The frontier of efficient portfolios

Ken Nyholm, April 20, 2024

Note: the first code-cell in the section "Solving the optimisation problem numerically" needs to be executed manually, after the notebook has been run for the first time. On the first execution this cell produces an error, probably because the JuMP module (used for numerical optimisation) is not yet completely integrated into the Pluto notebook environment. But, just execute this part of the code manually a second time, and it will work (press the "run" symbol at the end of the cell, or place the cursor in the cell and press "Shift+Enter"). JuMP is an excellent optimisation package and it is a small price to pay to have to execute the module twice!

The goal of this workbook is to use the Julia programming language to compute the efficient portfolio frontier, a concept deeply rooted in financial theory. This concept was first introduced by **Markowitz in 1952**, who demonstrated the calculation under the assumption of either a quadratic utility function for an individual's wealth assessment or, equivalently, the multivariate normal distribution of financial returns. Although these assumptions may seem stringent, the principles Markowitz developed continue to serve as a fundamental cornerstone in financial theory.

A portfolio is said to be *efficient* if it is the least risky (i.e. it has the lowest variance) among all portfolios that have identical levels of expeced return or, similarly, if it has the highest expected return among all portfolios that have identical levels of risk. Mathematically this can be expressed as:

$$\begin{aligned} & \min_{w} \ \frac{1}{2} w^\top \cdot V \cdot w \\ \text{s.t.} \quad & \\ & w^\top \cdot r = E[r_p] \\ & w^\top \cdot \iota = 1, \end{aligned}$$

where w is a vector of portfolio weights, V is the variance-covariance matrix of the eligible asset universe, r is the vector of expected returns for each asset, and ι is a vector of 1's.
In words, this states that we want to find the asset weights w that minimises half the portfolio variance $\sigma_p^2 = w^\top \cdot V \cdot w$, subject to a given level of portfolio return, and under the requirement of full investment, i.e. that $\sum_{j=1}^{J} w_{\{j\}} = 1$, where j refers to holdings of individual instruments in the portfolio, for example, $w_{\{j=3\}} = 0.10$ mean that 10% of the investor's money is placed in asset number 3.

An equivalet formulation of the problem maximises the portfolio return, subject to a given risk level and full investment.

It is of course inconsequential whether the objective function in the minimisation problem above is multiplied by a scalar or not. The use of $\frac{1}{2}$ in the objective function is a practical convention that simplifies the calculation. When we solve the problem analytically, the first derivative of the matrix expression and set it equal to zero, and we solve for w. This is similar to how we would solve a problem in regular calculus.

Differentiating the expression: $w^\top \cdot V \cdot w$, with respect to w, gives: $2 \cdot V \cdot w$. Multiplying the objective function by $\frac{1}{2}$ at the start effectively cancels out the 2 that appears from the differentiation, simplifying the expression.

Example data

I have used Google Sheets to download equity data from Google Finance for a set of international equity market indices, and then I have copied these data to an Excel workbook to store the data locally on my computer. The workbook is named 'ExampleEquityData.xlsx' and it holds weekly prices covering the period from January 1999 to January 2024. For our example it is not really relevant which indices or data we use, but having the data in Excel gives us the opportunity to see how to transfer data from Excel to Julia. And, this is interesting in itself.
To this end the 'XLSX' and 'DataFrames' packages are needed to read data and to handle the data.

```
begin
    using XLSX
    using DataFrames
end
```

	Dates	SP500	DowJones	Nasdaq	DAX	UKX	CAC	Stoxx	
1	1999-01-01	1229.23	9181.43	2192.69	5006.57	5882.6	3942.66	3342.32	1
2	1999-01-08	1275.09	9643.32	2344.41	5006.57	5882.6	3942.66	3342.32	1
3	1999-01-15	1212.19	9643.32	2344.41	5370.51	6147.2	4245.42	3669.67	1
4	1999-01-22	1243.26	9340.55	2348.2	4973.78	5941	4054.81	3306.42	1
5	1999-01-29	1243.26	9120.67	2338.88	5008.21	5861.2	4019.33	3473.31	1
6	1999-02-05	1279.64	9358.83	2505.89	5180.29	5896	4251.8	3547.15	1
7	1999-02-12	1239.4	9304.24	2373.62	5097.48	5855.3	4147.3	3489.98	1
8	1999-02-19	1230.13	9274.89	2321.89	4896.74	5950.7	4060.36	3405.68	1
9	1999-02-26	1239.19	9339.95	2283.6	4823.26	6031.2	4130.48	3415.52	1
10	1999-03-05	1238.33	9306.58	2288.03	4903.96	6175.1	4092.94	3484.24	1
	more								
1307	2024-01-12	4697.24	37466.1	14524.1	16594.2	7689.61	7420.69	4463.51	3

```
begin
    dtable = XLSX.readtable("ExampleEquityData.xlsx","Sheet1")
    dfPrices = DataFrame(dtable)
end
```

Prices P_t observed at time t for each of the the equity indices are now stored in the DataFrame named *dfPrices*, and percentage annualised returns are then calculated by:
$r_t = 100\% \cdot log\left(\frac{P_{t+1}}{P_t}\right)$.

