560051

# Using the C language to approximate non-linear stochastic systems

SERGIO E. SERRANO

*Department of Civil Engineering, University of Kentucky, Lexington, Kentucky 40506-0046, USA*

**Recognizing the growing popularity and demand for scientific or numerical libraries in the C computer language, and the shortage of systematic methods for the increasingly important non-linear stochastic systems, this article attempts to present a method to approximate the solution of systems governed by non-linear stochastic differential equations and the corresponding algorithm and computer code in the C language. It was found that the C language offers promising possibilities for the stochastic systems modeler. It is hoped the use of C for the analysis of these type of problems will be encouraged.**

Key Words: stochastic differential equations, non-linear systems, Green's function approximation, C programming language.

## 1. INTRODUCTION

The 'C' programming language evolved from a succession of programming languages developed at Bell Laboratories in the early 1970s. It was not until the late 1970s, however, that this programming language began to gain widespread popularity and support. This was because until that time C compilers were not readily available for commercial use outside of Bell Laboratories. Furthermore, C's growth in popularity has been spurred on in part by the equal, if not faster, growth in popularity of the UNIX operating system. This operating system, which was also developed at Bell Laboratories, has C as its standard programming language. In fact, over 90% of the operating system itself is written in the C language. C is generally available in most large 'main frame' computer systems today. In recent years a variety of C compilers under the very popular DOS microcomputer operating systems have become available to the scientific community (i.e., Microsoft Turbo C, and the new Microsoft Quick C).

C is so called a 'higher-level language', yet it provides capabilities that enable the user to 'get close' with the hardware and deal with the computer on a much lower level. This is because, while the C language is a general-purpose structured programming language, it was designed with systems programming applications in mind and as such provides the user with an enormous amount of power and flexibility. In fact, programming applications exist that could be easily handled by the C language but that would be difficult, if not impossible, to develop in other languages, such as PASCAL, FORTRAN, or BASIC. Thus the C language is operator-rich, giving direct access to most capabilities of a machine-language instruction set. It has a large variety of intrinsic data types (short and long, signed and unsigned integers; floating and double precision reals; pointer types; etc.), and a concise syntax for effecting conversions and indirections. It defines an arithmetic on pointers (addresses) that relates gracefully to array addressing and is highly compatible with the index register structure of many computers.

Portability has always been a strong point of the C language. Both the C language and the UNIX operating system have by now been implemented in literally hundreds of different computers. The language universality, portability and flexibility have attracted an increasing number of scientists and engineers to it. It is commonly used for the real-time control of experimental hardware, often in spite of the fact that the standard UNIX kernel is less than ideal as an operating system for this purpose.

The use of C for higher level scientific calculations such as data analysis, modeling, and floating-point numerical work has generally been slower in developing. In part this is due to the entrenched position of FORTRAN as the mother-tongue of virtually all scientists and engineers born before 1960, and most born after. In part, also, the slowness of C's penetration into scientific computing has been due to deficiencies in the language that computer scientists have been slow to recognize. Examples are the lack of a good way to raise numbers to small integer powers, and the implicit conversion of float to double. Many, though not all, of these deficiencies are overcome in the Proposed American National Standards Institute (ANSI) C Standard. Some remaining deficiencies will undoubtedly be corrected over time.

Yet another inhibition to the increasing popularity of C among scientists has been the shortage of high-quality scientific or numerical libraries, although important contributions have already been released to the public (i.e., Press *et al.*[7]). The existing libraries have made

Accepted January 1990. Discussion ends September 1990.

available a wide variety of numerical techniques commonly used in scientific computing of engineering problems governed by linear deterministic systems. However, very little has been done with respect to stochastic non-linear systems, a problem we feel is shared by other computer languages. The problem lies in the important difficulties in the development of techniques for the approximation of systems governed by non-linear differential equations. Much research remains to be done in order to develop standard approximation techniques, similar to the ones available for linear differential equations. The problem becomes more complex when we consider engineering systems governed by non-linear differential equations whose coefficients, forcing functions, boundary conditions or initial conditions are defined in stochastic terms, because the measurement uncertainty or the erratic environmental variability of these quantities is better described in stochastic terms. Although some numerical methods are available for the discretization and approximation of non-linear deterministic differential equations, the approximations may not be valid in the case of non-linear stochastic differential equations, because the discretized equation may not satisfy the original differential equation.

