MS&E 345
Final Project Report
Pricing High Dimensional American Options

Ajaykumar Rajasekharan (05335598)

March 17, 2008

1 Introduction

Methods for pricing American options have been increasingly important with the development of more and more complex contracts, for eg. valuations of basket options, swaptions etc. In high dimensions the grid techniques (such as finite difference) becomes intractable as they encounter the curse of dimensionality. Consequently, Monte Carlo techniques have become attractive and a lot of techniques have been developed [1, 2, 3, 4, 5].

This project intends to review and implement two different approaches to price the American options (in high dimensions), namely the Least-Squares Method (LSM) of of Longstaff & Schwartz [1] and the Irregular Grid Method (IGM) in Berridge & Schumacher [4, 5], and finally compare their performance on a benchmark problem of Geometric Average Option described in Berridge & Schumacher [5].

2 General Summary of Methods

There are in general two different types of methods used to value american options, namely the traditional methods and simulation based methods.

2.1 Traditional Methods

The following are the two most commonly used traditional methods

- Lattice Methods (e.g. Binomial Tree) : These methods use a dynamic programming by working up backwards in time by discounting backwards.
- Finite Difference, Finite Element etc.. : These methods essentially solve the semi-discrete version (discretize separately in space and time – method of lines) of the following complementarity problem for the price of the derivative v

$$\begin{aligned} \frac{\partial v}{\partial t} + \mathcal{L}v &\leq 0 \\ v - \psi &\geq 0 \end{aligned}$$

$$(\frac{\partial v}{\partial t} + \mathcal{L}v)'(v - \psi) \quad = \quad 0 \tag{1}$$

for $(x, s) \in \mathbb{R}^N \times [t, T]$ with terminal condition $v(., T) = \psi(., T)$. In the above equation, $\mathcal{L}$ is the generator (Black-Scholes pde) and ψ is the payoff function ($(K - S)^+$ in the case of standard American Put).

Both these methods are good for early exercise computations but are limited by the number of stochastic factors $O(N^d)$, where, N is the number of grids (curse of dimensionality).

2.2 Simulation (Monte-Carlo) Based Methods

The main features of the Monte-Carlo approach can be summarized as follows:

- Requires to solve a dynamic optimization problem by dynamic programming. For an American option, this implies that the optimal exercise strategy is determined by choosing between the current exercise value and the conditional expectation of the payoff from continuing to keep the option alive. Hence, for an intrinsic value $I_i = max\{K - S_i, 0\}$ of an American Put, the price at time $V_i(S_i)$ can be computed as

$$V_i(S_i) = \max\left\{I_i(S_i), \mathrm{E}_i^Q\left[e^{-r\cdot\Delta t}V_{i+1}(S_{i+1})|S_i\right]\right\} \tag{2}$$

  where, S_i is the price of the underlying stock S_i, and the expectation is with respect to the risk-neutral measure Q.

- The direct generalization of the above is not feasible for multiple stochastic factors, due to an exponential increase in the number of state variables (curse of dimensionality). Hence one needs to find a good approximation of the expected value function $\mathrm{E}_i^Q\left[e^{-r\cdot\Delta t}V_{i+1}(S_{i+1})|S_i\right]$, which always provides a lower bound because of the discretization in time (i.e. not computing in continuous time).

3 Least-Squares Approach of Longstaff & Schwartz

Main Idea : In their approach, Longstaff & Schwartz estimate the conditional expectation $\mathrm{E}_i^Q\left[e^{-r\cdot\Delta t}V_{i+1}(S_{i+1})|S_i\right]$ described in the previous section by first regressing the subsequent realized cash flows from continuation on a set of basis functions of the values of the relevant state variables, for e.g.

$$\mathrm{E}_i^Q\left[e^{-r\cdot\Delta t}V_{i+1}(S_{i+1})|S_i\right] \approx a_1 + a_2 S_i + a_3 S_i^2 \tag{3}$$

Then they use the fitted value as an efficient unbiased estimate of the conditional expectation function to accurately estimate the optimal stopping rule for the option.