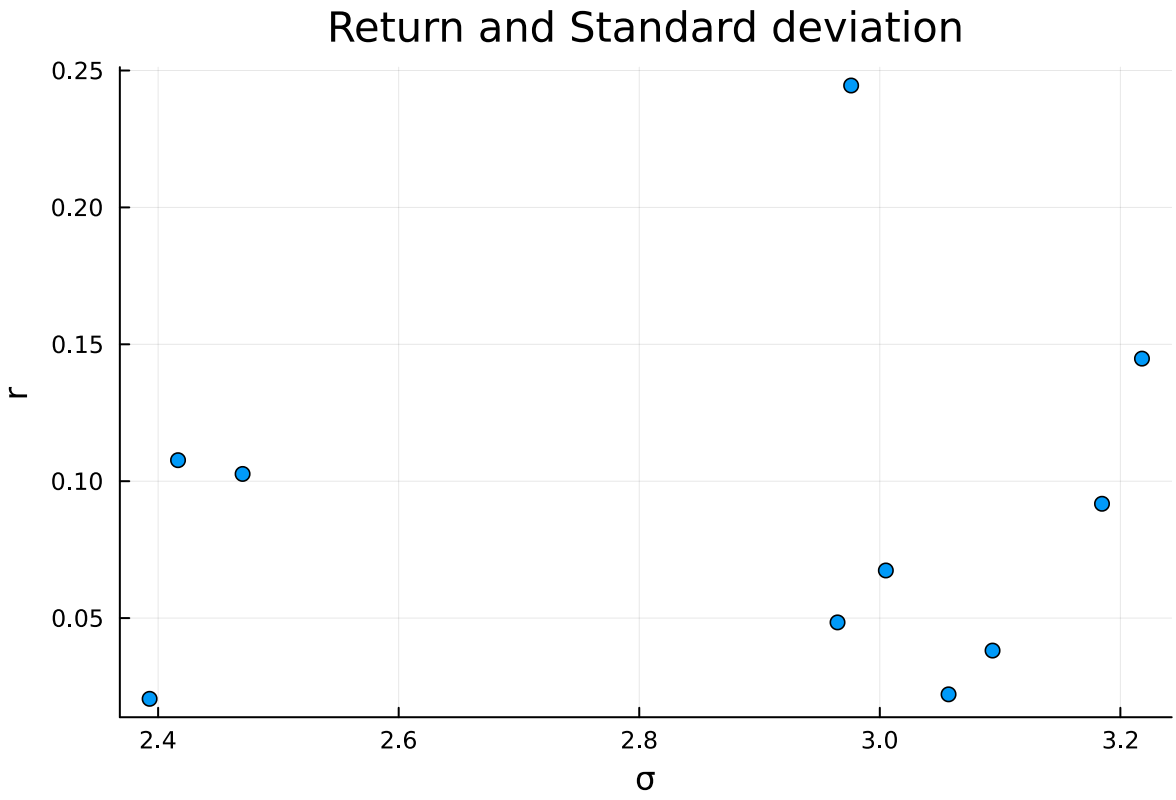
```
1306×10 Matrix{Float64}:
  3.66288    4.90825      6.69047    0.0       …  -3.30764    6.49317  10.5636
 -5.05881    0.0          0.0        7.01718        2.5585     0.0       0.0
  2.53083   -3.19003      0.161531  -7.67428        0.0       -5.51455  -5.79579
  0.0       -2.38219     -0.397689   0.689845       2.97972   -4.11284   2.40581
  2.88418    2.5777       6.89718    3.37825        2.40714   -2.40713   0.951522
 -3.19514   -0.585007    -5.42277   -1.61147   …  -4.23462   -3.38802  -3.14349
 -0.750754  -0.315946    -2.20347   -4.01766        0.542557   2.52726   3.56178
  ⋮                                                ⋱
  0.771035   2.38817      0.379397   2.27       …  -0.578673  -4.24097   2.36241
  0.211762   0.00653853   0.689264   2.18185       -3.41883   -2.99094   3.40259
  2.46313    2.8751       2.80634   -0.046433       2.03042    2.76424   2.29704
  0.748171   0.216385     1.20141   -0.270551       0.600247  -2.72727  -0.501098
  0.319178   0.80871      0.122516   0.271745       0.88581    4.23559   1.77346
 -1.53356   -0.594581    -3.29993   -0.944232  …  -0.259569  -3.04978  -0.0948387
```

```
begin
    using LinearAlgebra
    prices = Matrix(dfPrices)
    returns=100.0.*log.(prices[2:end,2:end]./prices[1:end-1,2:end])
end
```

To calculate the efficient frontier the covariance matrix V and the vector of expected asset returns r are needed, as shown above. These inputs are derived directly from the downloaded example data, since I here focus only on the underlying mechanics of efficient frontier calculations.