Since real-life engineering systems are governed by non-linear stochastic systems, there is a gradual transition from linear deterministic to non-linear and stochastic modeling in many fields of engineering analysis and design. This article attempts to satisfy two important needs, namely the need for systematic approximation techniques for non-linear stochastic systems and the need for their practical implementation with the increasingly important C computer language. The objective is to present the reader with a very important, not well-known, method for approximating non-linear stochastic differential equations and to illustrate the practical implementation of the corresponding algorithm in C.

In section 2 the approximation method is described through an example application to a non-linear stochastic differential equation of the type frequently appearing in engineering applications. Familiarity with the theory of stochastic systems is assumed, although emphasis is made on the practical aspects of the solution. The reader interested in the theoretical basis of the method may consult Serrano[9,10] or Adomian.[1] One important advantage of the method in contrast with other solution methods (i.e., perturbations techniques, hierarchy techniques), is that this method is not limited to small variance in the stochastic quantities. However the most important advantage of this method is its close relationship with traditional linear systems theory. Linear systems methods and the procedures currently employed for the derivation of the impulse response function (or the Green's function, or the semigroup) may also be used. Then the impulse response function will appear in each of the subsequent iterations of the approximation algorithm of the non-linear stochastic system.

In section 3, a description of the computational aspects of the example problem in section 2 is included. Familiarity with the C language is assumed. Readers not versed in the language may consult one of the extensive references in the topic (i.e., Kochan[6]). Finally, in Appendix A a list of the source code for the example problem is included along with an explanation.

## 2. APPROXIMATION OF NON-LINEAR STOCHASTIC SYSTEMS

We begin the illustration on the approximation of non-linear systems subject to random disturbances by considering the following differential equation subject to a random parameter:

$$\frac{d}{d\lambda}\left[D(\lambda,\omega)\frac{d\theta(\lambda,\omega)}{d\lambda}\right]+\frac{\lambda}{2}\frac{d\theta(\lambda,\omega)}{d\lambda}=0, \tag{1}$$

subject to

$$\theta(0)=\theta_0,\ \theta(\infty)=\theta_n$$

In this equation $\theta(\lambda,\omega)$ is the stochastic process representing the system output; $\lambda$ is the independent variable; $D(\theta)=D(\lambda,\omega)$ is the stochastic process representing the uncertain variability in the equation parameter; $\omega$ is the probability variable; and $\theta_0$ and $\theta_n$ are the left and right constant boundary conditions.

Physically this boundary value problem models the volumetric soil-water content evolution, $\theta(\mathrm{mm}^3/\mathrm{mm}^3)$, with respect to the Boltzman-transformed variable, $\lambda$ $(\mathrm{mm/hr}^{1/2})$ in a homogeneous horizontal porous media. Thus $D$ is the soil-water diffusivity (Bear[2]). Several laboratory experiments were conducted before a model equation was studied. The description of the experiments and the physical validity of the differential equation are outside the scope of the present article. Equation (1) is a special case of the Sturm-Liouville differential equation very commonly found in many engineering systems. It is a non-linear equation in the stochastic quantities and the solution techniques for this type of system are not readily available as the corresponding ones for linear deterministic systems.

Assuming that the most important element of variability, due to the combined effect of uncertainties in the unsaturated transport phenomenon and measurement errors, is the uncertainty in the space and time variability of $D$, we define this parameter as a random process of the form

$$D(\lambda,\omega)=\bar{D}+D'(\lambda,\omega) \tag{2}$$

where $\bar{D}$ represents the expected value in the soil-water diffusivity, assumed to be constant, $D'(\lambda,\omega)$ represents the random space and time variability of this parameter. We further assume that this random variability is one which increases erratically with $\lambda$, as observed from the experiments, and thus we choose a suitable random process such as a Brownian motion process with the properties (Jazwinski[5])

$$E\{D'(\lambda)\}=0,\qquad E\{D'(\lambda_1)D'(\lambda_2)\}=q\ \min(\lambda_1,\lambda_2) \tag{3}$$

where $E\{\ \}$ denotes the expectation operator; $q$ is the variance parameter; and $\min(\lambda_1,\lambda_2)$ denotes the minimum between $\lambda_1$ and $\lambda_2$.

