Julia Tutorial for optimization and operations research

Part 02 - Julia for Modelling and Optimization

Prof. Luiz-Rafael Santos - https://lrsantos11.github.io

UFSC, Brazil and MS&E/Stanford, US

- Tutorial repo on my Github page: https://github.com/lrsantos11/Tutorial-Julia-Opt

Modeling Packages

- Packages from JuMP-dev
  - JuMP: algebraic modeling language for linear, quadratic, and non-linear optimization (with or without constraints)
- Some solvers available
  - HiGHS for linear programming (LP) problems, convex quadratic programming (QP) problems, and mixed integer programming (MIP) problems. (open-source)
  - Ipopt for nonlinear optimization problems (open-source)
  - Gurobi, KNitro, CPLEX, Xpress, Mosek

Modeling and Solver packages

- Let's install:
  - `JuMP` : Julia defautl modeling package
  - `GLPK` (LP dual simplex), `Ipopt` (NLP Interior Point Method) and `Gurobi` (Comercial) solvers
    - The first two are open source and Julia will install not only the interface but the program or library itself
    - `Gurobi` depends on the program being installed and on a license (I have an academic one), so it may not work in any computing environment.

```julia
begin
    using Plots
    using LinearAlgebra
    using JuMP
    using HiGHS
    using Ipopt
end
```

Example 1 - Linear Programming

- The linear programming problem can be given in the form

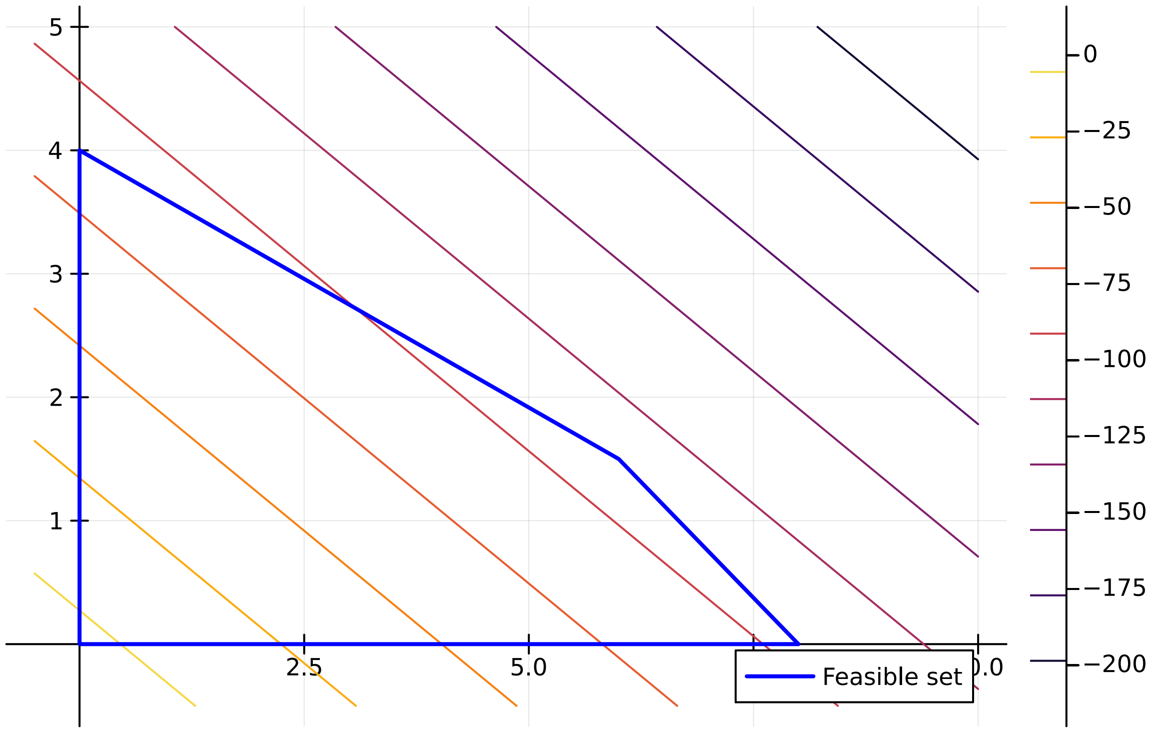
$$\begin{aligned} \min \quad & c^T x \\ \text{s.a} \quad & Ax = b. \\ & x \geq 0 \end{aligned}$$

