

# Exercise 1

## Machine Learning in Finance with Python (ECON5130)

Richard Foltyn  
*University of Glasgow*

Deadline: November 3, 12:00

This exercise is optional and will not be marked. Students are nevertheless highly encouraged to hand in their solutions on Moodle.

You can download this exercise as a Jupyter notebook from [GitHub](#). Use this notebook when typing up your solution.

## 1 Portfolio choice problem

Consider a portfolio choice problem where an investor chooses the fraction  $\alpha$  to invest in a risky asset in order to maximise expected utility,

$$\max_{\alpha \in [0,1]} E_t [u(W_{t+1})]$$

Assume that the investor consumes all of next-period's wealth  $W_{t+1}$  which is given by

$$W_{t+1} = R_{t+1}\alpha W_t + R_f(1 - \alpha)W_t$$

where  $W_t$  is the initial investable wealth in period  $t$ ,  $R_{t+1}$  is the gross return on the risky investment and  $R_f$  is the risk-free return on the fraction of the portfolio which is invested a risk-free asset (e.g., a bank deposit). The utility function is  $u(\bullet)$  has a constant relative risk aversion (CRRA) form and is given by

$$u(W) = \begin{cases} \frac{W^{1-\gamma}}{1-\gamma} & \text{if } \gamma \neq 1 \\ \log(W) & \text{if } \gamma = 1 \end{cases}$$

where  $\gamma$  is a parameter governing the investor's risk aversion (higher values of  $\gamma$  imply that the investor is more averse to taking risk).

For simplicity, let the gross risk-free return be  $R_f = 1$ . Finally, assume that the risky return can take on two realisations, high and low, with equal probability,

$$R_{t+1} = \begin{cases} 1 + \mu + \epsilon & \text{with probability } \frac{1}{2} \\ 1 + \mu - \epsilon & \text{with probability } \frac{1}{2} \end{cases}$$

where  $\mu > 0$  is the risk premium and  $\epsilon > 0$  parametrises the volatility of risky returns.

## 2 Expected utility function

Write a Python function that takes as arguments the risky share  $\alpha$ , the initial wealth  $W_t$ , and the parameters  $\mu$ ,  $\epsilon$  and  $\gamma$ , and returns the expected utility associated with the given values. Your function signature should look like this:

```
[1]: def expected_util(alpha, W, mu, epsilon, gamma):
      # Compute the associated expected utility
      # eu = ...
      return eu
```

Make sure that your function works correctly for both  $\gamma = 1$  and  $\gamma \neq 1$ . Moreover, the function should allow for the arguments  $\alpha$  and  $W$  to be passed as both scalar values as well as NumPy arrays!

### 3 Plot expected utility against the risky share

Use the function you wrote above to plot expected utility for 100 values of  $\alpha$  on the unit interval  $[0, 1]$ . Assume the following values for the remaining parameters:

```
[3]: import numpy as np

W = 1.0          # Initial wealth
gamma = 2.0      # Relative risk aversion
mu = 0.04        # Risk premium
epsilon = 0.2    # Standard deviation of risky return

# List of risky shares (alpha) for which to plot expected utility
alpha = np.linspace(0.0, 1.0, 100)
```

Label both axes and add a legend to your plot.

### 4 Optimal risky share

In the above exercise, you plotted expected utility for all possible risky shares. As a next step, you are asked to compute the *optimal* risky share for a given set of parameters and initial wealth  $W_t$ .

