04/15/96 - Changed name to RD_CALC and added comment lines
*                   indicating changed for storing the oscillator time series
*                   and computing the relative velocity and absolute
*                   acceleration

*   acc = acceleration time series
*    na = length of time series
* omega = 2*pi/per
*  damp = fractional damping (e.g., 0.05)
*    dt = time spacing of input
*    rd = relative displacement of oscillator

      dimension acc(*)
      omt=omega*dt
      d2=1-damp*damp
      d2=sqrt(d2)
      bom=damp*omega
*      d3 = 2.*bom                  ! for aa
      omd=omega*d2
      om2=omega*omega
      omdt=omd*dt
      c1=1./om2
      c2=2.*damp/(om2*omt)
      c3=c1+c2
      c4=1./(omega*omt)
      ss=sin(omdt)
      cc=cos(omdt)
      bomt=damp*omt
      ee=exp(-bomt)
      ss=ss*ee
      cc=cc*ee
      s1=ss/omd
      s2=s1*bom
      s3=s2+cc
      a11=s3
      a12=s1
      a21=-om2*s1
      a22=cc-s2
      s4=c4*(1.-s3)
      s5=s1*c4+c2
      b11=s3*c3-s5
      b12=-c2*s3+s5-c1
      b21=-s1+s4
      b22=-s4
      rd=0.
*      rv = 0.                              ! for rv
*      aa = 0.                              ! for aa
      n1=na-1
      y=0.
      ydot=0.
      do i=1,n1
        y1=a11*y+a12*ydot+b11*acc(i)+b12*acc(i+1)
        ydot=a21*y+a22*ydot+b21*acc(i)+b22*acc(i+1)
        y=y1    ! y is the oscillator output at time corresponding to index i
        z=abs(y)
        if (z.gt.rd) rd=z
*        z1 = abs(ydot)                     ! for rv
*        if (z1.gt.rv) rv = z1              ! for rv
*        ra = -d3*ydot -om2*y1              ! for aa
*        z2 = abs(ra)                       ! for aa
*        if (z2.gt.aa) aa = z2              ! for aa
      end do
      return
      end
*----------------- END RD_CALC -----------------------------


      include 'rvtdsubs.for'
      include 'recipes.for'
```

```
* ---------------- BEGIN FAS_DRVR --------------------------
      Program FAS_Drvr

* Obtains input parameters, calls FAS routine, and writes output.
* Write out Fourier displacement, velocity, and acceleration spectra,
* and Fourier spectra of the oscillator response, if used (max of 10 periods).

* Dates: 12/16/95 - Written by D.M. Boore; Modified from RV_DRVR
*        01/03/96 - Added oscillator response
*        04/12/96 - Changed name psv to prv
      character prvcalc*1, indvidper*1, bigbuf*150
      character fname_params*80, fname_out*80, buf*80
      character indvidfreq*1, f_col*80, message*80,
     :  date_begin*10, time_begin*11, time_start*11, time_stop*11
      logical fparam_exist, tdflag

      real freq(1000), fds(1000)
      real per(10), prv(1000, 10)

      integer  irecord_length

      include 'smsim.fi'

      pi = 4.0 * atan(1.0)
      twopi = 2.0 * pi

      write(*, '(a\)')
     :    ' Enter a message, if desired: '
      message = ' '
      read(*, '(a)') message

333   continue
      fname_params = ' '
      write(*, '(a\)')
     :    ' Enter name of file with parameters (cr to quit): '
      read(*, '(a)') fname_params
      if (fname_params(1:4) .eq. '    ') stop
      inquire(file=fname_params, exist=fparam_exist)
      if (.not. fparam_exist) then
        write(*,'(a)') ' ******* FILE DOES NOT EXIST ******* '
        go to 333
      end if

      fname_out = ' '
      write(*, '(a\)')
     :    ' Enter name of summary file: '
      read(*, '(a)') fname_out
      nout = 10
      open(unit=nout,file=fname_out,status='unknown')
      write(nout, '(a)') message
      write(nout, '(2a)') ' output file: ',
     :   fname_out(1:20)

      write(nout, '(a)')
     :  ' *** Results computed using FAS_DRVR ***'

      call get_date(date_begin)
      write(nout, '(2a)') ' Date: ', date_begin
      call get_time(time_begin)
      write(nout, '(2a)') ' Time: ', time_begin

      write(nout, '(2a)') ' file with parameters: ',
     :   fname_params(1:20)

      write(*, '(a\)')
     :  ' Enter individual frequencies(y/n)?: '
      indvidfreq = ' '
      read(*, '(a)') indvidfreq
      if (indvidfreq .eq. 'y' .or. indvidfreq .eq. 'Y') then
        write(*, '(a\)')
     :     ' Enter nfreqs: '
        read(*, *) nfreq
        do i = 1, nfreq
          write(*, '(a\)') '    Enter frequency for FAS: '
          read(*,*) freq(i)
        end do
      else
        write(*, '(a\)')
     :       ' Enter freqlow, freqhigh, nfreq (freq .ne. 0!): '
        read(*,*) freqlow, freqhigh, nfreq
        if (freqlow .eq. 0.0 .or. freqhigh .eq. 0.0) then
          write(*, '(a)')
     :     ' ******* FREQ = 0 NOT ALLOWED; ABORTING!! *******'
          go to 999
        end if
        do i = 1, nfreq
         freq(i) =
     :    freqlow * (freqhigh/freqlow)**(float(i-1)/float(nfreq-1))
        end do
      end if

      write(*, '(a\)') ' Compute prv (y/n): '
      prvcalc = 'n'
      read(*, '(a)') prvcalc
      if (prvcalc .eq. 'y' .or. prvcalc .eq. 'Y') then
        write(*, '(a\)')
     :  ' Enter individual periods(y/n)?: '
        indvidper = ' '
        read(*, '(a)') indvidper
        if (indvidper .eq. 'y' .or. indvidper .eq. 'Y') then
          write(*, '(a\)')
     :       ' Enter fractional damping, nperiods (max of 10): '
          read(*, *) damp, nper
          do i = 1, nper
            write(*, '(a\)') '    Enter oscillator period: '
            read(*,*) per(i)
          end do
        else
          write(*, '(2a\)')
     :     ' Enter fractional damping, perlow, perhigh,',
     :     ' nper (max of 10): '
          read(*,*) damp, perlow, perhigh, nper
          do i = 1, nper
            per(i) =
     :          perlow * (perhigh/perlow)**(float(i-1)/float(nper-1))
          end do
        end if
        if (nper .gt. 10) then
          write(*, '(a,i4,a)') ' *** ERROR ****** nper = ',
     :       nper, ' > 10; QUITTING!!!!' ! should check this when obtain
                                         ! nper and force the user to enter
                                         ! a valid number.  Perhaps this
                                         ! will be done in a future version.
          close(unit=nout)
          stop
        end if
      else
        nper = 0
      end if

      tdflag = .false.
      call get_params( fname_params, tdflag )
```

```
      call write_params( nout, tdflag)