Substituting equation (2) into equation (1), placing the terms containing the random coefficients in the right side and rearranging, we obtain a stochastically-forced

equation of the form

$$\frac{d^2\theta}{d\lambda^2} + \alpha\lambda \frac{d\theta}{d\lambda} = 2\alpha R(\lambda, \omega)\theta \tag{4}$$

where $\alpha = 1/2\bar{D}$; the operator

$$R(\lambda, \omega)\theta = \left[ -D'(\lambda, \omega) \frac{d^2}{d\lambda^2} - w(\lambda, \omega) \frac{d}{d\lambda} \right]\theta \tag{5}$$

$w = \partial D'/\partial\lambda$ is the formal derivative of the Brownian motion process, that is a White Gaussian noise process with the properties (Jazwinski[5])

$$E\{w(\lambda)\} = 0, \qquad E\{w(\lambda_1)w(\lambda_2)\} = q\delta(\lambda_1 - \lambda_2) \tag{6}$$

where $\delta(\ )$ is the Dirac's delta function. In the above equations $\omega$ has been dropped for convenience, but it is clear that because the differential equation contains random functions, the dependent variable, $\theta$, is also a random process.

The general solution of Equation (4) is given by (see Cakmak *et al.*[3] for a description of the Green's function method)

$$\theta(\lambda) = \Psi(\lambda) + \int_0^\infty G(\lambda, \xi)R(\xi)\theta(\xi)\, d\xi \tag{7}$$

where the kernel $G$ is the Green's function, and the function $\Psi$ is the partial solution satisfying the boundary conditions as given by

$$\Psi(\lambda) = \theta_0 - A\lambda + B\lambda^3 - C\lambda^5, \qquad 0 \leqslant \lambda \leqslant a \tag{8}$$

where $a$ is the limiting right value of $\lambda$ as given by the laboratory experiment, and A, B and C are constants. Equation (8) is actually an approximation of the series solution

$$\Psi(\lambda) = \theta_0 - E\left(\lambda - \frac{\alpha\lambda^3}{3!} + \frac{3\alpha^2\lambda^5}{5!} - \cdots\right)$$

where $E$ is a constant, which can easily be found by the method of Frobenius (Spiegel[17]). The Green's function in equation (7) is easy to derive as (see Reinhard[8] for a description of similar equations)

$$G(\lambda, \xi) = U(\lambda - \xi)\left\{\frac{Z(\xi)Z(\lambda)}{Z'(\xi)Z(a)} - \frac{Z(\xi)}{Z'(\xi)}\right\} + U(\xi - \lambda)\left\{\frac{[Z(\xi) - Z(a)]\, Z(\lambda)}{Z'(\xi)Z(a)}\right\} \tag{8}$$

where $Z(\xi) = [\Psi(\xi) - \theta_0]/E$, $Z'(\xi)$ is the first derivative of $Z$ evaluated at $\xi$, and $U(\ )$ denotes the unit step function.

The integral term in equation (7) contains $\theta$. Thus we will approximate this integral by expanding $\theta$ as an infinite series of partial solutions $\theta = \theta_1 + \theta_2 + \cdots$. For a description of this approximation procedure of nonlinear equations and a discussion of the convergence speed, the reader is referred to Serrano and Unny,[11] Serrano[10] and Adomian.[1] That is, we define $\theta = \sum_{i=1}^{\infty}\theta_i$ and equation (7) reduces to

$$\theta(\lambda) = \Psi(\lambda) + \sum_{i=1}^{\infty} \int_0^a G(\lambda, \xi)R(\xi)\theta_i(\xi)\, d\xi \tag{9}$$

For dissipative systems such as the one in question, the convergence speed of this approximation series is extremely fast and ordinarily only a few terms in the series are needed. Furthermore, since this is not a perturbation approximation, arbitrarily large variances in the stochastic terms can be included. We initiate the approximation by setting $\theta_1 = \Psi$, which is the previous approximation and compute recursively subsequent terms in the series by setting

$$\theta_i(\lambda, \xi) = \int_0^a G(\lambda, \xi)R(\xi)\theta_{i-1}(\xi)\, d\xi \tag{10}$$