3.1 Algorithm

1. Generate multi-dimensional correlated Geometric Brownian Motion sample paths with values at each time discretization $1, ..., n$

2. At each time slice moving backwards from $n-1, n-2, ..., 1$
   - Select all the sample paths that are in the money for continuation
   - Regress the payoff of continuation with the present values of the stock to obtain the conditional expectation function
   - Use the conditional expectation function to obtain the expected continuation value
   - If the present payoff (intrinsic value) is more than then expected continuation, the exercise the option, else wait further to exercise
3. After the above step, one obtains information regarding time to exercise the option for each sample path. Use this information and discount the payoff value for each sample path to the starting point using each of their corresponding time of exercise
4. Average the time 0 value for each sample path to obtain the American Put price

3.2 Advantages and Disadvantages of using this Method

The following are the advantages and disadvantages of the Least-Squares Method.

- Advantages
  - Quick solution obtainable (runs in MATLAB with 100000 sample paths, 50 time slices and 5 dimensions in less than 5 minutes)
  - Easy to code
- Disadvantages
  - Choice of basis function in the regression is tricky
  - Can get into numerical issues with singularity while solving the least-squares problem
  - Gives always a lower bound for the American Put value. Hence one needs to complement this value with the dual Monte-Carlo approach of L.C.G. Rogers [3] that gives and upper bound for the put value. This approach is based on simulating the paths of the options payoff by judiciously choosing a Lagrangian martingale $M(s)$. Taking the pathwise maximum of the payoff less the martingale provides the upper bound for the price of the option $V_i(S_i)$ i.e.

$$V_i(S_i) \quad = \quad \min\left\{I_i(S_i), \mathrm{E}_i^Q\left[\max\left(e^{-r\cdot\Delta t}V_{i+1}(S_{i+1}) - M(s)\right)|S_i\right]\right\}$$

4 Irregular Grid Method of Berridge & Schumacher

Main Idea : This method is a sandwitch between the sampling methods and the finite difference method for valuing American Put options. The discretization of the state space if first performed using Quasi Monte Carlo (QMC) trials with respect to the transition density of the process at expiry. The inifinitesimal generator matrix is then used as an approximation to the partial differential operator (PDO) on this grid, which is generated using some local consistency conditions. These conditions ensure that the approximating Markov chain has a local mean and variance that match those of the continuous process.

So, this method involves solving Eq. (1) in a semi-discrete fashion by approximating the generator $\mathcal{L}$ with and approximate generator A on thre state space using grid $\mathcal{X}$ without doing finite difference. After this approximation the complementarity problem reduces to

$$\begin{aligned} \frac{dv}{dt} + Av &\leq 0 \\ v - \psi &\geq 0 \\ (\frac{dv}{dt} + Av)'(v - \psi) &= 0 \end{aligned} \tag{4}$$

The matrix A is then approximated by solving the local consistency conditions.

4.1 Generating Irregular Grid ($\mathcal{X}$)

The dynamics of the stock process $\mathbf{S}$ which is a correlated Brownian Motion, in log coordinates $\mathbf{X} = log(\mathbf{S})$ is given by

$$d\mathbf{X} = \Big(\mathbf{r} - \frac{1}{2}\mathrm{d}iag(\Sigma)\Big)dt + \mathbf{R}'d\mathbf{W} \tag{5}$$

where, Σ is the covariance matrix, $\mathbf{R}$ is the first part of the Cholseky decomposition and $\mathbf{r}$ is the risk-free interest rate.

The gird is generated to be centred about $x_t + \Big(\mathbf{r} - \frac{1}{2}\mathrm{d}iag(\Sigma)\Big)(T-t)$. Hence, one way to generate this grid is to choose a grid density that is normal with respective mean and covariance matrix give by

$$\mu_g = x_t + \Big[\mathbf{r} - \frac{1}{2}\mathrm{d}iag(\Sigma)\Big](T-t) \tag{6}$$

$$\Sigma_g = \alpha\Sigma(T-t) \tag{7}$$

where, the parameter α is at least 1 for the grid ot be well adapted. The i^{th} grid point is hence given by

$$x_i = \mu_g + R_g'\Big(\Psi^{-1}(h_{i,1}).....\Psi^{-1}(h_{i,N})\Big)' \tag{8}$$

where, $h_{i,i}$ is the quasi random Halton sequence. For multiple runs, this quasi random sequence is randomized using

$$H_j = \{h_i + U_j \text{ mod } 1\} \tag{9}$$

where, U_j are independent uniform random vectors on $[0,1]^N$ and x mod 1 denotes the fractional part of x. A two-dimensional grid generated using the above procedure is shown in Fig. 1

