In [1]:
```
using CSV
using DataFrames
using JuMP
using Plots
using Random
using Statistics
using LinearAlgebra
using Distributions
using BipartiteMatching
using Gurobi
using LinearAlgebra
using SymPy
using NLsolve
using LaTeXStrings
```

Proof of Claim 3

Goal

The notebook aims to verify if the lower bound of $f_1'(x_1)$ exceeds 1 within each set [alphaf, alphaf + delta1) × [alpha,alpha + delta1) × [x1,x1 + delta2) in the claimed region. This lower bound assists in verifying the uniqueness of the solution for the original problem.

Arguments

- `x1` : The value of x_1.
- `alphaf` : Value of α^f.
- `alpha` : Value of α.
- `delta1` and `delta2` : Small positive increments.

Functions

- `lower_bound_derivative(x1, alphaf, alpha, delta1, delta2)` returns the computed lower bound of the derivative within the set [alphaf, alphaf + delta1) × [alpha,alpha + delta1) × [x1,x1 + delta2).
- `verify_unique_solution(alphaf, alpha, delta1, delta2)` returns whether the lower bound > 1.
- `iterate_alphaf_alpha(delta1, delta2)` examines all alphaf, alpha and x1 within the claimed region.

Outputs

- `iterate_alphaf_alpha(delta1, delta2)` should return nothing if all alphaf, alpha and x_1 pass the verification. This is indeed observed when delta1 = delta2 = 0.01.

In [2]:
```
# This function computes the lower bound of the derivative of the func
tion f_1(x_1).
function lower_bound_derivative(x1, alphaf, alpha, delta1, delta2)
    if x1 == 0
        return exp(-(1/2) * (alpha + alphaf) * x1 + 2 * (alpha+delta1)
* (alphaf+delta1) * (x1+delta2
                ) / (alpha + alphaf)+ (2 * alpha * log(x1+delta2)) /
(alpha + alphaf+2*delta1)
                ) * 1/2 * (-alpha - alphaf-2*delta1)+ (
                2 * (alphaf + 1/(x1+delta2))) / (alpha + alphaf+2*delt
a1)
    else
        return exp(-(1/2) * (alpha + alphaf) * x1 + 2 * (alpha+delta1)
* (alphaf+delta1) * (x1+delta2
                ) / (alpha + alphaf)+ (2 * alpha * log(x1+delta2)) /
(alpha + alphaf+2*delta1)
                ) * 1/2 * (-alpha - alphaf-2*delta1)+exp(-(1/2) * (alp
ha + alphaf+2*delta1) * (x1+delta2
                ) + 2 * (alpha) * (alphaf) * (x1) / (alpha + alphaf+2*
delta1)+ (
                2 * (alpha+delta1) * log(x1)) / (alpha + alphaf)) * (
                2 * alpha * (alphaf + 1/(x1+delta2)) / (alpha + alphaf
+2*delta1)) + (
                2 * (alphaf + 1/(x1+delta2))) / (alpha + alphaf+2*delt
a1)
    end
end

# This function returns a boolean indicating whether the derivative's
lower bound > 1 within each interval.
function verify_unique_solution(alphaf, alpha, delta1, delta2)

    x_intervals = 0:delta2:1.0
    for i in 1:length(x_intervals)-1
        x1 = x_intervals[i]
        x2 = x_intervals[i+1]
        f_prime_lower_bound = lower_bound_derivative(x1, alphaf, alph
a, delta1, delta2)

        if f_prime_lower_bound <= 1
            return false, "f' <= 1 in interval [$x1, $x2]"
        end
    end

    return true, "The equation has a unique solution for x."
end

# This function iterates over alphaf and alpha, printing out instances
where the derivative's lower bound <= 1.
function iterate_alphaf_alpha(delta1, delta2)
    alphaf_range, alpha_range = 0.0:delta1:exp(1)+delta1, 0.0:delta1:e
xp(1)+delta1
    alphaf_range, alpha_range = collect(alphaf_range), collect(alpha_r
ange)
    alphaf_range[1] = 0.0001
    alpha_range[1] = 0.0001
    for alphaf in alphaf_range
```

```
        for alpha in alpha_range
            if 0 <= alpha <= alphaf && alpha + alphaf <= exp(1)
                result, message = verify_unique_solution(alphaf, alph
a, delta1, delta2)
                if result == false
                    print("alphaf: ",alphaf,"\t alpha: ",alpha,"\t res
ult: ",result,'\n')
                end
            end
        end
    end
end
```

Out[2]: iterate_alphaf_alpha (generic function with 1 method)

In [3]:

```
# Specify precision values
delta1, delta2 = 0.01, 0.01

# Iterate over alphaf and alpha within the claimed region: nothing is
returned if all alphaf, alpha and x_1 pass the verification
iterate_alphaf_alpha(delta1, delta2)
```

In [ ]: