An Artificial Intelligence Approach to the Valuation of American-style Derivatives: A Use of Particle Swarm Optimization

Ren-Raw Chen*
Gabelli School of Business
Fordham University
45 Columbus Avenue
New York, NY 10019
rchen@fordham.edu

Jeffrey Huang
Bank SinoPac
Financial Markets
5F, #306, Bade Road, Section 2
Taipei, Taiwan (R.O.C.)
jeffrey.hcc@gmail.com

William Huang
Gabelli School of Business
Fordham University
45 Columbus Avenue
New York, NY 10019
khuang41@fordham.edu

Robert Yu
Gabelli School of Business
Fordham University
45 Columbus Avenue
New York, NY 10019
jyu115@fordham.edu

June 4, 2019
revised July 20, 2020

* Contact author. (212) 636-6471; rchen@fordham.edu. We thank the editor Dr. Joe Pimbley for his encouragement and valuable comments that make this paper substantially better.

An Artificial Intelligence Approach to the Valuation of American-style Derivatives: A Use of Particle Swarm Optimization

Abstract

In this paper, we evaluate American-style, path-dependent derivatives with an artificial intelligence technique. Specifically we use swarm intelligence to find the optimal exercise boundary for an American-style derivative. Swarm intelligence is particularly efficient (computation and accuracy) in solving high-dimensional optimization problems and hence perfectly suitable for valuing complex American-style derivatives (e.g. multiple-asset, path-dependent) which require a high-dimensional optimal exercise boundary.

Keywords: American Option, Monte Carlo, PSO

JEL code: G12, G13, G4

An Artificial Intelligence Approach to the Valuation of American-style Derivatives: A Use of Particle Swarm Optimization

I. Introduction

Evaluating American-style derivatives is a challenging task. In a uni-variate setting (e.g. option on one stock), lattice models – either the binomial model (e.g. Cox, Ross, and Rubinstein (1979)) or finite difference methods (e.g. see Hull (2015)) are an efficient method.[1] However, once the derivative contract is written on multiple assets (e.g. exchange options), lattice models would become infeasible (with regard to both computation time or memory space). Furthermore, path-dependent derivatives cannot be evaluated with lattice models.

As a result, modifying the Monte Carlo method to evaluate American-style derivatives is a popular alternative. There are two approaches to achieve this goal. The first approach is proposed by Longstaff and Schwartz (2001) who approximate the continuation value of the option by a regression function (functional form can be arbitrary). They recognize that the early exercise decision is merely a comparison of the exercise value and its continuation value of the option. If the continuation value can be reasonably accurately estimated, then the early exercise problem can be easily solved and hence one can readily compute the value of an American-style derivative. The drawback of this approach is apparent – it is hard to know in advance which functional form of the regression will provide an accurate estimate for the continuation value.

The other approach is to recognize that derivatives pricing in general is a free-boundary PDE (partial differential equation) problem. If we can accurately estimate the exercise boundary, then it is just an easy integration over the boundary (as the first passage time problem). In other words, if we can accurately estimate the boundary, then the value of an American-style derivative can be calculated as it would be a barrier option.[2]

This approach is more computational efficient than the Longstaff-Schwartz model; yet if suffers the same drawback of the Longstaff-Schwartz model – the accuracy of the American-style derivative value relies upon an accurate exercise boundary. Moreover, the literature of this approach lacks the evidence on derivatives on multiple assets.

[1] By efficient, we refer to the balance between speed and accuracy.
[2] For recent work, see Bunch and Johnson (2000) and Carr, Jarrow, and Myneni (2008). Also see Nunes (2008) for a nice review/comparison of various boundaries.

In this paper, we introduce an artificial intelligence method, i.e. swarm intelligence, to locate the optimal exercise boundary. In particular, we use an optimization algorithm within the realm of swarm intelligence named PSO (particle swarm optimization) to locate the optimal exercise boundary. The intelligence by the swarm can efficiently decide piece-wise values of the boundary, one for each time step, without an approximated functional form as in the literature. As in any artificial intelligence model, PSO is efficient in high dimensional optimization problems. In the case of a truly free boundary (i.e. piece-wise), we find that PSO can ideally provide the best solution to complex (e.g. American-style, multi-asset, path-dependent) derivatives problems.

II. Monte Carlo in American-style derivative Pricing

In this section, we briefly describe the two Monte Carlo methods in American-style derivative pricing. It is generally understood that Monte Carlo simulations are only suitable for pricing European-style derivatives. This is because American-style derivatives require a backward induction. In other words, the optimal exercise decision at any given time depends on all future optimal exercise decisions. The first method proposed by Longstaff and Schwartz (2001) recreates such a recursive structure in Monte Carlo and the second method adopts a free-boundary property in PDE (partial differential equation) solutions.

1. The Longstaff-Schwartz Model

The Longstaff-Schwartz model (2001) is the most popular model in the financial industry. It is an efficient Monte-Carlo model for American-style derivatives. Longstaff and Schwartz propose a regression method to estimate the "continuation value" at each time step.[3] The option value ξ_t at any time t is the larger of the exercise value E_t and the continuation value C_t as follows:

(1) $\xi_t = \max\{E_t, C_t\}$

where

(2) $C_t = \mathbb{E}_t^Q[\xi_{t+1}]$

is the continuation value of the option at time t (which is the risk-neutral expected value, $\mathbb{E}_t^Q[\cdot]$, of the next period's option price); and E_t is the exercise value at time t. In the case of a put option (which is what we use throughout the paper), $E_t = K - S_t$.

[3] As a reminder, a continuation value in option literature refers to the expected value of future maximum payoff at any given point in time. Since the continuation value is the expected future payoff, it is compared to the exercise value at the given time to see if (early) exercise is worthwhile.

Longstaff and Schwartz cleverly recognize that the conditional expectation of future maximum payoff is a function of today's stock price:

$$E_t^Q[\xi_{t+1}] = f(S_t) \tag{3}$$

where $f(\cdot)$ is an arbitrary function. They propose the simplest (and it works amazingly well) quadratic equation:

$$\begin{aligned} E_t^Q[\xi_{t+1}] &= f(S_t) \\ &= a_0 + a_1 S_t + a_2 S_t^2 \end{aligned} \tag{4}$$

which can turn into a regression model as follows:

$$\xi_{t+1} = a_0 + a_1 S_t + a_2 S_t^2 + e_{t+1} \tag{5}$$

with the boundary condition $\xi_T = \max\{K - S_T, 0\}$.

As mentioned earlier, the major criticism of the model is the choice of the functional form of the regression. It is ad-hoc and it is not possible to know which form is most suitable for which payoff.[4]

2. Explicit Boundary Method

In an alternative (relatively unsuccessful) attempt, researchers have tried to solve American-style derivatives by using an explicit exercise boundary.[5] The approach is built upon the nice property that option prices of any kind are solutions to a class of differential equations which can be solved as a "free boundary problem". In other words, as long as the exercise boundary of an option is known, its price is no more than a simple integration along the exercise boundary.

Unfortunately, not only is the exercise boundary of an American-style derivative unknown, but it is recursive (i.e. the boundary value at the current time depends on the boundary value at the immediately later time – resulting a recursively dependent structure of boundary values). In other words, the boundary function can only be achieved via a lattice model (e.g. binomial model). In doing so, the option is guaranteed to be exercised optimally and the valuation can hence be at the maximum.

[4] In the case of put options, the quadratic function works very well. Yet in other forms of payoff, Longstaff and Schwartz do not provide any guidance.
[5] For example, see Carr (1998).

As Carr (1998), among others, points out, if we solve an American-style derivative premium as a free-boundary problem, then we can use an explicit boundary function and the American-style derivative premium is simply an integration of payoff function (e.g. put) over the boundary.

(6) $\xi(t) = \mathbb{E}_t^Q \left[e^{-r\tau} \max\{[E(\tau), 0\} \right]$

where $E(\tau)$ is the exercise value at the stopping time τ. If it is a put option without dividends which is the case in this paper, then $E(\tau) = K - S(\tau)$. On the boundary, $S(\tau) = B(\tau)$ and hence $E(\tau) = K - B(\tau)$ where $B(\tau)$ is the boundary function given exogenously. The way the boundary function works is that it serves as a stopping time. Once the stock price at time t hits the boundary $B(t)$, the process stops and the option will be exercised and paid and hence the American-style derivative can be evaluated as a barrier option.

The easiest way to perform the integration is through Monte Carlo simulations. As the derivative price $\xi(t)$ is given as an expected value:

(7) $\xi(t) = \frac{1}{N} \sum_{j=1}^{N} e^{-r\tau_j} \max\{K - B(\tau_j), 0\}$

We note that the recursively determined boundary function (via a lattice model) maximizes the option value, any other exogenously specified boundary function will only be "sub-optimal", that is, generating a lower value than the lattice model. This sub-optimal argument is convenient in that now we can simply try a large number of boundary functions and use the one that generates the highest option value as a good approximation.

Researchers then have tried various approximations on the exercise boundary. These approximations are explicit functions and hence can be easily integrated (and hence American-style derivative value solved for). According to a recent survey by Nunes (2008), the literature has the following functional forms:

- Constant: $B(t) = a_0$
- Linear: $B(t) = a_0 + a_1 t$
- Exponential: $B(t) = a_o e^{a_1 t}$
- Exponential-constant: $a_0 + e^{a_1 t}$
- Polynomial: $B(t) = \sum_{i=1}^{n} a_i t^{i-1}$
- Carr-Jarrow-Myneni (2008): $B(t) = \min(K, \frac{r}{q} K) e^{-a\sqrt{T-t}} + E_\infty \left[1 - e^{-a\sqrt{T-t}} \right]$

Note that the boundary is not a function of the stock price (i.e. free boundary problem). Since these boundary functions are explicit, they can be easily integrated.