100   continue

      buf = ' '
      write(*, '(a\)') ' dist (cr to quit): '
      read(*, '(a)') buf
      if (buf(1:4) .eq. '    ') go to 999
      read(buf, '(f10.0)') r

      write(*, '(a\)') ' amag: '
      read(*, *) amag

      write(*, '(a\)')
     :     ' Enter name of column file for FAS results: '
      f_col = ' '
      read( *, '(a)') f_col

* Figure out record length and open the file:
      irecord_length = 10*(nper + 5)
      ncol = 50
      open(unit=ncol, file=f_col,
     :     recl = irecord_length, status='unknown')

* Write the header:
      bigbuf = ' '
      bigbuf(3:10) = 'freq(Hz)'
      bigbuf(15:20) = 'per(s)'
      bigbuf(22:30) = 'fds(cm-s)'
      bigbuf(34:40) = 'fvs(cm)'
      bigbuf(42:50) = 'fas(cm/s)'
      if (nper .gt. 0) then
        do iper = 1, nper
          iseg = 10*(iper + 4)
          write(bigbuf(iseg+3:iseg+10),'(a,f6.3)')
     :          'T=', per(iper)
        end do
      end if
      write(ncol, '(a)') bigbuf(1:irecord_length)

      call get_time(time_start)

      iaorins = 1
      idva = 0
      write(*,'(a)')
     : '  Patience! Computing displacement spectrum'
      call smsimfas(fds, freq, nfreq)

      if (prvcalc .eq. 'y' .or. prvcalc .eq. 'Y') then
        iaorins = 2
        idva = 0
        write(*,*)
        do iper = 1, nper
          write(*,'(a,i3,a,i3,a)')
     :    '+ Patience! Computing response spectra number ',
     :     iper, ' out of ', nper, ' periods.'
          perosc = per(iper)
          fosc = 1.0/perosc
          call smsimfas(prv(1,iper), freq, nfreq)
        end do
      end if

      write(nout,*)
      write(nout, '(a)') ' ********** NEW R AND M **********'
      write(nout, '(a, 1p2(1x,e10.3))')
     :     ' r, amag = ', r, amag
      write(nout, '(2x,2a)') ' Time Start: ', time_start
      write( nout, '(2x,a,1x,1pe10.3)') ' const= ', const
      write( nout, '(2x,a,1p4(1x,e10.3))')
     :  ' amag, stress, fa, fb= ', amag, stress, fa, fb
      write( nout, '(2x,a,1p2(1x,e10.3))') ' am0, am0b_m0fa= ',
     :                                     am0, am0b_m0

      write(*,*)

      do i = 1, nfreq
        per4freq = 1.0e10
        if (freq(i) .ne. 0.0) per4freq = 1.0/freq(i)
        fvs = (2.0*pi*freq(i)) * fds(i)
        fas = (2.0*pi*freq(i)) * fvs
        bigbuf = ' '
        write(bigbuf(1:50), '(1p5(e10.3))')
     :      freq(i), per4freq, fds(i), fvs, fas

        if (nper .gt. 0) then
          do iper = 1, nper
            iseg = 10*(iper + 4)
            write(bigbuf(iseg+1:iseg+10),'(1pe10.3)')
     :          prv(i,iper)
          end do
        end if

        write(ncol, '(a)') bigbuf(1:irecord_length)
      end do
      close(unit=ncol)

      call get_time(time_stop)
      write(nout, '(2x,2a)') ' Time Stop: ', time_stop
      call time_diff(time_start, time_stop, time_elapsed)
      write(nout, '(2x,a,1x,f6.1)')
     :     ' Elapsed time (sec): ', time_elapsed

      write(*,*)
      write(*, '(a\)')
     : ' Compute results for another r and M (y/n;cr=quit)? '
      read(*, '(a)') buf
      if (buf(1:4) .eq. '    ') go to 999
      if (buf(1:1) .eq. 'n' .or. buf(1:1) .eq. 'N') go to 999

      goto 100

999   continue
      close(unit=nout)
      stop
      end
* ---------------- END FAS_DRVR -------------------------
```

```
*----------------- BEGIN SMSIMFAS -----------------------------
      subroutine smsimfas(fas, freq, nfreq)
* Dates: 12/16/95 - Written by D.M. Boore, based on SMSIM_RV

      real fas(*), freq(*)
      include 'smsim.fi'

      pi = 4.0*atan(1.0)
      twopi = 2.0 * pi

* Set spectral parameters:
      call spect_scale()   ! Be sure to call Get_Params in driver

* Get frequency-independent factor:
      call const_am0_gsprd()

      do i = 1, nfreq
        fas(i) = spect_amp(freq(i))
      end do

      return
      end
*----------------- END SMSIMFAS -----------------------------

      include 'rvtdsubs.for'
      include 'recipes.for'
```

```
* start of smsim.fi (inserted into code via 'include smsim.fi')
      real rho, beta, prtitn, rtp, fs, const, fbdfa, amagc
      real w_fa, w_fb, pi, twopi
      real fr1, qr1, s1, ft1, ft2, fr2, qr2, s2, slast
      real rlow(10), slope(10)
      real rdur(10), dur(10), famp(20), amp(20)
      real tsimdur, dt, tshift, taper, seed
      integer nknots, numsource, nsprd_segs, namps, nruns, npts
      integer indxwind
      character title*70
      logical new_mr

      common /title/ title
      common /magdist/ r, amag
      common /const_params/ rho, beta, prtitn, rtp, fs, pi, twopi,
     :                      const, iaorins, idva, freq_indep_factor
      common /source_shape_params/ numsource, pf, pd
      common /source_scale_params/ stressc, dlsdm, amagc,
     :                             stress, fa, fb, am0, am0b_m0, fbdfa
      common /gsprd_params/ nsprd_segs, rlow, slope
      common /q_params/ fr1, qr1, s1, ft1, ft2, fr2, qr2, s2
      common /source_dur_params/ w_fa, w_fb
      common /path_dur_params/ nknots, rdur, dur, slast
      common /site_amp_params/ namps, famp, amp
      common /site_dimin_params/ fm, akappa
      common /rs/ perosc, fosc, damp
      common /rv/ fup, imom, zup, eps_int, amp_cutoff,
     :            eps_rv, ane, anz,
     :            pk_rms_cl_1, pk_rms_cl_2,
     :            pk_cl_eq68, pk_rms_cl_eq68,
     :            pk_cl_1, pk_cl_2
      common /lowcut_filter_params/ fcut, norder
      common /wind_params/indxwind,taper,twdtmotion,eps_wind,eta_wind
      common /tdsim_params/tsimdur,dt,tshift,seed,
     :                     nruns, npts, iseed, irmvdc
      common /misc/freq20, durex, new_mr
      save /title/, /const_params/
      save /source_shape_params/, /source_scale_params/
      save /gsprd_params/, /q_params/
      save /source_dur_params/, /path_dur_params/
      save /site_amp_params/, /site_dimin_params/
      save /rs/
      save /rv/
      save /process_params/
      save /wind_params/, /tdsim_params/
      save /misc/
* end of smsim.fi (inserted into code via "include smsim.fi"
```

```
*----------------- BEGIN GET_PARAMS ----------------------------
      subroutine get_params(fname_params, tdflag)

