Mathematical Optimization using Julia

A 30-min introduction

David García Heredia
garciaherediad@ryanair.com

Agenda

- Introduction to Mathematical Optimization
- Introduction to Julia
  - When may be interesting to use it?
- Mathematical optimization using Julia
  - Ecosystem
  - Hands on!

Introduction to Mathematical Optimization

Mathematical Optimization

Mathematical Programming

Solving decision problems

Math modelling or algorithms

Very demanding computationally

Example: Knapsack problem

Example: Knapsack problem

1. Define a binary variable for each item

Example: Knapsack problem

1. Define a binary variable for each item

Example: Knapsack problem

1. Define a binary variable for each item

Example: Knapsack problem

1. Define a binary variable for each item

Example: Knapsack problem

1. Define a binary variable for each item

Example: Knapsack problem

1. Define a binary variable for each item

Example: Knapsack problem

1. Define a binary variable for each item

2. Write your objective: Maximize revenue

Example: Knapsack problem

max

1. Define a binary variable for each item

2. Write your objective: Maximize revenue

Example: Knapsack problem

max (4$)$X_1$

1. Define a binary variable for each item

2. Write your objective: Maximize revenue

Example: Knapsack problem

max $(4\$)X_1 + (2\$)X_2$

1. Define a binary variable for each item

2. Write your objective: Maximize revenue

Example: Knapsack problem

max $(4\$)X_1 + (2\$)X_2 + (2\$)X_3$

1. Define a binary variable for each item

2. Write your objective: Maximize revenue

Example: Knapsack problem

max $(4\$)X_1 + (2\$)X_2 + (2\$)X_3 + (1\$)X_4$

1. Define a binary variable for each item

2. Write your objective: Maximize revenue

Example: Knapsack problem

max $(4\$)X_1 + (2\$)X_2 + (2\$)X_3 + (1\$)X_4 + (10\$)X_5$

1. Define a binary variable for each item

2. Write your objective: Maximize revenue

Example: Knapsack problem

$\max (4\$)X_1 + (2\$)X_2 + (2\$)X_3 + (1\$)X_4 + (10\$)X_5$

1. Define a binary variable for each item
2. Write your objective: Maximize revenue
3. Write the constraints of the problem

Example: Knapsack problem

$\max (4\$)X_1 + (2\$)X_2 + (2\$)X_3 + (1\$)X_4 + (10\$)X_5$

$(12kg)X_1$

1. Define a binary variable for each item
2. Write your objective: Maximize revenue
3. Write the constraints of the problem

Example: Knapsack problem

$\max (4\$)X_1 + (2\$)X_2 + (2\$)X_3 + (1\$)X_4 + (10\$)X_5$

$(12kg)X_1 + (2kg)X_2$

1. Define a binary variable for each item
2. Write your objective: Maximize revenue
3. Write the constraints of the problem

Example: Knapsack problem

$\max (4\$)X_1 + (2\$)X_2 + (2\$)X_3 + (1\$)X_4 + (10\$)X_5$

$(12kg)X_1 + (2kg)X_2 + (1kg)X_3$

1. Define a binary variable for each item
2. Write your objective: Maximize revenue
3. Write the constraints of the problem

Example: Knapsack problem

$\max (4\$)X_1 + (2\$)X_2 + (2\$)X_3 + (1\$)X_4 + (10\$)X_5$

$(12kg)X_1 + (2kg)X_2 + (1kg)X_3 + (1kg)X_4$

1. Define a binary variable for each item
2. Write your objective: Maximize revenue
3. Write the constraints of the problem

Example: Knapsack problem

$\max (4\$)X_1 + (2\$)X_2 + (2\$)X_3 + (1\$)X_4 + (10\$)X_5$

$(12kg)X_1 + (2kg)X_2 + (1kg)X_3 + (1kg)X_4 + (4kg)X_5$

1. Define a binary variable for each item
2. Write your objective: Maximize revenue
3. Write the constraints of the problem

Example: Knapsack problem

$\max (4\$)X_1 + (2\$)X_2 + (2\$)X_3 + (1\$)X_4 + (10\$)X_5$

$(12kg)X_1 + (2kg)X_2 + (1kg)X_3 + (1kg)X_4 + (4kg)X_5 <= 15kg$

1. Define a binary variable for each item
2. Write your objective: Maximize revenue
3. Write the constraints of the problem

