```
using Pkg;  Pkg.activate("Project.toml")
```

```
  Activating project at `D:\JuliaProjects\FirmChoice`
```

```
begin
    using Plots, ForwardDiff, Optim, JuMP, Ipopt
    plotly()
end;
```

Solving Optimisation Problems For Economics I

Introduction

This notebook is the first in a series of notebooks laying out the basics of how to solve an economic optimsation problem in Julia. In particular, this notebook covers a single optimisation problem relating to a firm's choice of cutting down on their CO$_2$ emissions. The question is designed to be similar to something which you may find in undergraduate or postgraduate economics problem sets with the caveat that solving them by hand may be quite time-consuming.

Firm's Choices - Optimal CO$_2$ Abatement

Problemset parameters

These parameters are global and can impact the results of all three parts of the question

```
begin
    Q = 20
    intensity_CO₂ = 0.1
    tax_CO₂_2025 = 120
    tax_CO₂_2035 = 240
    discount_rate = 0.02
end;
```

1.a – Firm's CO$_2$ Abatement Choice

Assume that a firm can produce Q units of output at a cost of $0.2Q + Q^{\frac{1}{2}}$, where each unit of output generates 0.1 tCO$_2$. In 2025, the government will impose a carbon tax of € 120 per tCO$_2$. However, the firm can also choose to abate a fraction a of their emissions at a cost of $730 \times a^{\frac{1}{a}}$ where $a \in [0, 1]$.

Once the tax comes in to effect, how much CO$_2$ will the firm emit and what will be the total annual cost of the carbon tax policy to the firm?

Answer

Intuitively, the firm wants to minimise its total costs with respect to its abatement decision a. Mathematically, this would be equal to setting $\frac{\partial a}{\partial f_{TC}} = 0$.

Our first course of action should be to write out the functions stated in the problem and formulate our objective function which we want to minimise. Once we have specified what the objective function is, we can use **JuMP.jl** (Julia for Mathematical Programming) to solve the constrained optimisation problem.

```
begin
    #Abatement cost function
    f_abatement(a) = 730*a^(1/a)

    #Production cost function
    f_prodcost(Q) = 0.2Q+Q^(1/2)

    #Objective function
    function f_objective(a; tax=tax_CO₂_2025, Q=Q, intensity=intensity_CO₂)
        return f_prodcost(Q) + tax*(Q*intensity)*(1-a) + f_abatement(a)
    end
end;
```

