Basics

of return calculations: returns, average returns and volatilities of portfolios.

Load Packages and Extra Functions

The notebook uses the functions `printmat()` and `printlnPs()` for formatted printing of matrices and numbers. These functions are in the included `src/printmat.jl` file and call on the `Printf` package.

Also, the `lag()` function (from `src/lag.jl`) lags a vector/matrix.

```
using Printf

include("src/printmat.jl")
include("src/lag.jl");       #; to suppress printing of last command in cell
```

Return Calculations

The return of holding the asset between $t-1$ and t is

$R_t = (P_t + D_t)/P_{t-1} - 1,$

where P_t is the price (measured after dividends) and D_t is the dividend.

We can calculate the returns by a loop or by a more compact notation, see below.

A Remark on the Code

For the vectorized version, notice that - `lag(P)` creates a vector `[NaN,P[1],...,P[end-1]`, that is, the previous (lagged) value of `P`. - use `[a,b]./[c,d]` to do element-by-element division. Also, use `[a,b] .- 1` to subtract 1 from each element. In contrast, `[a,b]-[c,d]` (or `+`) needs no dot.

```
P = [100,108,109]                       #prices (after dividends) for t=1,2,3
D = [0,2,0]                             #dividends, could also use [0;2;0]

R = zeros(length(P))                    #where to store the results
for t in 2:length(P)                     #P[2] is the 2nd element of P
    R[t] = (P[t] + D[t])/P[t-1] - 1
end
R = R[2:end]                            #get rid of R[1] since we have no return there

R_alt = (P + D)./lag(P) .- 1            #vectorized alternative, notice the ./ and .- 1

printmat(R*100,R_alt[2:end]*100,colNames=["return, %","return (alt), %"],rowNames=2:3,cell00="
```

```
period      return, %return (alt), %
2              10.000          10.000
3               0.926           0.926
```

Cumulating Returns

Net returns can be cumulated to calculate portfolio values as

$V_t = V_{t-1}(1 + R_t)$

where we need a starting value (initial investment) for the portfolio (a common choice is to normalise to $V_0 = 1$).

With log returns, $r_t = \log(1 + R_t)$, we instead do

$\ln V_t = \ln V_{t-1} + r_t$

If the return series is an excess return, add the riskfree rate to convert it to get net returns - and then cumulate as described above.

A Remark on the Code

- Use `cumprod([a,b]` to calculate `[a,a*b]` and `cumsum([a,b]` to calculate `[a,a+b]`.
- To add 1 to each element of an array `R`, do `1 .+ R` (Notice the dot and the space before the dot.)
- To calculate the logarithm of each value in a matrix `X`, do `log.(X)` Again, notice the dot. In general, a function that is defined for a scalar can be called like that to do the calculation for each element in an array (vector, matrix,...).

```
R   = [20,-35,25]/100                 #returns for t=1,2,3
V   = cumprod(1 .+ R)                 #V(t) = V(t-1)*(1+R(t)), starting at 1 in t=0
r   = log.(1 .+ R)                    #log returns
lnV = cumsum(r)                       #lnV(t) = lnV(t-1) + r(t)

printmat(R,V,lnV,colNames=["R","V","lnV"],rowNames=1:3,cell00="period")

printred("Check that lnV really equals log.(V). Also, try a loop instead")
```

```
period         R         V       lnV
1          0.200     1.200     0.182
2         -0.350     0.780    -0.248
3          0.250     0.975    -0.025

Check that lnV really equals log.(V). Also, try a loop instead
```

Portfolio Return

We form a portfolio by combining n assets: v is the vector of n portfolio weights. The portfolio return is

$R_v = v'R,$

where R is a vector of returns of the n assets.

```
v = [0.8,0.2]
R = [10,5]/100               #returns of asset 1 and 2
Rᵥ = v'R                     #R\_v[TAB] to get Rᵥ

printblue("Portfolio weights:")
printmat(v;rowNames=["asset 1","asset 2"])
```

```
printblue("Returns:")
printmat(R;rowNames=["asset 1","asset 2"])

printblue("Portfolio return: ")
printlnPs(Rᵥ)
```

```
Portfolio weights:
asset 1     0.800
asset 2     0.200

Returns:
asset 1     0.100
asset 2     0.050

Portfolio return:
     0.090
```

Portfolio Choice 1

This notebook analyses the effect of leverage and diversification on the portfolio performance, and solves a (simple) optimal portfolio choice problem.

Load Packages and Extra Functions

```
using Printf

include("src/printmat.jl");
```

```
using Plots
default(size = (480,320),fmt = :png)   #or :svg
```

Portfolio Return: Expected Value and Variance

We form a portfolio by combining n assets: v is the vector of n portfolio weights. The portfolio return is

$R_v = v'R,$

where R is a vector of returns of the n assets.

```
v = [0.8,0.2]
printblue("Portfolio weights:")
printmat(v;rowNames=["asset 1","asset 2"])
```

```
Portfolio weights:
asset 1     0.800
asset 2     0.200
```

The expected portfolio return and the portfolio variance can be computed as

$\mathrm{E}R_v = v'\mu$ and

$\mathrm{Var}(R_v) = v'\Sigma v$,

where μ is a vector of expected expected (average) returns of the n assets and Σ the $n \times n$ variance-covariance matrix.

Also, the covariance of two portfolios (with vectors of portfolio weights v and x, respectively) can be computed as

$\mathrm{Cov}(R_v, R_x) = v'\Sigma x$.

```
μ = [9,6]/100                        #\mu[TAB] to get μ
Σ = [256 96;                         #\Sigma[TAB]
     96 144]/100^2

printblue("expected returns*100: ")
printmat(μ*100;rowNames=["asset 1","asset 2"],prec=2)

printblue("covariance matrix*100^2:")
printmat(Σ*100^2;rowNames=["asset 1","asset 2"],colNames=["asset 1","asset 2"],prec=2)
```

```
expected returns*100:
asset 1      9.00
asset 2      6.00

covariance matrix*100^2:
          asset 1   asset 2
asset 1    256.00     96.00
asset 2     96.00    144.00
```

```
ERᵥ   = v'μ
VarRᵥ = v'Σ*v

printlnPs("Expected portfolio return: ",ERᵥ)
printlnPs("Portfolio variance and std:",VarRᵥ,sqrt(VarRᵥ))
```

```
Expected portfolio return:      0.084
Portfolio variance and std:     0.020     0.142
```

```
x = [0.3,0.7]              #weights for portfolio x

printlnPs("Covariance of Rᵥ and Rₓ: ",v'Σ*x)
```

```
Covariance of Rᵥ and Rₓ:      0.014
```

Leverage: A Risky and a Riskfree Asset

Suppose you can invest in a risky asset (with return R, expected return μ and standard deviation σ) and also in a riskfree asset (at the rate R_f).

With the portfolio weight v on the risky asset, the portfolio return is

$R_p = vR + (1 - v)R_f$.

$v > 1$ is called leverage (and it is financed by borrowing at the rate R_f).

The average and standard deviation of the portfolio are

$\mathrm{E}R_p = v\mu + (1 - v)R_f$

and

$\mathrm{Std}(R_p) = |v|\sigma$,

where (μ, σ) are the average return and standard deviation of the risky asset.

By considering different values of v, we can show what sort of combinations of $\mathrm{E}R_p$ and $\mathrm{Std}(R_p)$ that can be achieved.

A Remark on the Code

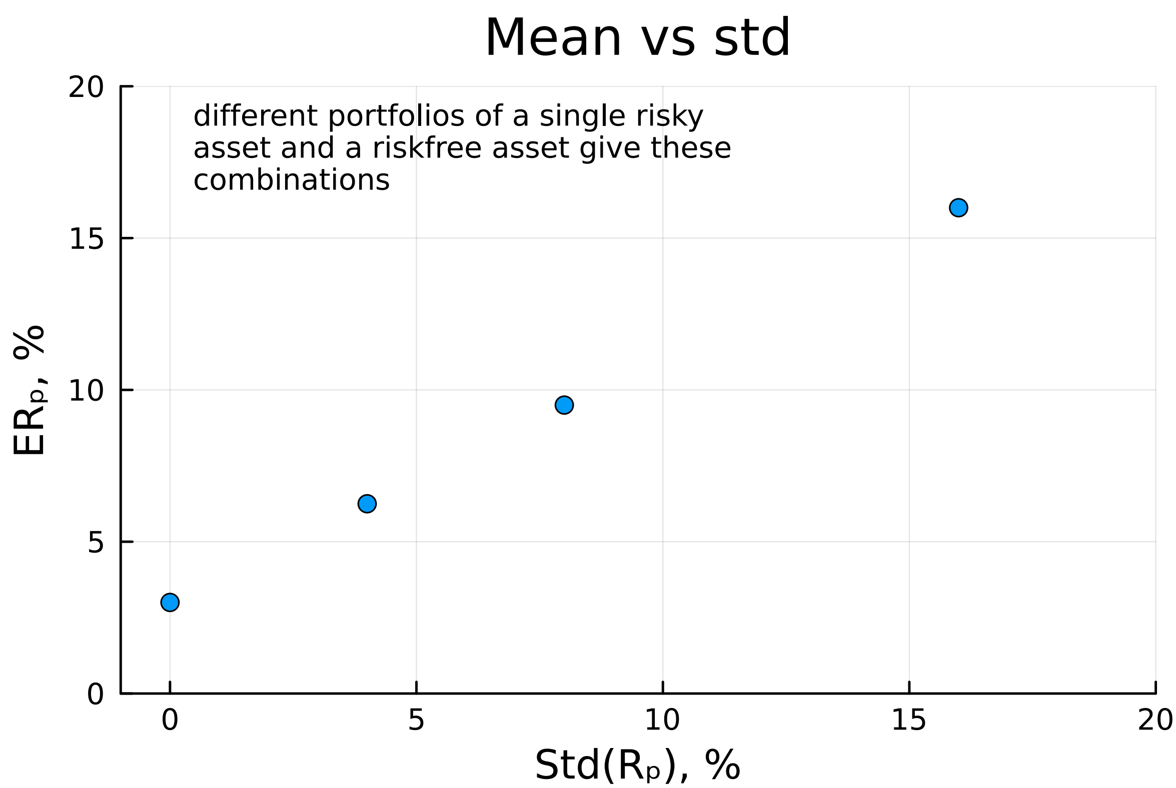
1. if `v=0.2`, then `println("hello is $v")` will print `hello is 0.2`
2. if `v_range=[0.5,0.25]`, then `1.0 .- v_range` will give `[1-0.5,1-0.25]`.

```
μ  = 9.5/100  #expected return of the risky asset
σ  = 8/100    #std of the risky asset
Rf = 3/100    #risk free return (interest rate)

v  = 0.5      #try a single value of v

ERₚ   = v*μ + (1 - v)*Rf
StdRₚ = abs(v)*σ
```

```
printlnPs("ERₚ and Std(Rₚ) when v=$v:",ERₚ,StdRₚ)
```

```
ERₚ and Std(Rₚ) when v=0.5:     0.062     0.040
```

```
v_range = [0,0.5,1,2]                           #try different values of v
ERₚ     = v_range*μ + (1 .- v_range)*Rf         #a vector of the same length as v_range, notice the
StdRₚ   = abs.(v_range)*σ                       #notice the dot.

txt = text("different portfolios of a single risky\nasset and a riskfree asset give these\ncom

p1 = scatter( StdRₚ*100,ERₚ*100,
              legend = false,
              ylim = (0,20),
              xlim = (-1,20),
              title = "Mean vs std",
              xlabel = "Std(Rₚ), %",
              ylabel = "ERₚ, %",
              annotation = (0.5,18,txt) )
display(p1)
```



Portfolio Choice: A Risky and a Riskfree Asset

Consider the objective function ("utility") $\mathrm{E}R_p - k/2 \times \mathrm{Var}(R_p)$.

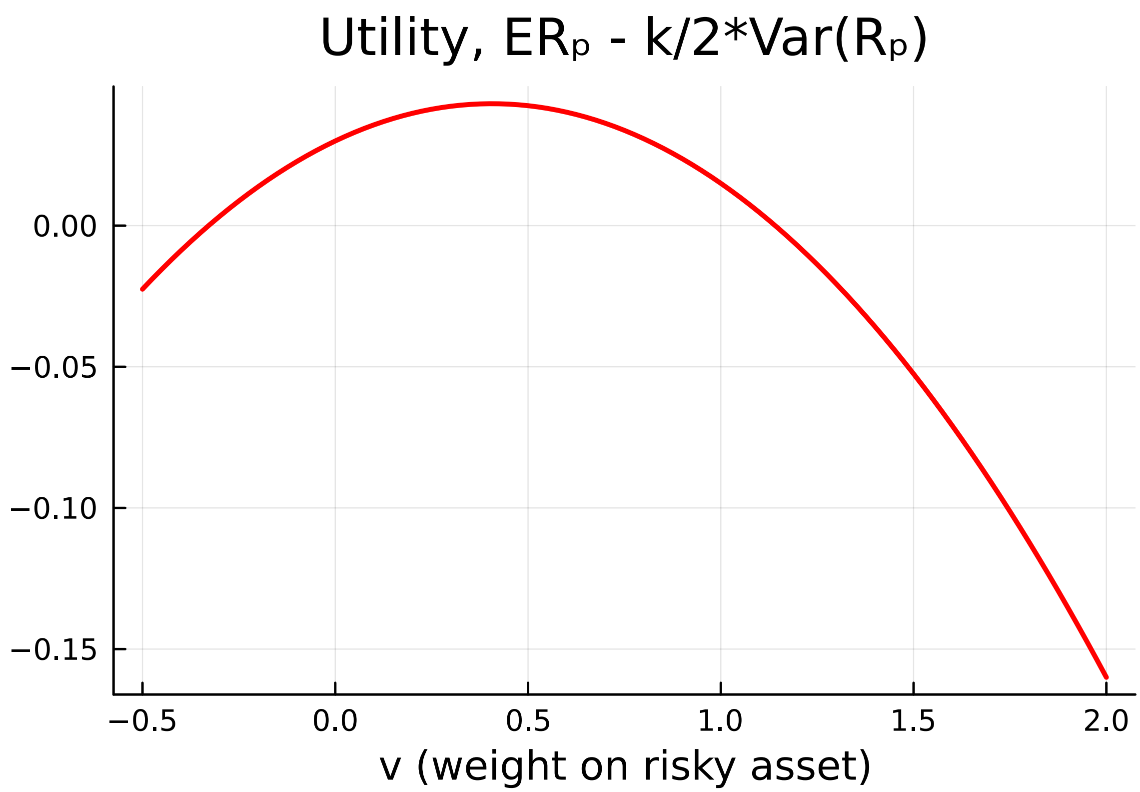
It depends on the weight v on the risky asset (and $1 - v$ on the riskfree asset), since both terms do (see the previous cells).

The optimal portfolio weight is

$v = \frac{\mu - R_f}{k\sigma^2}$

```
k       = 25      #risk aversion

v_range = range(-0.5,2,length=101)          #trying different portfolio weights
ERₚ     = v_range*μ + (1 .- v_range)*Rf     #a vector of the same length as v_range
VarRₚ   = v_range.^2*σ^2                    #v_range.^2 to square each element of v_range
Util    = ERₚ - k/2*VarRₚ

p1 = plot( v_range,Util,
```

```
           linecolor = :red,
           linewidth = 2,
           legend = false,
           title = "Utility, ERₚ - k/2*Var(Rₚ)",
           xlabel = "v (weight on risky asset)" )
display(p1)
```




```
vopt = (μ-Rf)/(k*σ^2)       #optimal solution (according to the lecture notes)

printblue("Optimal weights on risky and riskfree assets when k = $k: ")
printmat([vopt,1-vopt];rowNames=["risky","riskfree"])

printred("compare with the figure")
```

```
Optimal weights on risky and riskfree assets when k = 25:
risky         0.406
riskfree      0.594

compare with the figure
```

Diversification

The variance of an *equally weighted* portfolio of *two assets* is

$\text{Var}(R_p) = \sigma_{11}/4 + \sigma_{22}/4 + \sigma_{12}/2,$

where σ_{ii} the variance of asset i and σ_{ij} is the covariance of assets i and j. Notice that $\sigma_{12} = \rho\sqrt{\sigma_{11}\sigma_{22}}$, where ρ is the correlation.

Notice that we use σ_{ii} to denote a variance since it is tricky to write σ_i^2 in the code (it looks ugly, like `σ²₁`).

More generally, the variance of an equally weighted portfolio of *n assets* is

$\text{Var}(R_p) = (\bar{\sigma}_{ii} - \bar{\sigma}_{ij})/n + \bar{\sigma}_{ij},$

where $\bar{\sigma}_{ii}$ is the average variance (across the assets) and $\bar{\sigma}_{ij}$ is the average covariance. (In the code, we use `σᵢᵢ_avg` to denote this.)

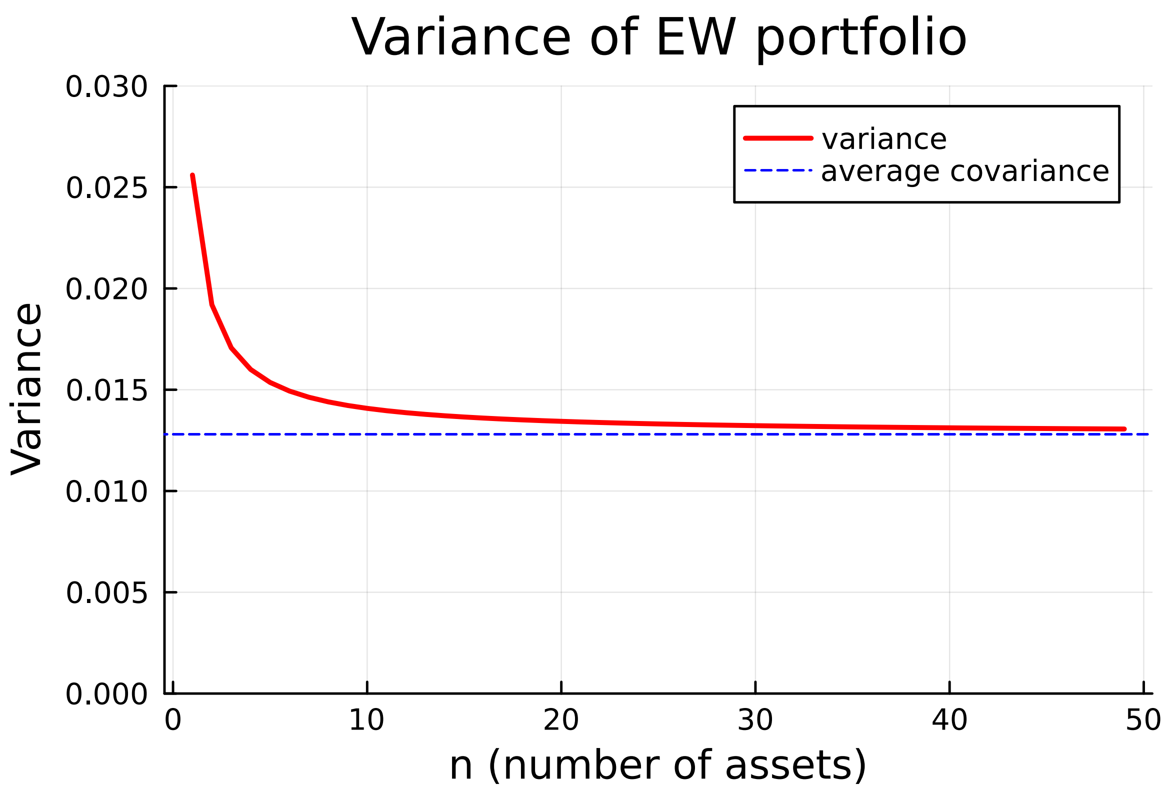
```
σ₁₁ = 256/100^2
σ₂₂ = σ₁₁             #assume the same variance of the two assets
ρ   = 0.5

VarRₚ = σ₁₁/4 + σ₂₂/4 + ρ*sqrt(σ₁₁*σ₂₂)/2

printmat([σ₁₁;VarRₚ];rowNames=["Individual variance","portfolio variance"])
```

```
Individual variance     0.026
portfolio variance      0.019
```

```
σᵢᵢ_avg = σ₁₁                       #average variance, for simplicity same as σ₁₁
σᵢⱼ_avg = ρ*sqrt(σᵢᵢ_avg*σᵢᵢ_avg)   #average covariance
n_range = 1:49                      #try all these values of n

VarRₚ = (σᵢᵢ_avg-σᵢⱼ_avg)./n_range .+ σᵢⱼ_avg     #variance of equally weighted portfolio

printblue("Portfolio variance and std, n = 1 to 3")
printmat([VarRₚ[1:3] sqrt.(VarRₚ[1:3])];colNames=["var","std"],rowNames=1:3,cell00="n")
```

```
Portfolio variance and std, n = 1 to 3
n       var       std
1     0.026     0.160
2     0.019     0.139
3     0.017     0.131
```

```
p1 = plot( n_range,VarRₚ,
           linecolor = :red,
           linewidth = 2,
           ylim = (0,0.03),
           label = "variance",
           title = "Variance of EW portfolio",
           xlabel = "n (number of assets)",
           ylabel = "Variance" )
hline!([σᵢⱼ_avg;σᵢⱼ_avg],line=(:dash,1),linecolor=:blue,label="average covariance")
display(p1)
```



Calculate the Average Correlation (extra)

A Remark of the Code

`tril(C,-1)` creates a new matrix where all $n(n-1)/2$ elements below the main diagonal are kept unchanged, and all other elements are replaced by zeros.

```
using LinearAlgebra

C = [1     0.50  0.25;         #a correlation matrix
     0.50  1     0.10;
     0.25  0.10  1   ]

n = size(C,1)
c_avg = sum(tril(C,-1))/(n*(n-1)/2)    #average of the n(n-1)/2
printlnPs(c_avg)                       #numbers below the main diagonal
```

```
0.283
```

Review of Statistics

This notebook shows some basic statistics needed for this course.

It uses the `Statistics` package (built in) for descriptive statistics (averages, autocorrelations, etc) and the Distributions.jl package for statistical distributions (pdf, cdf, etc).

Load Packages and Extra Functions

```
using Printf, Statistics, LinearAlgebra, DelimitedFiles, Distributions

include("src/OlsGMFn.jl")
include("src/printmat.jl");
```

```
using Plots, LaTeXStrings              #packages for plotting and LaTeX
default(size = (480,320),fmt = :png)  #or :svg
```

Distributions

Probability Density Function (pdf)

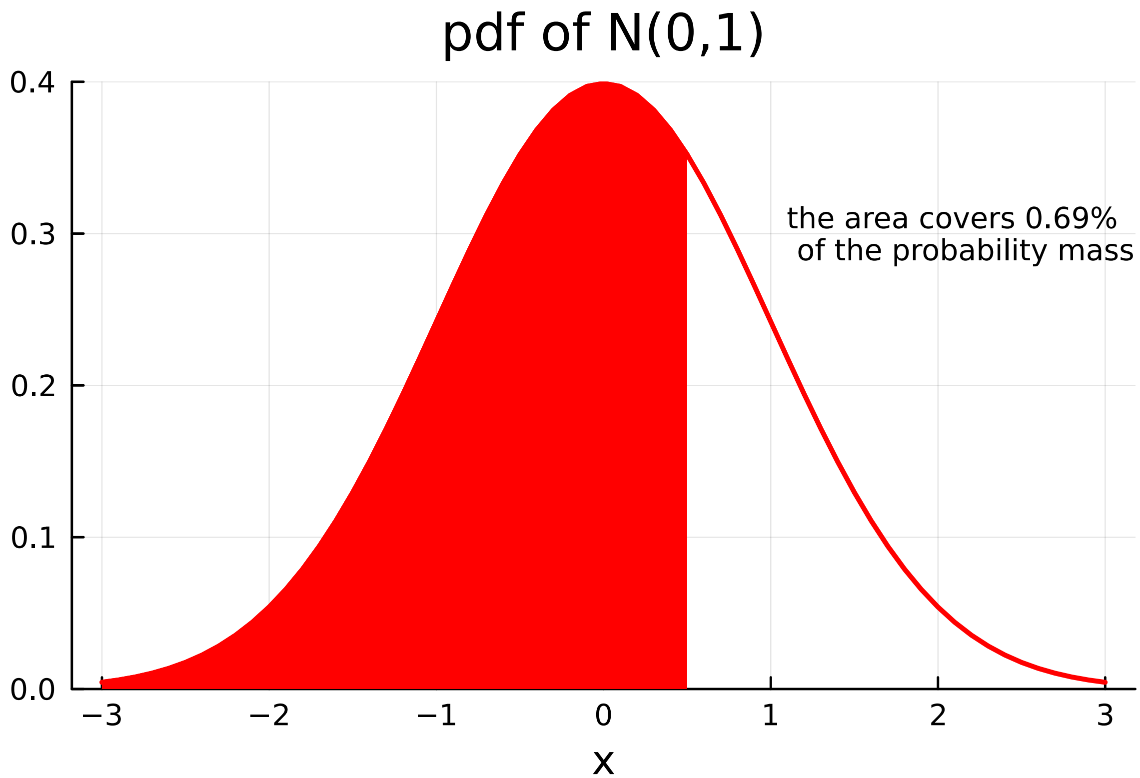
The cells below calculate and plot pdfs and cdfs of some distributions often used in the lecture notes. The `Distributions.jl` package has many more distributions.

A Remark on the Code

- Notice that the `Distributions.jl` package wants `Normal(μ,σ)`, where σ is the standard deviation. However, the notation in the lecture notes is $N(\mu, \sigma^2)$. For instance, $N(0,2)$ from the lectures is coded as `Normal(0,sqrt(2))`.
- `pdf.(Normal(0,1),x)` calculates the pdf of a standard normal variable at each value in the array `x`. Notice the dot (`.`).

```
x  = -3:0.1:3
xb = x[x.<=0.5]                     #pick out x values <= 0.5

pdfx  = pdf.(Normal(0,1),x)          #calculate the pdf of a N(0,1) variable
pdfxb = pdf.(Normal(0,1),xb)         #colour the area below this curve
Prb = cdf(Normal(0,1),0.5)
printlnPs("Pr(x<=0.5) ",Prb)

p1 = plot( x,pdfx,                            #plot pdf
           linecolor = :red,
           linewidth = 2,
           legend = nothing,
           ylims = (0,0.4),
           title = "pdf of N(0,1)",
           xlabel = "x",
           annotation = (1.1,0.3,text("the area covers $(round(Prb,digits=2))%\n of the probab
plot!(xb,pdfxb,linecolor=:red,linewidth=2,legend=nothing,fill=(0,:red))    #plot area under pd
display(p1)
```

```
Pr(x<=0.5)      0.691
```




```
printlnPs("5% quantile of N(0,1): ", quantile(Normal(0,1),0.05))
```

```
5% quantile of N(0,1):     -1.645
```

Expected Values and Variances

The random variable used in the example below can only take two values in the vector `x = [x₁,x₂]`, with probabilities in `p = [p₁,1-p₁]`. (π is already defined in one of the packages.)

```
x = [9.5,11]
p = [0.4,0.6]

μ = sum(p.*x)               #or `dot(π,x)`
printlnPs("μ: ",μ)

σ² =  dot(p,(x.-μ).^2)
printlnPs("σ²: ",σ²)
```

```
printlnPs("\nCheck the variance against the theoretical result:", p[1]*p[2]*(x[2]-x[1])^2)
```

```
       μ:     10.400
      σ²:      0.540

Check the variance against the theoretical result:     0.540
```

Expected Value of a function and Its Derivative

```
Elnx = dot(p,log.(x))     #expected value of log(x)
println("log(x) and its expected value")
Edlnx = dot(p,1.0./x)      #derivative of log(x) is 1/x

xut = hcat([x;μ],[log.(x);Elnx],[1.0./x;Edlnx])
printmat(xut;rowNames=["state 1","state 2","expected value"],colNames=["x","log(x)","dlog(x)/d
```

```
log(x) and its expected value
                         x      log(x)  dlog(x)/dx
state 1              9.500       2.251       0.105
state 2             11.000       2.398       0.091
expected value      10.400       2.339       0.097
```

Expected Value and Variance of Linear Combinations

```
μ = [11.5, 9.5, 6]/100          #expected returns for 3 assets
Σ = [166  34  58;               #covariance matrix
      34  64   4;
      58   4 100]/100^2
Rf = 0.03                       #riskfree return
```

```
0.03
```

```
w = [0.25,0.68,0.07]      #portfolio weights summing to 1
```

```
3-element Vector{Float64}:
 0.25
 0.68
 0.07
```

```
ERp = dot(w,μ)
VarRp = w'*Σ*w;          #or dot(w,Σ,w)
```

```
0.005442940000000001
```

```
xut = [ERp,VarRp,sqrt(VarRp)]
printmat(xut;rowNames=["mean","variance","std"])
```

```
mean          0.098
variance      0.005
std           0.074
```

Linear Regressions

using the included function `OlsGMFn(Y,X)` where `Y`

```
x = readdlm("Data/FFmFactorsPs.csv",',',skipstart=1)

                #yearmonth, market, small minus big, high minus low
(ym,Rme,RSMB,RHML) = (x[:,1],x[:,2]/100,x[:,3]/100,x[:,4]/100)
x = nothing

printlnPs("Sample size:",size(Rme))
```

```
Sample size:    (388,)
```

```
Y = Rme                    #T-vector, to get standard OLS notation
T = size(Y,1)
X = [ones(T) RSMB RHML]    #TxK matrix, regressors

(b,_,_,V,R²) = OlsGMFn(Y,X)
Stdb = sqrt.(diag(V))        #standard errors, assuming iid errors
```

```
printblue("OLS Results:\n")
xNames = ["c","SMB","HML"]
printmat(b,Stdb,colNames=["b","std"],rowNames=xNames)

printlnPs("R²: ",R²)
```

```
OLS Results:

            b       std
c       0.007     0.002
SMB     0.217     0.073
HML    -0.429     0.074

      R²:      0.134
```

Test a Single Coefficient

Suppose we want to test if the 2nd coefficient os equal to 0.3 THis can be done as follows. If the |t-stat|, is larger than 1.645, then you cannot reject the null hypothesis on the 10% significance level

```
t = (b[2]-0.3)/Stdb[2]

printlnPs("|t| : ",abs(t))
```

```
    |t| :      1.132
```

Test Several coefficients

or some linear combinations of them, then

Since the estimator $\hat{\beta}_{k\times 1}$ satisfies

$\hat{\beta} - \beta_0 \sim N(0, V_{k\times k})$,

we can easily apply various tests. Consider a joint linear hypothesis of the form

$H_0 : R\beta = q$,

where R is a $J \times k$ matrix and q is a J-vector. To test this, use

$(R\beta - q)'(RVR')^{-1}(R\beta - q) \overset{d}{\to} \chi^2_J.$

```
R = [0 1 0;
     0 0 1]
q = [0.3;0]              #β[2]=0.3 and β[3]=0
```

```
2-element Vector{Float64}:
 0.3
 0.0
```

```
Q = (R*b-q)'*inv(R*V*R')*(R*b-q)

printlnPs("Compare with 10% critical value of X^2(2): ",Q," ",quantile(Chisq(2),0.9))
```

```
Compare with 10% critical value of X^2(2):     34.669              4.605
```

Mean Variance Frontiers

This notebook calculates *(a)* the mean-variance frontiers when there are no portfolio restrictions; *(b)* the tangency portfolio.

Load Packages and Extra Functions

```
using Printf, LinearAlgebra

include("src/printmat.jl");

prec = 2;    #default number of decimal digits in printmat()
```

```
using Plots
default(size = (480,320),fmt = :png)   #or :svg
```

The MV Frontier

Mean variance (MV) analysis starts with providing the vector of expected returns μ and the covariance matrix Σ of the investable assets.

Then, it plots the "mean variance" frontier: it is a scatter plot showing the lowest possible portfolio standard deviation ($\text{Std}(R_p)$) on the horizontal axis (yes, the standard deviation, not the variance) at a required average return ($\text{E}R_p = \mu^*$) on the vertical axis. We consider many different μ^* values to create the scatter. In most figures we connect the dots to form a curve.