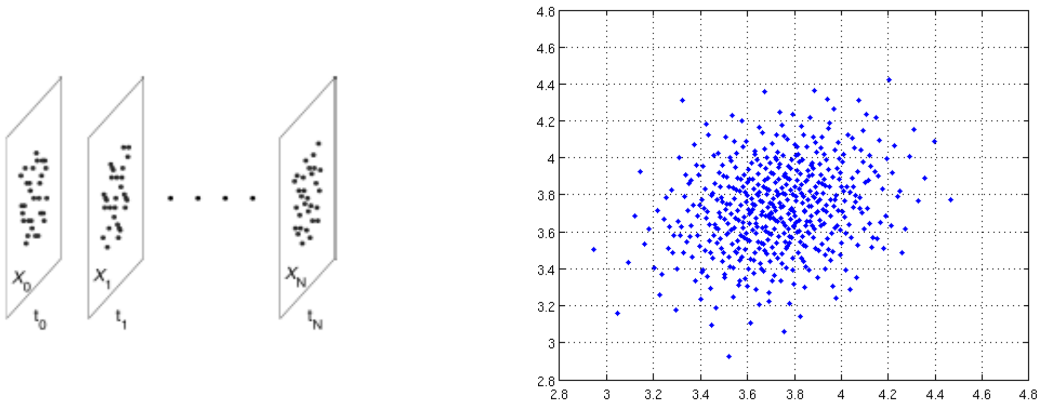

Figure 1: A QMC irregular grid in 2-dimensions (log coordinates)

4.2 Generation of Infinitesimal Generator (A)

The approximate infinitesimal generator matrix A can be computed by solving for the consistency conditions at each grid node i namely

$$\begin{aligned} \max f_j \cdot a_{i,j} \quad s.t.& \\ \Sigma(x_i) &= \sum_{j=1, j\neq i}^{n} (x_j - x_i)(x_j - x_i)' a_{i,j} \\ \mu_{RN} &= \sum_{j=1, j\neq i}^{n} (x_j - x_i) a_{i,j} \\ 1 &= \sum_{j=1, j\neq i}^{n} a_{i,j} \\ a_{i,j} &\geq 0, \quad i \neq j \\ a_{i,i} &= -\sum_{j\neq i}^{n} a_{i,j} \end{aligned} \tag{10}$$

where, $a_{i,j}$ is the (i,j)th entry of of the infinitesimal generator A. The above consistency conditions are just the method of moments (MOM) matches for the first and second moments of the continuous process. The objective function f_j is chosen as a distance function so as to ensure that the maximum probability of transition for a node i is to its nearby neighbors.

4.3 Boundary Condition

There are points in the above grid where the linear program (LP) for the local consistency conditions is not feasible. These points can be interpreted as implied boundary points, which mean that it is hard to transition from these points away to other points. Hence they are best modeled using absorbing boundary conditions. Another, interpretation, could be that these points are way in the tails of the distribution, so they are less likely to occur, and can be in principle eliminated from the grid. But, handling these points is very important step for the method to work.

4.4 Algorithm

- Choose grid size n
- Generate a constant Quasi Monte-Carlo QMC grid χ
- Compute generator matrix A
- Choose a time integrator (used second order implicit Crank-Nicholson)
- Solve the linear complementarity problem

4.5 Advantages and Disadvantages of using this Method

- Advantages
  - Since this method is a semi-discrete (finite-difference) style method, convergence is obtained by increasing the number of grid points and by increasing the order of the time-integration
  - This method uses an optimum grid positioning unlike finite difference which uses a constant box grid
  - Re usability of the infinitesimal generator A is a major highlight of this method
- Disadvantages
  - To obtain a constant grid, the grid is constructed with the probability distribution at the final time step. This is questionable as the state-space represented by the grid does not involve all the transitioning states at all time steps
  - Handling boundary conditions (i.e. points where no feasible solution to the LP) is not clear. Also one has to solve the LP to find out the infeasible points, and hence cannot be determined before hand.

