Optimizing Technical Trading Parameters Using Particle Swarm Optimisation Algorithm

Darren Logue (dazlog1975@gmail.com)
Author Affiliation(s)
E-mail

Abstract

This paper proposes a breakout trading strategy using Particle Swarm Optimization to optimize the parameters (entry, exit etc). The set of parameters is then used to determine trading decisions, i.e. long, short etc. The model is trained and back tested on historical stock prices from the NYSE. The performance is evaluated using the return on investment using the training set. The results are then compared with buy-and-hold strategy over the same period.

Keywords: *Particle Swarm Optimisation, Breakout Strategy, Metaheuristics, Trading*

1. Introduction

Quantitative researchers encounter problems of increasing complexity in the financial markets. They are often asked to solve an optimisation problem with respect to given parameters They are looking to minimize/maximize an objective function e.g maximize returns or to minimize drawdowns etc Classical optimisation techniques are a convenient way of discovering the minima or maxima of continuous, differentiable functions. They do however, have restricted scope in many practical scenarios as they may encounter non-continuous and/or non-differentiable objective functions.

In recent years a new class of optimization methods, referred to as Metaheuristics has appeared. They are a group of stochastic algorithms which can be adapted to a vast number of problems (such as those mentioned above). In this review I will concentrate on those algorithms based upon swarm intelligence, most notably Particle Swarm Optimisation (PSO)

Initially introduced by Kennedy and Eberhart in 1995. The basic idea behind PSO is to simulate a swarm of birds or the behavior of a fish school. It is known for efficiently finding optimal or near-optimal solutions in large search spaces. On a number of problems, PSO has shown to be faster at converging than other evolutionary algorithms . The original formula was improved by Shi and Eberhart with the introduction of an Random Inertia Weight. They found through experimentation that it increases convergence in the earlier iterations. It was not impossible for a particle to be able to move out of the search space either. Eberhart et al. (1996) countered this with the introduction of a 'maximum velocity' parameter.

The swarm is composed of a number of particles. Each with a velocity $V_i = (v_1, v_2, \ldots, v_N)$ a position $X_i = (x_1, x_2, \ldots, x_N)$ that are exploring the multi-dimensional problem space in search of solutions.

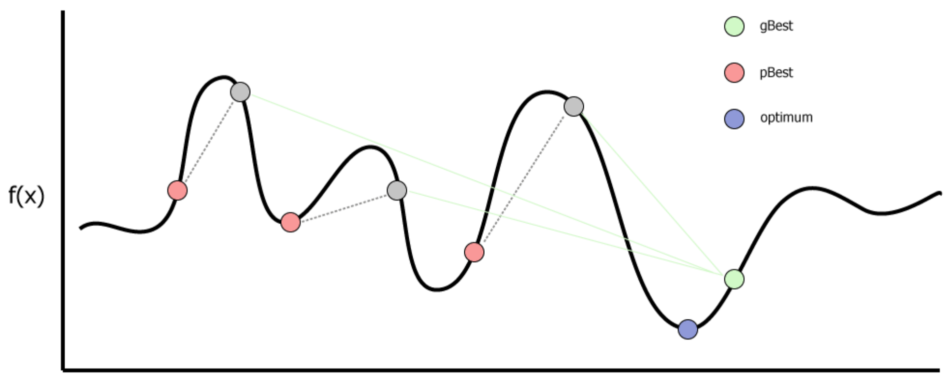
The PSO algorithm is initially set up with a set of random particle values. The positions and velocity of a particle is dictated by the following factors:

- inertia weight
- personal best/cognitive force (pBest)
- global best/social force (gBest)

Inertia allows for better control over the breadth of the search. Large values let particles move with large velocities making searches much quicker (exploration). Low values result in the particles searching a more limited neighbourhood (exploitation)

PSO re-evaluates every particle on each iteration by updating both the personal and global 'bests'. Personal best being the 'best' value the particle has accomplished and global best being the 'best' value achieved by the swarm.