Equation (9) can be used to generate sample functions of $\theta$ in order to observe the qualitative behaviour of the output function and to replicate experimental data. This is easily done by generating sample functions of the processes $D'$ and $w$ and solve for $\theta$ numerically. Theoretically we would be interested in the joint probability density function of all orders of the $\theta$ process, which is an enormous task in most cases. Furthermore, in most practical engineering problems we only have at hand the first and second order moments of the disturbing stochastic processes. Therefore we are interested in computing the first and second order moments, or more specifically the mean and the variance, of $\theta$ as a function of $\lambda$. The first two moments contain sufficient information on the statistical properties of the output process. After taking expectations on both sides of equation (9), and using equation (3), one obtains the mean of $\theta$ as

$$E\{\theta(\lambda)\} = \Psi(\lambda) \tag{11}$$

The variance of $\theta$ can also be deduced from equations (3), (6) and (9) after truncating at the first order term in the series because the higher order terms were very small in magnitude. Following some algebraic manipulation one obtains the variance $\sigma_\theta^2$ to be

$$\sigma_\theta^2 = \int_0^a \int_0^a G(\lambda, \xi)G(\lambda, \rho)E\{R(\xi)R(\rho)\}\Psi(\xi)\Psi(\rho)\, d\rho\, d\xi \tag{12}$$

where the correlation term $E\{R(\xi)R(\rho)\}$ is given by

$$E\{R(\xi)R(\rho)\} = [\min(\xi, \rho)\nabla_\xi^2\nabla_\rho^2 + U(\xi - \rho)\nabla_\xi^2\nabla_\rho + U(\rho - \xi)\nabla_\xi\nabla_\rho^2 + \delta(\xi - \rho)\nabla_\xi\nabla_\rho] \tag{13}$$

$\nabla_\xi^2 = \partial^2/\partial\xi$; $U(\ )$ denotes the unit step function, and the rest of the terms as before.

Several examples of the above methodology to approximate stochastic systems governed by various differential equations can be seen in Serrano *et al.*,[11–16] and Serrano.[9,10] Most of the applications have covered partial differential equations in one dimension, two dimensions and three dimensions, both in finite and semi-infinite domains, for the cases where the stochastic component appears in either the parameters, the boundary conditions, the initial conditions or the source terms. For applications to other ordinary differential equations the reader is referred to Adomian.[1]

## 3. COMPUTATIONAL RESULTS

Calculation of sample functions, the mean and the variance of the process $\theta$ can be done by choosing a numerical approximation of equations (9) through (13), using measured values for the parameters, and adopting an appropriate computational media for the calculations. Lower order forward finite difference approximations for the derivative terms were used in the present work. Although higher order approximations are more accurate, it is known that these approximations present unstability problems. This is specially true in the case of

'noisy' or stochastic functions. For the sample function calculations, trapezoidal rule of integration was used, since a more accurate integration scheme is not justified for stochastic data. For the variance calculations, a 24-point Gauss-Legendre quadrature method proved most accurate. The details on the approximation algorithm are not included since these are part of well-documented techniques in the literature (i.e., Chapra and Canale[4]).

The values of the parameters were selected from the several laboratory experiments of horizontal infiltration conducted previously. The following values were used: $\bar{D} = 2.77\ \text{mm}^2/\text{s}$, $\alpha = 0.1801\ \text{s/mm}^2$, $a = 2.7\ \text{mm/s}^{1/2}$ $\theta_0 = 0.458$, $\theta_n = 0.086$, $q = 0.0002$, $A = E = 3.217 \times 10^{-3}$, $B = 9.651 \times 10^4$, $C = 2.6057 \times 10^{12}$, and the computation interval $\Delta\lambda = 0.1\ \text{mm/s}^{1/2}$. It was noted that only two or three terms in the summation series in equation (9) were necessary since the convergence speed was very fast.

