A ROBUST ESTIMATOR OF THE EFFICIENT FRONTIER

Marcos López de Prado

This version: October 15, 2019

Marcos López de Prado is CIO of True Positive Technologies, LP, in New York, NY, and Professor of Practice at Cornell University's School of Engineering, in Ithaca, NY. E-mail: mldp@truepositive.com.

A ROBUST ESTIMATOR OF THE EFFICIENT FRONTIER

ABSTRACT

Convex optimization solutions tend to be unstable, to the point of entirely offsetting the benefits of optimization. For example, in the context of financial applications, it is known that portfolios optimized in-sample often underperform the naïve (equal weights) allocation out-of-sample. This instability can be traced back to two sources: (i) noise in the input variables; and (ii) signal structure that magnifies the estimation errors in the input variables. A first innovation of this paper is to introduce the nested clustered optimization algorithm (NCO), a method that tackles both sources of instability.

Over the past 60 years, various approaches have been developed to address these two sources of instability. These approaches are flawed in the sense that different methods may be appropriate for different input variables, and it is unrealistic to expect that one method will dominate all the rest under all circumstances. Accordingly, a second innovation of this paper is to introduce MCOS, a Monte Carlo approach that estimates the allocation error produced by various optimization methods on a particular set of input variables. The result is a precise determination of what method is most robust to a particular case. Thus, rather than relying always on one particular approach, MCOS allows users to apply opportunistically whatever optimization method is best suited in a particular setting.

Keywords: Monte Carlo, convex optimization, de-noising, clustering, shrinkage.

JEL Classification: G0, G1, G2, G15, G24, E44.
AMS Classification: 91G10, 91G60, 91G70, 62C, 60E.

1. MOTIVATION

Most industries face the need of making decisions based on incomplete or flawed information. A popular approach to making these decisions is to cast the answer as the solution to a convex optimization problem, where the goal is to maximize an objective function subject to a series of inequality constraints. For example, a financial firm may be interested in estimating the allocations to a variety of investments, such that the expected return is maximized subject to a target level of risk, or that the risk is minimized subject to a target level of expected return.

The critical line algorithm (CLA) is among the most common procedures used to solve convex optimization problems subject to inequality constraints (Markowitz [1952], Bailey and López de Prado [2013]). Although mathematically correct, CLA is known to be a poor estimator of the optimal solution out-of-sample (Michaud [1998], De Miguel et al. [2009], Guerard [2010]). The reason is, CLA exhibits extremely high estimation errors due to: (a) noisy inputs, and (b) signal structure that makes CLA unstable. While some literature has discussed noise-induced instability, signal-induced instability is often ignored or misunderstood.

Various analytical approaches have attempted to reduce CLA's estimation error, in three alternative ways (Fabozzi et al. [2007]): (i) introducing strong constraints (Clarke et al. [2002]); (ii) adding priors (Black and Litterman [1991]); and (iii) shrinking the covariance matrix (Ledoit and Wolf [2003]). Although these methods are helpful, they do not directly address the two underlying causes of CLA's instability. Approach (i) modifies the feasibility region in unpredictable ways, leading to unintended consequences. Approach (ii) attempts to add signal to help reduce the instability caused by noise, without dealing with the noise directly. That added signal is often arbitrary, in the form of beliefs. The overall effect can be an increase in the estimation error. Approach (iii) attempts to reduce the noise at the expense of weakening the signal. Shrinkage methods weaken the signal, because they do not discriminate between eigenvectors associated with noise and eigenvectors associated with signal.

This paper introduces two innovations. First, it proposes a novel solution to the problem of solving convex optimization problems, while controlling for instabilities caused by input noise and, critically, complex signal structure. This solution is implemented in the form of an algorithm, named nested clustered optimization, or NCO. Second, it applies Monte Carlo methods to evaluate the estimation error associated with a variety of optimization approaches (including NCO). With that knowledge, users can choose the optimization method that most robustly estimates a solution given a particular set of input variables. This Monte Carlo optimization selection (MCOS) method frees users from having to rely on a particular optimization method, allowing them to choose whatever method is best suited in the context of a particular problem.

2. PROBLEM STATEMENT

Without loss of generality, consider a system with N random variables, where the expected value of draws from these variables is represented by an array μ, and the variance of these draws is represented by the covariance matrix V. We would like to compute the solution vector ω that minimizes the variance of the system, measured as $\omega' V \omega$, subject to achieving a target $\omega' a$, where a characterizes the optimal solution. In its simplest form, the problem can be stated as

$$\min_{\omega} \frac{1}{2}\omega' V \omega$$
$$\text{s.t.: } \omega' a = 1$$

For example, in a financial application, the optimal allocation ω^* is known as the maximum Sharpe ratio portfolio when $a = \mu$ (i.e., it maximizes $\frac{\omega'\mu}{\sqrt{\omega' V \omega}}$, where 1_N is a vector of ones of size N). The optimal allocation ω^* is known as the minimum variance portfolio when $a = 1_N$ (i.e., it minimizes $\omega' V \omega$).