Certainly the accuracy of the American value depends on the accuracy of the approximated boundary function. The problem of this approach is that there is no consensus of which functional form of the boundary can consistently be the best. Often it varies with the parameters of the option (i.e. moneyness, interest rate, time to maturity, and volatility). As a result, no conclusion can be drawn on a particular functional form.

So far the literature has not reached any consensus and the boundary seems to be payoff-specific. In other words, different payoffs require different boundaries for accurate American-style derivative values. As a result, it is quite natural to allow the boundary function to be absolutely free (i.e. one value per time step). Yet this requires an optimization in high dimensions. As the number of time steps increases, the cost of computation becomes exponentially prohibitively high.

In this paper, we propose an artificial intelligence (AI) method which is based upon the theory of swarm (swarm intelligence, SI). In the SI model, a school of fish (or a group of ants and bees or a flock of birds) will move (swim) around to look for the maximum value of the option.

III. Swarm Intelligence

In this section, we briefly "open the black-box" of swarm intelligence which is a branch of recently popular artificial intelligence.

1. What is AI?

Artificial intelligence (AI), machine learning (ML), and big data (BD) have recently been adopted into FinTech and been the fastest growing area in finance, both in private industry and academia. While these three areas are frequently used in combinations in developing valuable applications, these three areas are fundamentally different and deserve separate research.

Strictly speaking, AI is a combination of computation (artificial) and biology (intelligence) which is quite different in nature from ML which is based upon statistical methodologies. In the past, statistics have predominantly been presented in a parametric fashion, mainly due to insufficient computation power and lack of data. This has been changed recently and non-parametric statistics with powerful computation capabilities fuel the growth of machine learning. As non-parametric statistics require a large amount of data, ML and BD (such as NLP, or natural language processing) have been combined in revolutionizing the financial world. Together, they facilitate the progress of AI.

AI has three major branches:

- swarm intelligence (birds, ants, bees, fish)
- genetic algorithm (genes)
- neural networks (neurons)

These AI theories are behavioral models in that they "artificialize" natural intelligence (specified in parentheses above) which reflects biological behaviors. As a result, they are different from ML methodologies. The connection (and hence confusion) of these two is due to the fact that these AI models can be efficiently used to find optimal solutions (e.g. PSO) which then are similar to ML models. Indeed, from the perspective of computation, one can hardly differentiate one tool from the other and in many instances these two distinctly different theories are used in combination.

As we shall demonstrate in this section, swarm intelligence is a behavioral model and PSO is an optimization tool.

2. Swarm Intelligence

Wikipedia describes swarm intelligence as "the collective behavior of decentralized, self-organized systems."[6] The basic idea of swarm intelligence is derived from those animals (such as birds, ants, bees, and fish) that rely on group effort to achieve their basic survival needs – seek food and avoid prey. The intelligence behind this collective behavior is how they communicate among one another. Reynold (1987)[7] was the first to "artificialize" such natural intelligence and create a computer algorithm named Boids (for bird-oid object). Reynold's algorithm is amazingly simple. For any given bird, Reynold devises a set of linear equations (vectors) combining which determines how the bird should fly to its next destination.

The factors that determine how various vectors are combined are: separation, alignment, and cohesion. As their names suggest, "separation" is to avoid collision with other birds, "alignment" decides how a particular bird should fly in a direction by referencing to its fellow birds, and "cohesion" decides how fast (speed) a particular bird should fly to its next target position.

[6] https://en.wikipedia.org/wiki/Swarm_intelligence.
[7] According to Wikipedia (footnote 6), Reynold created Boid in 1986: "Boids is an artificial life program, developed by Craig Reynolds in 1986, which simulates the flocking behaviour of birds."

There are countless versions of Boids.[8] One can add obstacles. One can add an objective destination (swim to target). One can do Boids in a maze. The basic Boids as described in Figure 1 can be described by the following algorithm.

[Figure 1 Here]

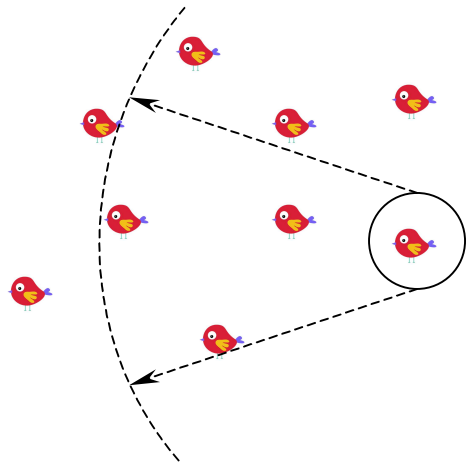
Formally, let there be m birds flying in an n-dimensional space. Also let:

- $f_t^{(i)}$ be the ith bird at time t
- $v_t^{(i)}$ be a vector in the $\mathbb{R}^n$ space representing the velocity of the ith bird
- $p_t^{(i)}$ be a vector in $\mathbb{R}^n$ space representing the position (coordinates) of the ith bird

Finally let $F = \{f_t^{(i)} \mid i = 1,\cdots,m\}$ be the collection of all birds. Define a mapping function $X_i = \wp(f_{t-1}^{(i)})$ which returns all $f_{t-1}^{(j\neq i)} \in F - f_{t-1}^{(i)}$ where a radius d and an angle a are predetermined such that ($|| x,y ||= \sqrt{x^2+y^2}$ and $\angle\{x,y\}$ is the angle between two vectors)

$$(8) \qquad \begin{matrix} || v_{t-1}^{(i)}, v_{t-1}^{(j\neq i)} ||< d \\ || p_{t-1}^{(i)}, p_{t-1}^{(j\neq i)} ||< d \end{matrix} \quad \text{and} \quad \begin{matrix} \angle\{v_{t-1}^{(j\neq i)}, v_{t-1}^{(i)}\} = a^\circ \\ \angle\{p_{t-1}^{(j\neq i)}, p_{t-1}^{(i)}\} = a^\circ \end{matrix}$$

are satisfied. In words, what (8) describes is that for any given bird i, where it is heading depends on a reference group of birds "nearby", described by a set of birds $X_i = \wp(f_{t-1}^{(i)})$. These reference birds must be "nearby" in the following sense –they must be within a distance (specified by the radius d) and within an angle (specified by a°), as depicted graphically as:[9]



[8] For example, see Google Scholar: https://scholar.google.com/scholar?q=boids+flocking+algorithm&hl=en&as_sdt=0&as_vis=1&oi=scholart

[9] These reference birds are like "my leaders" for a given bird.

where the circled bird is referencing to three nearby birds by the angle and the radius. The alignment and cohesion (we ignore separation parameter for the moment) parameters are calculated as follows:[10]

$$(9) \qquad \begin{aligned} v_{A,t}^{(i)} &= \text{avg}\left(v_{t-1}^{(j\neq i)} \mid f_{t-1}^{(j)} \in X \right) - v_{t-1}^{(i)} \\ v_{C,t}^{(i)} &= \text{avg}\left(p_{t-1}^{(j\neq i)} \mid f_{t-1}^{(j)} \in X \right) - p_{t-1}^{(i)} \end{aligned}$$

and then an average velocity is calculated as follows:

$$(10) \qquad \overline{v}_t^{(i)} = w_A v_{A,t}^{(i)} + w_C v_{C,t}^{(i)}$$

where $w_A + w_C = 1$ and each is positive. Finally, velocity and position of each bird are updated as follows:

$$(11) \qquad \begin{aligned} v_t^{(i)} &= v_{t-1}^{(i)} + \overline{v}_t^{(i)} \\ p_t^{(i)} &= p_{t-1}^{(i)} + \overline{v}_t^{(i)} \end{aligned}$$

As emphasized earlier, a swarm is a behavioral model which describes how birds (ants, bees, fish) move and an artificial swarm is a mathematical (linear algebraical) algorithm that imitates this natural behavior by animals. One can use an artificial swarm to solve a number of complex problems.[11]

As we can see, Reynold's boid model can be easily programmed and implemented. For the sake of easy exposition, we shall refer birds, ants, bees, or fish as particles for the rest of the paper.

Particle Swarm Optimization (PSO) can be viewed as simplified AI swarm. Its objective is to find the global optimum. While details can be seen in the next section, the idea of a PSO is to replace nearby birds/particles by the global optimum found by all birds/particles.

3. Particle Swarm Optimization

In theory, swarm intelligence is effective for optimization problems in a high-dimensional space. PSO is such an application. The original version of PSO was first proposed by Eberhart and Kennedy (1995) who modify the behavioral model of swarm into an objective-seeking algorithm.

[10] We ignore separation in our model because in our applications particles can take the same coordinates (i.e. collision is allowed).

[11] While this is out of the scope of this paper, we encourage the readers to view a popular YouTube clip on how drones use an artificial swarm: "'Skynet'Drones Work Together for 'Homeland Security'" (https://www.youtube.com/watch?v=oDyfGM35ekc).