The diagram below shows how each particle position in the swarm is re-evaluated in respect to the global best (green) and personal best positions (red)



adapted from http://www.turingfinance.com/portfolio-optimization-using-particle-swarm-optimization

Once we have these two values we can update, the position and velocity of the particles via the following equations:

$$V_i^{n+1} = \omega \cdot V_i^n + c_1 \cdot r_i^n \cdot (P_i^n X_i^n) + c_2 . r_2^n . (P_g^n X_i^n) \quad (1)$$

$$X_i^{n+1} = X_i^n + V_i^{n+1} \quad (2)$$

Where:

1. X_i - position of the ith particle;
2. V_ii - velocity of the ith particle;
3. P_i^n - previous best particle of the ith particle

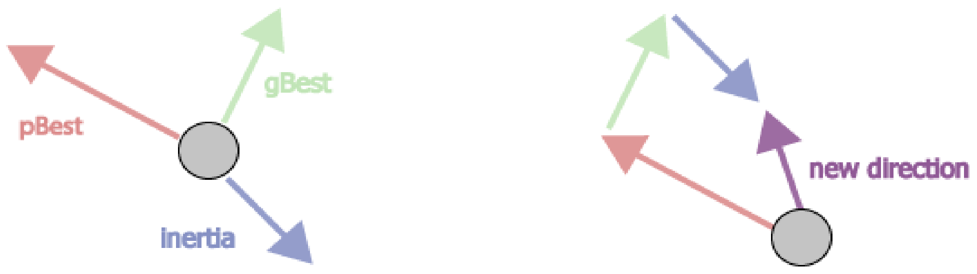
4. P_g^n - global best particle found by all particles so far.
5. r_1 and r_2 are two random vector within [0, 1],
6. ω - inertia weight,
7. c_1 and c_2 are two learning factors,

adapted from https://springerplus.springeropen.com/articles/10.1186/2193-1801-2-315

The terms in equation (1) can be evaluated as follows: the first term refers to inertia, the second recognises the 'cognitive' force of pBest and the third term indicates the 'social' force of gBest. c_1 and c_2 control the influence of pbest and gbest.

High c_1 values result in each particle being attracted to a different position, meaning the swarm is widely spread. A high c_2 value results in the particles converging to the present gbest position.

Each particle adjusts its position according to its own experience and that of the swarm.

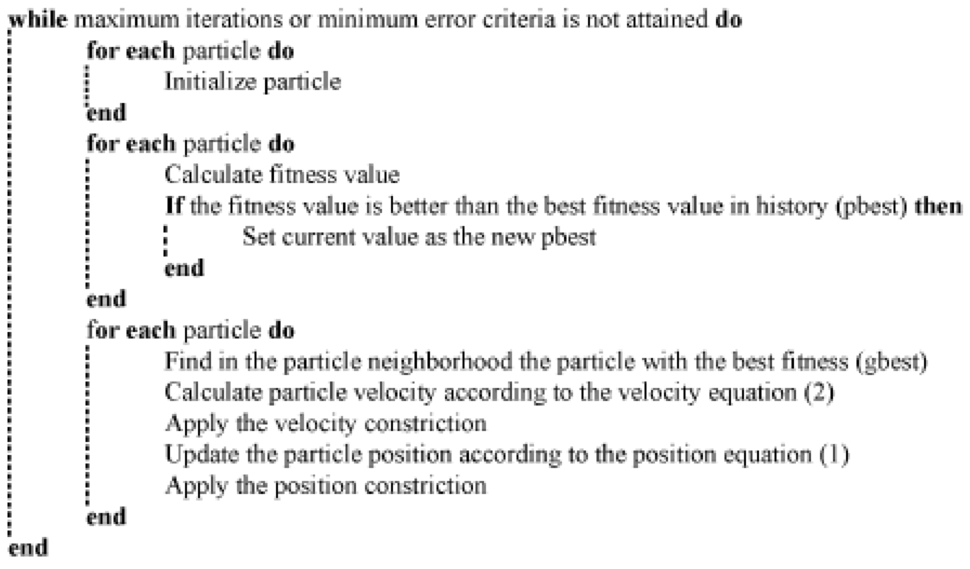


adapted from https://nathanrooy.github.io/posts/2016-08-17/simple-particle-swarm-optimization-with-python

Each particle will move to a new position according to equation (2). This repeats iteratively they all converge to the optimum

The following pseudo code best illustrates how the algorithm actually works

PseudoCode of PSO algorithm



```
while maximum iterations or minimum error criteria is not attained do
        for each particle do
                Initialize particle
        end
        for each particle do
                Calculate fitness value
                If the fitness value is better than the best fitness value in history (pbest) then
                        Set current value as the new pbest
                end
        end
        for each particle do
                Find in the particle neighborhood the particle with the best fitness (gbest)
                Calculate particle velocity according to the velocity equation (2)
                Apply the velocity constriction
                Update the particle position according to the position equation (1)
                Apply the position constriction
        end
end
```

adapted from https://springerplus.springeropen.com/articles/10.1186/2193-1801-2-315