Remember: to calculate the expected return and the variance of a portfolio with portfolio weights in the vector w, use

$\text{E}R_p = w'\mu$ and

$\text{Var}(R_p) = w'\Sigma w$.

Also, the sum of the portfolio weights should equal 1.

MV Frontier with Two Assets

With only two investable assets, all portfolios of them are on the MV frontier. We can therefore trace out the entire MV frontier by calculating the means and standard deviations of a range of portfolios with different weights $(w_1, 1 - w_1)$ on the two assets.

```
μ = [11.5, 6]/100            #expected returns
Σ = [166   58;               #covariance matrix
      58  100]/100^2

printblue("expected returns, %:")
printmat(μ*100;rowNames=["asset 1","asset 2"],prec)         #prec is defined above

printblue("covariance matrix, bp:")
printmat(Σ*100^2;rowNames=["asset 1","asset 2"],colNames=["asset 1","asset 2"],prec)
```

```
expected returns, %:
asset 1     11.50
asset 2      6.00

covariance matrix, bp:
          asset 1   asset 2
asset 1    166.00     58.00
asset 2     58.00    100.00
```

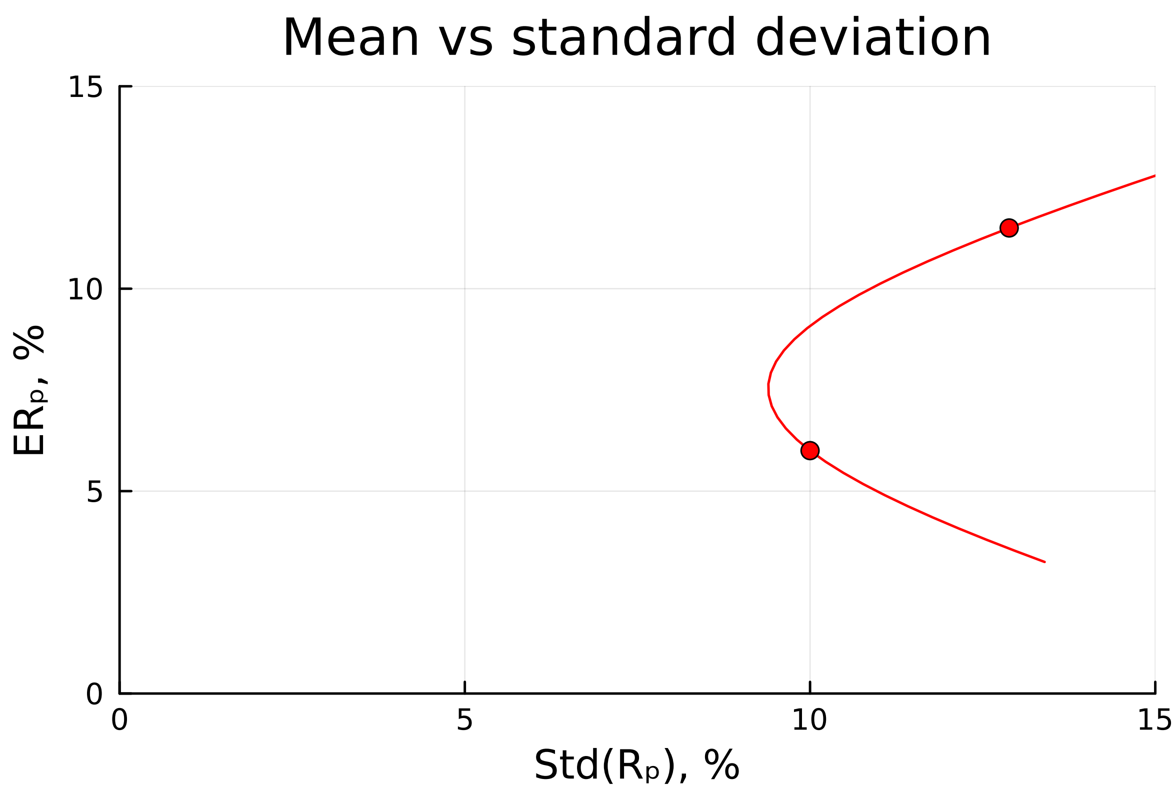
Remarks on the Code

For the the code in the next cell, notice the following:

1. `(ERₚ,StdRₚ) = (fill(NaN,L),fill(NaN,L))` creates two L-vectors filled with NaNs. (This is the same as having two lines of code.)
2. `local w` makes sure that `w` inside the loop does not affect any previously assigned `w`. In contrast, a `global w` would make the value created in the loop overwrite any previous `w`. In a notebook, `global` is implicitly assumed. In a script (.jl) file, you typically have to explicitly indicate local/global.

```
L        = 41                              #number of w₁ values to try
w₁_range = range(1.5,-0.5,length=L)        #different possible weights on asset 1

(ERₚ,StdRₚ) = (fill(NaN,L),fill(NaN,L))    #to put the results in
for i in 1:L
    #local w                                 #local/global is needed in script
    w        = [w₁_range[i],1-w₁_range[i]]  #weights on asset 1 and 2
    ERₚ[i]    = w'μ
    StdRₚ[i] = sqrt(w'Σ*w)
end
```

```
p1 = plot( StdRₚ*100,ERₚ*100,
           legend = nothing,
           linecolor = :red,
           xlim = (0,15),
           ylim = (0,15),
           title = "Mean vs standard deviation",
           xlabel = "Std(Rₚ), %",
           ylabel = "ERₚ, %" )
scatter!(sqrt.(diag(Σ))*100,μ*100,markercolor=:red)
display(p1)
```



Portfolios of 3 or More (Risky) Assets

The next few cells define the average returns and the covariance matrix for 3 assets and illustrate a few portfolios.

```
μ = [11.5, 9.5, 6]/100          #expected returns
Σ = [166  34  58;               #covariance matrix
      34  64   4;
      58   4 100]/100^2
Rf = 0.03                       #riskfree return

assetNames = ["A","B","C"]
printblue("μ and Rf in %:")
printmat(μ*100;rowNames=assetNames,prec)
printmat(Rf*100;prec)

printblue("Σ in bp:")
printmat(Σ*100^2;rowNames=assetNames,colNames=assetNames,prec)
```

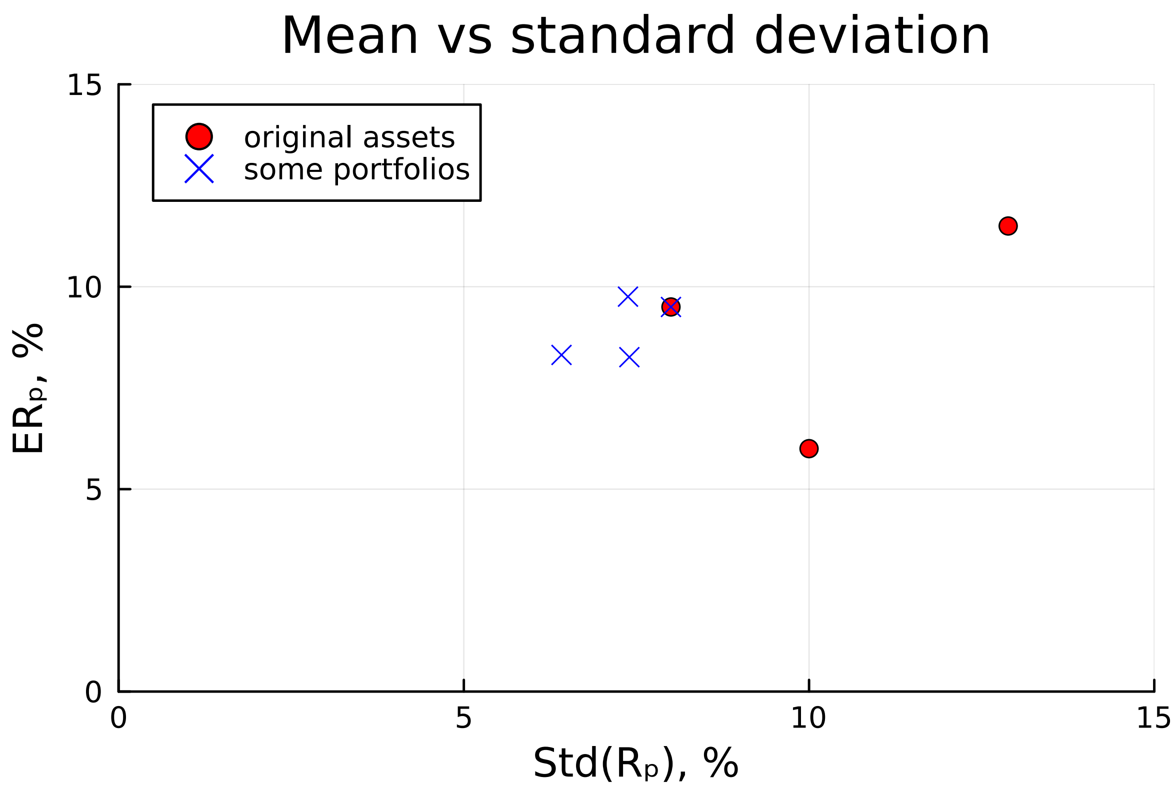
```
μ and Rf in %:
A     11.50
B      9.50
C      6.00

      3.00

Σ in bp:
          A         B         C
A    166.00     34.00     58.00
B     34.00     64.00      4.00
C     58.00      4.00    100.00
```

```
wM = [0 0.22 0.02 0.25;            #different portfolios (one in each column)
      1 0.30 0.63 0.68;
      0 0.48 0.35 0.07]
K = size(wM,2)                     #number of different portfolios

(ERₚ,StdRₚ) = (fill(NaN,K),fill(NaN,K))
for i in 1:K                      #loop over columns in wM
    ERₚ[i]   = wM[:,i]'μ
    StdRₚ[i] = sqrt(wM[:,i]'Σ*wM[:,i])
end

printblue("mean and std (in %) of portfolio: ")
printmat([ERₚ';StdRₚ']*100;colNames=["A","1","2","3"],rowNames=["mean","std"],prec)
```

```
mean and std (in %) of portfolio:
             A         1         2         3
mean      9.50      8.26      8.31      9.75
std       8.00      7.40      6.41      7.38
```

```
p1 = scatter( sqrt.(diag(Σ))*100,μ*100,
              markercolor = :red,
              label = "original assets",
              xlim = (0,15),
              ylim = (0,15),
              title = "Mean vs standard deviation",
              xlabel = "Std(Rₚ), %",
              ylabel = "ERₚ, %",
              legend = :topleft )
```

```
scatter!(StdRₚ*100,ERₚ*100,marker=:x,markercolor=:blue,label="some portfolios")
display(p1)
```



Calculating the MV Frontier: 3 or More (Risky) Assets

To find the MV frontier with 3 or more assets we have to solve the optimization problem:

$\min \mathrm{Var}(R_p)$ s.t. $\mathrm{E}R_p = \mu^*$ and $\sum_{i=1}^{n} w_i = 1$.

This can be done with a numerical minimization routine or by linear algebra (at least when we do not put any further restrictions on the portfolio weights). The next cells use the linear algebra approach: it solves for w from the following linear equations (first order conditions):

$$\begin{bmatrix} \Sigma & \mu & \mathbf{1}_n \\ \mu' & 0 & 0 \\ \mathbf{1}_n' & 0 & 0 \end{bmatrix} \begin{bmatrix} w \\ \lambda \\ \delta \end{bmatrix} = \begin{bmatrix} \mathbf{0}_n \\ \mu^* \\ 1 \end{bmatrix}$$

```
"""
    MVCalc(μˣ,μ,Σ)

Calculate the std and weights of a portfolio (with mean return μ⁺) on MVF of risky assets.
We use μˣ to denote the required average return, si

# Remark
- The code could be made quicker by calculating `Af = factorize(A)` once and then loop
over different elements in a vector of `μˣ` values as in `wλδ = ...` but using `Af`  instead o

"""
function MVCalc(μˣ,μ,Σ) #the std of a portfolio on MVF of risky assets
    n    = length(μ)
    A     = [Σ        μ ones(n);   #A is just a name of this matrix, it's not asset A
             μ'       0 0;
             ones(n)' 0 0];
    wλδ   = A\[zeros(n);μˣ;1]
    w     = wλδ[1:n]
    StdRp = sqrt(w'Σ*w)
    return StdRp,w
end
```

```
MVCalc
```

```
(StdAt10,wAt10) = MVCalc(0.1,μ,Σ)
printblue("Testing: the portfolio with a mean return of 10%")
printlnPs("\nstd: ",StdAt10)

printblue("\nw and its sum: ")
printmat([wAt10;sum(wAt10)];rowNames=[assetNames;"sum"],prec)
```

```
Testing: the portfolio with a mean return of 10%

std:      0.077

w and its sum:
A        0.29
B        0.69
C        0.02
sum      1.00
```

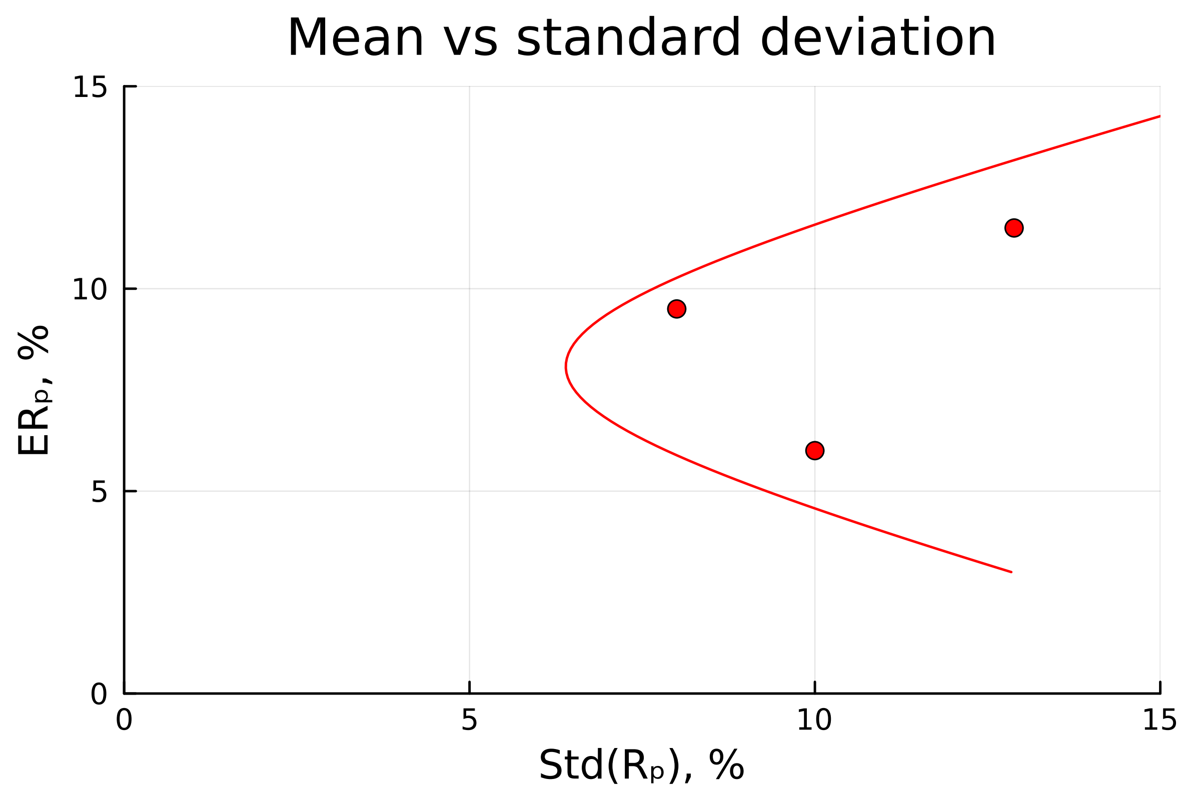
A Remark on the Code

- `MVCalc(μˣ,μ,Σ)[1]` picks out the first output from the `MVCalc` function.
- `StdRₚ = [MVCalc(μˣ,μ,Σ)[1] for μˣ in μˣ_range]` is a loop over the different μˣ values in `μˣ_range`

It is the same as writing an explicit loop as in

```
L      = length(μˣ_range)
StdRₚ = fill(NaN,L)
for i in 1:L
    StdRₚ[i] = MVCalc(μˣ_range[i],μ,Σ)[1]
end
```

```
μˣ_range = range(Rf,0.15,length=101)

StdRₚ = [MVCalc(μˣ,μ,Σ)[1] for μˣ in μˣ_range]  #loop over different required average returns,

p1 = plot( StdRₚ*100,μˣ_range*100,
           legend = nothing,
           linecolor = :red,
           xlim = (0,15),
           ylim = (0,15),
           title = "Mean vs standard deviation",
           xlabel = "Std(Rₚ), %",
           ylabel = "ERₚ, %" )
scatter!(sqrt.(diag(Σ))*100,μ*100,markercolor=:red)
display(p1)
```



Calculating the MV Frontier (of Risky and Riskfree Assets)

All portfolios on the MV frontier of both risky and riskfree have (a vector of) portfolio weights on the risky assets as in

$w = \frac{\mu^{*}-R_f}{\mu^{e\prime}\Sigma^{-1}\mu^{e}}\Sigma^{-1}\mu^{e}$,

where μ^{*} is the required average return.

The weight of the riskfree asset is $1 - \mathbf{1}'w$.

The cell also contains an alternative formulation based on the first order conditions.

```
"""
Calculate the std of a portfolio (with mean μˣ) on MVF of (Risky,Riskfree)

# Remark
- This code could be speeded up by calculating `(Σ_1*μᵉ)/(μᵉ'Σ_1*μᵉ) ` once and then
multiply with different values of ``(μˣ-Rf)`.
```

```
"""
function MVCalcRf(μˣ,μ,Σ,Rf)
    μᵉ    = μ .- Rf                 #expected excess returns
    Σ_1   = inv(Σ)
    w     = (μˣ-Rf)/(μᵉ'Σ_1*μᵉ) * Σ_1*μᵉ
    StdRp = sqrt(w'Σ*w)
    return StdRp,w
end


"""
Alternative calculation of the std of a portfolio (with mean μˣ) on MVF of (Risky,Riskfree)
"""
function MVCalcRfX(μˣ,μ,Σ,Rf)          #calculates the std of a portfolio
    n     = length(μ)                     #on MVF of (Risky,Riskfree)
    μᵉ    = μ .- Rf
    A     = [Σ μᵉ;
             μᵉ' 0]
    wλ    = A\[zeros(n);(μˣ-Rf)]
    w     = wλ[1:n]
    StdRp = sqrt(w'Σ*w)
    return StdRp,w
end
```

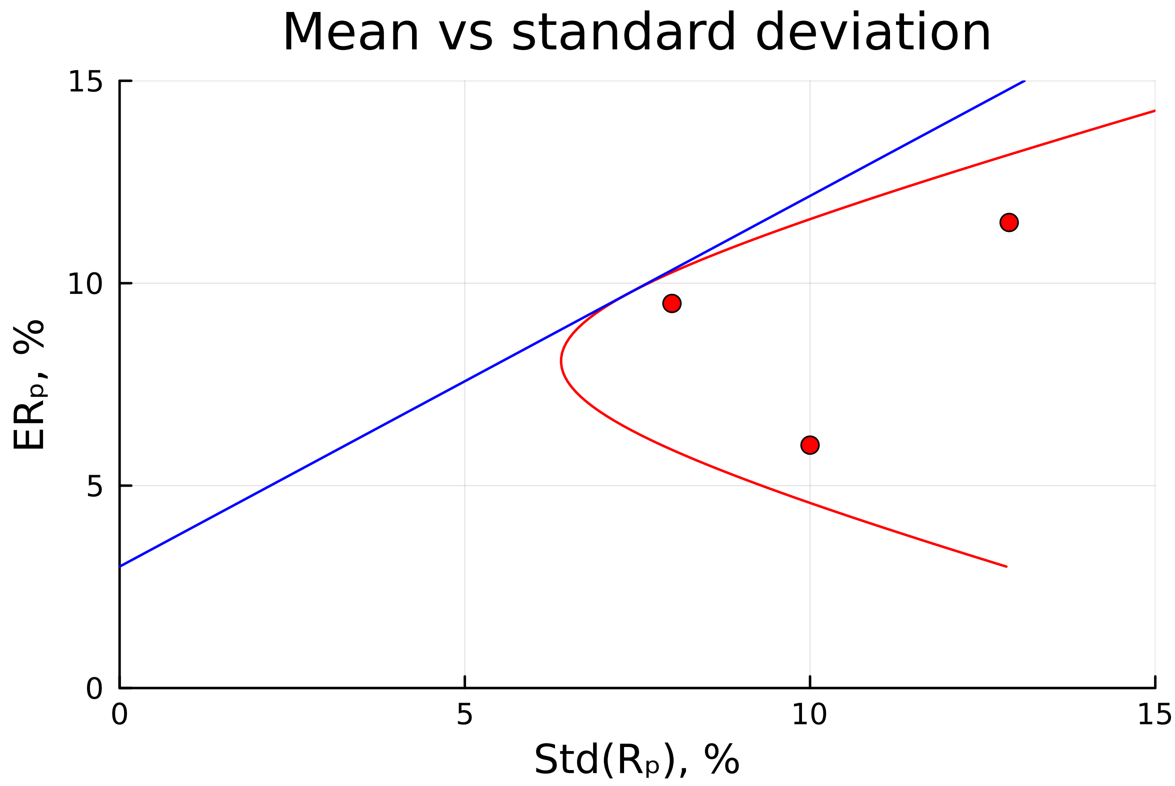
```
MVCalcRfX
```

```
(Std,w) = MVCalcRf(0.1,μ,Σ,Rf)

printblue("Testing: the portfolio with a mean return of 10%")
printlnPs("\nstd: ",Std)

printlnPs("\nw and its sum: ")
printmat([w;sum(w)];rowNames=[assetNames;"sum"],prec)

printlnPs("weight on riskfree:",1-sum(w))
```

```
Testing: the portfolio with a mean return of 10%

std:      0.076

w and its sum:
```

```
A        0.26
B        0.71
C        0.07
sum      1.04

weight on riskfree:    -0.037
```

```
StdRpRf = [MVCalcRf(μˣ,μ,Σ,Rf)[1] for μˣ in μˣ_range]

p1 = plot( [StdRₚ StdRpRf]*100,μˣ_range*100,
           legend = nothing,
           linecolor = [:red :blue],
           xlim = (0,15),
           ylim = (0,15),
           title = "Mean vs standard deviation",
           xlabel = "Std(Rₚ), %",
           ylabel = "ERₚ, %" )
scatter!(sqrt.(diag(Σ))*100,μ*100,markercolor=:red)
display(p1)
```



Tangency Portfolio

The tangency portfolio is a particular portfolio on the MV frontier of risky and riskfree, where the weights on the risky assets sum to 1. It is therefore also on the MV frontier of risky assets only. The vector of portfolio weights is

$$w_T = \frac{\Sigma^{-1}\mu^e}{1'\Sigma^{-1}\mu^e}$$

```
function MVTangencyP(μ,Σ,Rf)          #calculates the tangency portfolio
    n     = length(μ)
    μᵉ    = μ .- Rf                   #expected excess returns
    Σ_1   = inv(Σ)
    w     = Σ_1 *μᵉ/(ones(n)'Σ_1*μᵉ)
    muT   = w'μ + (1-sum(w))*Rf
    StdT  = sqrt(w'Σ*w)
    return w,muT,StdT
end
```

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

```
(wT,μT,σT) = MVTangencyP(μ,Σ,Rf)

printblue("Tangency portfolio: ")
printmat([wT;sum(wT)];rowNames=[assetNames;"sum"],prec)
printlnPs("mean and std of tangency portfolio, %: ",[μT σT]*100)
```

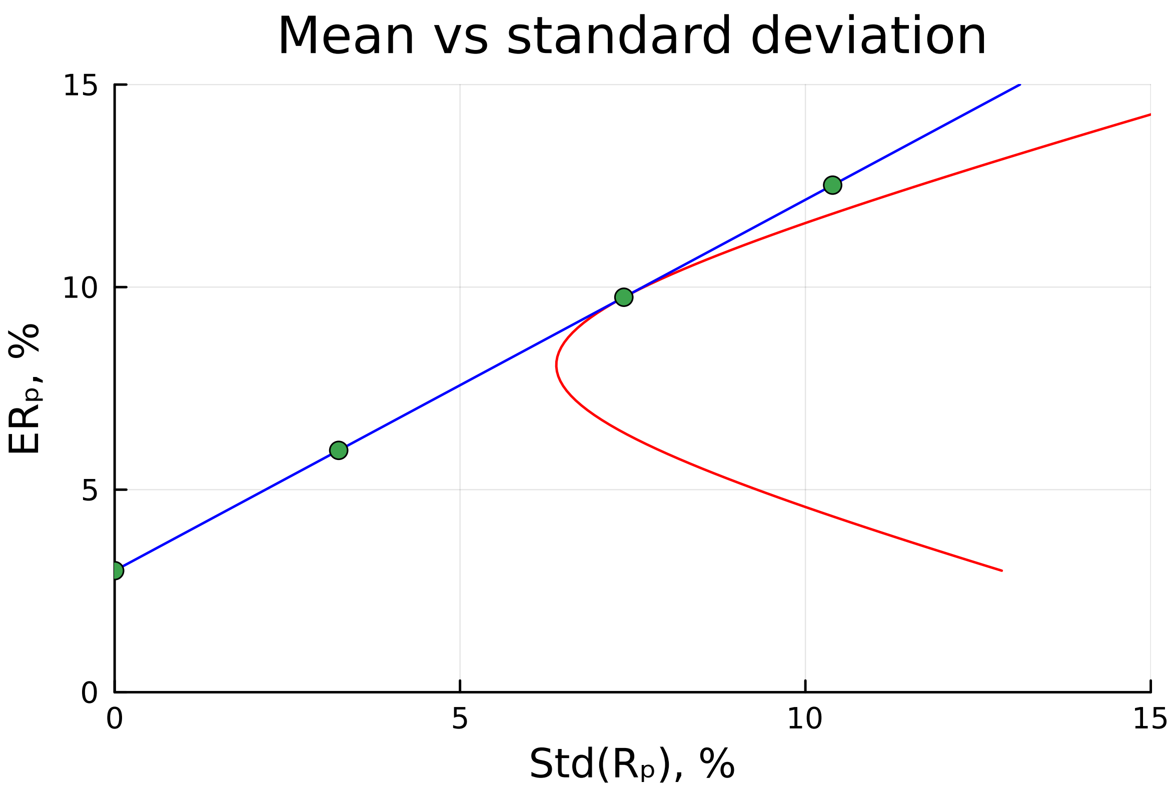
```
Tangency portfolio:
A        0.25
B        0.68
C        0.07
sum      1.00

mean and std of tangency portfolio, %:      9.750     7.372
```

By mixing the tangency portfolio and the riskfree, we can create any point on the MV frontier of risky and riskfree (also called the Capital Market Line, CML).

The code below shows the expected return and standard deviation of several portfolio (different v values) of the form

$R_v = vR_T + (1-v)R_f$ where $R_T = w_T'R$

```
v_range = [0;0.44;1;1.41]                                        #try different mixes of wT and Rf

ERᵥ    = v_range*μT + (1 .- v_range)*Rf
StdRᵥ = abs.(v_range)*σT

p1 = plot( [StdRₚ StdRpRf]*100,μˣ_range*100,
           legend= nothing,
           linecolor = [:red :blue],
           xlim = (0,15),
           ylim = (0,15),
           title = "Mean vs standard deviation",
           xlabel = "Std(Rₚ), %",
           ylabel = "ERₚ, %" )
scatter!(StdRᵥ*100,ERᵥ*100)
display(p1)
```



Examples of Tangency Portfolios

```
μb = [9; 6]/100                     #means
Σb = [ 256  0;
      0    144]/100^2
Rfb = 1/100
wT, = MVTangencyP(μb,Σb,Rfb)
printmat(wT;rowNames=["asset 1","asset 2"],prec)

wT, = MVTangencyP([13; 6]/100,Σb,Rfb)
printmat(wT;rowNames=["asset 1","asset 2"],prec)

Σb = [ 1  -0.8;
      -0.8    1]
wT, = MVTangencyP(μb,Σb,Rfb)
printmat(wT;rowNames=["asset 1","asset 2"],prec)

Σb = [ 1  0.8;
      0.8    1]
wT, = MVTangencyP(μb,Σb,Rfb)
printmat(wT;rowNames=["asset 1","asset 2"],prec)
```

```
asset 1      0.47
asset 2      0.53

asset 1      0.57
asset 2      0.43

asset 1      0.51
asset 2      0.49

asset 1      1.54
asset 2     -0.54
```

Drawing the MV Frontier Revisited (extra)

Recall: with only two investable assets, all portfolios of them are on the MV frontier.

We apply this idea by using the global minimum variance portfolio (see below for code) and the tangency portfolios (see above).

```
"""
Calculate the global minimum variance portfolio
"""
function MVMinimumVarP(μ,Σ,Rf)
    n      = length(μ)
    μᵉ     = μ .- Rf
    Σ_1    = inv(Σ)
    w      = Σ_1*ones(n)/(ones(n)'Σ_1*ones(n))
    mu     = w'μ + (1-sum(w))*Rf
    Std    = sqrt(w'Σ*w)
    return w,mu,Std
end
```

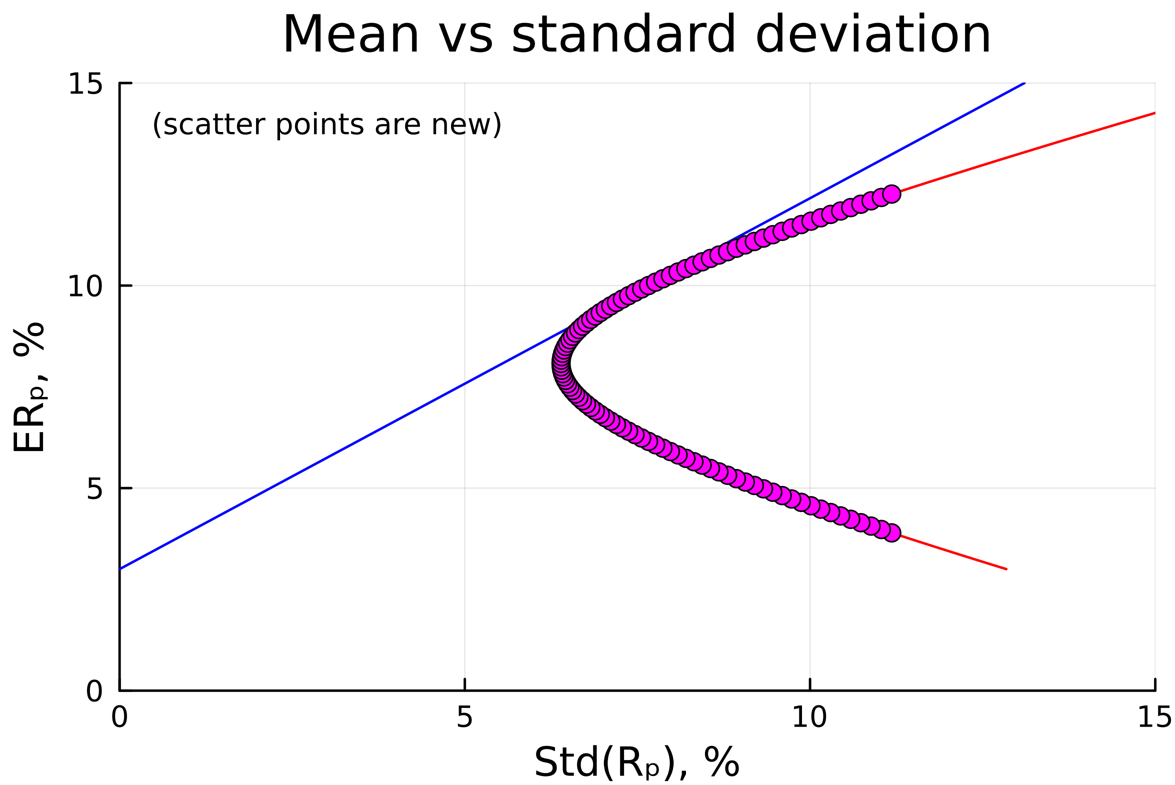
```
MVMinimumVarP
```

```
(wT,μT,σT) = MVTangencyP(μ,Σ,Rf)
(wMvp,muMvp,StdMvp) = MVMinimumVarP(μ,Σ,Rf)
printblue("Tangency and minimum variance portfolios: ")
printmat([wT wMvp];colNames=["tangency","min var"],rowNames=assetNames,prec)
```

```
Tangency and minimum variance portfolios:
   tangency   min var
A      0.25     -0.02
B      0.68      0.62
C      0.07      0.40
```

```
L = 101
v_range = range(-2.5,2.5,length=L)          #try different mixes of wT and wMvp

(ERᵥ,StdRᵥ)  = (fill(NaN,L),fill(NaN,L))
for i = 1:L
    #local w                                #local/global is needed in script
    w         = v_range[i]*wT + (1-v_range[i])*wMvp
    ERᵥ[i]    = w'μ
    StdRᵥ[i] = sqrt(w'Σ*w)
end

p1 = plot( [StdRₚ StdRpRf]*100,μˣ_range*100,
           legend = nothing,
           linecolor = [:red :blue],
           xlim = (0,15),
```

```
          ylim = (0,15),
          title = "Mean vs standard deviation",
          xlabel = "Std(Rₚ), %",
          ylabel = "ERₚ, %",
          annotation = (0.5,14,text("(scatter points are new)",8,:left)) )
scatter!(StdRᵥ*100,ERᵥ*100,markercolor=:magenta)
display(p1)
```



Mean Variance Frontier with Short Sales Constraints

This notebook alculates mean variances frontiers for two cases *(1)* when there are no restrictions on the portfolio weights and *(2)* when we impose the restriction that no weights can be negative.