Similar to Renold's, their model "artificializes" the group behavior of a flock of birds seeking food. Via bird-to-bird chirping (peer-to-peer communication), all birds fly to the loudest sound of chirping. Subsequently, Eberhart and Shi (1998) improve the model by adding an inertia term (symbolized as w later as we introduce the model) and it has become the standard PSO algorithm used today. Setting a proper value of the inertia term is to seek the balance between *exploitation* and *exploration*. A larger value of the inertia term gives more weight to exploration (as the bird is more likely to fly on its own) and a smaller value of the inertia term gives more weight to exploitation (as the bird intends more to fly toward other birds).[12]

One can compare PSO to the grid search. A grid search can find the global optimum and yet it takes an exploding amount of time to reach such a solution, especially in a high-dimensional space. PSO can be regarded as a "smart grid search" where each particle performs a "stupid search" and yet by communicating with other particles and by having a large number of such particles we can reach the global optimum quickly.

Imagine we would like to measure the deepest place of a lake whose bottom has an uneven surface. A two-dimension grid search can easily find the global minimum. An alternative would be PSO. Imagine we have a number of "fish" (particles) who swim in the lake. At each time step, all fish will measure the depth of the lake underneath them. And each fish is communicating with all the other fish to decide whose depth is the deepest (minimum). All fish now remember the minimum and then they swim for another time step. At each time step they update the global minimum so far. If we let these fish swim randomly for enough time, we will reach the global minimum.

In the case of the lake, we may find the grid search to be more accurate and time-effective. But in an n-dimensional lake, grid searches are becoming ineffective but the same number of fish may just do the same job with the same amount of time as in the two-dimensional lake.

Currently there have been some limited number of applications of PSO in finance, mostly in portfolio selection. In this paper, we use it for the first time in the literature to locate the exercise boundary of American-style derivatives (specifically, put option, option on min/max, and Asian option).

[12] Similar to PSO, an ACO (ant colony optimization) by Dorigo, Bonabeau, and Theraulaz (2000) and ACS (ant colony system) by Dorigo and Gambardella (1997) are both based upon swarm intelligence. The first ant system is first developed by Dorigo, Maniezzo, and Colorni (1991) and then popularized by Dorigo, Maniezzo, and Colorni (2000).

The PSO algorithm can be formally defined as follows. For $i = 1,\cdots,n$ particles and each particle is a vector of $j = 1,\cdots,m$ dimensions, we have:

$$(12) \qquad \begin{cases} \vec{v}_{i,j}(t+1) = w(t)\vec{v}_{i,j}(t) + r_1 c_1(\vec{p}_{i,j}(t) - \vec{x}_i(t)) + r_2 c_2(\vec{g}(t) - \vec{x}_{i,j}(t)) \\ \vec{x}_{i,j}(t+1) = \vec{x}_{i,j}(t) + \vec{v}_{i,j}(t+1) \end{cases}$$

where $\vec{v}_{i,j}(t)$ is velocity of the i th particle in the j th dimension at time t; $\vec{x}_{i,j}(t)$ is position of the i th particle in the j th dimension at time t; $w(t)$ is a "weight" (less than 1) which decides how the current velocity will be carried over to the next period (and usually it is set as $w(t) = \alpha w(t-1)$ and $\alpha < 1$ to introduce diminishing velocity);[13] and finally $r_1, r_2 \sim u(0,1)$ follow a uniform distribution.

In the swam literature, $w(t)\vec{v}_i(t)$ is called inertia; $r_1 c_1(\vec{p}_i(t) - \vec{x}_i(t))$ is called the cognitive component and $r_2 c_2(\vec{g}(t) - \vec{x}_i(t))$ is called the social component. Coefficients c_1 and c_2 are known as acceleration coefficients.

At each position there is "cost function" $f(\cdot)$ (sometimes called distance function) at which a "cost" (or penalty) is computed. This cost function is the objective function to be minimized (or maximized).

The global best at any given time is either the maximum or minimum value of the objective function generated by all particles at the time:

$$(13) \qquad \vec{g}(t) = \min_i \{f(\vec{p}_i(t))\}$$

and the personal best at the time is:

$$(14) \qquad \vec{p}_i(t) = \min_t \{f(\vec{x}_i(t))\}$$

and $f(\cdot) : \mathbb{R}^n \to \mathbb{R}$ is the "fitness function". The usual fitness function is

$$(15) \qquad f(\vec{x}_i(t)) = \|x_i - \underline{\chi}\| = \sum_{j=1}^{J} (x_{ij} - \chi_j)^2$$

where $\underline{\chi} =< \chi_1,\cdots,\chi_J >$ is a coordinate in a J -dimensional space.

Later, we illustrate via a very simple example how the process is so easily implemented.

[13] The reason is that as a particle is approaching the global best, the velocity should approach 0 (i.e. the particle should no longer move at the global optimum.)

As we can see, the algorithm (at least the standard one presented here) of PSO is quite different from that of a generic swarm by Reynolds (1978). Yet they both share the same behavioral pattern of a natural swarm. In other words, (1) both PSO and the generic swarm are based upon peer-to-peer communication in order to achieve the objective and (2) the particles in both PSO and the generic swarm are identical (like birds or ants) and each particle follows its neighbor particles. The difference is just how each particle weighs its neighbors. In PSO, each particle only cares of the global best discovered by its neighbors and in the generic swarm each neighbor's position is important.

i) Different Types of PSO

The literature on PSO is voluminous. Zhang, Wang, and Ji (2015) provide an excellent survey. They classify the existing PSO literature into the following strands:[14]

- modifications,[15]
- population topology,[16]
- hybridization,[17]
- extensions,[18]
- theoretical analysis,[19] and
- parallel implementation.[20]

However, Zhang, Wang, and Ji only provide applications in non-financial areas.[21, 22] To date, there have been very limited number of applications in the area of finance. Within the limited literature, most noticeably is in the area of portfolio selection.[23]

[14] PSO can also vary in terms of parameterization such as center mass (see Jamous, Tharwat, Seidy, and Bayoumi (2015)).
[15] This includes quantum-behaved PSO, bare-bones PSO, chaotic PSO, and fuzzy PSO.
[16] This includes von Neumann, ring, star, random, among others.
[17] This is to combine PSO with genetic algorithm, simulated annealing, Tabu search, artificial immune system, ant colony algorithm, artificial bee colony, differential evolution, harmonic search, and biogeography-based optimization.
[18] This includes multi-objective, constrained, discrete, and binary optimization.
[19] This includes parameter selection and tuning, and convergence analysis.
[20] This involves multi-core, multiprocessor, GPU, and cloud computing forms.
[21] They are electrical and electronic engineering, automation control systems, communication theory, operations research, mechanical engineering, fuel and energy, medicine, chemistry, and biology.
[22] Kumar et. al. (2013) examines performance of various PSO algorithms: Canonical PSO, Hierarchical PSO (HPSO), Time varying acceleration coefficient (TVAC) PSO, Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients (HPSO-TVAC), Stochastic inertia weight (Sto-IW) PSO and Time varying inertia weight (TVIW) PSO have been used for comparative study. These versions of PSO vary in only parameterization.
[23] See Huang (2019) for a survey.

ii) A PSO Demonstration

As a demonstration, we use a conic function as follows:

(16) $$f(x,y) = x^2 + y^2$$

The function is a cone as shown in the top plot in Figure 2. In Figure 2a, we can readily see how particles move toward the center of the cone which is the global minimum.

[Figure 2 Here]

Another function as follows has multiple local minima.

(17) $$f(x,y) = \frac{x^2 + y^2}{4000} - \cos[x]\cos\left[\frac{x}{\sqrt{2}}\right] + 1$$

and the results are shown in Figure 2b.

The major advantage of PSO is that it is particularly good at the problems with many local optima. As we can see in Figure 2b. However, in this situation, the convergence of the swarm is slower and each particle needs to "work harder" to identify the global minimum, since there are many local minima.

Another advantage of PSO is its superior capability to find the global optimum when the objective function is discrete. In Huang (2019), PSO is applied on the Sortino ratio maximization. Different from the Sharpe ratio, the Sortino ratio only concerns the "down-side risk" and as a result, the ratio is not a continuous function of the portfolio weights.

Thirdly, PSO is insensitive to initial value. However, given the heuristic nature of PSO (or any AI-based optimization), accuracies are not as good as competing parametric methods. As a result, PSO is best used in high-dimensional problems where parametric methods fail. In this paper, we demonstrate how PSO can be used in evaluating complex derivatives. These complex derivatives usually require optimization through a high-dimensional search, which leads to failures (or highly inaccurate estimates) by the parametric methods. For the sake of easy exposition, we demonstrate simple American-style options such as put, put on min/max, and Asian options.

IV. American-style derivative Pricing

As mentioned earlier, once the exercise boundary can be correctly specified, one can perform Monte Carlo simulations to solve for American-style derivative prices. Moreover, with this capability, one can further solve path-dependent options which are impossible to be solved by lattice models. Also mentioned earlier is the difficulty admitted in the literature how to identify such a truly free exercise boundary. In this section, we demonstrate how to take advantage of PSO to achieve this goal. In PSO, there is no need to specify any functional form for the exercise boundary. Particles will collectively set the exercise boundary with no constraints, which in theory gives the best American-style derivative value.

We first demonstrate a simple American put option on one and two assets where there are accurate estimates via a lattice model. Then we demonstrate a path-dependent option (Asian option) that cannot be evaluated easily by lattice models.