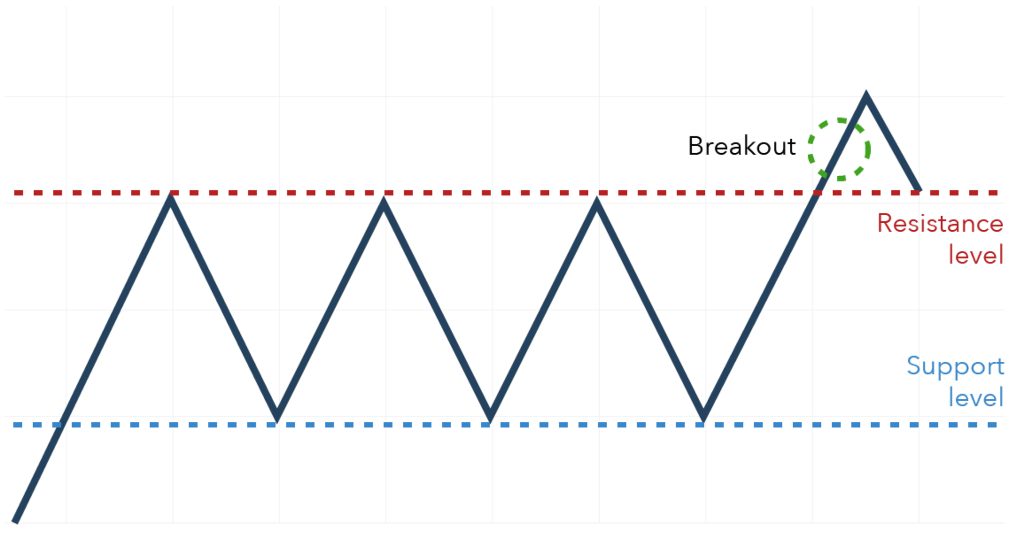
The maximum velocity parameter was introduced to bypass any chance of the algorithm diverging. Following this, a number of researchers such as Kennedy (1998), Ozcan et al. (1999) and Van den Bergh, (2002) investigated guaranteed convergence of the swarm. Clerc et al (2002) proposed using a 'constriction factor' on the algorithm's parameters (as noted in the psuedo code above) to guarantee swarm convergence. Van den Bergh (2002), Trelea (2003), Peer et al. (2003) and Zheng et al. (2003) also proposed variations on the PSO that would allow it to converge to a steady state.

Ray et al., (2002) proposed the use of a density matrix on the particle neighbourhood to encourage swarm diversity. Coello et al. (2002, 2004) proposed the use of Pareto dominance and splitting the search space into hypercubes to determine the particle flight direction

Srinivasan et al.(2003)) combined evolutionary algorithms to counter the weaknesses, and gain the strengths of each other. Because PSO is very adaptable it has been used as the basis of several hybrid optimization methods. A number of these are listed in the appendix below.

Breakout trading

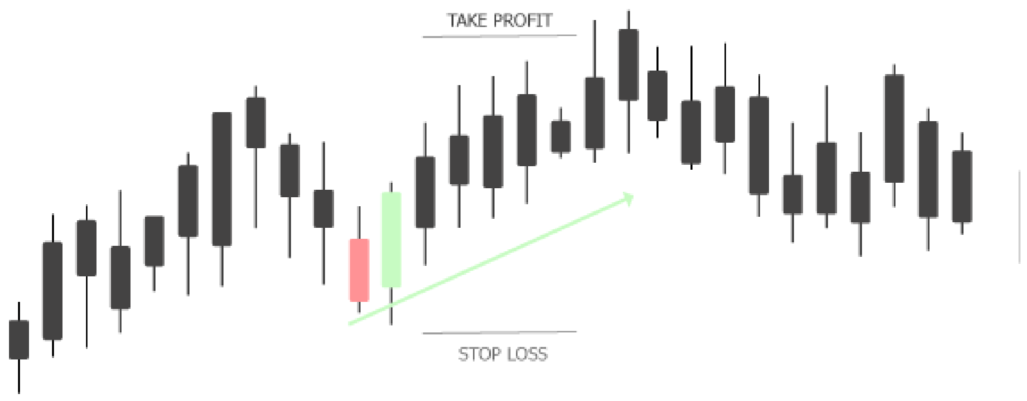
Breakout trading effectively tries to take a position within a trend's initial phase. The strategy can be the jump off point for large price movements or volatility spreads. Due to using stop losses the downside risk minimised if the strategy is applied appropriately



A 'breakout' can be defined as a price movement outside of a set trendline level accompanied by elevated volume. Typically the if price rises above the 'resistance' level the trader will enter a long position. If however, the price falls below the level set by the trendline a short position will be entered. The key to this strategy is clearly entry and exit points.