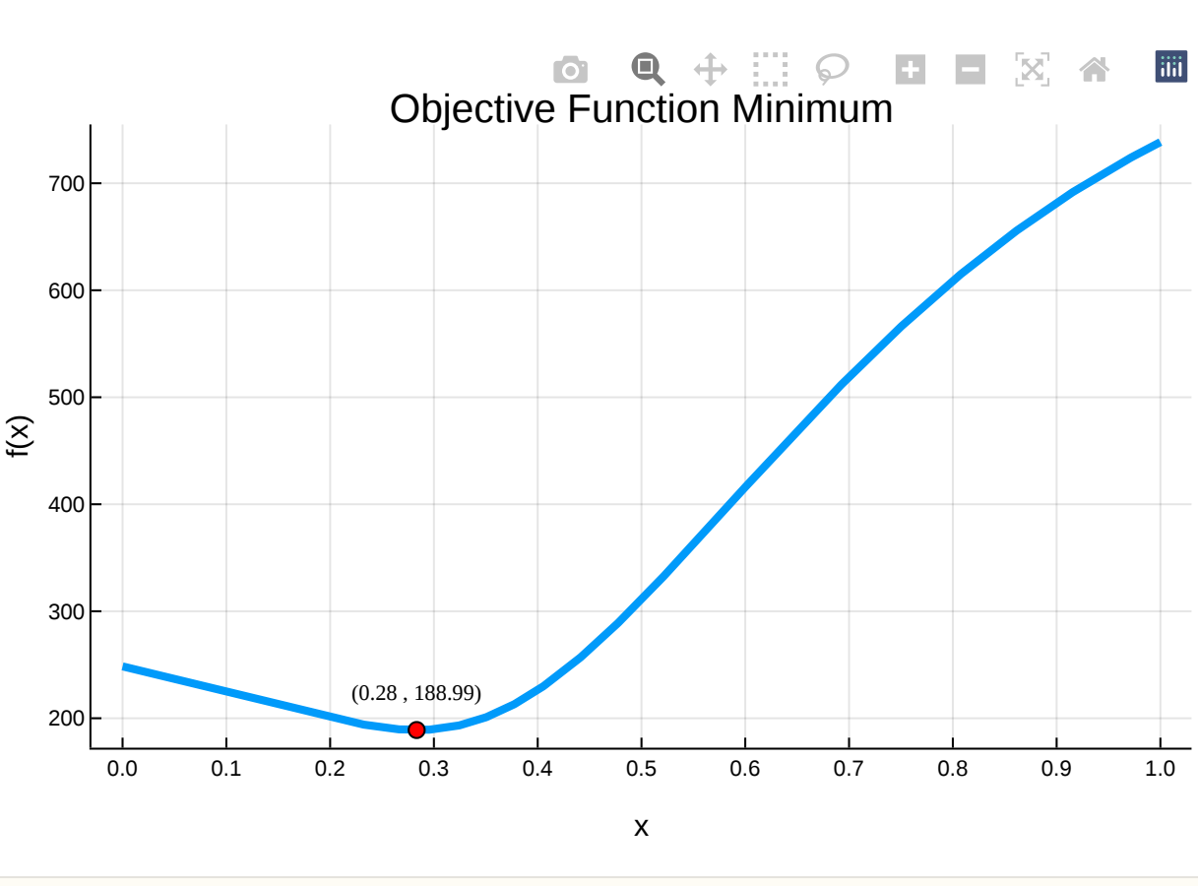
Once we have specified our functions, we pass it through to one of the solvers, **Ipopt.jl** in this case. Ipopt usually works for most optimisation problems in Economics, however, a full list of solvers can be found in the appendix.

```
begin
    #Initialise the model & suppress output print status
    model = Model(Ipopt.Optimizer)
    set_optimizer_attribute(model, "print_level", 0)

    #Register the objective function with one free variable (a)
    register(model, :f_obj, 1, f_objective; autodiff = true)

    #Specify the variables of interest & their constraints
    @variable(model, 0 <= aₘ <= 1)

    #Specify the objective(non-linear) which we want to minimise
    @NLobjective(model, Min, f_obj(aₘ))

    #Optimise the model
    optimize!(model)

    #Store optimal values
    a_optim = value(aₘ)
end;
```

```
******************************************************************************
******
This program contains Ipopt, a library for large-scale nonlinear optimiz
ation.
 Ipopt is released as open source code under the Eclipse Public License
 (EPL).
         For more information visit https://github.com/coin-or/Ipopt
******************************************************************************
******
```

After the model is finished solving, we can extract the optimal abatement level, `a_optim`, from the model.

We can easily visualise the results for inspection, just to make sure that the optimisation is working as it should.



```
begin
    firm_co₂ = Q * intensity_CO₂ * (1-a_optim)
    abate_cost = f_abatement(a_optim)
    tax_cost = (Q * intensity_CO₂ * (1-a_optim)) * tax_CO₂_2025
    production_cost = f_prodcost(Q)
    total_cost = production_cost + tax_cost + abate_cost
end;
```

From the above analysis, we can see that the optimal abatement rate for the firm would be 0.283, costing the firm € 8.52 in abatement costs. This would mean that after abating their emissions, the firm would emit 1.433 tCO$_2$ and pay € 171.99 in taxes. After including the production cost of € 8.47, the firm would be faced with a total cost of € 188.99.

For future reference to total cost calculations, we will use the following function for calculating total costs:

```
function calculate_tc(a_init; tax=tax_CO₂_2025, Q=Q, intensity=intensity_CO₂)
    #Get Optimal Abatement
    a_optim = optimize(x->f_objective(x[1],
                                      tax=tax,
                                      Q=Q,
                                      intensity=intensity),
                       [a_init]).minimizer[1]

    #Calculate Costs
    abate_cost = f_abatement(a_optim)
    tax_cost = (Q * intensity * (1-a_optim)) * tax
    production_cost = f_prodcost(Q)

    #Return total cost and optimal abatement
    return (tc=production_cost + tax_cost + abate_cost, a=a_optim)
end;
```

1.b – Net Present Value (NPV)

Assuming that we are currently in 2022, the first carbon tax of € 120 per tCO$_2$ will come in place in three years. The government has also announced that in 2035, it will increase the tax to € 240 per tCO$_2$ at which point this increased rate will stay in place for perpetuity.

