Davis-Lo Credit Contagion Model: Theory, Implementation and Calibration

A final project report for the course

"Credit Risk Modeling"

Submitted

by

Changwei Xiong

December 2009

ABSTRACT

Correlation of defaults in a credit portfolio plays a very important role in portfolio performance. In this report, a fully dynamic contagion correlation model, known as Davis-Lo model, is implemented and calibrated to market tranche quotes. Davis-Lo model describes the dependency of defaults through “infection” such that any bond may default either directly or may be infected by any other defaulting bond. Both Monte Carlo simulation method and Markov chain generator method are applied to compute the distribution of the number of defaults in a fixed time. A numerical optimization routine is used to find the optimal values for the model parameters that minimize the tranche quotes residues between the modeled and the market data.

TABLE OF CONTENTS

1. Introduction

Many correlation models have been proposed to address the default correlation. The first attempt by Duffie and Singleton [1] in 1998 uses the correlated dynamics of the intensity process to induce default correlation. However realistic default correlations are impossible with typical normal or lognormal dynamics. To overcome this issue, Duffie and Garleanu [2] proposed an affine jump diffusion model, which combines normal, lognormal, or square-root diffusive intensity process with jumps. However this approach was not widely adopted as it relies on a slow Monte Carlo implementation. In 2001, Davis and Lo [3] published their infectious default model, which is believed to be the first contagion model. In this model, an issuer can default either idiosyncratically or by being infected by the default of another issuer according to some infection probability. The strength of the default correlation in this model is controlled by the infectious probability. Since this model only allows one step of contagion, Davis and Lo [4] proposed another dynamic model to relax this constraint. This is the model the project is going to implement and calibrate against market quotes. A detailed explanation of the model will be present in the next section.

2. Tranche Pricing with Enhanced Risk Model

2.1 Davis-Lo Dynamic Contagion Model

Davis and Lo [4] proposed a simple dynamic risk model to address the correlation of defaults in a portfolio of credits. The basic idea is that a default of a credit in a certain industry sector may trigger off defaults of other credits in the same sector or even the inter-sectors. The correlation among the defaults of the credits plays a significant role in the process. Let's consider a simple portfolio of credits which consists of n independent credits each having a default time that follows an exponential distribution with a parameter λ. Any one credit in the portfolio will

have a default probability, $p = 1 - e^{-\lambda T}$, in a time interval T. The number of defaults in the portfolio in time T follows a binominal distribution, Binomial(n, p). The expected number of defaults in the interval [0, t] is $E[N_t] = np = n(1 - e^{-\lambda t})$, where n is the initial number of credit names in the portfolio. We define a process:

$$M_t = N_t - \int_0^t \lambda(n - N_u)du \tag{1}$$

Because:

$$E[M_t|\mathcal{F}_s] = E[M_t - M_s|\mathcal{F}_s] + E[M_s|\mathcal{F}_s] = E[M_t - M_s|\mathcal{F}_s] + M_s \tag{2}$$

And since $M_t - M_s$ is independent of the filtration $\mathcal{F}_s$ and

$$E[M_t - M_s|\mathcal{F}_s] = E[M_t - M_s] = E[N_t] - E[N_s] - \int_s^t \lambda(n - E[N_u])du = 0 \tag{3}$$

The stochastic process M_t is a martingale, and hence we have:

$$P[\text{default in } [t, t + dt]|N_t] = E[dN_t|N_t] = \lambda(n - N_t)dt \tag{4}$$

This is the hazard rate, which is proportional to the number of names having survived up to time t. With the hazard rate defined above, the default process of the portfolio can be simulated by generating successive jump times of N_t, denoting these $T_1, T_2, \cdots$. Thus the time intervals, $T_{i+1} - T_i$, follows an exponential distribution with a parameter $\lambda(n - i)$. A further improvement can be done by introducing some interaction effects. In the real market, there exists a common effect that, when a default occurs, credit spreads for other names are elevated to some level and then settled back to normal levels after some time. In mathematical terminology, the process is described as follows: initially, the process starts from a ground state, each name has hazard rate of λ and the total hazard rate is $n\lambda$. When a default occurs, the hazard rate is elevated to an excited state by a factor $a > 1$ for all remaining names and the elevated total hazard rate is

$a(n-1)\lambda$. The excited state lasts for a random time exponentially distributed with a parameter μ, then reverts to ground state until next default occurs.

2.2 Default Distribution Function

Given the parameters, λ, μ and a, we are able to construct the default distribution function:

$$\pi(i,t) = P[N_t = i] \tag{5}$$

at different time horizons. This function is the basis for tranche pricing and can be evaluated either by Monte Carlo simulations or by direct matrix manipulations through Markov chain generator.

2.3 Matrix Exponentiation Method

Note that the default counting process N_t is a piecewise-deterministic Markov process with state space $E = \{(i,j) : i = 0,1 \text{ and } j = 0,1,\cdots,n\}$. It can be modeled as a continuous-time Markov chain with generator matrix A_t, where $A_t = \{a_{ij}(t)\}_{i,j=0}^{n}$. The generator matrix A_t (notes that its indices start from 0) has $(n+1)\times(n+1)$ elements and its off-diagonal elements provide state transition probabilities for infinitesimal time intervals dt (for $j \neq i$):

$$P[N_{t+dt} = j | N_t = i] = a_{ij}(t)dt \tag{6}$$

Let's define a matrix $Q = \{q_{ij}(t,T)\}_{i,j=0}^{n}$ to account for finite-time transition probabilities whose row index denotes the number of defaults at start time and column index denotes the number of defaults at end time:

$$q_{ij}(t,T) = P[N_T = j | N_t = i] \tag{7}$$

The probability matrix Q and the generator A satisfies the forward equation:

$$\frac{\partial Q(t,T)}{\partial T} = Q(t,T) \cdot A_t \tag{8}$$

In Davis-Lo model, the generator matrix $A_t = A$ is constant. Given N_t is a piecewise-deterministic Markov process, the forward equation can be solved as:

$$q_{ij}(t_1, t_2) = \left(e^{(t_2 - t_1)A}\right)_{ij} \tag{9}$$

If A_t is a function of time, the exponential term in the above equation need be evaluated as

$$(t_2 - t_1)A = \int_{t_1}^{t_2} A_t ds. \tag{10}$$

In the context of Davis-Lo model, each element of generator matrix $A = \{a_{ij}\}_{i,j=0}^{n}$ is a (2×2) sub-matrix, which accounts for the paths through 2 different states (ground and excited state). The main diagonal elements are:

$$a_{ii} = \begin{bmatrix} -\lambda(n-i) & 0 \\ \mu & -\mu - a\lambda(n-i) \end{bmatrix}, \qquad i = 0, \cdots, n-1 \tag{11}$$

and $a_{nn} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ for the last. The adjacent off-diagonal elements are:

$$a_{i,i+1} = \begin{bmatrix} 0 & \lambda(n-i) \\ 0 & a\lambda(n-i) \end{bmatrix}, \qquad i = 0, \cdots, n-1 \tag{12}$$