When applying these methodologies in real-world scenarios, it's crucial to ensure that the expected returns and covariances align with the investor's future market expectations for the duration of the investment horizon. Creating models to predict returns is a complex task in itself, and is beyond the scope of this discussion. Here, our focus is solely on understanding the underlying mechanics of efficient frontier calculations. It's important to remember that while historical data can provide a useful starting point, it may not always accurately reflect future market conditions. Therefore, in practice, the use of return prediction models or other methods to estimate future returns and covariances may be necessary to ensure the relevance of the efficient frontier to the investor's specific situation.

To illustrate the historical relation ship between return and standard deviation each of the equity indices' return is plotted against its standard deviation, i.e. the square root of the diagonal elements of V.




```
begin
    using Statistics
    using Plots
    r = mean(returns, dims=1)'
    V = cov(returns)
    scatter(sqrt.(diag(V)), r,
            title = "Return and Standard deviation",
            xlabel = "σ", ylabel = "r", label = "")
end
```

Solving the optimisation problem analytically

It is possible to find the overall minimum-variance portfolio analytically, as long as no constraints are imposed on the portfolio weights w_p. The standard way to do this is by setting up the Lagrange function, by substituting the constraints (multiplied by a scalar constant) into the objective function:

$$\min \quad L_{\{w_p,\lambda_p,\gamma_p\}} = \frac{1}{2} w_p^\top \cdot V \cdot w_p + \lambda_p \left(r_p - w_p^\top \cdot r\right) + \gamma_p \left(1 - w_p^\top \cdot \iota\right)$$

Next, first derivatives are calculated and set equal to zero:

$$\begin{aligned}
\frac{\partial L}{\partial w_p} &= V \cdot w_p - \lambda_p \cdot r - \gamma_p \cdot \iota = 0 \\
\frac{\partial L}{\partial \lambda_p} &= r_p - w_p^\top \cdot r = 0 \\
\frac{\partial L}{\partial \gamma_p} &= 1 - w_p^\top \cdot \iota = 0
\end{aligned}$$

and, solving these three equations for the three unknowns proceedes in the following way: the first of the above three equations can be rewritten as:

$$w_p = V^{-1} \cdot \left(\lambda_p \cdot r + \gamma_p \cdot \iota\right)$$

which substituted into the two last derivatives from above, helps us find the vaues/expressions for λ_p and γ_p:

$$\begin{aligned}
r_p &= \left(\lambda_p \cdot V^{-1} \cdot r + \gamma_p \cdot V^{-1} \cdot \iota\right)^\top \cdot r \\
&= r^\top \cdot \left(\lambda_p \cdot V^{-1} \cdot r + \gamma_p \cdot V^{-1} \cdot \iota\right) \\
&= \lambda_p \cdot \underbrace{\left(r^\top \cdot V^{-1} \cdot r\right)}_{\text{known quantities}} + \gamma_p \cdot \underbrace{\left(r^\top \cdot V^{-1} \cdot \iota\right)}_{\text{known quantities}}
\end{aligned}$$

$$\begin{aligned}
1 &= \left(\lambda_p \cdot V^{-1} \cdot r + \gamma_p \cdot V^{-1} \cdot \iota\right)^\top \cdot \iota \\
&= \iota^\top \cdot \left(\lambda_p \cdot V^{-1} \cdot r + \gamma_p \cdot V^{-1} \cdot \iota\right) \\
&= \lambda_p \cdot \underbrace{\left(\iota^\top \cdot V^{-1} \cdot r\right)}_{\text{known quantities}} + \gamma_p \cdot \underbrace{\left(\iota^\top \cdot V^{-1} \cdot \iota\right)}_{\text{known quantities}}
\end{aligned}$$

