**Overview of the Discretized dynamical system**

The DDS is a method which can approximate a solution to a constrained optimization problem whereby we have a weighted function of many variables that we can pick values and simulate from.

Take the example with stocks in a portfolio where our objective function is the Sharpe ratio:

Sharpe ratio for a portfolio P is:

SRP = (ER(P) - rf)/σp

P = x1S1 + x2S2 + …+ xmSm

Where x = (x1,x2,…, xm) are the weights that we invest in each stock, they all sum to 1:

Σj=1m(xj) = 1

Sharpe ratio for a stock Sj is:

SRS = (ER(Sj) - rf)/σSj

We know that diversification reduces the variance of the portfolio, and hence there is value in holding multiple (if not all) stocks in a portfolio.

This is where the DDS is useful, in situations where the objective function has trade-offs and holding multiple stocks (in this case) creates synergy.

In other words, if x1 had the highest individual Sharpe ratio, the Sharpe ratio of the portfolio would not be monotonically increasing with x1 over [0,1] (increasing x1 would simultaneously decrease the weightings in other stocks).

Here, we are going to assume that we have estimates for the expected return and variance of each individual stock and that these remain constant throughout us performing the algorithm, hence, the Sharpe ratio of the portfolio can be expressed as a function of the weightings only:

SRP = (ER(P) - rf)/σp = f(x)

**DDS algorithm overview**

Let xopt = (x1opt, x2opt, …, xmopt) be the vector of optimal weighting to maximize a general objective function OB that satisfied the above conditions.

I.e. OB = OB(x), argmax(OB(x)) = xopt

We are going to be using a stepwise algorithm, denoting = (1, 2, …, m)as an estimate for xopt.

Define xijk as the proportion on combination number i, on stock j, whilst in phase k for:

i = 1, 2, …,n

j = 1, 2, …,m

k = 1, 2, …,m

Let me explain this further, we start the algorithm in phase 1, on phase 1 we focus solely on x1, the rest of the values are equal are determined automatically by the normalizing constraint Σj=1m(xj) = 1.

Before the algorithm starts, we pick a number for n, and in the first step a linear space is formed on [0,1] for x1, namely:

x0, x1, …, xn (or xi for i = 1, 2, …, n) such that:

xi = i/n.

For example, in the case with stocks in a portfolio, if n=100, in phase 1 we would test out 101 values for x1, namely:

0.00, 0.01, 0.02, …, 0.99, 1.00

In each 101 cases, x2 to xm would be chosen such that they are equal, and also the normalizing constraint is satisfied.

We would now evaluate OB(x) for each 101 unique values of x, we would identify the one that is largest, then find the value of xi value that corresponds to this maximal value of OB. We would then set x1 = 1. Phase 1 is complete.

Each phase varies the corresponding stock number discretely and allows the normalizing constraint (or the already set values if k≥2) to take care of the rest.

We complete the algorithm by carrying out each phase in turn, each setting a value for j

**DDS algorithm**

Now that I have explained the process, we now wish to express xijk for all i, j, k:

xi11 = i/n, for i = 1, 2, …, n

xij1 = (1 – xi11)/(m – k), for j = 2, 3, …, m

Now we would like to generalize this for all i,j,k:

1. xijk = ijk = j, when j<k
2. xijk = i/n (= xikk), when j=k
3. xijk = (1 - Σj=1k-1( j) – xikk)/(m-k)

The above 3 equations are what I call the “master equations”, this is because they govern the entire algorithm. The code to simulate this is mostly complete, but can be found in the repository.

There is of course a problem with this algorithm, as it attempts to approximate the global maxima through stepwise selection, it doesn’t test all the combinations.

A workaround to this would be to order the variables in the vector x in every possible combination, perform the DDS algorithm, re-order, and then average out each estimate of x1,x2,…,xm. This however, will become computationally infeasible for large m.

A workaround to this would be to randomly order the variables in the vector x before we perform the DDS algorithm, a number that is less than m! and computationally feasible. This should get us closer to the global maxima.

This algorithm can be applied in many scenarios, for example, in selecting the optimal copy, image and audience for a Facebook advert given a product or set of products that a client has to offer.