You can find the optimal  $\alpha$  numerically using one of the minimiser function included in SciPy's [optimization package](#). Specifically, for this task you should use the function `minimize_scalar()` with the argument `method='bounded'` since the risky share is a scalar variable which is bounded between  $[0, 1]$ . Use the `expected_util()` function from the previous question as the minimiser's objective:

```
[5]: from scipy.optimize import minimize_scalar

W = 1.0          # Initial wealth
gamma = 2.0      # Relative risk aversion
mu = 0.04        # Risk premium
epsilon = 0.2    # Standard deviation of risky return

# Compute optimal risky share for given parameters
result = minimize_scalar(
    # SciPy minimiser passes a single argument (alpha), which we plug into
    # expected_util() together with any other parameters.
    lambda alpha: - expected_util(alpha, W, mu, epsilon, gamma),
    method='bounded',
    bounds=[0, 1]
)

# Print optimal risky share
print(f'Optimal risky share: {result.x}')
```

Optimal risky share: 0.5155169286613541

The following comments should help you understand what the above code does:

- SciPy's routines all perform *minimisation*, whereas we need to *maximise* expected utility. We get around this problem by using a lambda expression and returning *negative* expected utility.
- The `expected_util()` function you wrote requires several arguments, SciPy's `minimize_scalar()`, however, expects a function that takes only a single argument. We can again get around this using the lambda expression as shown above.
- The maximisation result is returned as an `OptimizeResult` object. The only thing you need to know about this object is that the optimal value is stored in the attribute `x` and can be retrieved as illustrated above.

## 4.1 Optimal risky share by wealth

Consider a set of initial wealth levels  $W_t$  uniformly spread over the interval  $[1, 10]$ ,

```
[6]: W = np.linspace(1.0, 10.0, 100)
```

Write a loop that computes the optimal risky share for each of these wealth levels, using the same values for the remaining parameters as above:

```
[7]: gamma = 2.0          # Relative risk aversion
     mu = 0.04           # Risk premium
     epsilon = 0.2       # Standard deviation of risky return
```

Plot the optimal risky share against initial wealth. Set the plot range to the interval  $[0, 1.1]$  using `plt.ylim((0.0, 1.1))` to clearly see the results. How does the optimal risky share depend on initial wealth?

## 4.2 Optimal risky share by relative risk aversion (RRA)

Now consider a set of RRA parameters  $\gamma$  uniformly spread over the interval  $[1, 5]$ ,

```
[10]: gammas = np.linspace(1.0, 5.0, 100)
```

Write a loop that computes the optimal risky share for each  $\gamma$ , using the following values for initial wealth and the remaining parameters:

```
[11]: W = 1.0            # Initial wealth
     mu = 0.04           # Risk premium
     epsilon = 0.2       # Standard deviation of risky return
```

Plot the optimal risky share as a function of these RRA values. How does the optimal risky share depend on risk aversion  $\gamma$ ?

## 5 Portfolio choice with labour income

Now assume that the investor additionally receives constant labour income  $y$  which is independent of the portfolio choice so that next-period's wealth is given by

$$W_{t+1} = R_{t+1}\alpha W_t + R_f(1 - \alpha)W_t + y$$

The remainder of the maximisation problem remains unchanged.

Rewrite the `expected_util()` function from above to include  $y$  as an additional parameter:

```
[14]: def expected_util(alpha, W, y, mu, epsilon, gamma):
     # Compute the associated expected utility
     # eu = ...
     return eu
```

## 5.1 Optimal risky share by wealth

We now revisit the optimal risky share as a function of initial wealth in the presence of labour income. Consider a set of wealth levels  $W$  uniformly spread over the interval  $[1, 20]$ ,

```
[16]: W = np.linspace(1.0, 20.0, 100)
```

Write another loop to evaluate the optimal risky share for each wealth level, assuming the following values for labour income and the remaining parameters:

```
[17]: gamma = 2.0          # Relative risk aversion
      mu = 0.04           # Risk premium
      epsilon = 0.2       # Standard deviation of risky return
      y = 1.0            # Labour income
```

Plot the optimal risky share against initial wealth, and add a horizontal line indicating the optimal risky share in the case of  $y = 0$  from the previous exercise. How does the optimal risky share depend on initial wealth now?

*Hint:* You can add a horizontal line at the  $y$ -coordinate  $y$  by calling `plt.axhline(y)`.