The C programming language was selected as a source code for the reasons stated earlier. Implementation of equations (9) through (13) will require the repetitive calculation of the Green's function, $G(\ )$, the unit step function, $U(\ )$, the delta function, $\delta(\ )$, first and second derivatives of previous approximations, factorials, and a numerical integration algorithm. The program also needs the generation of a White Gaussian noise sequence and a Brownian Motion sequence for the generation of sample values of $\theta$. The random sequences are produced after the generation of uniform random numbers. These are functions very commonly used in stochastic systems analysis. It is best to program these mathematical functions as programming functions in C to be called from the main program, main(). As an illustration, Appendix A includes a list and explanation of the program to compute sample values of $\theta$ (equations (9) and (10)). It can be appreciated the compactness, clarity and flexibility of C. The main routine is confined within the main() section, which constitutes the core of the iteration, and the programming technique consists of subsequent calls to the several functions (also included in the list). One particularly interesting feature of the C language is the support of a capability known as the recursive function, in which a function may contain statements calling the execution of itself. See for example the function fact() in Appendix A. Recursive functions can be used to succinctly and effectively solve problems. They are commonly used in applications in which the solution to a problem can be expressed in terms of successively applying the same solution to subsets of the problem. Usually this may include the evaluation of expressions containing nested sets of parenthesized expressions. Other common applications involve the searching and sorting of data structures called trees and lists.

The programer must exercise some caution when writing functions. C will only allow the same type of constants or variables to be transferred from the main () routine to the function. Thus the information transfer, either a constant, a variable or an array, must be declared as integer (int), floating point (float) or double precision (double) in both the main () routine and the function itself.

In the view of this author two difficulties had to be faced when programming numerical algorithms in the C language. First, some intrinsic functions available in most computer languages do not exist in C. For example exponentiation or raising a variable $x$ to the power of $y$, which is almost universally written in other languages as $x^{**}y$, does not exist in C. This problem is solved by invoking a library of mathematical functions which is available in UNIX System V systems. Thus by writing the statement #include ⟨math.h⟩ at the top of the program this and other functions commonly used in numerical algorithms are available. The power raising statement will read pow($x$, $y$). The second difficulty to be faced when programming in C is the fact that all internal calculations in C are done in double precision. Although this feature may look very appealing from the point of view of accuracy, it certainly consumes a great deal of CPU time when conducting large scale numerical calculations. Some C compilers provide an optional compilation mode in which the implicit conversion of float to double is suppressed. The proposed ANSI C standard does not allow implicit conversion for arithmetic operations, but it requires it for function arguments.

Many other characteristics and recommendations on the use of C in scientific numerical computing, along with many programs commonly used in numerical algorithms, are included in the new C version of the old classical book Numerical Recipes by Press *et al.*

Although the programming functions included in Appendix A may not be 100% trouble free, a substantial effort was done to debug the program and we believe the reader working with computational aspects of stochastic systems may benefit from them.

The program was compiled and executed in three independent computer systems in order to study the portability of the code and the different execution times. The transport media for the code between the computer systems was BITNET electronic mail. A Harris HCX-7 minicomputer took about 3 seconds to execute; An AT&T 3B20 minicomputer took about 8 seconds and an AT&T PC 7300 microcomputer took about 15 seconds to execute. The operational systems in the three computers were UNIX System V. Compiling the program and linking with the mathematical libraries is simply done with the command cc program.c −lm, where program.c is the file containing the source code. If no syntax or logical errors are detected, the compiler creates an executable file.

Figure 1 illustrates the results of the computer program of Appendix A, which calculates the algorithm for equations (9) and (10). The figure shows the function $\theta$ with respect to $\lambda$ for three iterations. Note the effectiveness of the approximation method of non-linear stochastic systems presented in the previous section. After three iterations the contribution of the next iteration to the total value of $\theta$ becomes negligible at the typical scale at which $\theta$ is measured. It is also worth mentioning that stability problems may arise if the individual iterations produce erratic functions. This is easy to see in equation (9), where the derivative terms for the subsequent approximation may produce unstable values if the calculation is based on a previous approximation which is not smooth. This is particularly true when the functions involved are noisy, since for a wide-band random process the derivatives of the sample functions do not exist in the Riemann sense. Note the unstability of iteration number three in Fig. 1, which is not important

