Stochastic differential equations in r

Differential equations (DE) are mathematical equations that describe how quantity changes as a function of one or more variables (independent), often time or space. Differential equations play an important role in biology, chemistry, physics, engineering, economics and other disciplines. Differential equations can be separated into stochastic versus deterministic DEs. Problems can be divided into initial value issues versus boundary value issues. One also distinguishes ordinary differential equations from partial differential equations, differential algebraic equations and delays differential equations. All types of these DEs can be solved in the R. DE problem can be classified to be rigid or nonstiff; the previous type of problem is much more difficult to solve. The SIG dynamic model is a mailing list suitable for discussing the use of R to solve differential equations and other dynamic models such as individual or agent based models. This task view is created to give you an overview of the topic. If we forget something, or if a new package should be mentioned here, let us know. Stochastic Differential Equations (SDEs) In stochastic differential equations, an unknown quantity is a stochastic process. The sde package provides functions for simulation and inference for stochastic differential equations. This is the accompanying package of books by Iacus (2008). Pomp packs contain functions for statistical inference for partially observed Markov processes. The adaptive package and GillespieSSA implement Gillespie's exact stochastic simulation algorithm (direct method) and several approximate methods. The Sim.DiffProc package provides a function for simulating itô and Stratonovitch stochastic differential equations. Diffeqr packages can solve SDE problems using the DifferentialEquations.jl package from Julia's programming language. Ordinary Differential Equation (ODES) In ODE, an unknown quantity is a function of a single independent variable. Some packages offer to solve ODES. The odesolve package is the first to solve the usual differential equation in R. It contains two integration methods. This has been replaced by the deSolve package. The deSolve package contains several solvers to solve ODE, DAE, DDE and PDE. It can handle rigid and nonstiff problems. The odeintr package generates and compiles the C++ ODE solver quickly using rcpp and boost odeint. The R diffeqr package provides a seamless interface to the DifferentialEquations.jl package from Julia's programming language. It has unique high performance methods for solving ODE, SDE, DDE, DAE and more. Models can be written in R or Julia. This requires the installation of Julia's language. The pracma package implements several adaptive Runge-Kutta solvers such as ode23, ode23s, ode45, or Burlisch-Stoer algorithms to numeric numeric odes with higher accuracy. The rODE package (inspired by Gould, Tobochnik and Christian's book, 2016) aims to show physics, maths and engineering students how ODE solvers can be created with S4 R grades. Sundialr packages provide a way to call the 'CVODE' function of the ODE C solving library 'SUNDIALS'. This package requires that the ODE be written as an 'R' or 'Rcpp' function. Delay Differential Equations (DDEs) In DDE, derivatives at a given time are functions of variable values at the previous time. The dde package implements solvers for ordinary differential equations (ODE) and delay (DDE), where objective functions are written in R or C. Only suitable for non-rigid equations. Support is also included for iterational differences equations. The PBSddesolve package (originally published as ddesolve) includes solvers for DDE problems that are not rigid. The function in the deSolve package can solve the problem of rigid and not rigid DDE. Diffeqr packages can solve DDE problems using the DifferentialEquations.jl package from Julia's programming language. Pdes Partial Differential Equations (PDEs) are differential equations in which an unknown quantity is a function of several independent variables. Common classifications are elliptical equations (time-independent), hyperbolic (time-dependent and wave-like), and parabolic equations (time dependent and diffusive). One way to solve this is to rewrite PDES as a combined set of ODEs, and then use an efficient solver. ReacTran R package provides a function to convert PDEs into a set of ODES. Its main target is in the field of modeling "reactive transport", but it can be used to solve PDES of three main types. It provides a function to distinguish PDA on karteian, polar, cylindrical and round lattice. The deSolve package contains solvers specific to ode 1-D, 2-D and 3-D problems that vary in time as generated from PDES (e.g. by ReacTran). The root of theSolve package contains solvers optimized for 1-D, 2-D and 3-D algebraic problems resulting from (invariant time) PDEs. Thus it can be