Example: Knapsack problem

$\max (4\$)X_1 + (2\$)X_2 + (2\$)X_3 + (1\$)X_4 + (10\$)X_5$

$(12kg)X_1 + (2kg)X_2 + (1kg)X_3 + (1kg)X_4 + (4kg)X_5 <= 15kg$

$X_1, X_2, X_3, X_4, X_5 \in \{0, 1\}$

1. Define a binary variable for each item
2. Write your objective: Maximize revenue
3. Write the constraints of the problem

Example: Knapsack problem

$\max (4\$)X_1 + (2\$)X_2 + (2\$)X_3 + (1\$)X_4 + (10\$)X_5$

$(12kg)X_1 + (2kg)X_2 + (1kg)X_3 + (1kg)X_4 + (4kg)X_5 <= 15kg$

$X_1, X_2, X_3, X_4, X_5 \in \{0, 1\}$

1. Define a binary variable for each item
2. Write your objective: Maximize revenue
3. Write the constraints of the problem
4. Give the problem to a solver!

Real Applications

Real Applications

- Airlines:
  - Generate the best schedule to fly
  - Plan pilots’ trainings to not interfere with rostering requirements

Real Applications

- Airlines:
  - Generate the best schedule to fly
  - Plan pilots’ trainings to not interfere with rostering requirements
- Logistics
  - How to load items in a truck
  - Plan delivery routes

Real Applications

- Airlines:
  - Generate the best schedule to fly
  - Plan pilots’ trainings to not interfere with rostering requirements
- Logistics
  - How to load items in a truck
  - Plan delivery routes
- Sports
  - NFL schedule

Real Applications

- Airlines:
  - Generate the best schedule to fly
  - Plan pilots' trainings to not interfere with rostering requirements
- Logistics
  - How to load items in a truck
  - Plan delivery routes
- Sports
  - NFL schedule
- Hospitals
  - Plan rosters
  - Plan beds

Real Applications

- Airlines:
  - Generate the best schedule to fly
  - Plan pilots' trainings to not interfere with rostering requirements
- Logistics
  - How to load items in a truck
  - Plan delivery routes
- Sports
  - NFL schedule
- Hospitals
  - Plan rosters
  - Plan beds
- Any complex decision problem!

Introduction to Julia

What is Julia?

What is Julia?

Julia is a programming language, whose main characteristics are:

What is Julia?

Julia is a programming language, whose main characteristics are:

1. It is multi-dispatch, not OOP.
    a. It is weird at the beginning, but then you discover how real power feels like

What is Julia?

Julia is a programming language, whose main characteristics are:

1. It is multi-dispatch, not OOP.
    a. It is weird at the beginning, but then you discover how real power feels like

What is Julia?

Julia is a programming language, whose main characteristics are:

1. It is multi-dispatch, not OOP.
    a. It is weird at the beginning, but then you discover how real power feels like

2. Dynamic type system
    a. It feels like an scripting language (e.g. Python), but you can specify types like in a static type system (e.g. C++)

What is Julia?

Julia is a programming language, whose main characteristics are:

1. It is multi-dispatch, not OOP.
    a. It is weird at the beginning, but then you discover how real power feels like

2. Dynamic type system
    a. It feels like an scripting language (e.g. Python), but you can specify types like in a static type system (e.g. C++)
3. Very good Pkg system to handle dependencies

What is Julia?

Julia is a programming language, whose main characteristics are:

1. It is multi-dispatch, not OOP.
    a. It is weird at the beginning, but then you discover how real power feels like

2. Dynamic type system
    a. It feels like an scripting language (e.g. Python), but you can specify types like in a static type system (e.g. C++)
3. Very good Pkg system to handle dependencies
4. JIT compilation
    a. This is what gives you compiled code
    b. Because of this, the 1st execution of a function is slow
    c. Unlike static langs (e.g. C), you needn't to specify all types, Julia deduces them for you. This is the magic of Julia!!!

Some words of wisdom

Some words of wisdom

1. Julia == speed is not true *per se*. What is true is:

Some words of wisdom

1. Julia == speed is not true *per se*. What is true is:
    a. It is very well designed for scientific computing. Way less verbose than C++, while giving you:
        i. Compiled code
        ii. Great parallel computing (Shared and distributed memory environments)
        iii. Great GPU integration

Some words of wisdom

1. Julia == speed is not true *per se*. What is true is:
    a. It is very well designed for scientific computing. Way less verbose than C++, while giving you:
        i. Compiled code
        ii. Great parallel computing (Shared and distributed memory environments)
        iii. Great GPU integration
    b. High performance is only achieved if:
        i. You follow some guidelines (i.e., put care when coding)
        ii. You don't make clear mistakes (e.g. push to list instead of pre-allocating memory).