All the remaining elements of A are then $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$. Given the generator matrix, the solution to the forward equation (8) is $Q(t,T)$. Similar to matrix A, its elements are (2×2) sub-matrices:

$$q_{ij}(t,T) = \left(e^{(T-t)A}\right)_{ij}, \qquad i,j = 0, \cdots, n \tag{13}$$

Let's denote the indices of the sub-matrix as $k, l \in \{0, 1\}$, each element in the sub-matrix has a probabilistic meaning:

$$\left(q_{ij}(t,T)\right)_{kl} = P[N_T = j, S_T = l | N_t = i, S_t = k] \tag{14}$$

where S denotes the state with 0 for ground state and 1 for excited state. Since we eventually want to evaluate the default distribution function $\pi(j,t) = P[N_t = j]$, considering that the process starts only from a ground state, this is equivalent to:

$$\begin{aligned}\pi(j,t) &= P[N_t = j] \\ &= P[N_t = j, S_t = 0 | N_0 = 0, S_0 = 0] + P[N_t = j, S_t = 1 | N_0 = 0, S_0 = 0] \\ &= \left(q_{0,j}(0,t)\right)_{0,0} + \left(q_{0,j}(0,t)\right)_{0,1}\end{aligned} \tag{15}$$

Detailed explanation and implementation of the generator matrix have been discussed by Arnsdorf and Halperin [5].

2.4 Monte Carlo Simulation Method

Monte Carlo method is very straightforward to evaluate the default distribution function $\pi(i,t)$. We at first evenly subdivide the time horizon T into $\mathcal{M}$ steps, $t_i,\ i = 1, \cdots, \mathcal{M}$ and $t_0 = 0$, with increment $dt = T/\mathcal{M}$. Simulation starts with default counting number $N_0 = 0$ and $t_{ex} = t_0$, where t_{ex} is the end time of excited state. The initial value for $t_{ex} = t_0$, which means the simulation starts from a ground state. For each time step $t_i,\ i = 1, \cdots, \mathcal{M}$, we evaluate $h = c\lambda\left(n - N_{t_{i-1}}\right) \cdot dt$ with $c = 1$ if $t_i < t_{ex}$, or $c = a$ otherwise. If $h > U(0,1)$, where $U(0,1)$ is a uniform random variable between 0 and 1, default occurs, we then set $N_{t_i} = N_{t_{i-1}} + 1$ and set the $t_{ex} = t_i + \Theta(\mu)$, where $\Theta(\mu)$ is an exponentially distributed random number with parameter μ. We simulate $\Theta(\mu)$ by use of its inverse cumulative distribution function, which takes a $U(0,1)$ random variable and gives:

$$\Theta(\mu) = \frac{-ln[1 - U(0,1)]}{\mu} \tag{16}$$

We repeat performing the above simulation for $\mathcal{K}$ trials, then count the total number of occurrences for $N_{t_i} = k$ at time t_i, denoting this as C_{k,t_i}. The default distribution function can therefore be constructed as:

$$\pi(k, t_i) = P\left[N_{t_i} = k\right] = \frac{C_{k,t_i}}{\mathcal{K}} \tag{17}$$

To approximate $\pi(k, t)$ at an arbitrary time t, we will take $\pi(k, t_i)$ at t_i nearest to t. As $\mathcal{M}$ is sufficiently large, the error is negligible.

2.5 Tranche Pricing

Tranche pricing of a credit portfolio is based on the default distribution function $\pi(k, t)$. In the following, recovery rate R and risk free rate r are assumed constant for all names in the portfolio. We now consider the default leg. The default leg pays out the tranche loss amount at the time of the loss. The default leg DL of a tranche is given by:

$$DL = \int_0^T B(0, t) \cdot [-dEL(t, K_d, K_u)] \approx \sum_{i=1}^{M} \frac{(B_i + B_{i-1})(EL_i - EL_{i-1})}{2} \tag{18}$$

where the sum runs over all coupon dates, $t_i,\ i = 1, \cdots, M$, and $B_i = B(0, t_i) = \exp(-r \cdot t_i)$ is the risk-free discount factor, K_d and K_u are the tranche attachment and detachment point. In the above equation, the tranche expected loss function is $EL_i \equiv EL_{t_i} = \mathbb{E}[L(t_i, K_d, K_u)]$ and is given by:

$$\mathbb{E}[L(t_i, K_d, K_u)] = \mathbb{E}\left[\frac{(\mathcal{L}(t_i) - K_d)^+ - (\mathcal{L}(t_i) - K_u)^+}{K_u - K_d}\right] \tag{19}$$

where $\mathcal{L}(t_i) = (1 - R) \cdot \frac{N_{t_i}}{n}$. Given the default distribution function $\pi(k, t)$, we can calculate the expectation in the above equation by:

$$\mathbb{E}[L(t_i, K_d, K_u)] = \sum_{k=0}^{n} \left(\frac{1-R}{K_u - K_d} \cdot \left[\left(\frac{k}{n} - K_d \right)^+ - \left(\frac{k}{n} - K_u \right)^+ \right] \cdot \pi(k, t_i) \right) \tag{20}$$

The premium leg DL, on the other hand, paid by the protection buyer to the protection seller, is given by:

$$\begin{aligned} PL &= S(K_d, K_u) \cdot RA \\ &= S(K_d, K_u) \cdot \sum_{i=1}^{M} \Delta_i \left(B_i \cdot EN_i - \int_{t_{i-1}}^{t_i} \frac{u - t_{i-1}}{t_i - t_{i-1}} \cdot B(0,u) \cdot dEN(u, K_d, K_u) \right) \\ &\approx S(K_d, K_u) \cdot \sum_{i=1}^{M} \Delta_i B_i \frac{EN_{i-1} + EN_i}{2} \end{aligned} \tag{21}$$

where $S(K_d, K_u)$ is the tranche par spread, RA is the risk annuity, Δ_i is the day count fraction between t_{i-1} and t_i and

$$EN_i \equiv EN_{t_i} = \mathbb{E}[1 - L(t_i, K_d, K_u)] = 1 - EL_i \tag{22}$$

is the expected tranche outstanding notional at time t_i.