Collecting the expressions in a compact format, noting that the matrix multiplication of the known quantities are scalars, the above can be written as:

$$\begin{bmatrix} r_p \\ 1 \end{bmatrix} = \begin{bmatrix} X & Y \\ Y & Z \end{bmatrix} \begin{bmatrix} \lambda_p \\ \gamma_p \end{bmatrix}$$

where **Cramer's Rule** provides the solution, with:

$$\begin{aligned} X &= r^\top \cdot V^{-1} \cdot r \\ Y &= r^\top \cdot V^{-1} \cdot \iota = \iota^\top \cdot V^{-1} \cdot r \\ Z &= \iota^\top \cdot V^{-1} \cdot \iota \\ D &= X \cdot Z - Y^2 \end{aligned}$$

$$\lambda_p = \frac{\det\left(\begin{bmatrix} r_p & Y \\ 1 & Z \end{bmatrix}\right)}{\det\left(\begin{bmatrix} X & Y \\ Y & Z \end{bmatrix}\right)} = \frac{Z \cdot r_p - Y}{D}$$

$$\gamma_p = \frac{\det\left(\begin{bmatrix} X & r_p \\ Y & 1 \end{bmatrix}\right)}{\det\left(\begin{bmatrix} X & Y \\ Y & Z \end{bmatrix}\right)} = \frac{X - Y \cdot r_p}{D}$$

To complete the calculations, these expressions are substitited into the expression for the portfolio weights, $w_p = V^{-1} \cdot (\lambda_p \cdot r + \gamma_p \cdot \iota)$, giving:

$$\begin{aligned} w_p &= \frac{1}{D} \cdot V^{-1} \cdot [(Z \cdot r_p - Y) \cdot r + (X - Y \cdot r_p) \cdot \iota] \\ &= \frac{1}{D} \cdot V^{-1} \cdot [(X \cdot \iota - Y \cdot r) + (Z \cdot r - Y \cdot \iota) \cdot r_p] \\ &= g + h \cdot r_p \end{aligned}$$

where:

$$\begin{aligned} g &= \frac{1}{D} \cdot V^{-1} \cdot (X \cdot \iota - Y \cdot r) \\ h &= \frac{1}{D} \cdot V^{-1} \cdot (Z \cdot r - Y \cdot \iota) \end{aligned}$$

Unconstrained asset weights for efficient frontier portfolios are found as a linear/affine function of the required portfolio return. The whole frontier can then be calculated by varying the required portfolio return across all feasible values and finding the corresponding asset weights.

It is instructive to find the weights of the overall minimum variance/ minimum risk portfolio. The portfolio frontier (the whole frontier, not only the efficient part) traces out an upward-opening hyperbola that is turned 90 degrees clock-wise. We can therefore find the coordinates $\{r_{min}, \sigma^2_{min}\}$ from the quadratic expression for σ_p^2 as a fundtion of r_p (note that given the hyperbola is turned 90 dgrees clock-wise, r_p is the x-variable in this context):

$$\begin{aligned}
\sigma_p^2 &= w_p^\top \cdot V \cdot w_p \\
&= w_p^\top \cdot V \cdot \left(\lambda_p \cdot V^{-1} \cdot r + \gamma_p \cdot V^{-1} \cdot \iota\right) \\
&= \lambda_p \cdot w_p^\top \cdot r + \gamma_p \cdot w_p^\top \cdot \iota \\
&= \lambda_p \cdot r_p + \gamma_p \\
&= \frac{Z \cdot r_p - Y}{D} \cdot r_p + \frac{X - Y \cdot r_p}{D} \\
&= \frac{1}{D}\left(Z \cdot r_p^2 - 2 \cdot Y \cdot r_p + X\right)
\end{aligned}$$

The coordinates of the minimum point is found using the standard formula for the minimum point of a quadratic equation: $f(x) = a \cdot x^2 + b \cdot x + c$, which is:$\{x_{min}, y_{\min}\} = \left(-\frac{b}{2a}, f\left(-\frac{b}{2a}\right)\right)$:

$$r_{mvp} = \frac{2 \cdot Y \cdot D^{-1}}{2 \cdot Z \cdot D^{-1}} = \frac{Y}{Z}$$

And, to find corresponding minimum variance portfolio weights, this expression is inserted into the general expression for the frontier weights:

$$
\begin{aligned}
w_{mvp} &= g + h \cdot r_{mvp} \\
&= g + h \cdot \frac{Y}{Z} \\
&= V^{-1}\frac{1}{XZ - Y^2}\left[(X\iota - Yr) + (Zr - Y\iota)\frac{Y}{Z}\right] \\
&= V^{-1}\frac{1}{(XZ - Y^2)}\left(\frac{XZ\iota - rYZ + Y(rZ - Y\iota)}{Z}\right) \\
&= V^{-1}\frac{XZ\iota - rYZ + \left(rYZ - Y^2\iota\right)}{Z(XZ - Y^2)} \\
&= V^{-1}\frac{\left(XZ\iota - Y^2\iota + (rYZ - rYZ)\right)}{Z(XZ - Y^2)} \\
&= V^{-1}\frac{1}{(XZ - Y^2)}\left(\frac{XZ\iota - Y^2\iota}{Z}\right) \\
&= V^{-1}\frac{\iota\left(XZ - Y^2\right)}{Z(XZ - Y^2)} \\
&= V^{-1}\frac{\iota}{Z} \\
&= \frac{V^{-1}\iota}{\iota^\top \cdot V^{-1} \cdot \iota}
\end{aligned}
$$