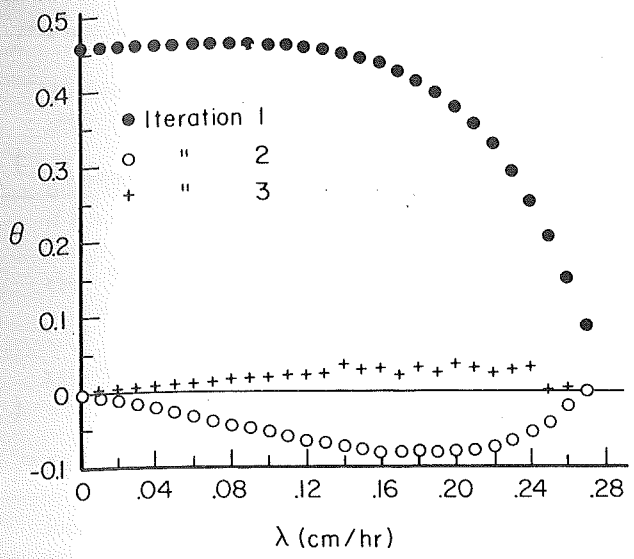


*Fig. 1. θ versus λ for three iterations*

in the present example since only three iterations are necessary. However, some smoothing procedure should be applied to each iteration before it is fedback in the next approximation.

It was found that with very little care the C language offers the programmer with universal, hardware independent, application programs. The conciseness and flexibility of the programming language presents very important possibilities for scientific computing of stochastic non-linear systems. This added to the increasing popularity of numerical libraries in C, the increasing availability of C compilers in a variety of hardware environments, and the possibilities of the C language for systems programming, makes of C a programming language with very promising possibilities for engineering computation in the future.

## 4. CONCLUSIONS

The C language is becoming an important computer language for scientific or numerical computation. As more engineers and scientists are interested to enjoy the possibilities of C, the need for high-quality scientific libraries increases. Although some important contributions have been made in the development of libraries for the computational aspects of linear deterministic systems, much work remains to be done in the creation of libraries for non-linear stochastic systems in C.

Research in this direction will have to face important difficulties posed by the solution of non-linear stochastic differential equations. In the present article, a systematic method for the solution approximation of non-linear stochastic systems was presented along with the corresponding computer code in C. It was found that the C language offers very promising possibilities for the modeler of non-linear stochastic systems.

## ACKNOWLEDGEMENTS

Thanks to the Department of Civil Engineering of the University of Kentucky and to the Kentucky Water Resources Research Institute for their financial and logistic support which made possible the present work.