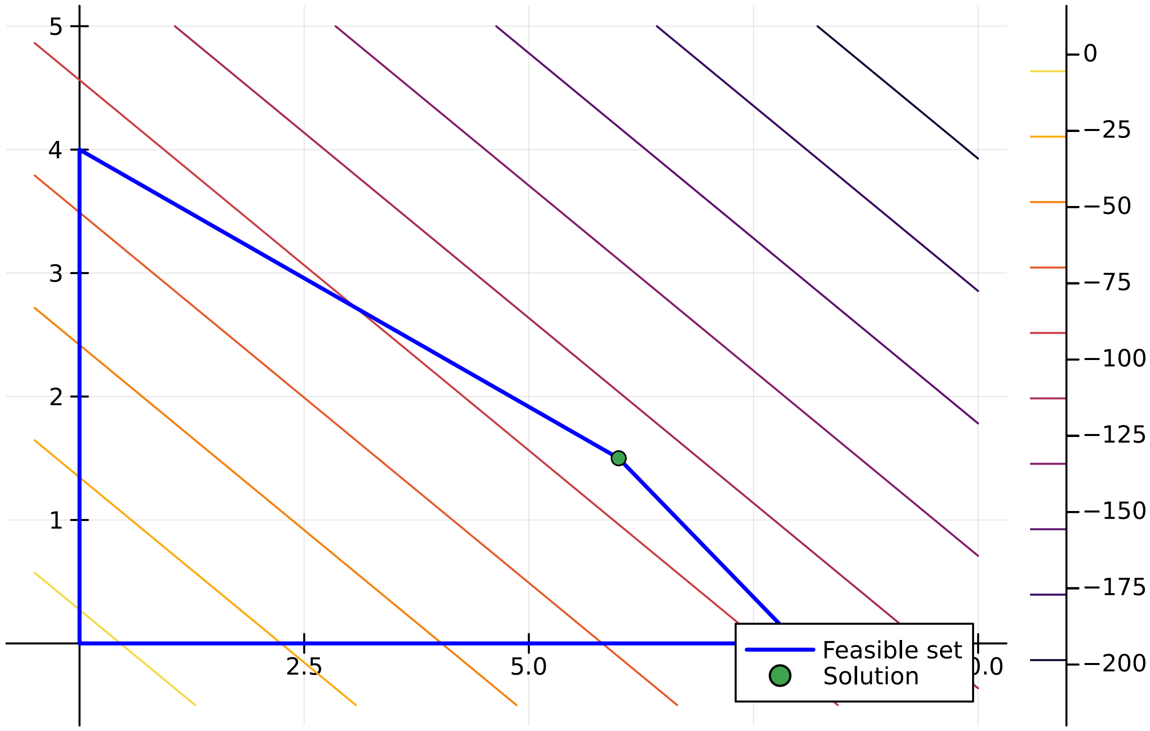
- Let's use `JuMP` to model a problem and the three (or two) available solvers to solve it.

- Consider problem

Selection deleted

$$\begin{aligned} \min \quad & -12x - 20y \\ \text{s.t.} \quad & 3x + 4y \leq 24 \\ & 5x + 12y \leq 48 \\ & x \geq 0 \\ & y \geq 0 \end{aligned}$$




5
4
3
2
1
2.5
5.0
0.0
0
−25
−50
−75
−100
−125
−150
−175
−200
Feasible set
Solution