Some words of wisdom

1. Julia == speed is not true *per se*. What is true is:
    a. It is very well designed for scientific computing. Way less verbose than C++, while giving you:
        i. Compiled code
        ii. Great parallel computing (Shared and distributed memory environments)
        iii. Great GPU integration
    b. High performance is only achieved if:
        i. You follow some guidelines (i.e., put care when coding)
        ii. You don't make clear mistakes (e.g. push to list instead of pre-allocating memory).
    c. Don't be fooled. C/C++ can in a lot of cases produce code 1.5x–2x faster (no gc, better control of memory…).

Some words of wisdom

1. Julia == speed is not true *per se*. What is true is:
    a. It is very well designed for scientific computing. Way less verbose than C++, while giving you:
        i. Compiled code
        ii. Great parallel computing (Shared and distributed memory environments)
        iii. Great GPU integration
    b. High performance is only achieved if:
        i. You follow some guidelines (i.e., put care when coding)
        ii. You don't make clear mistakes (e.g. push to list instead of pre-allocating memory).
    c. Don't be fooled. C/C++ can in a lot of cases produce code 1.5x–2x faster (no gc, better control of memory…).
2. Personal opinion:
    a. As being a newer language, it has copied the best of other langs (Python list comprehension, Matlab matrix approach, LISP macros…)

Some words of wisdom

1. Julia == speed is not true *per se*. What is true is:
    a. It is very well designed for scientific computing. Way less verbose than C++, while giving you:
        i. Compiled code
        ii. Great parallel computing (Shared and distributed memory environments)
        iii. Great GPU integration
    b. High performance is only achieved if:
        i. You follow some guidelines (i.e., put care when coding)
        ii. You don't make clear mistakes (e.g. push to list instead of pre-allocating memory).
    c. Don't be fooled. C/C++ can in a lot of cases produce code 1.5x–2x faster (no gc, better control of memory…).
2. Personal opinion:
    a. As being a newer language, it has copied the best of other langs (Python list comprehension, Matlab matrix approach, LISP macros…)
3. If you want resources to learn and understand julia, with emphasis in HPC, check:
    a. https://github.com/DavidGarHeredia/JuliaTalk

When to use Julia?

When to use Julia?

- You don’t want to change to Julia (e.g., from R or Python) if:
  a. You use most of the time functions from libs (e.g. Tidyverse, Pandas, Keras,...)
  b. You write code in those langs, but the task are not time consuming (e.g., automate IT tasks)
  c. A very small portion of your code is the bottleneck and you can use Pkgs such as Numba.

When to use Julia?

- You don’t want to change to Julia (e.g., from R or Python) if:
  a. You use most of the time functions from libs (e.g. Tidyverse, Pandas, Keras,...)
  b. You write code in those langs, but the task are not time consuming (e.g., automate IT tasks)
  c. A very small portion of your code is the bottleneck and you can use Pkgs such as Numba.

- You want to give it a try if:
  a. You have to develop code for high-demanding tasks (solving PDE, Optimization algorithms…)
  b. Parts of your code/script are heavily for-loop oriented.
  c. Dealing with huge datasets and standard R/Python vectorizations are not enough.

Why Julia is a great option for math opt practitioners?

Why Julia is a great option for math opt practitioners?

1. Waaaaaay much faster development than in C++

Why Julia is a great option for math opt practitioners?

1. Waaaaaay much faster development than in C++

2. Scripting langs also wins in the previous point, but they have 2 potential problems

Why Julia is a great option for math opt practitioners?

1. Waaaaaay much faster development than in C++

2. Scripting langs also wins in the previous point, but they have 2 potential problems
    a. If you are dealing with big models, the construction time becomes a problem (even in C++).

Why Julia is a great option for math opt practitioners?

1. Waaaaaay much faster development than in C++

2. Scripting langs also wins in the previous point, but they have 2 potential problems
    a. If you are dealing with big models, the construction time becomes a problem (even in C++).
    b. If solvers fail to deal with the problem, you have to develop algorithms for it (metaheuristics or exact methods). That usually means:
        i. A lot of computing-intensive for-loops
        ii. Parallelization to explore the solution space

Why Julia is a great option for math opt practitioners?