## REFERENCES

1 Adomian, G. *Stochastic Systems.* Academic Press, New York, 1983
2 Bear, J. *Hydraulics of Groundwater.* McGraw-Hill Book Company, New York, 1979
3 Cakmak, A. S., Botha, J. F. and Gray, W. G. *Computational and Applied Mathematics for Engineering Analysis.* Springer-Verlag, London, 1987
4 Chapra, S. C. and Canale, R. P. *Numerical Methods for Engineers.* Second Edition. McGraw-Hill Book Company, New York, USA, 1988
5 Jazwinski, A. H. *Stochastic Processes and Filtering Theory.* Academic Press, New York, 1970
6 Kochan, S. G. *Programming in C.* Revised Edition. Hayden Books, Indianapolis, Indiana, USA, 1988
7 Press, W. H., Flannery, B. P., Teukolsky, S. A. and Vetterling, W. T. *Numerical Recipes in C. The Art of Scientific Computing.* Cambridge University Press, New York, USA, 1988
8 Reihard, H. *Differential Equations. Foundations and Applications.* Macmillan Publishing Company, New York, USA, 1987
9 Serrano, S. E. General solution to random advective-dispersive equation in porous media. Part I: Stochasticity in the sources and in the boundaries. *Stochastic. Hydrol. Hydraul.*, 1988, **2**(2); 79–98
10 Serrano, S. E. General solution to random advective-dispersive equation in porous media. Part II: Stochasticity in the parameters. *Stochastic Hydrol. Hydraul.*, 1988, **2**(2), 99–112
11 Serrano, S. E. and Unny, T. E. Semigroup solutions to stochastic unsteady groundwater flow subject to random parameters. *Stochastic. Hydrol. Hydraul.*, 1987, **1**(4), 281–296
12 Serrano, S. E. and Unny, T. E. Predicting groundwater flow in a phreatic aquifer. *J. of Hydrol.*, 1987, **95**, 241–268
13 Serrano, S. E. and Unny, T. E. Boundary element solution of the two-dimensional groundwater flow equation with stochastic free-surface boundary condition. *Num. Meth. Part. Diff. Eqs.*, 1986, **2**, 237–258
14. Serrano, S. E., Unny, T. E. and Lennox, W. C. Analysis of stochastic groundwater flow problems. Part I: Deterministic partial differential equations in groundwater flow. A functional-analytic approach. *J. of Hydrol.*, 1985, **82** (3-4), 247–263
15 Serrano, S. E., Unny, T. E. and Lennox, W. C. Analysis of stochastic groundwater flow problems. Part II: Stochastic partial differential equations in groundwater flow. A functional-analytic approach. *J. of Hydrol.*, 1985, **82**(3-4), 265–284
16 Serrano, S. E., Unny, T. E. and Lennox, W. C. Analysis of stochastic groundwater flow problems. Part III: Approximate solution of stochastic partial differential equations. *J. of Hydrol.*, 1985, **82**(3-4); 285–306
17 Spiegel, M. R. *Advanced Mathematics for Engineers and Scientists.* Schaum's outline series, McGraw-Hill Book Company, New York, USA, 1980

```
/*=============================================================================
  Program program.c     May, 1989.       Sergio E. Serrano
  =============================================================================
  Program for the generation of a sample water content theta
  with respect to lambda .
  ============================================================================= */
/* Declaration of functions and libraries to use in the program */

#include <math.h>    /* call mathematical libraries  */
double drand48( );  /* generator of uniform random numbers */
double RAN( );/* generator of Gaussian and Brownian sequences */
double U( );   /* computes the unit step function */
long int fact( );  /* computes the factorial of a function */
double Z( );     /* function required bny the Green's function */
double ZPR( );   /* function required by the Green's function */
double G( );     /* computes the Green's function */
double FD( );    /* computes the first derivative of an array */
double SD( );    /* computes the second derivative of an array */

/* Function for the generation of a sample Brownian motion process
   as a limiting random walk sequence with step size s, and a white
   Gaussian noise sequence with mean zero and standard deviation s   */

   double RAN( br, rn, q )
   double br[28], rn[28], q;
{
   double ru[29], pi = 3.14159265, s;
   int i, step;

   s = sqrt(q);
   br[0] = 0.0;
   rn[0] = 0.0;
   for (i = 1; i < 27; ++i)
   {
      br[i]=0.0; /* generate uniform random numbers (0, 1)   */
      ru[i] = drand48(i);
      if ( ru[i] <= 0.5)
        step = - 1;
      else
        step = 1;

/* generate Brownian motion sample   */
      br[i] = br[i-1] + step*s;
   }

/* transform uniform random numbers into Gaussian random numbers  */
   for (i=1; i <= 26; i = i + 2)
   {