Applying convex optimization methods, it is easy to find that the optimal solution to the above problem is

$$\omega^* = \frac{V^{-1}a}{a' V^{-1} a}$$

We refer to this equation as the convex optimization solution (CVO). For the sake of clarity, the remainder of this paper will focus on the above problem statement. However, nothing in this discussion is intended to limit the applicability of NCO and MCOS to more general problems, including problems that incorporate a variety of inequality constraints, non-linear constraints, or penalty functions (e.g., functions that penalize deviations in ω^* from prior levels of ω^*).

3. NUMERICAL INSTABILITY

The input variables V and a are typically unknown. A common approach to estimating ω^* is to compute

$$\widehat{\omega}^* = \frac{\hat{V}^{-1}\hat{a}}{\hat{a}'\hat{V}^{-1}\hat{a}}$$

where $\hat{V}$ is the estimated V, and $\hat{a}$ is the estimated a. In general, replacing each variable with its estimate will lead to unstable solutions, that is, solutions where a small change in the inputs will cause extreme changes in $\widehat{\omega}^*$. This is problematic, because in many practical applications there are material costs associated with the re-allocation from one solution to another. We can trace back the instability of $\widehat{\omega}^*$ to two generic causes, as explained next.

3.1. INSTABILITY CAUSED BY NOISE

Consider a matrix of independent and identically distributed random observations X, of size TxN, where the underlying process generating the observations has zero mean and variance σ^2. The matrix $C = T^{-1}X'X$ has eigenvalues λ that asymptotically converge (as $N \to +\infty$ and $T \to +\infty$ with $1 < {}^{T}/_{N} < +\infty$) to the Marcenko-Pastur probability density function (PDF),

$$f[\lambda] = \begin{cases} \frac{T}{N}\frac{\sqrt{(\lambda_+ - \lambda)(\lambda - \lambda_-)}}{2\pi\lambda\sigma^2} & \text{if } \lambda \in [\lambda_-, \lambda_+] \\ 0 & \text{if } \lambda \notin [\lambda_-, \lambda_+] \end{cases}$$

where the maximum expected eigenvalue is $\lambda_+ = \sigma^2\left(1+\sqrt{\frac{N}{T}}\right)^2$, and the minimum expected eigenvalue is $\lambda_- = \sigma^2\left(1-\sqrt{\frac{N}{T}}\right)^2$. When $\sigma^2 = 1$, then C is the correlation matrix associated with X. This distribution is derived in Marcenko and Pastur [1967].

The Marcenko-Pastur distribution explains why empirical covariance matrices contain substantial amounts of noise. In many practical applications, $\frac{N}{T} \to 1$, thus the covariance's eigenvalues span a wide range $[\lambda_-, \lambda_+]$. Because $\lambda_- \to 0$ as $\frac{N}{T} \to 1$, the determinant of $\hat{V}$ approaches zero, and $\hat{V}^{-1}$ cannot be estimated robustly, leading to unstable solutions $\widehat{\omega}^*$. Potter et al. [2005] explains one possible way to de-noise covariance matrices, in order to obtain robust estimates of $\hat{V}^{-1}$. See López de Prado [2019] for details, examples and code.

3.2. INSTABILITY CAUSED BY SIGNAL

Certain covariance structures can make the optimal solution unstable. Without loss of generality, consider a correlation matrix between two variables,

$$C = \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}$$

where ρ is the correlation between their outcomes. Matrix C can be diagonalized as $CW = W\Lambda$, where

$$\Lambda = \begin{bmatrix} 1+\rho \\ 1-\rho \end{bmatrix}; W = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

The inverse of C is $C^{-1} = W\Lambda^{-1}W' = \frac{1}{|C|}\begin{bmatrix} 1 & -\rho \\ -\rho & 1 \end{bmatrix}$, where $|C|$ is the determinant of C, $|C| = \Lambda_{1,1}\Lambda_{2,2} = (1+\rho)(1-\rho) = 1-\rho^2$. From the above, we can see that the more ρ deviates from zero, the bigger one eigenvalue becomes relative to the other, causing the determinant of C to approach zero, which makes the values of C^{-1} explode.

There is an intuitive explanation for how signal makes $\widehat{\omega}^*$ unstable. When the correlation matrix is an identity matrix, the eigenvalue function is a horizontal line. Outside that ideal case, at least one subset of variables exhibits greater correlation among themselves than to the rest, forming a cluster within the correlation matrix. Clusters appear naturally, as a consequence of hierarchical relationships. When K variables form a cluster, they are more heavily exposed to a common eigenvector, which implies that the associated eigenvalue explains a greater amount of variance. But because the trace of the correlation matrix is exactly N, that means that an eigenvalue can only increase at the expense of the other $K-1$ eigenvalues in that cluster, resulting in a condition number greater than 1. Consequently, the greater the intra-cluster correlation is, the

higher the condition number becomes. This source of instability is distinct and unrelated to $\frac{N}{T} \to 1$ (noise).

