```
/*    mcmix1.c
      A Monte Carlo CT option pricing routine for the GARCH diffusi
on
      Uses the mixing theorem
      4/7/99
      Notes:
            1. uses ran1 from NR p. 280
            2. handles correlated log utility case
            3. assumes rf=dividend=0
            4. antithetic variate approach
            5. uses y = ln V(t)
            6. Calculates put only, then uses parity for call

      Remember: CTRL-P to insert a true TAB in the makefile
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)

float ran1 (long *idum)
{
int j;
long k;
static long iy=0;
static long iv[NTAB];
float temp;

if(*idum <= 0 || !iy)
  {
  if (-(*idum) < 1)
       *idum = 1;
```

```
  else
     *idum = -(*idum);
  for(j = NTAB+7; j >= 0; j--)
    {
    k = (*idum)/IQ;
    *idum = IA*(*idum-k*IQ)-IR*k;
    if (*idum < 0)
     *idum += IM;
    if (j < NTAB)
       iv[j] = *idum;
    }
    iy = iv[0];
  }
k = (*idum)/IQ;
*idum = IA*(*idum-k*IQ)-IR*k;
if (*idum < 0)
  *idum += IM;
j = iy/NDIV;
iy = iv[j];
iv[j] = *idum;
if ((temp = AM*iy) > RNMX)
  return (RNMX);
else
  return (temp);
 }


/* unit normal deviates  NR p 289 */
float gasdev(long *idum)
{
float ran1(long *idum);
static int iset = 0;
static float gset;
float fac,rsq,v1,v2;

if(iset == 0)
  {
  do {
      v1 = 2.0*ran1(idum)-1.0;
      v2 = 2.0*ran1(idum)-1.0;
      rsq = v1*v1+v2*v2;
      }
  while (rsq >= 1.0 || rsq == 0.0);
```

```
  fac = sqrt(-2.0*log(rsq)/rsq);
  gset = v1*fac;
  iset = 1;
  return (v2*fac);
  }
else
  {
  iset = 0;
  return (gset);
  }
 }


#define max(A,B)   ((A) > (B) ? (A) : (B))
#define min(A,B)   ((A) < (B) ? (A) : (B))

 /* cumulative normal function */
 /* source: GR 26.2.17 abs error  < 7.5 10**(-8) */
 double cumnormal(double x)
 {
 double t,ans;
 float C1 =.39894228040143;   /* 1/sqrt(2 Pi) */

 t = 1.0/(1.0 + .2316419*fabs(x));

 ans = 1.0 - exp(-.5*x*x)*C1*
      t*(.31938153 + t*(-.356563782 + t*(1.781477937+
      t*(-1.821255978 + 1.330274429*t))));

 return(x >= 0 ? ans : 1.0 - ans);
 }


/* Black-Scholes call option, no dividends */
double bsvalue(double S, double X, double r, double sig, double t)
{
double d1,d2,ans;
if(t == 0 || sig == 0)
  return max(S-exp(-r*t)*X,0);
d1 = (log (S/X) + (r+.5*sig*sig)*t)/(sig*sqrt(t));
d2 = d1 - sig*sqrt(t);
ans = S*cumnormal(d1)-exp(-r*t)*X*cumnormal(d2);
return ans;
```

```
}


#define MAX_ITERATIONS   15
#define ISIG_ERR 1.0e-10
#define PI  3.141592654
/* Black-Scholes implied sigma, no dividends */

double bs_impsigma(double S, double X, double r, double price, dou
ble t)
{
double d1,sigold,sig,dsig;
int i;

if(price <= max(S-exp(-r*t)*X,0))
  {printf("Implied sigma: fatal error -- option price too low\n");
   return -1;
  }

sigold = .10;

for (i = 1; i <= MAX_ITERATIONS; i++)
 {d1 = (log (S/X) + (r+.5*sigold*sigold)*t)/(sigold*sqrt(t));
  dsig = S*sqrt(t/(2.0*PI))*exp(-.5*d1*d1);
  sig = max(0.0,sigold - (bsvalue(S,X,r,sigold,t)-price)/dsig);
  if(fabs(sig-sigold) < ISIG_ERR)
    return(sig);
  sigold = sig;
  }

printf("Implied sigma: warning: no convergence: returning last
iteration\n");
  return sig;
}


long seed = -1;
long ns = 1;


/* parameters measured on annual basis */
double omega = .09;
double theta = 4.0;
```

```
double ksi = 1.0;
double rho;


main(argc,argv)
int argc;
char *argv[];
{
int p,pr;
long T,i,m,n,rpts;     /* T = time measured in days */
long tsteps;
double z1;             /* unit normal variate */
double x1,x2;          /* x = ln S(t) */
double y1,y2;          /* y = ln V(t) */
double v1,v2;          /* variance = sig^2 */
double iv1,iv2;        /* integrated volatility */
double q1;
double sig1,sig2;
double s2;             /* initial volatility (sig^2) */