The weights of the minimum variance portfolio are determined solely on the basis of the covariance between the asset in the investible universe, and that expected returns play no role here. We see that the weights are the row sums of the inverse covariance matrix scaled by the sum of all the elements of the inverse of the covariance matrix.

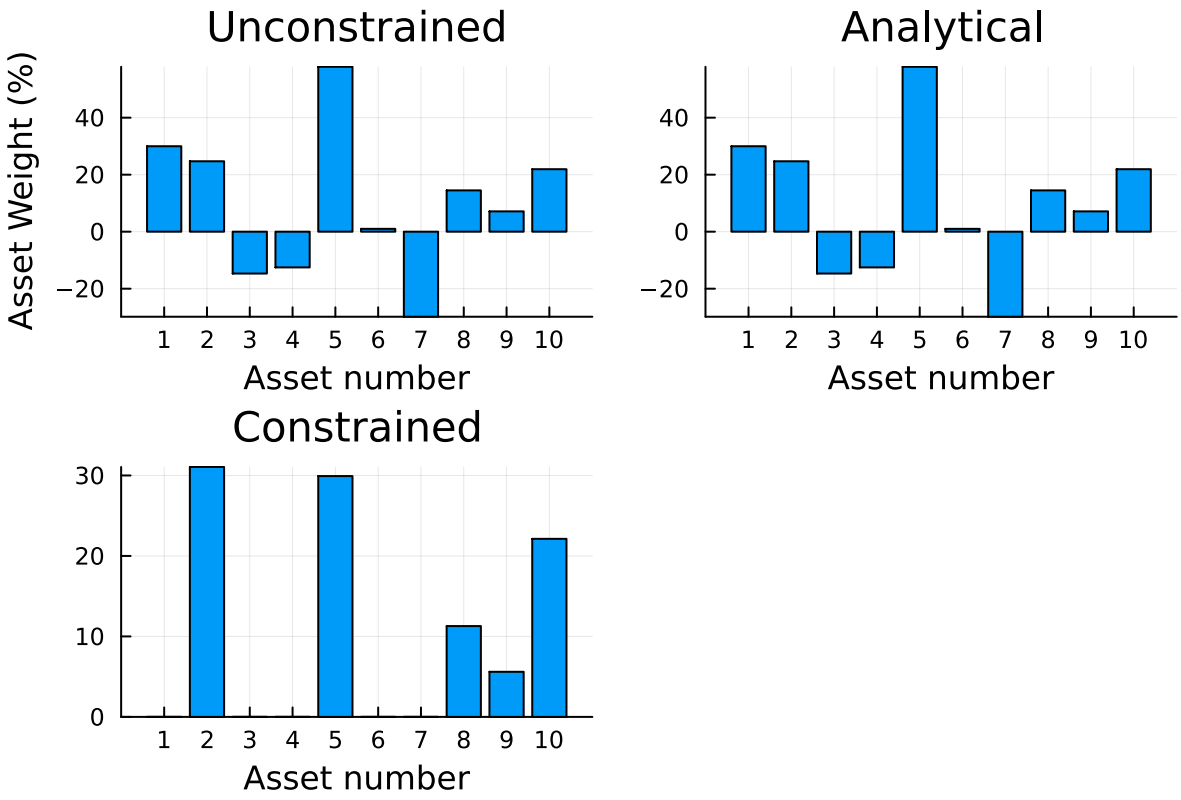
This result is valid only for the unconstrained solution to the portfolio problem. Once constraints are imposed, for example, once only non-negative weights are allowed, the situation changes and nummerical optimisation methods are needed.