4. MONTE CARLO ESTIMATION

This section explains the MCOS method for estimating $\widehat{\omega}^*$ while controlling for noise-induced and signal-induced instabilities. MCOS consists in five steps, as outlined below. The input variables are the array of expected outcomes (μ), and the covariance matrix of the expected outcomes (V). These variables may incorporate priors, following the Black-Litterman method or similar.

4.1. DRAWING EMPIRICAL MEANS AND COVARIANCES

MCOS treats the pair $\{\mu, V\}$ as true values, and draws a matrix of simulated observations X from the distribution characterized by $\{\mu, V\}$. The size of X is TxN, matching the number of variables (N) and sample length (T) used to compute the original $\{\mu, V\}$. From matrix X, MCOS derives the simulated pair $\{\hat{\mu}, \hat{V}\}$. Code snippet 1 implements function `simCovMu`, which executes this first step of MCOS. Boolean argument `shrink` applies Ledoit-Wolf shrinkage to X, if so desired.

```
import numpy as np,pandas as pd
from sklearn.covariance import LedoitWolf
#---------------------------------------------------------------------------
def simCovMu(mu0,cov0,nObs,shrink=False):
    x=np.random.multivariate_normal(mu0.flatten(),cov0,size=nObs)
    mu1=x.mean(axis=0).reshape(-1,1)
    if shrink:cov1=LedoitWolf().fit(x).covariance_
    else:cov1=np.cov(x,rowvar=0)
    return mu1,cov1
```

Snippet 1 – Drawing an empirical vector of means and an empirical covariance matrix

4.2. DE-NOISING

The second step is to de-noise the covariance matrix. The purpose of this step is to tackle the source of instability explained in section 3.1. Code snippet 2 implements function `findMaxEval`. This function uses a Kernel Density Estimate (KDE) algorithm to fit the Marcenko-Pastur distribution to the empirical distribution of eigenvalues. This has the effect of separating noise-related eigenvalues from signal-related eigenvalues.

```
def fitKDE(obs,bWidth=.25,kernel='gaussian',x=None):
    # Fit kernel to a series of obs, and derive the prob of obs
    # x is the array of values on which the fit KDE will be evaluated
    if len(obs.shape)==1:obs=obs.reshape(-1,1)
    kde=KernelDensity(kernel=kernel,bandwidth=bWidth).fit(obs)
    if x is None:x=np.unique(obs).reshape(-1,1)
    if len(x.shape)==1:x=x.reshape(-1,1)
    logProb=kde.score_samples(x) # log(density)
    pdf=pd.Series(np.exp(logProb),index=x.flatten())
    return pdf
#---------------------------------------------------------------------------
def mpPDF(var,q,pts):
    # Marcenko-Pastur pdf
```

```
    # q=T/N
    eMin,eMax=var*(1-(1./q)**.5)**2,var*(1+(1./q)**.5)**2
    eVal=np.linspace(eMin,eMax,pts)
    pdf=q/(2*np.pi*var*eVal)*((eMax-eVal)*(eVal-eMin))**.5
    pdf=pd.Series(pdf,index=eVal)
    return pdf
#-------------------------------------------------------------------------------
def errPDFs(var,eVal,q,bWidth,pts=1000):
    # Fit error
    pdf0=mpPDF(var,q,pts) # theoretical pdf
    pdf1=fitKDE(eVal,bWidth,x=pdf0.index.values) # empirical pdf
    sse=np.sum((pdf1-pdf0)**2)
    return sse
#-------------------------------------------------------------------------------
def findMaxEval(eVal,q,bWidth):
    # Find max random eVal by fitting Marcenko's dist to the empirical one
    out=minimize(lambda *x:errPDFs(*x),.5,args=(eVal,q,bWidth),
                 bounds=((1E-5,1-1E-5),))
    if out['success']:var=out['x'][0]
    else:var=1
    eMax=var*(1+(1./q)**.5)**2
    return eMax,var
```