The tranche par spread, $S(K_d, K_u)$, is determined from the par equation $DL = PL$. Most index quotes are given in terms of the par spread. For the equity tranche, however, the market convention is to charge an upfront payment from the protection buyer while fixing the running spread at 500 bps.

3. Calibration to Market Data

3.1 Summary of Market Data

The purpose of calibration is to determine the optimal estimations of the model parameters that fit the model to the market tranche quotes. The model is calibrated against 5-year tranche quotes (from 05/15/2009 to 12/20/2013) for a portfolio of 125 names. The time period

spans approximately $T = 4.6$ years and the coupon payment date is $t_1 = 0.1$ and $t_i = t_1 + 0.25(i-1)$ for $i = 2, \cdots, 18$. The recovery rate is assumed to be 30% for all names while the risk free rate $r = 0.05$. Each name has its own 5-year clean spread S_i. The hazard rate λ is then estimated by the credit triangle relationship as:

$$\lambda = \frac{1}{125}\sum_{i=1}^{125}\frac{S_i}{1-R} = 0.048153 \tag{23}$$

3.2 Demonstration of Correctness

One simple way to demonstrate the correctness of the implementation is to compare the default distribution function $\pi(k,t)$ generated by Matrix Exponentiation method and by Monte Carlo simulations. The figures below show the excellent agreement between the two methods. In the demonstration, $R = 0.3$, $T = 4.6$, $r = 0.05$, $\lambda = 0.048153$, the number of Monte Carlo trials is 400,000 and the number of time steps is 2,000. The following figures are generated from various μ and a values, in which red bars represent the results from Monte Carlo simulations and blue bars represent that from Matrix Exponentiation method.

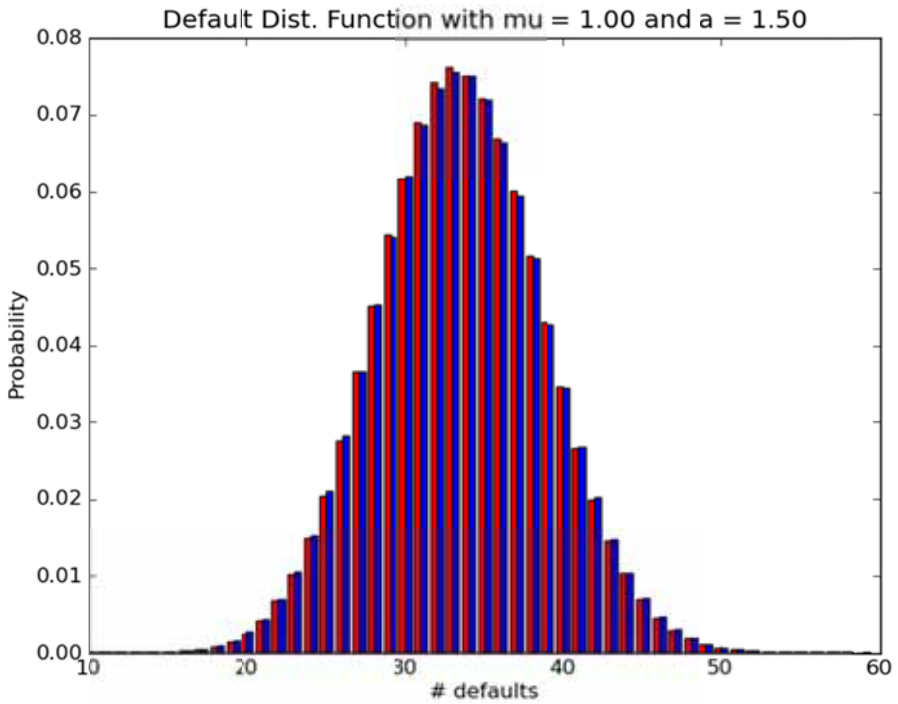

Figure 1. Default distribution function (Portfolio loss function) generated by Monte Carlo method (red) and Matrix Exponentiation method (blue).

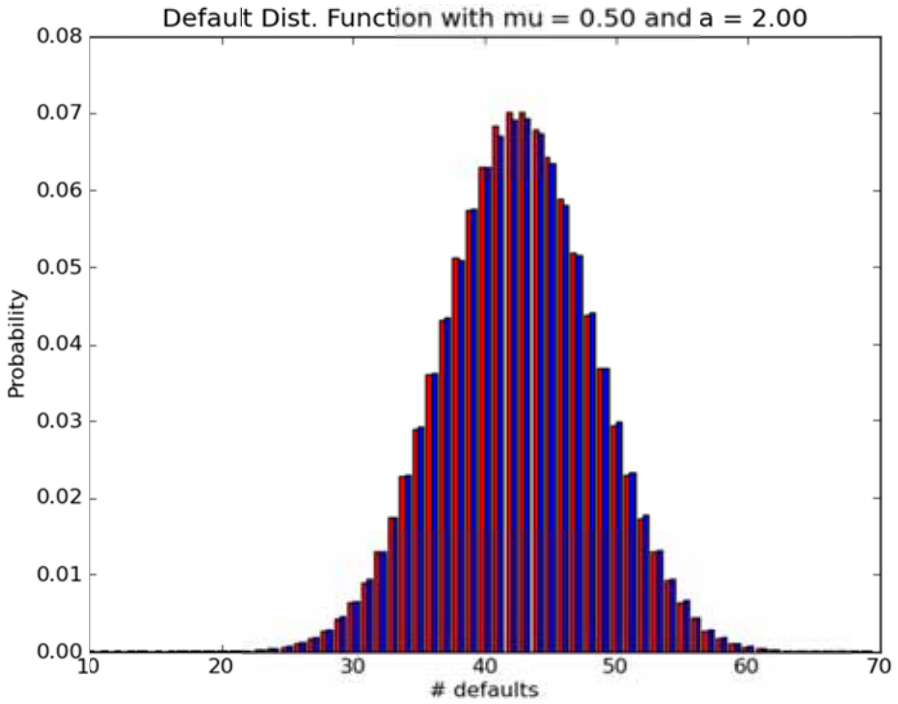

Figure 2. Default distribution function (Portfolio loss function) generated by Monte Carlo method (red) and Matrix Exponentiation method (blue).