What would be today's net present value of the firm's costs assuming an annual discount rate of 2.0%? How would your answer change if the government decided to instead apply a constant annual growth rate on the tax from 2025 to 2035, reaching the target rate of € 240 in 2035?

Answer

The NPV is a method for determining the current value of all future cash flows. Naturally, we would care more about the present and discount future events to account for opportunity costs. The NPV of a future cost at time t can be calculated according to:

$$NPV = \sum_{t=1}^{n} \frac{R_t}{(1+i)^t}$$

Where R_t is the net cash flows during period t, i is the discount rate, 0.02, and t is the number of time periods. For example, if a firm did not face any costs up to 2025, the NPV of their costs would be € 178.09 (as shown below).

```
NPV_2025 = total_cost/((1+discount_rate)^(2025-2022));
```

However, since the firm faces production costs, these costs need to be factored in to the discounting. Consequently, when we calculate the total cost, TC, faced by the firm we get the following summation.

$$\sum_{t=2023}^{2035} \frac{TC_t}{(1+i)^{t-2022}}$$

We can code this out programmatically, where the new tax policies are suddenly introduced in 2025 and 2035.

```
function npv_sudden(;final_year=2035)
    #Total costs in 2025 and 2035
    tc_2022 = calculate_tc(0.01, tax=0)[:tc]
    tc_2025 = calculate_tc(0.01, tax=tax_CO₂_2025)[:tc]
    tc_2035 = calculate_tc(0.01, tax=tax_CO₂_2035)[:tc]

    final_npv = 0
    for year in range(2023,final_year)
        t = year-2022

        #Before first tax policy implemented (only production cost)
        if year < 2025
            final_npv += tc_2022/(1+discount_rate)^t

        #After first tax policy (production cost plus tax)
        elseif year >= 2025 && year < 2035
            final_npv += tc_2025/(1+discount_rate)^t

        #After second tax policy (production cost plus new tax rate)
        elseif year >= 2035
            final_npv += tc_2035/(1+discount_rate)^t
        end
    end
    return final_npv
end;
```

```
sudden_NPV_2035 = 1648.134609678389
```

```
sudden_NPV_2035 = npv_sudden(final_year=2034)
```

```
sudden_NPV_perpetuity = 15660.788746476423
```

```
sudden_NPV_perpetuity = npv_sudden(final_year=10_000)
```

The cumulative costs the firm will face by 2035 in NPV will be € 1648, and € 15661 over the infinite horizon.

If the tax rate grows at a constant rate between 2025 and 2035, the NPV of the costs will be higher both by 2035 and over the infinite horizon due to higher costs being realised between 2026 and 2034. Using the same approach as before, we can loop through the years in order to calculate the net present value of the total cost to the firm.

```
function npv_gradual(;final_year=2035)
    #Total costs in 2025 and 2035
    tc_2022 = calculate_tc(0.01, tax=0)[:tc]
    tc_2025 = calculate_tc(0.01, tax=tax_CO₂_2025)[:tc]
    tc_2035 = calculate_tc(0.01, tax=tax_CO₂_2035)[:tc]

    #Calculate annual growth rate to achieve 2035 target rate
    tax_growth = 1 + (tax_CO₂_2035/tax_CO₂_2025) ^ (1/(2035-2025)) - 1

    final_npv = 0
    for year in range(2023,final_year)
        t = year-2022

        #Before first tax policy implemented (only production cost)
        if year < 2025
            final_npv += tc_2022/(1+discount_rate)^t

        #After first tax policy (production cost plus tax)
        elseif year == 2025
            final_npv += tc_2025/(1+discount_rate)^t

        #Growing tax by constant rate to reach 2035 target
        elseif year > 2025 && year < 2035
            tc_year = calculate_tc(0.01, tax=tax_CO₂_2025*tax_growth^(year-2025))[:tc]
            final_npv += tc_year/(1+discount_rate)^t

        #After second tax policy (production cost plus new tax rate)
        elseif year >= 2035
            final_npv += tc_2035/(1+discount_rate)^t
        end
    end
    return final_npv
end;
```

```
gradual_NPV_2035 = 2472.9674065084923
```

```
gradual_NPV_2035 = npv_gradual(final_year=2035)
```

```
gradual_NPV_perpetuity = 16210.863619055583
```

```
gradual_NPV_perpetuity = npv_gradual(final_year=10_000)
```