1. Uni-variate

We first demonstrate how PSO is used in a simple American put option without dividends. In this simple example, we can have the lattice result (binomial model) as the benchmark. With the help of the binomial model, we can clearly see the exercise boundary of the option. The input information to the American put option is as follows:

stock price	100
strike price	100
volatility	0.3
risk-free rate	0.03
time to maturity	1
time steps	100
Monte-Carlo paths	10,000

Given that the binomial model and the PSO use the same number of time steps, we caution that the binomial model does not provide accurately enough result due to not enough steps (only 100).[24]

We use PSO to evaluate various boundary specifications. Specifically, for any given boundary specification (i.e. flat, linear, exponential, piece-wise flat, and restricted piece-wise flat), we maximize equation (7) over

[24] Certainly, we can increase the number of periods in the binomial model to achieve more accurate American values and yet this is not our main focus.

(18) $\max_{\Theta} \xi(t)$

where Θ represents the set of parameters of the boundary function and $\xi(t)$ is the option value defined in equation (7):

$$\xi(t) = \frac{1}{N}\sum_{j=1}^{N} e^{-r\tau_j} \max\{K - B(\tau_j), 0\}$$

For example, in the linear boundary case, $\Theta = \{a_0, a_1\}$. Hence, equation (18) is a two-dimensional search. Note that $B(\tau_j)$ is the boundary value of the jth path at time τ_j. Take a concrete example. Given a boundary specification (e.g. linear $B(t) = a_0 + a_1 t$), τ_5 (the fifth path) could be time step 26, and τ_{42} (the forty-second path) could be time step 74.[25] The boundary values are consequently $B(\tau_5) = a_0 + a_1 T_{26}$ and $B(\tau_{42}) = a_0 + a_1 T_{74}$. In other words, at the fifth path of Monte Carlo, the option is early exercised at step 26, and the exercise value is equal to $K - B(\tau_5) = K - (a_0 + 0.26a_1)$. Similarly, at the forth-second path of the Monte Carlo, the option is exercised early at step 74, and the exercise value is $K - B(\tau_{42}) = K - (a_0 + 0.74a_1)$.

Note that in the piece-wise flat boundary case, there is no formula and each time period has its own boundary value. In this case, $\Theta = \{B_1, \cdots, B_{100}\}$ and equation (18) is a 100-dimensional search. In PSO, each particle is labeled with 100 coordinates. The particles communicate with one another to update their coordinates (global best) at each iteration. Iterations stop when all particles converge to the same set of coordinates and equation (18) is maximized.

We compare different boundary conditions. The results are given in Table 1.

[Table 1 Here]

The European value is 10.3656 by the Monte Carlo method, which is a little higher than the true value 10.3278 by the Black-Scholes model. The binomial value for the European option is 10.2984 which is lower than the Black-Scholes value. Hence we can infer that the American value which is 10.5917 by the binomial model should be underestimated. Hence we can view the binomial value as a lower bound.

The piece-wise PSO value is 10.7714 which is the highest American value as expected as it imposes no restriction. The restricted (monotonically) piece-wise PSO value is the next highest as 10.6908. Given that the true exercise boundary is very close to an exponential function

[25] Note that time step 100, or T_{100}, is equal to the maturity time which is 1 (year) in the example. Hence $T_{26} = 0.26$ and $T_{74} = 0.74$.

(provided later in Figure 3), the exponential boundary result of 10.6647 should be very close to the true value.

The Longstaff-Schwartz value (regression, which uses equation (5)) is 10.6217 which is lower than the above three results but higher than the linear boundary result of 10.5621 and the flat boundary result of 10.5591. These results seem reasonable.

We then compare the exercise boundaries from the various specifications and compare them to the "true exercise boundary" implied by the binomial model. The exercise boundary is plotted in Figure 3.

[Figure 3 Here]

From Figure 3, it is clear that the exercise boundaries implied by the binomial, exponential, and piece-wise-monotonic cases are close to one another. The unrestricted piece-wise boundary is also close if we ignore the low values but only focus on the high values. The unrestricted piece-wise boundary oscillates but clearly those low values have little impact on the valuation (as we can see from the result that this boundary yields the highest American-style derivative value in Table 1). The flat and linear boundaries perform poorly (Table 1) as no surprise as they are far from the correct boundary.

Clearly, both PSO and binomial algorithms can be improved. First, the zigzag form of the binomial boundary is disturbing.[26] This could be due to insufficient number of periods (which confirms the slow convergence of the binomial model). Second, there are a substantial number of low values (at 60) by the unrestricted piece-wise boundary. It is clear that these values are bad values and yet it does not impact the valuation much, which indicates that the exercise boundary does not need to be granular. This is a numerical issue worthy of further investigation. Yet it is future research and beyond the scope of the current paper.

2. Multi-variate

There are a number of multi-variate lattice models. In principle, the challenge in building such a multi-dimensional lattice is the exploding memory usage and computation time. In the simplest case where all assets are uncorrelated, the number of nodes necessary for the lattice is $((m-1)t+1)^n$ where m is the number of economic states for any given asset and n is the number of assets and t is the number of time steps in the lattice. For example in a tri-nomial

[26] As mentioned in footnote 24, we can increase the number of steps in the binomial model to smooth the exercise boundary further.

lattice, 100 time steps to evaluate a three-asset derivative requires over 8 million nodes at maturity.[27]

Another challenge for building a multi-variate lattice is the difficulty in incorporating the number of pair-correlations of assets. In other words, it is not possible to match the number of equations (i.e. branches) and the number of unknowns (i.e. correlation pairs).[28] In the simplest case where assets are independent, we need 2^n branches (where n is the number of assets) in each time step. In order to incorporate correlation, Boyle (1988) and then modified by Kamrad and Ritchken (1991) devise a five-branch model. The corner branches have the same stock prices as before and the middle branch assumes the same stock prices as the current. By matching moments, there are six equations and five unknowns. Hence, the solution is not so straightforward. Boyle (1988) shows that the usual binomial setup with two assets X and Y,[29] that is, $X_u = X_0 u_X = X_0 e^{\sigma_X \sqrt{\Delta t}}$ and $X_d = X_0 d_X = X_0 e^{-\sigma_X \sqrt{\Delta t}}$ and similarly for Y. Due to the mismatch of equations and unknowns, he must alter the assumption to $u_X = e^{\lambda \sigma_X \sqrt{\Delta t}}$ and $u_Y = e^{\lambda \sigma_Y \sqrt{\Delta t}}$ where λ is free parameter so that he could solve for one stock first and then search for the solution to the second stock.

The second model by Boyle, Evnine and Gibbs (1989) is a four-branch model. As we can see that if we use four branches (i.e. four equations), we will not be able to match unknowns and equations. Hence, Boyle, Evnine and Gibbs turn to characteristic functions. They note that the above probabilities can all be nonnegative only if the time step becomes sufficiently small. Hence, this method is not very efficient.

Finally is the model by Chen, Chung, and Yang (2002). Their model is based upon complete markets. In a complete market, the number of nodes does not grow exponentially but factorially, which save both computation time and memory usage. Furthermore, the complete market setting is consistent with the binomial model in a single asset case and as a result risk-free no-arbitrage can be established. In other words, like the binomial model, the Chen-Chung-Yang model is not just a numerical algorithm as Boyle (1988), Boyle-Evnine-Gibbs (1989), and Kamrad-Ritchken, but also an economic model.

In the complete market setting, Chen, Chung, and Yang (2002) discover that the number of branches in each time step exactly matches the number of equations. Consequently, one can

[27] For 4 assets, it requires over 1.6 billion nodes.
[28] This problem has been solved by Chen, Chung, and Yang (2002). Later we adopt their model as the benchmark for options on multiple assets.
[29] We assume the readers are fairly familiar with the standard binomial model of Cox, Ross, and Rubinstein (1989). The notation used here is quite standard (e.g. see Hull (2015)) and straightforward.

easily solve for the probabilities as in the binomial model. While the readers can find all the details in their original paper, in the Appendix, we excerpt a two-asset example where the two-dimensional "binomial tree" can be visualized.

We evaluate the following put option (note that the call option will never be early exercised:

(19) $$V_\tau = \max\left\{K - \max\{S_{1\tau}, S_{2,\tau}\}, 0\right\}$$

with the parameters of the two stocks are given as:

	asset 1	asset 2
price	40	40
volatility	0.2	0.3
strike	35	
time to maturity	7/12	
risk free rate	0.03	
correlation	0.5	

Implementing the PSO algorithm, we recognize that there is a certain relationship between functions $B_{1,\tau}$ and $B_{2,\tau}$. For example, it could be: $B_{1,\tau} = a + bB_{2,\tau}$ (linear) or $a^2 B_{1,\tau}^2 + b^2 B_{2,\tau}^2 = c^2$ (elliptical/concave) where a, b and c are arbitrary constants, along with $B_{2,\tau}$ to be decided by PSO. In the current execution, we assume $B_{1,\tau}$ and $B_{2,\tau}$ to be independent.

The results are given in Table 2. Also in Table 2, we implement the Longstaff-Schwartz model with the following quadratic regression (compared to equation (5)):

(20) $$\xi_{t+1} = a_0 + a_{11}S_{1t} + a_{12}S_{1t}^2 + a_{21}S_{2t} + a_{22}S_{2t}^2 + a_3 S_{1t}S_{2t}$$

[Table 2 Here]

Similar to Table 1, we find the PSO results and the Longstaff-Schwartz result to be very close to each other. The Black-Scholes European value is 0.1948 and the binomial American value (i.e. Chen-Chung-Yang model) is 0.2557 with a European value as 0.1884. Hence we know that the American value by the binomial model is underestimated.

