mvo-vs-naive-pc-loading

April 8, 2022

AUTHOR: SPARSHSAH

1 Loading of MVO vs naive portfolio on primary principal component

1.1 Definitions

1.1.1 The market

There are N assets, each at unit vol. The correlation matrix is Ω. The eigenvalues, from largest to smallest, are $\lambda_0, \ldots, \lambda_{N-1}$ with corresponding eigenvectors $e_0, \ldots, e_{N-1}$, each at unit vol. The ER's of each asset are the vector $\mu := (\mu_0, \ldots, \mu_{N-1})$.

1.1.2 The portfolios

The naive portfolio $u := \mu$. The MVO portfolio $v = \Omega^{-1}\mu$. In fact, let us scale these to unit vol, so that

$$u = \frac{\mu}{\sqrt{\mu'\Omega\mu}},$$

and

$$v = \frac{\Omega^{-1}\mu}{\sqrt{(\Omega^{-1}\mu)'\Omega(\Omega^{-1}\mu)}} = \frac{\Omega^{-1}\mu}{\sqrt{\mu'\Omega^{-1}\Omega\Omega^{-1}\mu}} = \frac{\Omega^{-1}\mu}{\sqrt{\mu'\Omega^{-1}\mu}}.$$

Notice that these are all “risk weights”.

1.1.3 Loadings

Given that all views are at unit vol, we define loading as (risk-model) correlation, which is equivalent to (risk-model) covariance. We have loading of u on e_0 is

$$\ell_u := u'\Omega e_0 = \frac{1}{\sqrt{\mu'\Omega\mu}}\mu'\Omega e_0,$$

and loading of v on e_0 is

$$\ell_v := v'\Omega e_0 = \frac{1}{\sqrt{\mu'\Omega^{-1}\mu}}\mu'\Omega^{-1}\Omega e_0 = \frac{1}{\sqrt{\mu'\Omega^{-1}\mu}}\mu' e_0.$$

This is non-standard notation, but because it doesn't matter where I put the scalar, let me suggestively write:

$$\ell_u = \mu'\frac{\Omega}{\sqrt{\mu'\Omega\mu}}e_0,$$

$$\ell_v = \mu' \frac{1}{\sqrt{\mu'\Omega^{-1}\mu}} e_0.$$

1.2 The game

It's easy to construct an example where $\ell_v = \ell_u$: Just make the assets i.i.d., so that $\Omega = I_N$. But, is it possible that $|\ell_v| > |\ell_u|$? I worked on this with a bunch of smart people and we came up empty-handed, so let's just cop out by generating pseudorandom combinations of correlation matrices and ER vectors, and check whether it's ever the case that $|\ell_v| > |\ell_u|$.

1.3 Simulations

Spoiler: Simulations suggest that this will never happen.

1.3.1 Utility functions

A lot of these are just copy-pasted (shhh!) from here.