In this case, the cumulative costs the firm will face by 2035 in NPV will be € 2473, and € 16211 over the infinite horizon – € 825 and € 550 higher respectively.

1.c – Permit markets

Instead of implementing a carbon tax, the government is thinking that launching a permit scheme may be a more effective way to reduce emissions. They calculated that in order to reach the net-zero target, emissions cannot be higher than 2.8 from 2025 onward.

The permit market will apply to two firms, both producing Q =20 units of output, where each unit of output emits 0.1 tCO$_2$. Firm A faces a production cost of $0.2Q + Q^{\frac{1}{2}}$ and an abatement cost of $730 \times a^{\frac{1}{a}}$ and firm B faces the same production cost, but an abatement cost of $500 \times a^{\frac{1}{3a}}$.

Assuming that the permits are divided equally between the two firms, and they are not allowed to trade, how much will each firm emit? How about if they can freely trade with no frictions? If each permit counts for 0.1 tCO$_2$, what will be the equilibrium price of the permits?

Answer

As with most questions, the first step will be to set up the objective function which we want to minimise. In this case, the variables of interest would be:

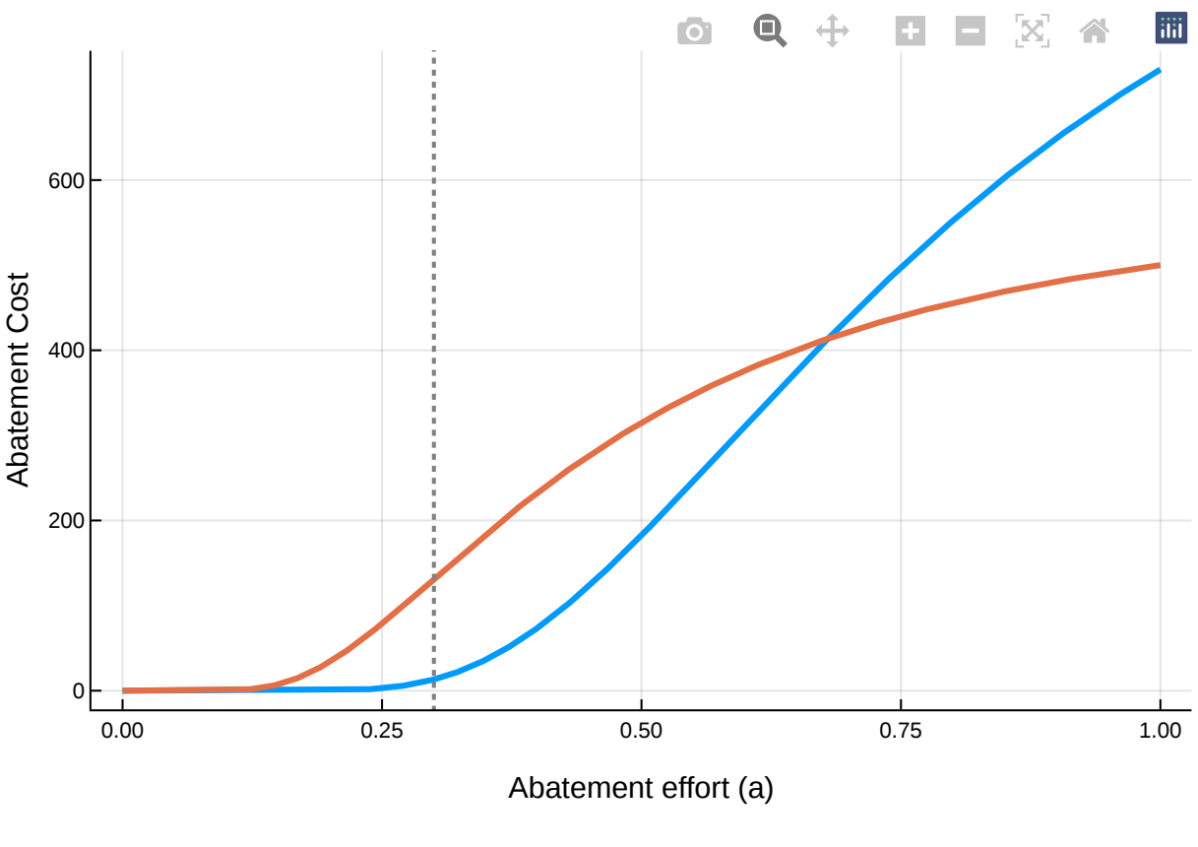
```
begin
    #Firm abatement costs function
    f_abate_A(a) = 730*a^(1/a)
    f_abate_B(a) = 500*a^(1/(3a))

    #Total abatement cost of both firms
    f_tac(a1,a2) = f_abate_A(a1) + f_abate_B(a2)
end;
```