Solving the optimisation problem numerically

Using the ingredients prepared above, the overall minimum variance portfolio can be found. It is done using numerical methods and imposing the constraint that all portfolio weights must be non-negative, that is: $w_{\{j\}} \geq 0,\ \forall j$. Below the constrained and unconstrained solutions are obtained numerically, as well as the analytical solution for the unconstrained minimum variance portfolio, using the derived expression.

```
begin
    using JuMP
    using Ipopt

    nAssets = size(V,1)            # number of assets in the investment universe
    σ2p(w) = w' * V * w
    rp(w) = w' * r
    model = Model(Ipopt.Optimizer)
    set_silent(model)
    @variable(model, w[1:nAssets])
    @constraint(model, sum(w) == 1)                    # full investment
    @expression(model, variance, σ2p(w))               # portfolio variance
    @expression(model, expret, rp(w))                  # portfolio return
    @objective(model, Min, variance)                   # objective function

    optimize!(model)
    w_mvp_unc = value.(w)

    @constraint(model, [j=1:nAssets], 0 <= w[j] <= 1)  # weights between 0 and 1
    optimize!(model)
    w_mvp = value.(w)

    tmp = V^(-1)*ones(nAssets,1) ./ ones(nAssets,1)'*V^(-1)*ones(nAssets,1)

    w_mvp_analytical = tmp./sum(tmp)   # analytical solution normalised to sum
to 1

    println(solution_summary(model))
    println("Minimum stdev = ", sqrt(σ2p(w_mvp)))
    println("ExpRet = ", rp(w_mvp))
    println("w_mvp_unc(%) = ", round.(w_mvp_unc.*100; digits=2))
    println("w_mvp_Analytical(%) = ", round.(w_mvp_analytical.*100; digits=2))
    println("w_mvp(%) = ", round.(w_mvp.*100; digits=2))
end
```

As it should be, the analytical and unconstrained numerical solutions produce identical weights for the minimum variance portfolio, while the numerical solution to the solution where asset weights are constrained to be non-negative produce a solution that fulfills these constraints.




```
begin
    x = [1:1:10;]
    plot(bar(x, [w_mvp_unc.*100], xticks=1:10,
           xlabel="Asset number", ylabel="Asset Weight (%)",
           title="Unconstrained", legend = false), bar(x,
[w_mvp_analytical.*100],
           xticks=1:10, xlabel="Asset number", title="Analytical", legend =
false),
           bar(x, [w_mvp.*100], xticks=1:10, xlabel="Asset number",
           title="Constrained", legend = false) )
end
```

Finding frontier portfolios

We can now write a general function that determines the whole efficient frontier. A flexible way of specifying linear asset weight constraints is adopted, such that both individual and group constraints can be accommodated:

$$l \leq C \cdot w_p \leq u$$

where, l is the lower bound, u is the upper bound, and C is a matrix of dimension number of constraints by number of assets, that express the imposed constraints. For example,

- let n signify the number of elements in the respective vectors, with n being the number assets: $C = diag(\iota_n)$, $l = 0_n$, and $u = \iota_n$, impose the constraint that all individual weights are between (or equal to) 0 and 1
- when $C = \iota_n'$, $l = 1$, and $u = 1$ it is insured that the sum of the weights equal 1.

And ν specifies the number of portfolios that are calculated on the efficient frontier.

The function then takes the following inputs:

- expected returns: r
- covariance matrix: V
- lower weight constraints: l
- upper weight constraints: u
- definition of the imposed linear constraints: C
- number of efficient frontier portfolios to be calculated: ν