The Monte-Carlo European value is 0.1974 which is close to the Black-Scholes value. The Longstaff-Schwartz value is 0.2386 which is lower than the binomial value. Among all PSO values, again the unrestricted piece-wise boundary yields the highest value of 0.2426 followed by the exponential boundary of 0.2361. The flat boundary continues to be the worst case at a value of 0.2318. It is a little surprising to see that the exponential boundary yields a higher option value than the piece-wise-monotonic boundary of 0.2352.

3. Path-dependent

The lattice approach for the valuation of American-style derivatives does not apply to those contracts whose payoffs depend on past values (i.e. path-dependent options). On the other hand, Monte Carlo simulations are good for European path-dependent options. Yet, there has been no good approach to evaluate American path-dependent options.

Asian (Averaging) Option

We use the simple Asian option as a demonstration. An Asian option is an option whose payoff depends on a historical average (arithmetic or geometric, weighted or unweighted) of past values of the underlying asset. As a result, an Asian option cannot be evaluated using the standard lattice method in that a lattice does not keep track of the historical values of the underlying asset. As a result, a Monte Carlo algorithm must be employed. However, the Monte Carlo method cannot evaluate American style options. As a result, evaluating American style Asian options remains a challenge.

To date, there has been no other alternative to the Longstaff-Schwartz (1996) model which provides an approximation value to the American-style Asian option. In this paper, a more superior alternative, using PSO, is proposed.

First, we have to turn the valuation to a free-boundary problem. As discussed earlier, PSO is suitable to evaluate any free-boundary valuation problem. An American-style Asian option has the following payoff:

$$V_\tau = \max\{A_\tau - K, 0\} \tag{21}$$

where τ is the (early) exercise date and:

$$A_\tau = \frac{1}{n}\sum_{i=0}^{n-1} S_{\tau-i}$$

is the average of the stock price (in this example the average is arithmetic). An American-style Asian option is to compare the above exercise value against the continuation value. This nature, which is same for all American options, now is applicable to Asian options. In other words, there exists a critical value touching which triggers the early exercise. Hence, we can now use PSO to locate the exercise boundary.

Note that now the exercise boundary is located along the averaging value path A_τ. In Monte-Carlo simulations, this can be handled along each path with no difficulty. Valuation can be performed on A_τ just as it is on S_t. The results are in Table 3.

[Table 3 Here]

There is no closed-form solution to the European-style Asian option evaluated here. Neither is there a benchmark American value by the lattice model. Without knowing a benchmark, we cannot assess the accuracy of various PSO results and the Longstaff-Schwartz result. Hence, Table 3 can only provide a comparison between the results by Longstaff-Schwartz and PSO.

First, we can see that flat, linear, and exponential boundaries can hardly be accurate in that they generate an identical value to the American-style derivative (9.0117) which is very close to the European value (9.0109). Secondly, piece-wise boundaries, restricted and unrestricted both, provide substantially higher value than the other three cases, 9.1912 and 9.1925 respectively. This indicates that we obtain substantially higher value once the boundary function is flexible. Lastly, Longstaff-Schwartz value is the highest (9.2415) and yet it is unclear if their value overestimates or underestimates the true value. Hence, it is unable to assess the performance in this situation.

4. Computational Efficiency

In this section, we examine the issue of computation efficiency. In general, AI-based algorithms are not fast. As a result, computational efficiencies can be gained only in high dimensions. This is because the increase of dimensionalities and the increase of particles are both linearly proportional to computation time. This is sharply different from the traditional methods that suffer the well-known "dimensionality curse" where the increase of dimensions results in exploding computational time. As a result, there is no benefit in using an AI-based model in low dimensions.

Table 4 presents the results of (A) simple American put option and (B) American put option on two assets in various simulations. For 100 particles, the computation time ranges from 25.10 seconds to 46.88 seconds (with different seeds). Note that there is no clear relationship between the accuracy of values and speed. The fastest seed (#3143) takes 25.10 seconds but produces the second highest value; while the slowest seed (#41675) takes 46.88 seconds but produces the third highest value.

[Table 4 Here]

We have the following observations. First, given the heuristic nature of PSO, we provide results with various Monte Carlo seeds. As we can see the variation in results is non-trivial. Different Monte Carlo paths affect the results quite substantially. Fortunately, we can observe a pleasant pattern in mean (average across seeds), max (maximum across seeds) and min (minimum across seeds). In these results, more particles (higher swarm size) do take longer to compute and do converge to more accurate results (option values).

Secondly, and more interestingly, we do not find differences in computation times between Panel A which is option on single asset and Panel B which is option on two assets. This confirms the conjecture that PSO is not affected significantly by the number of assets. This is drastically different from previous models where dimensionality matters. For example, for a swarm size of 100, the mean computation times are 37.48 seconds for 1 asset and 40.82 seconds for 2 assets.

Lastly, note that one of the advantages of PSO is that computation time is linearly related to swarm size (number of particles). Hence, to increase accuracy, we can simply increase the swarm size and the cost only increases linearly. For example, the average speed for swarm size of 50 is 20.71 seconds and for swarm size of 500 is 229.67 which is roughly 10 times more (and similarly swarm size of 100 is roughly twice (46.88 seconds) and swarm size of 200 is four times (87.67 seconds).

V. Conclusion

In this paper, we demonstrate how complex (multi-asset or path-dependent) American-style derivatives can benefit from an artificial intelligent tool – PSO (particle swarm optimization). These options are otherwise nearly impossible to evaluate accurately and effiently. In other words, PSO is particularly suitable for evaluating these complex derivatives.

PSO is an optimization tool particularly suitable for high-dimensional problems. Compared to other optimization tools (e.g. stochastic gradient descend), PSO is intelligence-based. One can regard PSO (or any intelligence-based tools such as genetic algorithm and neutral networks) as "non-parametric" while other optimization tools (e.g. stochastic gradient descend) as "parametric. This analogy point out that PSO has more flexibility and can more likely find the better value.

Another extraordinary advantage of PSO is its capability in parallel computing. In other words, PSO can be GPU'ized (graphic processing unit). This indicates that the computation time of PSO can be infinitely minimized (by adding GPUs). The experiments on GPU computation is beyond the scope of this paper.

We also discover, presented in Table 4, PSO is quite sensitive to Monte Carlo paths. Particles behave quite differently in a different environment. This opens a door for another future research.

VI. References

Boyle, Phelim P., 1988 (March), "A Lattice Framework for Option Pricing with Two State Variables," Journal Of Financial And Quantitative Analysis Vol. 23, No. 1,1-12.

Boyle, Phelim P., Jeremy Evnine, and Stephen Gibbs, 1989 (April), "Numerical Evaluation of Multivariate Contingent Claims," The Review of Financial Studies, Volume 2, Issue 2, 241-250.

Carr, Peter, 1998, "Randomizing and the American Put," Review of Financial Studies, Vol 11, No 3, pp 597-626.

Carr, Peter, Jarrow, Robert A., and Myneni, Ravi, 2008 (January), "Alternative Characterizations of American Put Options," Financial Derivatives Pricing, pp. 85-103.

Chen, Ren-Raw, San-Lin Chung and Tyler T. Yang, 2002 (December), "Option Pricing in a Multi-Asset, Complete Market Economy," The Journal of Financial and Quantitative Analysis, Vol. 37, No. 4, 649-666.

Cox, J., S. Ross, and M. Rubinstein, 1979, "Option Pricing, A Simplified Approach," Journal of Financial Economics.

David S. Bunch and Herb Johnson, 2000 (October), "The American Put Option and Its Critical Stock Price," Journal of Finance, Vol. 55, No. 5, pp. 2333-2356.

Dorigo, Marco, Luca Maria Gambardella, 1997, "Ant Colony System: a Cooperative Learning Approach to The Traveling Salesman Problem," IEEE Transactions on Evolutionary Computation, , 1(1), 53- 66.

Dorigo, Marco, Vittorio Maniezzo, and Alberto Colorni, 1991, "Ant System: An Autocatalytic Optimizing Process," Technical Report 91-016.

Dorigo, Marco, Vittorio Maniezzo, and Alberto Colorni, 1996, "The Ant System: Optimization by a Colony of Cooperating Agents," IEEE Transactions on Systems, Man, and Cybernetics-Part B, Vol.26, No.1, pp.1-13.

Eberhart, Russell C., and James Kennedy, 1995, "A New Optimizer Using Particle Swarm Theory," Sixth International Symposium on Micro Machine and Human Science, IEEE.

Huang, Kaihua, 2019, "Particle Swarm Optimization Central Mass on Portfolio Construction," Gabelli School of Business, Fordham University.

Hull, John, 2015, Options, Futures and Other Derivatives, Prentice Hall.

Jamous, R.A., Al-Aguizy Tharwat, Essam El Seidy, and B.I. Bayoumi, 2015, "A New Particle Swarm with Center of Mass Optimization," International Journal of Engineering Research and Technology, 4(5), 312 – 317.

Kamrad, Bardia and Peter Ritchken, 1991, "Multinomial Approximating Models for Options with k State Variables," Management Science, vol. 37, issue 12, 1640-1652.