double dt,halfdt,sqrtdt;   /* time step (years), .5 dt, sqrt(dt) *
/
double dpy = 250.0;    /* days per year */
double sf;             /* final stock price at expiration */
double pbar;     /* final Monte Carlo put  price estimate */
double vpbar;
double term,ans;
double pbarerr;
double rho2;
double risk_adj,theta_adj;
double isig;
double t_yr;           /* time in years */
int A = 1;             /* CPRA parameter */
double ymin = -50.0;   /* min log volatility */
double xmin = -50.0;   /* min log stock price */
double ymax = 100.0;   /* max log volatility */
double bs;             /* Black-Scholes price */
double sigeff;         /* effective vol. for B-S formula */
double c1,c2,c3,c4;    /* time-related constants */
long ck1,ck2;          /* clock counters */


if (argc != 7)
  {
```

```
  printf("Usage: mcmix1 power T m power_rpts v0 rho\n");
  return(0);
  }
else
  { p = atoi(argv[1]);
    T = atol(argv[2]);
    m = atoi(argv[3]);
    pr = atoi(argv[4]);
    s2 = atof(argv[5]);
    rho = atof(argv[6]);
    n =  pow(10.0,(double) p);
    rpts = pow(10.0,(double) pr);

    if(n > 1000000)
      printf("WARNING: more than 1 million simulations\n");
    printf("power=%d n=draws=%d T=days=%d m=steps per day=%d\n",p,
n,T,m);
  }

rho2 = sqrt(1.0- rho*rho);

printf("omega=%e theta=%9.6f ksi=%9.6f rho=%9.6f\n",
        omega,theta,ksi,rho);
printf("CPRA risk aversion parameter A = %d\n",A);


 theta_adj = theta + .5*ksi*ksi;
 if(A == 1)
   risk_adj = 0;
 else
    risk_adj = -(1.0-A)*rho*ksi;
 printf("risk_adj =%e theta_adj = %e\n",risk_adj,theta_adj);


 /* some time-related constants */
 dt = 1.0/(m*dpy);       /* measured in years */
 t_yr = T/dpy;
 sqrtdt = sqrt(dt);
 halfdt = .5*dt;
 c1 = .5*rho*rho*dt;
 c2 = rho*sqrtdt;
 c3 = ksi*sqrtdt;
 c4 = rho2*sqrt(t_yr);
```

```
printf("s2 = %12.8f dt=%12.10f years\n",s2,dt);

bs =  bsvalue(100.0,100.0,0,sqrt(s2),t_yr);
printf("B-S call = %12.8f\n",bs);

printf("the first deviate = %10.6f\n",gasdev(&seed));

pbar = vpbar = 0;
ck1 = clock();

for(i = 1; i <= n; i++)
  {
  x1 = x2 = 0;
  v1 = v2 = s2;
  sig1 = sig2 = sqrt(s2);
  y1 = y2 = log(s2);
  iv1 = iv2 = 0;


  for(tsteps = 1; tsteps <= T*m; tsteps++)
    {
     z1 = gasdev(&ns);

     x1 = max(x1 - c1*v1 + c2*sig1*z1,xmin);
     x2 = max(x2 - c1*v2 - c2*sig2*z1,xmin);


     if(v1 > 0)
       {q1 = (omega/v1 - theta_adj + risk_adj*sig1)*dt;
        iv1 += v1*dt;
        y1 = min(y1 + q1 + c3*z1,ymax);
        v1 = exp(y1);
        }
     else
        {v1 = omega*dt;
         if(v1 == 0)
           y1 = ymin;
         else
           y1 = log(v1);
        }
     sig1 = sqrt(v1);

     if(v2 > 0)
```

```
        {q1 = (omega/v2 - theta_adj + risk_adj*sig2)*dt;
         iv2 += v2*dt;
         y2 = min(y2 + q1 - c3*z1,ymax);
         v2 = exp(y2);
         }
      else
         {v2 = omega*dt;
          if(v2 == 0)
            y2 = ymin;
          else
            y2 = log(v2);

         }
      sig2 = sqrt(v2);

      }

    sf = exp(x1);
    sigeff = rho2*sqrt(iv1);
    term = .5*(bsvalue(sf,1.0,0,sigeff,1.0)-sf+1.0);

    sf = exp(x2);
    sigeff = rho2*sqrt(iv2);
    term += .5*(bsvalue(sf,1.0,0,sigeff,1.0)-sf+1.0);

    pbar += term;
    vpbar += term*term;

   if(i % rpts == 0)
          printf("i=%d put=%6.2f\n",i, 100.*pbar/i);

   }
   pbar = pbar/n;
   vpbar = (vpbar/n - pbar*pbar)/n;
   pbarerr = 100.0*sqrt(vpbar);
   pbar = 100.0*pbar;
   ck2 = clock();


 printf("After %d runs option value (put) = %10.6f stderr=%10.6f\n
",
         n,pbar,pbarerr);
 isig =  100.0*bs_impsigma(100.0,100.0,0,pbar,t_yr);
```

```
 printf("Years = %10.6f Implied sigma = %10.6f percent\n\n",t_yr,i
sig);
 printf("\n (runtime=%8.2f secs)\n",(float)(ck2-ck1)/CLOCKS_PER_SE
C);

 }
```