In the first instance, where the firms are not allowed to trade and are handed the same amount of permits, we simply need to plug the level of abatement, $a \in [0, 1]$, into the respective abatement cost functions. The total level of abatement, a, required will be $1 - \frac{2.8}{TotalCO_2}$, or 0.3 for each firm.

We know that total emissions need to be reduced by 0.3, and if both firm A and firm B are allocated the same amount of permits, each one will need to abate the same quantity of CO$_2$ as they are faced with the same production cost function.

```
begin
    #Total firm emissions with no abatement
    e = Q*intensity_CO₂

    #Firm emissions after abatement
    firm_emissions = (A = (1-total_abatement_effort)*e,
                      B = (1-total_abatement_effort)*e)
end;
```

In other words, the firms will emit the same amount where (A = 1.4, B = 1.4) tCO$_2$. Visually, this can be observed as being the vertical line on the diagram below.



However, in this instance, the two firms do not face the same abatement cost function. At lower levels of abatement, Firm B faces a much higher abatement cost than firm A, consequently, under equal allocation with no trading, Firm B is overburdened. This can be seen when comparing the cost faced by the two firms.

```
"Firm A's Cost: 13.194523446119558     ---     Firm B's Cost: 131.21806243858993"
```

In this case, firm A faces a total abatement cost of € 13.19, while firm B faces a cost of € 131.22, resulting in a total cost of both firms of € 144.41.

If we now allow firms to trade freely we can do much better. Firms will trade to the point where their marginal abatement costs equalise. This is because that is the point where both will face the same cost of abating one additional unit and no further surplus can be gained from trading. This is essentially the same as saying that we want to minimise the total abatement costs, where we allow each firm to freely choose their abatement level, as long as the sum of abatement efforts equal 2 × 0.3 .

The first way which one could solve this is through a simple brute-force grid search. As our function isn't very nice to differentiate, we can try just plugging in different values for a_1 and a_2 and then choose the combination of (a_1^*, a_2^*) which have the lowest abatement costs.

```
"Firm A's abatement (grid search): 0.39   ---   Firm B's abatement (grid search): 0.21'
```

```
begin
    #Calculate cost from combinations
    window = range(0,1, length=101)
    grid = [[a₁,a₂,f_tac(a₁,a₂)] for a₁ in window, a₂ in window]

    #Extract cost element from grid
    grid_costs = map(
        g -> ifelse(g[1]+g[2] >= 0.6, g[3], Inf),
        grid
    )

    #Get minimum values from the grid
    a1_grid,a2_grid,c = grid[findmin(grid_costs)[2]]

    #Display values
    "Firm A's abatement (grid search): $a1_grid   ---   Firm B's abatement (grid
    search): $a2_grid"
end
```