Kumar, Sajjan, Susmita Sau, Diptendu Pal, Bhimsen Tudu, Swadhin K. Mandal, Nilanjan Chakraborty, 2013, "Parametric Performance Evaluation of Different Types of Particle Swarm Optimization Techniques Applied in Distributed Generation System," Proceedings of the International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) pp 349-356.

Longstaff, Francis, and Eduardo Schwartz, 2001, "Valuing American-style derivatives by Simulation," Journal of Finance.

Nunes, João Pedro Vidal, Nunes, J., 2009, "Pricing American-style derivatives under the Constant Elasticity of Variance Model and Subject to Bankruptcy," Journal of Financial and Quantitative Analysis 44, 1231-1263.

Reynolds, Craig, 1987, "Flocks, herds and schools: A distributed behavioral model," Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, Association for Computing Machinery, pp. 25-34.

Zhang, Yudong, Shuihua Wang, and Genlin Ji, "A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications," Mathematical Problems in Engineering, Volume 2015, Article ID 931256, 38 pages.

VII. Figures and Tables

Table 1: American put

The option payoff is: $\max\{K - S\},0\}$

stock price	100
strike price	100
volatility	0.3
risk-free rate	0.03
time to maturity	1
time steps	100
Monte-Carlo paths	10,000

Put Option		
	European	American
Black-Scholes	10.3278	
binomial (CRR)	10.2984	10.5917
Longstaff-Schwartz	10.3656	10.6217
PSO-flat	10.3656	10.5591
PSO-linear	10.3656	10.5621
PSO-exponential	10.3656	10.6647
PSO-piece-wise	10.3656	10.7714
PSO-piece-wise(restricted)	10.3656	10.6908

Note: Monte Carlo results are based upon 10,000 paths, 100 time steps. The Longstaff-Schwartz model (1991) uses a quadratic function in the regression. The PSO uses a swarm size of 500. The two parameters of the PSO are (equation (12)): w = 0.5, c_1 = 0.5 and c_2 = 0.5. The computation stops when the improvement of the value is less than 10^{-6}. The binomial model is Cox-Ross-Rubinstein (1979) and is performed with 100 time steps. The performance of PSO is provided in Table 4.

Table 2: Put option on Min/Max

The option payoff is $\max\{K - \max\{S_1, S_2\}, 0\}$

	asset 1	asset 2
price	40	40
volatility	0.2	0.3
strike	35	
time to maturity	7/12	
risk free rate	0.03	
correlation	0.5	

Min/Max Option		
	European	American
BS	0.1948	
binomial (CCY)	0.1884	0.2557
Longstaff-Schwartz	0.1974	0.2386
PSO-flat	0.1974	0.2318
PSO-linear	0.1974	0.2349
PSO-exponential	0.1974	0.2361
PSO-piece-wise	0.1974	0.2426
PSO-piece-wise(restricted)	0.1974	0.2352

Note: Monte Carlo results are based upon 10,000 paths, 100 time steps. The Longstaff-Schwartz model (1991) uses a quadratic function in the regression. The PSO uses a swarm size of 500. The two parameters of the PSO are (equation (12)): $w = 0.5$, $c_1 = 0.5$ and $c_2 = 0.5$. The computation stops when the improvement of the value is less than 10^{-6}. The binomial model is Chen-Chung-Yang (2002) and is performed with 100 time steps. The performance of PSO is provided in Table 4.

Table 3: Path-dependent Asian Option

Payoffs $\max\{K - \overline{S}(T_1, T_2), 0\}$

where $\overline{S} = \frac{1}{n}\sum_{j=1}^{n} S_j$

Average Option		
	European	American
Longstaff-Schwartz	9.0109	9.2415
PSO-flat	9.0109	9.0117
PSO-linear	9.0109	9.0117
PSO-exponential	9.0109	9.0117
PSO-piece-wise	9.0109	9.1925
PSO-piece-wise(restricted)	9.0109	9.1912

Note: Monte Carlo results are based upon 10,000 paths, 100 time steps. The Longstaff-Schwartz model (1991) uses a quadratic function in the regression. The PSO uses a swarm size of 500. The two parameters of the PSO are (equation (12)): w = 0.5, $c_1 = 0.5$ and $c_2 = 0.5$. The computation stops when the improvement of the value is less than 10^{-6}. The binomial model is performed with 100 time steps. The performance of PSO is provided in Table 4.

Table 4: Performance of PSO

(A) Put Option (1-asset)

Value ($)													
Seed	**69905**	**80302**	**8249**	**26795**	**967**	**12128**	**81917**	**26488**	**3143**	**41675**	**Mean**	**Max**	**Min**
Swarm Size													
50	10.0368	9.7001	10.2470	10.1109	9.8039	10.1586	10.2712	10.5006	9.3164	9.4168	9.9562	10.5006	9.3164
100	10.0388	10.4329	9.2902	10.0156	10.5760	10.2345	10.5512	10.2584	9.8404	9.7683	10.1006	10.5760	9.2902
200	10.0744	9.7776	9.8149	10.5076	10.3039	10.6329	10.0999	10.7066	10.5164	9.8415	10.2276	10.7066	9.7776
500	10.5811	10.0202	10.5791	10.7459	10.7074	10.5237	10.4950	10.6977	10.7714	10.4127	10.5534	10.7714	10.0202

Computation Time (seconds)													
Seed	**69905**	**80302**	**8249**	**26795**	**967**	**12128**	**81917**	**26488**	**3143**	**41675**	**Mean**	**Max**	**Min**
Swarm Size													
50	19.6829	18.8187	18.8304	18.5710	18.3021	18.9599	18.8633	19.2746	18.8743	19.0645	18.9242	19.6829	18.3022
100	37.3748	36.9218	38.1397	36.7419	37.8845	37.1103	37.9693	37.1976	37.7589	37.7402	37.4839	38.1397	36.7419
200	75.1492	75.1461	77.2891	75.0346	74.6849	75.8589	76.6391	73.8078	76.6213	73.0110	75.3242	77.2891	73.0110
500	186.9830	185.8080	178.4430	182.4440	186.9290	185.8810	184.2930	183.5500	185.3380	178.0620	183.7732	186.9831	178.0624

(B) Min/Max Option (2-asset)

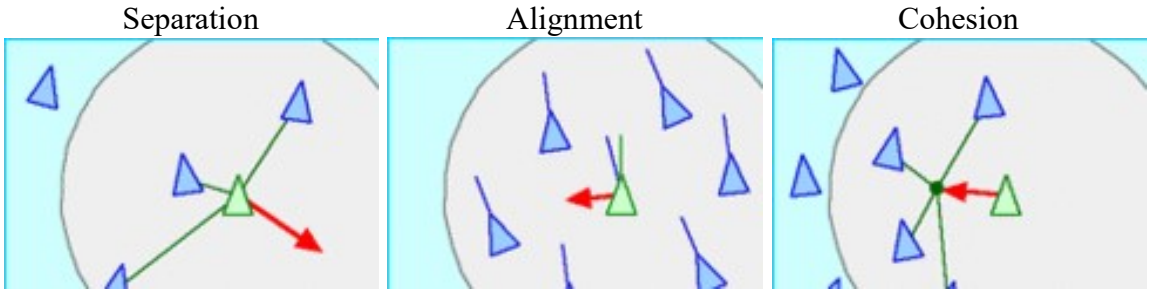
Value ($)													
Seed	**69905**	**80302**	**8249**	**26795**	**967**	**12128**	**81917**	**26488**	**3143**	**41675**	**Mean**	**Max**	**Min**
Swarm Size													
50	0.2268	0.2358	0.2350	0.2368	0.2374	0.2347	0.2383	0.2393	0.2257	0.2353	0.2345	0.2393	0.2257
100	0.2334	0.2344	0.2398	0.2379	0.2364	0.2342	0.2342	0.2381	0.2384	0.2382	0.2365	0.2398	0.2334
200	0.2383	0.2380	0.2368	0.2315	0.2374	0.2335	0.2338	0.2387	0.2409	0.2320	0.2361	0.2409	0.2315
500	0.2359	0.2342	0.2366	0.2412	0.2413	0.2426	0.2414	0.2392	0.2416	0.2362	0.2390	0.2426	0.2342

Computation Time (seconds)													
Seed	**69905**	**80302**	**8249**	**26795**	**967**	**12128**	**81917**	**26488**	**3143**	**41675**	**Mean**	**Max**	**Min**
Swarm Size													
50	10.7312	20.6707	20.5986	20.5549	20.7069	16.1223	20.6824	20.6649	6.97153	20.7073	17.8411	20.7073	6.9715
100	41.1003	44.5624	45.634	41.0282	41.201	40.7916	41.1614	40.6881	25.1049	46.8837	40.8156	46.8837	25.1049
200	87.6694	73.1363	78.1617	83.9155	86.7515	73.6531	81.172	86.2793	81.5649	40.9068	77.3211	87.6694	40.9068
500	203.568	202.788	219.504	229.666	222.398	202.404	210.968	203.651	207.758	202.525	210.5230	229.6660	202.4040

Note: The two parameters of the PSO are (equation (12)): $w = 0.5$, $c_1 = 0.5$ and $c_2 = 0.5$. The computation stops when the improvement of the value is less than 10^{-6}. The Longstaff-Schwartz value is (Table 2) 0.2386. The binomial model value is (Table 2) 0.2557.