1.5000000146581798

```
let
     # Model and Solver
    lpmodel = Model(Ipopt.Optimizer)
    # lpmodel = Model(HiGHS.Optimizer)

     # Variables, bounds and type
        @variable(lpmodel, x ≥ 0)
        @variable(lpmodel, y ≥ 0)

     # Constraints
        @constraint(lpmodel, 3x + 4y ≤ 24)
        @constraint(lpmodel, 5x + 12y ≤ 48)

     # objective function
        @objective(lpmodel, Min, -12x - 20y)
    # Print Model
    print(lpmodel)
     # Solver call
    optimize!(lpmodel)
     # declare solution
    @show value(x)
    @show value(y)
end
```

```
   4 -1.0180081e+02 0.00e+00 1.00e-06  -1.0 2.24e-01    -  1.00e+00 1.00e+00f
1
   5 -1.0199360e+02 0.00e+00 2.83e-08  -2.5 1.26e-01    -  1.00e+00 1.00e+00f
1
   6 -1.0199970e+02 0.00e+00 1.50e-09  -3.8 4.95e-03    -  1.00e+00 1.00e+00f
1
   7 -1.0200000e+02 0.00e+00 1.85e-11  -5.7 1.99e-04    -  1.00e+00 1.00e+00f
1
   8 -1.0200000e+02 0.00e+00 2.78e-14  -8.6 2.46e-06    -  1.00e+00 1.00e+00f
1

Number of Iterations....: 8

                                   (scaled)                 (unscaled)
Objective...............:  -1.0200000101498792e+02   -1.0200000101498792e+02
Dual infeasibility......:   2.7756131639819227e-14    2.7756131639819227e-14
Constraint violation....:   0.0000000000000000e+00    0.0000000000000000e+00
Variable bound violation:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity.........:   2.5062831816896763e-09    2.5062831816896763e-09
Overall NLP error.......:   2.5062831816896763e-09    2.5062831816896763e-09


Number of objective function evaluations             = 9
Number of objective gradient evaluations             = 9
Number of equality constraint evaluations            = 0
Number of inequality constraint evaluations          = 9
Number of equality constraint Jacobian evaluations   = 0
Number of inequality constraint Jacobian evaluations = 1
Number of Lagrangian Hessian evaluations             = 1
Total seconds in IPOPT                               = 0.151

EXIT: Optimal Solution Found.
value(x) = 6.0000000601520265
value(y) = 1.5000000146581798
```

-102.0

Let's take a look at each step of the code

- A model is an object that contains variables, constraints, solver options
- Are created with the `Model()` function.
- A model can be created without the solver

```
model = Model(HiGHS.Optimizer)
```

- A variable is modeled using `@variable(model_name, variable_name_and_limits, variable_type)`.
- Limits can be higher or lower. If not defined, the variable is treated as real

```
@variable(model, x >= 0)
@variable(model, y >= 0)
```

- A constraint is modeled using `@constraint(model_name, constraint)`.

```
@constraint(model, 3x + 4y <= 24)
@constraint(model, 5x + 12y <= 48)
```

- The objective function is declared using `@objective(model_name, Min/Max, function to be optimized)`
- `print(model_name)` prints the model (optional).

```
@objective(model, Min, -12x - 20y)
print(model)
```

- To solve the optimization problem we call the `optimize` function

```
optimize!(model)
```

- `x` and `y` are variables that are in the *workspace* but to get their values we need the `value` function
- In the same way, to obtain the value of the objective function at the optimum we use `objective_value(model_name)`

```
@show value(x);
@show value(y);
@show objective_value(model);
```

Some Julia Tricks

List Compreehension

- Comprehensions provide a general and powerful way to construct arrays, and the syntax is similar to the set construction notation from mathematics.

```
A = [f(x, y, ...) for x in X, y in Y, ...]
```

```
julia> X = [0.4, 2.3, 4.6];

julia> Y = [1.4, -3.1, 2.4, 5.2];

julia> A = [exp((x^2 - y^2)/2)/2 for x in X, y in Y]
3×4 Matrix{Float64}:
    0.203285    0.00443536     0.030405   7.27867e-7
    2.64284     0.0576626      0.395285   9.46275e-6
    7382.39     161.072        1104.17    0.0264329
```