used to solve elliptical equations. Note that, to date, PDes in R can only be solved using limited differences. At some point, we expect that limited elements and spectral methods will be available. Differential Algebra Equations (DAES) Differential algebraic equations consist of differential and algebraic terms. An important feature of DAE is its differentiation index; the higher this index, the harder it is to crack DAE. The deSolve package provides two solvers, which can handle DAEs up to an index of 3. The diffeqr package can solve dae problems using the DifferentialEquations.jl package from Julia's programming language. BVP Boundary Value Issue (DVD) derivative solutions and/or conditions specified on independent variable limits. ReacTran package can solve the BVP to the reactive transport equation class. The diffeqr package can also complete the BVP using the DifferentialEquations.jl package from Julia's programming language. Other simecol packages provide an interactive environment for implementing and simulating dynamic models. In addition to the ED model, it also provides functions for grid-oriented, individually based, and particle diffusion models. The scaRabee package offers a framework for simulation and optimization of the Pharmacokinetics-Pharmacodynamic Model. In the FME package is a function for reverse modeling (fitting with data), sensitivity analysis, identification and Monte Carlo analysis of ED models. The nlmeODE package has a function for modeling mixed effects using differential equations. mkin provides a routine for fitting kinetic models with one or more country variables for chemical degradation data. The dMod package provides functions for generating reaction network ODEs, parameter transformations, observation functions, residual functions, etc. It follows the paradigm that derived information should be used for optimization whenever possible. The CollocInfer package implements collocation-inference for continuous and discrete-time stochastic processes. Root findings, balance, and stable ODEs analysis can be performed with rootSolve packages. The PBSmodelling package adds GUI functionality to the model. The cOde package supports automatic creation of dynamically linked code for deSolve packages (or built-in implementations of sundials cvode solver) from inline C embedded in R. Rodeo packages are object-oriented systems and code generators that create and compile efficient Fortran code for deSolve from models defined in stoichiomatry matrix notation. The ecolMod package contains numbers, datasets and examples from books on ecological modeling (Soetaert and Herman, 2009). In many cases, you may want to share the term noise throughout the system. This is known as non-diagonal noise. SDE's DifferentialEquations.jl tutorial explains how the matrix form of a diffusion term corresponds to the summation style of some Wiener processes. Basically, the row corresponds to the system where the term is applied, and the column is which noise term. So du[i,j] is the amount of noise due to the jth Wiener process applied to you[i]. We solved the Lorenz system with the following correlated noise:f &lt;- JuliaCall::julia_eval(function f(du,u,p,t) du[1] = 10.0*(u[2]-u[1]) du[2] = u[1]*(28.0-u[3]) - u[2] du [3] = u[3] = u[3] 1]*u[2] - (8/3)*u[3] end) g &lt;- JuliaCall::julia_eval(function g(du,u,p,t) du[1,1] = 0.3u[1] du[2,1] = 0.6u[1] du[1] du[2,1] = 20,6u[1] du[1] du[2,1] = 0.6 3.1] = 0.2u[1] du[1,2] = 1.2u[2] du[2,2] = 0.2u[2] du[3,2] = 0.3u[2] end) u0 &lt;- c(1,0,0,0,0,0pan &lt;- c(0.0,100.0) noise_rate_prototype &lt;- matrix(c(0,0,0,0,0,0,0,0,0,0,0,0,0,0), = 3, ncol = 2) JuliaCall::julia_assign(u0, u0) JuliaCall::julia_assign(tspan, tspan) tspan) noise_rate_prototype) prob &lt;- JuliaCall::julia_eval(SDEProblem(f, g, u0, tspan, p, noise_rate_prototype=noise_rate_prototype)) sole &lt;- de$solve(prob) udf &lt;- as.data.frame(t(sapply(sol$u,identity))) plotly::p lot_ly(uly x = ~V1, y = ~V2, z = ~V3, type = 'scatter3d', mode = 'lines') noise_corr Here you can see that the warping effect of the noise correlation is quite noticeable! Note that we apply JIT compilation as it is quite necessary for difficult stochastic examples. Description Argument Use Detailed Value Author Examples Generic interface to various method simulation solutions for stochastic differential equations. sde.sim(t0 = 0, T = 1, X0 = 1, N = 100, delta, drift, sigma, drift.x, sigma.x, drift.xx, sigma.xx, drift.t, method = c(euler, milstein, KPS, milstein2, cdist,ozaki,shoji,EA), alpha =0.5, eta = 0.5, pred.corr = T, rcdist = NULL, theta = NULL, model =c(CIR, VAS, OU, BS), k1, k2, phi, max.psi = 1000, rh, A, M=1) t0 origin time. T horizon simulation. The initial X0 value of the process. N number of simulation steps. M number of passes. delta time step simulation. drift drift coefficient: the expression of two variables t and x. sigma diffusion coefficient: the expression