We use the package OSQP.jl which solves problems of the type:

$\min 0.5\theta' P\theta + q'\theta$ subject to $l \leq A\theta \leq u$.

As an alternative, consider the EfficientFrontier.jl package.

Load Packages and Utility Functions

```
using Printf, LinearAlgebra, SparseArrays, OSQP

include("src/printmat.jl");
```

```
using Plots
default(size = (480,320),fmt = :png)
```

Inputs to MV Calculations

```
μ = [11.5, 9.5, 6]/100       #expected returns
Σ = [166  34  58;            #covariance matrix
      34  64   4;
      58   4 100]/100^2
Rf = 0.03
```

```
assetNames = ["A","B","C"]
printblue("μ in %:")
printmat(μ*100;rowNames=assetNames,prec=2)

printblue("Σ in bp:")
printmat(Σ*10000;rowNames=assetNames,colNames=assetNames,prec=2)

printblue("Rf in %:")
printlnPs(Rf*100)
```

```
μ in %:
A     11.50
B      9.50
C      6.00

Σ in bp:
          A         B         C
A    166.00     34.00     58.00
B     34.00     64.00      4.00
C     58.00      4.00    100.00

Rf in %:
     3.000
```

Traditional MV Calculations

(when there are no constraints) from the chapter on MV analysis.

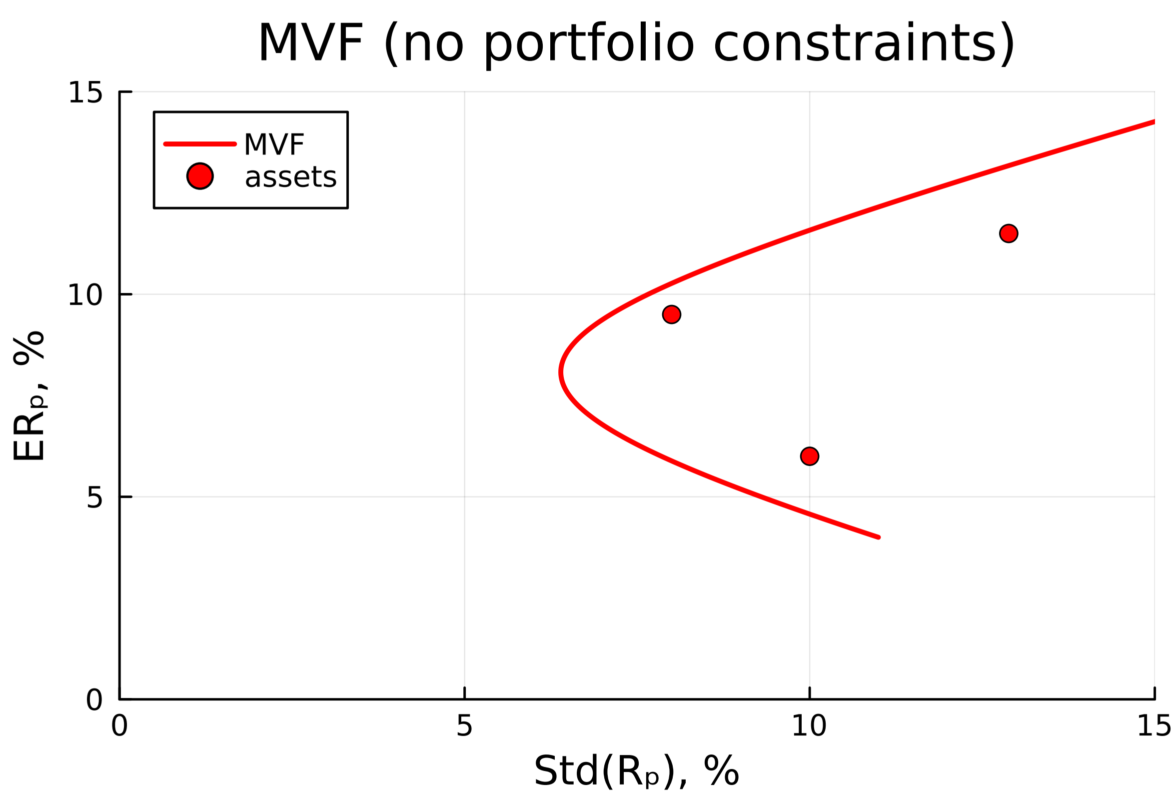
The file included below contains the function `MVCalc()` from the chapter on MV analysis. It calculates the (MV efficient) portfolio standard deviation for a given required return (`μˣ`), for the case of no restrictions on the portfolio except that the weights sum to one.

A Remark on the Code

- `[MVCalc(μˣ,μ,Σ)[1] for μˣ in μˣ_range]` is the same as writing a loop over all values in `μˣ_range` and extracting the first output from `MVCalc()` for each iteration.

```
include("src/MvCalculations.jl");
```

```
μˣ_range = range(0.04,0.15,length=201)

StdRₚ  = [MVCalc(μˣ,μ,Σ)[1] for μˣ in μˣ_range]        #MV calculations without portfolio res

p1 = plot( StdRₚ*100,μˣ_range*100,
           linecolor = :red,
           linewidth = 2,
           label = "MVF",
           legend = :topleft,
           xlim = (0,15),
           ylim = (0,15),
           title = "MVF (no portfolio constraints)",
           xlabel = "Std(Rₚ), %",
           ylabel = "ERₚ, %" )
scatter!(sqrt.(diag(Σ))*100,μ*100,color=:red,label="assets")
display(p1)
```



MV Frontier when Short Sales are Not Allowed

The code below solves the problem

$\min \operatorname{Var}(R_p)$ s.t. $\mathrm{E}R_p = \mu^*$,

and where we also require $w_i \geq 0$ and $\sum_{i=1}^{n} w_i = 1$.

It is straightforward to add other constraints, for instance, that all weights are between 0 and 0.5.

A Remark on the Code

- The function `MeanVarNoSS(μ,Σ,μˣ)` takes a vector `μˣ_range` as input and loops over those `μˣ_range[i]` values that are feasible (must be between the lowest and highest values in μ).

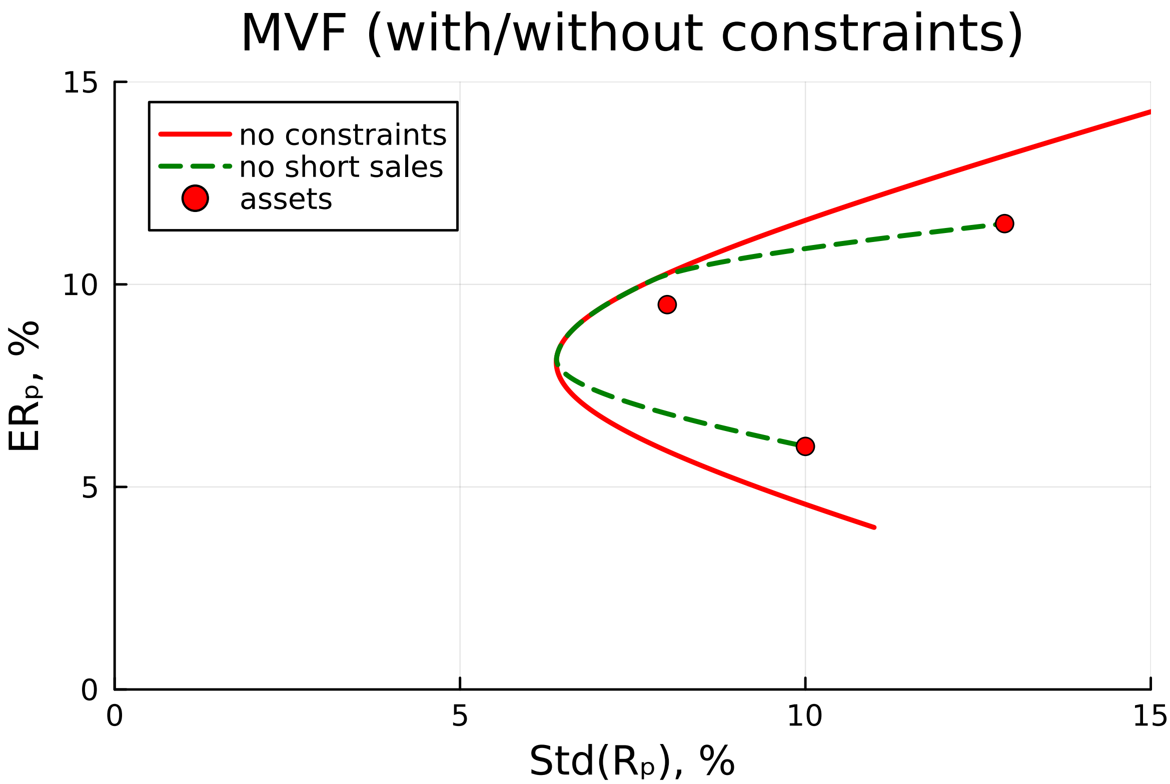
```
"""
    MeanVarNoSS(μ,Σ,μˣ_range_range)

Calculate mean variance frontier when short sales are not allowed.


# Input
- `μ::Vector`:        n vector, mean returns
- `Σ::Matrix`:        nxn, covariance matrix of returns, can contain riskfree assets
- `μˣ_range::Vector`: K vector, mean returns to calculate results for

# Output
- `StdRp::Vector`:    K vector, standard deviation of mean-variance portfolio (risky only) wit
- `w_p::Matrix`:      Kxn, portfolio weights of       ""

# Requires
- LinearAlgebra, SparseArrays, OSQP

"""
function MeanVarNoSS(μ,Σ,μˣ_range)   #MV with no short-sales, numerical minimization

    (K,n) = (length(μˣ_range),length(μ))
    vv    = findall( minimum(μ) .<= μˣ_range .<= maximum(μ) )  #solve only if feasible

    P = sparse(2*Σ)                   #P and A must be sparse
    q = zeros(n)
    A = sparse([μ';ones(1,n);I])
    l = [NaN;1;zeros(n)]
```

```
    u = [NaN;1;ones(n)]

    prob = OSQP.Model()              #initial set up of problem
    settings = Dict(:verbose => false)
    OSQP.setup!(prob;P=P,q=q,A=A,l=l,u=u,settings...)

    (w_p,StdRp) = (fill(NaN,K,n),fill(NaN,K))
    for i in vv         #loop over (feasible) μˣ_range elements
        (l[1],u[1]) = (μˣ_range[i],μˣ_range[i])
        OSQP.update!(prob;l=l,u=u)   #update problem
        result = OSQP.solve!(prob)
        w = result.info.status == :Solved ? result.x : NaN
        if !any(isnan,w)
            w_p[i,:] = w
            StdRp[i] = sqrt(w'Σ*w)
        end
    end

    return StdRp, w_p

end
```

```
MeanVarNoSS
```

```
StdRₚ_no_ss = MeanVarNoSS(μ,Σ,μˣ_range)[1];    #MV calculations with no short sales
```

```
p1 = plot( [StdRₚ StdRₚ_no_ss]*100,μˣ_range*100,
           linecolor = [:red :green],
           linestyle = [:solid :dash],
           linewidth = 2,
           label = ["no constraints" "no short sales"],
           xlim = (0,15),
           ylim = (0,15),
           legend = :topleft,
           title  = "MVF (with/without constraints)",
           xlabel = "Std(Rₚ), %",
           ylabel = "ERₚ, %" )
scatter!(sqrt.(diag(Σ))*100,μ*100,color=:red,label="assets")
display(p1)
```


MVF (with/without constraints)
no constraints
no short sales
assets
ERp, %
Std(Rp), %
0
5
10
15
0
5
10
15

Betas and Covariance Matrices

This notebook estimates (single and multi-) index models. It uses those to construct (alternative) estimates of the covariance matrix of the asset returns.

Load Packages and Extra Functions

```
using Printf, LinearAlgebra, Statistics, DelimitedFiles

include("src/printmat.jl")
include("src/OlsM.jl");
```

Loading Data

We load data from two data files: for the returns on Fama-French equity factors and then also for the 25 Fama-French portfolios. To keep the output simple (easy to display...), we use only 5 of those portfolios.

```
x    = readdlm("Data/FFmFactorsPs.csv",',',skipstart=1)
Rme  = x[:,2]                #market excess return
RSMB = x[:,3]                #small minus big firms
RHML = x[:,4]                #high minus low book-to-market ratio
Rf   = x[:,5]                #interest rate

x  = readdlm("Data/FF25Ps.csv",',') #no header line: x is matrix
R  = x[:,2:end]                     #returns for 25 FF portfolios
Re = R .- Rf                        #excess returns for the 25 FF portfolios
Re = Re[:,[1,7,13,19,25]]           #use just 5 assets to make the printing easier

(T,n) = size(Re)                    #no. obs and  no. test assets
```

```
(388, 5)
```

Covariance Matrix with Average Correlations

The next cell contains to two functions that will help us construct a covariance matrix from *(a)* estimated standard deviations and *(b)* a common (average) number for all correlations.

```
"""
    vech(x,k=0)

Stack the matrix elements on and below the principal diagonal into a vector (k=0).
With k=-1, instead stacks just the elements below the diagonal.
"""
function vech(x,k=0)
    vv = tril(trues(size(x)),k)
    y  = x[vv]
    return y
end



"""
Cov_withSameCorr(σ,ρ)

Compute covariance matrix from vector of standard deviations (σ)
and a single correlation (assumed to be the same across all pairs).
"""
function Cov_withSameCorr(σ,ρ)
    σ = vec(σ)                           #to make sure it is a vector
    n = length(σ)
    CorrMat = ones(n,n)*ρ + (1-ρ)*I     #corr matrix, correlation = ρ
    CovMat  = (σ*σ').*CorrMat
    return CovMat
end
```

```
Cov_withSameCorr
```

```
S = cov(Re)  #Covariance matrix calculated from data (to compare with)
#printblue("Covariance matrix calculated from data (to compare with):")
#printmat(S)

C = cor(Re)              #nxn correlation matrix
printblue("Correlation matrix:")
printmat(C)
```

```
ρ_avg = mean(vech(C,-1))
printblue("Average correlation:")
printlnPs(ρ_avg)
```

```
Correlation matrix:
     1.000     0.821     0.664     0.581     0.468
     0.821     1.000     0.919     0.819     0.696
     0.664     0.919     1.000     0.909     0.805
     0.581     0.819     0.909     1.000     0.852
     0.468     0.696     0.805     0.852     1.000

Average correlation:
     0.753
```

```
Σ₁ = Cov_withSameCorr(std(Re,dims=1),ρ_avg)

printblue("Covariance matrix assuming same correlation:")
printmat(Σ₁)

printblue("Difference to data:")
printmat(Σ₁-S)
```

```
Covariance matrix assuming same correlation:
    73.475    39.483    33.433    32.422    32.509
    39.483    37.371    23.844    23.123    23.185
    33.433    23.844    26.796    19.580    19.632
    32.422    23.123    19.580    25.200    19.039
    32.509    23.185    19.632    19.039    25.335

Difference to data:
     0.000    -3.540     3.962     7.410    12.321
    -3.540    -0.000    -5.229    -2.013     1.778
     3.962    -5.229    -0.000    -4.052    -1.349
     7.410    -2.013    -4.052    -0.000    -2.494
    12.321     1.778    -1.349    -2.494    -0.000
```

Covariance Matrix from a Single-Index Model

Step 1: Do Regressions

A Remark on the Code

- The function `OlsM` is included in the file `OlsM.jl` (see the first cell of the notebook). It does OLS estimation and reports the point estimates and standard deviations of the residuals.
- The functions reports for several dependent variables (different columns in the first function input).

```
x          = [ones(T) Rme]                    #regressors

(b,σ)      = OlsM(Re,x)
(β,VarRes) = (b[2:2,:],vec(σ).^2)

colNames = [string("asset ",i) for i=1:n]
printblue("β for $n assets, from OLS of Re on constant and Rme:")
printmat(β,colNames=colNames,rowNames=["β on Rme"])
```

```
β for 5 assets, from OLS of Re on constant and Rme:
           asset 1   asset 2   asset 3   asset 4   asset 5
β on Rme     1.341     1.169     0.994     0.943     0.849
```

Step 2: Use OLS Estimates to Calculate the Covariance Matrix

The single index model implies that the covariance of assets i and j is

$\sigma_{ij} = \beta_i \beta_j \text{Var}(R_{mt}) + \text{Cov}(\varepsilon_{it}, \varepsilon_{jt})$,

but where we *assume* that $\text{Cov}(\varepsilon_{it}, \varepsilon_{jt}) = 0 \ \text{ if } \ i \neq j$

The betas are typically estimated by a linear regression (OLS), as above.

A Remark on the Code

- The `CovFromIndexModel()` function can handle both the case with one factor (as used here) and with several factors (a multi-index model, as will be used further down). In the code β is a $K \times n$ matrix, where K is the number of factors ($K = 1$ for the single index model) and n is the number of assets.

```
"""
    CovFromIndexModel(b,VarRes,Ω)

Calculate covariance matrix from a multi-factor model.

b is Kxn where K is the number of factors and n is the number of assets.

Cov(Ri,Rj) = bᵢ'*Ω*bⱼ, where Ω is the Cov(indices) and bᵢ is the vector of regression
coefficients when regressing Ri on a constant and the indices (and bⱼ is for asset j).
"""
function CovFromIndexModel(b,VarRes,Ω)    #coefs for regression i is in b[:,i]
    n    = length(VarRes)
    CovR = fill(NaN,n,n)
    for i = 1:n, j = 1:n          #loop over both i and j
        CovR[i,j] = i == j ? b[:,i]'Ω*b[:,i] + VarRes[i] : b[:,i]'Ω*b[:,j]
    end
    return CovR
end
```

```
CovFromIndexModel
```

```
σₘₘ = var(Rme)
Σ₂ = CovFromIndexModel(β,VarRes,σₘₘ)

printblue("Covariance matrix calculated from betas:")
printmat(Σ₂)

printblue("Difference to data:")
printmat(Σ₂-S)
```

```
Covariance matrix calculated from betas:
    73.475    33.232    28.269    26.797    24.146
    33.232    37.371    24.644    23.361    21.049
    28.269    24.644    26.796    19.872    17.906
    26.797    23.361    19.872    25.200    16.973
    24.146    21.049    17.906    16.973    25.335

Difference to data:
    -0.000    -9.791    -1.202     1.784     3.958
    -9.791    -0.000    -4.428    -1.775    -0.357
    -1.202    -4.428    -0.000    -3.760    -3.075
```

```
     1.784     -1.775     -3.760     -0.000     -4.560
     3.958     -0.357     -3.075     -4.560      0.000
```

Covariance Matrix from a Multi-Index Model

A multi-index model is based on

$R_{it} = a_i + b_i' I_t + \varepsilon_{it}$,

where b_i is a K-vector of slope coefficients.

If Ω is the covariance matrix of the indices I_t, then the covariance of assets i and j is

$\sigma_{ij} = b_i' \Omega b_j + \text{Cov}(\varepsilon_{it}, \varepsilon_{jt})$,

but where we assume that $\text{Cov}(\varepsilon_{it}, \varepsilon_{jt}) = 0$ if $i \neq j$

```
x          = [ones(T) Rme RSMB RHML]              #regressors

(b,σ)      = OlsM(Re,x)
(β,VarRes) = (b[2:end,:],vec(σ).^2)

printblue("OLS slope coefficients:")
printmat(β,colNames=colNames,rowNames=["β on Rme", "β on RSMB", "β on RHML"])
```

```
OLS slope coefficients:
            asset 1   asset 2   asset 3   asset 4   asset 5
β on Rme      1.070     1.080     1.035     1.056     1.041
β on RSMB     1.264     0.768     0.437     0.153    -0.088
β on RHML    -0.278     0.160     0.487     0.603     0.770
```

```
Ω = cov(x[:,2:end])      #covariance matrix of the (non-constant) factors

Σ₃ = CovFromIndexModel(β,VarRes,Ω)

printblue("Covariance matrix calculated from betas:")
printmat(Σ₃)

printblue("Difference to data:")
printmat(Σ₃-S)
```

```
Covariance matrix calculated from betas:
    73.475    41.847    31.050    25.449    18.940
    41.847    37.371    27.384    24.141    20.145
    31.050    27.384    26.796    22.239    20.109
    25.449    24.141    22.239    25.200    20.717
    18.940    20.145    20.109    20.717    25.335

Difference to data:
    -0.000    -1.176     1.578     0.436    -1.248
    -1.176    -0.000    -1.688    -0.995    -1.261
     1.578    -1.688    -0.000    -1.393    -0.873
     0.436    -0.995    -1.393    -0.000    -0.816
    -1.248    -1.261    -0.873    -0.816    -0.000
```

A Shrinkage Estimator

is

$\Sigma = \delta F + (1 - \delta)S$,

where $0 < \delta < 1$, F is the "target covariance matrix" (for instance, from the constant correlation approach or one of the index models) and S is the sample variance-covariance matrix.

This approach will (by definition) be worse in-sample, but may well be better out-of-sample (a forecast for the coming period).

```
δ = 0.6
Σ₄ = δ*Σ₁ + (1-δ)*S

printmat(Σ₄)
```

```
    73.475    40.899    31.849    29.458    27.581
    40.899    37.371    25.935    23.928    22.473
    31.849    25.935    26.796    21.201    20.172
    29.458    23.928    21.201    25.200    20.036
    27.581    22.473    20.172    20.036    25.335
```

Portfolio Choice 2

This notebook solves a portfolio optimization problem with several risky assets. The objective function of the investor trades off the portfolio expected return and variance. There are no restrictions on the portfolio weights, except that they must sum to 1 (across all assets, risky and riskfree).

Load Packages and Extra Functions

```
using Printf, LinearAlgebra

include("src/printmat.jl");
```

```
using Plots
default(size = (480,320),fmt = :png)    #or :svg
```

From Chapter on Mean-Variance Analysis

The file included below contains functions from the chapter on MV analysis, in particular, `MVTangencyP()` which calculates the tangency portfolio, `MVCalc()` which calculates the MV frontier from risky assets, and `MVCalcRf()` which does the same but from both risky assets and a riskfree asset.

```
include("src/MvCalculations.jl");
```

Optimal Portfolio Choice

An investor who maximizes

$\mathrm{E}R_p - \frac{k}{2}\mathrm{Var}(R_p)$,

subject to

$R_p = v'R^e + R_f$

will pick the portfolio weights (on the risky assets)

$v = \frac{1}{k}\Sigma^{-1}\mu^e$

The portfolio weight on the riskfree asset is $1 - \mathbf{1}'v$

```
"""
Calculate optimal portfolio weights
"""
function OptimalPortfolio(μ,Σ,Rf,k)
    μᵉ    = μ .- Rf               #expected excess returns
    v     = inv(Σ) * μᵉ/k         #optimal weights risky assets, 1-sum(v) in riskfree
    ERp   = v'μ + (1-sum(v))*Rf #expected return and std of optimal portfolio
    StdRp = sqrt(v'Σ*v)
    return v,ERp,StdRp
end
```

```
OptimalPortfolio
```

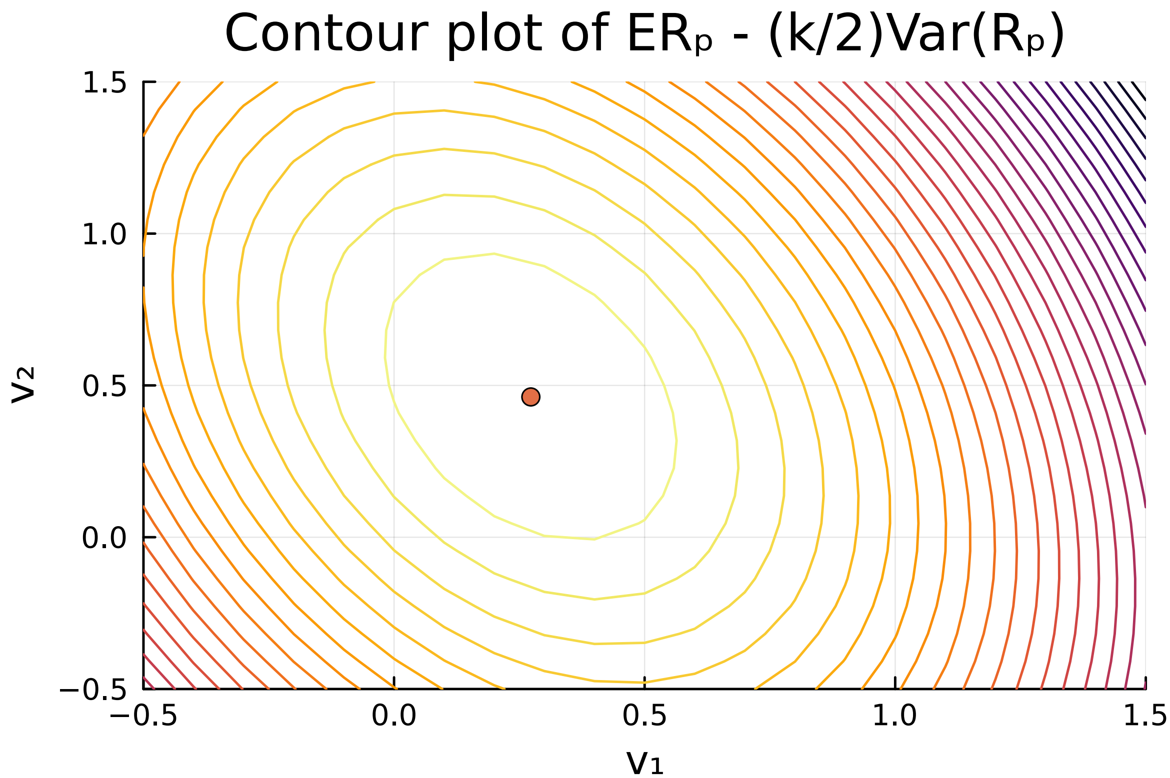
An Example with 2 Risky Assets and a Riskfree Asset

We first solve an example with 2 risky assets and a riskfree asset. In this case, we can plot how the objective function depends on the portfolio weights.

```
μₐ  = [8.5, 6.5]/100            #means, μₐ to indicate that this is case a
Σₐ  = [166  34;                 #covariance matrix
        34  64]/100^2
Rfₐ = 3/100;                    #riskfree rate
```

```
k = 9

(n₁,n₂)  = (21,23)
v₁_range = range(-0.5,1.5,length=n₁)    #portfolio weights asset 1
v₂_range = range(-0.5,1.5,length=n₂)    # ""                     2

Util = fill(NaN,n₁,n₂)
for i in 1:n₁, j in 1:n₂       #loop across portfolio weights
    #local v                   #local/global is needed in script
    v         = [v₁_range[i];v₂_range[j]]
    Util[i,j] = v'μₐ + (1-sum(v))*Rfₐ - (k/2)*v'Σₐ*v
end
```

```
p1 = contour( v₁_range,v₂_range,Util',         #notice the transpose
              xlims = (-0.5,1.5),
              ylims = (-0.5,1.5),
              legend = false,
              levels = 31,
              title = "Contour plot of ERₚ - (k/2)Var(Rₚ)",
              xlabel = "v₁",
              ylabel = "v₂")
scatter!([0.273],[0.462])
display(p1)
```



The next cell calls on `OptimalPortfolio()` to calculate the optimal weights - and also compares with the tangency portfolio.

```
vOpt, = OptimalPortfolio(μₐ,Σₐ,Rfₐ,k)        #find optimal portfolio

printblue("optimal portfolio weights on risky assets and riskfree when k = $k:")
assetNames = ["Asset 1","Asset 2","Riskfree"]
printmat([vOpt;(1-sum(vOpt))];rowNames=assetNames)

wT, = MVTangencyP(μₐ,Σₐ,Rfₐ)             #calculate the tangency portfolio and compare with

printblue("optimal weights/tangency portfolio:")
printmat(vOpt./wT;colNames=["vOpt/wTangency"],rowNames=assetNames)

printmagenta("...the optimal portfolio is proportional to the tangency portfolio")
```

```
optimal portfolio weights on risky assets and riskfree when k = 9:
Asset 1      0.273
Asset 2      0.462
```

```
Riskfree     0.264

optimal weights/tangency portfolio:
        vOpt/wTangency
Asset 1      0.736
Asset 2      0.736

...the optimal portfolio is proportional to the tangency portfolio
```

An Example with 3 Risky Assets (and a Riskfree)

We now apply the same approach on more assets - and compare the optimal portfolios for different investors (with different risk aversions).