   rn[i] = s * sqrt (-2.0 * log ( ru[i] )) * cos ( 2.0 * pi * ru[i+1] );
   rn[i+1] = s * sqrt (-2.0 * log ( ru[i] )) * sin ( 2.0 * pi * ru[i+1] );
   }
}
```

```
/* Function to compute the unit step function */

   double U(a, b)
   double a, b;
{
   double u;
   if ((a - b) < 0.0)
     u = 0.0;
   else
     u = 1.0;

   return (u);
}

/* Function to compute the factorial of a number */

   long int fact( n )
   int n;
{
   long int result;

   if ( n==0 )
     result = 1;
   else
     result = n * fact ( n-1 );

   return (result);
}

/* Function to compute the function Z( ) in the Green's function */

   double Z( alpha, xi )
   double alpha, xi;
{
   double result;

   result = xi - (alpha * pow (xi, 3.) / fact(3))
            + (3. * pow(alpha, 2.) * pow(xi, 5.) / fact(5)):

   return (result);
}

/* Function ZPR( ) ("Z prime) for the evalculation of the Green's
  funtion                                                    */

   double ZPR( alpha, xi)
   double alpha, xi;
{
   double result;

   result = 1. - (alpha * pow(xi, 2.) / 2.)
        + (3. * pow(alpha, 2.) * pow(xi, 4.) / fact(4));

   return (result);
}
```

```
/* Function for the computation of the Green's function    */

    double G( lambda, xi, a, ald, alpha)
    double lambda, xi, a, ald, alpha;
{
    double g, f1, f2, f3, f4, g1, g2, F, I, H;

    f1 = Z(alpha, xi);
    f2 = Z(alpha, lambda);
    f3 = Z(alpha, a);
    f4 = ZPR(alpha, xi);

    F = (ald*( f1 - f3 )) / (f4*f3);
    I = -ald*f1 / f4;
    H = ald*f1 / (f4*f3);
    g1 = F*f2;
    g2 = I + H*f2;

    g = U(lambda, xi)*g2 + U(xi, lambda)*g1;

    return (g);
}

/* Function for the computation of the forward finite-difference
   approximation of the first derivative of an array */

double FD( elem, dx) double elem[28], dx;
{
    int j;

    for (j=0; j <= 26; ++j)
        elem[j] = (elem[j+1] - elem[j]) / dx;

    elem[27] = 0.0;
}

/* Function for the computation of the forward finite-difference
   approximation of the second derivative of an array */

double SD(elem, dx) double elem[28], dx;
{
    double dx2;
    int j;

    dx2 = dx*dx;
    for (j=0; j <= 25; ++j)
        elem[j] = (elem[j+2] - 2.0*elem[j+1] + elem[j]) / dx2;

    elem[26] = 0.0;
    elem[27] = 0.0;
}
```

```
/* main routine of the program */
main ()
{
   double alpha = 0.1801, xi, a = 2.7, dbar = 2.777;
   double alpha1=1.8e8, a1=2.7e-3, dx1=0.0001, b1=0.003217;
   double lambda, q=0.0002, fd[28], sd[28], rn[28], br[28];
   double tho=0.458, thn=0.086, b=0.167151, dx=0.1;
   double iint[28], lambda1, es=0.01;
   int i, j, l;
   double thpr[28], theta[28];

   printf("This file contains simulated sample values of moisture.0);
   printf("Initial values.0);

/* generate Brownian motion and White noise sample functions */

   RAN(br, rn, q);

/* compute first approximation of theta */
   lambda = 0.0;
   thpr[0] = tho;
   fd[0] = tho;
   sd[0] = tho;
   thpr[27] = thn;
   fd[27] = thn;
   sd[27] = thn;
   lambda1 = 0.0;
   for (j=1; j<= 26; ++j)
      {
      lambda1 = lambda1 + dx1;
      lambda = lambda + dx;
      thpr[j] = tho - b*Z(alpha, lambda);
      theta[j] = tho - b1*Z(alpha1, lambda1);
      fd[j] = theta[j];
      sd[j] = theta[j];
      printf("theta[%d]= %f    thpr[%d]= %f0, j, theta[j], j, thpr[j]);
      }

/* iteration loop */
 printf(" Subsequent approximations.0);

 for (i=0; i <= 1; ++i)
 {

/* compute first and second derivative of previous approximation */
      FD(fd, dx);
      SD(sd, dx);

/* compute the internal integral */

      iint[0] = 0.0;

      for (j=1; j <= 26; ++j)
         iint[j] = iint[j-1] + 2.*dx*(br[j]*sd[j] + rn[j]*fd[j]);

      iint[27] = iint[26];
```

```
/* compute the next approximation of theta */
   lambda = 0.0;
   thpr[0] = 0.0;
   for (l=1; l <= 27; ++l)
   {
      lambda = lambda + dx;
      thpr[l] = 0.0;

      xi = 0.0;
      fd[0] = 0.0;
      fd[27] = 0.0;
      for (j=1; j <= 26; ++j)
      {
         xi = xi + dx;
         fd[j] = G(lambda, xi, a, dbar, alpha);
      }

      FD(fd, dx);

      for (j=1; j <= 26; ++j)       /* external integral */
         thpr[l] = thpr[l] + fd[j]*iint[j]*dx;

      thpr[l] = -thpr[l];
      theta[l] = theta[l] + thpr[l];

      printf ("i=%d     thpr(%d)=%f0, i, l, thpr[l]);
   }

/* update for new iteration of theta  */

      for (j=0; j <= 27; ++j)
      {
         fd[j] = thpr[j];
         sd[j] = thpr[j];
      }

 }

/* print values of theta */

   printf(" Resulting sample function.0);
   theta[0] = tho;
   theta[27] = thn;
   for (j=0; j <= 27; ++j)
      printf ("theta(%d)=%f0, j, theta[j]);

}
```