Snippet 2 – Upper bound for the eigenvalues associated with noise

Code snippet 3 implements function `deNoiseCov`. This function computes the correlation matrix associated with a given covariance matrix, and derives the eigenvalues and eigenvectors for that correlation matrix. Function `denoisedCorr` shrinks the eigenvalues associated with noise, and returns a de-noised correlation matrix. Function `corr2cov` recovers the covariance matrix from the de-noise correlation matrix. In summary, this step shrinks only the eigenvalues associated with noise, leaving the eigenvalues associated with signal unchanged.

```
from sklearn.neighbors.kde import KernelDensity
from scipy.optimize import minimize
#-------------------------------------------------------------------------------
def corr2cov(corr,std):
    cov=corr*np.outer(std,std)
    return cov
#-------------------------------------------------------------------------------
def cov2corr(cov):
    # Derive the correlation matrix from a covariance matrix
    std=np.sqrt(np.diag(cov))
    corr=cov/np.outer(std,std)
    corr[corr<-1],corr[corr>1]=-1,1 # numerical error
    return corr
#-------------------------------------------------------------------------------
def getPCA(matrix):
    # Get eVal,eVec from a Hermitian matrix
    eVal,eVec=np.linalg.eigh(matrix)
    indices=eVal.argsort()[::-1] # arguments for sorting eVal desc
    eVal,eVec=eVal[indices],eVec[:,indices]
    eVal=np.diagflat(eVal)
    return eVal,eVec
#-------------------------------------------------------------------------------
def denoisedCorr(eVal,eVec,nFacts):
```

```
    # Remove noise from corr by fixing random eigenvalues
    eVal_=np.diag(eVal).copy()
    eVal_[nFacts:]=eVal_[nFacts:].sum()/float(eVal_.shape[0]-nFacts)
    eVal_=np.diag(eVal_)
    corr1=np.dot(eVec,eVal_).dot(eVec.T)
    corr1=cov2corr(corr1)
    return corr1
#-------------------------------------------------------------------------------
def deNoiseCov(cov0,q,bWidth):
    corr0=cov2corr(cov0)
    eVal0,eVec0=getPCA(corr0)
    eMax0,var0=findMaxEval(np.diag(eVal0),q,bWidth)
    nFacts0=eVal0.shape[0]-np.diag(eVal0)[::-1].searchsorted(eMax0)
    corr1=denoisedCorr(eVal0,eVec0,nFacts0)
    cov1=corr2cov(corr1,np.diag(cov0)**.5)
    return cov1
```

Snippet 3 – De-noising the empirical covariance matrix

The left panel in Exhibit 1 illustrates how MCOS separates the eigenvalues associated with noise (enveloped by the Marcenko-Pastur distribution) from the eigenvalues associated with signal in a correlation matrix. The right panel in Exhibit 1 shows that MCOS shrinks only the eigenvalues associated with noise, leaving the eigenvalues associated with signal unaltered. This is in contrast with shrinkage methods (e.g., Ledoit-Wolf method), which do not discriminate between the two kinds of eigenvalues, hence shrinking the covariance matrix at the expense of diluting part of the signal.

[EXHIBIT 1 HERE]

4.3. OPTIMAL ALLOCATIONS

The third step is to estimate $\widehat{\omega}^*$ from $\{\hat{\mu}, \hat{V}\}$ according to various alternative methods. MCOS is agnostic with regards to the specific methods applied. Possible methods include CVO, and the nested clustered optimization (NCO) algorithm, which is introduced next.

The NCO method estimates $\widehat{\omega}^*$ while controlling for the signal-induced estimation errors explained in section 3.2. NCO works as follows. First, we cluster the covariance matrix into subsets of highly-correlated variables. One possible clustering algorithm is the partitioning method discussed in López de Prado and Lewis [2019], but hierarchical methods may also be applied. The result is a partition of the original set, that is, a collection of mutually disjoint non-empty subsets of variables. Second, we compute optimal allocations for each of these clusters separately. This allows us to collapse the original covariance matrix into a reduced covariance matrix, where each cluster is represented as a single variable. The collapsed correlation matrix is closer to an identity matrix than the original correlation matrix was, and therefore more amenable to optimization problems (recall the discussion in section 3.2). Third, we compute the optimal allocations across the reduced covariance matrix. Fourth, the final allocations are the dot-product of the intra-cluster allocations (step 2) and the inter-cluster allocations (step 3). By splitting the problem into two separate tasks, NCO contains the instability within each cluster: the instability caused by intra-cluster noise does not propagate across clusters. See López de Prado [2019] for examples, code and additional details regarding NCO.

Code snippet 4 implements the NCO algorithm. Function `optPort_nco` invokes function `clusterKMeansBase` to find the optimal partition of clusters (see López de Prado and Lewis [2019] for details), and then invokes function `optPort` on each cluster. Function `optPort` can apply the optimization method preferred by the user, and `optPort_nco` is agnostic as to what that method does. For illustration purposes, without loss of generality, in this particular implementation, `optPort` estimates the solution outlined in section 2. When argument `mu` is `None`, function `optPort_nco` returns the minimum variance portfolio, whereas when `mu` is not `None`, function `optPort_nco` returns the maximum Sharpe ratio portfolio.