5 Numerical Experiments

Both the Least-Square's Method (LSM) and the Irregular Grid Method (IGM) have been implemented in MATLAB (see code attached), and were used to price of a Geometric Average American Put benchmark test case described in the next subsection. In this section the results obtained from both methods are presented and then compared with the results presented in Berridge & Schumacher [5].

5.1 Benchmark - Geometric Average Option

A geometric average put option written on d assets with the following payoff function is considered here

$$\psi(s) = \Big(K - \Big(\prod s_i\Big)^{1/d}\Big)^+ \tag{11}$$

This option is equivalent to a standard put option on an asset with starting value $\exp \overline{X}_0$, strike price K, risk free rate r and continuous dividend stream $\delta = \frac{1}{2}\Big(\frac{1}{d}\sum \sigma_i^2 - \overline{\sigma}^2\Big)$

The values of various parameters in the simulation are $S_i = 40$ for all i, $K = 40$, $r = 0.06$, $\sigma_i = 0.2$ and correlation coefficient is $\rho_{ij} = 0.25$ for $i \neq j$.

5.2 Results

The following table summarizes the results obtained by using both the LSM and IGM to solve the Geometric Average Option. For the LSM a total number of $100,000$ sample paths were generated and 50 time slices were used, while for the IGM, 10 separate grids with a total number of 2000 randomized QMC grid points were generated and then the results averaged.

Table 1: American Put Sol

Dimension	Exact Sol	Longstaff-Schwartz	Berridge-Schumacher
2	1.7787	1.6051	—-
		(0.0122)	
3	1.5597	1.3903	1.6412
		(0.0106)	(0.0072)
4	1.4392	1.2721	1.4067
		(0.0098)	(0.0255)
5	1.3625	1.1946	1.1305
		(0.0093)	(0.0397)

The values in the bracket in the table are the 95% confidence intervals. Its seen from the table that the LSM approaches the optimal solution described in the paper from below, but is not up to a good level of accuracy. With the IGM, no correct convergence pattern is observed, although the trend in the solution is achieved. This could be due to incorrect solution of the LP or due to incorrect handling of the implied boundary points.

6 Conclusion

The following are the important conclusions of this report.

- The project looked at using two methods for pricing high dimensional American Options, namely – Least-Squares Method and the Irregular Grid Method
- Both have advantages and disadvantages as described in the previous sections

- The methods were coded and applied to obtain prices for a geometric average option. It was observed that the IGM didn't give accurate results and had no convergence pattern, with can be due to the non-optimal solution of LP with MATLAB or some innate features in the grid generation process or incorrect handling of boundary conditions

References

[1] F. A. Longstaff and E. S. Schwartz, Valuing American Option by Simulation : A Simple Least Squares Approach, *Review of Financial Studies*, **14**(1), 113-147, 2001. 1990.

[2] M. Broadie and P. Glasserman, A Stochastic Mesh Method for Pricing High-Dimensional American Options, *Journal of Computational Finance*, **7**(4), 39-88, 2001.

[3] L.C.R. Rogers, Monte Carlo Valuation of American Options, *Mathematical Finance*, **12**, 271-286, 2002.

[4] S. J. Berridge and J.M. Schumacher, An Irregular Grid Approach for pricing High-Dimensional American Options, *CentER Discussion Paper, 99*, 2002.

[5] S. J. Berridge and J.M. Schumacher, Pricing High-Dimensional American Options using Local Consistency Conditions, *CentER Discussion Paper No. 2004-19. Available at SSRN: http://ssrn.com/abstract=557745* 2004.