We then show that the optimal portfolios are on the mean-variance frontier.

```
μ = [11.5, 9.5, 6]/100
Σ = [166  34  58;
      34  64   4;
      58   4 100]/100^2
Rf = 0.03;
```

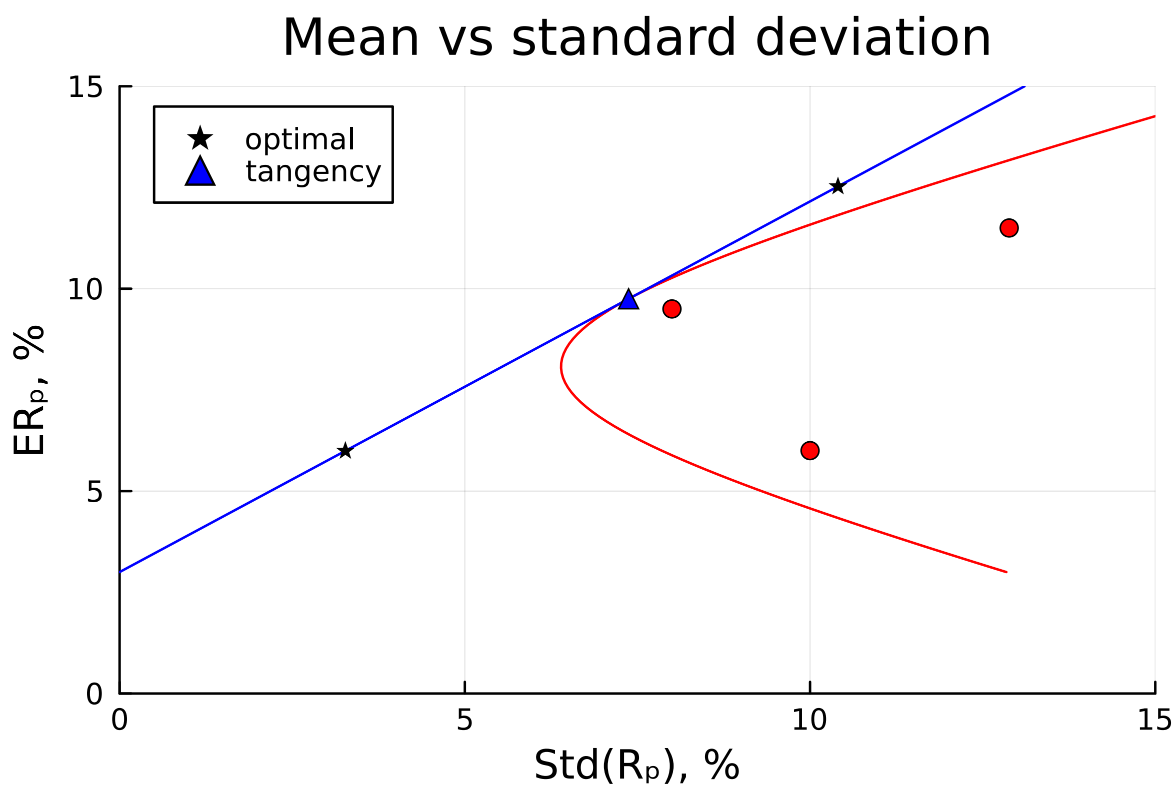
```
(vD,muD,StdD) = OptimalPortfolio(μ,Σ,Rf,28)        #high risk aversion
(vE,muE,StdE) = OptimalPortfolio(μ,Σ,Rf,8.8)       #low risk aversion

(wT,muT,StdT) = MVTangencyP(μ,Σ,Rf)                #tangency portfolio

assetNames = ["Asset A","Asset B","Asset C","Riskfree"]

printblue("optimal portfolio weights:")
printmat([vD;(1-sum(vD))],[vE;(1-sum(vE))];colNames=["D (high k)","E (low k)"],rowNames=assetN

printblue("optimal weights/tangency portfolio:")
printmat(vD./wT,vE./wT;colNames=["D (high k)","E (low k)"],rowNames=assetNames)
```

```
optimal portfolio weights:
        D (high k) E (low k)
Asset A      0.110     0.350
Asset B      0.302     0.962
Asset C      0.031     0.099
Riskfree     0.556    -0.411
```

```
optimal weights/tangency portfolio:
        D (high k) E (low k)
Asset A      0.444     1.411
Asset B      0.444     1.411
Asset C      0.444     1.411
```

```
L = 101
μstar_range = range(Rf,0.15,length=L)

StdRₚ = fill(NaN,L,2)                #loop over required average returns (μstar)
for i in 1:L
    StdRₚ[i,1] = MVCalc(  μstar_range[i],μ,Σ)[1]         #risky only
    StdRₚ[i,2] = MVCalcRf(μstar_range[i],μ,Σ,Rf)[1]      #risky and riskfree
end
```

```
p1 = plot( StdRₚ*100,μstar_range*100,
           label = "",
           linecolor = [:red :blue],
           xlim = (0,15),
           ylim = (0,15),
           title = "Mean vs standard deviation",
           xlabel = "Std(Rₚ), %",
           ylabel = "ERₚ, %" )
scatter!(sqrt.(diag(Σ))*100,μ*100,color=:red,label="")
scatter!([StdD,StdE]*100,[muD,muE]*100,color=:black,marker=:star,label="optimal")
scatter!([StdT]*100,[muT]*100,color=:blue,marker=:utriangle,label="tangency",legend=:topleft)
display(p1)
```



Interpreting the First Order Conditions (extra)

The first order conditions for optimal portfolio choice are

$\mu^e = k\Sigma v$

Notice that Σv is a vector of covariances of each asset with the portfolio v.

The next cell shows what happens to the first order condition at/off the optimal choice of v.

```
v, = OptimalPortfolio(μₐ,Σₐ,Rf,k)         #find optimal portfolio

printblue("Checking the first order conditions at optimal v:")
printmat(v,μₐ.-Rfₐ,k*Σₐ*v;colNames=["v","μᵉ","kΣv"],rowNames=["Asset 1","Asset 2"])

v   = [0.10,vOpt[2]]             #lower v₁, same v₂

printblue("Checking the first order conditions at another v (lower v₁, same v₂):")
printmat(v,μₐ.-Rfₐ,k*Σₐ*v;colNames=["v","μᵉ","kΣv"],rowNames=["Asset 1","Asset 2"])
```

```
printmagenta("the difference between μᵉ and kΣv suggests that we should increase v₁")
```

```
Checking the first order conditions at optimal v:
                v        μᵉ       kΣv
Asset 1     0.273     0.055     0.055
Asset 2     0.462     0.035     0.035

Checking the first order conditions at another v (lower v₁, same v₂):
                v        μᵉ       kΣv
Asset 1     0.100     0.055     0.029
Asset 2     0.462     0.035     0.030

the difference between μᵉ and kΣv suggests that we should increase v₁
```

Maximising the Sharpe Ratio (extra)

The cells below defines the Sharpe ratio of a portfolio. To normalise the portfolio, the weights on the n risky assets a forced to sum to one (by setting the weight on asset n to be $1-\sum_{i=1}^{n-1} w_i$). The optimization is over the $n-1$ first weights. (The need for a normalisation stems from the fact that any portfolio on the CML is a solution - and that they are all scalings of each other.)

The Optim.jl package is used for the optimisation.

```
using Optim
```

```
"""
    SRFn(w,μ,Σ,Rf)

Calculate the Sharpe ratio of a portfolio based on the portfolio weights `v=[w;1-sum(w)]`, the
expected asset returns `μ`, their covariance matrix `Σ` and the riskfree rate `Rf`.
"""
function SRFn(w,μ,Σ,Rf)
    v     = [w;1-sum(w)]   #last asset has the weight 1-sum(w): this makes sum(v)=1
    ERp   = v'μ
    StdRp = sqrt(v'Σ*v)
    SRp   = (ERp-Rf)/StdRp
    return SRp
end
```

```
SRFn
```

```julia
Sol = optimize(w -> -SRFn(w,μ,Σ,Rf),zeros(2))  #maximise SR -> minimize -SR

v = Optim.minimizer(Sol)
v = [v;1-sum(v)]

printblue("Portfolio of risky assets that maximizes the Sharpe ratio:")
printmat([wT v];colNames=["tangency","optimal"],rowNames=assetNames)
```

```
Portfolio of risky assets that maximizes the Sharpe ratio:
          tangency   optimal
Asset A      0.248     0.248
Asset B      0.682     0.682
Asset C      0.070     0.070
```

CAPM

This notebook summarizes the implications of CAPM and then tests them by OLS.

Load Packages and Extra Functions

```
using Printf, DelimitedFiles, Statistics

include("src/printmat.jl")
include("src/OlsGMFn.jl");            #contains a function for OLS
```

```
using Plots, LaTeXStrings
default(size = (480,320),fmt = :png)
```

The Theoretical Predictions of CAPM

The following section illustrates the theoretical predictions of CAPM by taking the following steps:

1. define a set of investable assets
2. find the tangency portfolio
3. calculate the betas of each asset against the tangency portfolio
4. check whether the average returns are in accordance with CAPM: $\mathrm{E}R_i = R_f + \beta_i(\mu_T - R_f)$ or equivalently $\mathrm{E}R_i^e = \beta_i\mu_T^e$, where superscript ^{e} indicates an excess return.

Characteristics of Three Assets: Means, Variances and Covariances

```
μ = [0.115, 0.095, 0.06]     #expected returns
Σ = [166  34  58;            #covariance matrix
      34  64   4;
      58   4 100]/100^2
Rf = 0.03

assetNames = ["A","B","C"]

printblue("expected returns, in %:")
printmat(μ*100;rowNames=assetNames)

printblue("covariance matrix, in bp:")
printmat(Σ*10_000;colNames=assetNames,rowNames=assetNames)
```

```
expected returns, in %:
A    11.500
B     9.500
C     6.000

covariance matrix, in bp:
          A         B         C
A   166.000    34.000    58.000
B    34.000    64.000     4.000
C    58.000     4.000   100.000
```

The Tangency Portfolio

The file included below contains, among other things, the function `MVTangencyP()` from the chapter on MV analysis. It calcuates the tangency portfolio.

```
include("src/MvCalculations.jl");
```

```
(wT,μT,σT) = MVTangencyP(μ,Σ,Rf)          #tangency portfolio weights and implied mean return
printblue("Tangency portfolio weights:")
printmat(wT;rowNames=assetNames)

printblue("mean and std of tangency portfolio:")
printmat([μT,σT];rowNames=["μT","σT"])
```

```
Tangency portfolio weights:
A     0.248
B     0.682
C     0.070

mean and std of tangency portfolio:
μT     0.097
σT     0.074
```

(Theoretical)   of the Assets

The tangency portfolio is a portfolio of the investable assets ($R_T = w_T'R$). It is therefore straightforward to calculate the covariance of each of the assets with R_T. The β values are then obtained by dividing the covariances with the variance of R_T.

Details: The covariance of R_i and R_T is $\sigma_{iT} = w_i'\Sigma w_T$ where w_i is the vector of portfolio weights that creates return R_i. Then, calculate $\beta_i = \sigma_{iT}/\sigma_T^2$.

Since w_i is a trivial vector (1 in position i and zeros elsewhere), we could (here) equally well calculate the vector of all betas by $\Sigma w_T/\sigma_T^2$. (Alternatively, in a loop do `w=zeros(n); w[i]=1; β[i] = w'*Σ*wT/σT^2`.)

```
β = Σ*wT/σT^2

printblue("β of the assets:")
printmat(β,rowNames=assetNames)
```

```
β of the assets:
A     1.259
B     0.963
C     0.444
```

Trying CAPM on the Assets

CAPMS says that

$\mathrm{E}R_i = R_f + \beta_i(\mu_T - R_f)$ or equivalently $\mathrm{E}R_i^e = \beta_i\mu_T^e$, where superscript ^{e} indicates an excess return.

Notice that there is no intercept in the second expression.

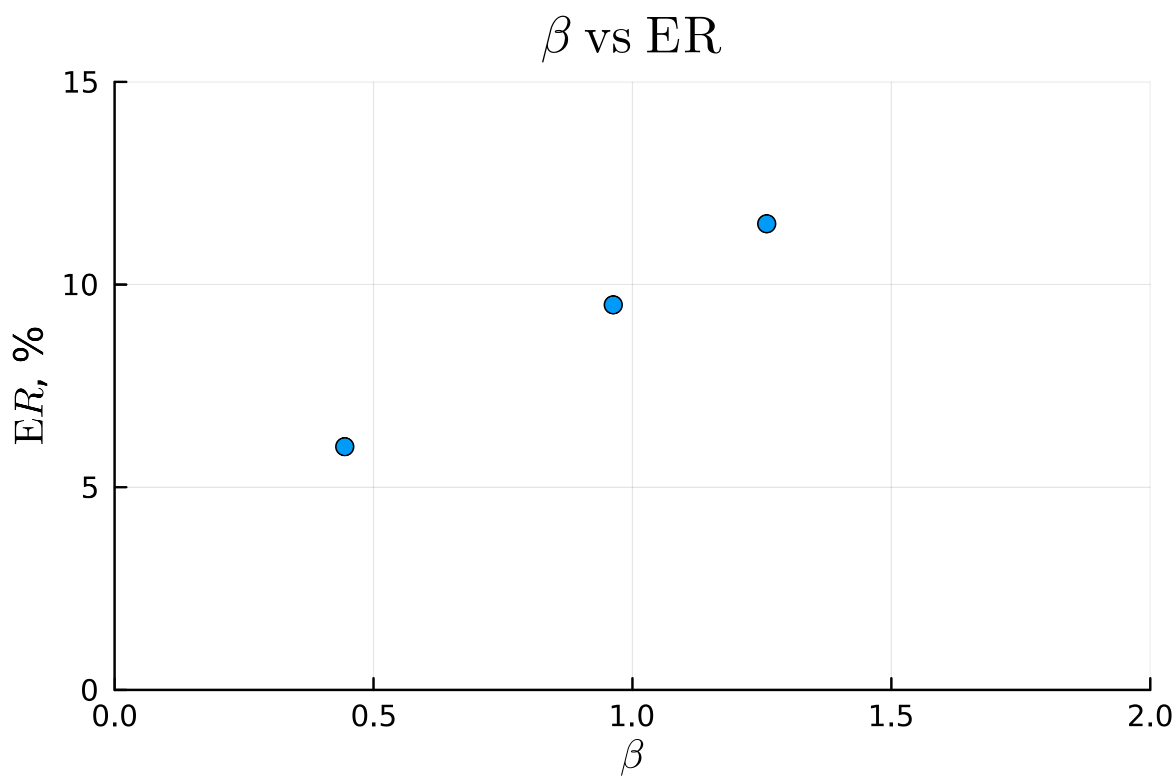
This can be compared with the actual average returns (or average excess returns for the second expression).

If we have done the previous calculations correctly, then CAPM should hold for assets A,B and C).

```
ER_CAPM = Rf .+ β*(μT-Rf)      #ER_CAPM is a vector since β is a vector

printblue("μ and ER as suggested by CAPM: ")
printmat([μ ER_CAPM];rowNames=assetNames,colNames=["actual","from CAPM"])

printblue("μᵉ and ERᵉ as suggested by CAPM (subtract Rf=$Rf from the previous results): ")
printmat([(μ.-Rf) (ER_CAPM.-Rf)];rowNames=assetNames,colNames=["actual","from CAPM"])
```

```
μ and ER as suggested by CAPM:
     actual from CAPM
A     0.115     0.115
B     0.095     0.095
C     0.060     0.060

μᵉ and ERᵉ as suggested by CAPM (subtract Rf=0.03 from the previous results):
     actual from CAPM
A     0.085     0.085
B     0.065     0.065
C     0.030     0.030
```

```
p1 = scatter( β,ER_CAPM*100,      #points on the security market line
              xlim = (0,2),
              ylim = (0,15),
              legend = false,
              title = L"\beta \mathrm{\ vs \ ER}",
              xlabel = L"\beta",
              ylabel = L"$\mathrm{E}R$, %" )
display(p1)
```



An Empirical Test of CAPM

The next section performs an empirical test of CAPM. First, we load data. Then, we run linear regressions and test whether the intercepts are zero (the CAPM prediction) or not. (We test each asset separately.)

Loading Data

We load data from two data files: for the returns on Fama-French equity factors and then also for the 25 Fama-French portfolios. To keep the output simple, we use only 5 of those return series.

```
x    = readdlm("Data/FFmFactorsPs.csv",',',skipstart=1)
Rme  = x[:,2]                   #market excess return
RSMB = x[:,3]                   #small minus big firms
RHML = x[:,4]                   #high minus low book-to-market ratio
Rf   = x[:,5]                   #interest rate
```

```
x  = readdlm("Data/FF25Ps.csv",',') #no header line: x is matrix
R  = x[:,2:end]                     #returns for 25 FF portfolios
Re = R .- Rf                        #excess returns for the 25 FF portfolios
Re = Re[:,[1;7;13;19;25]]           #use just 5 assets to make the printing easier

(T,n) = size(Re)                    #no. obs and  no. test assets
```

```
(388, 5)
```

OLS Estimation and Testing $= 0$

We now use the market return as a proxy for the tangency portfolio - and test if CAPM holds for the test assets.

To do that, we estimate (α_i, b_i) in the CAPM regression

$R^e_{it} = \alpha_i + \beta_i R^e_{mt} + \varepsilon_{it}$,

where R^e_{it} and R^e_{mt} are excess return, and then test if $\alpha_i = 0$. This corresponds to the CAPM prediction that $\mathrm{E}R^e_i = \beta_i \mu^e_m$. Compared with the theoretical expression above, we use the market excess return as a measure of the excess return of the tangency portfolio.

A Remark on the Code

- The function `OlsGMFn` is included in the file `OlsGMFn.jl` (see the first cell of the notebook). It does OLS estimation and reports the point estimates, residuals and more. The variance-covariance matrix of the coefficients is based on the Gauss-Markov assumptions.
- In the call on `printmat()`, `;colNames` (notice: semicolon) is the same as `,colNames=colNames`.

```
x = [ones(T) Rme]                   #regressors

(α,tstat) = (fill(NaN,n),fill(NaN,n))
for i in 1:n                              #loop over the different test assets
    #local b_i, residual, Covb           #local/global is needed in script
    (b_i,_,_,Covb,) = OlsGMFn(Re[:,i],x)
    α[i]            = b_i[1]                      #estimated α
    tstat[i]        = (α[i]-0)/sqrt(Covb[1,1])    #t-stat of H₀: true α=0
end

printblue("OLS intercepts and t-stats:")
```

```
colNames = [string("asset ",i) for i=1:n]
rowNames = ["α","t-stat"]
printmat([α';tstat'];colNames,rowNames)
```

```
OLS intercepts and t-stats:
          asset 1   asset 2   asset 3   asset 4   asset 5
α          -0.504     0.153     0.305     0.279     0.336
t-stat     -1.656     1.031     2.471     2.163     2.073
```

Risk Measures

This notebook illustrates the *Value at Risk* (VaR) and *Expected Shortfall*. To improve the performance, a model for time-varying risk is estimated and incorporated into the calculations. The model is backtested on a data set.

We use the the Distributions.jl package to calculate probability values and quantiles.

Load Packages and Extra Functions

```
using Printf, Dates, Distributions, DelimitedFiles

include("src/printmat.jl");
```

```
using Plots
default(size = (480,320),fmt = :png)
```

Value at Risk (VaR) for a N(, ^{2}) Return

$\text{VaR}_{95\%} = -(5^{th}$ percentile of the return distribution)

With a $N(\mu, \sigma^2)$ distribution this gives approximately

$\text{VaR}_{95\%} = -(\mu - 1.64\sigma)$

A Remark on the Code

- The Distributions.jl package defines a normal distribution as `Normal(μ,σ)`. Notice that it uses the standard deviation, not the variance. For instance, to calculate the 5th quantile, use `quantile(Normal(μ,σ),0.05)` and to calculate the pdf value at each element of a vector `x`, use `pdf.(Normal(μ,σ),x)`.

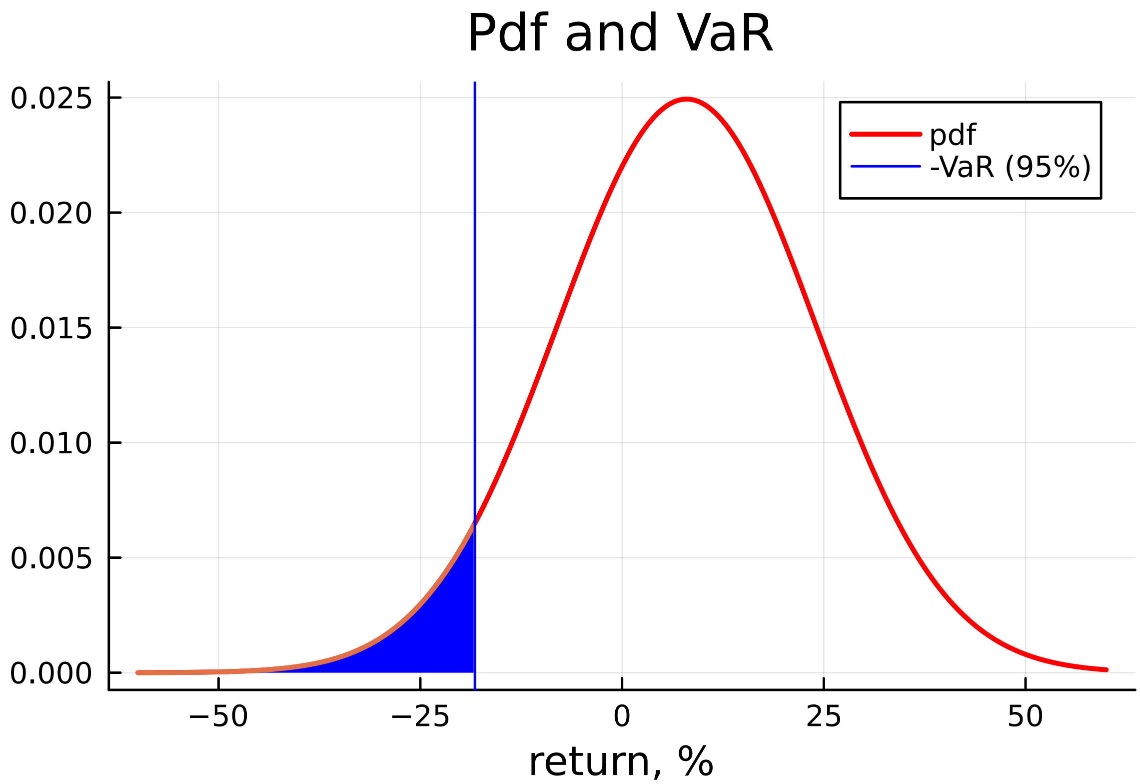
```
(μ,σ) = (8,16)

VaR₉₅ = -(μ - 1.64*σ)

printblue("with μ=$μ and σ=$σ, we have approximately:\n")
printmat([μ - 1.64*σ,VaR₉₅];rowNames=["5th quantile","VaR 95%"])
```

```
with μ=8 and σ=16, we have approximately:

5th quantile   -18.240
VaR 95%         18.240
```

```
#exact calculation of quantile 0.05, notice: σ
printlnPs("get an exact result by using the quantile() function: ",-quantile(Normal(μ,σ),0.05)
```

```
get an exact result by using the quantile() function:     18.318
```

```
R    = range(-60,60,length=301)
pdfR = pdf.(Normal(μ,σ),R)
Rb   = R[R .<= -VaR₉₅]             #or filter(<=(-VaR₉₅),R)

p1 = plot( R,pdfR,
           linecolor = :red,
           linewidth = 2,
           label = "pdf",
           title = "Pdf and VaR",
           xlabel = "return, %" )
plot!(Rb,pdf.(Normal(μ,σ),Rb),fillcolor=:red,linewidth=2,fill=(0,:blue),label="")
vline!([-VaR₉₅],linecolor=:blue,label="-VaR (95%)")
display(p1)
```



Loading Daily S&P 500 Data

```
x  = readdlm("Data/SP500RfPs.csv",',',skipstart=1)
SP = x[:,2]                                #S&P 500 level
R  = (SP[2:end]./SP[1:end-1] .- 1)*100     #returns, %
T  = length(R)

dN = Date.(x[2:end,1],"d/m/y");     #convert to Date

println("Days in the sample: $T")
```

```
Days in the sample: 9352
```

Backtesting a Static VaR from N() on Data

To backtest a VaR model, we study the relative frequency of loss > VaR.

Backtesting for the Full Sample

The code below does this for different confidence levels (0.95,0.96,...) of the VaR. For instance, at the 0.95 confidence levels we 1. calculate the VaR as the (negative of the) 0.05 quantile of a normal distribution (with the mean and std estimated from the sample, which is just one possible choice) 2. count the relative frequency of the loss > VaR (it should be 0.05).

```
μemp = mean(R)                             #mean and std of data (empirical)
σemp = std(R)

confLev = 0.95:0.005:0.995               #different confidence levels
L       = length(confLev)
loss    = -R

(VaR_L,BreakFreq_L) = (fill(NaN,L),fill(NaN,L))   #for different confidence levels
for i in 1:L                      #loop over confidence levels
    VaR_L[i]       = -quantile(Normal(μemp,σemp),1-confLev[i])
    BreakFreq_L[i] = mean(loss .> VaR_L[i]) #freq of breaking the VaR
end

printblue("Backtesting a static VaR:\n")
colNames = ["conf level","N()-based VaR","break freq"]
printmat([confLev VaR_L BreakFreq_L];colNames,width=18)

printred("The break frequency should be 1-confidence level")
```

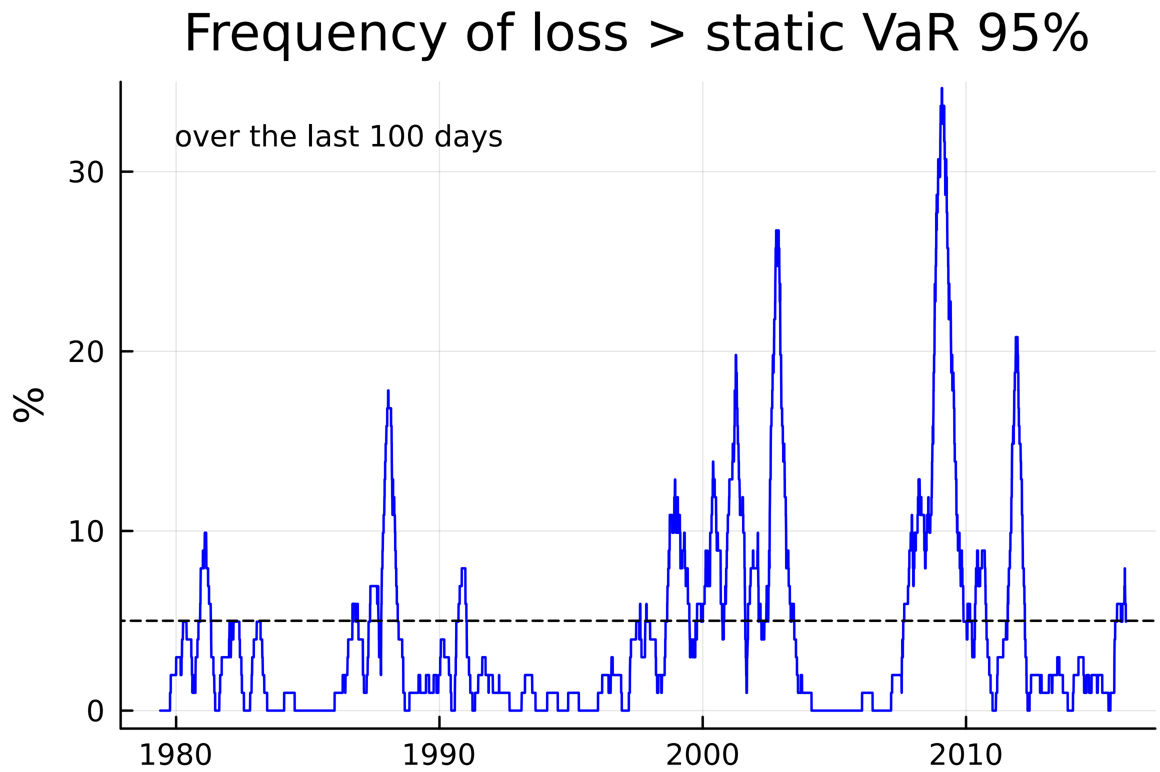
```
Backtesting a static VaR:

        conf level     N()-based VaR        break freq
             0.950             1.790             0.041
             0.955             1.846             0.037
             0.960             1.907             0.034
             0.965             1.975             0.032
             0.970             2.052             0.029
             0.975             2.140             0.026
             0.980             2.244             0.023
             0.985             2.373             0.019
             0.990             2.547             0.016
             0.995             2.824             0.012

The break frequency should be 1-confidence level
```

Backtesting on a Moving Data Window

The code below also studies the relative frequency of loss > VaR, but over a *moving data window* of 100 days (and only for a single confidence level, 0.95). This allows us to investigate if there are long periods of failures of the VaR model.

```
VaR = -quantile(Normal(μₑₘₚ,σₑₘₚ),0.05)  #static VAR at 95% confidence level

BreakFreq = fill(NaN,T)                #vector, freq(loss>VaR) on moving data window
for t in 101:T                         #loop over end-points of the data window
    BreakFreq[t] = mean(loss[t-100:t] .> VaR)
end
```

```
xTicksLoc = [Date(1980);Date(1990);Date(2000);Date(2010)]
xTicksLab = Dates.format.(xTicksLoc,"Y")

p1 = plot( dN,BreakFreq*100,
           linecolor = :blue,
           ylim = (-1,35),
           legend = false,
           xticks = (xTicksLoc,xTicksLab),
           title = "Frequency of loss > static VaR 95%",
           ylabel = "%",
           annotation = (Date(1980),32,text("over the last 100 days",8,:left)) )
hline!([5],linecolor=:black,line=(:dash,1))
display(p1)
```



A Simple Dynamic VaR with Time-Varying Volatility

To improve the performance of the VaR model, we combine a simple time series model for volatility with an N()-based VaR calculation.

We first construct an simple estimate of σ_t^2 and μ_t as a backward looking exponential moving average (cf. RiskMetrics)

$\sigma_t^2 = \lambda\sigma_{t-1}^2 + (1-\lambda)(R_{t-1} - \mu_{t-1})^2$, where $\mu_t = \lambda\mu_{t-1} + (1-\lambda)R_{t-1}$,

where we use $0.9 \leq \lambda < 1$.

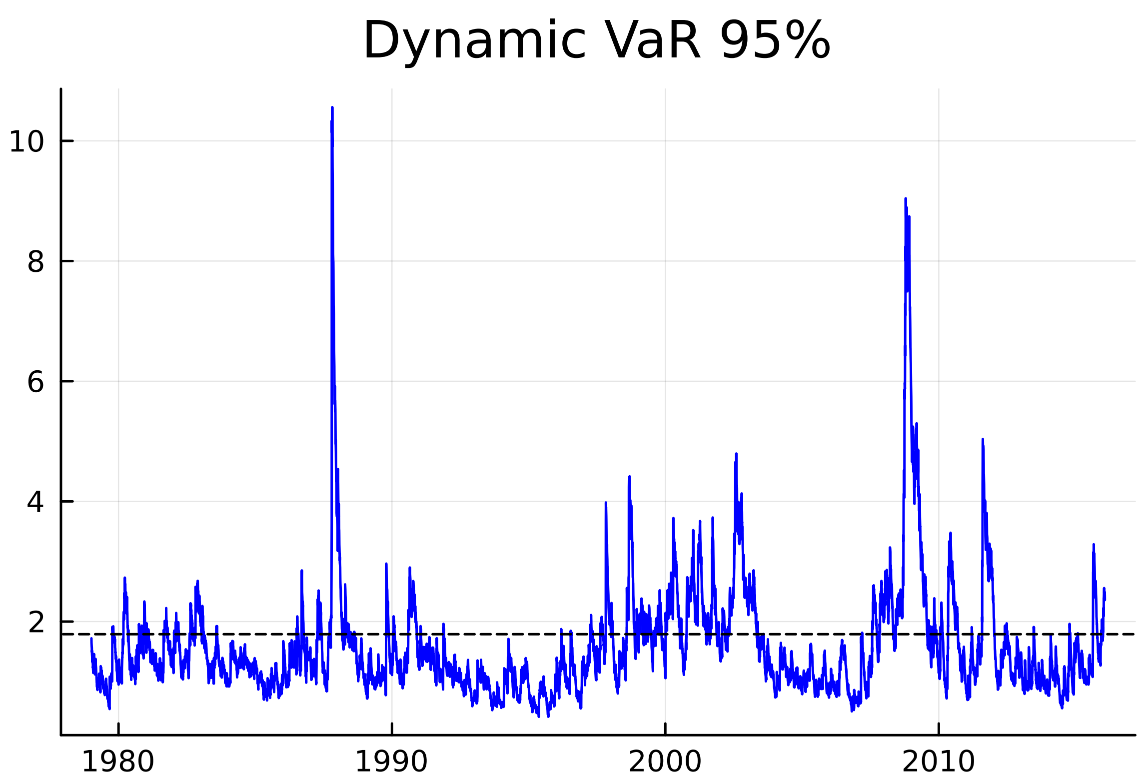
We then redo the $\text{VaR}_{95\%}$ calculation using

$\text{VaR}_t = -(\mu_t - 1.64\sigma_t)$

and study if it has better properties than the static VaR.