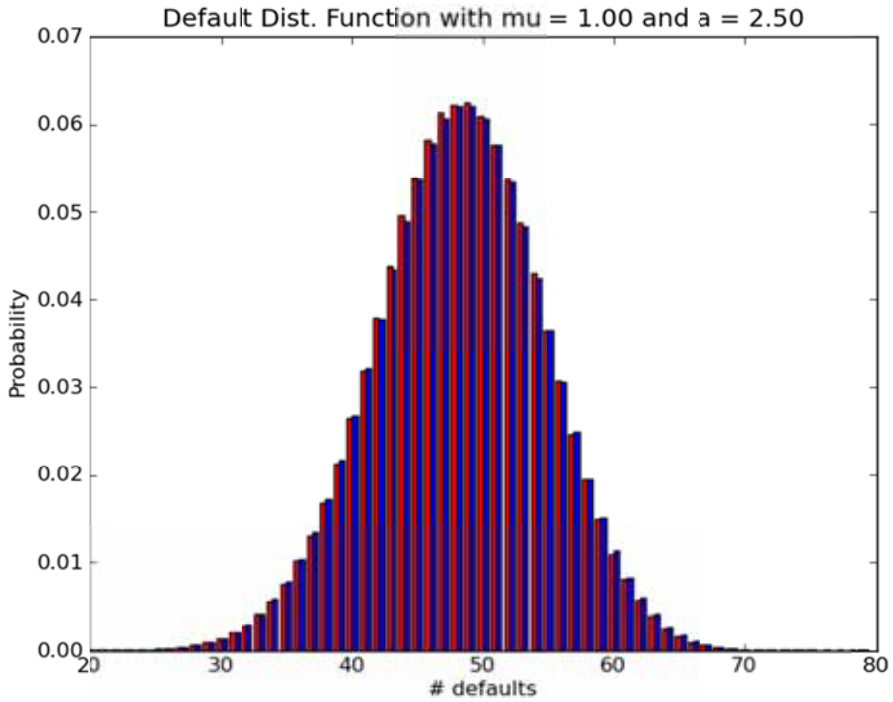

Figure 3. Default distribution function (Portfolio loss function) generated by Monte Carlo method (red) and Matrix Exponentiation method (blue).

3.3 Calibration to Tranche Quotes

The market data provides the quotes for 6 tranches of the credit portfolio with detachment/attachment points defined as (0%, 3%, 7%, 10%, 15%, 30% and 100%). In mathematical terminology, calibration of the model to market data is equivalent to finding the optimal estimations for the model parameters μ and a such that the residual between the modeled quotes and the market quotes is minimized. The residual ϵ is defined as follows:

$$\epsilon^2 = \sum_{i \in EQ} \left(1 - \frac{UP_{model}}{UP_{market}}\right)^2 + \sum_{i \in SE} \left(1 - \frac{S_{model}}{S_{market}}\right)^2 \tag{24}$$

where S is the tranche par spread, UP is the tranche upfront payment and computed as $UP_{model} = DL - 0.05 \cdot RA$, EQ denotes the collection of equity tranches whose market upfront payment is greater than zero, and SE denotes the collection of senior tranches whose market upfront payment is zero.

To ensure the model parameters is constrained by $\mu > 0$ and $a \geq 1$, two new variables are introduced such that $\mu = \max(x^2, \varepsilon)$ and $a = 1 + y^2$, where ε is a very small number and usually takes 1e-10 to prevent division by zero. Unconstrained optimization procedures can now be used to find optimal values for x and y, and therefore μ and a. Model tranche quotes are computed by Matrix Exponentiation method for its high efficiency and accuracy. Since the whole period spans $T = 4.6$ years and the coupon payment date is $t_1 = 0.1$ and $t_i = t_1 + 0.25(i-1)$ for $i = 2, \cdots, 18$, the Matrix Exponentiation method only needs compute two transition matrices, $Q_{0.1}$ and $Q_{0.25}$, for time interval $\Delta_1 = 0.1$ and $\Delta_2 = 0.25$, then the transition matrix Q_{t_i} for all the other coupon payment dates can be computed as:

$$Q_{t_1} = Q_{0.1} \qquad and \qquad Q_{t_i} = Q_{t_{i-1}} \cdot Q_{0.25}\,, \qquad i = 2, \cdots, 18 \tag{25}$$

This greatly reduces the computational time for the Matrix Exponentiation method.

Nelder-Mead Simplex [6] algorithm is used to minimize the residual. The process starts from an initial guess $\mu = 1$ and $a = 2$, and ends with optimal values $\mu = 3.4534$ and $a = 1.0$ after 53 iterations. The minimal residual at the optimal solution is $\epsilon = 8.435$ and its decomposition to each tranche is shown in Table 1. As you can see, the performance of model-fitting is very poor as all of the modeled values are greatly off of the market values. However, given the meaningful ranges for the parameters, this is the best solution we can obtain. Note that if $a = 1.0$, the model is no longer dependent on μ and this is equivalent to the case when $\mu = \infty$. Therefore the optimal solution to our problem is not unique, instead it is a union:

$$\{(\mu, a) : a = 1, \mu > 0\} \cup \{(\mu, a) : \mu = \infty, a \geq 1\} \tag{26}$$

Table 1. Residue decomposition at optimal solution
($\mu = 3.4534$ and $a = 1.0$)

Tranche #	K_d	K_u	Market	Model	Residue
1	0.00000	0.03000	0.68000	0.94691	**0.39252**
2	0.03000	0.07000	0.37250	0.84517	**1.26892**
3	0.07000	0.10000	0.08750	0.73032	**7.34653**
4	0.10000	0.15000	0.03360	0.16071	**3.78312**
5	0.15000	0.30000	0.01085	0.00738	**0.31996**
6	0.30000	1.00000	0.00525	0.00000	**1.00000**

To illustrate this property, the residue as a function of μ and a is computed in the range of $\mu \in [0.001,\ 1000]$ and $a \in [1,\ 10]$ and shown in Figure 4. The axes of μ has taken a logarithmic transformation for a better visualization. It clearly shows that the global minimal of residual is at the domain edges $a = 1$ and $\mu = \infty$. Monte Carlo simulation exhibits the same feature. Since Monte Carlo simulation is much slower than matrix exponentiation, Figure 5 is generated on a much coarser grid.