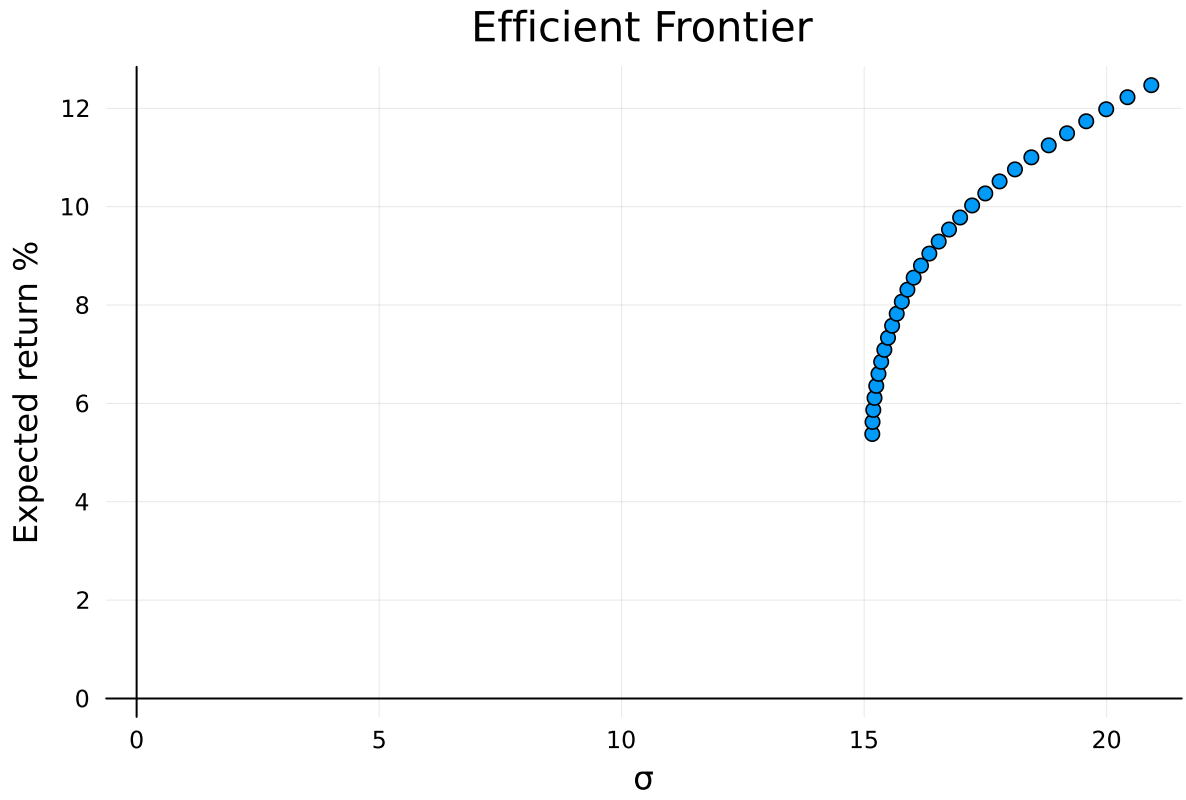
It produces the folloing output:

- vector of dimension ν by 1 of returns for frontier portfolios: r_p
- vector of dimension ν by 1 of standard deviations for frontier portfolios: σ_p

```
PortfolioFrontier (generic function with 1 method)
```

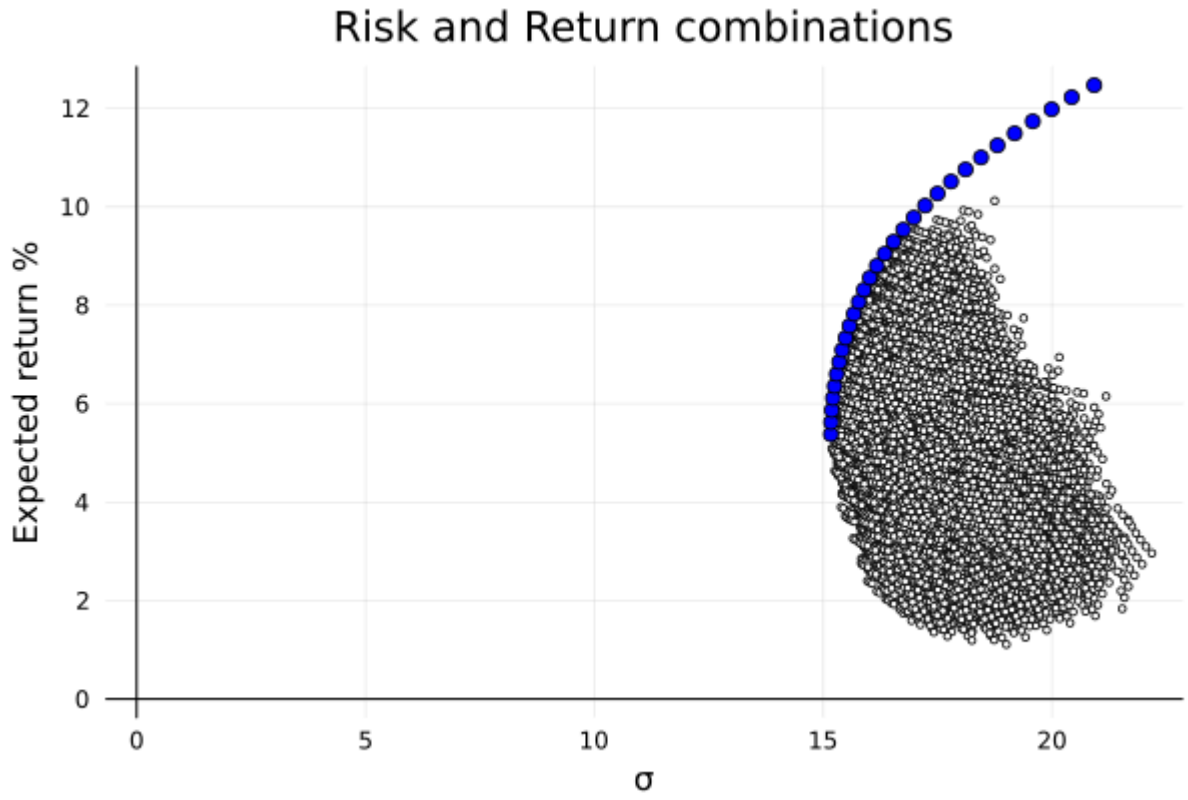
```
function PortfolioFrontier(r,V,l,u,C,ν)

    nAssets = size(V,1)          # number of assets in the investment universe
    w = 1/nAssets
    σ2p(w) = w'*V*w*52           # portfolio variance, annualised with 52 weeks

    rp(w) = dot(w', r)*52        # portfolio return, annualised with 52 weeks


    optimizer = optimizer_with_attributes(Ipopt.Optimizer, "tol" => 1e-100,
"max_iter" => 10000)
    model = Model(Ipopt.Optimizer; add_bridges = false)
    model = Model(optimizer)

    set_silent(model)

    # step 1: finding the minimum variance portfolio
    @variable(model, w[1:nAssets;])
    @constraint(model, l .<= C*w .<= u)               # imposing constraints

    @expression(model, variance, σ2p(w))              # portfolio variance
    @expression(model, expret, rp(w))                 # portfolio return
    @objective(model, Min, variance)                  # objective function
    optimize!(model)
    w_mvp = value.(w)

    # step 2: finding the maximum return portfolio
    @objective(model, Max, expret)                  # objective function
    optimize!(model)
    w_max = value.(w)

    # step 3: find portfolios on the efficient frontier between min and max
return
    r_target = collect(range(start=rp(w_mvp), length=ν+1, stop=rp(w_max)))

    # Initialise the output containers
    Wp = zeros(nAssets, ν)   # portfolio weights
    Rp = zeros(ν, 1)         # portfolio returns
    Sp = zeros(ν, 1)         # portfolio standard deviations

    # Loop over each return requirement
    for j in 1:ν
        @constraint(model, r_target[j,1] <= expret )
        @objective(model, Min, variance)                  # objective function
        optimize!(model)
        w_tmp = value.(w)
        Wp[:, j] = w_tmp
        Rp[j, 1] = rp(w_tmp)
        Sp[j, 1] = sqrt(σ2p(w_tmp))
    end

    return Rp, Sp, Wp
end
```