* Dates: 06/07/95 - Written by D.M. Boore
*        10/17/95 - Added tdflag to control if more data need be read for
*                   the time series processing; also added irmvdc as an input
*                   parameter
*        11/14/95 - Read eps_int from data file
*        11/15/95 - changed input of spectral parameters
*        11/16/95 - Read amp_cutoff from data file and determine fup
*        11/27/95 - changed input of spectral parameters again
*        12/06/95 - Get low-cut filter parameters before rv and td params
*        12/07/95 - Get taper with window parameters
*        12/19/95 - Added zup to input parameters

      character fname_params*(*)
      logical tdflag

      include 'smsim.fi'

      pi = 4.0 * atan(1.0)

      nin = 98
      open(unit=nin, file=fname_params, status='unknown')

* title:
      read(nin, '(a)') title

* rho, beta, prtitn, fs:
      call skip(nin, 1)
      read(nin, *) rho, beta, prtitn, rtp, fs

c     Define constant, for r=1 km
      const=prtitn*rtp*fs*(1.e-20)/(4.*pi*rho*beta**3)

* source shape:
      call skip(nin, 4)
      read(nin, *) numsource, pf, pd

* source scaling: (note: stress is now read in here; to write a driver
* that loops over stress, just assign the desired values of stress
* after calling get_params)
      call skip(nin, 4)
      read(nin, *) stressc, dlsdm, fbdfa, amagc

* gsprd:
      call skip(nin, 1)
      read(nin, *) nsprd_segs
      do i = 1, nsprd_segs
        read(nin, *) rlow(i), slope(i)
      end do
*  assume that rlow(1) = 1, for proper scaling when construct gsprd.
*  Check this and quit program if it isn't:
      if ( rlow(1) .ne. 1.0) then
         write(*, '(a, e10.3, a)')
     :  ' !!!!! rlow(1) = ', rlow(1), ', .ne. 1.0; quitting!!!!!'
         stop
      end if

* Q:
      call skip(nin, 1)
      read(nin, *) fr1, qr1, s1, ft1, ft2, fr2, qr2, s2

* source duration:
      call skip(nin, 1)
      read(nin, *) w_fa, w_fb


* path duration:
      call skip(nin, 1)
      read(nin, *) nknots
      do i = 1, nknots
        read(nin, *) rdur(i), dur(i)
      end do
      read(nin, *) slast

* site amps:
      call skip(nin, 1)
      read(nin, *) namps
      do i = 1, namps
        read(nin, *) famp(i), amp(i)
      end do

* site diminution:
      call skip(nin, 1)
      read(nin, *) fm, akappa

* low-cut filter parameters:
      call skip(nin, 1)
      read(nin, *) fcut, norder

* parameters for rv integration:
      call skip(nin, 1)
      read(nin, *) zup, eps_int, amp_cutoff
      if( akappa .eq. 0.0) then
        fup = fm/amp_cutoff**0.25
      else
        fup =
     :   amin1(fm/amp_cutoff**0.25, -alog(amp_cutoff)/(pi*akappa))
      end if

* Read more if time domain method:
      if (.not. tdflag) goto 999

* window parameters:
      call skip(nin, 1)
      read(nin, *) indxwind, taper, twdtmotion, eps_wind, eta_wind

* timing stuff:
      call skip(nin, 1)
      read(nin, *) tsimdur, dt, tshift, seed, nruns

* Flag controlling the removal of the dc from the random series:
      call skip(nin, 1)
      read(nin, *) irmvdc

* compute smallest power of two for which the resulting duration is greater
* than or equal to tsimdur:
      npts = 2.0**ifix(alog10(tsimdur/dt + 1.0)/alog10(2.0))
      if ( (npts-1)*dt .lt. tsimdur) npts = 2 * npts

999   continue
      close(unit=nin)

      return
      end
*----------------- END GET_PARAMS ----------------------------

*----------------- BEGIN WRITE_PARAMS ----------------------------
      subroutine write_params(nout, tdflag)
```

```
* Dates: 08/18/95 - Written by D. Boore
*        10/17/95 - Added tdflag to control if more data need be read for
*                   the time series processing; also added irmvdc as an input
*                   parameter
*        11/14/95 - Write eps_int
*        11/16/95 - Write amp_cutoff, fup
*        12/06/95 - Write low-cut filter params before rv and td params
*        12/07/95 - Write taper with window parameters
*        12/19/95 - Added zup to input parameters

      logical tdflag

      include 'smsim.fi'

      write( nout, '(a)') ' Title:'
      write( nout, '(4x,a)') title

      write( nout, '(a)') ' rho, beta, prtitn, rtp, fs:'
      write( nout,   *  )   rho, beta, prtitn, rtp, fs

      write( nout, '(2a/2a/a/a)')
     : ' spectral shape: source number (1=Single Corner;2=Joyner;',
     : '3=A93;4=custom)',
     : ' pf, pd (1-corner spectrum = ',
     : '1/(1+(f/fc)**pf)**pd; 0.0 otherwise)',
     : ' (usual model: pf=2.0,pd=1.0; Butterworth: pf=4.0,pd=0.5)',
     : ' (Note: power of high freq decay --> pf*pd)'
      write( nout, *) numsource, pf, pd

      write( nout, '(a/a/a/a)')
     : ' spectral scaling: stressc, dlsdm, fbdfa, amagc',
     : ' (stress=stressc*10.0**(dlsdm*(amag-amagc))',
     : ' (fbdfa, amagc for Joyner model, usually 4.0, 7.0)',
     : ' (not used for srce 3, but placeholders still needed)'
      write( nout, *) stressc, dlsdm, fbdfa, amagc

      write( nout, '(2a)') ' gsprd: nsegs, (rlow(i), slope(i))',
     :                     '  (Set rlow(1) = 1.0)'
      write( nout, *) nsprd_segs
      do i = 1, nsprd_segs
        write( nout, *) rlow(i), slope(i)
      end do

      write( nout, '(a)') ' q: fr1, Qr1, s1, ft1, ft2, fr2, qr2, s2'
      write( nout, *) fr1, qr1, s1, ft1, ft2, fr2, qr2, s2

      write( nout, '(a)') ' source duration: weights of 1/fa, 1/fb'
      write( nout, *) w_fa, w_fb

      write( nout, '(2a)') ' path duration: nknots, (rdur(i), dur(i),',
     :                     ' slope of last segment'
      write( nout, *) nknots
      do i = 1, nknots
        write( nout, *) rdur(i), dur(i)
      end do
      write( nout, *) slast

      write( nout, '(2a)') ' site amplification: namps, (famp(i), ',
     :                     'amp(i))'
      write( nout, *) namps
      do i = 1, namps
        write( nout, *) famp(i), amp(i)
      end do

      write( nout, '(a)') ' site diminution parameters: fm, akappa'
      write( nout, *) fm, akappa

      write( nout, '(a)') ' low-cut filter parameters: fcut, norder'
      write( nout, *) fcut, norder

      if (.not. tdflag) then
        write( nout, '(a)')
     :  ' parameters for rv integration: zup, eps_int, amp_cutoff'
        write( nout, *) zup, eps_int, amp_cutoff
        write(nout, '(a,1pe10.3)') ' calculated fup = ', fup
      end if