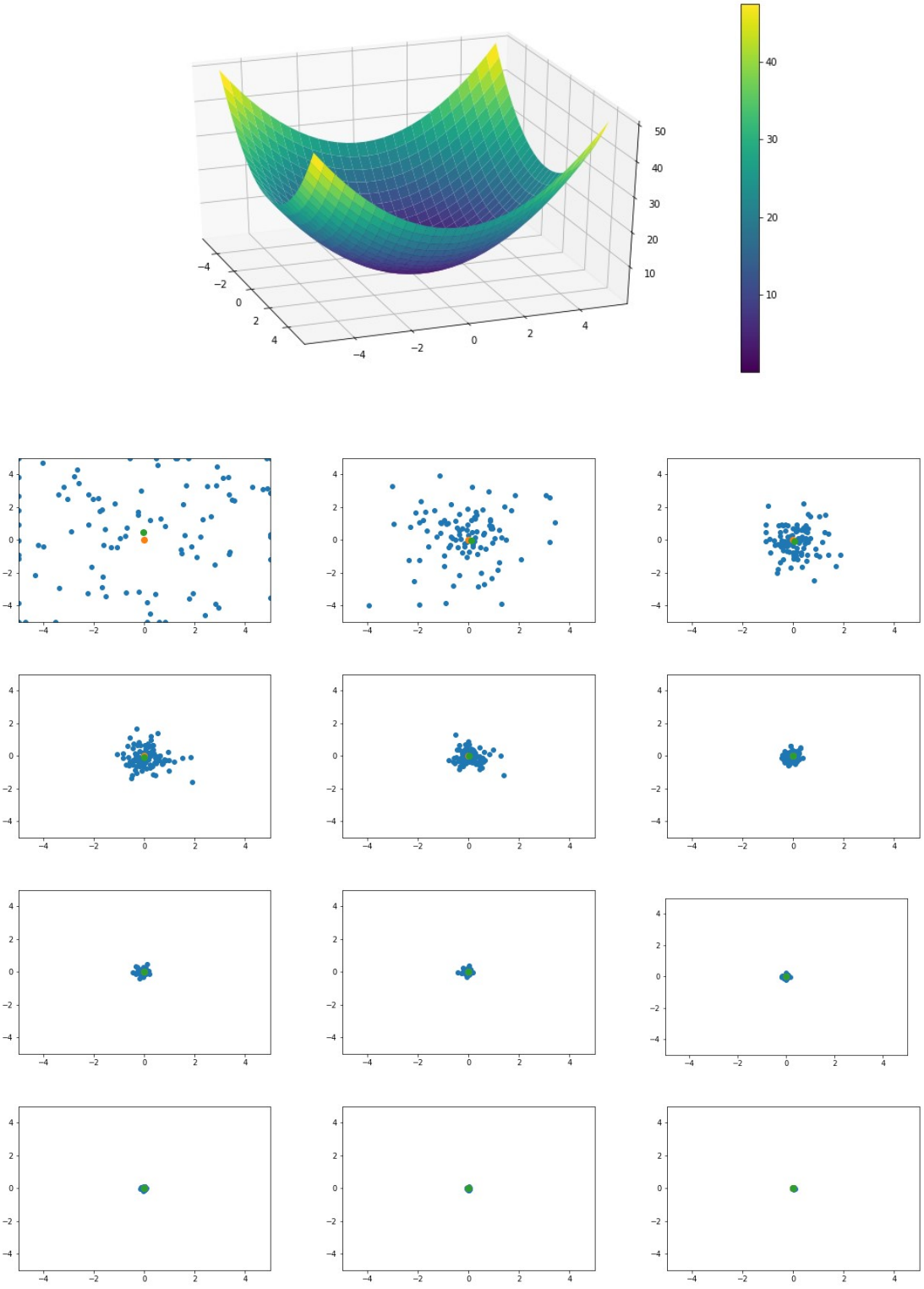
Figure 1

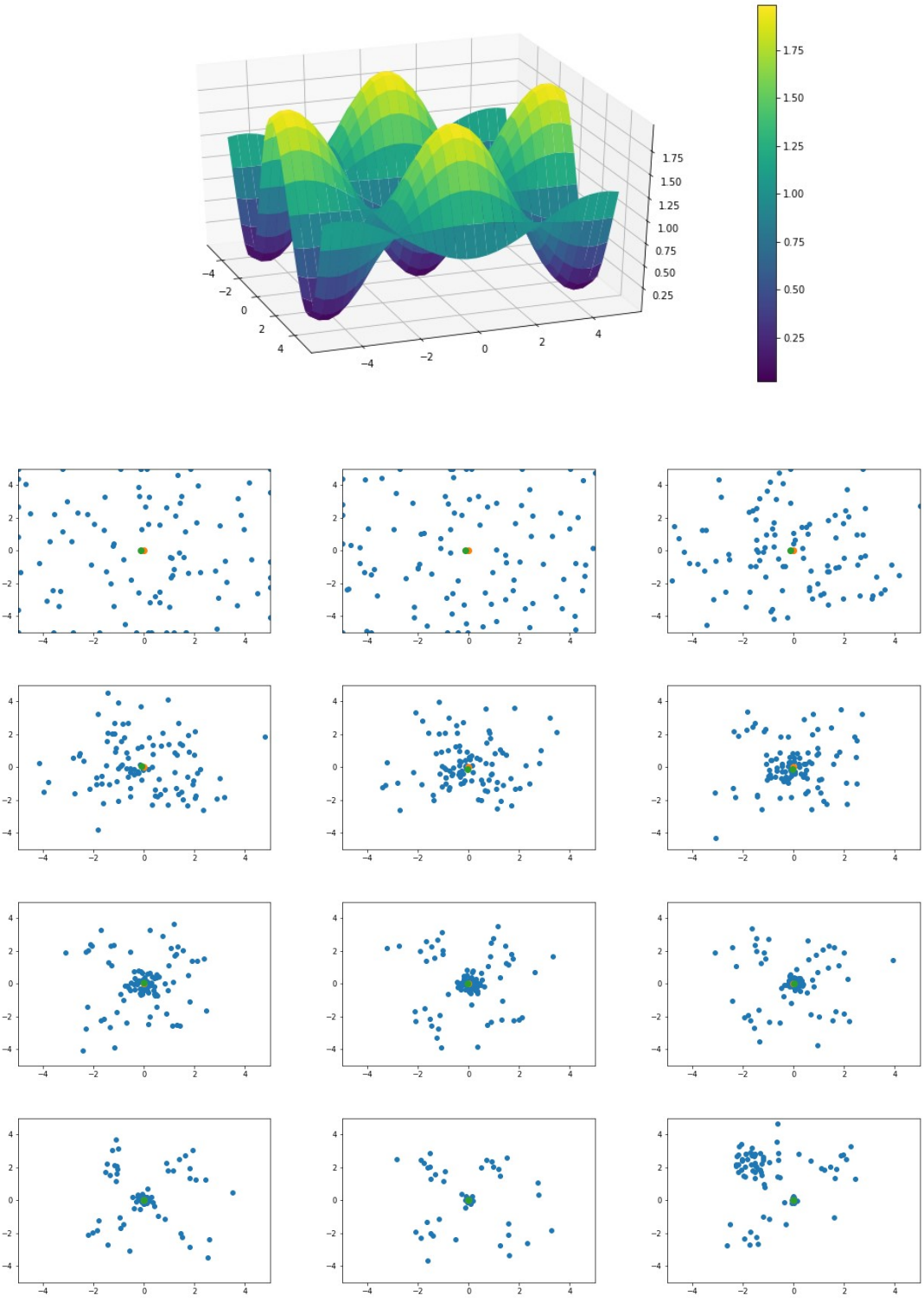
Three major parameters in a swarm.



Sources: https://en.wikipedia.org/wiki/Boids

Figure 2

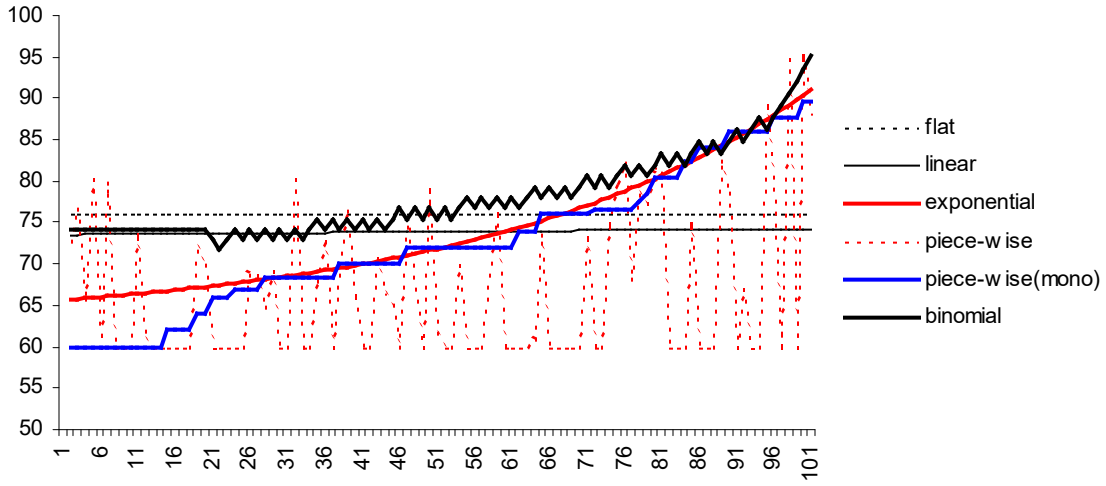




For an animated demonstration, see for example, https://en.wikipedia.org/wiki/Particle_swarm_optimization#/media/File:ParticleSwarmArrowsAnimation.gif

Figure 3

Various Exercise Boundaries.