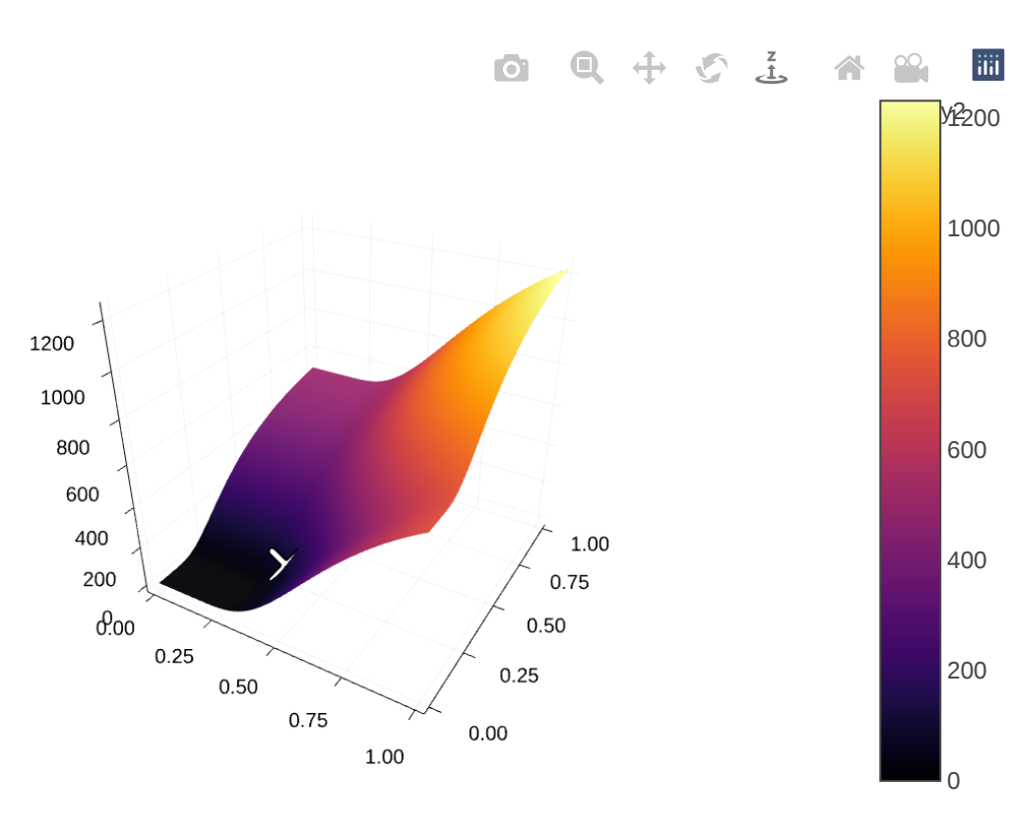
This approach is good for getting a ball-park figure and getting values for non-continuous functions, however, solution time quickly increases if you want to get a more accurate estimate. In this case, as we are only working in a continuous space, we can simply set up an optimisation problem using JuMP.jl and analytically solve it using forward differentiation.

```
"Firm A's abatement (JuMP): 0.388234850595085 --- Firm B's abatement (JuMP): 0.2117651:
```

```
begin
    #Initialise the model & limit print statements
    m2 = Model(Ipopt.Optimizer)
    set_optimizer_attribute(m2, "print_level", 0)

    #Register the objective function with two free variables
    register(m2, :f_obj, 2, f_tac; autodiff = true)

    #Set the variable space
    @variable(m2, 0 <= aₘ₁ <= 1)
    @variable(m2, 0 <= aₘ₂ <= 1)

    #Pass objective function
    @NLobjective(m2, Min, f_obj(aₘ₁, aₘ₂))

    #Set the constraint
    @constraint(m2, (aₘ₁ + aₘ₂) >= 0.6)

    #Optimise the function
    optimize!(m2)

    #Store optimal values
    a1_jump,a2_jump = value(aₘ₁), value(aₘ₂)

    "Firm A's abatement (JuMP): $a1_jump --- Firm B's abatement (JuMP): $a2_jump"
end
```

Notice the significant increase in accuracy we get from using JuMP for solving the optimisation problem. Additionally, the solution time is generally much lower compared to the grid-search approach.

Below is a visualisation of the results, where the abatement efforts of firm 1 and firm 2 are shown on the X and Y axes respectively, with the Z axis representing the total abatement cost. The white "**X**" marker indicates the optimal combination of abatement efforts to minimise the cost.