`diag` and `reverse`

- `diag` selects the (sub)diagonal of a matrix and
- `reverse` *"transpose"* in the inverse way

```
M = 3×3 Matrix{Int64}:
     2  3  4
     3  4  5
     4  5  6
```

```
M = [i+j for i in 1:3, j in 1:3]
```

```
[[4], [3, 5], [2, 4, 6], [3, 5], [4]]
```

```
[diag(M,i) for i in -2:2]
```

```
3×3 Matrix{Int64}:
 6  5  4
 5  4  3
 4  3  2
```

```julia
reverse(M)
```

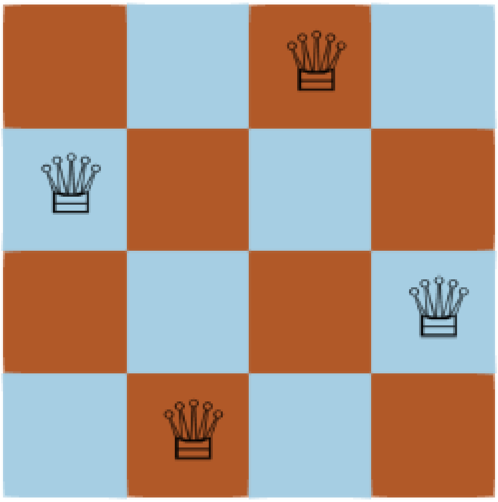
```
[[4], [5, 3], [6, 4, 2], [5, 3], [4]]
```

```julia
[diag(reverse(M),i) for i in -2:2]
```

Example 2 - N-Queens Problem

Feasibility Problem

> The N-queens problem consists of a chessboard of size $N \times N$ on which you want to place N queens in such a way that no queen can attack another. In chess, a queen can move vertically, horizontally and diagonally. This way, there cannot be more than one queen in any row, column or diagonal of the board.



```julia
begin
    NQueens = Model(HiGHS.Optimizer)
    # set_silent(NQueens)


    #let's create an N x N chessboard of binary values. 0 will represent an empty space on the board and 1 will represent a space occupied by one of our queens
    @variable(NQueens, queen[1:N,1:N], Bin)

    #There must be exactly one queen in a given row/column

    # Constraints on the sum of rows (for loop version)
    for i in 1:N
        @constraint(NQueens, sum(queen[i,:]) .== 1)
    end
    # Constraints on sum of columns (list compreehension)
    @constraint(NQueens, [sum(queen[:,j]) for j=1:N] .== 1)


    # Main diagonal constraints
    @constraint(NQueens,[sum(diag(queen,i)) for i = -(N-1):(N-1)] .<= 1)

    # Secondary diagonal constraints
    @constraint(NQueens,[sum(diag(reverse(queen,dims=1),i)) for i = -(N-1):(N-1)] .<= 1)

    optimize!(NQueens)
    print(NQueens)
end
```

```
Running HiGHS 1.5.1 [date: 1970-01-01, git hash: 93f1876e4]
Copyright (c) 2023 HiGHS under MIT licence terms
Presolving model
18 rows, 16 cols, 60 nonzeros
6 rows, 2 cols, 12 nonzeros
0 rows, 0 cols, 0 nonzeros
Presolve: Optimal

Solving report
  Status            Optimal
  Primal bound      0
  Dual bound        0
  Gap               0% (tolerance: 0.01%)
  Solution status   feasible
                    0 (objective)
                    0 (bound viol.)
                    0 (int. viol.)
                    0 (row viol.)
  Timing            0.00 (total)
                    0.00 (presolve)
                    0.00 (postsolve)
  Nodes             0
  LP iterations     0 (total)
                    0 (strong br.)
                    0 (separation)
                    0 (heuristics)
Feasibility
Subject to
 queen[1,1] + queen[1,2] + queen[1,3] + queen[1,4] = 1.0
 queen[2,1] + queen[2,2] + queen[2,3] + queen[2,4] = 1.0
 queen[3,1] + queen[3,2] + queen[3,3] + queen[3,4] = 1.0
 queen[4,1] + queen[4,2] + queen[4,3] + queen[4,4] = 1.0
 queen[1,1] + queen[2,1] + queen[3,1] + queen[4,1] = 1.0
 queen[1,2] + queen[2,2] + queen[3,2] + queen[4,2] = 1.0
```

```
4×4 Matrix{VariableRef}:
 queen[1,1]  queen[1,2]  queen[1,3]  queen[1,4]
 queen[2,1]  queen[2,2]  queen[2,3]  queen[2,4]
 queen[3,1]  queen[3,2]  queen[3,3]  queen[3,4]
 queen[4,1]  queen[4,2]  queen[4,3]  queen[4,4]
```