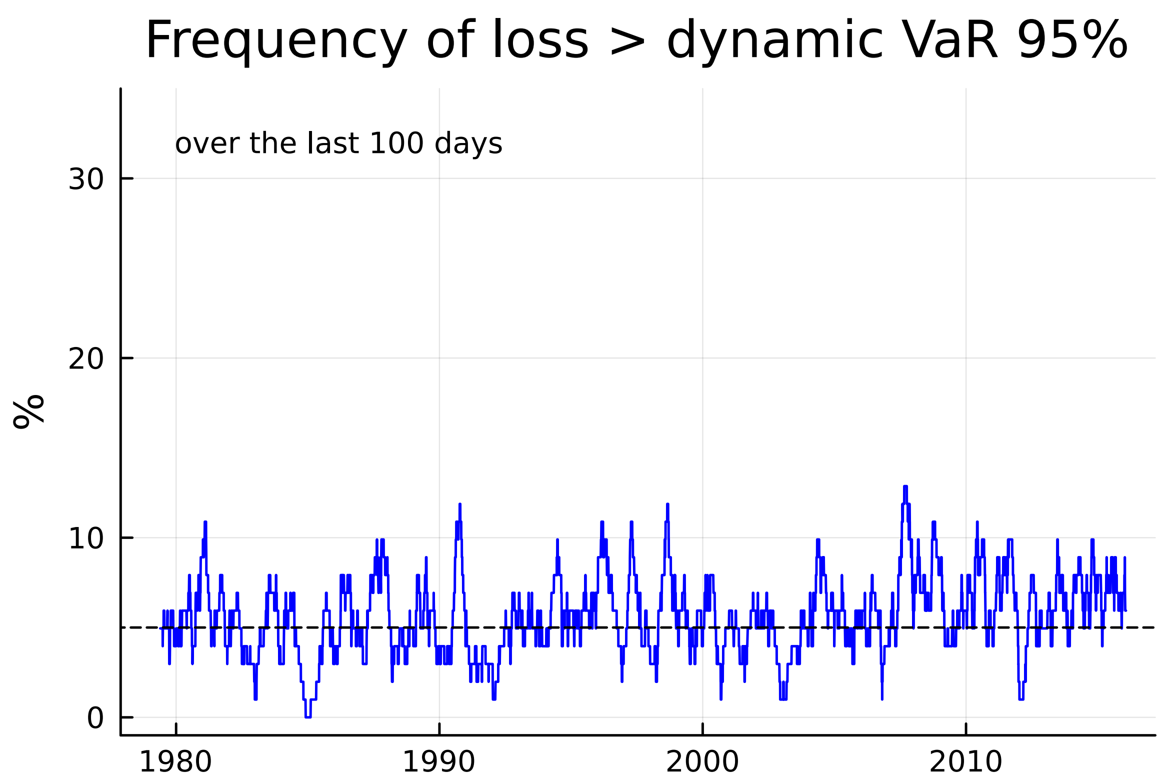
```
λ   = 0.94
(μ,σ²,VaRd) = (fill(μₑₘₚ,T),fill(σₑₘₚ^2,T),fill(NaN,T))
for t in 2:T
    μ[t]    = λ*μ[t-1]  + (1-λ)*R[t-1]
    σ²[t]   = λ*σ²[t-1] + (1-λ)*(R[t-1]-μ[t-1])^2
    VaRd[t] = -quantile(Normal(μ[t],sqrt(σ²[t])),0.05)  #dynamic VaR
end
```

```
p1 = plot( dN,VaRd,
           linecolor = :blue,
           #ylim = (-1,35),
           legend = false,
           xticks = (xTicksLoc,xTicksLab),
           title = "Dynamic VaR 95%",
           ylabel= "",
           annotation = (Date(1980),32,text("over the last 100 days",8,:left)) )
hline!([VaR],linecolor=:black,line=(:dash,1))
display(p1)
```



```
BreakFreq = fill(NaN,T)          #freq(loss>VaR) on moving data window
for t in 101:T
    BreakFreq[t] = mean(loss[t-100:t] .> VaRd[t-100:t])
end
```

```
p1 = plot( dN,BreakFreq*100,
           linecolor = :blue,
           ylim = (-1,35),
           legend = false,
           xticks = (xTicksLoc,xTicksLab),
           title = "Frequency of loss > dynamic VaR 95%",
           ylabel= "%",
           annotation = (Date(1980),32,text("over the last 100 days",8,:left)) )
hline!([5],linecolor=:black,line=(:dash,1))
display(p1)
```



Expected Shortfall

The Expected Shortfall (ES) is the average loss, conditional on exceeding the VaR level. In terms of the returns (not losses) this is

$\mathrm{ES}_\alpha = -\mathrm{E}(R|R \leq -\mathrm{VaR}_\alpha)$

For a normally distributed return $R \sim N(\mu, \sigma^2)$ we have

$\mathrm{ES}_{95\%} = -\left(\mu - \frac{\phi(-1.64)}{0.05}\sigma\right),$

where $\phi()$ is the pdf of a $N(0,1)$ variable. For other confidence levels, change -1.64 and 0.05.

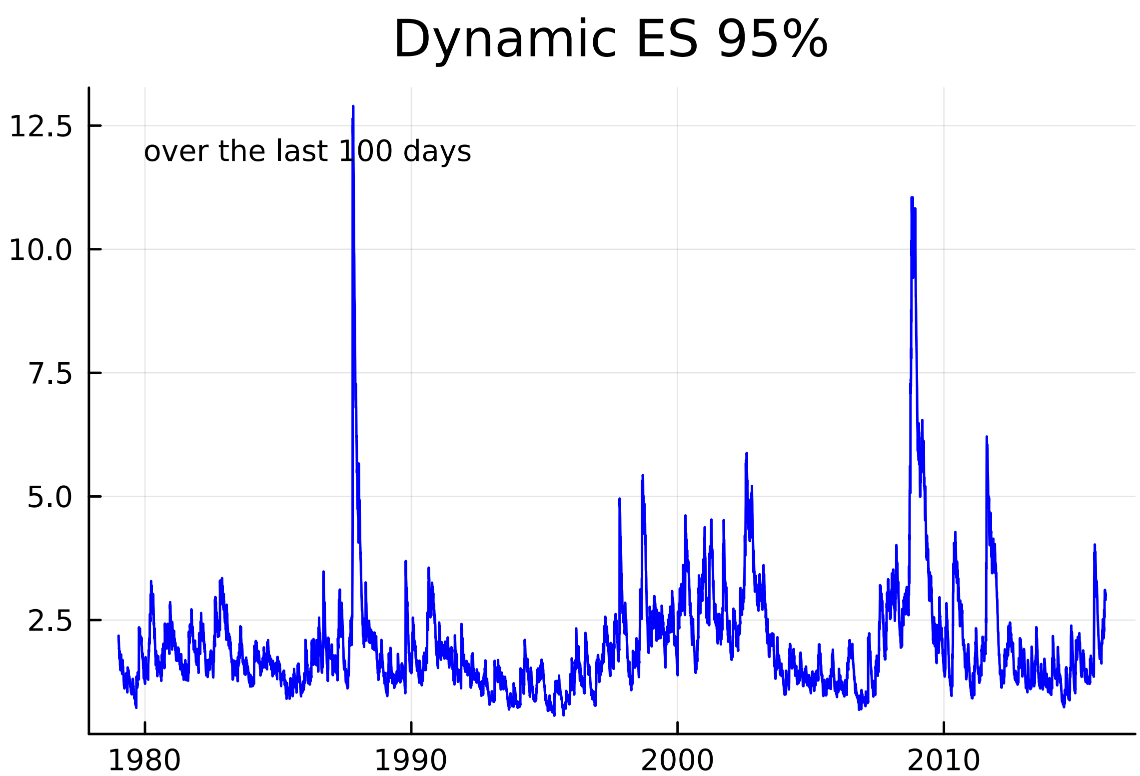
Alternatively, we can calculate the ES as `-mean(truncated(Normal(μ,σ);upper=q))` where `q` is the quantile of the return distribution.

```
(μb,σb) = (8,16)

ES = -(μb - pdf(Normal(0,1),-1.64)/0.05*σb)

qb  = quantile(Normal(μb,σb),0.05)
ESb = -mean(truncated(Normal(μb,σb);upper=qb))        #more exact calculation

printblue("ES 95% with μ=$μb and σ=$σb is:\n")
printmat([ES,ESb];rowNames=["approx","exact"])
```

```
ES 95% with μ=8 and σ=16 is:

approx    25.268
exact     25.003
```

```
(VaRd,ESd) = (fill(NaN,T),fill(NaN,T))          #calculate a dynamic ES at the 95% conf level
for t = 2:T
    #local q_n, ER_n        #only needed in script
    q_n  = quantile(Normal(μ[t],sqrt(σ²[t])),0.05)
    ER_n = mean(truncated(Normal(μ[t],sqrt(σ²[t]));upper=q_n))
    VaRd[t] = -q_n
    ESd[t]  = -ER_n
end
```

```
p1 = plot( dN,ESd,
           linecolor = :blue,
           legend = false,
           xticks = (xTicksLoc,xTicksLab),
           title = "Dynamic ES 95%",
           ylabel= "",
           annotation = (Date(1980),12,text("over the last 100 days",8,:left)) )
display(p1)
```



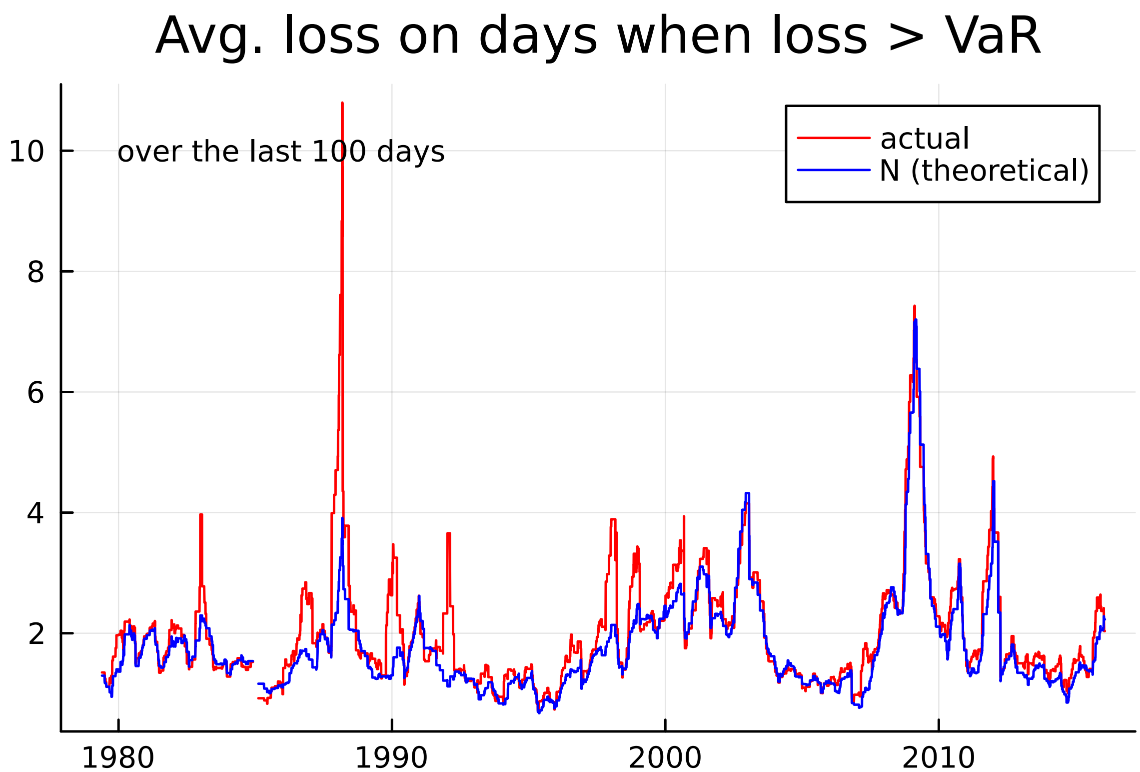
Backtesting ES

by comparing the empirical average return (conditional on loss > VaR) with the predictions from the model with $\mathrm{N}(\mu_t, \sigma_t^2)$.

```
ES_backtest = fill(NaN,T,2)         #backtesting ES
for t in 101:T
  vv_t             = loss[t-100:t] .> VaRd[t-100:t]
  ES_backtest[t,1] = mean(loss[t-100:t][vv_t])          #mean empirical loss|loss>VaR
```

```
  ES_backtest[t,2] = mean(ESd[t-100:t][vv_t])          #mean N()-theoretical loss|loss>VaR
end
```

```
p1 = plot( dN,ES_backtest,
           linecolor = [:red :blue],
           label = ["actual" "N (theoretical)"],
           xticks = (xTicksLoc,xTicksLab),
           title = "Avg. loss on days when loss > VaR",
           ylabel= "",
           annotation = (Date(1980),10,text("over the last 100 days",8,:left)) )
display(p1)
```



Utility Theory

This notebook summarizes basic utility theory and how it can be used in portfolio optimisation.

Load Packages and Extra Functions

Optim.jl is an optimization package.

```
using Printf, Optim

include("src/printmat.jl");
```

```
using Plots
default(size = (480,320),fmt = :png)
```

Utility Functions

Several of the examples will use the CRRA utility function, which is $U(x) = \frac{x^{1-\gamma}}{1-\gamma}$, where γ is the relative risk aversion.

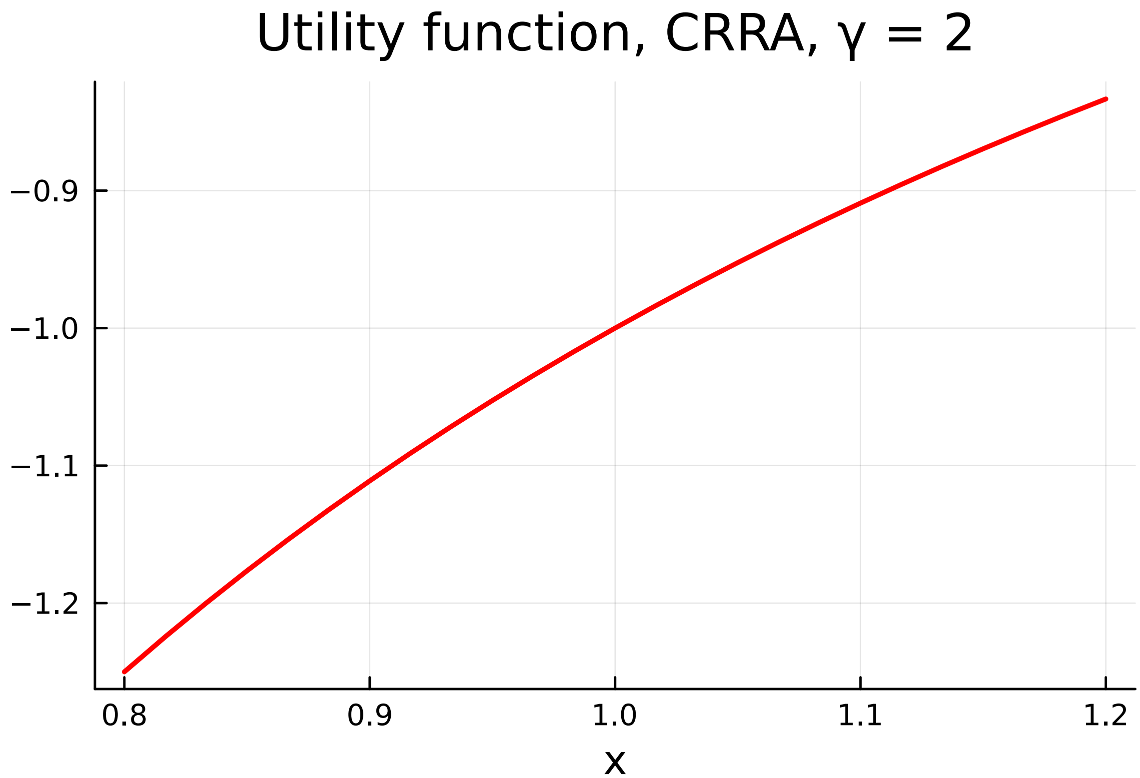
A Remark on the Code

1. A Julia function can be created in several ways. For a simple one-liner, it is enough to do like this: `MySum(x,y) = x + y`
2. If the function `U(x,γ)` is written for a scalar `x` value, then we can calculate its value at each element of an array or range `x_range` by using `U.(x_range,γ)`.

```
"""
    U(x,γ)

CRRA utility function, γ is the risk aversion.
"""
U(x,γ) = x^(1-γ)/(1-γ)



"""
    U_1(u,γ)

Inverse of CRRA utility function. Solves for x st. U(x,γ) = u.
"""
U_1(u,γ) = (u*(1-γ))^(1/(1-γ))
```

U_1

```
x_range = range(0.8,1.2,length=25)            #possible outcomes
γ = 2

p1 = plot( x_range,U.(x_range,γ),            #notice the dot(.)
           linecolor = :red,
           linewidth = 2,
           legend = false,
           title = "Utility function, CRRA, γ = $γ",
           xlabel = "x" )
display(p1)
```



Expected Utility

Recall: if π_s is the probability of outcome ("state") x_s and there are S possible outcomes, then the expected value is

$\mathrm{E}x = \sum_{s=1}^{S} \pi_s x_s$.

Similarly, the expected utility is

$\mathrm{E}U(x) = \sum_{s=1}^{S} \pi_s U(x_s)$

A Remark on the Code

When `x` is a vector, then `U.(x,γ)` is a vector of the corresponding utility values. Therefore, `E(π,U.(x,γ))` sums `π[i]*U(x[i],γ)` across all `i`.

```
"""
    E(π,z)

Calculate the expected value from vector of outcomes `z` and a vector of their probabilities `
"""
E(π,z) = sum(π.*z)           #alternatively, dot(π,z) or π'z
```

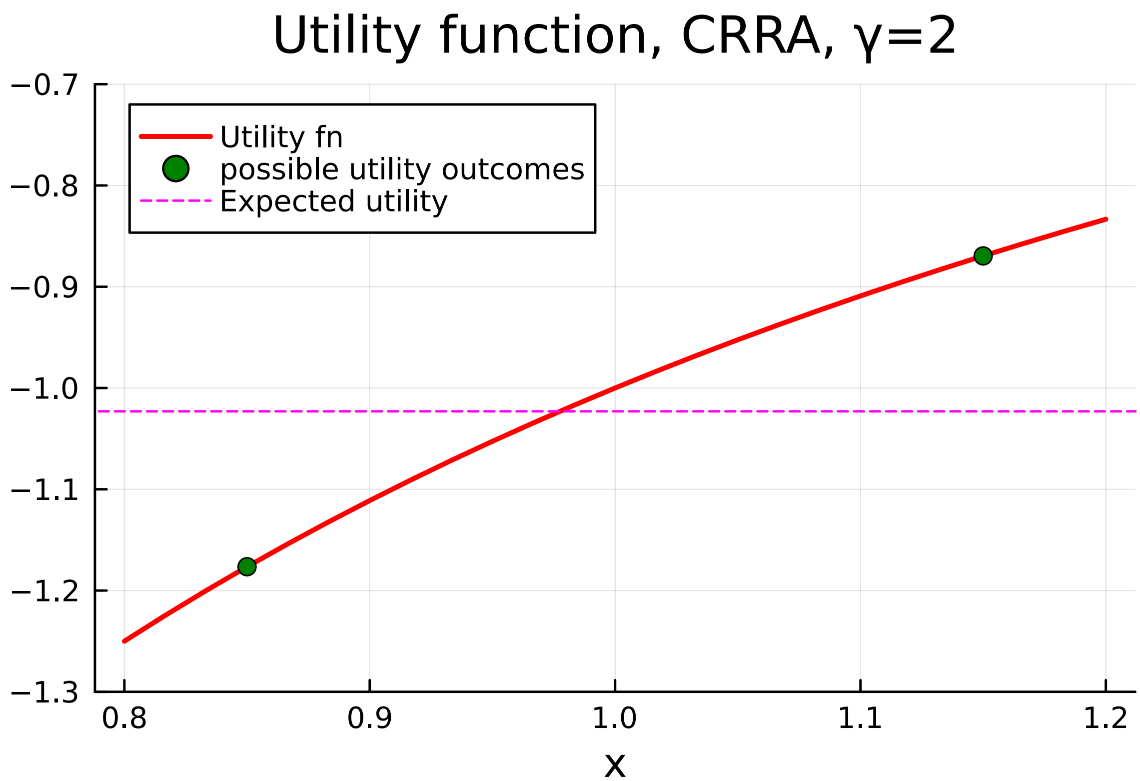
```
E
```

```
γ = 2                   #risk aversion

(x₁,x₂) = (0.85,1.15)             #possible outcomes
(π₁,π₂) = (0.5,0.5)               #probabilities of outcomes

state₁ = [x₁,U(x₁,γ),π₁]          #for printing
state₂ = [x₂,U(x₂,γ),π₂]

printblue("Different states: wealth, utility and probability:\n")
printmat([state₁ state₂];colNames=["state 1","state 2"],rowNames=["wealth","utility","probabil

Ex      = E([π₁,π₂],[x₁,x₂])         #expected wealth
ExpUtil = E([π₁,π₂],U.([x₁,x₂],γ))   #expected utility

printmat([Ex,ExpUtil];rowNames=["Expected wealth","Expected utility"])
```

```
Different states: wealth, utility and probability:

              state 1   state 2
wealth          0.850     1.150
utility        -1.176    -0.870
probability     0.500     0.500

Expected wealth      1.000
Expected utility    -1.023
```

```
p1 = plot( x_range,U.(x_range,γ),
           linecolor = :red,
           linewidth = 2,
           label = "Utility fn",
           ylim = (-1.3,-0.7),
```

```
            legend = :topleft,
            title = "Utility function, CRRA, γ=$γ",
            xlabel = "x" )
scatter!([x₁,x₂],U.([x₁,x₂],γ),markercolor=:green,label="possible utility outcomes")
hline!([ExpUtil],linecolor=:magenta,line=(:dash,1),label="Expected utility")
display(p1)
```



Certainty Equivalent

The certainty equivalent (here denoted P) is the sure value that solves

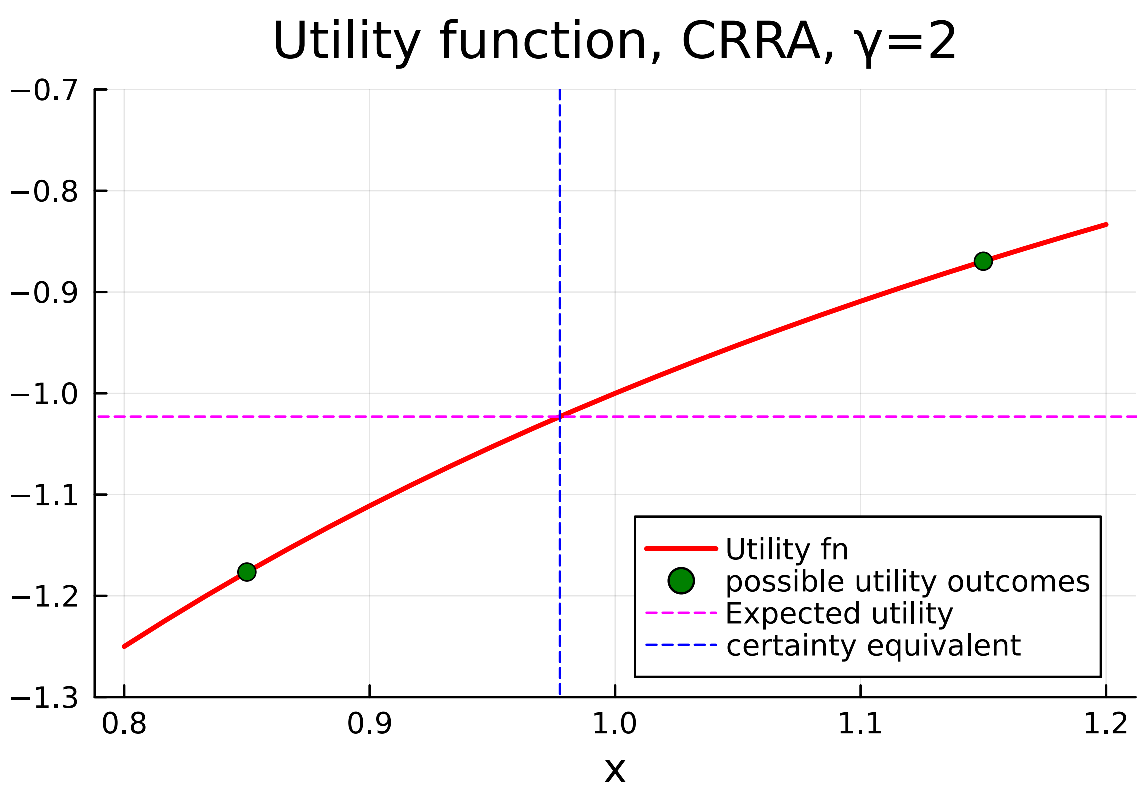
$U(P) = \mathrm{E}U(x)$,

where the right hand side is the expected utility from the random x. P is thus the highest price the investor is willing to pay for "asset" x.

The code below solves for P by inverting the utility function, first for the same risk aversion (γ) as above, and later for different values of γ.

We can think of $\mathrm{E}(x)/P - 1$ as the expected return on x that the investor requires in order to buy the asset. It is increasing in the risk aversion γ.

```
EU_i = E([π₁,π₂],U.([x₁,x₂],γ))  #expected utility
P    = U_1(EU_i,γ)               #certainty equivalent (inverting the utility fn)

p1 = plot( x_range,U.(x_range,γ),
           linecolor = :red,
           linewidth = 2,
           label = "Utility fn",
           ylim = (-1.3,-0.7),
           legend = :bottomright,
           title = "Utility function, CRRA, γ=$γ",
           xlabel = "x" )
scatter!([x₁,x₂],U.([x₁,x₂],γ),markercolor=:green,label="possible utility outcomes")
hline!([ExpUtil],linecolor=:magenta,line=(:dash,1),label="Expected utility")
vline!([P],linecolor=:blue,line=(:dash,1),label="certainty equivalent")
display(p1)
```



```
γM = [0,2,5,10,25,50,100]                    #different risk aversions
L  = length(γM)

(P,ERx) = (fill(NaN,L),fill(NaN,L))
for i in 1:L                                  #loop over γ values
    #local EU_i                                #local/global is needed in script
    EU_i   = E([π₁,π₂],U.([x₁,x₂],γM[i]))      #expected utility with γM[i]
    P[i]   = U_1(EU_i,γM[i])                   #inverting the utility fn
    ERx[i] = Ex/P[i] - 1                       #required expected net return
end

printblue("risk aversion and certainly equivalent (recall: E(wealth) = $Ex):\n")
printmat(γM,P,ERx;colNames= ["γ","certainty eq","expected return"],width=20)
```

```
risk aversion and certainly equivalent (recall: E(wealth) = 1.0):

                   γ        certainty eq     expected return
               0               1.000               0.000
               2               0.977               0.023
               5               0.947               0.056
              10               0.912               0.097
              25               0.875               0.143
              50               0.862               0.160
             100               0.856               0.168
```

Utility-Based Portfolio Choice with One Risky Asset

In the example below, the investor maximizes $\mathrm{E}\ln(1+R_p)$, with $R_p = vR^e + R_f$ by choosing v (the weight on the only risky asset). For simplicity, there are only two possible outcomes for R^e with equal probabilities.

This particular problem can be solved by pen and paper, but this becomes very difficult when the number of states increases - and even worse when there are many assets. To prepare for these tricker cases, we apply a numerical optimization algorithm already to this simple problem.

Remark on the Code

To solve the optimization problem we use `optimize()` from the Optim.jl package. The key steps are:

1. Define a function for expected utility, `EUlog(v,π,Rᵉ,Rf)`. The value depends on the portfolio choice `v`, as well as the properties of the asset (probabilities and returns for different states, the vectors `π` and `Rᵉ`) and the riskfree return (`Rf`).

2. To create data for the plot, we loop over `v[i]` values and calculate expected utility as `EUlog(v[i],π,Rᵉ,Rf)`. (Warning: you can assign a value to π provided you have not used the built-in constant π (3.14156...) first.)

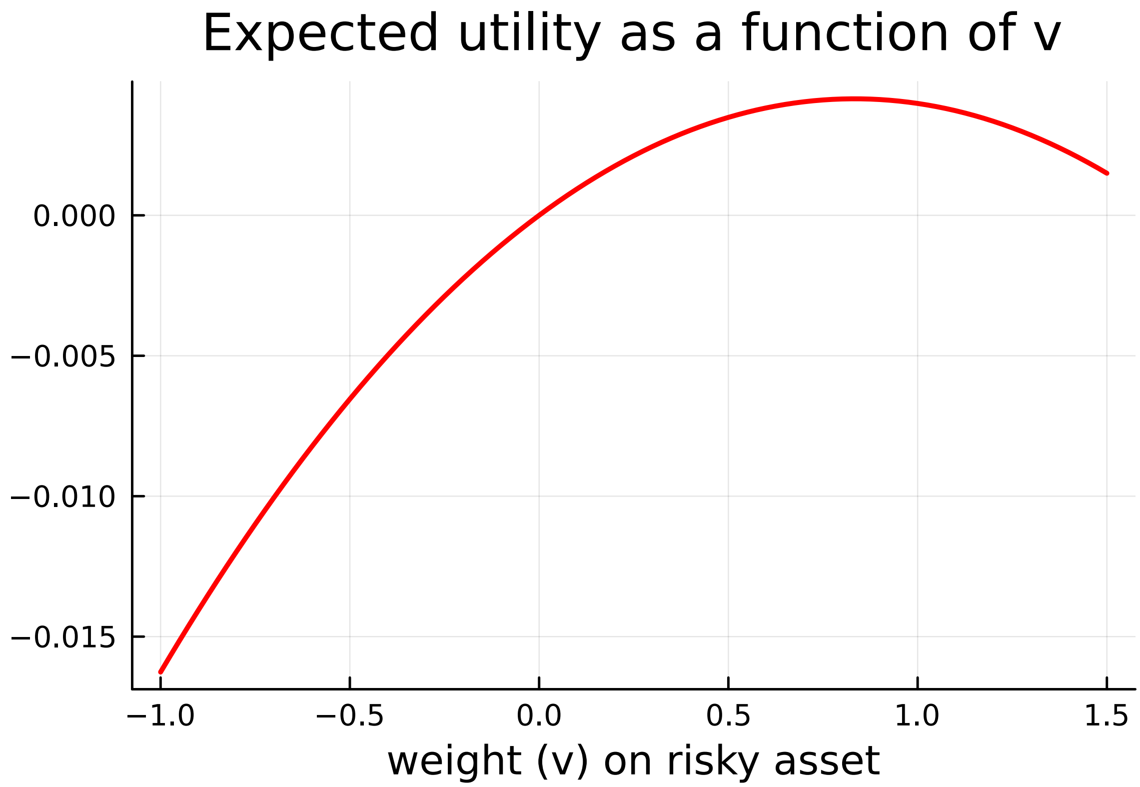
3. For the optimization, we minimize the anonymous function `v->-EUlog(v,π,Rᵉ,Rf)`. This is a function of `v` only and we multiply by -1 since `optimize()` is a *minimization* routine.

```
"""
    EUlog(v,π,Rᵉ,Rf)

Calculate expected utility (log(1+Rp)) from investing into one risky and one riskfree asset

v:  scalar
π:  S vector probabilities of the different states
Rᵉ: S vector, excess returns of the risky asset in different states
Rf: scalar, riskfree rate

"""
function EUlog(v,π,Rᵉ,Rf)          #expected utility, utility fn is logarithmic
    Rₚ = v*Rᵉ .+ Rf                #portfolio return
    eu = E(π,log.(1 .+ Rₚ))        #expected utility
    return eu
end
```

```
EUlog
```

```
π  = [0.5,0.5]                  #probabilities for different states
Rᵉ = [-0.10,0.12]               #excess returns in different states
Rf = 0                          #riskfree rate

v_range = range(-1,1.5,length=101)    #try different weights on risky asset
L       = length(v_range)
EUv = fill(NaN,L)
for i in 1:L                                 #loop over v values
    EUv[i] = EUlog(v_range[i],π,Rᵉ,Rf)
end

p1 = plot( v_range,EUv,
           linecolor = :red,
```

```
            linewidth = 2,
            legend = false,
            title = "Expected utility as a function of v",
            xlabel = "weight (v) on risky asset" )
display(p1)
```




```
Sol = optimize(v->-EUlog(v,π,Rᵉ,Rf),-1,1)  #minimize -EUlog
printlnPs("Optimum at: ",Optim.minimizer(Sol))

printred("\nCompare with the figure")
```

```
Optimum at:      0.833

Compare with the figure
```

Portfolio Choice with Several Risky Assets

This optimization problem has several risky assets and states and a CRRA utility function. Numerical optimization is still straightforward.