Although the graph looks correct, lets also check our results with our intuition. If it we truly are at an optimal point, then we know that the first derivatives of the two abatement functions should be the same. We can check this using ForwardDiff.jl.

```
"Firm A's marginal abatement cost: 823.9765752472267 --- Firm B's marginal abatement co
```

```
begin
    #First derivative of abatement cost function
    df_abate_A(a) = ForwardDiff.derivative(f_abate_A, a)
    df_abate_B(a) = ForwardDiff.derivative(f_abate_B, a)

    #Values at the first derivatives
    mac_A = df_abate_A(a1_jump)
    mac_B = df_abate_B(a2_jump)

    #Display results
    "Firm A's marginal abatement cost: $mac_A --- Firm B's marginal abatement
    cost: $mac_B"
end
```

Voila! We can see that the marginal abatement cost faced by the two firms are almost identical.

Finally, the last part of the question asks for the equilibrium price of permits. Assuming that there are no costs associated with trading, at the optimal abatement, neither firm will have an incentive to trade. If the permit price is higher than their marginal abatement cost, the firm simply wouldn't buy the permits, and as soon as the permit price drops below the marginal abatement cost, the firm would buy permits to avoid paying the abatement cost.

Since each firm emits 2.0 tCO$_2$ and each firm permit counts for 0.1 tCO$_2$, a permit accounts for an abatement effort, a, of 0.05 . Consequently, to get the cost per permit, we multiply the marginal abatement cost with the relative abatement effort of one permit. Solving for the permit price we get the equilibrium price of € 41.2 .

```
begin
    #Marginal abatement cost (using firm A, but value equal for firm B)
    mac = df_abate_A(a1_jump)

    #Cost for a single permit at equilibrium
    equilibrium_permit_price = mac * 0.1/(Q*intensity_CO₂)
end;
```

Appendix

Variable rounding

```
begin
    rounded_a_optim = round(a_optim,digits=3)
    rounded_firm_co₂ = round(firm_co₂,digits=3)
    rounded_abate_cost = round(abate_cost, digits=2)
    rounded_tax_cost = round(tax_cost, digits=2)
    rounded_production_cost = round(production_cost, digits=2)
    rounded_total_cost = round(total_cost, digits=2)
    rounded_NPV_2025 = round(NPV_2025,digits=2)
    rounded_sudden_NPV_2035 = Int(round(sudden_NPV_2035))
    rounded_sudden_NPV_perpetuity = Int(round(sudden_NPV_perpetuity))
    rounded_gradual_NPV_2035 = Int(round(gradual_NPV_2035))
    rounded_gradual_NPV_perpetuity = Int(round(gradual_NPV_perpetuity))
    rounded_change_NPV_2035 = Int(round(change_NPV_2035))
    rounded_change_NPV_perpetuity = Int(round(change_NPV_perpetuity))
    rounded_total_abatement_effort = round(total_abatement_effort,digits=2)
    rounded_notrade_cost_A = round(notrade_cost_A,digits=2)
    rounded_notrade_cost_B = round(notrade_cost_B,digits=2)
    rounded_total_notrade_cost = round(total_notrade_cost,digits=2)
    rounded_firm_emission = round(firm_emission, digits=2)
    rounded_permit_abatement_share = round(permit_abatement_share, digits=2)
    rounded_equilibrium_permit_price = round(equilibrium_permit_price, digits=2)
end;
```

Miscellaneous parameters

```
begin
    #Part 1.b
    percent_discount_rate = discount_rate*100
    change_NPV_2035 = gradual_NPV_2035 - sudden_NPV_2035
    change_NPV_perpetuity = gradual_NPV_perpetuity - sudden_NPV_perpetuity

    #Part 1.c
    permit_limit = round(0.70*2Q*intensity_CO₂, digits=1)
    total_abatement_effort = 1 - 2.8/(2Q*intensity_CO₂);
    total_notrade_cost = notrade_cost_A + notrade_cost_B
    firm_emission = intensity_CO₂*Q
    permit_abatement_share = 0.1/(firm_emission)
end;
```

Useful resources

Julia for economists – Automatic differentiation and optimization

JuMP.jl solvers

Lastly, if you have any questions, spot any mistakes, please feel free to reach out!