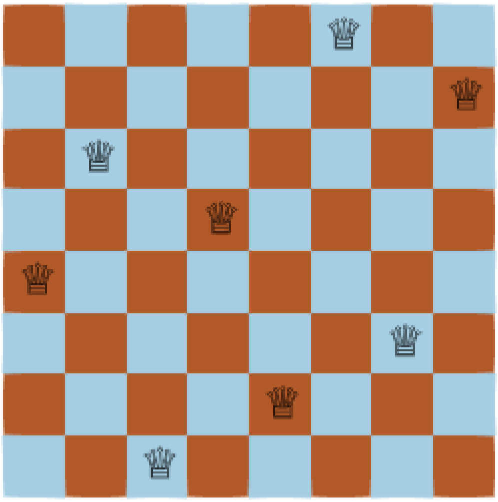
```julia
queen
```

```
solution = 4×4 Matrix{Int64}:
            0  0  1  0
            1  0  0  0
            0  0  0  1
            0  1  0  0
```

```julia
solution = round.(Int, value.(queen))
```

4

```julia
@bind N Slider(4:32, show_value=true)
```



```julia
Enter cell code...
```

How about nonlinear optimization?

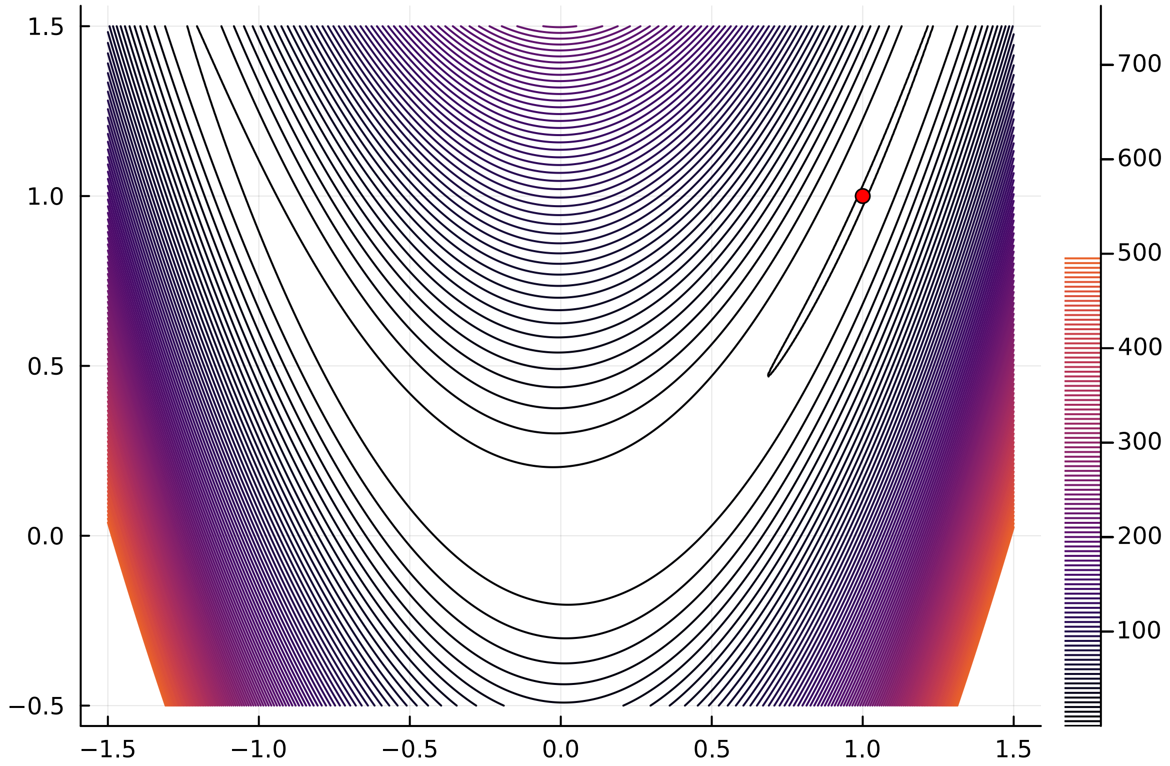
- **Example 3** Rosenbrock functions

$$f(x, y) = (a - x)^2 + b(y - x^2)^2$$

- Local minimizer at (a, a^2) with $f(a, a^2) = 0$
- Let us use `JuMP` and `Ipopt` to minimze Rosenbrock function

$$f(x) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$$

whose local solution is $(1, 1)$.




```
let
    f(x) = (1-x[1])^2 + 100(x[2]-x[1]^2)^2
    x_r = range(-1.5, 1.5, length=400)
    y_r = range(-0.5 , 1.5, length=400)
    contour(x_r,y_r,(x,y) -> f([x;y]), levels = 0.1:5.0:500)
    scatter!([1.0],[1.0],c=:red,label=:false)
end
```

- Use `@NLobjective` instead of `@objective`
- Use `@NLconstraint` instead of `@constraint`

```julia
begin
    rosen = Model(Ipopt.Optimizer)

    @variable(rosen, x_rosen[1:2])

    @NLobjective(rosen, Min, (1 - x_rosen[1])^2 + 100(x_rosen[2] - x_rosen[1]^2)^2)

    print(rosen)

    optimize!(rosen)
end
```

```
Min (1.0 - x_rosen[1]) ^ 2.0 + 100.0 * (x_rosen[2] - x_rosen[1] ^ 2.0) ^ 2.
0
Subject to
This is Ipopt version 3.14.10, running with linear solver MUMPS 5.5.1.

Number of nonzeros in equality constraint Jacobian...:        0
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:        3

Total number of variables............................:        2