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples
#-------------------------------------------------------------------------------
def clusterKMeansBase(corr0,maxNumClusters=None,n_init=10):
    dist,silh=((1-corr0.fillna(0))/2.)**.5,pd.Series() # distance matrix
    if maxNumClusters is None:maxNumClusters=corr0.shape[0]/2
    for init in range(n_init):
        for i in xrange(2,maxNumClusters+1): # find optimal num clusters
            kmeans_=KMeans(n_clusters=i,n_jobs=1,n_init=1)
            kmeans_=kmeans_.fit(dist)
            silh_=silhouette_samples(dist,kmeans_.labels_)
            stat=(silh_.mean()/silh_.std(),silh.mean()/silh.std())
            if np.isnan(stat[1]) or stat[0]>stat[1]:
                silh,kmeans=silh_,kmeans_
    newIdx=np.argsort(kmeans.labels_)
    corr1=corr0.iloc[newIdx] # reorder rows
    corr1=corr1.iloc[:,newIdx] # reorder columns
    clstrs={i:corr0.columns[np.where(kmeans.labels_==i)[0]].tolist() for \
            i in np.unique(kmeans.labels_)} # cluster members
    silh=pd.Series(silh,index=dist.index)
    return corr1,clstrs,silh
#-------------------------------------------------------------------------------
def optPort(cov,mu=None):
    inv=np.linalg.inv(cov)
    ones=np.ones(shape=(inv.shape[0],1))
    if mu is None:mu=ones
    w=np.dot(inv,mu)
    w/=np.dot(ones.T,w)
    return w
#-------------------------------------------------------------------------------
def optPort_nco(cov,mu=None,maxNumClusters=None):
    cov=pd.DataFrame(cov)
    if mu is not None:mu=pd.Series(mu[:,0])
    corr1=cov2corr(cov)
    corr1,clstrs,_=clusterKMeansBase(corr1,maxNumClusters,n_init=10)
    wIntra=pd.DataFrame(0,index=cov.index,columns=clstrs.keys())
    for i in clstrs:
        cov_=cov.loc[clstrs[i],clstrs[i]].values
        mu_=(None if mu is None else mu.loc[clstrs[i]].values.reshape(-1,1))
        wIntra.loc[clstrs[i],i]=optPort(cov_,mu_).flatten()
    cov_=wIntra.T.dot(np.dot(cov,wIntra)) # reduce covariance matrix
    mu_=(None if mu is None else wIntra.T.dot(mu))
    wInter=pd.Series(optPort(cov_,mu_).flatten(),index=cov_.index)
    nco=wIntra.mul(wInter,axis=1).sum(axis=1).values.reshape(-1,1)
```

```
    return nco
```

Snippet 4 – Implementation of the NCO algorithm

A common question is whether we should cluster `corr0` or `corr0.abs()`. When all correlations are non-negative, clustering `corr0` and `corr0.abs()` yields the same outcome. When some correlations are negative, this technology tries both clusterings, and determines which type of clustering works best for a particular `corr0` in Monte Carlo experiments.

Exhibit 2 illustrates how a correlation matrix of 500 variables is partitioned into 10 highly-correlated clusters, each of size 50. In this stylized example, NCO first computes the optimal allocation of each cluster separately, and collapses the 500x500 matrix into a 10x10 matrix. Then NCO computes the optimal allocations for the 10x10 matrix. The final weights are the dot-product of the intra-cluster and inter-cluster weights.

[EXHIBIT 2 HERE]

4.4. MONTE CARLO SIMULATIONS

The fourth step is to combine all previous steps into a Monte Carlo experiment, whereby optimal allocations $\widehat{\omega}^*$ are computed on a large number of simulated pairs $\{\hat{\mu}, \hat{V}\}$. Code snippet 5 implements this step, by invoking all the functions explained earlier. The arguments are:

- `mu0`: the original vector of expected outcomes.
- `cov0`: the original covariance matrix of outcomes.
- `nObs`: the number of observations T used to compute `mu0` and `cov0`.
- `nSims`: the number of simulations run in the Monte Carlo.
- `bWidth`: the bandwidth of the KDE functions used to de-noise the covariance matrix.
- `minVarPortf`: when True, the minimum variance solution is computed. Otherwise, the maximum Sharpe ratio solution is computed.
- `shrink`: when True, the covariance matrix is subjected to the Ledoit-Wolf shrinkage procedure.

```
def monteCarlo(mu0,cov0,nObs,nSims,bWidth,minVarPortf,shrink):
    w1=pd.DataFrame(columns=xrange(cov0.shape[0]),
                            index=xrange(nSims),dtype=float)
    w1_d=w1.copy(deep=True)
    for i in range(nSims):
        mu1,cov1=simCovMu(mu0,cov0,nObs,shrink)
        if minVarPortf:mu1=None
        if bWidth>0:cov1=deNoiseCov(cov1,nObs*1./cov1.shape[1],bWidth)
        w1.loc[i]=optPort(cov1,mu1).flatten()
        w1_d.loc[i]=optPort_nco(cov1,mu1,int(cov1.shape[0]/2)).flatten()
    return
```

Snippet 5 – Monte Carlo simulations

Function `monteCarlo` repeats `nSims` times the following sequence: (1) given a pair $\{\mu, V\}$, `simCovMu` draws `nObs` observations, and uses those observations to compute $\{\hat{\mu}, \hat{V}\}$ (with shrinkage, if `shrink=True`); (2) if `minVarPortf`, then drop $\hat{\mu}$, because we do not need it to

estimate the minimum variance portfolio; (3) if `bWidth>0`, then `deNoiseCov` de-noises $\hat{V}$; (4) use $\{\hat{\mu}, \hat{V}\}$ to compute alternative solutions. In this particular implementation, without loss of generality, code snippet 5 estimates the solution outlined in section 2 as well as the NCO solution, however MCOS may consider many more alternative solutions.

4.5. ESTIMATION ERRORS

So far, MCOS has estimated allocations $\widehat{\omega}^*$ from simulated pairs $\{\hat{\mu}, \hat{V}\}$. In the fifth step, MCOS computes the true optimal allocation ω^* from the pair $\{\mu, V\}$, and compares that result with the estimated $\widehat{\omega}^*$. For each variable in the system, MCOS computes the standard deviation of the differences between $\widehat{\omega}^*$ and ω^*. The mean of the standard deviations across all variables gives us the estimation error associated with a particular optimization method. In alternative implementations, MCOS may evaluate the estimation error in terms of the average decay in performance, measured as: (i) the mean difference in expected outcomes, $(\omega^* - \widehat{\omega}^*)'\mu$; (ii) the mean difference in variance, $(\omega^* - \widehat{\omega}^*)'V(\omega^* - \widehat{\omega}^*)$; (iii) the mean difference in Sharpe ratio, $\frac{(\omega^* - \widehat{\omega}^*)'\mu}{\sqrt{(\omega^* - \widehat{\omega}^*)'V(\omega^* - \widehat{\omega}^*)}}$; or (iv) ratios computed on any of the above statistics.

In this last step, MCOS identifies what optimization method yields the most robust allocations for a given pair $\{\mu, V\}$. Code snippet 6 presents one possible implementation of this fifth step.