A Remark on the Code

- The code below is fairly similar to the log utility case solved before, but extended to handle CRRA utility and several assets and states.
- With several choice variables, the call to `optimize()` requires a vector of starting guesses as input.

```
"""
    EUcrra(v,π,Rᵉ,Rf,γ)

Calculate expected utility from investing into n risky assets and one riskfree asset

v:  n vector (weights on the n risky assets)
π:  S vector (S possible "states")
Rᵉ: nxS matrix, each column is the n vector of excess returns in one of the states
Rf: scalar, riskfree rate
γ:  scalar, risk aversion
"""
function EUcrra(v,π,Rᵉ,Rf,γ)
    S  = length(π)
    Rp = fill(NaN,S)
    for s in 1:S           #loop over states, Rp in state s
        Rp[s] = v'Rᵉ[:,s] + Rf
    end                     #same as Rp = Rᵉ'v .+ Rf
    eu = E(π,U.(1 .+ Rp,γ))  #expected utility when using portfolio v
    return eu
end
```

```
EUcrra
```

```
R  = [-0.03 0.08 0.20;           #2 assets, 3 states
      -0.04 0.22 0.15]           #R[i,j] is the return of asset i in state j
π = [1/3,1/3,1/3]                #probs of the states
Rf = 0.065
Rᵉ = R .- Rf                     #excess returns in different states
γ  = 5
```

```
Sol = optimize(v->-EUcrra(v,π,Rᵉ,Rf,γ),[-0.6,1.2])     #minimize -EUcrra
v   = Optim.minimizer(Sol)                              #extract the solution

printblue("optimal portfolio weights from max EUcrra():\n")
printmat([v;1-sum(v)];rowNames=["asset 1","asset 2","riskfree"])
```

```
optimal portfolio weights from max EUcrra():

asset 1     -0.726
asset 2      1.317
riskfree     0.409
```

Mean-Variance and the Telser Criterion

Let μ be a vector of expected returns of the investible assets and Σ be their covariance matrix.

The Telser criterion solves a problem of the following type:

$\max_v \mathrm{E}R_p$ subject to $\mathrm{VaR}_{95\%} < 0.1$, where $\mathrm{E}R_p = v'\mu^e + R_f$.

If the returns are normally distributed then

$\mathrm{VaR}_{95\%} = -[\mathrm{E}R_p - 1.64\mathrm{Std}(R_p)]$,

It follows that the VaR restriction can be written

$\mathrm{E}R_p > -0.1 + 1.64\mathrm{Std}(R_p)$.

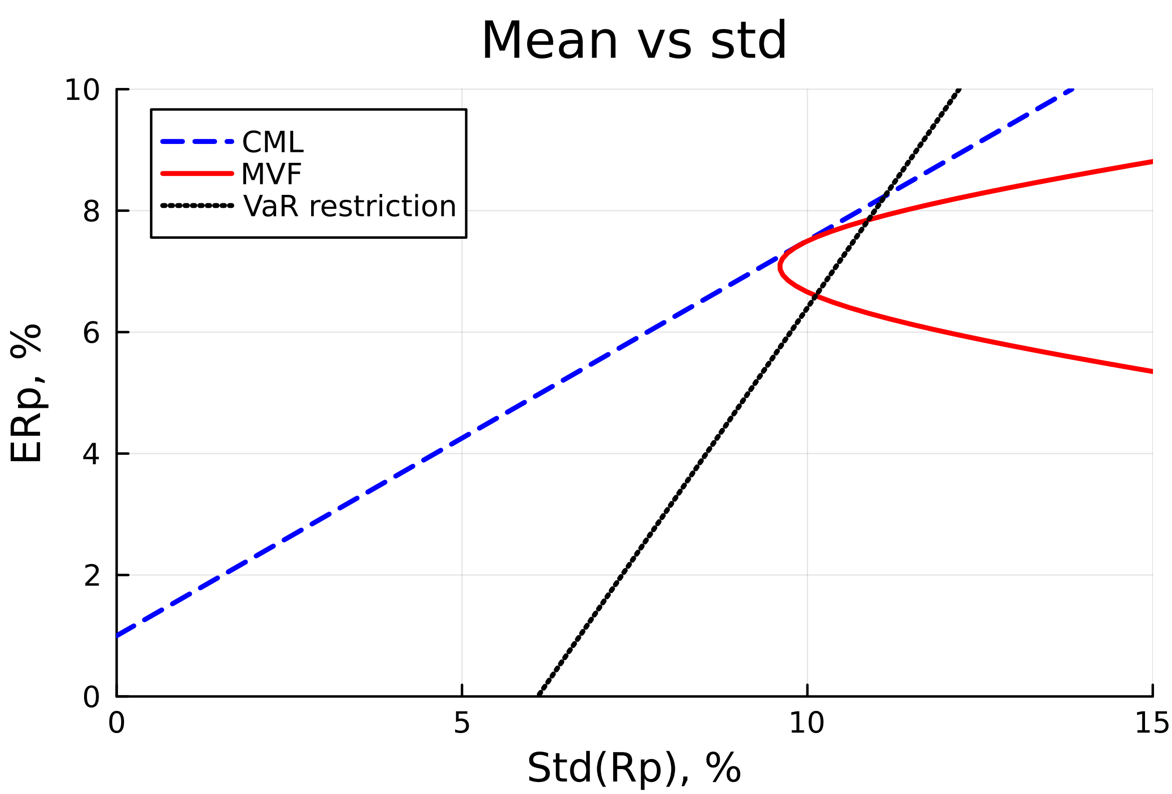
The figure below illustrates that the optimal portfolio is *on the CML* when the returns are normally distributed.

```
include("src/MvCalculations.jl");    #functions for traditional MV frontiers
```

```
μ = [9, 6]/100                          #mean returns of investable assets
Σ = [ 256    0;                         #covariance matrix
        0  144]/10000
Rf = 1/100

μˣ_range = range(Rf,0.1,length=101)      #required average returns
L        = length(μˣ_range)

(σMVF,σCML) = (fill(NaN,L),fill(NaN,L))
```

```
for i in 1:L                                          #loop over different required returns (μˣ)
    σMVF[i] = MVCalc(μˣ_range[i],μ,Σ)[1]          #std of MVF (risky only) at μˣ
    σCML[i] = MVCalcRf(μˣ_range[i],μ,Σ,Rf)[1]     #std of MVF (risky&riskfree) at μˣ
end

VaRRestr = -0.1 .+ 1.64*σCML;     #the portfolio mean return must be above this
```

```
p1 = plot( [σCML σMVF σCML]*100,[μˣ_range μˣ_range VaRRestr]*100,
           linestyle = [:dash :solid :dot],
           linecolor = [:blue :red :black],
           linewidth = 2,
           label = ["CML" "MVF" "VaR restriction"],
           xlim = (0,15),
           ylim = (0,10),
           legend = :topleft,
           title = "Mean vs std",
           xlabel = "Std(Rp), %",
           ylabel = "ERp, %" )
display(p1)
```



Multi-Factor Models

The first part of this notebook summarizes a simple economic model that leads to a multi-factor representation of the expected returns. The second part tests a multi-factor model by OLS.

Load Packages and Extra Functions

```
using Printf, DelimitedFiles, Statistics, LinearAlgebra

include("src/printmat.jl")
include("src/OlsGMFn.jl");                #funtcion for doing OLS
```

```
using Plots, LaTeXStrings
default(size = (480,320),fmt = :png)
```

Portfolio Choice with Background Risk

The investor maximizes $\mathrm{E}R_p - \frac{k}{2}\mathrm{Var}(R_p)$, where

$R_p = (1-\phi)R_{Fin} + \phi R_c$ with $R_{Fin} = w'R + (1-\mathbf{1}'w)R_f$.

In this definition of the portfolio return, ϕ is the fraction of total wealth that is bound to non-traded "assets": this is the background risk.

Notice that $\mathrm{E}R_p = v'\mu^e + \phi\mu_c^e + R_f$, where $v = w(1-\phi)$ and where μ^e is the vector of excess returns on the risky assets and μ_c^e is the excess return on the background risk. Also, $\mathrm{Var}(R_p) = v'\Sigma v + \phi^2\sigma_{cc} + 2\phi v'S_c$, where S_c is a vector of covariances of the risky assets with the background risk, and σ_{cc} is the variance of the background risk.

The next cell defined functions for expected utility and the optimal portfolio choice.

A Remark on the Code

The code uses the subscript h (as in S_h etc) instead of c. The reason is that the Julia unicode symbols do not include a subscripted c.

```
"""
    EU

Expected utility for the case with background risk.

The notation follows the text above, except that the subscript h (as in Sₕ) for the non-traded
"""
function EU(v,ϕ,k,μᵉ,Σ,Sₕ,σₕₕ,μᵉₕ,Rf)
    ERp   = v'μᵉ + ϕ*μᵉₕ + Rf
    VarRp = v'*Σ*v + ϕ^2*σₕₕ + 2*ϕ*v'*Sₕ
    EUtil = ERp - k/2*VarRp
    return EUtil
end

"""
    PortFOpt

Solve for the optimal weight on risky asset (v), ϕ in non-traded asset (background risk), 1-v-

The notation follows the text above.
"""
function PortFOpt(ϕ,k,μᵉ,Σ,Sₕ)
    vopt  = inv(Σ)*(μᵉ/k-ϕ*Sₕ)
    wopt  = vopt/(1-ϕ)          #weight on risky in financial subportfolio
    return vopt, wopt
end
```

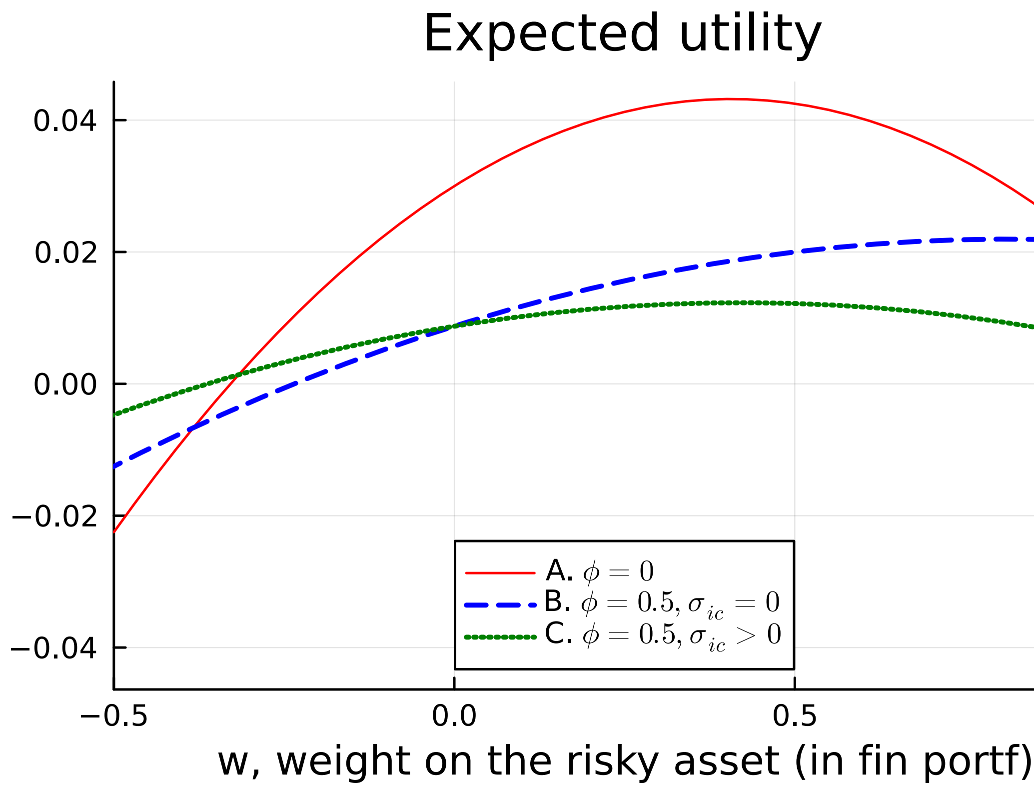
```
PortFOpt
```

One Risky Asset

We first consider the case when there is only one risky asset (v is a scalar). We will illustrate several cases (A, B, C) which differ with respect to the fraction of non-traded asset (denoted ϕ) and its covariance with the risk assets (denoted S_c as in the lecture notes or S_h as in the code).

```
Σ    = 0.08^2                 #covariance (matrix) of risky assets, here only one
Sₕ_A = 0                      #covariance(s) of risky with background, case A
Sₕ_C = 0.0025                 #case C
(μᵉ,k,σₕₕ,μᵉₕ,Rf) = (0.065,25,0.01,0.02,0.03)

v_range = range(-0.5,1,length=51)  #weight on risky investable asset
L       = length(v_range)

                                     #expected utility at different v values
(EU_case_A,EU_case_B,EU_case_C) = (fill(NaN,L),fill(NaN,L),fill(NaN,L))
for i in 1:L
    EU_case_A[i] = EU(v_range[i],0,k,μᵉ,Σ,Sₕ_A,σₕₕ,μᵉₕ,Rf)      #ϕ=0
    EU_case_B[i] = EU(v_range[i],0.5,k,μᵉ,Σ,Sₕ_A,σₕₕ,μᵉₕ,Rf)    #ϕ>0, Sₕ=0
    EU_case_C[i] = EU(v_range[i],0.5,k,μᵉ,Σ,Sₕ_C,σₕₕ,μᵉₕ,Rf)    #ϕ>0, Sₕ>0
end
```

```
p1 = plot( [v_range v_range/(1-0.5) v_range/(1-0.5)],[EU_case_A EU_case_B EU_case_C],
           linecolor = [:red :blue :green],
           linestyle = [:solid :dash :dot],
           linewidth = [1 2 2],
           label = [L"A. $\phi=0$" L"B. $\phi=0.5, \sigma_{ic}=0 $" L"C. $\phi=0.5, \sigma_{ic
           xlims = (-0.5,1),
           legend = :bottom,
           title = "Expected utility",
           xlabel = "w, weight on the risky asset (in fin portf)" )
display(p1)
```




```
(v_A,w_A) = PortFOpt(0,k,μᵉ,Σ,Sₕ_A)          #optimal portfolio choice
(v_B,w_B) = PortFOpt(0.5,k,μᵉ,Σ,Sₕ_A)
(v_C,w_C) = PortFOpt(0.5,k,μᵉ,Σ,Sₕ_C)

printblue("\nOptimal weight on (a single) risky asset in three cases (ϕ >= 0):\n")
xx = [v_A w_A;v_B w_B;v_C w_C]
colNames = ["in total portf","in financial portf"]
rowNames = ["A. ϕ=0","B. ϕ=0.5, σᵢₕ=0","C. ϕ=0.5, σᵢₕ>0"]
printmat(xx;colNames,rowNames,width=20)

printred("\nCompare with the plot")
```

```
Optimal weight on (a single) risky asset in three cases (ϕ >= 0):

                     in total portf  in financial portf
A. ϕ=0                        0.406               0.406
B. ϕ=0.5, σᵢₕ=0               0.406               0.812
C. ϕ=0.5, σᵢₕ>0               0.211               0.422
```

```
Compare with the plot
```

Several Risky Assets

We now consider several risky assets. Notice that S_c (or S_h) is now a vector of covariances of each of the investable risky assets with the non-traded asset.

A Remark on the Code

- The S_h vector is here created in such a way that we prespecify the correlations (and then scale with the product of the standard deviations of the non-traded assets and the risky assets).

```
μ = [11.5, 9.5, 6]/100          #expected returns
Σ = [166  34  58;               #covariance matrix
      34  64   4;
      58   4 100]/100^2
Rf = 0.03

ϕ   = 0.3                            #fraction of non-traded asset
σₕₕ = 0.25^2
Sₕ  = [0.5,0.9,-0.1].*sqrt(σₕₕ).*sqrt.(diag(Σ))  #a vector of covariances
μₕ  = 0.1
k   = 8

(v,w) = PortFOpt(ϕ,k,μ.-Rf,Σ,Sₕ)

printblue("optimal weights (inside the financial subportfolio):\n")
printmat([w;1-sum(w)],rowNames=["A","B","C","Rf"])
```

```
optimal weights (inside the financial subportfolio):

A      0.235
B      0.453
C      0.488
Rf    -0.176
```

Asset Pricing Implications of a Multifactor Model

If several factors affect the portfolio choice, then they will (in equilibrium) also affect prices and returns. In fact, the expected excess return on asset i is

$\mathrm{E}R_i^e = \beta_i' \mu_F^e$,

where β_i is the vector of regressions coefficients from regressing R_i^e on the factors and μ_F^e is the vector of expected excess returns on those factors.

The example below assumes there are two factors and we only consider the pricing of a single investable asset. The numbers used have nothing in particular to do with those used in the previous examples. (That link could be done, but requires some intermdiate steps that we here skip.)

Remark

A vector of regression slopes (y regressed on the vector x) can be calculated as

$\beta = \Sigma_{xx}^{-1} S_{xy}$,

where Σ_{xx} is the variance-covariance matrix of x and S_{xy} is a vector of covariance of each x with y.

```
μᵉF = [0.08,0.06]            #avg. excess returns of factors

Ψ = [1 0.5;                  #covariance matrix of the factors
     0.5 2]
Sᵢ = [0.5,0.7]               #Cov(Ri,factor1) and Cov(Ri,factor2)

β = inv(Ψ)*Sᵢ                #coeffs in ERᵉ = β'Factors
printlnPs("The multiple regression coefficients: ")
printmat(β)

μᵉ = β'μᵉF
printlnPs("μᵉ for asset i according to 2-factor model:")
printmat(μᵉ)
```

```
The multiple regression coefficients:
     0.371
     0.257

μᵉ for asset i according to 2-factor model:
     0.045
```

Empirical Test of a 3-Factor Model: Loading Data

Load Data

```
x    = readdlm("Data/FFmFactorsPs.csv",',',skipstart=1)
Rme  = x[:,2]                #market excess return
RSMB = x[:,3]                #small minus big firms
RHML = x[:,4]                #high minus low book-to-market ratio
Rf   = x[:,5]                #interest rate


x  = readdlm("Data/FF25Ps.csv",',') #no header line: x is matrix
R  = x[:,2:end]                     #returns for 25 FF portfolios
Re = R .- Rf                        #excess returns for the 25 FF portfolios
Re = Re[:,[1,7,13,19,25]]           #use just 5 assets to make the printing easier

(T,n) = size(Re)                    #no. obs and  no. test assets
```

```
(388, 5)
```

OLS Estimation and Testing $= 0$

Recall: estimate (α_i, b_i) in the factor model

$R^e_{it} = \alpha_i + b_i' f_t + \varepsilon_{it}$,

where f_t is a vector of excess returns of the factors.

Test if $\alpha_i = 0$

```
x = [ones(T) Rme RSMB RHML]         #regressors

(α,tstat) = (fill(NaN,n),fill(NaN,n))
for i in 1:n                              #loop over the different test assets
    #local b_i,Covb                      #local/global is needed in script
    (b_i,_,_,Covb,_) = OlsGMFn(Re[:,i],x)
    α[i]       = b_i[1]
    tstat[i] = (α[i]-0)/sqrt(Covb[1,1])  #t-stat for α[i]=0?
end

printblue("Regression of Re on constant and 3 factors:\n")
```

```
colNames = [string("asset ",i) for i=1:n]
printmat([α';tstat'];colNames,rowNames=["α","t-stat"])
```

```
Regression of Re on constant and 3 factors:

          asset 1   asset 2   asset 3   asset 4   asset 5
α          -0.513    -0.006     0.030    -0.020    -0.015
t-stat     -2.306    -0.066     0.328    -0.206    -0.133
```

Efficient Markets

This notebook describes and tests the predictability of asset returns (autocorrelations, autoregressions, out-of-sample R2, Mariano-Diebold test) and implements a simple trading strategy.

Load Packages and Extra Functions

```
using Printf, Dates, DelimitedFiles, Statistics, LinearAlgebra, StatsBase

include("src/printmat.jl")
include("src/OlsGMFn.jl")   #OLS with covariance matrix etc
include("src/OlsM.jl");     #OLS, basic
```

```
using Plots
default(size = (480,320),fmt = :png)
```

Load Data

The data set contains daily data of the equity market return, riskfree rate and the returns of the 25 Fama-French portfolios. All returns are in percent.

```
x   = readdlm("Data/MomentumSR.csv",',')
dN  = Date.(x[:,1],"yyyy-mm-dd")                    #Julia dates
y   = convert.(Float64,x[:,2:end])

(Rm,Rf,R25) = (y[:,1],y[:,2],y[:,3:end])
R           = R25[:,end]                          #R are returns of 'large value' firms
                                                  #R25 will be used later
println("\nThe first few rows of data")
printmat(dN[1:4],Rm[1:4],Rf[1:4],R[1:4];colNames=["dN","Rm","Rf","R"])
```

```
println("size of dN, Rm, Rf, R")
println(size(dN),"\n",size(Rm),"\n",size(Rf),"\n",size(R))

T = length(R);                          #number of periods
```

```
The first few rows of data
        dN         Rm         Rf          R
1979-01-02     0.615      0.035      1.420
1979-01-03     1.155      0.035      1.750
1979-01-04     0.975      0.035      1.560
1979-01-05     0.685      0.035      1.430

size of dN, Rm, Rf, R
(9837,)
(9837,)
(9837,)
(9837,)
```

Autocorrelations

The sth autocorrelation is

$\rho_s = \text{Corr}(R_t, R_{t-s})$

In large samples, $\sqrt{T}\hat{\rho}_s \sim N(0,1)$ if the true value is $\rho_s = 0$ for all s.

A Remark on the Code

The StatsBase.jl package contains methods for estimating autocorrelations (see `autocor()` below).

```
plags = 1:5        #different lags to calculate autocorrelations for
ρ     = autocor(R,plags)

printblue("autocorrelations and their t-stats:\n")
printmat(ρ,sqrt(T)*ρ;colNames=["ρ","t-stat"],rowNames=plags,cell00="lag")
```

```
autocorrelations and their t-stats:

lag          ρ     t-stat
1       -0.027    -2.697
2       -0.028    -2.747
3       -0.024    -2.343
4       -0.014    -1.350
5       -0.020    -2.015
```

Autoregressions

An AR(1) is

$R_t = c + a_1 R_{t-1} + \varepsilon_t.$

We also consider an asymmetric AR(1)

$R_t = \alpha + \beta Q_{t-1} R_{t-1} + \gamma(1 - Q_{t-1}) R_{t-1} + \varepsilon_t,$

where $Q_{t-1} = 1$ if $R_{t-1} < 0$ and zero otherwise.

Both models can be estimated by OLS.

A Remark on the Code

- The function `OlsGMFn()` does OLS and reports, among other things, the variance-covariance matrix of the estimates (called `Covb` below).
- To get standard errors, we first extract the variances from the variance-covariance matrix (`diag(Covb)`) and then calculate the square roots (`sqrt.()`). Notice the dot. in the latter calculation: to calculate the square root of each element in the vector of variances.
- The t-stats are then just the point estimates (minus the null hypothesis, which is here 0) divided by the standard errors.

```
x  = [ones(T-1) R[1:end-1]]            #R(t) is regressed on (1,R(t-1))
(b,_,_,Covb,) = OlsGMFn(R[2:end],x)
Stdb   = sqrt.(diag(Covb))            #diag() picks out the diagonal of the variance-covarianc
tstat = b./Stdb

printblue("Results from an AR(1):\n")
printmat([b tstat];colNames=["coef","t-stat"],rowNames=["constant","slope"])
```

```
Results from an AR(1):

                 coef    t-stat
constant        0.061     4.314
slope          -0.027    -2.698
```

```
Q = R[1:end-1] .< 0          #dummy: 1 if R(t-1) < 0
x = [ones(T-1) Q.*R[1:end-1] (1.0.-Q).*R[1:end-1]]   #R(t) is regressed on (1,Q(t-1)R(t-1),[1-

(b,_,_,Covb,) = OlsGMFn(R[2:end],x)
Stdb  = sqrt.(diag(Covb))
tstat = b./Stdb

printblue("Results from an AR(1) with dummies:\n")
printmat([b tstat];colNames=["coef","t-stat"],rowNames=["constant","slope (down)","slope (up)"
```

```
Results from an AR(1) with dummies:

                     coef    t-stat
constant           -0.007    -0.385
slope (down)       -0.102    -6.134
slope (up)          0.048     2.872
```

Recursive Estimation and Out-of-Sample R2

The next cell does recursive estimation (longer and longer sample) and predicts one period ahead (outside of the sample). The performance of this prediction model is measured by an "out-of-sample" R^2_{OOS} defined as

$R^2_{OOS} = 1 - \frac{\text{MSE(forecasting model)}}{\text{MSE(benchmark forecast)}}$

A Remark on the Code

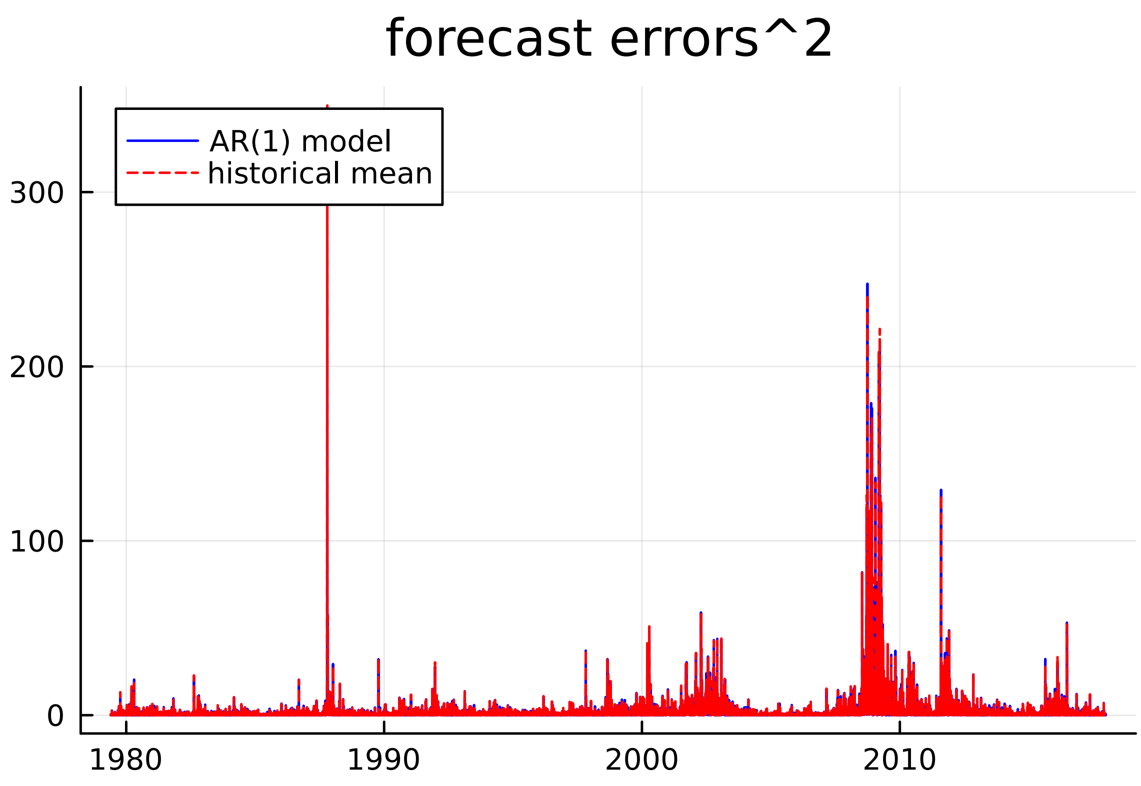
- The code loops over `t=100:Tb` where `Tb` is the effective length of the sample. Inside the loop we use data on `y[1:t-1]` and `x[1:t-1,:]`. This means that the first estimation uses the first 99 observations, the second estimation the first 100 observations and so forth. After the loop, we discard the first 100 observations from the output since they are just `NaN`.
- For estimation we use the basic OLS function `OlsM()`. Clearly, `OlsGMFn()` gives the same results (but is a tiny bit slower).

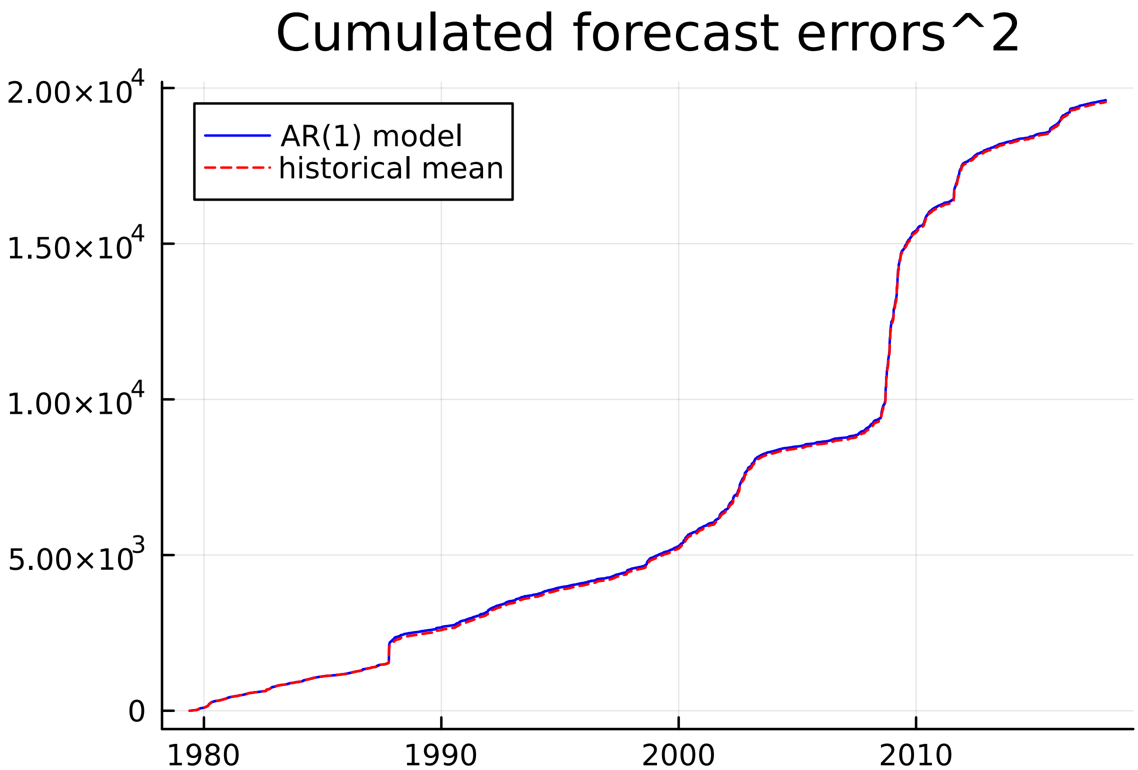
```
y   = R[2:end]      #traditional symbol for the LHS variable
dNb = dN[2:end]     #corresponding dates, used for plotting
Tb  = length(y)     #length of the effective sample

x   = [ones(Tb) Q.*R[1:end-1] (1.0.-Q).*R[1:end-1]]     #predictors

(ϵ,e) = (fill(NaN,Tb),fill(NaN,Tb))
for t in 100:Tb
    local b
    b,    = OlsM(y[1:t-1],x[1:t-1,:])    #estimate on sample 1:t-1
    ϵ[t] = y[t] - x[t,:]'b         #forecast error for t, model
    e[t] = y[t] - mean(y[1:t-1])   #forecast error for t, historical average
end
(ϵ,e,dNb) = (ϵ[100:end],e[100:end],dNb[100:end])    #drop periods without forecasts

MSE_Model = mean(abs2,ϵ)    #MSE for out-of-sample forecasts, same as mean(ϵ.^2) but quicker
MSE_Bench = mean(abs2,e)
R2oos     = 1 - MSE_Model/MSE_Bench

printblue("Performance of out-of-sample forecasting:\n")
printmat([MSE_Model;MSE_Bench;R2oos];rowNames=["MSE of AR(1)","MSE of hist avg","R2_oos"])
```

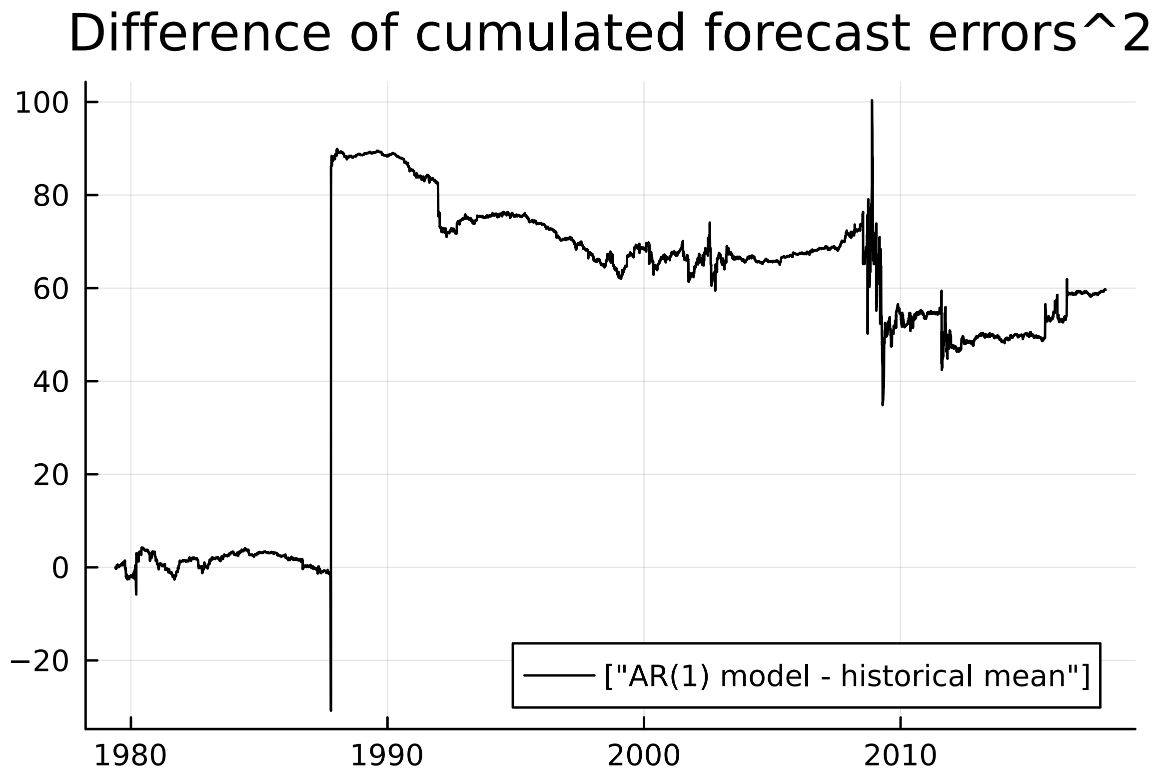
```
Performance of out-of-sample forecasting:

MSE of AR(1)        2.014
MSE of hist avg     2.008
R2_oos             -0.003
```

```
xTicksLoc = [Date(1980);Date(1990);Date(2000);Date(2010)]
xTicksLab = Dates.format.(xTicksLoc,"Y")     #formatting of date axis


p1 = plot( dNb,[ϵ.^2 e.^2],
      linecolor = [:blue :red],
      linestyle = [:solid :dash],
      label = ["AR(1) model" "historical mean"],
      legend = :topleft,
      xticks = (xTicksLoc,xTicksLab),
      title = "forecast errors^2" )
display(p1)
```

```
p2 = plot( dNb,[cumsum(ϵ.^2) cumsum(e.^2)],
      linecolor = [:blue :red],
      linestyle = [:solid :dash],
      label = ["AR(1) model" "historical mean"],
      legend = :topleft,
      xticks = (xTicksLoc,xTicksLab),
      title = "Cumulated forecast errors^2" )
display(p2)

p3 = plot( dNb,cumsum(ϵ.^2) - cumsum(e.^2),
      linecolor = :black,
      linestyle = :solid,
      label = ["AR(1) model - historical mean"],
      xticks = (xTicksLoc,xTicksLab),
      title  = "Difference of cumulated forecast errors^2" )
display(p3)
```




Cumulated forecast errors^2
AR(1) model
historical mean
2.00×10^4
1.50×10^4
1.00×10^4
5.00×10^3
0
1980
1990
2000
2010



Mariano-Diebold and Clark-West Tests (extra)

The Mariano-Diebold and Clark-West tests compare the prediction errors of two models (e: benchmark; ϵ: your model). Notice that the Mariano-Diebold test is not well suited for nested models (when your model is an augmented version of the baseline model). Use the Clark-West in that case.