Matlab Codes

Longstaff-Schwartz Least Squares Method

```
% This is a higher dimensional modification of the 1-dimensional code
% presented in P. Brandimarte (Numerical Methods in Finance), 2002

NDIMS = 5;       % No of dimensions
NRepl = 100000;  % No of sample paths
NSteps = 50;     % No of time slices

randn('state',0);

% problem parameters
S0 = 40; % spot value
sigma = 0.25 *0.04* ones(NDIMS,NDIMS); % variance matrix
for i = 1:NDIMS
    sigma(i,i) = 0.04;
end
r = 0.06; % interest rate
T = 1; % maturity
t = 0;
dt = (T-t)/NSteps; % time step
X = 40; % strike

% discount rates over different time intervals
discountVet = exp(-r*dt*(1:NSteps))';

% regression parameters (cubic function)
a = zeros(4,1);

% generate sample paths
SPaths = GBMMUL(NDIMS,NSteps,S0,r,sigma,T,NRepl);

SPaths(:,1,:) = []; % get rid of starting prices

%
CashFlows = max(0, X - prod(SPaths(:,NSteps,:),3).^(1/NDIMS));

% first set exercise time at expiration for convenience
ExerciseTime = NSteps*ones(NRepl,1);


for step = NSteps-1:-1:1
```

```
    InMoney = find((prod(SPaths(:,step,:),3).^(1/NDIMS)) < X);
    XData = prod(SPaths(InMoney,step,:),3).^(1/NDIMS);
    RegrMat = [ones(length(XData),1), XData, XData.^2, XData.^3];
    YData = CashFlows(InMoney) .* discountVet(ExerciseTime(InMoney) - step);
    a = RegrMat \ YData;
    IntrinsicValue = X - XData;
    ContinuationValue = RegrMat * a;
    Exercise = find(IntrinsicValue > ContinuationValue);
    k = InMoney(Exercise);
    CashFlows(k) = IntrinsicValue(Exercise);
    ExerciseTime(k) = step;
end % for

price = mean(CashFlows.*discountVet(ExerciseTime))
error = 1.96 * std(CashFlows.*discountVet(ExerciseTime)) / sqrt(NRepl)


--------------------------------------------------------------------------------

% This routine generates NRepl samples of correlated multidimensional
% geometric brownian motions (GBM)

function [S] = GBMMUL(dim,p,S0,r,Sigma,T,NRepl)

dt = T/p;

A = chol(Sigma)';

aa(1,1,:) = (r-(diag(Sigma).^2)/2)*dt;

S = repmat(aa,[NRepl p 1]) + reshape((A*randn(dim,NRepl*p)*sqrt(dt))',NRepl,p,dim);
S = S0*exp(cumsum(S,2));
S =  [S0*ones(NRepl,1,dim) S];


--------------------------------------------------------------------------------
```