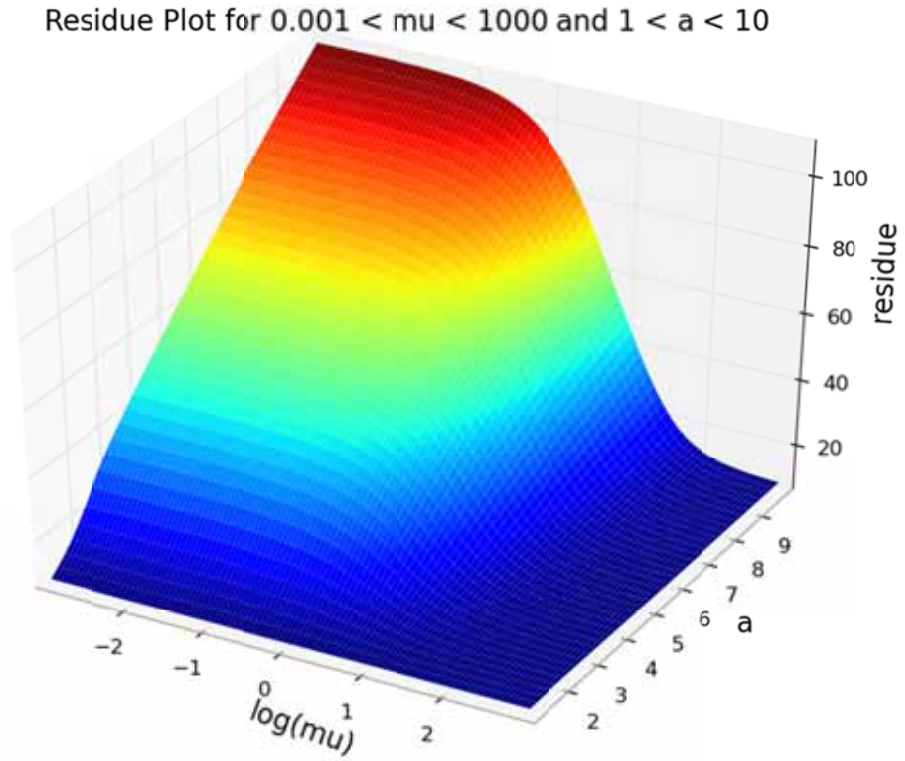

Figure 4. Residue as a function of μ and a, obtained by Matrix Exponentiation

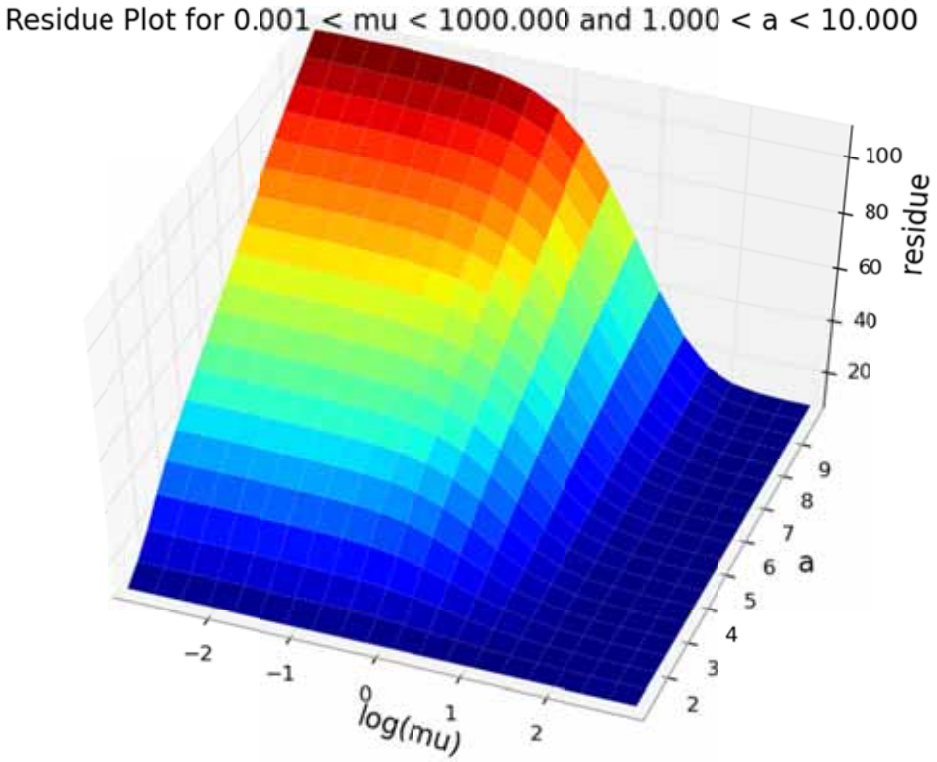

Figure 5. Residue as a function of μ and a, obtained by Monte Carlo simulation

Figure 6 shows the default distribution function when $T = 4.6$, and Figure 7 shows the time evolution of the distribution function. With the optimal μ and a, the model indeed assumes a homogeneous portfolio with independent names. Thus the default process follows a binomial distribution with each name has a default probability, $p = 1 - e^{-\lambda T}$. This is apparently not the case for real markets. As the model has only two parameters, it is highly restrictive and rigid. Calibration of this model to market data turns out to be extremely challenging. One possible reason for this is that, we estimate the homogenous hazard rate λ by take the average of a basket of CDS in the market, the hazard rate could be highly overestimated because a small number of the names have overwhelmingly high clean spreads. A simple improvement to the method is to estimate the average of hazard rate weighted by the market capitalization of the names. However this may not be of help eventually, as relaxing the hazard rate to be the third parameter in the model still does not show convergence property through optimization. The values of μ and a keep growing unboundedly.

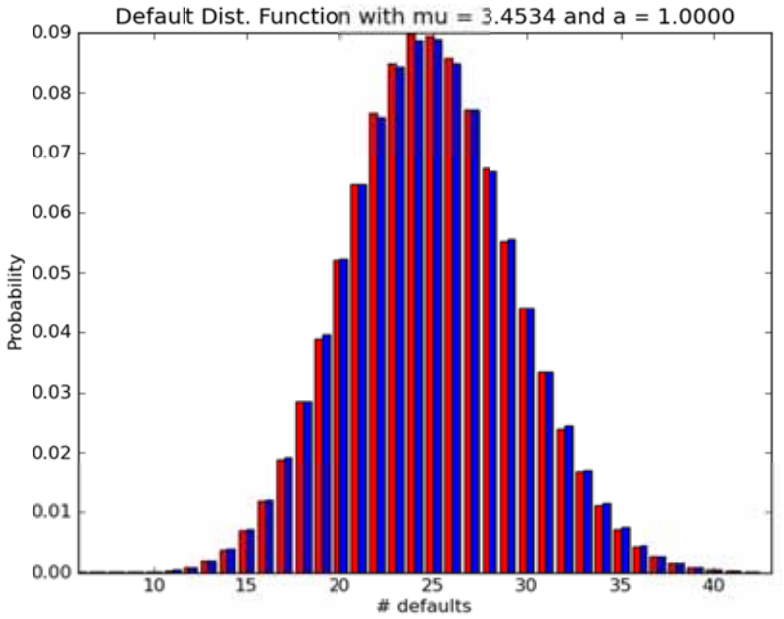

Figure 6. Default distribution function (Portfolio loss function) with optimal μ and a and $T = 4.6$, generated by Monte Carlo method (red) and Matrix Exponentiation method (blue).