1. Waaaaaay much faster development than in C++

2. Scripting langs also wins in the previous point, but they have 2 potential problems
    a. If you are dealing with big models, the construction time becomes a problem (even in C++).
    b. If solvers fail to deal with the problem, you have to develop algorithms for it (metaheuristics or exact methods). That usually means:
        i. A lot of computing-intensive for-loops
        ii. Parallelization to explore the solution space

So if any of these 2 problems arise, you probably don't want to redo all your code in another language

Mathematical Optimization using Julia

Some Pkgs

- Modeling language
  - JuMP.jl
  - It has the most complete interface to solvers that I have never seen for a programming lang
- Solvers for Convex and Non-Convex problems
  - OSQP.jl
  - COSMO.jl
- And many more:
  - Column generation: Coluna.jl
  - Stochastic Programming
    - StochasticPrograms.jl
    - SDDP.jl
  - Evolutionary algorithms: Evolutionary.jl
  - Bilevel Optimization: BilevelJuMP.jl

Let’s see some code!

Struct and functions to read file with data

Let’s see some code!

Struct and functions to read file with data

```
struct Item
    weight::Int
    value::Int
end
```

Check Struct vs Mutable Struct

Let’s see some code!

Struct and functions to read file with data

```
struct Item
    weight::Int
    value::Int
end
```

```
function _get_n_items_and_knapsack_weight(file::IOStream)
    line = readline(file)
    line_split = split(line, " ")
    n_items = parse(Int, line_split[1])
    knapsack_weight = parse(Int, line_split[2])
    return n_items, knapsack_weight
end
```

Let’s see some code!

Struct and functions to read file with data

```
struct Item
    weight::Int
    value::Int
end
```

```
function _get_n_items_and_knapsack_weight(file::IOStream)
    line = readline(file)
    line_split = split(line, " ")
    n_items = parse(Int, line_split[1])
    knapsack_weight = parse(Int, line_split[2])
    return n_items, knapsack_weight
end
```

```
function get_problem_data(path_to_file::String)
    file = open(path_to_file)
    n, knapsack_weight = _get_n_items_and_knapsack_weight(file)
    items = Vector{Item}(undef, n)
    for i in 1:n
        line = readline(file)
        line_split = split(line, " ")
        value = parse(Int, line_split[1])
        weight = parse(Int, line_split[2])
        items[i] = Item(weight, value)
    end
    return knapsack_weight, items
end
```

Empty vector of size n and type Item

Let’s see some code!

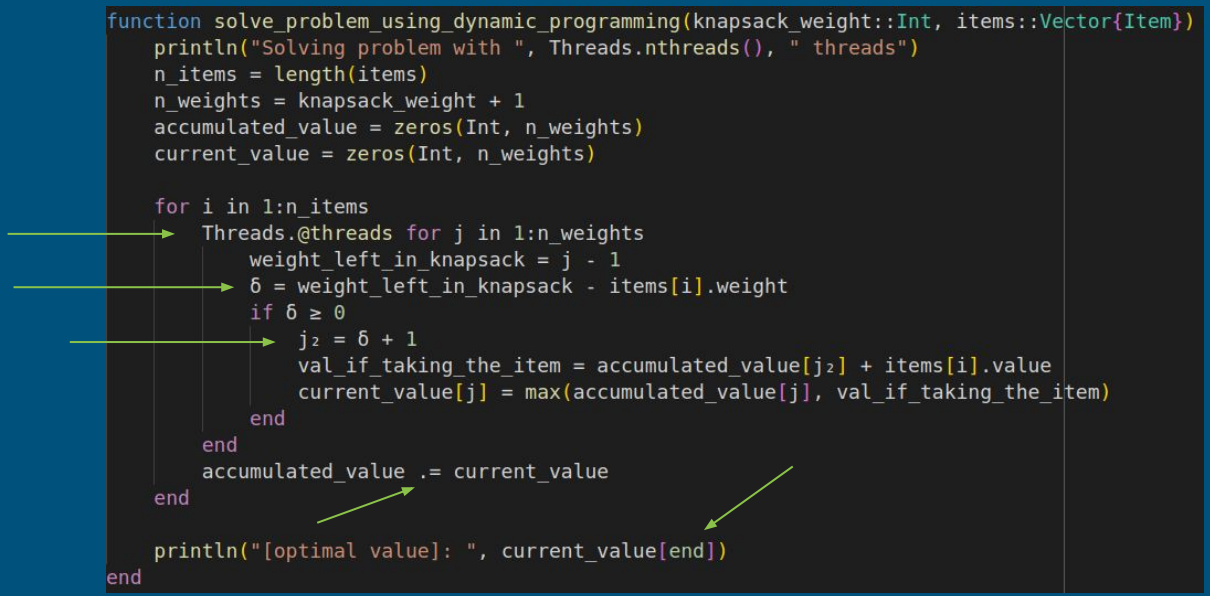
Solving the problem using JuMP and Cbc

```julia
using JuMP
using Cbc

include("reader.jl")

function solve_problem_using_jump(knapsack_weight::Int, items::Vector{Item})
    model = Model(Cbc.Optimizer)
    set_optimizer_attribute(model, "logLevel", 1)

    n_items = length(items)
    @variable(model, x[i=1:n_items], Bin)

    @constraint(model, sum(items[i].weight * x[i] for i in 1:n_items)
                       ≤ # This is not the font, but syntax supported by Julia!
                       knapsack_weight)

    @objective(model, Max, sum(items[i].value * x[i] for i in 1:n_items))
    optimize!(model)
end
```