We use the example data from above to test the PortfolioFrontier function.




```
begin
    nPort = 30
    C1 = Matrix(I, nAssets, nAssets)    # lower and upper bounds on the weights
    C2 = ones(1, nAssets)               # sum(w)==1
    CC = [C1;C2]
    l1 = zeros(nAssets,1)
    l2 = [1]
    ll = [l1;l2]                        # lower bound for the constraints
    uu = ones(nAssets+1,1)              # upper bound for the constraints

    Rp, Sp, Wp = PortfolioFrontier(r,V,ll,uu,CC,nPort)

    plot(Sp, Rp, title="Efficient Frontier", xlabel="σ", ylabel="Expected return
%", seriestype=:scatter, marker=:circle, legend=false, framestyle=:zerolines)
end
```

It can be an illustrative exercise to calculate all feasible portfolio compositions and to plot them together with the efficient allocations derived above. To achieve this, all asset compositions must be explored and the ones where the weights sum to 1 are collected as a feasible composition. For each of these feasible portfolios the expected return and standard deviation is then plotted. Finally, the efficient frontier from above is superimposed on the plot. This is done below using the example data. We set the precision of the individual asset weights to 10%, i.e. asset weights are changed in 10% increments, and the minimum allow weight is set to 0% while the maximum weight is set to 50%. These constraints are chosen to limit the calculation time.




```
begin
    Rp_all=[]
    Sp_all=[]
    rnge = 0.0:0.1:0.5
    for w1 in rnge, w2 in rnge, w3 in rnge, w4 in rnge, w5 in rnge, w6 in rnge, w7 in rnge, w8 in rnge, w9 in rnge, w10 in rnge
        w_ = [w1,w2,w3,w4,w5,w6,w7,w8,w9,w10]
        if sum(w_)==1
            push!(Rp_all, dot(w_', r)*52 )        # portfolio return,
            push!(Sp_all, sqrt(w_'*V*w_*52))      # portfolio variance
        end
    end

    plot(Sp_all, Rp_all, title="Risk and Return combinations", xlabel="σ", ylabel="Expected return %", seriestype=:scatter, marker=:circle, legend=false, framestyle=:zerolines, markercolor=:white, markersize=2 )
    plot!(Sp, Rp, seriestype=:scatter, markershape=:circle, legend=false, framestyle=:zerolines, markercolor=:blue )
end
```