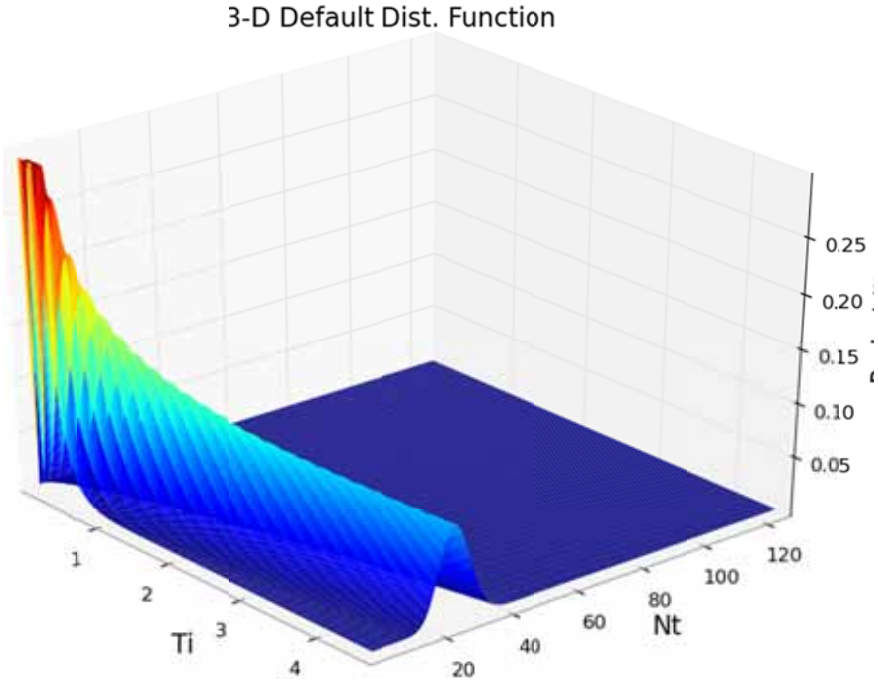

Figure 7. Time evolution of default distribution function (Portfolio loss function) with optimal μ and a (generated by Matrix Exponentiation).

4. Conclusions

The process described by Davis-Lo model is a piecewise deterministic Markov chain process. Default distribution function of the process can be easily constructed by Monte Carlo simulation or matrix exponentiation. Since matrix exponentiation is a direct method, it performs much faster than Monte Carlo simulation.

Calibration of the model to the market data is not very successful, because the model is quite simple and highly restrictive. It imposes many unrealistic assumptions to the problem. For example, it assumes a constant homogeneous hazard rate. This is obviously not true in the markets, as some of the names with exceptionally high clean spread are more prone to default than those with small spreads. Secondly, the model assumes only two states for the whole process: ground state and excited state. However, in the real market, default hazard is more like to aggregate if clustering defaults occur. Improvements to this model can be made by relaxing

some of the constraints. By introducing more parameters into this model, it enhances the model's flexibility and make the model more dynamic, and therefore leads to better fitting to the market data.

REFERENCES

1. Duffie, D.;Singleton, K., *simulation correlated defaults*, Working paper, 1998, Graduate School of Business, Stanford University

2. Duffie, D.; Garleanu, N., *Risk and valuation of collateralized debt obligations*, Working paper, 1999, Graduate School of Business, Stanford University

3. Davis M.H.A.; Lo, V., *Infectious defaults*, Quantitative Finance, 2001, 1(4), pp 382–387

4. Davis M.H.A.; Lo, V., *Modeling default correlation in bond portfolios*, Mastering Risk vol 2: Applications, Financial Times/Prentice-Hall, pp. 141-151

5. Arnsdorf, M.; Halperin, I., *BSLP: Markovian Bivariate Spread-Loss Model for Portfolio Credit Derivatives*, Journal of Computational Finance, 2008, vol 12(2)

6. J. A. Nelder and R. Mead, *A simplex method for function minimization*, Computer Journal, 1965, vol 7, pp 308–313

```
# This program is written in Python v2.6, libraries required are:
# numpy, scipy, matplotlib, xlrd, xlwt
# This program demostrates the Davis-Lo model and calibrates the model to market data
#
# It reads in the data from 2 sources:
#     'Inputs_for_model_calibration.xls' :
#         contains all the market data necessary for model calibration. The program reads in and
#         preprocess the data.
#     'Inputs.xls' :
#         contains some extra constants for the program to run, such as parameters for Monte Carlo
#         simulation, etc.
#
# Optimal solution mu and a are output to an excel file called 'Outputs.xls'
#
# Author: Changwei Xiong
# Date:   12/09/2009

import numpy as np
from scipy import optimize as spopt, linalg as spla
import matplotlib.pyplot as mp
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
from random import random as rand
from math import log, log10, ceil, floor, modf, exp, sqrt
import time
import datetime as dt
from xlrd import open_workbook, cellname, xldate_as_tuple
from xlwt import Workbook

class DavisLo(object):

    # constructor, reads from input files and initializes the constants
    def __init__(self):
        ret = self.read_xls('Inputs.xls', 'Inputs_for_model_calibration.xls')

        self.M = int(ret['M'])      # timesteps
        self.P = int(ret['P'])      # Monte Carlo trials
        self.r = float(ret['rf'])   # risk free rate
        T = float(ret['T'])         # time period
        delta = float(ret['delta']) # coupon payment day fraction

        self.lmd = float(ret['lambda'])  # lambda
        self.R = float(ret['recovery'])  # recovery rate
        self.mkt_tr = ret['market_tranches'] # market tranche quotes
        self.N = int(ret['total_names']) # number of total names
        self.ix = float(ret['ix'])       # initial guess for x : mu = x*x
        self.iy = float(ret['iy'])       # initial guess for y : a = y*y+1

        self.Di = [x*delta for x in [modf(T/delta)[0]]+[1]*int(modf(T/delta)[1])] # time slices
        self.Ti = [sum(self.Di[:i]) for i in xrange(len(self.Di)+1)] # times
        self.Bi = [exp(-self.r*t) for t in self.Ti]   # discounts

    # main program, everything is done here!!!
    # please comment/uncomment to toggle the features
    def final(self):
        mp.close()