```
w0=optPort(cov0,None if minVarPortf else mu0)
w0=np.repeat(w0.T,w1.shape[0],axis=0) # true allocation
err=(w1-w0).std(axis=0).mean()
err_d=(w1_d-w0).std(axis=0).mean()
```

Snippet 6 – Estimation errors

5. A NUMERICAL EXAMPLE

In this section, we subject the NCO algorithm to controlled experiments, and compare its performance to the CVO approach in the context of investment portfolio optimization. We discuss two characteristic portfolios of the efficient frontier, namely the minimum variance and maximum Sharpe ratio solutions, since any member of the unconstrained efficient frontier can be derived as a convex combination of the two (a result sometimes known as the "separation theorem").

Code snippet 7 creates a random vector of means and a random covariance matrix that represent a stylized version of a 50 securities portfolio, grouped in 10 blocks with intra-cluster correlations of 0.5. This vector and matrix characterize the "true" process that generates observations, $\{\mu, V\}$. We set a seed for the purpose of reproducing results across runs with different parameters. In practice, the pair $\{\mu, V\}$ does not need to be simulated, and MCOS receives $\{\mu, V\}$ as an input.

```
def formTrueMatrix(nBlocks,bSize,bCorr,std0=None):
    corr0=formBlockMatrix(nBlocks,bSize,bCorr)
    corr0=pd.DataFrame(corr0)
    cols=corr0.columns.tolist()
    np.random.shuffle(cols)
    corr0=corr0[cols].loc[cols].copy(deep=True)
    if std0 is None:std0=np.random.uniform(.05,.2,corr0.shape[0])
    else:std0=np.array([std0]*corr0.shape[1])
```