If $g_1 = e^2$ and $g_2 = \epsilon^2$ are not autocorrelated, then the standard deviation of the sample average is $\text{Std}(g_i)/\sqrt{T}$. For simplicity, this assumption is used below. (Otherwise, use a Newey-West estimator or similar.)

```
"""
    MDCW(e,ϵ)

Moment conditions doing MD or CW tests of two series of forrecast errors (e: benchmark,ϵ: from

"""
MDCW(e,ϵ) = hcat(e.^2 - ϵ.^2, 2*e.*(e - ϵ))   #Mariano-Diebold and Clark&West
```

```
MDCW
```

```
g  = MDCW(e,ϵ)          #e,ϵ are from the recursive estimation (above)
Tg = size(g,1)

μ      = mean(g,dims=1)
Stdμ   = std(g,dims=1)/sqrt(Tg)
tstats = μ./Stdμ

printblue("t-stats of tests of difference in performance (errors^2)\n")
printmat(tstats';rowNames=["Mariano-Diebold","Clark-West"])
```

```
t-stats of tests of difference in performance (errors^2)

Mariano-Diebold    -0.430
Clark-West          0.814
```

A Trading Strategy

This section implements a momentum strategy (buy past winners, short sell past losers), which is rebalanced daily. For simplicity, we disregard trading costs.

Implementing the Strategy

1. Rank the 25 assets according to the lagged return R_{t-1}.
2. (In the evening of) period $t-1$ buy 1/5 of each of the 5 best assets based on the ranking in point 1. Similarly, buy -1/5 (short-sell) each of the 5 worst assets. Collect these portfolio weights in a vector w_t.
3. In period t , the return on the portfolio is $R_{p,t} = w_t'R_t$.
4. Repeat for all periods

A Remark on the Code

- With `x = [9,7,8]`, the `rankPs(x)` function (see below) gives the output `[3,1,2]`. This says, for instance, that 7 is the lowest number (rank 1).
- The portfolio weights are not stored for use outside the loop. Only the strategy (portfolio) return is.

```
"""
    rankPs(x)

Calculates the ordinal rank of eack element in a vector `x`. As an aternative,
use `ordinalrank` from the `StatsBase.jl` package.

"""
rankPs(x) = invperm(sortperm(x))
```

```
rankPs
```

```
m = 5   #number of assets in long/short leg of portfolio

R = copy(R25)      #we recycly the `R` notation: it is now a Tx25 matrix
(T,n) = size(R)

Rₚ = fill(NaN,T)
for t in 2:T          #loop over periods, save portfolio returns
    #local r,w            #local/global is needed in script
    r     = rankPs(R[t-1,:])
    w     = (r.<=m)*(-1/m) + ((n-m+1).<=r)*(1/m)
    Rₚ[t] = w'R[t,:]            #same as sum(w.*R[t,:])
end

Rₚ = Rₚ[2:end];
```

We calculate the mean (excess) return, its standard deviation and the Sharpe ratio. Annualising is done by assuming 250 trading days per year (means are multiplied by 250 and standard deviation by $\sqrt{250}$). Compare with the excess return on passively holding an equity market index.

```
μ = mean(Rₚ)                 #strategy
σ = std(Rₚ)

Rme = Rm - Rf                #market
μₘ  = mean(Rme[2:end])
σₘ  = std(Rme[2:end])

printblue("Annualised results:\n")
result  = [μ*250;σ*sqrt(250);μ/σ*sqrt(250)]
resultₘ = [μₘ*250;σₘ*sqrt(250);μₘ/σₘ*sqrt(250)]
printmat(result,resultₘ,colNames=["Strategy","market"],rowNames=["mean","std","SR"])
```

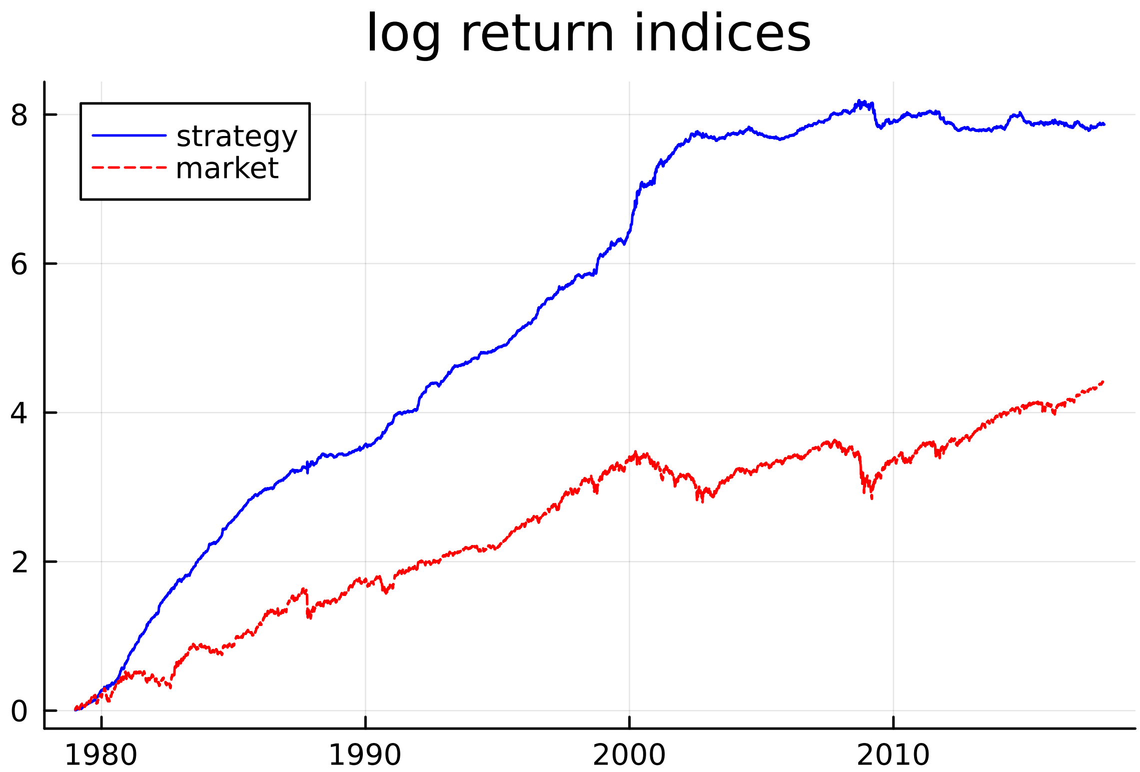
```
Annualised results:

      Strategy    market
mean    16.134     8.307
std      9.871    16.770
SR       1.634     0.495
```

To cumulate the returns to a return index, use $(1 + R_1)$, $(1 + R_1)(1 + R_2)$, etc. However, this does not work for excess returns, so convert them to net returns by adding the riskfree rate.

It is often more useful to show the logarithm of the return index. The slope can then be interpreted as a return.

```
Rbₚ = Rₚ + Rf[2:end]                #add Rf to make it a net return

Vₚ = cumprod(1 .+ Rbₚ/100)          #cumulate to return index ("value")
Vₘ = cumprod(1 .+ Rm[2:end]/100);  #notice /100 since percentage returns
```

```
xTicksLoc = [Date(1980);Date(1990);Date(2000);Date(2010)]
xTicksLab = Dates.format.(xTicksLoc,"Y")

plot( dN[2:end],log.([Vₚ Vₘ]),
      linecolor = [:blue :red],
      linestyle = [:solid :dash],
      label = ["strategy" "market"],
      legend = :topleft,
      xticks = (xTicksLoc,xTicksLab),
      title = "log return indices" )
```



log return indices
strategy
market
0
2
4
6
8
1980
1990
2000
2010

Performance Evaluation

The first part of this notebook summarizes the conventional performance measures for an investment fund. The second part does a "style analysis" to find out how the fund has changed its portfolio weights over time.

Load Packages and Extra Functions

```
using Printf, Dates, DelimitedFiles, Statistics

include("src/printmat.jl")
include("src/OlsM.jl");
```

```
using Plots

gr(size=(480,320))
default(fmt = :png)
```

Loading Data

The weekly return data for the mutual funds that we will evaluate are in the matrix `R`, the benchmarks (a number of returns on different asset indices) in `Rb` and the riskfree rate in the vector `Rf`.

A Remark on the Code

In the code below, `x` from `readdlm()` is an `Any` matrix (the first column are strings, the other columns are numbers). It often helps to convert to the right type. For the numbers in column `2:end` we can do that by `Float64.(x[:,2:end])` (or alternatively by `convert.(Float64,x[:,2:end])`).

```
(x,header) = readdlm("Data/Fin1PerfEvalEmp.csv",',',header=true)

(IndNames,FundNames) = (header[2:9],header[10:11])     #names of variables

dN        = Date.(x[:,1],"yyyy-mm-dd")
(Rb,R,Rf) = (Float64.(x[:,2:9]),Float64.(x[:,10:11]),Float64.(x[:,12]))  #benchmarks, funds, r

printblue("The data is weekly. The first 4 observations are:")
printmat(first(dN,4))
```

```
The data is weekly. The first 4 observations are:
1999-01-15
1999-01-22
1999-01-29
1999-02-05
```

Performance Evaluation

The next few cells report a number of different performance measures for the funds. We annualize the weekly return statistics by multiplying by $\sqrt{52}$ or 52, depending on the measure. In many cases, we also convert the numbers to percentage (%) by multiplying by 100.

A Remark on the Code

We use `Re` to denote excess returns for the funds and `Reₘ` the excess return of the market. (`Rᵉₘ` looks ugly, so we avoid it.) Similarly for `μe` and `μeₘ`.

```
Re  = R .- Rf             #excess returns of the funds
Reₘ = Rb[:,1] - Rf;        #excess returns of the marketₘ (S&P 500)
```

Average Returns, Sharpe Ratio and M²

The Sharpe ratio (SR) is μ^e/σ, where μ^e is the average excess return and σ the standard deviation of the excess return. The M^2 is the difference of the SRs of a fund and the market, scaled by the standard deviation of the market.

Mean returns are annualized by $\times 52$ and standard deviations by $\times\sqrt{52}$, so a SR by $\times\sqrt{52}$.

```
"""
    SRFn(Re,Reₘ,AnnFactor=[])

Calculate Sharpe rations and M²

"""
function SRFn(Re,Reₘ,AnnFactor=Int[])
    μᵉ  = mean(Re,dims=1)
    μeₘ = mean(Reₘ)
    σ   = std(Re,dims=1)         #std, 1xn
    σₘ  = std(Reₘ)
    SR  = μᵉ./σ
    SRₘ = μeₘ/σₘ
    MM  = (SR .- SRₘ)*σₘ
    MMₘ = 0                              #(SRₘ.-SRₘ)*σₘ=0
    if !isempty(AnnFactor)
        (μᵉ,μeₘ,MM) = (μᵉ,μeₘ,MM).*AnnFactor
        (SR,SRₘ)    = (SR,SRₘ).*sqrt(AnnFactor)
    end
    return μᵉ, μeₘ, SR, SRₘ, MM
end
```

```
SRFn
```

```
(μᵉ, μeₘ, SR, SRₘ, MM) = SRFn(Re,Reₘ,52)

xut = hcat([μeₘ;μᵉ']*100,[SRₘ;SR'],[0;MM']*100)   #returns and MM in percent
printmat(xut;colNames=["ERe","SR","M²"],rowNames=["Market";FundNames])
```

```
                                       ERe         SR         M²
Market                               4.696      0.268      0.000
Putnam Asset Allocation: Growth A    4.143      0.283      0.265
Vanguard Wellington                  5.234      0.489      3.865
```

Appraisal Ratio, Traynor's Ratio and

Appraisal Ratio

Estimate a single index model

$R_t^e = \alpha + \beta R_{mt}^e + \epsilon_t$

The appraisal ratio (AR) is $\alpha/\text{Std}(\epsilon)$.

Treynor's Ratio and T²

Treynor's ratio is μ^e/β, where β is the slope from the single index regression.

A Remark on The Code

- The `OlsM(y,x)` function (included above) does a regression for each of the n columns in `y` on the same set of K regressors in `x`. It returns $(K \times n)$ coefficients `b` and $(1 \times n)$ standard deviations of the residuals σ.

```
"""
    AppraisalRatioFn(Re,Reₘ,AnnFactor=52)

Calculate appraisal ratio, Treynor's ratio and α.

"""
function ARTRFn(Re,Reₘ,AnnFactor=Int[])
    μᵉ  = mean(Re,dims=1)
    μeₘ = mean(Reₘ)
    T   = size(Re,1)
    (b,σₑ) = OlsM(Re,[ones(T) Reₘ])   #OLS regression for each column in Reₚ
    (α,β)  = (b[1:1,:],b[2:2,:])      #intercept and slopes (1xn)
    AR     = α./σₑ                    #α and σₑ are 1xn
    (Re == Reₘ) && (AR=0)             #impose this because α,σₑ close to 0 may give stange rat
    TR     = μᵉ./β
    TT      = TR .- μeₘ
    if !isempty(AnnFactor)
        (α,TR,TT) = (α,TR,TT).*AnnFactor
        AR        = AR.*sqrt(AnnFactor)
    end
    return AR, TR, TT, α
end
```

```
ARTRFn
```

```
(AR,TR,TT,α)      = ARTRFn(Re,Reₘ,52)
(ARₘ,TRₘ,TTₘ,αₘ) = ARTRFn(Reₘ,Reₘ,52)

xut = hcat([ARₘ;AR'],[TRₘ;TR']*100,[TTₘ;TT']*100,[αₘ;α']*100)   #TR, TT and α in percent
printmat(xut,colNames=["AR","TR","T²","α"],rowNames=["Market";FundNames])
```

```
                                         AR        TR        T²         α
Market                                0.000     4.696    -0.000    -0.000
Putnam Asset Allocation: Growth A     0.092     5.193     0.497     0.396
Vanguard Wellington                   0.671     9.158     4.462     2.550
```

Style Analysis

The regression is $Y = \sum_{j=1}^{K} b_j X_j + \varepsilon$, where $0 <= b_j$ and $\sum_{j=1}^{K} b_j = 1$.

Write the sum of squared residuals of the regression as

$(Y - Xb)'(Y - Xb) = Y'Y - 2Y'Xb + b'X'Xb$.

Only the two last terms matter for the choice of b.

A Remark on the Code

- The code uses the OSQP.jl package which solves problems of the type: $\min 0.5b'Pb + q'b$ subject to $l \leq Ab \leq u$.
- The restricted regression (above) can easily be written in this form by letting $P = X'X/T$ and $q = -X'Y/T$. These choices give the loss function $0.5b'X'Xb/T - Y'Xb/T$, which is proportional $(0.5/T)$ to the last two terms in the sum of squared residuals. Dividing by T improves the numerical stability.
- The A matrix is used to construct the restrictions (b sums to 1 and $0 \leq b \leq 1$).

```
using LinearAlgebra, SparseArrays, OSQP   #OSQP needs SparseArrays
```

```
"""
    StyleAnalysisPs(Y,X)


# Input:
- `Y::Vector`:          T-vector, returns of a fund
- `X ::Matrix`:         TxK matrix, returns on m benchmarks
```

```
 # Output:
- `b_sa::Vector`:       K-vector, restricted regression coefficients
- `b_ls::Vector`:       K-vector, OLS regression coefficients

"""
function StyleAnalysisPs(Y,X)

    (T,K) = (size(X,1),size(X,2))
    b_ls = X\Y                          #LS estimate, no restrictions

    settings = Dict(:verbose => false)
    P = X'X/T
    q = -X'Y/T
    A = [ones(1,K);I]                  #1st restriction: sum(b)=1,
    l = [1;zeros(K)]                   #the rest: 0<=b<=1
    u = [1;ones(K)]

    settings = Dict(:verbose => false)
    model = OSQP.Model()    #(P,A) must be `Sparse`, (q,l,u) vectors of `Float64`
    OSQP.setup!(model; P=sparse(P), q=q, A=sparse(A), l=l, u=u, settings...)
    result = OSQP.solve!(model)
    b_sa = result.info.status == :Solved ? result.x : NaN

    return b_sa, b_ls

end
```

```
StyleAnalysisPs
```

The next cell makes a "style analysis regression" based on the entire sample. The dependent variable is the first mutual fund in R (see data loading) and the regressors include all indices (again, see data loading).

```
(b,b_ls) = StyleAnalysisPs(R[:,1],Rb)

printblue("OLS and style analysis coeffs:")
colNames = ["OLS","Restricted LS"]
xut      = [b_ls b;sum([b_ls b],dims=1)]
printmat(xut;colNames,rowNames=[IndNames;"sum"],width=15)

printred("Notice that the restricted LS has (approximately) no negative coeffs and that sum(co
```

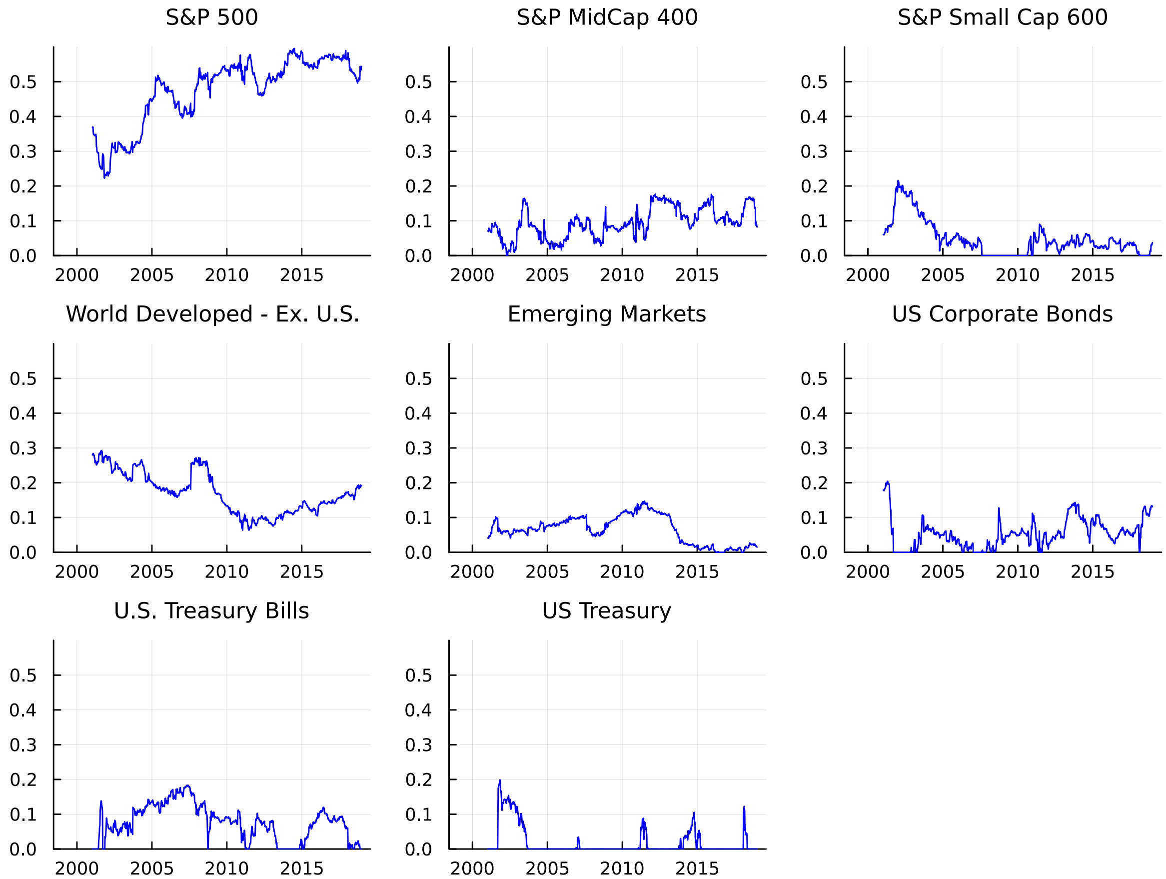
```
OLS and style analysis coeffs:
                                      OLS  Restricted LS
S&P 500                             0.429          0.429
S&P MidCap 400                      0.086          0.087
S&P Small Cap 600                   0.067          0.069
World Developed - Ex. U.S.          0.199          0.203
Emerging Markets                    0.054          0.058
US Corporate Bonds                  0.176          0.071
U.S. Treasury Bills                 0.085          0.083
US Treasury                        -0.151         -0.000
sum                                 0.944          1.000

Notice that the restricted LS has (approximately) no negative coeffs and that sum(coeffs) = 1
```

Redo the Style Analysis on a Moving Data Window

of size `WinSize` (see below). Then plot to see how the coefficients change over time.

```
(T,K)   = size(Rb)
WinSize = 104

b = fill(NaN,T,K)
for t = (WinSize+1):T
    #local vv                 #local/global is needed in script
    vv     = (t-WinSize):t    #moving data window
    b[t,:] = StyleAnalysisPs(R[vv,1],Rb[vv,:])[1]
end
```

```
xTicksLoc = [Date(2000);Date(2005);Date(2010);Date(2015)]
xTicksLab = Dates.format.(xTicksLoc,"Y")

p1 = plot( dN,b,
           layout = @layout[a a a;a a a;a a _],   #_ to get blank subplot
           legend = false,
           size = (800,600),
           linecolor = :blue,
           xticks = (xTicksLoc,xTicksLab),
           ylims = (0,0.6),
           title = reshape(string.(IndNames),1,:),
           titlefontsize = 10 )
display(p1)
```


S&P 500
S&P MidCap 400
S&P Small Cap 600
World Developed - Ex. U.S.
Emerging Markets
US Corporate Bonds
U.S. Treasury Bills
US Treasury
0.0
0.1
0.2
0.3
0.4
0.5
2000
2005
2010
2015

Long Run Portfolio Choice

This notebook summarizes how the distribution of returns is affected by the investment horizon.

Load Packages and Extra Functions

```
using Printf, LinearAlgebra, Distributions

include("src/printmat.jl")
include("src/lag.jl");
```

```
using Plots, LaTeXStrings
default(size = (480,320),fmt = :png)
```

Distribution of Long-Run Returns in the iid Case

If the excess log return over one period r^e is iid $N(\mu,\sigma^2)$, then excess log return over q periods is z_q^e is $N(q\mu,q\sigma^2)$. We use this result to draw the density function and to calculate the probability of $z_q^e < 0$.

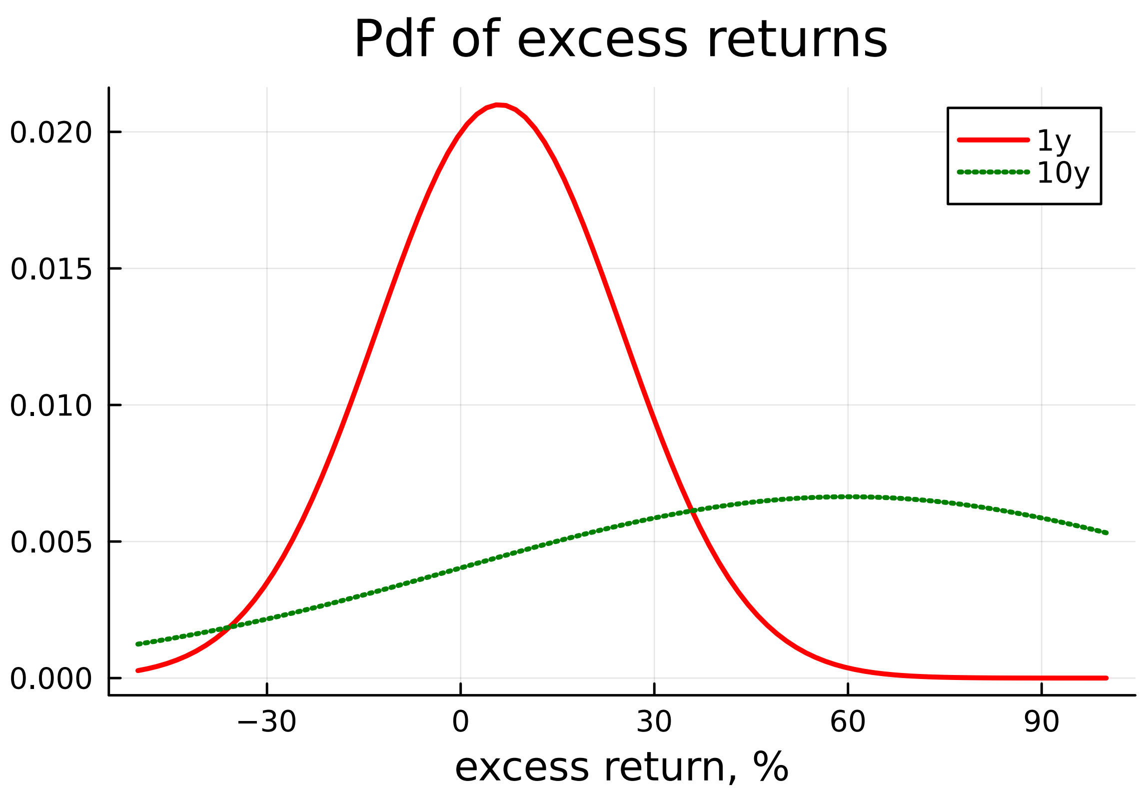
A Remark on the Code

The Distributions.jl package defines a normal distribution by `Normal(μ,σ)`, that is, using the standard deviation (not the variance) as the 2nd argument. For horizon `q` we thus use `Normal(q*μ,sqrt(q)*σ)`.