```

```
58          # optimization routine for finding optimal solution
59          (x, y) = spopt.fmin(self.residue_xy, (self.ix, self.iy))
60          mu = max(x**2, 1e-8) # mu > 0
61          a = y**2 + 1   # a >= 1
62
63          print '\nOptimal Solution: mu = %.6f, a = %.6f'%(mu, a)
64          self.write_xls('Outputs.xls', mu, a)
65
66          ### plot default distribution function at optimal solution
67          ### takes 1-2 minutes to run,  uncomment it to run
68          #self.plot_LossFunction(mu=mu, a=a)
69
70          ### make residue plot as a function of mu and a
71          ### takes 20 minutes to run, uncomment it to run
72          #self.plot_residue(1e-3, 1e3, 1, 10)
73
74      # make a 3-D plot of residue as a function of mu an a
75      def plot_residue(self, mu_d, mu_u, a_d, a_u):
76          # meshgrid of 30X30
77          MU = np.linspace(log10(mu_d), log10(mu_u), 30)
78          A = np.linspace(a_d, a_u, 30)
79          res = np.zeros((len(MU),len(A)))
80          for i, mu in enumerate(MU):
81              for j, a in enumerate(A):
82                  res[i,j] = self.residue(10.0**mu, a, method='MC')
83          MUg, Ag = np.meshgrid(MU, A)
84          ax = Axes3D(mp.figure())
85          ax.plot_surface(MUg, Ag, res.T, rstride=1, cstride=1, cmap=cm.jet)
86          ax.set_xlabel('log(mu)', fontsize=16)
87          ax.set_ylabel('a', fontsize=16)
88          ax.set_zlabel('residue', fontsize=16)
89          mp.suptitle('Residue Plot for %.3f < mu < %.3f and %.3f < a < %.3f'\
90                      %(mu_d, mu_u, a_d, a_u),\
91                      fontsize=16)
92          mp.show()
93
94      # loss function (i.e. default probability function)
95      def plot_LossFunction(self, mu, a):
96          width = 0.4
97          st = 0
98          ed = self.N+1
99          index = -1
100         Pr = self.MonCar(mu, a)# monte carlo method
101         mp.bar(np.arange(st,ed)-width, Pr[index][st:ed], width, color='r')
102         Pr = self.MatExp(mu, a)# matrix exponentiation
103         mp.bar(np.arange(st,ed), Pr[index][st:ed], width, color='b')
104         mp.xlim(st,ed)
105         mp.xlabel('# defaults')
106         mp.ylabel('Probability')
107         mp.title('Default Dist. Function with mu = %.4f and a = %.4f' % (mu,a))
108
109         #plot 3-D default distribution function
110         Nt = np.arange(self.N+1)
111         Ti = np.array(self.Ti)
112         Ti, Nt = np.meshgrid(Ti, Nt)
113         Pr = np.array(Pr).T
114         Pr[Pr>0.3] = 0.3 # clamp the steep peak for visualization
115         ax = Axes3D(mp.figure())
116         ax.plot_surface(Ti, Nt, Pr, rstride=1, cstride=1, cmap=cm.jet)
117         ax.set_zlim3d(0, 0.3)
118         ax.set_xlabel('Ti', fontsize=16)
```

```
119             ax.set_ylabel('Nt', fontsize=16)
120             ax.set_zlabel('Probability', fontsize=16)
121             mp.suptitle('3-D Default Dist. Function', fontsize=16)
122 
123             mp.show()
124 
125         # expected tranche loss function
126         def ELoss(self, P, Kd, Ku):
127             v = (1-self.R)/self.N
128             return sum(p*(max(i*v-Kd,0)-max(i*v-Ku,0))/(Ku-Kd) for i, p in enumerate
    (P))
129 
130         # compute tranche spread or upfront payment from model
131         def Model_SP_UP(self, Pr, tr):
132             Kd = tr['Kd']
133             Ku = tr['Ku']
134             B = np.array(self.Bi)
135             D = np.array(self.Di)
136             EL = np.array([self.ELoss(P, Kd, Ku) for P in Pr])
137             DL = np.dot(0.5*(B[:-1]+B[1:]), EL[1:]-EL[:-1])
138             RA = np.dot(D*B[1:], 1-0.5*(EL[:-1]+EL[1:]))
139             #print DL, RA
140             return DL-0.05*RA if tr['UP']>0 else DL/RA
141 
142         # change of variables
143         def residue_xy(self, param):
144             mu = max(param[0]**2, 1e-8) # mu > 0
145             a = param[1]**2 + 1         # a > 1
146             return self.residue(mu, a)
147 
148         #residual calculation, primarily use matrix exponentiation
149         def residue(self, mu, a, method='ME'):
150             st = time.time()
151             if method == 'ME':
152                 Pr = self.MatExp(mu=mu, a=a)
153             else:
154                 Pr = self.MonCar(mu=mu, a=a)
155 
156             resid = 0.0
157             print ('%s\t\t'*5)%('Kd', 'Ku', 'market', 'model', 'residue')
158             for i, tr in enumerate(self.mkt_tr):
159                 if 1 or i in (0, 5):
160                     mkt = tr['UP'] if tr['UP'] > 0 else tr['SP']
161                     spup = self.Model_SP_UP(Pr,tr)
162                     resid += (1-spup/mkt)**2.0
163                     print ('%.5f\t\t'*5)%(tr['Kd'], tr['Ku'], mkt, spup, abs(1-spup/
    mkt))
164             resid = sqrt(resid)
165             self.mu = mu
166             self.a = a
167             print 'Total residue = %.4f, mu = %.6f, a = %.6f' % (resid, mu, a)
168             print 'time elapsed :%.4f seconds' % (time.time()-st)
169             print
170 
171             return resid
172 
173         # write the outputs to an excel file
174         def write_xls(self, filename, mu, a):
175             wb = Workbook()
176             ws = wb.add_sheet('Outputs')
177 
```

```
178             # for your reference, output some of the constants
179             # used in the models to the output file
180             ws.write(0,0,'Some of the constants used:');
181             ws.write(1,0,'lambda');
182             ws.write(1,1,self.lmd)
183             ws.write(2,0,'recovery rate');
184             ws.write(2,1,self.R)
185             ws.write(3,0,'T');
186             ws.write(3,1,self.Ti[-1])
187             ws.write(4,0,'# names');
188             ws.write(4,1,self.N)
189             ws.write(5,0,'risk free rate');
190             ws.write(5,1,self.r)
191 
192             # save solution to the output file
193             ws.write(7,0,'Optimal Solution:');
194             ws.write(8,0,'mu');
195             ws.write(8,1,mu)
196             ws.write(9,0,'a');
197             ws.write(9,1,a)
198 
199             wb.save(filename)
200 
201         # inputs parser
202         def read_xls(self, inputfile, datafile):
203             # read in input data
204             book = open_workbook(filename=inputfile)
205             for sheetid in xrange(book.nsheets):