```
    cov0=corr2cov(corr0,std0)
    mu0=np.random.normal(std0,std0,cov0.shape[0]).reshape(-1,1)
    return mu0,cov0
#-------------------------------------------------------------------------------
def formBlockMatrix(nBlocks,bSize,bCorr):
    block=np.ones((bSize,bSize))*bCorr
    block[range(bSize),range(bSize)]=1
    corr=block_diag(*([block]*nBlocks))
    return corr
#-------------------------------------------------------------------------------
from scipy.linalg import block_diag
nBlocks,bSize,bCorr =10,5,.5
np.random.seed(0)
mu0,cov0=formTrueMatrix(nBlocks,bSize,bCorr)
```

Snippet 7 – Data generating process

5.1. MINIMUM VARIANCE PORTFOLIO

We use function `simCovMu` to simulate a random empirical vector of means and a random empirical covariance matrix based on 1,000 observations drawn from the true process. When `shrink=True`, the empirical covariance matrix is subjected to Ledoit-Wolf shrinkage. Using that empirical covariance matrix, function `optPort` estimates the minimum variance portfolio according to CVO, and function `optPort_nco` estimates the minimum variance portfolio applying the NCO algorithm. This procedure is repeated on 1,000 different random empirical covariance matrices. Note that, because `minVarPortf=True`, the random empirical vectors of means are discarded. We can run the Monte Carlo with and without shrinkage. Exhibit 3 reports the estimation errors, following the definition stated in section 4.5.

[EXHIBIT 3 HERE]

NCO computes the minimum variance portfolio with 44.95% of CVO's estimation error, i.e. a 55.05% reduction in the estimation error. While Ledoit-Wolf shrinkage helps reduce the estimation error, that reduction is relatively small, around 13.91%. Combining shrinkage and NCO, it is possible to achieve a 61.30% reduction in the estimation error of the minimum variance portfolio. The combination of de-noising with NCO yields a 63.63% reduction in estimation error.

Note that the combination of CVO with de-noising delivers a similar estimation error as NCO, however the combination of NCO with de-noising beats the combination of CVO with de-noising, thus the benefits of NCO and de-noising do not perfectly overlap. The triad of NCO with shrinkage and de-noising achieves the greatest reduction in the estimation error, in the order of 68.00%.

The implication is that NCO delivers substantially lower estimation error than CVO's solution, even for a small portfolio of only 50 securities, particularly when we combine NCO with de-noising and (to a lesser extent) with shrinkage. It is easy to test that NCO's advantage widens for larger portfolios.

5.2. MAXIMUM SHARPE RATIO PORTFOLIO

By setting `minVarPortf=False`, we can re-run the Monte Carlo to derive the estimation error associated with the maximum Sharpe ratio portfolio. Exhibit 4 reports the results from this experiment.

[EXHIBIT 4 HERE]

NCO computes the maximum Sharpe ratio portfolio with 42.20% of CVO's estimation error, i.e. a 57.80% reduction in the estimation error. Combining shrinkage and NCO, it is possible to achieve a 74.34% reduction in the estimation error of the maximum Sharpe ratio portfolio. The triad of NCO with shrinkage and de-noising achieves the greatest reduction in estimation error, at 76.31%.

6. CONCLUSIONS