Macros are not decorators!!!

Let’s see some code!

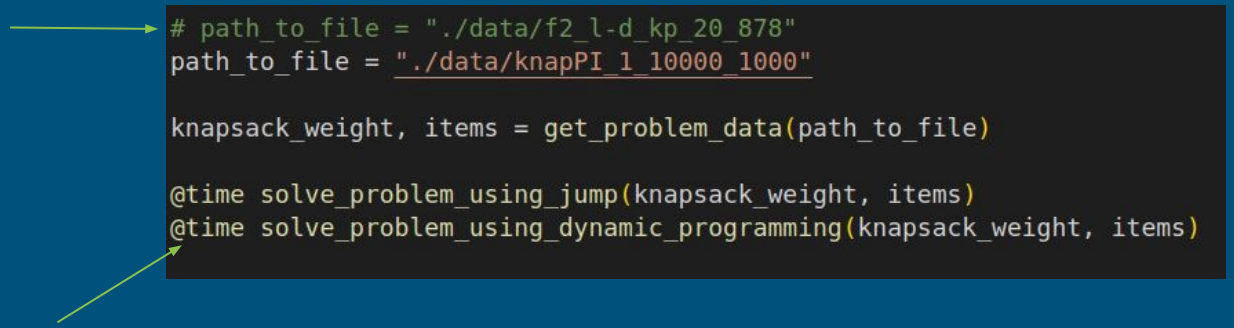
Solving the problem using dynamic programming



```
function solve_problem_using_dynamic_programming(knapsack_weight::Int, items::Vector{Item})
    println("Solving problem with ", Threads.nthreads(), " threads")
    n_items = length(items)
    n_weights = knapsack_weight + 1
    accumulated_value = zeros(Int, n_weights)
    current_value = zeros(Int, n_weights)

    for i in 1:n_items
        Threads.@threads for j in 1:n_weights
            weight_left_in_knapsack = j - 1
            δ = weight_left_in_knapsack - items[i].weight
            if δ ≥ 0
                j₂ = δ + 1
                val_if_taking_the_item = accumulated_value[j₂] + items[i].value
                current_value[j] = max(accumulated_value[j], val_if_taking_the_item)
            end
        end
        accumulated_value .= current_value
    end

    println("[optimal value]: ", current_value[end])
end
```

Let’s see some code!

Calling functions



```
# path_to_file = "./data/f2_l-d_kp_20_878"
path_to_file = "./data/knapPI_1_10000_1000"

knapsack_weight, items = get_problem_data(path_to_file)

@time solve_problem_using_jump(knapsack_weight, items)
@time solve_problem_using_dynamic_programming(knapsack_weight, items)
```

Summary

- Mathematical Optimization
  - Addresses decision problems
  - It is computationally demanding
- Julia
  - It is a script-like programming language, but with JIT compilation code
  - It was born with performance in mind, so it has had a big impact in scientific community
  - If you are familiar with:
    - Python, R or Matlab, learning Julia is easy!
    - Compiled code and good coding-performance practices, getting high performance is easy!

THANKS!

QUESTIONS?