206                 sheet = book.sheet_by_index(sheetid)
207                 if sheet.name == 'Input':
208                     ncol = 0
209                     dcol = 2
210                     stcl = sheet.cell
211                     for row in xrange(sheet.nrows):
212                         if stcl(row, ncol).value == 'Monte Carlo timesteps':
213                             MCsteps = stcl(row,dcol).value
214                         if stcl(row, ncol).value == 'Monte Carlo trials':
215                             MCpaths = stcl(row,dcol).value
216                         if stcl(row, ncol).value == 'risk free rate':
217                             rf = stcl(row,dcol).value
218                         if stcl(row, ncol).value == 'tranche time period':
219                             timeperiod = stcl(row,dcol).value
220                         if stcl(row, ncol).value == 'coupon day fraction':
221                             dayfrac = stcl(row,dcol).value
222                         if stcl(row, ncol).value == 'initial guess for x':
223                             ix = stcl(row,dcol).value
224                         if stcl(row, ncol).value == 'initial guess for y':
225                             iy = stcl(row,dcol).value
226 
227             # read in credit portfolio tranche data
228             book = open_workbook(filename=datafile)
229             for sheetid in xrange(book.nsheets):
230                 sheet = book.sheet_by_index(sheetid)
231                 if sheet.name == 'CDX IG11 single name data':
232                     hdrow = 0
233                     col5y = 0
234                     colR = 0
235                     names = sheet.nrows-1
236                     stcl = sheet.cell
237                     for col in xrange(sheet.ncols):
238                         if stcl(hdrow, col).value == '5Y clean':
```

```
                        col5y = col
                    if stcl(hdrow, col).value == 'recovery':
                        colR = col
                lmd = sum(stcl(row,col5y).value/(1-stcl(row,colR).value) \
                          for row in xrange(1,sheet.nrows))\
                        /names/10000.0
                R = sum(stcl(row, colR).value for row in xrange(1,sheet.nrows))/
names
            if sheet.name == 'CDX IG11 tranche quotes':
                hdrow = 0
                colup = 0
                colsprd = 0
                colk1 = 0
                colk2 = 0
                coldt = 0
                stcl = sheet.cell
                for col in xrange(sheet.ncols):
                    if stcl(hdrow, col).value == 'Lower':
                        colk1 = col
                    if stcl(hdrow, col).value == 'Upper':
                        colk2 = col
                    if stcl(hdrow+1, col).value == 'Maturity':
                        coldt = col
                    if stcl(hdrow+1, col).value == 'Payment (%)':
                        colup = col
                    if stcl(hdrow+1, col).value == 'Fee (bps)':
                        colsp = col
                mkt_tr = []
                for row in xrange(hdrow+2, sheet.nrows):
                    if dt.date(*xldate_as_tuple(stcl(row, coldt).value, 0)[:3]).
year == 2013:
                        mkt_tr.append({'Kd':float(stcl(row, colk1).value),
                                       'Ku':float(stcl(row, colk2).value),
                                       'UP':float(stcl(row, colup).value),
                                       'SP':float(stcl(row, colsp).value)/
10000.0})
                mkt_tr.sort(cmp=lambda x,y:cmp(x['Kd'], y['Kd']))

        return {'M':MCsteps,
                'P':MCpaths,
                'rf':rf,
                'T':timeperiod,
                'delta':dayfrac,
                'ix':ix,
                'iy':iy,
                'lambda':lmd,
                'total_names':names,
                'market_tranches':mkt_tr,
                'recovery': R}

    # Monte Carlo Simulation
    def MonCar(self, mu, a):
        mu = float(mu)
        a = float(a)
        N = self.N
        M = self.M
        P = self.P
        lmd = self.lmd
        dt = self.Ti[-1]/M
        idx = [int(round(t/dt)) for t in self.Ti]+[-1]
        pr = [[0.0]*(N+1) for t in self.Ti]
```

```
297
298             lmd_dt = lmd*dt
299             a_lmd_dt = a*lmd_dt
300             for p in xrange(P):
301                 n = 0
302                 x = 0
303                 h = 0
304                 ii = idx[h]
305                 for i in xrange(M+1):
306                     if i == ii:# ii = premium payment time index
307                         pr[h][n] += 1
308                         h += 1
309                         ii = idx[h]
310                     t = dt * i
311                     def_pr = (N-n) * (a_lmd_dt if t < x else lmd_dt)
312                     if def_pr > rand() and n < N: # default occurs
313                         n += 1
314                         x  = t - log(1.0-rand())/mu # exp. dist. y = 1-exp(-mu*x)
315             return (np.array(pr)/P).tolist()
316
317         # Matrix Exponentiation Method
318         def MatExp(self, mu, a):
319             N = self.N
320             lmd = self.lmd
321             A = np.zeros((2*(N+1), 2*(N+1)))
322             for i in xrange(N):
323                 j = 2*i
324                 A[j  , j  ] = -lmd*(N-i)
325                 A[j+1, j  ] = mu
326                 A[j+1, j+1] = -mu - a*lmd*(N-i)
327                 A[j  , j+3] = lmd*(N-i)
328                 A[j+1, j+3] = a*lmd*(N-i)
329
330             pr = []
331             #for t in self.Ti:
332             #    pm = spla.expm(A*t);
333             #    pr.append((pm[0,::2]+pm[0,1::2]).tolist())
334
335             pm0 = np.eye(2*(N+1))
336             pr.append((pm0[0,::2]+pm0[0,1::2]).tolist())
337             pm1 = spla.expm(A*self.Di[0])
338             pr.append((pm1[0,::2]+pm1[0,1::2]).tolist())
339             pm2 = spla.expm(A*self.Di[1])
340             pm = pm1
341             for t in self.Ti[2:]:
342                 pm = np.dot(pm, pm2)
343                 pr.append((pm[0,::2]+pm[0,1::2]).tolist())
344             return pr
345
346
347     if __name__ == '__main__':
348         try:
349             import psyco
350             psyco.full()
351             print 'Running psyco to speed up...'
352         except:
353             pass
354
355         DavisLo().final()
356
```