Berridge-Schumacher Irregular Grid Method

```
% This is the driver routine for the American Put pricing by computing the
% approximate infinitesimal generator

rand('state',1);
p = 10; % No of QMC grids

for kkk = 1:p

  NUMPTS = 2000;
  NDIMS = 5;
  NTIME = 50;

  x_t = log(40) * ones(NDIMS,1);
  alpha = 1;
  sigma = 0.25 * 0.04 * ones(NDIMS,NDIMS);
  for i = 1:NDIMS
     sigma(i,i) = 0.04;
  end
  r = 0.06 * ones(NDIMS,1);
  q = 0 * ones(NDIMS,1);
  T = 1;
  t = 0;
  dt = (T-t)/NTIME;
  K = 40;

  % Step-1 : Create the irregular grid
  G = irrgrid(x_t,r,q,sigma,T,t,alpha,NUMPTS,NDIMS); % S in log coordinates

  % Step-2 : Generating the Infinitesimal Generator P
  generateInfinitesimalGen(G,r,sigma,NUMPTS,NDIMS);

  % Step-3 : Solving complementarity problem to determine price
  load 'delPlace.mat'
  load 'infGen.mat'

  % Payoff
  GG = max(K - prod(exp(G)).^(1/NDIMS),0)';

  % Treating boundary conditions
  GG(II) = [];
  P(II,:) = [];
  P(:,II) = [];
  NUMPTS = NUMPTS - length(II);
  for ii = 1:NUMPTS
      P(ii,ii) = P(ii,ii) - sum(P(ii,:));
  end

  % European pricing (Crank-Nicholson time integration)
  v  = GG;
  Iden = sparse(NUMPTS,NUMPTS);
  Iden = Iden + (1/dt) * eye(NUMPTS);
  P = 0.5*P;
  for i = NTIME-1:-1:1
      v = (Iden-P)\((Iden+P)*v);
  end
  eursuper(kkk) = mean(v);

  % American pricing (Crank-Nicholson time integration)
  v = GG;
  Iden = sparse(NUMPTS,NUMPTS);
  Iden = Iden + (1/dt) * eye(NUMPTS);
  P = 0.5*P;
  for j = 1:NTIME
     v = (Iden-P)\((Iden+P)*v);
     v = max(GG, v);
  end
  amsuper(kkk) =  mean(v);

end

disp('european price and confidence interval')
mean(eursuper)
1.96 * std(eursuper) / sqrt(p)

disp('american price and confidence interval')
mean(amsuper)
1.96 * std(amsuper) / sqrt(p)

--------------------------------------------------------------------------------

% This routine creates the QMC irregular grid

function G = irrgrid(x_t,r,q,sigma,T,t,alpha,NUMPTS,NDIMS)

mu_g = x_t + (r - q - 0.5*diag(sigma))*(T-t);
sigma_g = alpha * sigma * (T-t);

H = halton(NUMPTS, NDIMS);
```

```
H = mod(H+repmat(rand(1, NDIMS),NUMPTS,1),1); % Cranley-Peterson Rotations

R_g = chol(sigma_g)';

for i = 1:NUMPTS
    G(:,i) = mu_g + R_g * (icdf('normal',H(i,:),0,1))';
end

--------------------------------------------------------------------------------

% The routine computes the approximate infinitesimal generator matrix P by
% solving the consitency conditions LP

function [] = generateInfinitesimalGen(G,r,sigma,NUMPTS,NDIMS)

P = sparse(NUMPTS,NUMPTS);

eta_d = 0.5*NDIMS*(NDIMS+3)+1;
noPts = 20*eta_d; % no of points used for linear program
kk = 1;

for i = 1:NUMPTS

    % sets up constraints, objective function and
    % solves the LP program using linprog

    ndist = sum((G - repmat(G(:,i),1,NUMPTS)).^2).^(1/2);
    [Y,I] = sort(ndist);
    clear Y

    A = ones(1,noPts);
    B = G(:,I(1:noPts)) - repmat(G(:,i),1,noPts);
    for j = 1:noPts
        S(:,:,j) = (G(:,I(j))-G(:,i))*(G(:,I(j))-G(:,i))';
        dist(j,1) = norm(G(:,I(j))-G(:,i));
    end

    l = 1;
    for j = 1:NDIMS
        for k = 1:NDIMS
            C(l,:) =  S(k,j,:);
            l = l+1;
        end
    end
    a = 0;
    b = (r - 0.5*diag(sigma));
    c = reshape(sigma,NDIMS*NDIMS,1);

    AA = [];
    bb = [];

    Aeq = [A; B; C];
    beq = [a; b; c];

    LB = zeros(noPts,1);
    LB(1) = -Inf;
    UB = [];

    sdist = sort(dist);
    for j  = 1:noPts
        f(j,1) = find(sdist == dist(j));
    end

    f = f.^3;

    pij = linprog(f,AA,bb,Aeq,beq,LB,UB);

    P(i,I(1:noPts)) = pij;

    if(abs(sum(pij)) > 0.000000001)
        II(kk) =  i;   % absorbing bc for grid points that give rise to
                       % infeasible optimization problem
        kk = kk+1;
    end
end

% Saving the matrix P and absorbing bc info II
save infGen P
save delPlace II

--------------------------------------------------------------------------------

% This routine generates the Halton Sequence (obtained from MATLAB repository)