* Write more if time domain method:
      if (.not. tdflag) goto 999

      write( nout, '(2a)')
     :  ' window params: indxwind(0=box,1=exp), ',
     :  ' taper, twdtmotion, eps_wind, eta_wind'
      write( nout, *) indxwind, taper, twdtmotion, eps_wind, eta_wind

      write( nout, '(a)')
     :  ' timing stuff: tsimdur, dt, tshift, seed, nruns'
      write( nout, *) tsimdur, dt, tshift, seed, nruns

      write( nout, '(2a/2a)')
     :         ' parameter to control whether dc is',
     :         ' removed from random series before',
     :         ' transformation to frequency domain',
     :         ' (0=no, 1=yes)'
      write( nout, *) irmvdc

999   continue
      return
      end
*----------------- END WRITE_PARAMS ----------------------------

*----------------- BEGIN SPECT_AMP -----------------------------
* Dates: 06/07/95 - Written by D.M. Boore
      function spect_amp(f)

      include 'smsim.fi'

      tempf = 1.0
      if ( idva .ne. 0 ) tempf = (twopi*f)**float(idva)
      spect_amp =   freq_indep_factor * ( tempf ) *
     :      buttrlcf(f, fcut, norder) *
     :      spect_shape(f, fa, fb, pf, pd, am0b_m0, numsource) *
     :      site_amp_factor(f, namps, famp, amp) *
     :      dimin(f)
                                ! Could save some multiplications
                                ! by removing freq_indep_factor outside the
                                ! function, but at the cost of possible
                                ! confusion.

      go to ( 301, 302, 303 ), iaorins
c no instrument response.
301   h=1
      go to 300
c prv
302   v = twopi * fosc
      h = v * harmoscf( f, fosc, damp, idva )
      go to 300
* for customized response (network instruments, etc)
303   h=1
      go to 300
300   continue
      spect_amp = h * spect_amp   ! spect_amp contains the spectral amplitude.
```

```
      return
      end
*---------------- END SPECT_AMP ----------------------------

*---------------- BEGIN CONST_AM0_GSPRD ----------------------
* Dates: 11/14/95 - Written by D.M. Boore
      subroutine const_am0_gsprd()

      include 'smsim.fi'

      freq_indep_factor = const*am0*gsprd(r, nsprd_segs, rlow, slope)
*                         (const from Get_Params, am0 from Spect_Scale)

      return
      end
*---------------- END CONST_AM0_GSPRD ----------------------

*---------------- BEGIN GSPRD ----------------------
* Dates: 06/07/95 - Written by D.M. Boore
      function gsprd(r, nsprd_segs, rlow, slope)
      real rlow(*), slope(*), geff(10)
      geff(1) = 1.0/rlow(1)  ! assume rlow(1) = 1.0
      do i = 2, nsprd_segs
        geff(i) = geff(i-1)*(rlow(i)/rlow(i-1))**slope(i-1)
      end do
      if (r .le. rlow(1)) then
        j = 1
      else if (r .ge. rlow(nsprd_segs)) then
        j = nsprd_segs
      else
        call locate(rlow, nsprd_segs, r, j)
      end if
      gsprd = (geff(j)) * (r/rlow(j))**slope(j)

      return
      end
*---------------- END GSPRD ----------------------------

*  ------------------ BEGIN BUTTRLCF -------------------------------
* Dates: 06/07/95 - Written by D.M. Boore
      function buttrlcf( f, fcut, norder)
c
c Computes the response of an norder, bidirectional
* high-pass Butterworth filter.  This is the filter
* used by the AGRAM processing (the equation was
* provided by April Converse).

* Modification: 3/27/89 - created by modifying HiPassF

      real buttrlcf
      buttrlcf = 1.0
      if ( fcut.eq.0.0 ) return

      buttrlcf = 0.0

      if ( f .eq. 0.0) return

      buttrlcf = 1.0/ (1.0+(fcut/f)**(2.0*norder))

      return
      end
*  ------------------ END BUTTRLCF -------------------------------

*---------------- BEGIN SPECT_SHAPE ----------------------------
* Dates: 06/07/95 - Written by D.M. Boore
*        11/15/95 - changed source types
      function spect_shape(f, fa, fb, pf, pd, am0b_m0, numsource)
      real spect_shape
      goto (1, 2, 3, 4), numsource

      write(*, '(a, i5, a)') ' !!!!!! numsource = ',
     : numsource, ', .ne. legal value; quitting !!!!!!'
      stop

* Single corner frequency:
1     sb = 1.0
      sa = 1.0/( 1.0 + (f/fa)**pf )**pd
      go to 900

* Joyner model
2     sb = 1.0/ ( 1.0 + (f/fb)**2 )**0.25
      sa = 1.0/ ( 1.0 + (f/fa)**2 )**0.75
      go to 900

* Atkinson 1993 model
3     sb = 1.0
      sa = (1.0 - am0b_m0)/( 1.0 + (f/fa)**2 )
     :      +      (am0b_m0)/( 1.0 + (f/fb)**2 )
      go to 900

* For customized relation:
4     sb = 1.0
      sa = 1.0
      go to 900

900   continue
      spect_shape = sa*sb

      return
      end
*---------------- END SPECT_SHAPE ----------------------------

*---------------- BEGIN SPECT_SCALE ----------------------------
* Dates: 06/07/95 - Written by D.M. Boore
      subroutine spect_scale()

      include 'smsim.fi'

      am0 = 10.**(1.5*amag + 16.05)
      am0b_m0 = 0.0

      goto (1, 2, 3, 4), numsource

      write(*, '(a, i5, a)') ' !!!!!! numsource = ',
     : numsource, ', .ne. legal value; quitting !!!!!!'
      stop

* Single corner frequency:
1     stress = stressc*10.0**(dlsdm*(amag-amagc))
      fa = (4.906e+06) * beta * (stress/am0)**(1.0/3.0)
      fb = fa
      return

* Joyner scaling:
2     am0c = 10.0 ** ( 1.5*amagc + 16.05 )
      stress = stressc*10.0**(dlsdm*(amag-amagc))
      rat = stress/am0
      dum = 4.906e+06
      if ( am0 .gt. am0c ) rat = stress/am0c
      fb = ( dum*beta ) * ( fbdfa )**(3./4.) * ( rat )**(1./3.)
      fa = ( dum*beta ) * (stress)**(1./3.) * (am0c)**(1./6.)
```

```
     *     * ( fbdfa )**(-0.25) * ( am0 )**(-0.5)
      if ( am0 .lt. am0c ) fa = fb / fbdfa
      return

* Atkinson 93 scaling:
3     fa = 10.0**(2.41 - 0.533 * amag)
      fb = 10.0**(1.43 - 0.188 * amag)
      am0b_m0 = 10.0**(2.52 - 0.637 * amag)
      return

* For customized scaling:
4     fa = 0.0
      fb = 0.0
      return

      end
*---------------- END SPECT_SCALE ----------------------------

*---------------- BEGIN SITE_AMP_FACTOR ----------------------------
* Dates: 06/07/95 - Written by D.M. Boore
      function site_amp_factor(f, namps, famp, amp)
      real famp(*), amp(*), site_amp_factor

      if ( f .le. famp(1) ) then
        site_amp_factor = amp(1)
      else if ( f .ge. famp(namps) ) then
        site_amp_factor = amp(namps)
      else
        call locate( famp, namps, f, j)
        site_amp_factor = amp(j)*10.0**(alog10(f/famp(j))
     :             *alog10(amp(j+1)/amp(j))
     :             /alog10(famp(j+1)/famp(j)))
       end if

       return
       end
*---------------- END SITE_AMP_FACTOR ----------------------------