[1]:
```python
from typing import Tuple, Optional
import numpy.random as random
import pandas as pd
import numpy as np

CorrelMatrix = pd.DataFrame
ErVector = pd.Series
Portfolio = pd.Series  # risk-weight vector
PrincipalComponent = Portfolio

N = 2  # default number of assets, simple
MAX_NUM_ASSETS = 10
NUM_TRIALS = 1_000


# data generation

def maybe(val, otherwise):
    return otherwise if val is None else val

def _get_asset_name(n: int=0) -> str:
    return f"X{n}"

def _get_eigen_name(n: int=0) -> str:
    return f"E{n}"

def gen_Omega(k: Optional[int]=None, dim: int=N) -> CorrelMatrix:
    """Generate $dim \times dim$ pseudorandom correlation matrix,
    with some strong pairwise correlations if $k \in (0, dim)$.
    [source](https://stats.stackexchange.com/a/125017).
    """
```

```
    k = maybe(k, otherwise=int(dim/2))
    # \in \mathbb{R}^{k \times dim}
    factor_loadings = pd.DataFrame(random.randn(k, dim))
    # \in \mathbb{R}^{dim \times dim}, and symmetric
    Omega = factor_loadings.T @ factor_loadings
    # \in [0,1)^{dim \times dim}
    perturbation = pd.DataFrame(np.diag(random.rand(dim)))
    # make Omega nonnegative-definite
    Omega = Omega + perturbation
    # \in \mathbb{R}^{dim \times dim}
    normalizer = np.diag( np.diag(Omega)**-0.5 )
    Omega = normalizer @ Omega @ normalizer
    # checks
    assert Omega.abs().max().max() <= 1 + 1e-6, Omega.abs().max().max()
    # raises LinAlgException if not nonnegative-definite else passes
    _ = np.linalg.cholesky(Omega)
    Omega = Omega.rename(index=_get_asset_name, columns=_get_asset_name)
    return Omega

def gen_mu(dim: int=N) -> ErVector:
    """Generate ER's."""
    mu = pd.Series(random.randn(dim))
    mu = mu.rename(index=_get_asset_name)
    return mu


# calculations

def get_max_abs_nondiag(mat: pd.DataFrame) -> float:
    mat_diag = pd.DataFrame(np.diag(np.diag(mat)))
    # zero out diagonals
    mat = mat - mat_diag
    return mat.abs().max().max()

def inv(mat: pd.DataFrame) -> pd.DataFrame:
    ix, cols = mat.index, mat.columns
    mat = np.linalg.inv(mat)
    mat = pd.DataFrame(mat, index=ix, columns=cols)
    return mat

def eig(mat: pd.DataFrame) -> Tuple[pd.Series, pd.DataFrame]:
    """Eigendecompose `mat`,
    returning eigenvalues `W` and corresponding eigenvectors `V`,
    such that the eigenvector associated with eigenvalue `W[n]` is `V[:, n]`.

    Random variables are indexed as f"X{n}",
    Eigenv's are indexed as f"E{n}".
```

```python
    E.g. Get first eigenvector as `V.loc[:, "E0"]`.
    """
    W, V = np.linalg.eig(mat)
    W = pd.Series(W)
    V = pd.DataFrame(V, index=mat.index)
    # sort in order of explained variance, then reorder v to match
    W = W.sort_values(ascending=False)
    V = V.reindex(columns=W.index)
    # the order it came out of `np.eig` is not meaningful, drop it
    W = W.reset_index(drop=True)
    # stupid hack, there is no `pd.DataFrame.reset_columns()`
    V = V.T.reset_index(drop=True).T
    # make the column names more suggestive
    W = W.rename(index=_get_eigen_name)
    V = V.rename(columns=_get_eigen_name)
    # i hate vec's with negative heads, so if i find one, negate the entire vec
    sign_of_V_heads = np.sign(V.loc[_get_asset_name(0), :])
    V = V.mul(sign_of_V_heads, axis="columns")
    return W, V

def get_nth_pc(mat: pd.DataFrame, n: int=0) -> pd.Series:
    W, V = eig(mat=mat)
    pc = V.loc[:, _get_eigen_name(0)]
    return pc

def get_exante_covar(Omega: CorrelMatrix, a: Portfolio, b: Portfolio) -> float:
    """Get covariance of given portfolios assuming given asset risk models."""
    return a.T @ Omega @ b

def get_exante_vol(Omega: CorrelMatrix, pflio: Portfolio) -> float:
    """Get volatility of given portfolio assuming given asset risk model."""
    return get_exante_covar(Omega=Omega, a=pflio, b=pflio)**0.5

def _get_loading(Omega: CorrelMatrix, of: Portfolio, on: Portfolio) -> float:
    """Get loading of pflio `of` on pflio `on`."""
    return get_exante_covar(Omega=Omega, a=of, b=on)

def get_loading(
        Omega: CorrelMatrix, of_pflio: Portfolio, on_pc_num: int=0
    ) -> float:
    """Get loading of `pflio` on `on_pc`th PC."""
    pc = get_nth_pc(mat=Omega, n=on_pc_num)
    return _get_loading(Omega=Omega, of=of_pflio, on=pc)


# portfolio calculations
```

```python
def get_pflio(Omega: CorrelMatrix, mu: ErVector, mvo: bool=False) -> Portfolio:
    pflio = inv(Omega) @ mu if mvo else mu
    # normalize
    norm = get_exante_vol(Omega=Omega, pflio=pflio)
    pflio = pflio / norm
    assert np.isclose(get_exante_vol(Omega=Omega, pflio=pflio), 1), \
        get_exante_vol(Omega=Omega, pflio=pflio)
    return pflio

def get_er(mu: ErVector, pflio: Portfolio) -> float:
    return pflio @ mu

def get_sharpe(Omega: CorrelMatrix, mu: ErVector, pflio: Portfolio) -> float:
    """Sharpe ratio."""
    er = get_er(mu=mu, pflio=pflio)
    vol = get_exante_vol(Omega=Omega, pflio=pflio)
    sr = er / vol
    return sr


# analyze

def _abs_loading_of_a_exceeds_abs_loading_of_b(
        Omega: CorrelMatrix,
        a: Portfolio, b: Portfolio,
        on_pc_num: int=0
    ) -> bool:
    beta_a = get_loading(Omega=Omega, of_pflio=a, on_pc_num=on_pc_num)
    beta_b = get_loading(Omega=Omega, of_pflio=b, on_pc_num=on_pc_num)
    return abs(beta_a) > abs(beta_b)

def abs_mvo_loading_exceeds_abs_naive_loading(
        Omega: CorrelMatrix, mu: ErVector
    ) -> bool:
    naive_pflio = get_pflio(Omega=Omega, mu=mu)
    mvo_pflio = get_pflio(Omega=Omega, mu=mu, mvo=True)
    return _abs_loading_of_a_exceeds_abs_loading_of_b(
        Omega=Omega, a=mvo_pflio, b=naive_pflio
    )

def __run(num_assets: int=N, raise_if_exceeds=False) -> bool:
    """Run single trial -> Whether MVO exceeded naive (abs) loading."""
    Omega = gen_Omega(dim=num_assets)
    mu = gen_mu(dim=num_assets)
    flag = abs_mvo_loading_exceeds_abs_naive_loading(Omega=Omega, mu=mu)
    if raise_if_exceeds:
        assert not flag, \
```

```python
            f"\nOmega:\n{Omega}\n...\nmu:\n{mu}"
    return flag

def _run(num_assets: int=N, num_trials: int=NUM_TRIALS) -> float:
    """Run many trials -> Fraction where MVO exceeded (abs) naive loading."""
    res = [__run(num_assets=num_assets) for _ in range(num_trials)]
    return np.mean(res)

def run(max_num_assets=MAX_NUM_ASSETS) -> pd.Series:
    """Get fraction of trials where MVO exceeded (abs) naive loading,
    across a domain of asset-universe sizes.
    """
    num_assets_domain = range(2, MAX_NUM_ASSETS)
    res = pd.Series({num_assets:
        _run(num_assets=num_assets)
    for num_assets in num_assets_domain})
    return res
```

1.3.2 Analysis

[3]:
```python
random.seed(1337)
# number of assets -> fraction of trials where mvo exceeded naive (abs) loading
res = run()
# see that the % is always 0
any(res)
```

[3]: False