% http://people.scs.fsu.edu/~burkardt/m_src/halton/halton.html

function H = halton(NUMPTS,NDIMS);


if (NDIMS < 12)
```

```
        P = [2 3 5 7 11 13 17 19 23 29 31];
else
        P = primes(1.3*NDIMS*log(NDIMS));
        P = P(1:NDIMS);
end


if isequal(size(NUMPTS),[1 1])
        int_pts = [1:NUMPTS];
else %User has put in the points to sample.
        int_pts = NUMPTS;
        NUMPTS = length(int_pts);
end

H = zeros(NUMPTS,NDIMS);

for i = 1:NDIMS %Generate the components for each dimension.
        %V = fliplr(dec2base(int_pts,P(i)));
        %V = V-'0'; %Converts string to a matrix of doubles with correct numeric
                                   %values.
        V = fliplr(dec2bigbase(int_pts,P(i)));
        pows = -repmat([1:size(V,2)],size(V,1),1);
        H(:,i) = sum(V.*(P(i).^pows),2);
end

function s = dec2bigbase(d,base,n)

error(nargchk(2,3,nargin));

if size(d,2) ~= 1, d = d(:); end

base = floor(base);
if base < 2, error('B must be greater than 1.'); end
if base == 2,
  [x,nreq] = log2(max(d));
else
  nreq = ceil(log2(max(d) + 1)/log2(base));
end

if nargin == 3
    nreq = max(nreq,1);
    n = max(n,nreq);
    last = n - nreq + 1;
else
    n = max(nreq,1);
    last = 1;
end

s(:,n) = rem(d,base);
while n ~= last
    n = n - 1;
    d = floor(d/base);
    s(:,n) = rem(d,base);
end

--------------------------------------------------------------------------------
```

```
% This is the driver routine for the American Put pricing by computing the
% probability transition matrix

clear all;
close all;

NUMPTS = 2000;
NDIMS = 5;
NTIME = 1000;

x_t = log(40) * ones(NDIMS,1);
alpha = 1.5;
sigma = 0.25 * 0.04 * ones(NDIMS,NDIMS);
for i = 1:NDIMS
    sigma(i,i) = 0.04;
end
r = 0.06 * ones(NDIMS,1);
q = 0 * ones(NDIMS,1);
T = 1;
t = 0;
dt = (T-t)/NTIME;
K = 40;

% Step-1 : Create the irregular grid

G = irrgrid(x_t,r,q,sigma,T,t,alpha,NUMPTS,NDIMS);

figure(1);
plot(G(1,:),G(2,:),'.')

% Step-2 : Generating the P matrix by solving a Linear Program
generateTransitionMatrix(G,r,sigma,dt,NUMPTS,NDIMS);

% Step-3 : Backward dynamic programing to find price

load 'transmatrix.mat'

Payoff = max(K - prod(exp(G)).^(1/NDIMS),0)';

disfactor = exp(-r(1)*dt);

v = Payoff;

for i = NTIME-1:-1:1
```

```
    v = max(Payoff, disfactor*P*v);
end

american = mean(v)
payoff = mean(Payoff)

-----------------------------------------------------------------------------------

% The routine computes the transition probability matrix P by
% solving the consitency conditions LP


function [] = generateTransitionMatrix(G,r,sigma,dt,NUMPTS,NDIMS)

P = sparse(NUMPTS,NUMPTS);

eta_d = 0.5*NDIMS*(NDIMS+3)+1;
noPts = 20*eta_d; % no of points used for linear program
kk = 1;

for i = 1:NUMPTS

    % sets up constraints, objective function and
    % solves the LP program using linprog

    ndist = sum((G - repmat(G(:,i),1,NUMPTS)).^2).^(1/2);
    [Y,I] = sort(ndist);
    clear Y

    A = ones(1,noPts);
    B = G(:,I(1:noPts)) - repmat(G(:,i),1,noPts);
    for j = 1:noPts
        S(:,:,j) = (G(:,I(j))-G(:,i)-r*dt)*(G(:,I(j))-G(:,i)-r*dt)';
        dist(j,1) = norm(G(:,I(j))-(G(:,i)+r*dt));
    end

    l = 1;
    for j = 1:NDIMS
        for k = 1:NDIMS
            C(l,:) =  S(k,j,:);
            l = l+1;
        end
    end

    a = 1;
    b = (r - 0.5*diag(sigma)) * dt;
    c = reshape(sigma,NDIMS*NDIMS,1)*dt;

    AA = [];
    bb = [];

    Aeq = [A; B; C];
    beq = [a; b; c];

    LB = zeros(noPts,1);
    UB = [];

    sdist = sort(dist);
    for j  = 1:noPts
        f(j,1) = find(sdist == dist(j));
    end

    f = f.^3;

    pij = linprog(f,AA,bb,Aeq,beq,LB,UB);
    P(i,I(1:noPts)) = pij;

    if(sum(pij) > 1.0000001)
        P(i,:) = 0.0; % absorbing bc for grid points that give rise to
                      % infeasible optimization problem
        P(i,i) = 1.0;
    end
end

% Saving the matrix P

save transmatrix P

--------------------------------------------------------------------------------
```