*---------------- BEGIN DIMIN ----------------------------
* Dates: 06/07/95 - Written by D.M. Boore
      function dimin(f)
      real dimin
      include 'smsim.fi'

      akappaq = r/(beta*q(f))

      dimin = exp( -pi*(akappa + akappaq) * f)/
     :   sqrt( 1. + (f/fm)**8.0 )


      return
      end
*---------------- END DIMIN ----------------------------

*---------------- BEGIN Q ----------------------------
* Dates: 06/07/95 - Written by D.M. Boore
*        12/14/95 - Added check for equal transition frequencies
      function q(f)
      logical firstcall / .true. /
      save qt1, qt2, st
      include 'smsim.fi'

      q = 9999.0
      if (f .eq. 0.0) return

      if (firstcall) then
        qt1 = qr1*(ft1/fr1)**s1
        qt2 = qr2*(ft2/fr2)**s2
        st = 0.0
        if (ft1 .ne. ft2) then
          st = alog10(qt2/qt1)/alog10(ft2/ft1)
        end if
        firstcall = .false.
      end if

      if ( f .le. ft1) then
        q = qr1*(f/fr1)**s1
      else if ( f .ge. ft2) then
        q = qr2*(f/fr2)**s2
      else
        q = qt1*(f/ft1)**st
      end if

      return
      end
*---------------- END Q ----------------------------

*---------------- BEGIN HARMOSCF ----------------------------
      function harmoscf( f, fosc, damp, idva)
c harmonic oscillator response to ground
c displacement.
* idva = 0 for response to displacement
* idva = 2 for response to acceleration
* idva = 1 returns 0 for the response
* The response is normalized to be unity in the flat portion.

* Written by D. Boore, 12/01/83
c latest modification:  3/25/89

      if (idva .ne. 0 .and. idva .ne. 2) dum = 0.0
      if (idva .eq. 0) dum = f**2
      if (idva .eq. 2) dum = fosc**2

      harmoscf = dum / sqrt( ( fosc*fosc - f*f )**2
     * + ( 2.0*f*fosc*damp )**2 )
      return
      end
*---------------- END HARMOSCF ----------------------------

*---------------- BEGIN DURSOURCE ----------------------------
* Dates: 06/07/95 - Written by D.M. Boore
      function dursource(w_fa, fa, w_fb, fb)
      real dursource
      dursource = w_fa/fa + w_fb/fb
      return
      end
*---------------- END DURSOURCE ----------------------------

*---------------- BEGIN DURPATH ----------------------------
* Dates: 06/07/95 - Written by D.M. Boore
      function durpath(r, nknots, rdur, dur, slast)
      real rdur(*), dur(*), durpath

      if ( r .ge. rdur(nknots) ) then
        durpath = (r -rdur(nknots))*slast + dur(nknots)
      else
        call locate(rdur, nknots, r, j)
        durpath = (r - rdur(j))*( dur(j+1)-dur(j))
     :            /(rdur(j+1)-rdur(j)) + dur(j)
      end if

      return
```

```
      end
*---------------- END DURPATH ----------------------------

*---------------- BEGIN SKIP -----------------------------
      subroutine SKIP(lunit, nlines)
        do i = 1, nlines
           read(lunit, *)
        end do
        return
      end
*---------------- END SKIP -------------------------------

*--------------------- BEGIN GET_DATE  ---------------------
      subroutine get_date(date_c)

      character date_c*(*)
      integer*2 iyr, imon, iday

* Microsoft compiler version:
*      call GETDAT (iyr, imon, iday)

*      date_c = ' '         ! character date_c*10

*      date_c(3:3) = '/'
*      date_c(6:6) = '/'

*      write(date_c(1:2),'(i2.2)') imon
*      write(date_c(4:5),'(i2.2)') iday
*      write(date_c(7:10),'(i4.4)') iyr

* Lahey compiler version:
      call DATE(date_c)   ! character date_c*8, but *10 is OK

      return
      end
*--------------------- END GET_DATE  ---------------------

*--------------------- BEGIN GET_TIME  ---------------------
      subroutine get_time(time_c)

      character time_c*(*)
      integer*2 ihr, imin, isec, i100th

* Microsoft compiler version:
*      call GETTIM (ihr, imin, isec, i100th)

*      time_c = ' '             ! character time_c*11 for both compilers

*      time_c(3:3) = ':'
*      time_c(6:6) = ':'
*      time_c(9:9) = '.'

*      write(time_c(1:2),'(i2.2)') ihr
*      write(time_c(4:5),'(i2.2)') imin
*      write(time_c(7:8),'(i2.2)') isec
*      write(time_c(10:11),'(i2.2)') i100th

* Lahey compiler version:
      call TIME(time_c)

      return
      end
*--------------------- END GET_TIME  ---------------------

* --------------- BEGIN TIME_DIFF --------------------------------
* Dates: 06/07/95 - Written by D.M. Boore
      subroutine time_diff(time_start, time_stop, time_elapsed)
      character time_start*(*), time_stop*(*)
      read(time_start(1:11),'(i2,1x,i2,1x,i2,1x,i2)')
     :                      ihb, imb, isb, ihsb
      read(time_stop(1:11),'(i2,1x,i2,1x,i2,1x,i2)')
     :                      ihe, ime, ise, ihse
      tbegin = 3600.0*ihb + 60.0*imb + isb + float(ihsb)/100.0
      tend   = 3600.0*ihe + 60.0*ime + ise + float(ihse)/100.0
      time_elapsed = tend - tbegin
      return
      end
* --------------- END TIME_DIFF ----------------------------------
```

```
*---------------- BEGIN QMIDPNT -----------------------------
      SUBROUTINE qmidpnt(func,a,b,s)
* Dates: 06/09/95 - qtrap, with midpnt substituted for trapzd.

* The user must obtain a copy of the Numerical Recipes Programs
* from the publisher (see the text for ordering information)
*---------------- END QMIDPNT -------------------------------

*---------------- BEGIN MIDPNT ------------------------------
      SUBROUTINE midpnt(func,a,b,s,n)
* The user must obtain a copy of the Numerical Recipes Programs
* from the publisher (see the text for ordering information)
*---------------- END MIDPNT --------------------------------

*---------------- BEGIN LOCATE ------------------------------
      SUBROUTINE locate(xx,n,x,j)
* The user must obtain a copy of the Numerical Recipes Programs
* from the publisher (see the text for ordering information)
*---------------- END LOCATE --------------------------------

* --------------- BEGIN ODEINT ------------------------------
      SUBROUTINE odeint(ystart,nvar,x1,x2,eps,h1,hmin,nok,nbad,derivs)

*  6/01/95 - D. Boore removed rkqs from the list of calling arguments
*            and from the external statement.