Pdfs of Long-Run Returns (for Different Horizons)

```
zᵉ_range = range(-50,100,length=101)
μᵉ = 0.06*100                                           #average excess return of annual data
σ  = 0.19*100                                           #std of annual data

pdf_1y  = pdf.(Normal(μᵉ,σ),zᵉ_range)                       #pdf of 1-year returns
pdf_10y = pdf.(Normal(10*μᵉ,sqrt(10)*σ),zᵉ_range);  #pdf of 10-year returns
```

```
p1 = plot( zᵉ_range,[pdf_1y pdf_10y],
           linecolor = [:red :green],
           linestyle = [:solid :dot],
           linewidth = 2,
           label = ["1y" "10y"],
           title = "Pdf of excess returns",
           xlabel = "excess return, %" )
display(p1)
```

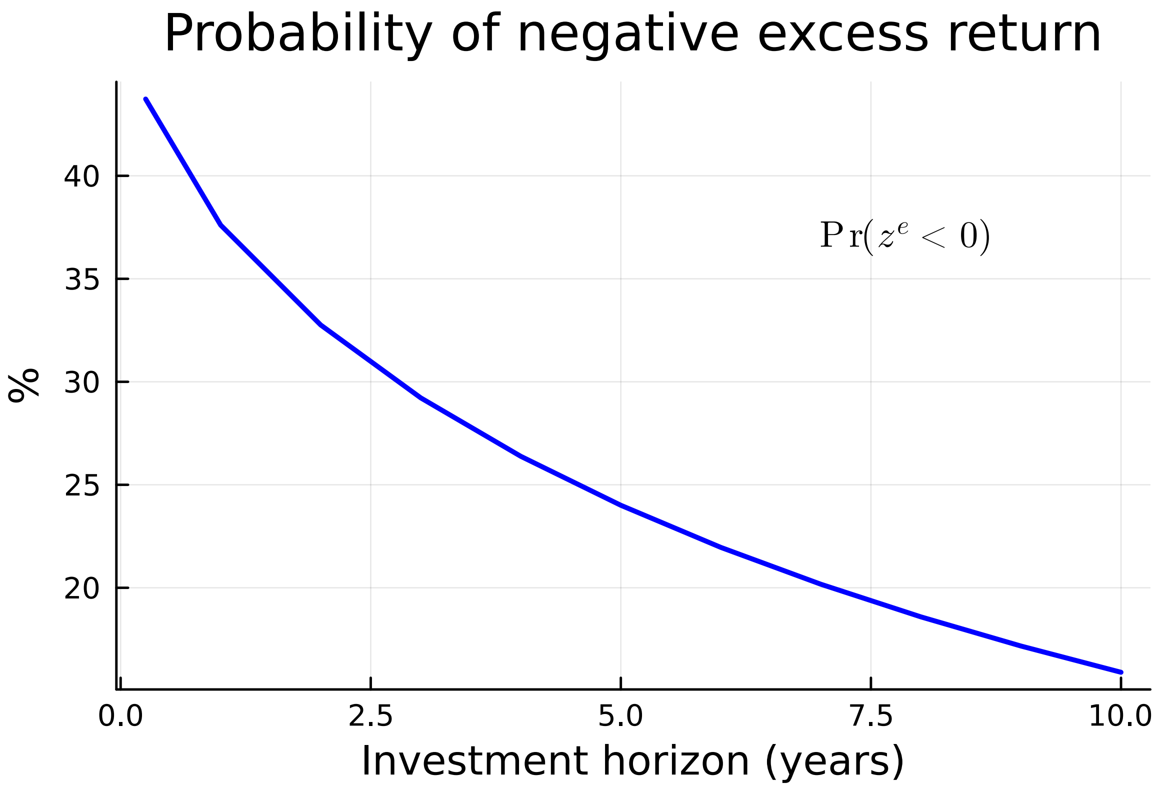


Prob($z < 0$) for Different Horizons

A Remark on the Code

code like `y = [exp(q) for q in 1:10]` creates a vector with 10 elements (`exp(1),exp(2)...`). It's a short form of a loop. In some cases, this is more conveniently writen as `exp.(1:10)`, but in other cases a loop works better (here it does since we have to change the distribution for each `q`).

```
q_max         = 10      #longest horizon considered in the calculations
qM = vcat(0.25,1:q_max)
ProbNegReturn = [cdf(Normal(q*μᵉ,sqrt(q)*σ),0) for q in qM]   #Pr(z<=0) for different horizons

txt = L"\mathrm{Pr}(z^e<0)"
p1 = plot( qM,ProbNegReturn*100,
           linecolor = :blue,
           linewidth = 2,
           legend = false,
           title = "Probability of negative excess return",
           xlabel = "Investment horizon (years)",
           ylabel = "%",
           annotation = (7,37,text(txt,10,:left)) )
display(p1)
```



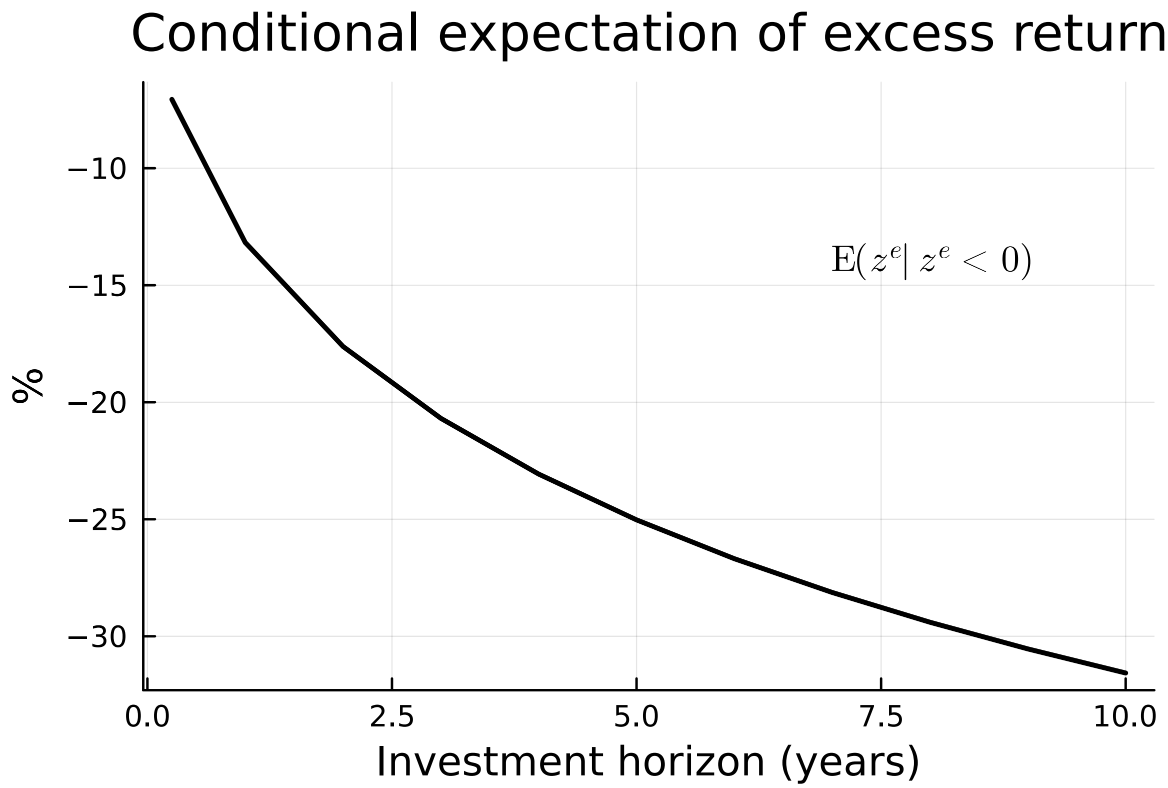
E(z |z < 0) for Different Horizons

The `Distributions.jl` package has methods fot trunctated distributions. We calculate $\mathrm{E}(z^e|z^e < 0)$ for horizon q by using

```
mean(truncated(Normal(q*μᵉ,sqrt(q)*σ);upper=0)
```

```
CondER = [mean(truncated(Normal(q*μᵉ,sqrt(q)*σ);upper=0)) for q in qM]

txt = L"\mathrm{E}(z^e|z^e<0)"
p1 = plot( qM,CondER,
           linecolor = :black,
           linewidth = 2,
           legend = false,
           title = "Conditional expectation of excess return",
           xlabel = "Investment horizon (years)",
           ylabel = "%",
```

```
            annotation = (7,-14,text(txt,10,:left)) )
display(p1)
```



Mean-Variance Portfolio Choice

This section runs the time series model presented in the lecture notes. It therefore more code than usual

```
include("src/Var1IR.jl");   #impulse response of VAR(1), Cov(x) from MA(q)
```

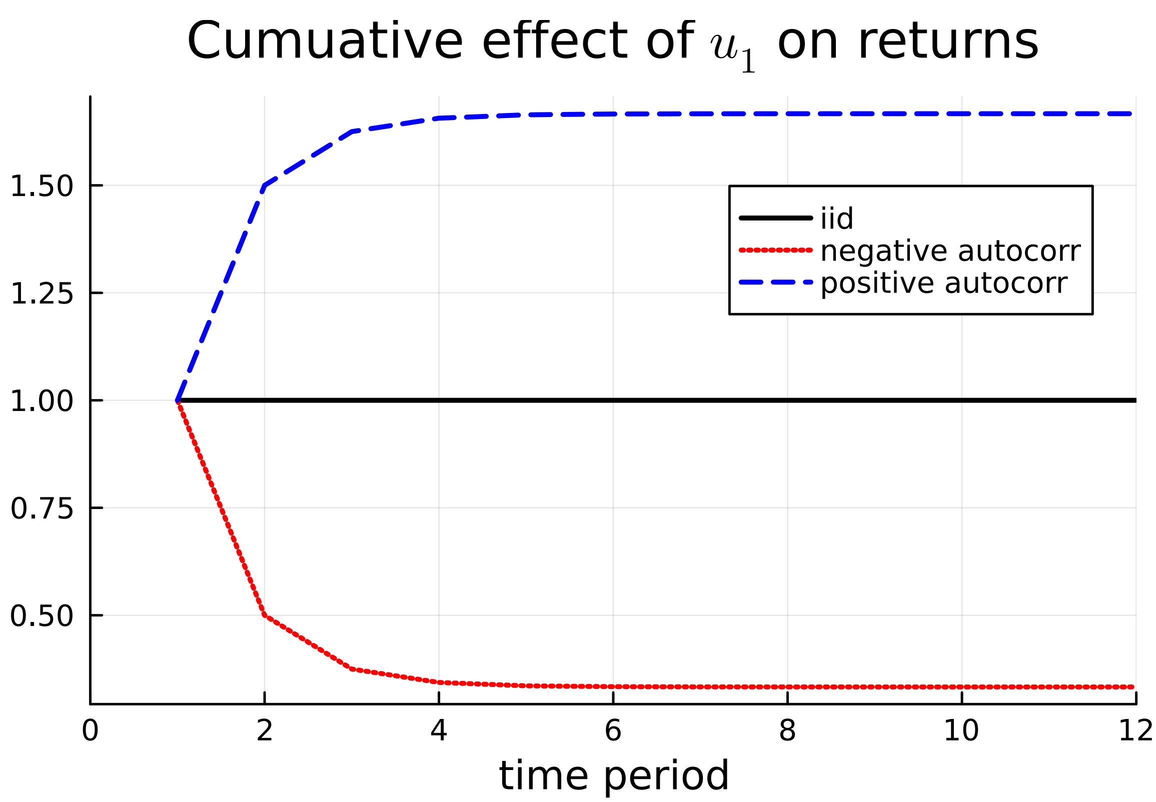
```
"""
    xzModel3(ϕ,θ,T)
"""
function xzModel3(ϕ,θ,T)
  A = [0 1;
       0 ϕ]
  B = [1 0;
```

```
      θ 1]
  C = VAR1IR(A,T,B)
  r = C[1,:,:]'
  z = C[2,:,:]'
  return r, z
end

"""
    xzModel3Cov(ϕ,θ,σᵤ,σ_eta,q)

Calculate conditional variance-covariance matrix of q future
returns (r_1,r_2,...,r_q) by first finding the MA representation
and then apply MACov.
"""
function xzModel3Cov(ϕ,θ,σᵤ,σ_eta,q)

  (r_IR,z_IR) = xzModel3(ϕ,θ,q+10)
  r_IR .*= [σᵤ σ_eta]
  z_IR .*= [σᵤ σ_eta]

  k = 2                                       #no. variables
  C = fill(NaN,q,k,q)
  for t in 1:q
    r_t = vcat(zeros(q-t,k),r_IR[1:t,:])      #t=1: [zeros(9,2);[1 0]]
    C[t,:,:] = r_t'                           #t=2: [zeros(8,2);[-0.5 1];[1 0]]
  end

  CovM = MACov(C,1.0I(k))

  return CovM, r_IR, C

end
```

```
xzModel3Cov
```

```
(ϕ,σᵤ,σ_eta,T) = (0.25,0.05,0.02,12)

(r_0,z_0) = xzModel3(0,0,T)
(r_m,z_m) = xzModel3(ϕ,-0.5,T)
(r_p,z_p) = xzModel3(ϕ,0.5,T)
```

```
p1 = plot( 1:T,cumsum([r_0[:,1] r_m[:,1] r_p[:,1]],dims=1),
           xlims = (0,T),
           linestyle = [:solid :dot :dash],
           linecolor = [:black :red :blue],
           linewidth = 2,
           label = ["iid" "negative autocorr" "positive autocorr"],
           legend = (0.7,0.8),
           title = L"Cumuative effect of $u_1$ on returns",
           xlabel = "time period" )
display(p1)
```

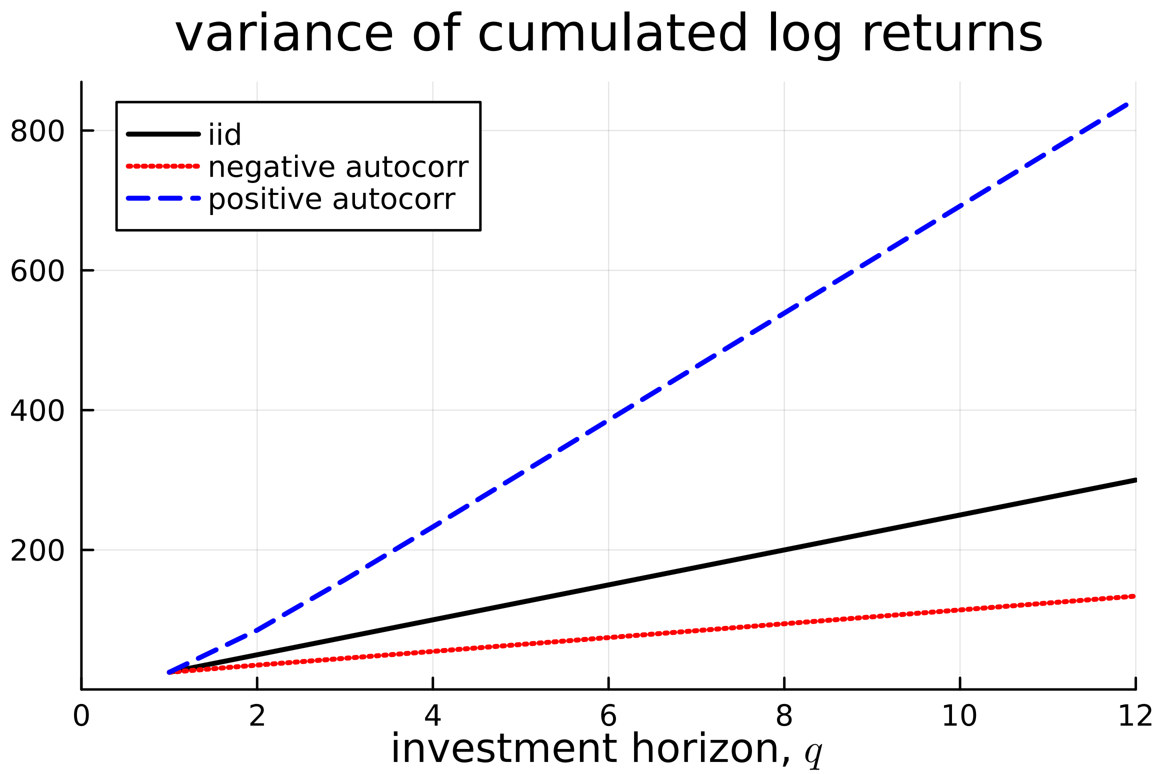



```
qM = 1:T

Var_q = fill(NaN,T,3)
for q in qM
  local CovM_m, CovM_0, CovM_p
  CovM_0, = xzModel3Cov(ϕ,0,σᵤ,0,q)                  #iid
  CovM_m, = xzModel3Cov(ϕ,-0.5,σᵤ,σ_eta,q)           #reversal
  CovM_p, = xzModel3Cov(ϕ,0.5,σᵤ,σ_eta,q)            #momentum
```

```
  Var_q[q,:] = [sum(CovM_0) sum(CovM_m) sum(CovM_p)]   #var(r₁+r₂+...+r_q)
end
printmat(Var_q*10_000);
```

```
   25.000    25.000    25.000
   50.000    35.250    85.250
   75.000    45.016   157.516
  100.000    54.860   232.985
  125.000    64.737   309.269
  150.000    74.623   385.756
  175.000    84.511   462.295
  200.000    94.400   538.846
  225.000   104.289   615.400
  250.000   114.178   691.956
  275.000   124.067   768.511
  300.000   133.956   845.067
```

```
p1 = plot( qM,Var_q*10_000,
           xlims = (0,T),
           linestyle = [:solid :dot :dash],
           linecolor = [:black :red :blue],
           linewidth = 2,
           title = "variance of cumulated log returns",
           label = ["iid" "negative autocorr" "positive autocorr"],
           xlabel = L"investment horizon, $q$" )
```

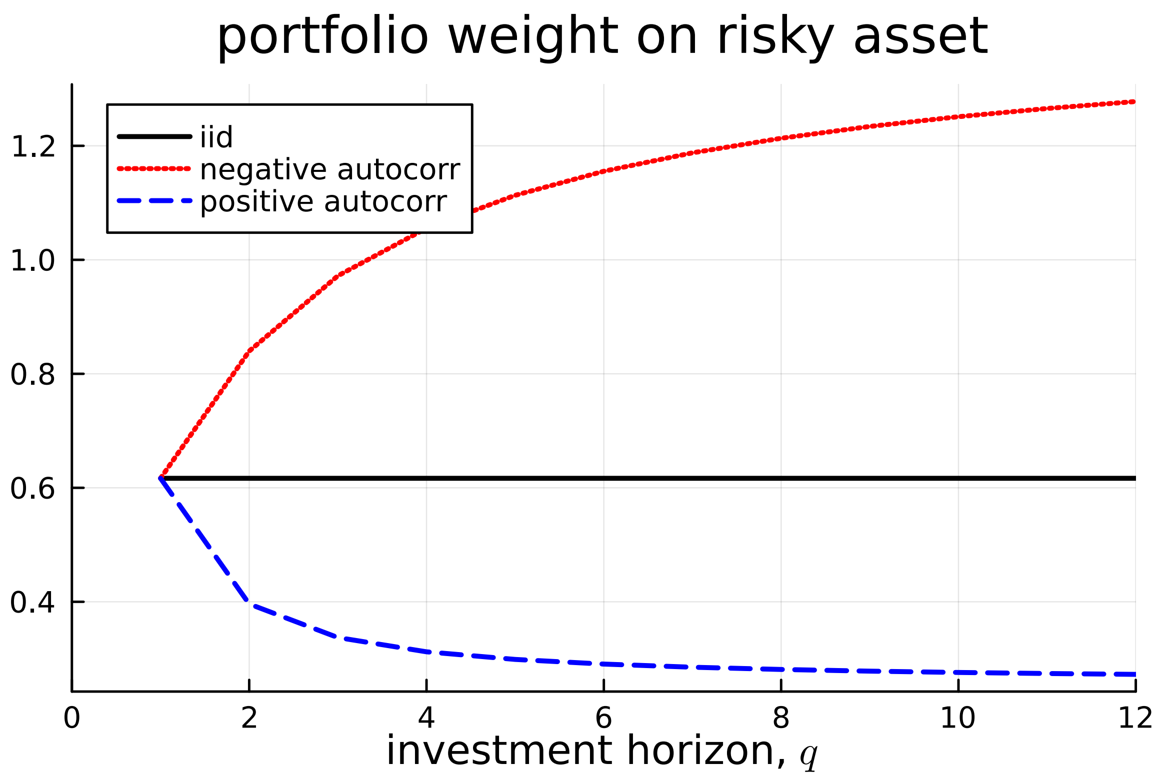



```
"""
    vWeight(μᵉ,σ²,k)

Optimal portfolio weight on the risky asset, so (1-v) is in the risk-free asset

"""
vWeight(μᵉ,σ²,k) = μᵉ/((1+k)*σ²) + 1/(2*(1+k))
```

```
vWeight
```

```
(μᵉ,k) = (0.8/100,5)
v_q = vWeight.(μᵉ.*qM,Var_q,k)

p1 = plot( qM,v_q,
           xlims = (0,T),
           linestyle = [:solid :dot :dash],
           linecolor = [:black :red :blue],
           linewidth = 2,
           title = "portfolio weight on risky asset",
```

```
         label = ["iid" "negative autocorr" "positive autocorr"],
         xlabel = L"investment horizon, $q$" )
```



Dynamic Portfolios

and intertemporal hedging in a simple discrete time model.

WARNING: the lecture notes for this topic has been revised, and this notebook has still not been synced with that.

Load Packages and Extra Functions

```
using Printf, LinearAlgebra, Distributions

include("src/printmat.jl");
```

Log Utility

We first find the *myopic* (period by period) optimal portfolio when the investor has log utility,

$U(R_p) = \ln(1 + R_p)$,

where R_p is the portfolio return.

In all examples discussed below, the log returns are normally distributed, but the vector of *expected* returns changes across time. In particular, the vector of expected returns take on two different values (state A and state B).

```
rf   = 0.03           #riskfree rate
σ²   = 100/10000      #variance of risky asset
μe1A = 1/100          #mean excess return of single risky asset, state A
μe1B = 0.5/100        #state B

vA = (μe1A + σ²/2)/σ²
vB = (μe1B + σ²/2)/σ²
```

```
rowNames1 = ["risky asset","riskfree"]
printblue("Portfolio weights in the two states (the case of a single risky asset):")
printmat([vA vB;1-sum(vA) 1-sum(vB)];colNames=["state A","state B"],rowNames=rowNames1)
```

```
Portfolio weights in the two states (the case of a single risky asset):
              state A   state B
risky asset     1.500     1.000
riskfree       -0.500     0.000
```

```
"""
Calculate optimal portfolio for log utility case, when the log returns are N(μe+rf,Σ),
Campbell&Viceira.
"""
function OptPortLogUtil(Σ,μe,rf)
  v     = inv(Σ)*(μe+diag(Σ)/2)
  Erp   = rf + v'μe + v'diag(Σ)/2 - v'Σ*v/2
  Varrp = v'Σ*v
  return v, Erp, Varrp
end
```

```
OptPortLogUtil
```

```
Σ  = [166  34  58;              #3 risky assets
       34  64   4;
       58   4 100]/10000
μeA = [2.0, 1.0, 0.5]/100       #expected excess returns in state A
μeB = [2.0, 0.0, 0.5]/100       #expected excess returns in state B

vA, = OptPortLogUtil(Σ,μeA,rf)  #myopic portfolio choice in each state
vB, = OptPortLogUtil(Σ,μeB,rf)

printblue("Portfolio weights in the two states (several risky assets):")
rowNames3 = ["asset 1","asset 2","asset 3","riskfree"]
printmat([vA vB;1-sum(vA) 1-sum(vB)];colNames=["state A","state B"],rowNames=rowNames3)
```

```
Portfolio weights in the two states (several risky assets):
           state A   state B
asset 1      1.384     1.810
asset 2      1.318    -0.460
asset 3      0.144    -0.031
riskfree    -1.847    -0.319
```

CRRA Utility

We now turn to the *myopic* (period by period) optimal portfolio when the investor has CRRA utility

$U(R_p) = (1 + R_p)^{1-\gamma}/(1-\gamma)$,

where R_p is the portfolio return. As $\gamma \to 1$, this becomes the same as log utility (need to take the limit to prove that).

```
γ = 3                                   #risk aversion

vA = (μe1A + σ²/2)/(σ²*γ)              #with a single risky asset
vB = (μe1B + σ²/2)/(σ²*γ)

printblue("Portfolio weights in the two states (the case of a single risky asset):")
printmat([vA vB;1-sum(vA) 1-sum(vB)];colNames=["state A","state B"],rowNames=rowNames1)
```

```
Portfolio weights in the two states (the case of a single risky asset):
              state A   state B
risky asset     0.500     0.333
riskfree        0.500     0.667
```

```
"""
Calculate optimal portfolio for CRRA utility case, when the log returns are N(μe+rf,Σ),
Campbell&Viceira.
"""
function OptPortCRRA(Σ,μe,rf,γ)
  v     = inv(Σ)*(μe+diag(Σ)/2)/γ
  Erp   = rf + v'μe + v'diag(Σ)/2 - v'Σ*v/2
  Varrp = v'Σ*v
  return v, Erp, Varrp
end
```

```
OptPortCRRA
```

```
γ = 3

vA, = OptPortCRRA(Σ,μeA,rf,γ)              #with several risky assets
vB, = OptPortCRRA(Σ,μeB,rf,γ)

printblue("Portfolio weights in the two states (several risky assets):")
printmat([vA vB;1-sum(vA) 1-sum(vB)];colNames=["state A","state B"],rowNames=rowNames3)
```

```
Portfolio weights in the two states (several risky assets):
           state A   state B
asset 1      0.461     0.603
asset 2      0.439    -0.153
asset 3      0.048    -0.010
riskfree     0.051     0.560
```

Intertemporal Hedging

We now consider the more difficult case when the CRRA investor considers several periods.

In the case of log utility ($\gamma = 1$), this actually gives a myopic solution: in each period, $\ln(1 + R_p)$ is maximized, where R_p is the one period portfolio return.

With CRRA this may no longer hold. In particular, if there are some predictable (non-iid) features of the asset returns, then today's investment may be influenced by how the return over the next period is correlated with the investment opportunities in the subsequent periods.

The optimal solution used in the next few cells (see lecture notes for details) is for the case when the vector (n assets) of excess returns follow

$r^e_{t+1} = a + z_t + u_{t+1}$,

where the vector z_t follows the VAR(1)

$z_{t+1} = \phi z_t + \eta_{t+1}$, with η_{t+1} being $N(\mathbf{0}, \Sigma_\eta)$.

The covariance matrix of u_{t+1} and η_{t+1}, which plays a key role of the analysis, is denoted by $\Sigma_{u\eta}$.

Notice that the first equation implies that *(1)* returns are predictable, that is, the expected returns change over time; *(b)* and that those expectations potentially correlates with today's return. This may lead to *intertemporal hedging*, where the portfolio weights for an investment between t and $t + 1$ is affected by how the return in $t + 1$ correlates with the investment oppurtunity set in $t + 1$, that is, with the return distribution in $t + 2$.

A Single Risky Asset

and a riskfree

```
"""
Optimal dynamic portfolio choice when there is a single risky asset.
"""
function CRRAPortOpt1(Σ_u,Σ_η,Σ_uη,a,ϕ,γ,z)
  v_myop  = inv(Σ_u)*(a + z + Σ_u/2)/γ
```

```
  Σ       = 2*Σ_u + Σ_η + 2*Σ_uη
  v_noreb = inv(Σ)*(2*a + (I+ϕ)*z + Σ/2)/γ
  Ev1     = inv(Σ_u)*(a + ϕ*z + Σ_u/2)/γ
  v_rebal = inv(Σ_u)*(a + z + Σ_u/2 + (1-γ)*Σ_uη*Ev1)/γ
  return v_myop, v_noreb, v_rebal
end
```

```
CRRAPortOpt1
```

```
(γ,Σ_u,Σ_η,a,ϕ) = (3,100/10000,75/10000,0.75/100,0.9)     #parameters in example with 1 risky
Σ_uη0 = 0.0                         #corr(u,η)=0.0
(v_myopH,v_norebH,v_rebalH) = CRRAPortOpt1(Σ_u,Σ_η,Σ_uη0,a,ϕ,γ,0)

printblue("Portfolio weights: myopic and 2-period investor (who can rebalance), corr(u,η)=0")
xx = [v_myopH   v_rebalH;
      1-sum(v_myopH) 1-sum(v_rebalH)]
printmat(xx;colNames=["myopic","2-period investor"],rowNames=rowNames1,width=20)

printblue("no intertemporal hedging if corr(u,η) = 0.0")
```

```
Portfolio weights: myopic and 2-period investor (who can rebalance), corr(u,η)=0
                         myopic   2-period investor
risky asset               0.417               0.417
riskfree                  0.583               0.583

no intertemporal hedging if corr(u,η) = 0.0
```

```
Σ_uηL  = -0.5*sqrt(Σ_u*Σ_η)     #corr(u,η)=-0.5
(v_myopL,v_norebL,v_rebalL) = CRRAPortOpt1(Σ_u,Σ_η,Σ_uηL,a,ϕ,γ,0)

printblue("Portfolio weights: myopic and 2-period investor (who can rebalance), corr(u,η)=-0.5
xx = [v_myopL   v_rebalL;
      1-sum(v_myopL) 1-sum(v_rebalL)]
printmat(xx;colNames=["myopic","2-period investor"],rowNames=rowNames1,width=20)

printblue("the risky asset hedges changes in future investment opportunities when corr(u,η) <
```

```
Portfolio weights: myopic and 2-period investor (who can rebalance), corr(u,η)=-
0.5
                         myopic   2-period investor
```

```
risky asset                0.417               0.537
riskfree                   0.583               0.463
```

```
the risky asset hedges changes in future investment opportunities when corr(u,η) < 0
```

```
Σ_uηH  =  0.5*sqrt(Σ_u*Σ_η)     #corr(u,η)=0.5
(v_myopH,v_norebH,v_rebalH) = CRRAPortOpt1(Σ_u,Σ_η,Σ_uηH,a,ϕ,γ,0)

printblue("Portfolio weights: myopic and 2-period investor (who can rebalance), corr(u,η)=0.5"
xx = [v_myopH   v_rebalH;
      1-sum(v_myopH) 1-sum(v_rebalH)]
printmat(xx;colNames=["myopic","2-period investor"],rowNames=rowNames1,width=20)

printblue("the risky asset 'anti-hedges' changes in future investment opportunities when corr(
```

```
Portfolio weights: myopic and 2-period investor (who can rebalance), corr(u,η)=0.5
                          myopic   2-period investor
risky asset                0.417               0.296
riskfree                   0.583               0.704
```

```
the risky asset 'anti-hedges' changes in future investment opportunities when corr(u,η) > 0
```

Several Risky Assets

and a riskfree

```
"""
Optimal dynamic portfolio choice when there are several risky assets
"""
function CRRAPortOpt(Σ_u,Σ_η,Σ_uη,a,ϕ,γ,z)
  v_myop  = inv(Σ_u)*(a + z + diag(Σ_u)/2)/γ
  Σ       = 2*Σ_u + Σ_η + 2*Σ_uη
  v_noreb = inv(Σ)*(2*a + (I+ϕ)*z + diag(Σ)/2)/γ
  Ev1     = inv(Σ_u)*(a + ϕ*z + diag(Σ_u)/2)/γ
  v_rebal = inv(Σ_u)*(a + z + diag(Σ_u)/2 + (1-γ)*Σ_uη*Ev1)/γ
  return v_myop, v_noreb, v_rebal
end
```

```
CRRAPortOpt
```

```
γ = 3

Σ_u    = [166  34  58;
           34  64   4;
           58   4 100]/10000
Σ_η    = [0  0   0;
          0  100 0;
          0  0   0]/10000       #only asset 2 has dynamics in expected returns
a      = [2,0.5,0.5]/100
ϕ      = [0  0   0;             #doesn't matter for 2-period problem when z_t=0
          0  0.9 0;
          0  0   0]

Σ_uηL =  [0  0   0;             #Cov(u,η)<0 for asset 2
          0  -40  0;
          0  0    0]/10000
Σ_uηH =  [0  0   0;             #Cov(u,η)>0 for asset 2
          0  40   0;
          0  0    0]/10000

printblue("The model parameters:\n")

println("Σ_u*10000")
printmat(Σ_u*10000)

println("Σ_η*10000")
printmat(Σ_η*10000)

println("a*100")
printmat(a*100)

println("\nTwo cases: negative and positive Cov(u,η) for asset 2")
printmat(Σ_uηL*10000)
printmat(Σ_uηH*10000)
```

```
The model parameters:

Σ_u*10000
   166.000    34.000    58.000
    34.000    64.000     4.000
    58.000     4.000   100.000
```

```
Σ_η*10000
     0.000     0.000     0.000
     0.000   100.000     0.000
     0.000     0.000     0.000

a*100
     2.000
     0.500
     0.500


Two cases: negative and positive Cov(u,η) for asset 2
     0.000     0.000     0.000
     0.000   -40.000     0.000
     0.000     0.000     0.000

     0.000     0.000     0.000
     0.000    40.000     0.000
     0.000     0.000     0.000
```

```
(v_myopL,v_norebL,v_rebalL) = CRRAPortOpt(Σ_u,Σ_η,Σ_uηL,a,ϕ,γ,zeros(3))     #different Cov(u,η
(v_myopH,v_norebH,v_rebalH) = CRRAPortOpt(Σ_u,Σ_η,Σ_uηH,a,ϕ,γ,zeros(3))


printblue("Portfolio weights: myopic and 2-period investor (who can rebalance), corr(u,η) < 0"
xx = [v_myopL   v_rebalL;
      1-sum(v_myopL) 1-sum(v_rebalL)]
printmat(xx;colNames=["myopic","2-period investor"],rowNames=rowNames3,width=20)

printblue("Portfolio weights: myopic and 2-period investor (who can rebalance), corr(u,η) > 0"
xx = [v_myopH   v_rebalH;
      1-sum(v_myopH) 1-sum(v_rebalH)]
printmat(xx;colNames=["myopic","2-period investor"],rowNames=rowNames3,width=20)
```

```
Portfolio weights: myopic and 2-period investor (who can rebalance), corr(u,η) < 0
                      myopic   2-period investor
asset 1                0.532               0.516
asset 2                0.143               0.211
asset 3                0.019               0.026
riskfree               0.306               0.247

Portfolio weights: myopic and 2-period investor (who can rebalance), corr(u,η) > 0
```

	myopic	2-period investor
asset 1	0.532	0.549
asset 2	0.143	0.075
asset 3	0.019	0.012
riskfree	0.306	0.364