Trading Strategy Algorithm

The strategy is very simple, we are attempting to capture the point at which the price breaks out of the trendline in an inverse direction. Two pivot points establish the line. In the case of the upside line the next down pivot point will be above the last one



The 'down' pivot will be the low price of the bar with the lowest price set in a range of at least two bars on either side. The opposite is true for an 'up' pivot

The long position entry point happens when the price traverses the downside trend line (closing beyond this). The opposite of this will indicate the short position entry point.

Take profit and stop loss points will be preset by the model in order to determine the exit points

Parameters to be evaluated

- the number of bars (for pivot point),
- number of bars multiplication factor,
- take profit level,
- stop loss level.

The swarm size and number of iterations for the experiment will likely be determined by processing power. The defaults are set at 100. This may prove to be too costly in terms of time when running on a local machine. It may be necessary to run this via a third party (such as Google CoLab etc)

Methodology

- Download data from appropriate source
- Split into training & test sets (80/20)
- Write code for strategy, PSO and objective function
- Train the model to achieve the optimal parameters
- This will be backtested using Python library Backtrader
- Backtest again using the test set
- Evaluate the data

The PSO strategy will be evaluated against a buy and hold strategy over the exact same period. It will be compared using a number of common metrics such as ROI, drawdowns VaR etc

Note to reviewer: The scope of the project will have to be evaluated after an element of the code has been written in order to see what is viable given processing power. May have to limit number of stocks etc

References:

Coello et al. (2002) [Coello Coello et al., 2002] C.A. Coello Coello, D.A. Veldhuizen, G.B. Lamont. (2002). Evolutionary Algorithms for Solving Multi-Objective Problems,

R.C. Eberhart, J. Kennedy, “A new optimizer using particle swarm theory,” Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, 1995, pp. 39–43.

J. Kennedy, R.C. Eberhart, “Particle swarm optimization,” Proceedings of the IEEE International Conference on Neural Networks, Piscataway, NJ, USA, 1995, Vol. 4, pp. 1942–1948.

Kennedy, J., Eberhart, R.C., Swarm Intelligence. Morgan Kaufmann Publishers. 200

Kennedy (1998) [Kennedy, 1998] J. Kennedy. (1998). The behavior of particles, Proceedings of the 7th Conference on Evolutionary Computation, pp. 581-589, LNCS, Springer.

Ozcan et al. (1999) [Ozcan et al., 1999] E. Ozcan, C.K. Mohan. (1999). _Particle Swarm Optimization : surfing the waves_, Proceeding of the 1999 IEEE Congress on Evolutionary Computation, pp. 1939-1944, IEEE Press.

Ray et al., (2002) [Ray et al., 2002] T. Ray, K.M. Liew. (2002). _A swarm metaphor for multiobjective design optimization _, Engineering Optimization, Vol. 34, N°2, pp. 141-153, 2002.

Srinivasan et al. (2003) [Srinivasan et al., 2003] D. Srinivasan, T.H. Seow. (2003). _Particle swarm inspired evolutionary algorithm for multiobjective optimization problem_, Proceedings of the 2003 Congress on Evolutionary Computation, Vol. 3, pp. 2292-2297, IEEE Press.

Van den Bergh (2002) [Van den Bergh, 2002] F. Van den Bergh. (2002). _An analysis of Particle Swarm Optimizers_, PhD thesis, University of Pretoria, South Africa, 2002.

Zheng et al. (2003) [Zheng et al., 2003] Y. Zheng, L. Ma, L. Zhang, J. Qian. (2003). _On the convergence analysis and parameter selection in particle swarm optimization_, Proceedings of International Conference on Machine Learning and Cybernetics, 2003, pp. 1802-1807, IEEE Press.

Yuhui Shi and Russell Eberhart A Modified Particle Swarm Optimizer Department of Electrical Engineering Indiana University Purdue University Indianapolis

Inertia Weight Strategies in Particle Swarm Optimization J. C. Bansal, P. K. Singh Mukesh Saraswat, Abhishek Verma, Shimpi Singh Jadon, Ajith Abraham Indian Institute of Information Technology & Management, Gwalior, India

Psuedo code and mathematical formulas adapted from

https://springerplus.springeropen.com/articles/10.1186/2193-1801-2-315