* The user must obtain a copy of the Numerical Recipes Programs
* from the publisher (see the text for ordering information)
* --------------- END ODEINT --------------------------------

* --------------- BEGIN RKQS --------------------------------
      SUBROUTINE rkqs(y,dydx,n,x,htry,eps,yscal,hdid,hnext,derivs)
* The user must obtain a copy of the Numerical Recipes Programs
* from the publisher (see the text for ordering information)
* --------------- END RKQS ----------------------------------

* --------------- BEGIN RKCK --------------------------------
      SUBROUTINE rkck(y,dydx,n,x,h,yout,yerr,derivs)
* The user must obtain a copy of the Numerical Recipes Programs
* from the publisher (see the text for ordering information)
* --------------- END RKCK ----------------------------------

*  ------------------ BEGIN GASDEV -------------------------
      FUNCTION gasdev(idum)
* The user must obtain a copy of the Numerical Recipes Programs
* from the publisher (see the text for ordering information)
*  ------------------ END GASDEV ---------------------------

*  ------------------ BEGIN RAN1 ---------------------------
      FUNCTION ran1(idum)
* The user must obtain a copy of the Numerical Recipes Programs
* from the publisher (see the text for ordering information)
*  ------------------ END RAN1 -----------------------------

*  ------------------ BEGIN REALFT -------------------------
      FUNCTION realft(data,n,isign)
* The user must obtain a copy of the Numerical Recipes Programs
* from the publisher (see the text for ordering information)
*  ------------------ END REALFT ---------------------------

*  ------------------ BEGIN FOUR1 --------------------------
      FUNCTION four1(data,n,isign)
* The user must obtain a copy of the Numerical Recipes Programs
* from the publisher (see the text for ordering information)
*  ------------------ END FOUR1 ----------------------------
```

```
      Program Site_Amp

* Convert a velocity and density model into site amplifications,
* using the square-root of the impedance.

* NOTE: if want program to assign density, enter 0.0 in third column; in this
* case also note that velocities must be in km/s.  If want to use some other
* units, the densities must all be assigned.

* For a future version:
*  calculate amplification at specified frequencies

* Dates:  10/30/95 - Written by D.M. Boore
*         11/21/95 - Changed column labels to help identify results
*                    when making plot of several amplifications and removed
*                    debug write statements.
*         12/05/95 - Added new density routine
*         12/08/95 - Added possibility of obtaining coefficients for
*                    an equation relating density and velocity.
*         12/12/95 - Modified output format
*         12/18/95 - Included tt and constant velocities in output
*         04/15/96 - Added Inquire to asking for input file

      real depth(600), vel(600), dens(600),
     :     slope(600), tt(600), d2b(0:600), avgdens(600),
     :     column1(600), column2(600), column3(600)

      real vel_layer(600), dens_layer(600), thick_layer(600)

      real depth_out(0:600), tt_out(0:600), avgvel_out(600),
     :     avgdens_out(600), freq_out(600), ampli_out(600)

      character f_in*80, f_out*80, constant_vel*1, dtbot*1,
     :          answer*1, outbuf*140, col7*12, col8*12, tempbuf*8

      logical specify_dens_coeff, f_in_exist

666   continue

* Get name of input file and open the file:

      write(*,*)

      f_in_exist = .false.
      do while (.not. f_in_exist)
        f_in = ' '
        write(*, '(a\)')
     :       ' Enter name of input file (cr to quit): '
        read(*, '(a)') f_in
        if (f_in(1:4) .eq. '    ') stop
        inquire(file=f_in, exist=f_in_exist)
        if (.not. f_in_exist) then
          write(*,'(a)') ' ******* FILE DOES NOT EXIST ******* '
        end if
      end do

      nu_in = 10
      open(unit=nu_in, file=f_in, status='unknown')

* Get name of output file and open the file:

      write(*, '(a\)') ' Enter name of output file (cr=end):'
      f_out = ' '
      read(*, '(a)') f_out
      if(f_out(1:3) .eq. '   ') goto 999

      nu_out = 50
      open(unit=nu_out, file=f_out, status='unknown')

* Find out if want to input density coefficients:

      write(*, '(a\)')
     :  ' Do you want to specify density coefficients (y,n)? '
      tempbuf = ' '
      read(*, '(a)') tempbuf
      specify_dens_coeff = .false.
      if (tempbuf(1:1) .eq. 'y' .or. tempbuf(1:1) .eq. 'Y') then
        specify_dens_coeff = .true.
        write(*, '(a\)') '    Enter dens_low, vel_low: '
        read(*, *) dens_low, vel_low
        write(*, '(a\)') '    Enter dens_high, vel_high: '
        read(*, *) dens_high, vel_high
      end if

* Skip column headers:
      read(nu_in,*)

      nentries = 0
100   continue
      nentries = nentries + 1
      read(nu_in, *, end=200)
     :    column1(nentries), column2(nentries), column3(nentries)
      if(column2(nentries) .eq. 0.0) then      ! check for the last layer
        goto 200
      else
        goto 100
      endif

200   continue
      nentries = nentries - 1

      close(unit=nu_in)

* Find out something about the type of velocity model:
      constant_vel = 'y'
      if (column1(1) .eq. 0.0) then
        write(*, '(a/a\)')
     :  '   The model appears to be piecewise continuous;',
     :  '      if so, press ENTER; if not, press n: '
        answer = ' '
        read(*,'(a)') answer
        if (answer .eq. ' ') constant_vel = 'n'
      end if
      if (constant_vel .eq. 'y' .or. constant_vel .eq. 'Y') then
        dtbot = 'y'
        do i = 2, nentries
          if (column1(i) .le. column1(i-1)) then
            write(*, '(a/a\)')
     :       '   The first column appears to be layer thickness;',
     :       '      if so, press ENTER; if not, press n: '
            answer = ' '
            read(*,'(a)') answer
            if (answer .eq. ' ') dtbot = 'n'
            goto 444
          endif
        end do
        write(*, '(a/a\)')
     :       '   The first column appears to be depth-to-bottom;',
     :       '      if so, press ENTER; if not, press n: '
        answer = ' '
        read(*,'(a)') answer
        if (answer .eq. 'n' .or. answer .eq. 'N') dtbot = 'n'
```

```
444     continue
      end if

* If constant velocity layers, convert into a velocity-depth function (with
* repeated depths at interfaces) so that it is in the same format as a
* piecewise continuous function.

      if (constant_vel .eq. 'y' .or. constant_vel .eq. 'Y') then
        nlayers = nentries
        idepth = 0
        d2b(0) = 0.0
        do ilayer = 1, nlayers
* first fill in depth-to-bottom array:
          if (dtbot .eq. 'y' .or. dtbot .eq. 'Y') then
            d2b(ilayer) = column1(ilayer)
          else
            d2b(ilayer) = d2b(ilayer-1) + column1(ilayer)
          end if
* now assign array values for the interfaces above and below each layer:
*   first above:
          idepth = idepth + 1
          depth(idepth) = d2b(ilayer-1)
          vel(idepth) = column2(ilayer)
          dens(idepth) = column3(ilayer)
*   now below:
          idepth = idepth + 1
          depth(idepth) = d2b(ilayer)
          vel(idepth) = column2(ilayer)
          dens(idepth) = column3(ilayer)
        end do
        ndepths = idepth
      else
        ndepths = nentries
        do idepth = 1, ndepths
          depth(idepth) = column1(idepth)
          vel(idepth) = column2(idepth)
          dens(idepth) = column3(idepth)
*  Fill in density from velocity-density relation if density = 0.0
          if (dens(idepth) .eq. 0.0) then
            dens(idepth) = density(vel(idepth),
     :                     specify_dens_coeff,
     :                     dens_low, vel_low,
     :                     dens_high, vel_high)
          end if
        end do
      end if

