

Computational Methods in Economics - HW 1

Answers

1.1. Consider the function

$$\exp\left(-\left((-\ln(u))^a + (-\ln(v))^a\right)^{1/a}\right)$$

for $u, v \in [0, 1]$ and $a \in [1, \infty)$. The function has the property that it is increasing in both u and v and that $C \in [0, 1]$. Other properties of this function are

$$\begin{aligned}C(u, v, 1) &= uv \\C(u, v, \infty) &= \min(u, v) \\C(u, 0, a) &= C(0, v, a) = 0 \\C(u, 1, a) &= u \\C(1, v, a) &= v \\C(u, v, a) &= C(v, u, a)\end{aligned}$$

If one implements this function as written, numerical difficulties can arise as a gets large. Rewrite this function so it avoids this problem. Code this as a function in MATLAB. Be sure that it runs for compatible matrices of values of u and v (you can assume that a is scalar).

Test whether your implementation satisfies the above criteria for a grid of values of u and v and for a grid of values of a .

Answer.

For a large, overflow can be avoided by defining $\tilde{u} = -\ln(u)$, $\tilde{v} = -\ln(v)$, $\eta = \max(\tilde{u}, \tilde{v})$ and using

$$C(u, v) = \exp\left(\eta\left(1 + \left(\frac{\min(\tilde{u}, \tilde{v})}{\eta}\right)^a\right)^{1/a}\right)$$

This works for all values of a and hence can be used generally. See `copfunc.m` for an implementation. Run `coptest.m` to test the properties of this function.

The code for the copula function is

```
% COPFUNC Computes the value of an unnamed copula
% USAGE
%   C=copfunc(u,v,a);
% INPUTS
```

```

% u,v : values of realizations on [0,1]
% a   : parameter value on [1,inf)
% OUTPUT
% C    : copula value
%
% The copula implemented here is defined as
% C = exp(-((-ln(u))^a+(-ln(v))^a)^(1/a))

function C=copfunc(u,v,a)
% error checks
u(u<0)=0; u(u>1)=1; v(v<0)=0; v(v>1)=1;
if any(a<1), error('Inappropriate value for a'); end
% compute C
warning off
minuv = min(u,v);
maxuv = max(u,v);
eta   = real(log(minuv));
C     = exp(eta.*(1+(real(log(maxuv))./eta).^a).^(1./a));
% Handle special cases
C(minuv<=0) = 0;
ind         = maxuv>=1;
C(ind)     = minuv(ind);
warning on

```

A few points are worth noting here. First, there can be divide by zero and log of 0 warnings, so the warnings are turned off (this is to avoid annoying messages). The cases in which these issues arise are dealt with at the end. The possibility of negative values of u or v is also addressed. MATLAB will compute the log of a negative number and the result is complex. To avoid a complex result, only the real part of the (potentially) complex number is kept after the log function is called. Also note how variables are used to avoid performing the same computations multiple times. This uses more memory but will result in faster execution.

The expression $\exp(\ln(x)b)$ is mathematically equivalent to x^b . Although the first form may seem to involve three operations (calling the log function, multiplying and calling the exponential function) whereas the latter only makes a call to a power operator, there is generally not much difference in the two expressions computationally. The reason is that, except possibly for special values of b (like 1 or 2), the power operator typically uses the log and exponential functions to perform the calculation (this may not be true for all hardware, however).

- 1.2. Continuous time stochastic processes are generally defined using stochastic differential equations such as

$$dS = \mu(S)dt + \sigma(S)dz$$

where μ and σ are functions of S and z is a standard Weiner process.

One way to think of such a process is as a limit of a discrete time process as the time increment between periods (Δ) goes to 0, leading to the approximation

$$S_{t+\Delta} \approx S_t + \mu(S_t)\Delta + \sigma(S_t)\sqrt{\Delta}e_t$$

where e_t is a standard Gaussian random variable (i.e., $e_t \sim N(0, 1)$), with e_t independent of e_s for all $t \neq s$.

The simplest case of this (so-called Brownian motion) occurs when μ and σ are constants. Write a MATLAB function with the syntax

```
S=brownian(mu,sigma,S0,Delta,n),
```

where S_0 is an initial value and n is the number of time steps. This function should accept 5 scalar arguments and return an $1 \times (n + 1)$ vector (it should include S_0 as its first element).

You will need to use the function `randn` and will need to know how to define loops to write this function.

Write a script function that demonstrates the use of this function by generating 1000 time paths with $\mu = 0.1$, $\sigma = 0.2$, $S_0 = 1$, $\Delta = 0.01$ and $n = 500$. The script should also verify that the sample mean and variance over 1000 time paths are approximately equal to their expected values of $S_0 + \mu n\Delta$ and $\sigma^2 n\Delta$.

For extra credit, write the function so it accepts μ and S_0 as d vectors and σ as a $d \times d$ matrix and returns a $d \times (n + 1)$ matrix. Write this so it doesn't loop over the d elements.

Answer.

The following code handles both scalar and vector arguments.

```
% BROWNIAN Generated Brownian motion time paths
% USAGE
% S=brownian(mu,sigma,S0,Delta,n);
% INPUTS
% mu : dx1 vector of drift coefficients
% sigma : dxd matrix of diffusion coefficients
% S0 : dx1 vector of starting values
```

```

% Delta : size of the time step (scalar)
% n      : number of time steps (scalar)
% OUTPUT
% S0     : dx(n+1) matrix of time path values

```

```

function S=brownian(mu,sigma,S0,Delta,n)
d=length(mu);
S=randn(d,n+1);
S(:,1)=S0;
mu=mu*Delta;
sigma=sigma*sqrt(Delta);
for i=1:n
    S(:,i+1)=S(:,i)+mu+sigma*S(:,i+1);
end

```

The script file to check the mean and variance of the final values of the sample paths is given below.

```

% Script to demonstrate use of BROWNIAN
% Parameter values
mu=0.1;      % drift coefficient
sigma=0.2;   % diffusion coefficient
S0=1;       % starting value
Delta=0.01; % time increment
n=500;      % number of time steps

k=1000;     % number of replications

Send=zeros(k,1); % initialize vector for replications
% loop over replications
for i=1:k
    S=brownian(mu,sigma,S0,Delta,n);
    Send(i)=S(end);
end

% compare sample and population values
disp('Compare sample mean and variance with expected values')
disp([mean(Send) var(Send)])
disp([S0+mu*Delta*n sigma*sigma*Delta*n])

```