of two variables t and x. drift.x partial derivative of drift coefficient w.r.t. x: function two variables t and x. sigma.x partial derivative diffusion coefficient w.r.t. x: function two variables t and x. drift.xx second partial derivative of drift coefficient w.r.t. x: function two variables t and x. sigma.xx second partial derivative of diffusion coefficient w.r.t. x. x : function two variables t and x. drift.t partial derivative drift coefficient w.r.t. t: function two variables t and x. simulation method; See the details. alpha weight alpha from the predictor-corrector scheme. eta eta weight of the predictor-corrector scheme. pred.corr boolean: whether to apply predictor-correct adjustments; See the details. rcdist function which is a random number generator of conditional distribution processes; See the details. vector parameters for cdist; See the details. model model from which to simulate; See the details. k1 is lower bound to psi(x); See the details. k2 is bound up to psi(x); See the details. phi function psi(x) - k1. max.psi values over psi support to find the maximum. rh rejection function; See the details. A(x) is an integral of drift between 0 and x. The function returns a ts object N+1 long; that is, X0 and the simulated value of the new N if M=1. For M&gt;1, mts (multidimensional ts object) is returned, which means that the independent path of M is simulated. If the initial value of X0 is not of M length, the value is recycled to have the original vector of the correct length. If delta is not specified, delta = (T-t0)/N. If delta is specified, then the N value of the sde solution is generated and the T time horizon is adjusted to N* delta. The psi function is psi(x) = 0.5*drift(x)^2 + 0.5*drift.x(x). If any of drift.x, drift.xx, drift.t, sigma.x, and sigma.xx are not specified, then numeric derivation is attempted when needed. If sigma is not specified, it is assumed to be the function of constant 1. Simulation methods can be among: euler, KPS, milstein, milstein2, cdist, EA, ozaki, and shoji. There are no assumptions on the coefficient or on the cdist examined: the user is responsible for using the appropriate method for the simulation process object. This model is one in between: CIR: Cox-Ingersoll-Ross, VAS: Vasicek, OU Ornstein-Uhlenbeck, BS: Black and Scholes. There are no assumptions on theta coefficients examined: the user is responsible for using the appropriate. If this method is cdist, then the process is simulated according to a known conditional distribution. Rcdist random generator must be a function of n, the number of random numbers; dt, time lag; x, the process value at that time t - dt; and vector theta parameters. For the proper algorithm method EA: if it is missing k1 and k2 as well as A, rh and phi are calculated numerically by the function. x Restore invisible object ts Stefano Maria Iacus See Chapter 2 text. 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39# Ornstein-Uhlenbeck process set.seed(123) d &lt;- expression(-5 * x) s &lt;- expression(3.5) sde.sim(X0=10,drift=d, sigma=s) -&gt; X plot(X,main=Ornstein-Uhlenbeck) # Multiple trajectories of the O-U process set.seed(123) sde.sim(X0=10,drift=d, sigma=s, M=3) -&gt; X plot(X,main=Multiple trajectories of O-U) # Cox-Ingersoll-Ross process # dXt = (6-3*Xt)*dt + 2*sqrt(Xt)*dWt set.seed(123) d &lt;- expression(6-3*x) s &lt;- expression(2*sqrt(x)) sde.sim(X0=10,drift=d, sigma=s) -&gt; X plot(X ,main=Cox-Ingersoll-Ross) # Cox-Ingersoll-Ross using the conditional distribution rcCIR set.seed(123) sde.sim(X0=10 , theta=c(6, 3, 2), rcdist=rcCIR, method=cdist) -&gt; X plot(X, main=Cox-Ingersoll-Ross) set.seed(123) sde.sim(X0=10, theta=c(6, 3, 2), model=CIR) -&gt; X plot(X, main=Cox-Ingersoll-Ross) # Precise simulation set.seed(123) d &lt;- expression(sin(x)) d.x &lt;- expression(cos(x)) A&lt;- function(x) 1-cos(x) sde.sim(method=EA, EA, EA, delta=1/20, X0=0, N=500, drift=d, drift.x=d.x, A=A) -&gt; X plot(X, main=Drift periodically) Loading the required package: Package required MASS loading: stats4 Loading required package fda Loading: splines Loading package required: Matrix Attaching package: 'fda' The following object is masked from 'package package :graphics': matplot Loading required package: zoo Attaching package: 'zoo' The following object is masked from 'package:base': as. Date, as. Package Date.numeric sde 2.0.15 'Simulations and Conclusions for Stochastic Differential Equations With R Examples' Iacus, Springer NY, (2008) To examine the corrige errata book, sigma.x vignette(sde.errata) type is not provided, try symbolic derivation. sigma.x is not provided, attempting symbolic derivatives. sigma.x is not provided, attempting symbolic derivatives. T set to = 25.000000 k1 missing, trying to minimize numerical ... (k1=-0.500) k2 is missing, trying to maximize numerically... (k2=0.625) rejection rate: 0.215 sde documentation built on May 2, 2019, 8:30 a.m. a.m.