* Now the information is in the same format and we can treat it in the
* same way.

      tt(1) = 0.0
      slope(1) = 0.0
      avgdens(1) = dens(1)
      do idepth = 2, ndepths
        if (vel(idepth) .eq. vel(idepth-1)) then   ! constant velocity
          slope(idepth) = 0.0
          tt(idepth) = tt(idepth-1) +
     :                 (depth(idepth)-depth(idepth-1))/vel(idepth)
        else
          if(depth(idepth) .eq. depth(idepth-1)) then  ! step change
            slope(idepth) = 1.0e10
            tt(idepth) = tt(idepth-1)
          else
            slope(idepth) =
     :       (vel(idepth)-vel(idepth-1))/(depth(idepth)-depth(idepth-1))
            tt(idepth) = tt(idepth-1) +
     :       (1.0/slope(idepth))*alog(vel(idepth)/vel(idepth-1))
          end if
        end if
        avgdens(idepth) = (1.0/tt(idepth))*
     :    (tt(idepth-1)*avgdens(idepth-1)+
     :     (tt(idepth)-tt(idepth-1))*(dens(idepth)+dens(idepth-1))/2.0)
      end do

      write(*,'(a)') '$ Enter source velocity and density: '
      read(*, *) velsource, denssource

* INSERT CODE TO INTERPOLATE HERE... BUT DON'T WANT EQUAL INCREMENTS FOR
* ALL DEPTHS... BETTER TO WORK WITH APPROXIMATE FREQUENCY SPACING AND DIVIDE
* DEPTHS AS NEEDED.

      write(nu_out, '(a,1p2(1x,e10.3))')
     :  ' source vel & dens = ',  velsource, denssource
      if (specify_dens_coeff) then
        write(nu_out, '(a, 4(1x,f6.3))')
     :   '  Density coefficients specified; dens, vel, low, high = ',
     :    dens_low, vel_low, dens_high, vel_high
      else
        write(nu_out, '(2a, 4(1x,f6.3))')
     :   '  Density coeffs not specified; ',
     :   'assumed dens, vel, low, high = ',
     :    dens_low, vel_low, dens_high, vel_high
      end if

* Set up output arrays... only need this because want to output both the input
* model and the results in the same file, side by side, and don't want
* the output repeated when two depths are the same

      nout = 0
      depth_out(0) = 0.0
      tt_out(0) = 0.0
      do idepth = 2, ndepths
        if (depth(idepth) .eq. depth(idepth-1)) goto 777  ! skip repeated depths
        nout = nout + 1
        depth_out(nout) = depth(idepth)
        tt_out(nout) = tt(idepth)
        thick_layer(nout) = depth_out(nout) - depth_out(nout-1)
        vel_layer(nout) = thick_layer(nout)/
     :                    (tt_out(nout) - tt_out(nout-1))
        dens_layer(nout) = (dens(idepth)+dens(idepth))/2.0
        avgvel_out(nout) = depth(idepth)/tt(idepth)
        avgdens_out(nout) = avgdens(idepth)
        freq_out(nout) = 1.0/(4.0*tt(idepth))
        ampli_out(nout) = sqrt( (denssource*velsource) /
     :   (avgdens_out(nout)*avgvel_out(nout)) )
777     continue
      end do

      write(nu_out, '(t4,a,t14,i4,t20,a,t27,i4)')
     : 'ndepths = ', ndepths, 'nout = ', nout

      tempbuf = ' '
      tempbuf(1:8) = f_in(1:index(f_in,'.')-1)
      col7 = ' '
      col7(9-len_trim(tempbuf):8) = tempbuf(1:len_trim(tempbuf))
      col7(9:12) = '-frq'
      col8 = ' '
      col8(9-len_trim(tempbuf):8) = tempbuf(1:len_trim(tempbuf))
      col8(9:12) = '-amp'

      write(nu_out, 309) col7, col8
309   format( t2,'depth_in', t14,'vel_in', t23,'dens_in',
```

```
     :        t33,'depth_out', t43,'travltime',
     :        t53,'thck_lyr', t62,'vel_layer', t72,'dens_layr',
     :        t85,'avgvel', t94,'avgdens', t102,a, t115,a)

* Assume nout .le. ndepths... perhaps this should be checked
      do i = 1, nout
         outbuf = ' '
         write(outbuf, '(1x,1pe8.2,2(1x,e9.3),3x,1x,e8.2,
     :    1x,e9.3,1x,e8.2,
     :    4(1x,e9.3),2(4x,e9.3))')
     :    depth(i), vel(i), dens(i),
     :    depth_out(i), tt_out(i),
     :    thick_layer(i), vel_layer(i), dens_layer(i),
     :    avgvel_out(i), avgdens_out(i),
     :    freq_out(i), ampli_out(i)
         write(nu_out, '(a)') outbuf
      end do
      do i = nout+1, ndepths
         outbuf = ' '
         write(outbuf,
     :   '(1x,1pe8.2,2(1x,e9.3),t41,a,t51,a,t60,a,t70,a,t80,a,
     :    t90,a,t100,a,t113,a,t126,a)')
     :    depth(i), vel(i), dens(i), '.', '.' ,'.' ,'.' ,'.' ,
     :    '.' ,'.' ,'.' ,'.' ,'.'
         write(nu_out, '(a)') outbuf
      end do

      close(unit=nu_out)

      write(*, '(a)') ' FINISHED --- Loop back for another case'
      goto 666
999   continue

      stop
      end

* -------------------------- BEGIN DENSITY --------------------------
      function density(vels,
     :                 specify_dens_coeff,
     :                 dens_low, vel_low,
     :                 dens_high, vel_high)

* Assigns a density based on shear velocity.  If specify_dens_coeff = .true.,
* then a straight line is fit between the coefficients (and density = dens_low
* for vels < vel_low and density = dens_high for vels > vel_high).
* If specify_dens_coeff = .false, then the coefficients are set internally
* to the following values:
*     dens  velocity
*     2.5   0.30   (close to near-surface velocity for WNA generic rock site
*     2.8   3.50

* Another relation to consider  density = 1.73 + 0.29* vels
* (From fitting data in a model provided by H. Magistrale, probably based
* on a Nafe & Drake relation).