This paper has introduced two innovations to tackle the instability of convex optimization solutions. First, the NCO algorithm splits the optimization problem into two tasks. One task aims at the containment of the instability caused by clusters of highly-correlated variables, and the other task finds the optimal allocation inter-clusters. Second, the MCOS method derives the estimation errors associated with a variety of optimization algorithms on a set of input variables. The result is a precise determination of what algorithm is most robust to a particular case. Thus, rather than relying always on one particular algorithm, MCOS allows users to apply opportunistically the optimization algorithm that is best suited to their particular problem.

These two innovations have practical applications in a variety of settings. As evidenced in section 5, a clear candidate is the optimization of investment portfolios, with substantial reductions in the estimation error, as well as substantial reductions in the transaction costs associated with the rebalancing of unstable portfolios.

REFERENCES

Bailey, D. and M. López de Prado (2013): "An open-source implementation of the critical-line algorithm for portfolio optimization." *Algorithms*, 6(1), pp. 169-196. http://www.mdpi.com/1999-4893/6/1

Black F. and R. Litterman (1991): "Asset Allocation Combining Investor Views with Market Equilibrium." *Journal of Fixed Income*, 1(2), pp. 7-18.

Clarke, R., H. De Silva, and S. Thorley (2002): "Portfolio constraints and the fundamental law of active management." *Financial Analysts Journal*, Vol. 58, pp. 48–66.

De Miguel, V., L. Garlappi, and R. Uppal (2009): "Optimal versus naïve diversification: how inefficient is the 1/N portfolio strategy?" *Review of Financial Studies*, 22(5), pp. 1915-1953.

Fabozzi, F., P. Kolm, D. Pachamanova, and S. Focardi (2007): *Robust Portfolio Optimization and Management*. Wiley Finance, First Edition.

Guerard, J. (2010): *Handbook of portfolio construction*. Springer, First edition.

Ledoit, O. and M. Wolf (2003): "A Well-Conditioned Estimator for Large-Dimensional Covariance Matrices." *Journal of Multivariate Analysis*, 88(2), pp. 365-411.

López de Prado, M. (2016): "Building diversified portfolios that outperform out-of-sample." *The Journal of Portfolio Management*, 42(4), pp. 59-69. https://jpm.iijournals.com/content/42/4/59

López de Prado, M. (2018): *Advances in financial machine learning*. Wiley, First edition.

López de Prado, M. (2019): *Machine Learning for Asset Managers*. Cambridge University Press, First edition (forthcoming).

López de Prado, M. and M. Lewis (2019): "Detection of false investment strategies using unsupervised learning methods." *Quantitative Finance*, Vol. 19, No. 9, pp.1555-1565.

Marčenko, V. and L. Pastur (1967): "Distribution of eigenvalues for some sets of random matrices." *Matematicheskii Sbornik*, Vol. 72, No. 4, pp. 507-536.

Markowitz, H. M. (1952): "Portfolio selection." *Journal of Finance*, 7(1), pp. 77–91.

Michaud, R. (1998): *Efficient asset allocation: a practical guide to stock portfolio optimization and asset allocation*. Boston, MA: Harvard Business School Press.

Potter, M., J.P. Bouchaud, L. Laloux (2005): "Financial applications of random matrix theory: Old laces and new pieces." *Acta Physica Polonica B*, Vol. 36, No. 9, pp. 2767-2784.

EXHIBITS

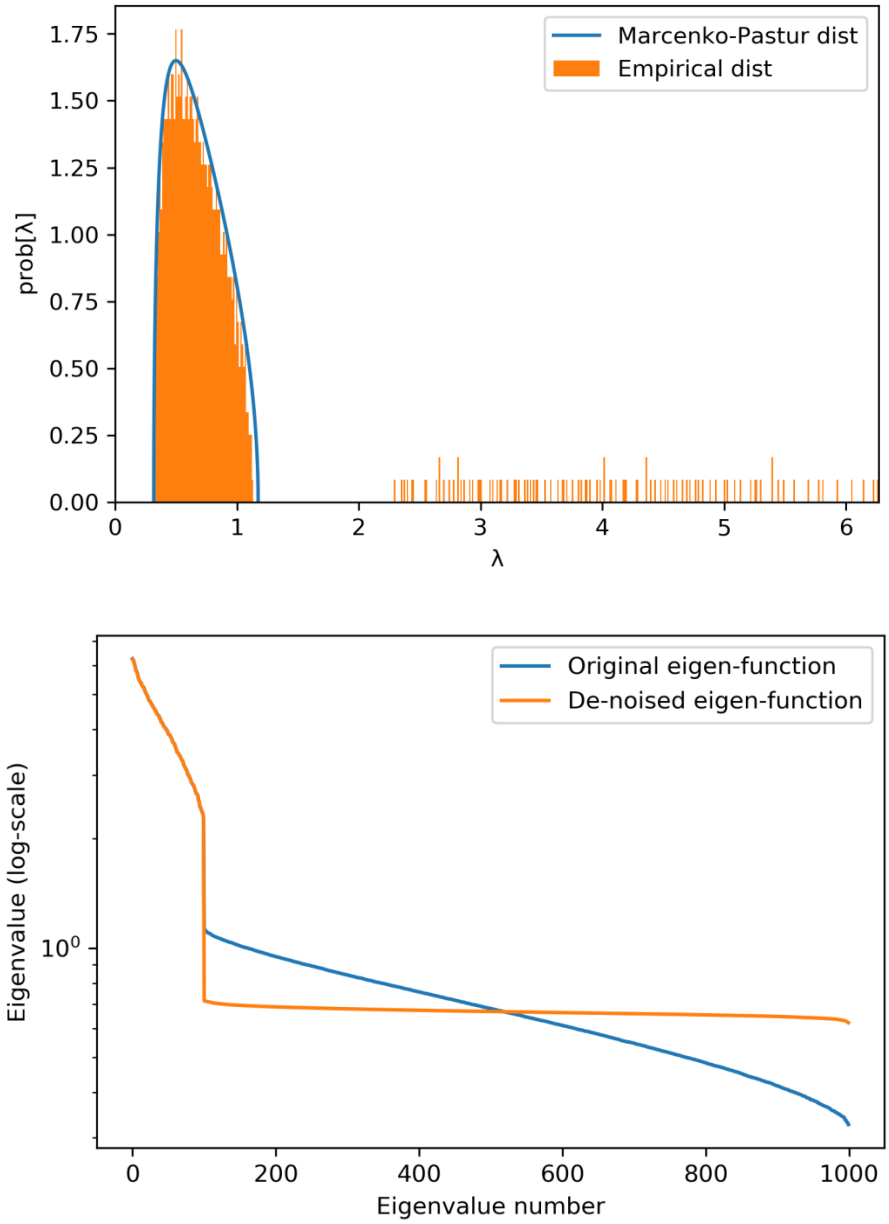

Exhibit 1 – De-noising of the correlation matrix

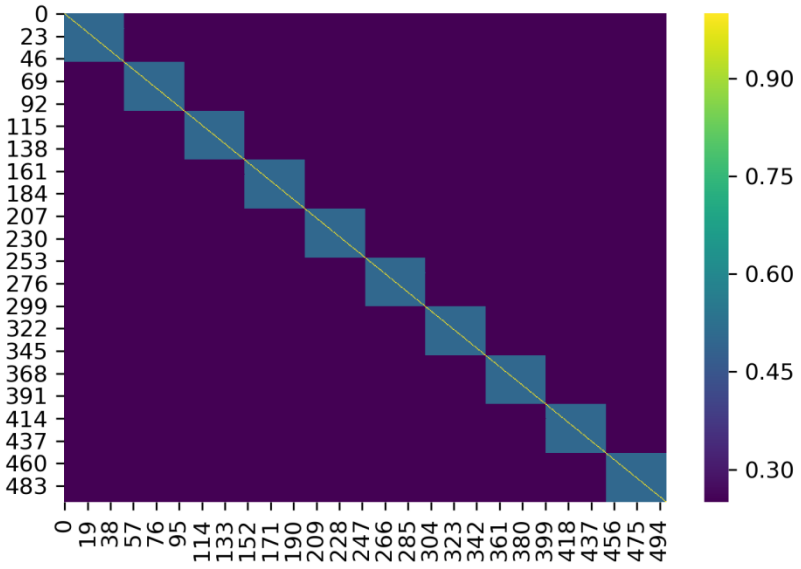

Exhibit 2 – Clustered correlation matrix

	CVO	NCO
Raw	7.27E-03	3.27E-03
Shrunk (S)	6.26E-03	2.82E-03
De-noised (D)	3.20E-03	2.65E-03
S+D	3.01E-03	2.33E-03

Exhibit 3 – Estimation errors for the minimum variance portfolio

	Markowitz	NCO
Raw	6.34E-02	2.67E-02
Shrunk (S)	3.50E-02	1.63E-02
De-noised (D)	3.52E-02	2.64E-02
S+D	2.04E-02	1.50E-02

Exhibit 4 – Estimation errors for the maximum Sharpe ratio portfolio