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:        0
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr
ls
   0  1.0000000e+00 0.00e+00 2.00e+00  -1.0 0.00e+00    -  0.00e+00 0.00e+00
0
   1  9.5312500e-01 0.00e+00 1.25e+01  -1.0 1.00e+00    -  1.00e+00 2.50e-01f
3
   2  4.8320569e-01 0.00e+00 1.01e+00  -1.0 9.03e-02    -  1.00e+00 1.00e+00f
1
   3  4.5708829e-01 0.00e+00 9.53e+00  -1.0 4.29e-01    -  1.00e+00 5.00e-01f
2
   4  1.8894205e-01 0.00e+00 4.15e-01  -1.0 9.51e-02    -  1.00e+00 1.00e+00f
1
   5  1.3918726e-01 0.00e+00 6.51e+00  -1.7 3.49e-01    -  1.00e+00 5.00e-01f
2
   6  5.4940990e-02 0.00e+00 4.51e-01  -1.7 9.29e-02    -  1.00e+00 1.00e+00f
```

```
[1.0, 1.0]
```

```julia
value.(x_rosen)
```

```julia
begin

    using PlutoUI
    using PlutoReport
    using HypertextLiteral: @htl, @htl_str

    struct Foldable{C}
        title::String
        content::C
    end

    function Base.show(io, mime::MIME"text/html", fld::Foldable)
        write(io,"<details><summary>$(fld.title)</summary><p>")
        show(io, mime, fld.content)
        write(io,"</p></details>")
    end

    struct TwoColumn{L, R}
        left::L
        right::R
    end

    function Base.show(io, mime::MIME"text/html", tc::TwoColumn)
        write(io, """<div style="display: flex;"><div style="flex: 50%;">""")
        show(io, mime, tc.left)
        write(io, """</div><div style="flex: 50%;">""")
        show(io, mime, tc.right)
        write(io, """</div></div>""")
    end
    # apply_css_fixes()
    # @bind _pcon presentation_controls(aside=true)
end
```