* Dates: 12/05/95 - Written by D.M. Boore
*        12/08/95 - Added specify_dens_coeff and the coefficients
*        04/16/96 - Changed vel_low from 0.291 to 0.3

      logical specify_dens_coeff, firstcall/.true./
      save firstcall

      if (firstcall) then
        firstcall = .false.
        if (.not. specify_dens_coeff) then
          dens_low = 2.5
          vel_low = 0.3
          dens_high = 2.8
          vel_high = 3.5
        end if
        slope = (dens_high - dens_low)/(vel_high - vel_low)
      end if

      if (vels .lt. vel_low) then
        density = dens_low
      else if (vels .gt. vel_high) then
        density = dens_high
      else
        density = dens_low + (vels - vel_low)*slope
      end if

      return
      end
* -------------------------- END DENSITY --------------------------
```

```
      Program F4Rattle

* Reads in an output file from Site_Amp and makes a file for use
* as input to Rattle

* Dates: 12/17/95 - Written by D.M. Boore
*        12/21/95 - modified I/O

      character f_in*80, f_out*80

      logical f_in_exist

      f_in_exist = .false.
      do while (.not. f_in_exist)
        f_in = ' '
        write(*, '(a\)')
     :    ' Enter name of input file (cr to quit): '
        read(*, '(a)') f_in
        if (f_in(1:4) .eq. '    ') stop
        inquire(file=f_in, exist=f_in_exist)
        if (.not. f_in_exist) then
          write(*,'(a)') ' ******* FILE DOES NOT EXIST ******* '
        end if
      end do

      nu_in = 10
      open(unit=nu_in, file=f_in, status='unknown')

      f_out = ' '
      write(*, '(a\)')
     :  ' Enter name of output file (cr=rattle.in): '
      read(*, '(a)') f_out
      if (f_out(1:4) .eq. '    ') f_out = 'rattle.in'
      nu_out = 20
      open(unit=nu_out, file=f_out, status='unknown')

      read(nu_in, '(t23,e10.3, t34,e10.3)') velsource, denssource
      call skip(nu_in,1)
      read(nu_in,'(t27,i4)') nout
      call skip(nu_in,1)

      write(*, '(a\)')
     :  ' Enter zdepth (usually 0.0): '
      read(*, *) zdepth

      write(*, '(a\)')
     :  ' Enter theta (angle of incidence from vertical): '
      read(*, *) theta

      write(*, '(a\)')
     :  ' Enter Q of layers: '
      read(*, *) q

      write(*, '(a\)')
     :  ' Enter sps, mx (see text): '
      read(*, *) sps, mx

      write(nu_out, '(2a)') ' File from Site_Amp: ', f_in

      write(nu_out, '(1x,i4, 1p4(1x,e11.4),1x,0pi4)')
     :  nout, sps, theta, velsource,
     :  denssource, mx

      do i = 1, nout
        read(nu_in,'(t53,e8.2,t62,e9.3,t72,e9.3)')
     :              di, vel, dens
        write(nu_out, *) i, di, vel, q, dens
      end do
      write(nu_out, *) zdepth

      close(unit=nu_in)
      close(unit=nu_out)

      stop
      end

      subroutine SKIP(lunit, nlines)
        do i = 1, nlines
          read(lunit, *)
        end do
        return
      end
```

```
Parameters for Atkinson and Boore (1995)
rho, beta, prtitn, rtp, fs:
    2.8 3.8 0.707 0.55 2.0
spectral shape: source number (1=Single Corner;2=Joyner;3=A93;4=custom),
                pf, pd (1-corner spectrum = 1/(1+(f/fc)**pf)**pd; 0.0 otherwise)
                (usual model: pf=2.0,pd=1.0; Butterworth: pf=4.0,pd=0.5)
                (Note: power of high freq decay --> pf*pd)
    3 2.0 1.0
spectral scaling: stressc, dlsdm, fbdfa, amagc
                (stress=stressc*10.0**(dlsdm*(amag-amagc))
                (fbdfa, amagc for Joyner model, usually 4.0, 7.0)
                (not used for source 3, but placeholders still needed)
    100.0 0.0 4.0 7.0
gsprd: nsegs, (rlow(i), slope(i))  (Set rlow(1) = 1.0)
    3
      1.0 -1.0
     70.0  0.0
    130.0 -0.5
q: fr1, Qr1, s1, ft1, ft2, fr2, qr2, s2
    1.0 680 0.36 1.0 1.0 1.0 680 0.36
source duration: weights of 1/fa, 1/fb
    0.5 0.0
path duration: nknots, (rdur(i), dur(i), slope of last segment
    4
      0.0 0.0
     10.0 0.0
     70.0 9.6
    130.0 7.8
    0.04
site amplification: namps, (famp(i), amp(i))
    1
    1.0 1.0
site diminution parameters: fm, akappa
    50.0 0.0
low-cut filter parameters: fcut, norder
    0.0 2
rv integration params: zup, eps_int (integration accuracy), amp_cutoff (for fup)
    10.0 0.00001  0.001
window params: indxwind(0=box,1=exp), taper(<1), twdtmotion, eps_wind, eta_wind
    1 0.05 1.0 0.2 0.05
timing stuff: tsimdur, dt, tshift, seed, nruns
    40.0  0.005 5.0 123.0 100
remove dc from random series before transforming to freq. domain (0=no;1=yes)?
    0
```


Appendix F: Input-Parameter File for Atkinson and Boore (1995) -- File AB95.DAT, Page 1 of 1

```
Coastal California model (SUBJECT TO CHANGE! DON'T USE BEYOND 100 KM)
rho, beta, prtitn, radpat, fs:
    2.8 3.5 0.707 0.55 2.0
spectral shape: source number (1=Single Corner;2=Joyner;3=A93;4=custom),
                pf, pd (1-corner spectrum = 1/(1+(f/fc)**pf)**pd; 0.0 otherwise)
                (usual model: pf=2.0,pd=1.0; Butterworth: pf=4.0,pd=0.5)
                (Note: power of high freq decay --> pf*pd)
    1 2.0 1.0
spectral scaling: stressc, dlsdm, fbdfa, amagc
                (stress=stressc*10.0**(dlsdm*(amag-amagc))
                (fbdfa, amagc for Joyner model, usually 4.0, 7.0)
                (not used for source 3, but placeholders still needed)
    70.0 0.0 4.0 7.0
gsprd: nsegs, (rlow(i), slope(i))  (Set rlow(1) = 1.0)
    1
      1.0 -1.0
q: fr1, Qr1, s1, ft1, ft2, fr2, qr2, s2
    0.1 275 -2.0 0.2 0.6 1.0 88.0 0.9
source duration: weights of 1/fa, 1/fb
    1.0 0.0
path duration: nknots, (rdur(i), dur(i), slope of last segment)
    1
      0.0 0.0
    0.05
site amplification: namps, (famp(i), amp(i))
    11
    0.01          1.00
    0.09          1.10
    0.16          1.18
    0.51          1.42
    0.84          1.58
    1.25          1.74
    2.26          2.06
    3.17          2.25
    6.05          2.58
    16.6          3.13
    61.2          4.00
site diminution parameters: fm, akappa
    100.0 0.035
low-cut filter parameters: fcut, norder
    0.0 2
rv integration params: zup, eps_int (integration accuracy), amp_cutoff (for fup)
    10.0 0.00001  0.001
window params: indxwind(0=box,1=exp), taper(<1), twdtmotion, eps_wind, eta_wind
    1 0.05 1.0 0.2 0.05
timing stuff: tsimdur, dt, tshift, seed, nruns
    40.0  0.005 7.0 123.0 100
remove dc from random series before transforming to freq. domain (0=no;1=yes)?
    0
```