VIII. Appendix

1. American Call Option on Min/Max Will Never be Early Exercised

Jansen's Inequality states that ff $f(x)$ is a convex function then $f(E[x]) \leq E[f(x)]$ and vice versa. Hence the American call option will never be early exercised. It is wellknown that the simple call option's continuation value is always greater than the exercise value:

$$\begin{aligned} e^{-r\Delta t}E[\max\{S_T - K, 0\}] &\geq e^{-r\Delta t}\max\{E[S_T] - K, 0\} \\ &= \max\{e^{-r\Delta t}E[S_T] - e^{-r\Delta t}K, 0\} \\ &> \max\{S_{T-1} - K, 0\} \end{aligned}$$

Also the exchange option will never be exercised.

$$\begin{aligned} e^{-r\Delta t}E[\max\{S_{1,T} - S_{2,T}, 0\}] &\geq e^{-r\Delta t}\max\{E[S_{1,T}] - E[S_{2,T}], 0\} \\ &= \max\{e^{-r\Delta t}E[S_{1,T}] - e^{-r\Delta t}E[S_{2,T}], 0\} \\ &= \max\{S_{1,T-1} - S_{2,T-1}, 0\} \end{aligned}$$

Finally the min/max call option will never be exercised:

$$\begin{aligned} e^{-r\Delta t}E[\max\{\max\{S_{1,T}, S_{2,T}\} - K, 0\}] &\geq e^{-r\Delta t}\max\{E[\max\{S_{1,T}, S_{2,T}\}] - K, 0\} \\ &\geq e^{-r\Delta t}\max\{\max\{E[S_{1,T}], E[S_{2,T}\} - K, 0\} \\ &> \max\{\max\{S_{1,T-1}, S_{2,T-1}\} - K, 0\} \end{aligned}$$

2. Option on Min/Max

The closedform solution to the put option on min/max can be derived from the call option solutions provided by Stulz (1988). Our objective is to derive the closedform solution to $P = \max\{K - \max\{S_1, S_2\}, 0\}$ from the Stulz solution to $C = \max\{\max\{S_1, S_2\} - K, 0\}$. The following payoff analysis demonstrates that:

	$C = \max\{\max\{S_1, S_2\} - K, 0\}$	$P = \max\{K - \max\{S_1, S_2\}, 0\}$	$C - P$
$S_1 > S_2$	$\max\{S_1 - K, 0\}$	$\max\{K - S_1, 0\}$	$S_1 - K$
$S_1 < S_2$	$\max\{S_2 - K, 0\}$	$\max\{K - S_2, 0\}$	$S_2 - K$
	$\max\{\max\{S_1, S_2\} - K, 0\}$	$\max\{K - \max\{S_1, S_2\}, 0\}$	$\max\{S_1, S_2\} - K$

As a result, we have:

$$\begin{aligned} C(T) - P(T) &= \max\{S_1(T), S_2(T)\} - K \\ &= S_2(T) + \max\{S_1(T) - S_2(T), 0\} - K \end{aligned}$$

Given that this is a European option, we can discount it back to today and have:

$$P(t) = C(t) - S_2(t) - X(S_1, S_2) + e^{-r(T-t)}K$$

where $X(S_1, S_2)$ is the standard exchange option. Stulz presents the call option on max/max as follows:

$$\max\{\max\{S_1, S_2\} - K, 0\} = C_{BS}(S_1) + C_{BS}(S_2) - M(S_1, S_2)$$

where S_1 and S_2 are the two underlying assets, $C_{BS}(\cdot)$ is the BlackScholes call option on a given underlying asset, and $M(S_1, S_2)$ is given as:

$$\begin{aligned} M(S_1, S_2) &= \max\{\min\{S_1, S_2\} - K, 0\} \\ &= S_1 N_2(a_1, b_1; \rho_1) + S_2 N_2(a_2, b_2; \rho_2) - K e^{-r(T-t)} N_2(g_1, g_2, \rho_{12}) \end{aligned}$$

where

$$a_j = g_j + \sigma_j \sqrt{T-t}$$

$$\begin{aligned} b_1 &= \frac{\ln S_1 - \ln S_2 - \tfrac{1}{2}\sigma^2 (T-t)}{\sigma\sqrt{T-t}} \\ b_2 &= \frac{\ln S_2 - \ln S_1 - \tfrac{1}{2}\sigma^2 (T-t)}{\sigma\sqrt{T-t}} \end{aligned}$$

$$g_j = \frac{\ln S_j - \ln K + (r - \tfrac{1}{2}\sigma_j^2 (T-t))}{\sigma_H \sqrt{T-t}}$$

$$\begin{aligned} \rho_1 &= \frac{\rho_{12}\sigma_1 - \sigma_2}{\sigma} \\ \rho_2 &= \frac{\rho_{12}\sigma_2 - \sigma_1}{\sigma} \end{aligned}$$

$$\sigma^2 = \sigma_1^2 + \sigma_2^2 - 2\rho_{12}\sigma_1\sigma_2$$

and ρ_{12} is the correlation between S_1 and S_2.

3. Illustration of the Chen-Chung-Yang Model

Here we illustrate how to implement the Chen-Chung-Yang model to evaluate American-style derivatives on multiple assets. A geometrical demonstration is provided for the two-asset case as follows:

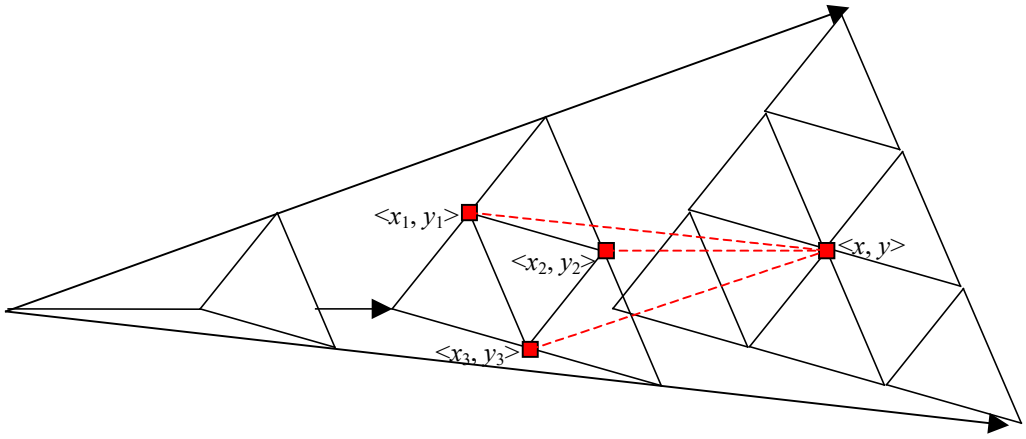


Figure A: A Two-asset Chen-Chung-Yang Model

In the above demonstration, as we travel along the lattice forward, the number of nodes increases in the following geometric series: $j = \frac{(i+1)(i+2)}{2!}$ where $i = 1, 2, \cdots, n$ as the time steps of the lattice. The general case for m number of assets is: $j = \frac{1}{m!}\Pi_{k=1}^{m}(i+k)$ as in the following table:[30]

num of assets	m=1	m=2	m=3	...	m=m
i	j	j	j		j
0	1	1	1		1
1	2	3	4		5
2	3	6	10		15
3	4	10	20		35
...					
n	$n+1$	$\frac{(n+1)(n+2)}{2!}$	$\frac{(n+1)(n+2)(n+3)}{3!}$	...	$\frac{\prod_{k=1}^{m}(n+k)}{m!}$

To implement the model as described in Figure A, we index the states as follows (where the first subscript is time and the second is state):

$$\begin{bmatrix} x_{01} & y_{01} \end{bmatrix} \qquad \begin{bmatrix} x_{11} & y_{11} \\ x_{12} & y_{12} \\ x_{13} & y_{13} \end{bmatrix} \qquad \begin{bmatrix} x_{21} & y_{21} \\ x_{22} & y_{22} \\ x_{23} & y_{23} \\ x_{24} & y_{24} \\ x_{25} & y_{25} \\ x_{26} & y_{26} \end{bmatrix}$$

[30] Note that even in the simplest independence case, the number of nodes at the time step n is $(n+1)^m$. For example, for three periods, a four-asset model has 256 nodes as opposed to 35 nodes in the CCY model.

In a general case where we move from any time i to time $i+1$, state j will become $< j_0, j_1, j_2 >$ as follows:

$$(i,j) \rightarrow \begin{cases} (i+1, j_0) \\ (i+1, j_1) \\ (i+1, j_2) \end{cases}$$

where

$$\begin{aligned} j_0 &= j \\ j_1 &= \frac{i(i+1)}{2} + k \\ j_2 &= j_1 + 1 \end{aligned}$$

As $i = 1 \sim n$, we have:

$$\begin{aligned} j &= \left\{ \frac{(i-1)i}{2} + 1 \right\} \sim \left\{ \frac{i(i+1)}{2} \right\} \\ k &= k+1 \qquad (1 \sim i) \end{aligned}$$